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

0:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\ApplicationContextProvider.java

```
package com.mindata.blockchain;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.context.ApplicationEvent;
import org.springframework.stereotype.Component;

/**
 * @author wuweifeng on 2018/3/13.
 */
@Component
public class ApplicationContextProvider implements ApplicationContextAware {
    private static ApplicationContext context;

    public static ApplicationContext getApplicationContext() {
        return context;
    }

    @Override
    public void setApplicationContext(ApplicationContext ac)
            throws BeansException {
        context = ac;
    }

    public static <T> T getBean(Class<T> tClass) {
        return context.getBean(tClass);
    }

    public static <T> T getBean(String name, Class<T> tClass) {
        return context.getBean(name, tClass);
    }

    public static void publishEvent(ApplicationEvent event) {
        context.publishEvent(event);
    }
}
```

1:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\Block.java

```
package com.mindata.blockchain.block;

import cn.hutool.crypto.digest.DigestUtil;

/**
 *
 * @author wuweifeng wrote on 2018/2/27.
 */
public class Block {
    /**
     *
     */
    private BlockHeader blockHeader;
    /**
     * body
     */
    private BlockBody blockBody;
    /**
     * hash
     */
    private String hash;

    /**
     * sha256
     * @return
     * sha256hex
     */
    private String calculateHash() {
        return DigestUtil.sha256Hex(
                    blockHeader.toString() + blockBody.toString()
        );
    }

    public BlockHeader getBlockHeader() {
        return blockHeader;
    }

    public void setBlockHeader(BlockHeader blockHeader) {
        this.blockHeader = blockHeader;
    }
```

```
    public BlockBody getBlockBody() {
        return blockBody;
    }

    public void setBlockBody(BlockBody blockBody) {
        this.blockBody = blockBody;
    }

    public String getHash() {
        return hash;
    }

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

    @Override
    public String toString() {
        return "Block{" +
                "blockHeader=" + blockHeader +
                ", blockBody=" + blockBody +
                ", hash='" + hash + '\'' +
                '}';
    }
}
```

2:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\BlockBody.java

```
package com.mindata.blockchain.block;

import java.util.List;

/**
 * body
 * @author wuweifeng wrote on 2018/2/28.
 */
public class BlockBody {
    private List<Instruction> instructions;

    @Override
    public String toString() {
```

```
        return "BlockBody{" +
                "instructions=" + instructions +
                '}';
    }

    public List<Instruction> getInstructions() {
        return instructions;
    }

    public void setInstructions(List<Instruction> instructions) {
        this.instructions = instructions;
    }
}
```

3:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\BlockHeader.java

```
package com.mindata.blockchain.block;

import java.util.List;

/**
 *
 * @author wuweifeng wrote on 2018/2/27.
 */
public class BlockHeader {
    /**
     *
     */
    private int version;
    /**
     * hash
     */
    private String hashPreviousBlock;
    /**
     * merkle treehash
     */
    private String hashMerkleRoot;
    /**
     *
     */
    private String publicKey;
    /**
```

```
 *
 */
private int number;
/**
 *
 */
private long timeStamp;
/**
 * 32
 */
private long nonce;
/**
 * hashhashhash
 */
private List<String> hashList;

@Override
public String toString() {
    return "BlockHeader{" +
            "version=" + version +
            ", hashPreviousBlock='" + hashPreviousBlock + '\'' +
            ", hashMerkleRoot='" + hashMerkleRoot + '\'' +
            ", publicKey='" + publicKey + '\'' +
            ", number=" + number +
            ", timeStamp=" + timeStamp +
            ", nonce=" + nonce +
            ", hashList=" + hashList +
            '}';
}

public int getVersion() {
    return version;
}

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

public String getHashPreviousBlock() {
    return hashPreviousBlock;
}
```

```
public void setHashPreviousBlock(String hashPreviousBlock) {
    this.hashPreviousBlock = hashPreviousBlock;
}

public String getHashMerkleRoot() {
    return hashMerkleRoot;
}

public void setHashMerkleRoot(String hashMerkleRoot) {
    this.hashMerkleRoot = hashMerkleRoot;
}

public String getPublicKey() {
    return publicKey;
}

public void setPublicKey(String publicKey) {
    this.publicKey = publicKey;
}

public int getNumber() {
    return number;
}

public void setNumber(int number) {
    this.number = number;
}

public long getTimeStamp() {
    return timeStamp;
}

public void setTimeStamp(long timeStamp) {
    this.timeStamp = timeStamp;
}

public long getNonce() {
    return nonce;
}

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

```
    }

    public List<String> getHashList() {
        return hashList;
    }

    public void setHashList(List<String> hashList) {
        this.hashList = hashList;
    }
}
```

4:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\check\BlockChecker.java

```
package com.mindata.blockchain.block.check;

import com.mindata.blockchain.block.Block;

/**
 *
 * @author wuweifeng wrote on 2018/3/13.
 */
public interface BlockChecker {
    /**
     * num
     * @param block
     *
     * @return
     *
     */
    int checkNum(Block block);

    /**
     *
     * @param block
     * block
     * @return
     * 0
     */
    int checkPermission(Block block);

    /**
     * hashprevHashhashmerkle tree root hash
```

```
     * @param block
     * block
     * @return
     * 0
     */
    int checkHash(Block block);

    /**
     *
     * @param block  block
     * @return block
     */
    int checkTime(Block block);

    /**
     *
     * @param block  block
     * @return block
     */
    int checkSign(Block block);

    /**
     * blockhash
     * @param block
     * @return
     */
    public String checkBlock(Block block);

}
```

5:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\check\CheckerManager.java

```
package com.mindata.blockchain.block.check;

import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.socket.body.RpcCheckBlockBody;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 *
```

```
 * @author wuweifeng wrote on 2018/3/14.
 */
@Component
public class CheckerManager {
    @Resource
    private BlockChecker blockChecker;

    /**
     *
     * @param block block
     * @return
     */
    public RpcCheckBlockBody check(Block block) {
    int code= blockChecker.checkSign(block);
    if (code != 0) {
            return new RpcCheckBlockBody(-1, "block");
        }

        int number = blockChecker.checkNum(block);
        if (number != 0) {
             return new RpcCheckBlockBody(-1, "blocknumber");
        }
        int time = blockChecker.checkTime(block);
        if (time != 0) {
            return new RpcCheckBlockBody(-4, "block");
        }
        int hash = blockChecker.checkHash(block);
        if (hash != 0) {
            return new RpcCheckBlockBody(-3, "hash");
        }
        int permission = blockChecker.checkPermission(block);
        if (permission != 0) {
            return new RpcCheckBlockBody(-2, "");
        }

        return new RpcCheckBlockBody(0, "OK", block);
    }

}
```

6:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\check\local\DbBlockChecker.jav

a

```
package com.mindata.blockchain.block.check.local;

import cn.hutool.core.util.StrUtil;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.check.BlockChecker;
import com.mindata.blockchain.common.Sha256;
import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.core.manager.PermissionManager;
import com.mindata.blockchain.core.requestbody.BlockRequestBody;
import com.mindata.blockchain.core.service.BlockService;

import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 * Blockblock
 * @author wuweifeng wrote on 2018/3/13.
 */
@Component
public class DbBlockChecker implements BlockChecker {
    @Resource
    private DbBlockManager dbBlockManager;
    @Resource
    private PermissionManager permissionManager;

    @Resource
    private BlockService blockService;

    @Override
    public int checkNum(Block block) {
        Block localBlock = getLastBlock();
        int localNum = 0;
        if (localBlock != null) {
            localNum = localBlock.getBlockHeader().getNumber();
        }
        //+1
        if (localNum + 1 == block.getBlockHeader().getNumber()) {
            //
            return 0;
```

```
        }

        //
        return -1;
    }

    @Override
    public int checkPermission(Block block) {
        //
        return permissionManager.checkPermission(block) ? 0 : -1;
    }

    @Override
    public int checkHash(Block block) {
        Block localLast = getLastBlock();
        //prevlast hash
        if (localLast == null && block.getBlockHeader().getHashPreviousBlock() == null) {
            return 0;
        }
        if (localLast != null && StrUtil.equals(localLast.getHash(),
block.getBlockHeader().getHashPreviousBlock())) {
            return 0;
        }
        return -1;
    }

    @Override
    public int checkTime(Block block) {
        Block localBlock = getLastBlock();
        //
        if (localBlock != null && block.getBlockHeader().getTimeStamp() <=
localBlock.getBlockHeader().getTimeStamp()) {
            //
            return -1;
        }
        return 0;
    }

    @Override
    public int checkSign(Block block) {
    if(!checkBlockHashSign(block)) {
    return -1;
```

```
    }
    return 0;
    }

    private Block getLastBlock() {
        return dbBlockManager.getLastBlock();
    }

    public String checkBlock(Block block) {
    if(!checkBlockHashSign(block)) return block.getHash();

    String preHash = block.getBlockHeader().getHashPreviousBlock();
    if(preHash == null) return null;

    Block preBlock = dbBlockManager.getBlockByHash(preHash);
    if(preBlock == null) return block.getHash();

int localNum = preBlock.getBlockHeader().getNumber();
        //+1
        if (localNum + 1 != block.getBlockHeader().getNumber()) {
            return block.getHash();
        }
        if(block.getBlockHeader().getTimeStamp() <= preBlock.getBlockHeader().getTimeStamp()) {
        return block.getHash();
        }



    return null;
    }

    /**
     * hash
     * @param block
     * @return
     */
private boolean checkBlockHashSign(Block block) {
BlockRequestBody blockRequestBody = new BlockRequestBody();
blockRequestBody.setBlockBody(block.getBlockBody());
blockRequestBody.setPublicKey(block.getBlockHeader().getPublicKey());
try {
if(blockService.check(blockRequestBody) != null) return false;
} catch (TrustSDKException e) {
```

```
return false;
}

String hash = Sha256.sha256(block.getBlockHeader().toString() +
block.getBlockBody().toString());
if(!StrUtil.equals(block.getHash(),hash)) return false;

return true;
}

}
```

7:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\db\DbInitConfig.java

```
package com.mindata.blockchain.block.db;

import org.iq80.leveldb.DB;
import org.iq80.leveldb.impl.Iq80DBFactory;
import org.rocksdb.Options;
import org.rocksdb.RocksDB;
import org.rocksdb.RocksDBException;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.io.File;
import java.io.IOException;

/**
 * dbWindowsrocksDBlevelDB
 * @author wuweifeng wrote on 2018/3/13.
 */
@Configuration
public class DbInitConfig {

    @Bean
    @ConditionalOnProperty("db.rocksDB")
    public RocksDB rocksDB() {
        RocksDB.loadLibrary();

        Options options = new Options().setCreateIfMissing(true);
        try {
```

```
            return RocksDB.open(options, "./rocksDB");
        } catch (RocksDBException e) {
            e.printStackTrace();
            return null;
        }
    }

    @Bean
    @ConditionalOnProperty("db.levelDB")
    public DB levelDB() throws IOException {
        org.iq80.leveldb.Options options = new org.iq80.leveldb.Options();
        options.createIfMissing(true);
        return Iq80DBFactory.factory.open(new File("./levelDB"), options);
    }
}
```

8:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\db\DbStore.java

```
package com.mindata.blockchain.block.db;

/**
 * key-valueDB
 * @author wuweifeng wrote on 2018/3/26.
 */
public interface DbStore {
    /**
     * key value
     *
     * @param key
     *         key
     * @param value
     *         value
     */
    void put(String key, String value);

    /**
     * get By Key
     *
     * @param key
     *         key
     * @return value
     */
```

```
    String get(String key);

    /**
     * remove by key
     *
     * @param key
     *         key
     */
    void remove(String key);
}
```

9:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\db\LevelDbStoreImpl.java

```
package com.mindata.blockchain.block.db;

import org.iq80.leveldb.DB;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

import static org.iq80.leveldb.impl.Iq80DBFactory.asString;
import static org.iq80.leveldb.impl.Iq80DBFactory.bytes;

/**
 * levelDB
 *
 * @author wuweifeng wrote on 2018/4/20.
 */
@Component
@ConditionalOnProperty("db.levelDB")
public class LevelDbStoreImpl implements DbStore {
    @Resource
    private DB db;

    @Override
    public void put(String key, String value) {
        db.put(bytes(key), bytes(value));
    }

    @Override
    public String get(String key) {
```

```
        return asString(db.get(bytes(key)));
    }

    @Override
    public void remove(String key) {
        db.delete(bytes(key));
    }
}
```

10:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\db\RocksDbStoreImpl.java

```
package com.mindata.blockchain.block.db;

import com.mindata.blockchain.socket.common.Const;
import org.rocksdb.RocksDB;
import org.rocksdb.RocksDBException;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.io.UnsupportedEncodingException;

/**
 * rocksDB
 * @author wuweifeng wrote on 2018/3/13.
 */
@Component
@ConditionalOnProperty("db.rocksDB")
public class RocksDbStoreImpl implements DbStore {
    @Resource
    private RocksDB rocksDB;

    @Override
    public void put(String key, String value) {
        try {
            rocksDB.put(key.getBytes(Const.CHARSET), value.getBytes(Const.CHARSET));
        } catch (RocksDBException | UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
```

```
    @Override
    public String get(String key) {
        try {
            byte[] bytes = rocksDB.get(key.getBytes(Const.CHARSET));
            if (bytes != null) {
                return new String(bytes, Const.CHARSET);
            }
            return null;
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public void remove(String key) {
        try {
            rocksDB.delete(rocksDB.get(key.getBytes(Const.CHARSET)));
        } catch (RocksDBException | UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

}
```

11:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\Instruction.java

```
package com.mindata.blockchain.block;

/**
 * body
 * @author wuweifeng wrote on 2018/3/2.
 */
public class Instruction extends InstructionBase {
    /**
     *
     */
    private String json;
    /**
     *
     */
    private Long timeStamp;
```

```
/**
 *
 */
private String publicKey;
/**
 *
 */
private String sign;
/**
 * hash
 */
private String hash;


public String getJson() {
    return json;
}

public void setJson(String json) {
    this.json = json;
}

public String getPublicKey() {
    return publicKey;
}

public void setPublicKey(String publicKey) {
    this.publicKey = publicKey;
}

public Long getTimeStamp() {
    return timeStamp;
}

public void setTimeStamp(Long timeStamp) {
    this.timeStamp = timeStamp;
}

public String getSign() {
    return sign;
}
```

```
    public void setSign(String sign) {
        this.sign = sign;
    }

    public String getHash() {
        return hash;
    }

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

    @Override
    public String toString() {
        return "Instruction{" +
                "json='" + json + '\'' +
                ", timeStamp=" + timeStamp +
                ", publicKey='" + publicKey + '\'' +
                ", sign='" + sign + '\'' +
                ", hash='" + hash + '\'' +
                '}';
    }
}
```

12:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\InstructionBase.java

```
package com.mindata.blockchain.block;

/**
 * blockBody
 * @author wuweifeng wrote on 2018/4/4.
 */
public class InstructionBase {
    /**
     * 1-12
     */
    private byte operation;
    /**
     *
     */
    private String table;
    /**
```

```
     * json
     */
    private String oldJson;
    /**
     * idsqlwhereId
     */
    private String instructionId;

    public byte getOperation() {
        return operation;
    }

    public void setOperation(byte operation) {
        this.operation = operation;
    }

    public String getTable() {
        return table;
    }

    public void setTable(String table) {
        this.table = table;
    }

    public String getOldJson() {
        return oldJson;
    }

    public void setOldJson(String oldJson) {
        this.oldJson = oldJson;
    }

    public String getInstructionId() {
        return instructionId;
    }

    public void setInstructionId(String instructionId) {
        this.instructionId = instructionId;
    }

    @Override
    public String toString() {
```

```
        return "InstructionReverse{" +
                "operation=" + operation +
                ", table='" + table + '\'' +
                ", oldJson='" + oldJson + '\'' +
                ", instructionId='" + instructionId + '\'' +
                '}';
    }
}
```

13:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\InstructionReverse.java

```
package com.mindata.blockchain.block;

/**
 *
 * @author wuweifeng wrote on 2018/3/2.
 */
public class InstructionReverse extends InstructionBase {

}
```

14:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\m\MerkleHash.java

```
package com.mindata.blockchain.block.m;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;

public class MerkleHash {

    /**
     * Hash value as byte array.
     */
    private byte[] value;

    public MerkleHash() {
    }

    /**
```

```
     * Create a MerkleHash from an array of bytes.
     *
     * @param buffer of bytes
     * @return a MerkleHash
     */
    public static MerkleHash create(byte[] buffer) {
        MerkleHash hash = new MerkleHash();
        hash.computeHash(buffer);
        return hash;
    }

    /**
     * Create a MerkleHash from a string. The string needs
     * first to be transformed in a UTF8 sequence of bytes.
     * Used for leaf hashes.
     *
     * @param buffer string
     * @return a MerkleHash
     */
    public static MerkleHash create(String buffer) {
        return create(buffer.getBytes(StandardCharsets.UTF_8));
    }

    /**
     * Create a MerkleHash from two MerkleHashes by concatenation
     * of the byte arrays. Used for internal nodes.
     *
     * @param left  subtree hash
     * @param right subtree hash
     * @return a MerkleHash
     */
    public static MerkleHash create(MerkleHash left, MerkleHash right) {
        return create(concatenate(left.getValue(), right.getValue()));
    }

    /**
     * Get the byte value of a MerkleHash.
     *
     * @return an array of bytes
     */
    public byte[] getValue() {
        return value;
```

```
}

/**
 * Compare the MerkleHash with a given byte array.
 *
 * @param hash as byte array
 * @return boolean
 */
public boolean equals(byte[] hash) {
    return Arrays.equals(this.value, hash);
}

/**
 * Compare the MerkleHash with a given MerkleHash.
 *
 * @param hash as MerkleHash
 * @return boolean
 */
public boolean equals(MerkleHash hash) {
    boolean result = false;
    if (hash != null) {
        result = Arrays.equals(this.value, hash.getValue());
    }
    return result;
}

@Override
public int hashCode() {
    return Arrays.hashCode(value);
}

/**
 * Encode in Base64 the MerkleHash.
 *
 * @return the string encoding of MerkleHash.
 */
@Override
public String toString() {
    return Base64.getEncoder().encodeToString(this.value);
}

/**
```

```
     * Compute SHA256 hash of a byte array.
     *
     * @param buffer of bytes
     */
    private void computeHash(byte[] buffer) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            this.value = digest.digest(buffer);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    /**
     * Concatenate two array of bytes.
     *
     * @param a is the first array
     * @param b is the second array
     * @return a byte array
     */
    public static byte[] concatenate(byte[] a, byte[] b) {
        byte[] c = new byte[a.length + b.length];
        System.arraycopy(a, 0, c, 0, a.length);
        System.arraycopy(b, 0, c, a.length, b.length);
        return c;
    }
}
```

15:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\m\MerkleNode.java

```
package com.mindata.blockchain.block.m;

import java.security.InvalidParameterException;
import java.util.Objects;

public class MerkleNode {
    private MerkleHash hash;
    private MerkleNode leftNode;
    private MerkleNode rightNode;
    private MerkleNode parent;
```

```
public MerkleNode() {
}

public MerkleNode(MerkleHash hash) {
    this.hash = hash;
}

public MerkleNode(MerkleNode left, MerkleNode right) {
    this.leftNode = left;
    this.rightNode = right;
    this.leftNode.parent = this;
    if (this.rightNode != null) this.rightNode.parent = this;

    this.computeHash();
}

public boolean isLeaf() {
    return this.leftNode == null && this.rightNode == null;
}

@Override
public String toString() {
    return hash.toString();
}

public void setLeftNode(MerkleNode node) {
    if (node.hash == null) {
        throw new InvalidParameterException("Node hash must be initialized!");
    }

    this.leftNode = node;
    this.leftNode.parent = this;

    this.computeHash();
}

public void setRightNode(MerkleNode node) {
    if (node.hash == null) {
        throw new InvalidParameterException("Node hash must be initialized!");
    }

    this.rightNode = node;
```

```
        this.rightNode.parent = this;

        if (this.leftNode != null) {
          this.computeHash();
        }
    }

    public boolean canVerifyHash() {
        return (this.leftNode != null && this.rightNode != null) || (this.leftNode != null);
    }

    public boolean verifyHash() {
        if (this.leftNode == null && this.rightNode == null) return true;
        if (this.rightNode == null) return hash.equals(leftNode.hash);

        if (this.leftNode == null) {
            throw new InvalidParameterException("Left branch must be a node if right branch is a
node!");
        }

        MerkleHash leftRightHash = MerkleHash.create(this.leftNode.hash, this.rightNode.hash);
        return hash.equals(leftRightHash);
    }

    public boolean equals(MerkleNode other) {
        return this.hash.equals(other.hash);
    }

    public MerkleHash getHash() {
        return hash;
    }

    public MerkleNode getParent() {
        return parent;
    }

    public MerkleNode getLeftNode() {
        return leftNode;
    }

    public MerkleNode getRightNode() {
        return rightNode;
```

```
    }

    public void computeHash() {
        if (this.rightNode == null) {
            this.hash = this.leftNode.hash;
        } else {
            this.hash = MerkleHash.create(MerkleHash.concatenate(
                    this.leftNode.hash.getValue(), this.rightNode.hash.getValue()));
        }

        if (this.parent != null) {
            this.parent.computeHash();
        }
    }

    @Override
    public int hashCode() {

        return Objects.hash(hash, leftNode, rightNode, parent);
    }
}
```

16:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\m\MerkleProofHash.java

```
package com.mindata.blockchain.block.m;

public class MerkleProofHash {
    public enum Branch {
        LEFT,
        RIGHT,
        OLD_ROOT
    }

    public MerkleHash hash;
    public Branch direction;

    public MerkleProofHash(MerkleHash hash, Branch direction) {
        this.hash = hash;
        this.direction = direction;
    }

    public MerkleHash getHash() {
```

```
        return hash;
    }

    public Branch getDirection() {
        return direction;
    }

    @Override
    public String toString() {
        String hash = this.hash.toString();
        String direction = this.direction.toString();
        return hash.concat("  is ".concat(direction).concat(" Child"));
    }
}
```

17:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\m\MerkleTree.java

```
package com.mindata.blockchain.block.m;

import java.security.InvalidParameterException;
import java.util.ArrayList;
import java.util.List;

public class MerkleTree {

    private MerkleNode root;
    private List<MerkleNode> nodes;
    private List<MerkleNode> leaves;

    public MerkleTree() {
        this.nodes = new ArrayList<>();
        this.leaves = new ArrayList<>();
    }

    public List<MerkleNode> getLeaves() {
        return leaves;
    }
    public List<MerkleNode> getNodes() {
        return nodes;
    }
    public MerkleNode getRoot() {
        return root;
```

```
    }

    public MerkleNode appendLeaf(MerkleNode node) {
        this.nodes.add(node);
        this.leaves.add(node);
        return node;
    }

    public void appendLeaves(MerkleNode[] nodes) {
        for (MerkleNode node : nodes) {
            this.appendLeaf(node);
        }
    }

    public MerkleNode appendLeaf(MerkleHash hash) {
        return this.appendLeaf(new MerkleNode(hash));
    }

    public List<MerkleNode> appendLeaves(MerkleHash[] hashes) {
        List<MerkleNode> nodes = new ArrayList<>();
        for (MerkleHash hash : hashes) {
            nodes.add(this.appendLeaf(hash));
        }
        return nodes;
    }

    public MerkleHash addTree(MerkleTree tree) {
        if (this.leaves.size() <= 0) throw new InvalidParameterException("Cannot add to a tree with
no leaves!");
        tree.leaves.forEach(this::appendLeaf);
        return this.buildTree();
    }

    public MerkleHash buildTree() {
        if (this.leaves.size() <= 0) throw new InvalidParameterException("Cannot add to a tree with
no leaves!");
        this.buildTree(this.leaves);
        return this.root.getHash();
    }

    public void buildTree(List<MerkleNode> nodes) {
        if (nodes.size() <= 0) throw new InvalidParameterException("Node list not expected to be
```

```
empty!");

        if (nodes.size() == 1) {
            this.root = nodes.get(0);
        } else {
            List<MerkleNode> parents = new ArrayList<>();
            for (int i = 0; i < nodes.size(); i += 2) {
                MerkleNode right = (i + 1 < nodes.size()) ? nodes.get(i + 1) : null;
                MerkleNode parent = new MerkleNode(nodes.get(i), right);
                parents.add(parent);
            }
            buildTree(parents);
        }
    }

    public List<MerkleProofHash> auditProof(MerkleHash leafHash) {
        List<MerkleProofHash> auditTrail = new ArrayList<>();

        MerkleNode leafNode = this.findLeaf(leafHash);

        if (leafNode != null) {
            if (leafNode.getParent() == null) throw new InvalidParameterException("Expected leaf to
have a parent!");
            MerkleNode parent = leafNode.getParent();
            this.buildAuditTrail(auditTrail, parent, leafNode);
        }

        return auditTrail;
    }

    public static boolean verifyAudit(MerkleHash rootHash, MerkleHash leafHash,
List<MerkleProofHash> auditTrail) {
        if (auditTrail.size() <= 0) throw new InvalidParameterException("Audit trail cannot be
empty!");

        MerkleHash testHash = leafHash;

        for (MerkleProofHash auditHash : auditTrail) {
            testHash = auditHash.direction == MerkleProofHash.Branch.RIGHT
                    ? MerkleHash.create(testHash, auditHash.hash)
                    : MerkleHash.create(auditHash.hash, testHash);
        }
```

```
        return testHash.equals(rootHash);
    }

    private MerkleNode findLeaf(MerkleHash hash) {
        return this.leaves.stream()
                .filter((leaf) -> leaf.getHash().equals(hash))
                .findFirst()
                .orElse(null);
    }

    private void buildAuditTrail(List<MerkleProofHash> auditTrail, MerkleNode parent, MerkleNode
child) {
        if (parent != null) {
            if (child.getParent() != parent) {
                throw new InvalidParameterException("Parent of child is not expected parent!");
            }

            MerkleNode nextChild = parent.getLeftNode() == child ? parent.getRightNode() :
parent.getLeftNode();
            MerkleProofHash.Branch direction = parent.getLeftNode() == child
                    ? MerkleProofHash.Branch.RIGHT
                    : MerkleProofHash.Branch.LEFT;

            if (nextChild != null) auditTrail.add(new MerkleProofHash(nextChild.getHash(), direction));

            this.buildAuditTrail(auditTrail, parent.getParent(), child.getParent());
        }
    }
}
```

18:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\m\Test.java

```
package com.mindata.blockchain.block.m;

import java.util.List;

/**
 * @author wuweifeng wrote on 2018/3/6.
 */
public class Test {
    public static void main(String[] args) {
```

```
        MerkleTree merkleTree = new MerkleTree();
        MerkleNode merkleNode0 = new MerkleNode(MerkleHash.create("a"));
        MerkleNode merkleNode1 = new MerkleNode(MerkleHash.create("b"));
        MerkleNode merkleNode2 = new MerkleNode(MerkleHash.create("c"));
        MerkleNode merkleNode3 = new MerkleNode(MerkleHash.create("d"));

        merkleTree.appendLeaf(merkleNode0);
        merkleTree.appendLeaf(merkleNode1);
        merkleTree.appendLeaf(merkleNode2);
        merkleTree.appendLeaf(merkleNode3);
        merkleTree.buildTree();
        System.out.println(merkleTree.getRoot().getHash());
        List<MerkleProofHash> hashes = merkleTree.auditProof(MerkleHash.create("a"));
    }
}
```

19:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\merkle\MerkleTree.java

```
package com.mindata.blockchain.block.merkle;

import cn.hutool.crypto.digest.DigestUtil;

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

/**
 * merkle tree
 *
 * @author wuweifeng wrote on 2018/2/27.
 */
public class MerkleTree {
    /**
     *  transaction List
     */
    private List<String> txList;
    /**
     *  Merkle Root
     */
    private String root;

    /**
```

```
     * constructor
     *
     * @param txList
     *          transaction List List
     */
    public MerkleTree(List<String> txList) {
        this.txList = txList;
        root = "";
    }

    /**
     * execute merkle_tree and set root.
     */
    public MerkleTree build() {
        List<String> tempTxList = new ArrayList<>(this.txList);

        List<String> newTxList = getNewTxList(tempTxList);

        while (newTxList.size() != 1) {
            newTxList = getNewTxList(newTxList);
        }

        this.root = newTxList.get(0);
        return this;
    }

    /**
     * return Node Hash List.
     *
     * @param tempTxList
     * list
     * @return
     * hash
     */
    private List<String> getNewTxList(List<String> tempTxList) {
        List<String> newTxList = new ArrayList<>();
        int index = 0;
        while (index < tempTxList.size()) {
            // left
            String left = tempTxList.get(index);
            index++;
            // right
```

```
            String right = "";
            if (index != tempTxList.size()) {
                right = tempTxList.get(index);
            }
            // sha2 hex value
            String sha2HexValue = DigestUtil.sha256Hex(left + right);
            newTxList.add(sha2HexValue);
            index++;
        }

        return newTxList;
    }

    /**
     * Get Root
     *
     * @return
     * hash
     */
    public String getRoot() {
        return this.root;
    }

}
```

20:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\Operation.java

```
package com.mindata.blockchain.block;

/**
 * @author wuweifeng wrote on 2018/3/20.
 */
public interface Operation {
    byte ADD = 1;
    byte DELETE = -1;
    byte UPDATE = 2;

}
```

21:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\block\PairKey.java

```
/**
```

```
 * Project Name:trustsql_sdk
 * File Name:PairKey.java
 * Package Name:com.tencent.trustsql.sdk.bean
 * Date:Jul 26, 201710:27:04 AM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 */

package com.mindata.blockchain.block;

/**
 * ClassName:PairKey <br/>
 * Date:     Jul 26, 2017 10:27:04 AM <br/>
 * @author Rony
 * @since JDK 1.7
 */
public class PairKey {

    private String publicKey;
    private String privateKey;

    public String getPublicKey() {
        return publicKey;
    }

    public void setPublicKey(String publicKey) {
        this.publicKey = publicKey;
    }

    public String getPrivateKey() {
        return privateKey;
    }

    public void setPrivateKey(String privateKey) {
        this.privateKey = privateKey;
    }

}
```

22:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\algorithm\AESAlgorithm.java

```
/**
```

```
 * Project Name:trustsql_sdk
 * File Name:EncryptUtil.java
 * Package Name:com.tencent.trustsql.sdk.util
 * Date:Jul 26, 20172:48:58 PM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
 */

package com.mindata.blockchain.common.algorithm;

import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

/**
 * ClassName:EncryptUtil <br/>
 * Date: Jul 26, 2017 2:48:58 PM <br/>
 *
 * @author Rony
 * @since JDK 1.7
 */
public class AESAlgorithm {

/**
 * aesEncode:aes . <br/>
 *
 * @author Rony
 * @param key
 *
 * @param data
 *
 */
public static byte[] aesEncode(byte[] key, byte[] data) throws Exception {
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
return cipher.doFinal(data);
}

/**
 * aesDecode: aes . <br/>
 *
 * @author Rony
```

```
 * @param key key
 * @param encryptedText  encryptedText
 * @return encryptedText
 * @throws Exception Exception
 * @since JDK 1.7
 */
public static byte[] aesDecode(byte[] key, byte[] encryptedText) throws Exception {
Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
SecretKeySpec secretKey = new SecretKeySpec(key, "AES");
cipher.init(Cipher.DECRYPT_MODE, secretKey);
return cipher.doFinal(encryptedText);
}

}
```

23:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\algorithm\Base58Algorithm.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:Base58.java
 * Package Name:com.tencent.trustsql.sdk.util
 * Date:Jul 26, 20172:48:58 PM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
 */
package com.mindata.blockchain.common.algorithm;

import java.math.BigInteger;
import java.util.Arrays;

/**
 * Base58 is a way to encode Bitcoin addresses (or arbitrary data) as
 * alphanumeric strings.
 * <p>
 * Note that this is not the same base58 as used by Flickr, which you may find
 * referenced around the Internet.
 * <p>
 * You may want to consider working with {@link VersionedChecksummedBytes}
 * instead, which adds support for testing the prefix and suffix bytes commonly
 * found in addresses.
 * <p>
```

```
 * Satoshi explains: why base-58 instead of standard base-64 encoding?
 * <ul>
 * <li>Don't want 0OIl characters that look the same in some fonts and could be
 * used to create visually identical looking account numbers.</li>
 * <li>A string with non-alphanumeric characters is not as easily accepted as an
 * account number.</li>
 * <li>E-mail usually won't line-break if there's no punctuation to break at.</li>
 * <li>Doubleclicking selects the whole number as one word if it's all
 * alphanumeric.</li>
 * </ul>
 * <p>
 * However, note that the encoding/decoding runs in O(n&sup2;) time, so it is
 * not useful for large data.
 * <p>
 * The basic idea of the encoding is to treat the data bytes as a large number
 * represented using base-256 digits, convert the number to be represented using
 * base-58 digits, preserve the exact number of leading zeros (which are
 * otherwise lost during the mathematical operations on the numbers), and
 * finally represent the resulting base-58 digits as alphanumeric ASCII
 * characters.
 */
public class Base58Algorithm {
public static final char[] ALPHABET =
"123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz".toCharArray();
private static final char ENCODED_ZERO = ALPHABET[0];
private static final int[] INDEXES = new int[128];
static {
Arrays.fill(INDEXES, -1);
for (int i = 0; i < ALPHABET.length; i++) {
INDEXES[ALPHABET[i]] = i;
}
}

/**
 * Encodes the given bytes as a base58 string (no checksum is appended).
 *
 * @param input
 *            the bytes to encode
 * @return the base58-encoded string
 */
public static String encode(byte[] input) {
if (input.length == 0) {
```

```
return "";
}
// Count leading zeros.
int zeros = 0;
while (zeros < input.length && input[zeros] == 0) {
++zeros;
}
// Convert base-256 digits to base-58 digits (plus conversion to ASCII
// characters)
input = Arrays.copyOf(input, input.length); // since we modify it
// in-place
char[] encoded = new char[input.length * 2]; // upper bound
int outputStart = encoded.length;
for (int inputStart = zeros; inputStart < input.length;) {
encoded[--outputStart] = ALPHABET[divmod(input, inputStart, 256, 58)];
if (input[inputStart] == 0) {
++inputStart; // optimization - skip leading zeros
}
}
// Preserve exactly as many leading encoded zeros in output as there
// were leading zeros in input.
while (outputStart < encoded.length && encoded[outputStart] == ENCODED_ZERO) {
++outputStart;
}
while (--zeros >= 0) {
encoded[--outputStart] = ENCODED_ZERO;
}
// Return encoded string (including encoded leading zeros).
return new String(encoded, outputStart, encoded.length - outputStart);
}

/**
 * Decodes the given base58 string into the original data bytes.
 *
 * @param input
 *            the base58-encoded string to decode
 * @return the decoded data bytes
 * @throws AddressFormatException
 *             if the given string is not a valid base58 string
 */
public static byte[] decode(String input) throws RuntimeException {
if (input.length() == 0) {
```

```
return new byte[0];
}
// Convert the base58-encoded ASCII chars to a base58 byte sequence
// (base58 digits).
byte[] input58 = new byte[input.length()];
for (int i = 0; i < input.length(); ++i) {
char c = input.charAt(i);
int digit = c < 128 ? INDEXES[c] : -1;
if (digit < 0) {
throw new RuntimeException("Illegal character " + c + " at position " + i);
}
input58[i] = (byte) digit;
}
// Count leading zeros.
int zeros = 0;
while (zeros < input58.length && input58[zeros] == 0) {
++zeros;
}
// Convert base-58 digits to base-256 digits.
byte[] decoded = new byte[input.length()];
int outputStart = decoded.length;
for (int inputStart = zeros; inputStart < input58.length;) {
decoded[--outputStart] = divmod(input58, inputStart, 58, 256);
if (input58[inputStart] == 0) {
++inputStart; // optimization - skip leading zeros
}
}
// Ignore extra leading zeroes that were added during the calculation.
while (outputStart < decoded.length && decoded[outputStart] == 0) {
++outputStart;
}
// Return decoded data (including original number of leading zeros).
return Arrays.copyOfRange(decoded, outputStart - zeros, decoded.length);
}

public static BigInteger decodeToBigInteger(String input) throws RuntimeException {
return new BigInteger(1, decode(input));
}

/**
 * Decodes the given base58 string into the original data bytes, using the
 * checksum in the last 4 bytes of the decoded data to verify that the rest
```

```
 * are correct. The checksum is removed from the returned data.
 *
 * @param input
 *            the base58-encoded string to decode (which should include the
 *            checksum)
 * @throws AddressFormatException
 *             if the input is not base 58 or the checksum does not
 *             validate.
 *
 *             public static byte[] decodeChecked(String input) throws
 *             AddressFormatException { byte[] decoded = decode(input); if
 *             (decoded.length < 4) throw new
 *             AddressFormatException("Input too short"); byte[] data =
 *             Arrays.copyOfRange(decoded, 0, decoded.length - 4); byte[]
 *             checksum = Arrays.copyOfRange(decoded, decoded.length - 4,
 *             decoded.length); byte[] actualChecksum =
 *             Arrays.copyOfRange(Sha256Hash.hashTwice(data), 0, 4); if
 *             (!Arrays.equals(checksum, actualChecksum)) throw new
 *             AddressFormatException("Checksum does not validate"); return
 *             data; }
 */

/**
 * Divides a number, represented as an array of bytes each containing a
 * single digit in the specified base, by the given divisor. The given
 * number is modified in-place to contain the quotient, and the return value
 * is the remainder.
 *
 * @param number
 *            the number to divide
 * @param firstDigit
 *            the index within the array of the first non-zero digit (this
 *            is used for optimization by skipping the leading zeros)
 * @param base
 *            the base in which the number's digits are represented (up to
 *            256)
 * @param divisor
 *            the number to divide by (up to 256)
 * @return the remainder of the division operation
 */
private static byte divmod(byte[] number, int firstDigit, int base, int divisor) {
// this is just long division which accounts for the base of the input
```

```
// digits
int remainder = 0;
for (int i = firstDigit; i < number.length; i++) {
int digit = (int) number[i] & 0xFF;
int temp = remainder * base + digit;
number[i] = (byte) (temp / divisor);
remainder = temp % divisor;
}
return (byte) remainder;
}

}
```

24:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\algorithm\BaseAlgorithm.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:BaseAlgoUtil.java
 * Package Name:com.tencent.trustsql.sdk.algo
 * Date:Jul 26, 20175:54:22 PM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
 */

package com.mindata.blockchain.common.algorithm;


import org.bouncycastle.jce.provider.BouncyCastleProvider;

import java.security.MessageDigest;
import java.security.Security;

/**
 * ClassName:BaseAlgoUtil <br/>
 * Date: Jul 26, 2017 5:54:22 PM <br/>
 *
 * @author Rony
 * @since JDK 1.7
 */
public class BaseAlgorithm {

static {
```

```
Security.addProvider(new BouncyCastleProvider());
}

/**
 * encode bytes
 *
 * @param algorithm  algorithm
 * @param data  data
 * @return byte[]
 */
public static byte[] encode(String algorithm, byte[] data) {
if (data == null) {
return null;
}
try {
MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
messageDigest.update(data);
return messageDigest.digest();
} catch (Exception e) {
throw new RuntimeException(e);
}
}

/**
 * encodeTwice bytes
 *
 * @param algorithm  algorithm
 * @param data data
 * @return byte[]
 */
protected static byte[] encodeTwice(String algorithm, byte[] data) {
if (data == null) {
return null;
}
try {
MessageDigest messageDigest = MessageDigest.getInstance(algorithm);
messageDigest.update(data);
return messageDigest.digest(messageDigest.digest());
} catch (Exception e) {
throw new RuntimeException(e);
}
}
```

```
}
```

25:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\algorithm\DESAlgorithm.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:DESAlgoUtil2.java
 * Package Name:com.tencent.trustsql.sdk.algo
 * Date:Jul 28, 201710:38:59 AM
 * Copyright (c) 2017, NUCC All Rights Reserved.
 *
*/

package com.mindata.blockchain.common.algorithm;

import javax.crypto.Cipher;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESedeKeySpec;
import java.security.Key;

/**
 * ClassName:DESAlgoUtil2 <br/>
 * Function: TODO ADD FUNCTION. <br/>
 * Reason: TODO ADD REASON. <br/>
 * Date:     Jul 28, 2017 10:38:59 AM <br/>
 * @author   Rony
 * @since    JDK 1.7
 */
public class DESAlgorithm {
    /**
     *
     * */
    public static final String KEY_ALGORITHM = "DESede";

    /**
     * ///
     * */
    public static final String CIPHER_ALGORITHM = "DESede/ECB/PKCS5Padding";

    /**
     *
```

```
     *
     * @param key
     *
     * @return Key
     * */
    public static Key toKey(byte[] key) throws Exception {
        // Des
        DESedeKeySpec dks = new DESedeKeySpec(key);
        //
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(KEY_ALGORITHM);
        //
        return keyFactory.generateSecret(dks);
    }

    /**
     *
     *
     * @param data
     *
     * @param key
     *
     * @return byte[]
     * */
    public static byte[] encrypt(byte[] data, byte[] key) throws Exception {
        //
        Key k = toKey(key);
        //
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
        //
        cipher.init(Cipher.ENCRYPT_MODE, k);
        //
        return cipher.doFinal(data);
    }

    /**
     *
     *
     * @param data
     *
     * @param key
     *
     * @return byte[]
```

```
     * */
    public static byte[] decrypt(byte[] data, byte[] key) throws Exception {
        //
        Key k = toKey(key);
        //
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
        //
        cipher.init(Cipher.DECRYPT_MODE, k);
        //
        return cipher.doFinal(data);
    }
}
```

26:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\algorithm\ECDSAAlgorithm.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:ECDSAAlgoUtil.java
 * Package Name:com.tencent.trustsql.sdk.algo
 * Date:Jul 26, 20175:17:04 PM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 */

package com.mindata.blockchain.common.algorithm;


import com.google.common.base.Objects;
import com.mindata.blockchain.common.Constants;
import org.apache.commons.codec.binary.Base64;
import org.bouncycastle.jce.ECNamedCurveTable;
import org.bouncycastle.jce.spec.ECNamedCurveParameterSpec;
import org.bouncycastle.math.ec.ECPoint;
import org.spongycastle.asn1.ASN1InputStream;
import org.spongycastle.asn1.ASN1Integer;
import org.spongycastle.asn1.DERSequenceGenerator;
import org.spongycastle.asn1.DLSequence;
import org.spongycastle.asn1.x9.X9ECParameters;
import org.spongycastle.crypto.digests.SHA256Digest;
import org.spongycastle.crypto.ec.CustomNamedCurves;
import org.spongycastle.crypto.params.ECDomainParameters;
```

```
import org.spongycastle.crypto.params.ECPrivateKeyParameters;
import org.spongycastle.crypto.params.ECPublicKeyParameters;
import org.spongycastle.crypto.signers.ECDSASigner;
import org.spongycastle.crypto.signers.HMacDSAKCalculator;
import org.spongycastle.math.ec.FixedPointUtil;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.SecureRandom;

/**
 * ClassName:ECDSAAlgoUtil <br/>
 * Date: Jul 26, 2017 5:17:04 PM <br/>
 *
 * @author Rony
 * @since JDK 1.7
 */
public class ECDSAAlgorithm {
    public static final ECDomainParameters CURVE;
    public static final BigInteger HALF_CURVE_ORDER;

    static {
        X9ECParameters CURVE_PARAMS = CustomNamedCurves.getByName("secp256k1");
        // Tell Bouncy Castle to precompute data that's needed during secp256k1
        // calculations. Increasing the width
        // number makes calculations faster, but at a cost of extra memory usage
        // and with decreasing returns. 12 was
        // picked after consulting with the BC team.
        FixedPointUtil.precompute(CURVE_PARAMS.getG(), 12);
        CURVE = new ECDomainParameters(CURVE_PARAMS.getCurve(),
CURVE_PARAMS.getG(), CURVE_PARAMS.getN(),
                CURVE_PARAMS.getH());
        HALF_CURVE_ORDER = CURVE_PARAMS.getN().shiftRight(1);
    }

    public static String generatePrivateKey() {
        SecureRandom secureRandom;
        try {
            secureRandom =
```

```
SecureRandom.getInstance(Constants.RANDOM_NUMBER_ALGORITHM,
                Constants.RANDOM_NUMBER_ALGORITHM_PROVIDER);
        } catch (Exception e) {
            secureRandom = new SecureRandom();
        }
        // Generate the key, skipping as many as desired.
        byte[] privateKeyAttempt = new byte[32];
        secureRandom.nextBytes(privateKeyAttempt);
        BigInteger privateKeyCheck = new BigInteger(1, privateKeyAttempt);
        while (privateKeyCheck.compareTo(BigInteger.ZERO) == 0 ||
privateKeyCheck.compareTo(Constants.MAXPRIVATEKEY) > 0) {
            secureRandom.nextBytes(privateKeyAttempt);
            privateKeyCheck = new BigInteger(1, privateKeyAttempt);
        }
        String result = Base64.encodeBase64String(privateKeyAttempt);
        result = result.replaceAll("[\\s*\t\n\r]", "");
        return result;
    }

    /**
     * encodetrue
     * @param privateKeyBase64String
     *
     * @param encode
     * base64
     * @return
     *
     */
    public static String generatePublicKey(String privateKeyBase64String, boolean encode) {
        try {
            byte[] privateKeyBytes = Base64.decodeBase64(privateKeyBase64String);
            ECNamedCurveParameterSpec spec =
ECNamedCurveTable.getParameterSpec("secp256k1");
            ECPoint pointQ = spec.getG().multiply(new BigInteger(1, privateKeyBytes));
            String result = Base64.encodeBase64String(pointQ.getEncoded(encode));
            result = result.replaceAll("[\\s*\t\n\r]", "");
            return result;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

```
    /**
     *
     * @param privateKeyBase64String
     *
     * @return
     *
     */
    public static String generatePublicKey(String privateKeyBase64String) {
        return generatePublicKey(privateKeyBase64String, false);
    }

    public static String decodePublicKey(String encodePubKeyBase64String) {
        try {
            byte[] encodePubkeyBytes = Base64.decodeBase64(encodePubKeyBase64String);
            ECNamedCurveParameterSpec spec =
ECNamedCurveTable.getParameterSpec("secp256k1");
            ECPoint pointQ = spec.getG().getCurve().decodePoint(encodePubkeyBytes);
            String result = Base64.encodeBase64String(pointQ.getEncoded(false));
            result = result.replaceAll("[\\s*\t\n\r]", "");
            return result;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     *
     */
    public static void main(String[] args) throws Exception {
        String priKey = generatePrivateKey();
        System.out.println(priKey);
        String pubKey = generatePublicKey(priKey, true);
        String pubKey1 = generatePublicKey(priKey);
        System.out.println(pubKey);
        System.out.println(pubKey1);
        String sign = sign(priKey, "abc");
        System.out.println(sign);
        boolean verify = verify("abc", sign, pubKey);
        System.out.println(verify);
    }

    /**
```

```
     * address
     * @param publicKey
     *
     * @return
     * Address
     * @throws Exception
     * exception
     */
    public static String getAddress(String publicKey) throws Exception {
        return getAddress(publicKey.getBytes("UTF-8"), 0);
    }

    /**
     *
     * @param keyBytes
     *
     * @param version
     *
     * @return
     * address
     * @throws Exception
     * exception
     */
    public static String getAddress(byte[] keyBytes, int... version) throws Exception {
        byte[] hashSha256 = BaseAlgorithm.encode("SHA-256", keyBytes);
        MessageDigest messageDigest = MessageDigest.getInstance("RipeMD160");
        messageDigest.update(hashSha256);
//byte[] hashRipeMD160 = messageDigest.digest();
//byte[] versionHashBytes = new byte[1 + hashRipeMD160.length];
//if(version == null || version.length == 0) {
//versionHashBytes[0] = 0;
//} else {
//versionHashBytes[0] = (byte) version[0];
//}
//System.arraycopy(hashRipeMD160, 0, versionHashBytes, 1, hashRipeMD160.length);
//byte[] checkSumBytes = BaseAlgorithm.encodeTwice("SHA-256", versionHashBytes);
//byte[] rawAddr = new byte[versionHashBytes.length + 4];
//System.arraycopy(versionHashBytes, 0, rawAddr, 0, versionHashBytes.length);
//System.arraycopy(checkSumBytes, 0, rawAddr, versionHashBytes.length, 4);
        byte[] hashRipeMD160 = messageDigest.digest();
        byte[] hashDoubleSha256 = BaseAlgorithm.encodeTwice("SHA-256", hashRipeMD160);
        byte[] rawAddr = new byte[1 + hashRipeMD160.length + 4];
```

```
        rawAddr[0] = 0;
        System.arraycopy(hashRipeMD160, 0, rawAddr, 1, hashRipeMD160.length);
        System.arraycopy(hashDoubleSha256, 0, rawAddr, hashRipeMD160.length + 1, 4);
        return Base58Algorithm.encode(rawAddr);
    }

    public static String sign(String privateKey, String data) throws UnsupportedEncodingException {
        return sign(privateKey, data.getBytes("UTF-8"));
    }

    public static String sign(String privateKey, byte[] data) {
        byte[] hash256 = BaseAlgorithm.encode("SHA-256", data);
        ECDSASigner signer = new ECDSASigner(new HMacDSAKCalculator(new
SHA256Digest()));
        BigInteger pri = new BigInteger(1, Base64.decodeBase64(privateKey));
        ECPrivateKeyParameters privKey = new ECPrivateKeyParameters(pri, CURVE);
        signer.init(true, privKey);
        BigInteger[] components = signer.generateSignature(hash256);
        byte[] content = new ECDSASignature(components[0],
components[1]).toCanonicalised().encodeToDER();
        String result = Base64.encodeBase64String(content);
        result = result.replaceAll("[\\s*\t\n\r]", "");
        return result;
    }

    /**
     *
     * @param srcStr
     *
     * @param sign
     * sign
     * @param pubKey
     *
     * @return
     *
     * @throws Exception
     * Exception
     */
    public static boolean verify(String srcStr, String sign, String pubKey) throws Exception {
        byte[] hash256 = BaseAlgorithm.encode("SHA-256", srcStr.getBytes("UTF-8"));
        ECDSASignature eCDSASignature =
ECDSASignature.decodeFromDER(Base64.decodeBase64(sign));
```

```
        ECDSASigner signer = new ECDSASigner();
        org.spongycastle.math.ec.ECPoint pub =
CURVE.getCurve().decodePoint(Base64.decodeBase64(pubKey));
        ECPublicKeyParameters params = new
ECPublicKeyParameters(CURVE.getCurve().decodePoint(pub.getEncoded()), CURVE);
        signer.init(false, params);
        return signer.verifySignature(hash256, eCDSASignature.r, eCDSASignature.s);
    }

    public static class ECDSASignature {
        /** The two components of the signature. */
        public final BigInteger r, s;

        /**
         * Constructs a signature with the given components. Does NOT
         * automatically canonicalise the signature.
         */
        public ECDSASignature(BigInteger r, BigInteger s) {
            this.r = r;
            this.s = s;
        }

        /**
         * Returns true if the S component is "low", that means it is below
         *  See <a href=
         * "https://github.com/bitcoin/bips/blob/master/bip-
0062.mediawiki#Low_S_values_in_signatures">
         * BIP62</a>.
         */
        public boolean isCanonical() {
            return s.compareTo(HALF_CURVE_ORDER) <= 0;
        }

        /**
         * Will automatically adjust the S component to be less than or equal to
         * half the curve order, if necessary. This is required because for
         * every signature (r,s) the signature (r, -s (mod N)) is a valid
         * signature of the same message. However, we dislike the ability to
         * modify the bits of a Bitcoin transaction after it's been signed, as
         * that violates various assumed invariants. Thus in future only one of
         * those forms will be considered legal and the other will be banned.
         */
```

```
public ECDSASignature toCanonicalised() {
    if (!isCanonical()) {
        // The order of the curve is the number of valid points that
        // exist on that curve. If S is in the upper
        // half of the number of valid points, then bring it back to the
        // lower half. Otherwise, imagine that
        // N = 10
        // s = 8, so (-8 % 10 == 2) thus both (r, 8) and (r, 2) are
        // valid solutions.
        // 10 - 8 == 2, giving us always the latter solution, which is
        // canonical.
        return new ECDSASignature(r, CURVE.getN().subtract(s));
    } else {
        return this;
    }
}

/**
 * DER is an international standard for serializing data structures
 * which is widely used in cryptography. It's somewhat like protocol
 * buffers but less convenient. This method returns a standard DER
 * encoding of the signature, as recognized by OpenSSL and other
 * libraries.
 */
public byte[] encodeToDER() {
    try {
        return derByteStream().toByteArray();
    } catch (IOException e) {
        // Cannot happen.
        throw new RuntimeException(e);
    }
}

public static ECDSASignature decodeFromDER(byte[] bytes) {
    ASN1InputStream decoder = null;
    try {
        decoder = new ASN1InputStream(bytes);
        DLSequence seq = (DLSequence) decoder.readObject();
        if (seq == null) {
            throw new RuntimeException("Reached past end of ASN.1 stream.");
        }
        ASN1Integer r, s;
```

```
        try {
            r = (ASN1Integer) seq.getObjectAt(0);
            s = (ASN1Integer) seq.getObjectAt(1);
        } catch (ClassCastException e) {
            throw new IllegalArgumentException(e);
        }
        // OpenSSL deviates from the DER spec by interpreting these
        // values as unsigned, though they should not be
        // Thus, we always use the positive versions. See:
        // http://r6.ca/blog/20111119T211504Z.html
        return new ECDSASignature(r.getPositiveValue(), s.getPositiveValue());
    } catch (IOException e) {
        throw new RuntimeException(e);
    } finally {
        if (decoder != null) {
            try {
                decoder.close();
            } catch (IOException x) {
            }
        }
    }
}

protected ByteArrayOutputStream derByteStream() throws IOException {
    // Usually 70-72 bytes.
    ByteArrayOutputStream bos = new ByteArrayOutputStream(72);
    DERSequenceGenerator seq = new DERSequenceGenerator(bos);
    seq.addObject(new ASN1Integer(r));
    seq.addObject(new ASN1Integer(s));
    seq.close();
    return bos;
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o == null || getClass() != o.getClass()) {
        return false;
    }
    ECDSASignature other = (ECDSASignature) o;
```

```
            return r.equals(other.r) && s.equals(other.s);
        }

        @Override
        public int hashCode() {
            return Objects.hashCode(r, s);
        }
    }

}
```

27:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\AppId.java

```
package com.mindata.blockchain.common;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import javax.annotation.PostConstruct;

/**
 * @author wuweifeng wrote on 2018/3/17.
 */
@Component
public class AppId {
    /**
     *
     */
    @Value("${appId}")
    private String appId;
    /**
     *
     */
    @Value("${name}")
    private String name;

    public static String value;
    public static String nameValue;

    @PostConstruct
    public void init() {
        value = appId;
```

```
        nameValue = name;
    }
}
```

28:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\CommonUtil.java

```
package com.mindata.blockchain.common;

import java.net.InetAddress;
import java.net.NetworkInterface;
import java.util.Enumeration;
import java.util.UUID;

/**
 * @author wuweifeng wrote on 2018/3/8.
 */
public class CommonUtil {
    public static Long getNow() {
        return System.currentTimeMillis();
    }

    public static void main(String[] args) {
        InetAddress inetAddress = getLocalHostLANAddress();
        System.out.println(inetAddress.getHostName());
    }

    public static String getLocalIp() {
        InetAddress inetAddress = getLocalHostLANAddress();
        if (inetAddress != null) {
            return inetAddress.getHostAddress();
        }
        return null;
    }

    public static String generateUuid() {
        return UUID.randomUUID().toString();
    }

    /**
     * ip
     */
    private static InetAddress getLocalHostLANAddress() {
```

```
        try {
            InetAddress candidateAddress = null;
            //
            for (Enumeration ifaces = NetworkInterface.getNetworkInterfaces();
ifaces.hasMoreElements(); ) {
                NetworkInterface iface = (NetworkInterface) ifaces.nextElement();
                // IP
                for (Enumeration inetAddrs = iface.getInetAddresses(); inetAddrs.hasMoreElements(); )
{
                    InetAddress inetAddr = (InetAddress) inetAddrs.nextElement();
                    // loopback
                    if (!inetAddr.isLoopbackAddress()) {
                        if (inetAddr.isSiteLocalAddress()) {
                            // site-local
                            return inetAddr;
                        } else if (candidateAddress == null) {
                            // site-local
                            candidateAddress = inetAddr;
                        }
                    }
                }
            }
            if (candidateAddress != null) {
                return candidateAddress;
            }
            //  non-loopback.
            return InetAddress.getLocalHost();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

29:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\Constants.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:Constants.java
 * Package Name:com.tencent.trustsql.sdk
 * Date:Jul 26, 201711:17:18 AM
 * Copyright (c) 2017, Tencent All Rights Reserved.
```

```
 *
*/

package com.mindata.blockchain.common;

import java.math.BigInteger;

/**
 * ClassName:Constants <br/>
 * @author   wuweifeng
 */
public interface Constants {

int PUBKEY_DIGEST_LENGTH = 90; // public key length
int PRVKEY_DIGEST_LENGTH = 45; //private key length
int ADDR_DIGEST_LENGTH = 35;   // address length
int SIGN_DIGEST_LENGTH = 98;   // signature length
int KEY_DES3_DIGEST_LENGTH = 24;  // max size of key for DES3 encrypt
int KEY_AES128_DIGEST_LENGTH = 16; // max size of key for AES128 encrypt
int TRANSSQL_DIGEST_LENGTH = 8192; // max size of trans sql for TrustSQL

String RANDOM_NUMBER_ALGORITHM = "SHA1PRNG";
String RANDOM_NUMBER_ALGORITHM_PROVIDER = "SUN";
BigInteger MAXPRIVATEKEY = new
BigInteger("00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD03
64140", 16);

String INFO_SHARE_PUBKEY =
"BC8s/4qEAvVl4Sv0LwQOWJcVU6Q5hBd+7LlJeEivVmUbdtwP4RTfN8x/G+muMhN8SrweyyVVM
IcIrnMWoFqGfIA=";

/**
 * hashkeyvaluehash
 */
String KEY_LAST_BLOCK = "key_last_block";
    /**
     * hashkeyvaluehash
     */
    String KEY_FIRST_BLOCK = "key_first_block";
/**
 * hashkey valuekey{key_block_xxxxxxx -> blockJson}
 */
```

```
String KEY_BLOCK_HASH_PREFIX = "key_block_";

String KEY_REQUEST_PREFIX = "key_request_";
    /**
     * hashhashkeyhashvaluehash
     */
String KEY_BLOCK_NEXT_PREFIX = "key_next_";
/**
 * key
 */
String KEY_PERMISSION = "key_permission_";
}
```

30:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\exception\ErrorNum.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:ErrorNum.java
 * Package Name:com.tencent.trustsql.sdk
 * Date:Jul 26, 20172:59:02 PM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
*/

package com.mindata.blockchain.common.exception;
/**
 * ClassName:ErrorNum <br/>
 * Date:     Jul 26, 2017 2:59:02 PM <br/>
 * @author   Rony
 * @version
 * @since    JDK 1.7
 * @see
 */
public enum ErrorNum {

INVALID_PARAM_ERROR("001", ""),
DES3_ENCRYPT_ERROR("002", "DES3"),
AES_ENCRYPT_ERROR("003", "AES"),
ECDSA_ENCRYPT_ERROR("004", "ECDSA"),
SIGN_ERROR("005", ""),
GENERATE_SIGN_ERROR("006", ""),
```

```
GENERATE_SQL_ERROR("007", "SQL"),
VERIFY_SIGN_ERROR("008", "");

private String retCode;
private String retMsg;

ErrorNum(String retCode, String retMsg) {
this.retCode = retCode;
this.retMsg = retMsg;
}

public String getRetCode() {
return retCode;
}

public void setRetCode(String retCode) {
this.retCode = retCode;
}

public String getRetMsg() {
return retMsg;
}

public void setRetMsg(String retMsg) {
this.retMsg = retMsg;
}
}
```

31:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\exception\TrustSDKException.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:TrustSDKException.java
 * Package Name:com.tencent.trustsql.sdk.exception
 * Date:Jul 26, 201711:24:06 AM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
*/

package com.mindata.blockchain.common.exception;
```

```
import com.alibaba.fastjson.JSONObject;

/**
 * ClassName:TrustSDKException <br/>
 * Date:     Jul 26, 2017 11:24:06 AM <br/>
 * @author   Rony
 * @version
 * @since    JDK 1.7
 * @see
 */
public class TrustSDKException extends Exception {

private static final long serialVersionUID = -4214831807802264420L;

protected String rtnCd;
protected String rtnMsg;

public TrustSDKException(String rtnCd, String rtnMsg) {
super(rtnMsg);
this.rtnCd = rtnCd;
this.rtnMsg = rtnMsg;
}

public TrustSDKException(String rtnCd, String rtnMsg, Throwable t) {
super(rtnMsg, t);
this.rtnCd = rtnCd;
this.rtnMsg = rtnMsg;
}

public String getRtnCd() {
return rtnCd;
}

public void setRtnCd(String rtnCd) {
this.rtnCd = rtnCd;
}

public String getRtnMsg() {
return rtnMsg;
}
```

```
public void setRtnMsg(String rtnMsg) {
this.rtnMsg = rtnMsg;
}

@Override
public String toString() {
return JSONObject.toJSONString(this);
}
}
```

32:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\FastJsonUtil.java

```
package com.mindata.blockchain.common;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.alibaba.fastjson.serializer.JSONLibDataFormatSerializer;
import com.alibaba.fastjson.serializer.SerializeConfig;
import com.alibaba.fastjson.serializer.SerializerFeature;

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

/**
 * @author wuweifeng wrote on 2018/3/2.
 */
public class FastJsonUtil {
    private static final SerializeConfig CONFIG;

    static {
        CONFIG = new SerializeConfig();
        CONFIG.put(java.util.Date.class, new JSONLibDataFormatSerializer()); // json-lib
        CONFIG.put(java.sql.Date.class, new JSONLibDataFormatSerializer()); // json-lib
    }

    private static final SerializerFeature[] FEATURES = {SerializerFeature.WriteMapNullValue, //
            SerializerFeature.WriteNullListAsEmpty, // listnull[]null
            SerializerFeature.WriteNullNumberAsZero, // null0null
            SerializerFeature.WriteNullBooleanAsFalse, // Booleannullfalsenull
            SerializerFeature.WriteNullStringAsEmpty // null""null
    };
```

```
public static String toJSONString(Object object) {
    return JSON.toJSONString(object, CONFIG, FEATURES);
}

public static String toJSONNoFeatures(Object object) {
    return JSON.toJSONString(object, CONFIG);
}


public static Object toBean(String text) {
    return JSON.parse(text);
}

public static <T> T toBean(String text, Class<T> clazz) {
    return JSON.parseObject(text, clazz);
}

/**
 *
 */
public static <T> Object[] toArray(String text) {
    return toArray(text, null);
}

/**
 *
 */
public static <T> Object[] toArray(String text, Class<T> clazz) {
    return JSON.parseArray(text, clazz).toArray();
}

/**
 * List
  */
public static <T> List<T> toList(String text, Class<T> clazz) {
    return JSON.parseArray(text, clazz);
}

/**
 * stringjson
```

```
     */
    public static Object textToJson(String text) {
        return JSON.parse(text);
    }

    /**
     * jsonmap
     */
    public static Map stringToCollect(String s) {
        return JSONObject.parseObject(s);
    }

    /**
     * mapstring
     */
    public static String collectToString(Map m) {
        return JSONObject.toJSONString(m);
    }
}
```

33:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\PermissionType.java

```
package com.mindata.blockchain.common;

/**
 *
 * @author wuweifeng wrote on 2018/4/10.
 */
public interface PermissionType {
    /**
     *
     */
    byte OWNER = 1;
    /**
     *
     */
    byte ALL = 2;
    byte ADD = 3;
    byte UPDATE = 4;
    byte DELETE = 5;
    /**
     *
```

```
     */
    byte NONE = -1;
}
```

34:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\Sha256.java

```
package com.mindata.blockchain.common;

import cn.hutool.crypto.digest.DigestUtil;

/**
 * @author wuweifeng wrote on 2018/2/27.
 */
public class Sha256 {
    public static String sha256(String input) {
        return DigestUtil.sha256Hex(input);
    }

}
```

35:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\SwaggerConfig.java

```
package com.mindata.blockchain.common;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.context.request.async.DeferredResult;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import springfox.documentation.builders.ApiInfoBuilder;
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.builders.ResponseMessageBuilder;
import springfox.documentation.schema.ModelRef;
import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.service.ResponseMessage;
import springfox.documentation.spi.DocumentationType;
```

```
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;

/**
 * swagger
 */
@Configuration
@EnableSwagger2
@ComponentScan(basePackages = {"com.mindata"})
public class SwaggerConfig extends WebMvcConfigurerAdapter {

@Override
public void addViewControllers( ViewControllerRegistry registry ) {
/*registry.addViewController( "/" ).setViewName("redirect:/swagger-ui.html");*/
registry.addViewController( "/" ).setViewName("redirect:/doc.html");

registry.setOrder(Ordered.HIGHEST_PRECEDENCE );
}

/**
 *
 */
@Override
public void addResourceHandlers(ResourceHandlerRegistry registry) {
/**
 * swagger
 */
registry.addResourceHandler("swagger-ui.html").addResourceLocations("classpath:/META-
INF/resources/");
registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-
INF/resources/webjars/");
}

    /**/
private ArrayList<ResponseMessage> responseMessages = new ArrayList<ResponseMessage>()
{
private static final long serialVersionUID = 1L;
{
add(new ResponseMessageBuilder().code(200).message("").build());
add(new ResponseMessageBuilder().code(400).message("").responseModel(new
```

```
ModelRef("Error")).build());
add(new ResponseMessageBuilder().code(401).message("").responseModel(new
ModelRef("Error")).build());
add(new ResponseMessageBuilder().code(404).message("").responseModel(new
ModelRef("Error")).build());
add(new ResponseMessageBuilder().code(405).message("").responseModel(new
ModelRef("Error")).build());
add(new ResponseMessageBuilder().code(500).message("").responseModel(new
ModelRef("Error")).build());
}
};

/**
 * swaggerOpenApi
 * @return
 */
@Bean
public Docket customDocket() {
//
return new Docket(DocumentationType.SWAGGER_2)
.enable(true)
.groupName("OpenApi")
.genericModelSubstitutes(DeferredResult.class)
.useDefaultResponseMessages(false)
.forCodeGeneration(true)
.select()
.apis(RequestHandlerSelectors.basePackage("com.mindata.blockchain.core.controller"))
.paths(PathSelectors.any())
.build()
.globalResponseMessage(RequestMethod.GET, responseMessages)
.globalResponseMessage(RequestMethod.POST, responseMessages)
.globalResponseMessage(RequestMethod.PUT, responseMessages)
.globalResponseMessage(RequestMethod.DELETE, responseMessages)
.globalResponseMessage(RequestMethod.PATCH, responseMessages)
.apiInfo(apiInfo());
}
/**
 * swaggerOpenApi
 *
 * @return
 */
private ApiInfo apiInfo() {
```

```
Contact contact = new Contact("", "", "1335157415@qq.com");
StringBuffer sb=new StringBuffer(1024);
sb.append("        <strong>Swagger</strong>
RESTFUL+, RESTful Web<br>,API<br>");
sb.append("<strong>:</strong><ul><br>");
sb.append("            &
nbsp;   <li>1.,,<strong>POST</strong><br></li>");
sb.append("            &
nbsp;   <li>2.<strong></strong><strong></strong>,<strong></strong><strong>
</strong><br></li>");
sb.append("            &
nbsp;   <li>3.<strong></strong><strong></strong>,<strong></strong><strong>
</strong><br></li></ul>");
return new ApiInfoBuilder()
.title("API")
.description(sb.toString())
.contact(contact)
.version("1.1.0")
.build();
}

}
```

36:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\timer\TimerManager.java

```
package com.mindata.blockchain.common.timer;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;
import java.util.function.Supplier;

/**
 *
 * Thread.sleep()
 *
 * @author andylo25 wrote on 2018/6/11.
 */
public class TimerManager {

    private volatile static ScheduledExecutorService executorService =
Executors.newScheduledThreadPool(3);
```

```
    public static void schedule(Supplier<?> action, long delay) {
        executorService.schedule(new Runnable() {
            @Override
            public void run() {
                action.get();
            }
        }, delay, TimeUnit.MILLISECONDS);
    }

    public static void scheduleAtFixedRate(Supplier<?> action, long initialDelay, long period) {
        executorService.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                action.get();
            }
        }, initialDelay, period, TimeUnit.MILLISECONDS);
    }

    public static void scheduleWithFixedDelay(Supplier<?> action, long initialDelay, long period) {
        executorService.scheduleWithFixedDelay(new Runnable() {
            @Override
            public void run() {
                action.get();
            }
        }, initialDelay, period, TimeUnit.MILLISECONDS);
    }


}
```

37:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\common\TrustSDK.java

```
/**
 * Project Name:trustsql_sdk
 * File Name:TrustSDK.java
 * Package Name:com.tencent.trustsql.sdk
 * Date:Jul 26, 201710:30:31 AM
 * Copyright (c) 2017, Tencent All Rights Reserved.
 *
 */
```

```
package com.mindata.blockchain.common;

import com.mindata.blockchain.common.algorithm.ECDSAAlgorithm;
import com.mindata.blockchain.common.exception.ErrorNum;
import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.block.PairKey;
import org.apache.commons.codec.binary.Base64;
import org.springframework.util.StringUtils;

import java.io.UnsupportedEncodingException;

/**
 * ClassName:TrustSDK <br/>
 * Date: Jul 26, 2017 10:30:31 AM <br/>
 *
 * @author Rony
 * @since JDK 1.7
 */
public class TrustSDK {

/**
 * generatePairKey:, . <br/>
 *
 * @author Rony
 * @return
 * @throws TrustSDKException
     * TrustSDKException
 * @since JDK 1.7
 */
public static PairKey generatePairKey() throws TrustSDKException {
return generatePairKey(false);
}

/**
 * generatePairKey:. <br/>
 *
 * @author ronyyang
 * @param encodePubKey
 * @return PairKey
 * @throws TrustSDKException
     * TrustSDKException
 * @since JDK 1.7
```

```
 */
public static PairKey generatePairKey(boolean encodePubKey) throws TrustSDKException {
try {
PairKey pair = new PairKey();
String privateKey = ECDSAAlgorithm.generatePrivateKey();
String pubKey = ECDSAAlgorithm.generatePublicKey(privateKey.trim(), encodePubKey);
pair.setPrivateKey(privateKey);
pair.setPublicKey(pubKey);
return pair;
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * checkPairKey:. <br/>
 *
 * @author ronyyang
 * @param prvKey   PRVKEY_DIGEST_LENGTH
 * @param pubKey   PUBKEY_DIGEST_LENGTH
 * @return true   false
 * @throws TrustSDKException TrustSDKException
 * @since JDK 1.7
 */
public static boolean checkPairKey(String prvKey, String pubKey) throws TrustSDKException {
if (StringUtils.isEmpty(prvKey) || StringUtils.isEmpty(pubKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
String correctPubKey = ECDSAAlgorithm.generatePublicKey(prvKey.trim(), true);
            return pubKey.trim().equals(correctPubKey);
        } catch(Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * generatePubkeyByPrvkey: . <br/>
 *
```

```
 * @author Rony
 * @param privateKey
 *
 * @param encode
 *
 * @return
 * @throws TrustSDKException
 * TrustSDKException
 * @since JDK 1.7
 */
public static String generatePubkeyByPrvkey(String privateKey, boolean encode) throws
TrustSDKException {
if (StringUtils.isEmpty(privateKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
        return ECDSAAlgorithm.generatePublicKey(privateKey, encode);
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * generatePubkeyByPrvkey: . <br/>
 *
 * @author Rony
 * @param privateKey
 *
 * @return
 * @throws TrustSDKException TrustSDKException
 * @since JDK 1.7
 */
public static String generatePubkeyByPrvkey(String privateKey) throws TrustSDKException {
return generatePubkeyByPrvkey(privateKey, false);
}

public static String decodePubkey(String encodePubKey) throws TrustSDKException {
if (StringUtils.isEmpty(encodePubKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
```

```
}
try {
        return ECDSAAlgorithm.decodePublicKey(encodePubKey);
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * generateAddrByPubkey:. <br/>
 *
 * @author Rony
 * @param pubKey
 *
 * @return address
 * @throws TrustSDKException
    * TrustSDKException
 * @since JDK 1.7
 */
public static String generateAddrByPubkey(String pubKey) throws TrustSDKException {
if (StringUtils.isEmpty(pubKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
        return ECDSAAlgorithm.getAddress(Base64.decodeBase64(pubKey));
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * generateAddrByPrvkey:. <br/>
 *
 * @author Rony
 * @param privateKey
 *
 * @return Address
 * @throws TrustSDKException TrustSDKException
 * @since JDK 1.7
```

```
 */
public static String generateAddrByPrvkey(String privateKey) throws TrustSDKException {
if (StringUtils.isEmpty(privateKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
String pubKey = ECDSAAlgorithm.generatePublicKey(privateKey);
        return generateAddrByPubkey(pubKey);
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}

/**
 * signString:, . <br/>
 *
 * @author Rony
 * @param privateKey
 *
 * @param data
 *
 * @return
 * @throws TrustSDKException TrustSDKException
 * @since JDK 1.7
 */
public static String signString(String privateKey, byte[] data) throws TrustSDKException {
if (StringUtils.isEmpty(privateKey)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
        return ECDSAAlgorithm.sign(privateKey, data);
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.SIGN_ERROR.getRetCode(),
ErrorNum.SIGN_ERROR.getRetMsg(), e);
}
}

public static String signString(String privateKey, String data) throws TrustSDKException,
UnsupportedEncodingException {
```

```
return signString(privateKey, data.getBytes("UTF-8"));
}

/**
 * verifyString:. <br/>
 *
 * @author Rony
 * @param pubKey
 *
 * @param srcString
 *
 * @param sign
 *
 * @return  true: false:
 * @throws TrustSDKException TrustSDKException
 * @since JDK 1.7
 */
public static boolean verifyString(String pubKey, String srcString, String sign) throws
TrustSDKException {
if (StringUtils.isEmpty(pubKey) || StringUtils.isEmpty(srcString) || StringUtils.isEmpty(sign)) {
throw new TrustSDKException(ErrorNum.INVALID_PARAM_ERROR.getRetCode(),
ErrorNum.INVALID_PARAM_ERROR.getRetMsg());
}
try {
return ECDSAAlgorithm.verify(srcString, sign, pubKey);
} catch (Exception e) {
throw new TrustSDKException(ErrorNum.ECDSA_ENCRYPT_ERROR.getRetCode(),
ErrorNum.ECDSA_ENCRYPT_ERROR.getRetMsg(), e);
}
}



}

38:F:\git\coin\blockchain-
java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\BaseData.java
package com.mindata.blockchain.core.bean;

/**
 * @author wuweifeng wrote on 2017/10/23.
 */
public class BaseData {
```

```
private int code;
private String message;
private Object data;

@Override
public String toString() {
    return "BaseData{" +
            "code=" + code +
            ", message='" + message + '\'' +
            ", data=" + data +
            '}';
}

public BaseData setCode(ResultCode resultCode) {
    this.code = resultCode.code;
    return this;
}

public int getCode() {
    return code;
}

public BaseData setCode(int code) {
    this.code = code;
    return this;
}

public String getMessage() {
    return message;
}

public BaseData setMessage(String message) {
    this.message = message;
    return this;
}

public Object getData() {
    return data;
}

public BaseData setData(Object data) {
    this.data = data;
```

```
        return this;
    }
}
```

39:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\Member.java

```
package com.mindata.blockchain.core.bean;


import java.util.Date;

/**
 *
 * @author wuweifeng wrote on 2018/3/5.
 */
public class Member {
    /**
     * idappId
     */
    private String appId;
    /**
     *
     */
    private String name;
    /**
     * ip
     */
    private String ip;

    private Date createTime;

    private Date updateTime;

    @Override
    public String toString() {
        return "Member{" +
                "appId='" + appId + '\'' +
                ", name='" + name + '\'' +
                ", ip='" + ip + '\'' +
                '}';
    }
```

```
    public String getAppId() {
        return appId;
    }

    public void setAppId(String appId) {
        this.appId = appId;
    }

    public String getName() {
        return name;
    }

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

    public String getIp() {
        return ip;
    }

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

}
```

40:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\MemberData.java

```
package com.mindata.blockchain.core.bean;

import java.util.List;

/**
 * @author wuweifeng wrote on 2018/3/19.
 */
public class MemberData {
    private int code;
    private String message;
    private List<Member> members;

    public int getCode() {
        return code;
```

```
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public List<Member> getMembers() {
        return members;
    }

    public void setMembers(List<Member> members) {
        this.members = members;
    }
}
```

41:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\Permission.java

```
package com.mindata.blockchain.core.bean;

/**
 * memberADDUPDATEDELETE
 * @author wuweifeng wrote on 2018/3/5.
 */
public class Permission {
    /**
     *
     */
    private String tableName;
    /**
     * PermissionType
     */
    private byte permissionType;
    /**
     * member*
```

```
 */
private String publicKey;
/**
 * groupgroupgroup
 */
private String groupId;

@Override
public String toString() {
    return "Permission{" +
            "tableName='" + tableName + '\'' +
            ", permissionType=" + permissionType +
            ", publicKey='" + publicKey + '\'' +
            ", groupId='" + groupId + '\'' +
            '}';
}

public String getPublicKey() {
    return publicKey;
}

public void setPublicKey(String publicKey) {
    this.publicKey = publicKey;
}

public String getGroupId() {
    return groupId;
}

public void setGroupId(String groupId) {
    this.groupId = groupId;
}

public String getTableName() {
    return tableName;
}

public void setTableName(String tableName) {
    this.tableName = tableName;
}

public byte getPermissionType() {
```

```
        return permissionType;
    }

    public void setPermissionType(byte permissionType) {
        this.permissionType = permissionType;
    }
}
```

42:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\PermissionData.java

```
package com.mindata.blockchain.core.bean;

import java.util.List;

/**
 * @author wuweifeng wrote on 2018/4/10.
 */
public class PermissionData extends BaseData {
    private List<Permission> permissions;

    public List<Permission> getPermissions() {
        return permissions;
    }

    public void setPermissions(List<Permission> permissions) {
        this.permissions = permissions;
    }
}
```

43:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\ResultCode.java

```
package com.mindata.blockchain.core.bean;

/**
 * @author wuweifeng wrote on 2017/10/23.
 */
public enum ResultCode {
    //
    SUCCESS(200),
    //
    FAIL(400),
    //
```

```
    UNAUTHORIZED(401),
    //
    NO_LOGIN(402),
    //
    NO_PERMISSION(403),
    //
    NOT_FOUND(404),
    //
    STATE_ERROR(406),
    //
    INTERNAL_SERVER_ERROR(500),
    //
    PARAMETER_ERROR(10001),
    //
    ACCOUNT_ERROR(20001),
    //
    LOGIN_FAIL_ERROR(20002);


    public int code;

    ResultCode(int code) {
        this.code = code;
    }
}
```

44:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\bean\ResultGenerator.java

```
package com.mindata.blockchain.core.bean;

/**
 * @author wuweifeng wrote on 2017/10/23.
 */
public class ResultGenerator {
    private static final String DEFAULT_SUCCESS_MESSAGE = "SUCCESS";

    public static BaseData genSuccessResult() {
        return new BaseData()
                .setCode(ResultCode.SUCCESS)
                .setMessage(DEFAULT_SUCCESS_MESSAGE);
    }
```

```
    public static BaseData genSuccessResult(Object data) {
        return new BaseData()
                .setCode(ResultCode.SUCCESS)
                .setMessage(DEFAULT_SUCCESS_MESSAGE)
                .setData(data);
    }

    public static BaseData genFailResult(String message) {
        return new BaseData()
                .setCode(ResultCode.FAIL)
                .setMessage(message);
    }

    public static BaseData genFailResult(ResultCode resultCode, String message) {
        return new BaseData()
                .setCode(resultCode)
                .setMessage(message);
    }
}
```

45:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\controller\BlockController.java

```
package com.mindata.blockchain.core.controller;

import cn.hutool.core.collection.CollectionUtil;
import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.block.Operation;
import com.mindata.blockchain.block.check.BlockChecker;
import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.core.bean.BaseData;
import com.mindata.blockchain.core.bean.ResultGenerator;
import com.mindata.blockchain.core.event.DbSyncEvent;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.core.manager.MessageManager;
import com.mindata.blockchain.core.manager.SyncManager;
import com.mindata.blockchain.core.model.MessageEntity;
import com.mindata.blockchain.core.requestbody.BlockRequestBody;
import com.mindata.blockchain.core.requestbody.InstructionBody;
import com.mindata.blockchain.core.service.BlockService;
import com.mindata.blockchain.core.service.InstructionService;
```

```
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import org.apache.commons.lang3.StringUtils;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.data.domain.Pageable;
import org.springframework.data.web.PageableDefault;
import org.springframework.util.ObjectUtils;
import org.springframework.web.bind.annotation.*;
import springfox.documentation.annotations.ApiIgnore;

import javax.annotation.Resource;

/**
 * @author wuweifeng wrote on 2018/3/7.
 */
@Api(tags = "", description = "")
@RestController
@RequestMapping("/block")
public class BlockController {
    @Resource
    private BlockService blockService;
    @Resource
    private PacketSender packetSender;
    @Resource
    private DbBlockManager dbBlockManager;
    @Resource
    private InstructionService instructionService;
    @Resource
    private SyncManager syncManager;
    @Resource
    private MessageManager messageManager;
    @Resource
    private BlockChecker blockChecker;
    @Value("${publicKey:A8WLqHTjcT/FQ2IWhIePNShUEcdCzu5dG+XrQU8OMu54}")
    private String publicKey;
    @Value("${privateKey:yScdp6fNgUU+cRUTygvJG4EBhDKmOMRrK4XJ9mKVQJ8=}")
```

```
    private String privateKey;

    /**
     * blockInstructionController1-NinstructionBlock
     *
     * @param blockRequestBody
     *
     * @return
     */
    @ApiIgnore
    @PostMapping("/insert")
    @ApiOperation(value = "", notes = "", httpMethod = "POST", response = BaseData.class)
    public BaseData insert(@ApiParam(name = "blockRequestBody", value = "json", required =
true) @RequestBody BlockRequestBody blockRequestBody) throws TrustSDKException {
    String msg = blockService.check(blockRequestBody);
        if (msg != null) {
            return ResultGenerator.genFailResult(msg);
        }
        return ResultGenerator.genSuccessResult(blockService.addBlock(blockRequestBody));
    }

    /**
     * insert:BlockPairKeyController
     * @param content
     * sql
     */
    @GetMapping("/create")
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    public BaseData create(@ApiParam(name = "content", value = "", required = true)
@RequestParam(value = "content") String content) throws Exception {
        InstructionBody instructionBody = new InstructionBody();
        instructionBody.setOperation(Operation.ADD);
        instructionBody.setTable("message");
        instructionBody.setJson("{\"content\":\"" + content + "\"}");
/*instructionBody.setPublicKey("A8WLqHTjcT/FQ2IWhIePNShUEcdCzu5dG+XrQU8OMu54");
 instructionBody.setPrivateKey("yScdp6fNgUU+cRUTygvJG4EBhDKmOMRrK4XJ9mKVQJ8=");*/
        instructionBody.setPublicKey(publicKey);
        instructionBody.setPrivateKey(privateKey);
        Instruction instruction = instructionService.build(instructionBody);

        BlockRequestBody blockRequestBody = new BlockRequestBody();
        blockRequestBody.setPublicKey(instructionBody.getPublicKey());
```

```
        com.mindata.blockchain.block.BlockBody blockBody = new
com.mindata.blockchain.block.BlockBody();
        blockBody.setInstructions(CollectionUtil.newArrayList(instruction));

        blockRequestBody.setBlockBody(blockBody);

        return ResultGenerator.genSuccessResult(blockService.addBlock(blockRequestBody));
    }

    /**
     * update:BlockPairKeyController
     * @param id
     * @param content
     * sql
     */
    @GetMapping("update")
    @ApiOperation(value = "", notes = "ID", httpMethod = "GET", response = BaseData.class)
    public BaseData testUpdate(@ApiParam(name = "id", value = "", required = true)
@RequestParam(value = "id",required = true) String id,
                               @ApiParam(name = "content", value = "", required = true)
@RequestParam(value = "content") String content) throws Exception {
    if(StringUtils.isBlank(id)) ResultGenerator.genSuccessResult("");
    InstructionBody instructionBody = new InstructionBody();
    instructionBody.setOperation(Operation.UPDATE);
    instructionBody.setTable("message");
    instructionBody.setInstructionId(id);
    instructionBody.setJson("{\"content\":\"" + content + "\"}");
     /*instructionBody.setPublicKey("A8WLqHTjcT/FQ2IWhIePNShUEcdCzu5dG+XrQU8OMu54");
instructionBody.setPrivateKey("yScdp6fNgUU+cRUTygvJG4EBhDKmOMRrK4XJ9mKVQJ8=");*/
     instructionBody.setPublicKey(publicKey);
        instructionBody.setPrivateKey(privateKey);
    Instruction instruction = instructionService.build(instructionBody);

    BlockRequestBody blockRequestBody = new BlockRequestBody();
    blockRequestBody.setPublicKey(instructionBody.getPublicKey());
    com.mindata.blockchain.block.BlockBody blockBody = new
com.mindata.blockchain.block.BlockBody();
    blockBody.setInstructions(CollectionUtil.newArrayList(instruction));

    blockRequestBody.setBlockBody(blockBody);

    return ResultGenerator.genSuccessResult(blockService.addBlock(blockRequestBody));
```

```
    }

    /**
     * delete:BlockPairKeyController
     * @param id
     * sql
     */
    @GetMapping("delete")
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    public BaseData delete(@ApiParam(name = "id", value = "", required = true)
@RequestParam(value = "id",required = true) String id) throws Exception {
    if(StringUtils.isBlank(id)) ResultGenerator.genSuccessResult("");
    InstructionBody instructionBody = new InstructionBody();
    instructionBody.setOperation(Operation.DELETE);
    instructionBody.setTable("message");
    instructionBody.setInstructionId(id);
        MessageEntity message=messageManager.findById(id);
        String content=ObjectUtils.isEmpty(message)?"":message.getContent();
        instructionBody.setJson("{\"content\":\"" + content + "\"}");
     /*instructionBody.setPublicKey("A8WLqHTjcT/FQ2IWhIePNShUEcdCzu5dG+XrQU8OMu54");
instructionBody.setPrivateKey("yScdp6fNgUU+cRUTygvJG4EBhDKmOMRrK4XJ9mKVQJ8=");*/
     instructionBody.setPublicKey(publicKey);
        instructionBody.setPrivateKey(privateKey);
    Instruction instruction = instructionService.build(instructionBody);

    BlockRequestBody blockRequestBody = new BlockRequestBody();
    blockRequestBody.setPublicKey(instructionBody.getPublicKey());
    com.mindata.blockchain.block.BlockBody blockBody = new
com.mindata.blockchain.block.BlockBody();
    blockBody.setInstructions(CollectionUtil.newArrayList(instruction));

    blockRequestBody.setBlockBody(blockBody);

    return ResultGenerator.genSuccessResult(blockService.addBlock(blockRequestBody));
    }

    /**
     * sqlite
     */
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    @GetMapping("sqlite")
    public BaseData sqlite() {
```

```
    return ResultGenerator.genSuccessResult(messageManager.findAll());
}

/**
 * sqlitecontent
 */
@ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
@GetMapping("sqlite/content")
public BaseData content() {
    return ResultGenerator.genSuccessResult(messageManager.findAllContent());
}

/**
 * block
 */
@ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
@GetMapping("last")
public BaseData last() {
    return ResultGenerator.genSuccessResult(dbBlockManager.getLastBlock());
}

/**
 * sqlsqlite
 * @param pageable
 *
 * @return
 *
 */
@ApiIgnore
@ApiOperation(value = "sqlsqlite", notes = "", httpMethod = "GET", response = BaseData.class)
@GetMapping("sync")
public BaseData sync( @PageableDefault Pageable pageable) {
    ApplicationContextProvider.publishEvent(new DbSyncEvent(""));
    return ResultGenerator.genSuccessResult(syncManager.findAll(pageable));
}

/**
 *
 * @return
 * null -
 * hash - hash
 */
```

```
    @ApiIgnore
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    @GetMapping("check")
    public BaseData check() {

    Block block = dbBlockManager.getFirstBlock();

    String hash = null;
    while(block != null && hash == null) {
    hash = blockChecker.checkBlock(block);
    block = dbBlockManager.getNextBlock(block);
    }
    return ResultGenerator.genSuccessResult(hash);
    }

    /**
     *
     */
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    @GetMapping("/first")
    public BaseData first() {
        Block block = dbBlockManager.getFirstBlock();
        BlockPacket packet = new PacketBuilder<RpcBlockBody>()
                .setType(PacketType.NEXT_BLOCK_INFO_REQUEST)
                .setBody(new RpcBlockBody(block)).build();
        packetSender.sendGroup(packet);
        return ResultGenerator.genSuccessResult(block);
    }

    /**
     * ID
     *
     * @param id
     */
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    @GetMapping("/find")
    public BaseData find(@ApiParam(name = "id", value = "", required = true)
@RequestParam(value = "id",required = true) String id) throws Exception {
        if(StringUtils.isBlank(id)) ResultGenerator.genSuccessResult("");
        InstructionBody instructionBody = new InstructionBody();
        instructionBody.setOperation(Operation.UPDATE);
        instructionBody.setTable("message");
```

```
        instructionBody.setInstructionId(id);
        MessageEntity message=messageManager.findById(id);
        String content=ObjectUtils.isEmpty(message)?"":message.getContent();
        instructionBody.setJson("{\"content\":\"" + content + "\"}");
/*instructionBody.setPublicKey("A8WLqHTjcT/FQ2IWhIePNShUEcdCzu5dG+XrQU8OMu54");
 instructionBody.setPrivateKey("yScdp6fNgUU+cRUTygvJG4EBhDKmOMRrK4XJ9mKVQJ8=");*/
        instructionBody.setPublicKey(publicKey);
        instructionBody.setPrivateKey(privateKey);
        Instruction instruction = instructionService.build(instructionBody);
        BlockRequestBody blockRequestBody = new BlockRequestBody();
        blockRequestBody.setPublicKey(instructionBody.getPublicKey());
        com.mindata.blockchain.block.BlockBody blockBody = new
com.mindata.blockchain.block.BlockBody();
        blockBody.setInstructions(CollectionUtil.newArrayList(instruction));

        blockRequestBody.setBlockBody(blockBody);

        return ResultGenerator.genSuccessResult(blockService.addBlock(blockRequestBody));
    }


}
```

46:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\controller\InstructionController.java

```
package com.mindata.blockchain.core.controller;

import com.mindata.blockchain.core.bean.BaseData;
import com.mindata.blockchain.core.bean.ResultGenerator;
import com.mindata.blockchain.core.requestbody.InstructionBody;
import com.mindata.blockchain.core.service.InstructionService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import springfox.documentation.annotations.ApiIgnore;

import javax.annotation.Resource;

/**
 * bodycontroller
```

```
 * @author wuweifeng wrote on 2018/3/7.
 */
@ApiIgnore
@RestController
@RequestMapping("/instruction")
public class InstructionController {
    @Resource
    private InstructionService instructionService;

    /**
     *
     * @param instructionBody instructionBody
     * @return
     *
     */
    @PostMapping
    public BaseData build(@RequestBody InstructionBody instructionBody) throws Exception {
        if (!instructionService.checkKeyPair(instructionBody)) {
            return ResultGenerator.genFailResult("");
        }
        if (!instructionService.checkContent(instructionBody)) {
            return ResultGenerator.genFailResult("DeleteUpdateidjson");
        }
        return ResultGenerator.genSuccessResult(instructionService.build(instructionBody));
    }
}
```

47:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\controller\PairKeyController.java

```
package com.mindata.blockchain.core.controller;

import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.core.bean.BaseData;
import com.mindata.blockchain.core.bean.ResultGenerator;
import com.mindata.blockchain.core.service.PairKeyService;
import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import javax.annotation.Resource;
```

```
/**
 * @author wuweifeng wrote on 2018/3/7.
 */
@Api(tags = "", description = "")
@RestController
@RequestMapping("/pairKey")
public class PairKeyController {
    @Resource
    private PairKeyService pairKeyService;

    /**
     *
     */
    @ApiOperation(value = "", notes = "", httpMethod = "GET", response = BaseData.class)
    @GetMapping("/random")
    public BaseData generate() throws TrustSDKException {
        return ResultGenerator.genSuccessResult(pairKeyService.generate());
    }
}
```

48:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\event\AddBlockEvent.java

```
package com.mindata.blockchain.core.event;

import com.mindata.blockchain.block.Block;
import org.springframework.context.ApplicationEvent;

/**
 * blockEventrocksDBsqlite
 * @author wuweifeng wrote on 2018/3/15.
 */
public class AddBlockEvent extends ApplicationEvent {
    public AddBlockEvent(Block block) {
        super(block);
    }
}
```

49:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\event\ClientRequestEvent.java

```
package com.mindata.blockchain.core.event;
```

```
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.springframework.context.ApplicationEvent;

/**
 * Event
 * @author wuweifeng wrote on 2018/3/17.
 */
public class ClientRequestEvent extends ApplicationEvent {
    public ClientRequestEvent(BlockPacket blockPacket) {
        super(blockPacket);
    }
}
```

50:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\event\DbSyncEvent.java

```
package com.mindata.blockchain.core.event;

import org.springframework.context.ApplicationEvent;

/**
 * blocksqlite
 * @author wuweifeng wrote on 2018/3/21.
 */
public class DbSyncEvent extends ApplicationEvent {
    public DbSyncEvent(Object source) {
        super(source);
    }
}
```

51:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\event\NodesConnectedEvent.java

```
package com.mindata.blockchain.core.event;

import org.springframework.context.ApplicationEvent;
import org.tio.core.ChannelContext;

/**
 * Event
 * @author andylo25 wrote on 2018/6/15.
 */
public class NodesConnectedEvent extends ApplicationEvent {
private static final long serialVersionUID = 526755692642414178L;
```

```
public NodesConnectedEvent(ChannelContext channelContext) {
        super(channelContext);
    }

public ChannelContext getSource() {
        return (ChannelContext) source;
    }

}
```

52:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\manager\DbBlockGenerator.java

```
package com.mindata.blockchain.core.manager;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.check.CheckerManager;
import com.mindata.blockchain.block.db.DbStore;
import com.mindata.blockchain.common.Constants;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.core.event.DbSyncEvent;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;
import org.tio.utils.json.Json;

import javax.annotation.Resource;

/**
 * block
 * @author wuweifeng wrote on 2018/4/25.
 */
@Service
public class DbBlockGenerator {
    @Resource
    private DbStore dbStore;
    @Resource
    private CheckerManager checkerManager;
```

```
    private Logger logger = LoggerFactory.getLogger(getClass());

    /**
     *
     *
     * @param addBlockEvent
     *         addBlockEvent
     */
    @Order(1)
    @EventListener(AddBlockEvent.class)
    public synchronized void addBlock(AddBlockEvent addBlockEvent) {
        logger.info("block");
        Block block = (Block) addBlockEvent.getSource();
        String hash = block.getHash();
        //Block
        if (dbStore.get(hash) != null) {
            return;
        }
        //
        if (checkerManager.check(block).getCode() != 0) {
            return;
        }

        //
        if (block.getBlockHeader().getHashPreviousBlock() == null) {
            dbStore.put(Constants.KEY_FIRST_BLOCK, hash);
        } else {
            //key value
            dbStore.put(Constants.KEY_BLOCK_NEXT_PREFIX +
block.getBlockHeader().getHashPreviousBlock(), hash);
        }
        //rocksDB
        dbStore.put(hash, Json.toJson(block));
        //blockkey value
        dbStore.put(Constants.KEY_LAST_BLOCK, hash);

        logger.info("Block");

        //sqlite
        sqliteSync();
    }
```

```
    /**
     * sqliteblocksql
     */
    @Async
    public void sqliteSync() {
        //sqlite
        ApplicationContextProvider.publishEvent(new DbSyncEvent(""));
    }
}
```

53:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\manager\DbBlockManager.java

```
package com.mindata.blockchain.core.manager;

import cn.hutool.core.util.StrUtil;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.db.DbStore;
import com.mindata.blockchain.common.Constants;
import com.mindata.blockchain.common.FastJsonUtil;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;

/**
 * @author wuweifeng wrote on 2018/3/13.
 */
@Service
public class DbBlockManager {
    @Resource
    private DbStore dbStore;

    /**
     *
     *
     * @return Block
     */
    public Block getFirstBlock() {
        String firstBlockHash = dbStore.get(Constants.KEY_FIRST_BLOCK);
        if (StrUtil.isEmpty(firstBlockHash)) {
            return null;
        }
        return getBlockByHash(firstBlockHash);
```

```
}

/**
 *
 *
 * @return
 */
public Block getLastBlock() {
    String lastBlockHash = dbStore.get(Constants.KEY_LAST_BLOCK);
    if (StrUtil.isEmpty(lastBlockHash)) {
        return null;
    }
    return getBlockByHash(lastBlockHash);
}

/**
 * hash
 *
 * @return hash
 */
public String getLastBlockHash() {
    Block block = getLastBlock();
    if (block != null) {
        return block.getHash();
    }
    return null;
}

/**
 * blocknumber
 * @return number
 */
public int getLastBlockNumber() {
    Block block = getLastBlock();
    if (block != null) {
        return block.getBlockHeader().getNumber();
    }
    return 0;
}

/**
 * blockBlock
```

```
     *
     * @param block
     *         block
     * @return block
     */
    public Block getNextBlock(Block block) {
        if (block == null) {
            return getFirstBlock();
        }
        String nextHash = dbStore.get(Constants.KEY_BLOCK_NEXT_PREFIX + block.getHash());
        if (nextHash == null) {
            return null;
        }
        return getBlockByHash(nextHash);
    }

    public Block getNextBlockByHash(String hash) {
        if (hash == null) {
            return getFirstBlock();
        }
        String nextHash = dbStore.get(Constants.KEY_BLOCK_NEXT_PREFIX + hash);
        if (nextHash == null) {
            return null;
        }
        return getBlockByHash(nextHash);
    }

    public Block getBlockByHash(String hash) {
        String blockJson = dbStore.get(hash);
        return FastJsonUtil.toBean(blockJson, Block.class);
    }

}
```

54:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\manager\MessageManager.java

```
package com.mindata.blockchain.core.manager;

import com.mindata.blockchain.core.model.MessageEntity;
import com.mindata.blockchain.core.repository.MessageRepository;
import org.springframework.stereotype.Component;
```

```
import javax.annotation.Resource;
import java.util.List;
import java.util.stream.Collectors;

/**
 * @author wuweifeng wrote on 2018/3/28.
 */
@Component
public class MessageManager {
    @Resource
    private MessageRepository messageRepository;

    public List<MessageEntity> findAll() {
        return messageRepository.findAll();
    }

    public List<String> findAllContent() {
        return findAll().stream().map(MessageEntity::getContent).collect(Collectors.toList());
    }

    public MessageEntity findById(String id) {
        return messageRepository.findByMessageId(id);
    }
}
```

55:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\manager\PermissionManager.java

```
package com.mindata.blockchain.core.manager;

import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.common.PermissionType;
import com.mindata.blockchain.core.bean.Permission;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;

import java.util.*;

/**
 * Permission
```

```
 *
 * @author wuweifeng wrote on 2018/4/10.
 */
@Service
public class PermissionManager {
    private Logger logger = LoggerFactory.getLogger(getClass());

    /**
     *
     */
    public static final Map<String, List<Permission>> PERMISSION_MAP = new HashMap<>();

    /**
     * block
     * @param block
     * @return
     */
    public boolean checkPermission(Block block) {
        List<Instruction> instructions = block.getBlockBody().getInstructions();
        return checkPermission(instructions);
    }

    public boolean checkPermission(List<Instruction> instructions) {
        for (Instruction instruction : instructions) {
            String publicKey = instruction.getPublicKey();
            String tableName = instruction.getTable();
            byte operation = instruction.getOperation();
            //TODO
            if (!checkOperation(publicKey, tableName, operation)) {
                return false;
            }
        }
        return true;
    }


    /**
     *
     *
     * @param publicKey
     *
     * @param tableName
```

```
     *
     * @param operation
     *
     * @return true
     */
    private boolean checkOperation(String publicKey, String tableName, byte operation) {
        List<Permission> permissionList = PERMISSION_MAP.get(tableName);

        Set<Byte> userPermissionSet = new HashSet<>();
        for (Permission permission : permissionList) {
            //
            if ("*".equals(permission.getPublicKey())) {
                userPermissionSet.add(permission.getPermissionType());
            } else {
                //publicKey
                if (publicKey.equals(permission.getPublicKey())) {
                    userPermissionSet.add(permission.getPermissionType());
                }
            }
        }

        //operation
        return userPermissionSet.contains(PermissionType.OWNER)
                || userPermissionSet.contains(PermissionType.ALL)
                || userPermissionSet.contains(operation);
    }


    /**
     * statictablemap
     *
     * @param permissions
     *         permissions
     */
    public void savePermissionList(List<Permission> permissions) {
        PERMISSION_MAP.clear();
        for (Permission permission : permissions) {
            String key = permission.getTableName();
            if (!PERMISSION_MAP.containsKey(key)) {
                PERMISSION_MAP.put(key, new ArrayList<>());
            }
            PERMISSION_MAP.get(key).add(permission);
```

```
        }
        logger.info("" + PERMISSION_MAP);
    }

}
```

56:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\manager\SyncManager.java

```
package com.mindata.blockchain.core.manager;

import com.mindata.blockchain.core.model.SyncEntity;
import com.mindata.blockchain.core.repository.SyncRepository;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;

/**
 * @author wuweifeng wrote on 2018/3/21.
 */
@Service
public class SyncManager {
    @Resource
    private SyncRepository syncRepository;

    public SyncEntity findLastOne() {
        return syncRepository.findTopByOrderByIdDesc();
    }

    public SyncEntity save(SyncEntity syncEntity) {
        return syncRepository.save(syncEntity);
    }

    public Object findAll(Pageable pageable) {
        return syncRepository.findAll(pageable);
    }

    public void deleteAll() {
        syncRepository.deleteAll();
    }
}
```

57:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\model\base\BaseEntity.java

```
package com.mindata.blockchain.core.model.base;

import com.mindata.blockchain.common.CommonUtil;

import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;

/**
 * @author wuweifeng wrote on 2018/3/2.
 */
@MappedSuperclass
public class BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    private Long createTime;

    private Long updateTime = CommonUtil.getNow();
    /**
     *
     */
    private String publicKey;

    public String getPublicKey() {
        return publicKey;
    }

    public void setPublicKey(String publicKey) {
        this.publicKey = publicKey;
    }

    public Long getCreateTime() {
        return createTime;
    }

    public void setCreateTime(Long createTime) {
        this.createTime = createTime;
```

```
    }

    public Long getUpdateTime() {
        return updateTime;
    }

    public void setUpdateTime(Long updateTime) {
        this.updateTime = updateTime;
    }

    public Long getId() {
        return id;
    }

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

    @Override
    public String toString() {
        return "BaseEntity{" +
                "id=" + id +
                ", createTime=" + createTime +
                ", updateTime=" + updateTime +
                ", publicKey='" + publicKey + '\'' +
                '}';
    }
}
```

58:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\model\convert\ConvertTableName.java

```
package com.mindata.blockchain.core.model.convert;

import com.mindata.blockchain.core.model.base.BaseEntity;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.Map;

/**
```

```
 *
 * @author wuweifeng wrote on 2018/3/2.
 */
@Component
public class ConvertTableName<T extends BaseEntity> {
    @Qualifier(value = "metaMap")
    @Resource
    private Map<String, Class<T>> metaMap;

    /**
     * class
     * @return
     *
     */
    public Class<T> convertOf(String tableName) {
        return metaMap.get(tableName);
    }

    /**
     *
     * @param clazz
     *
     * @return
     *
     */
    public String convertOf(Class<T> clazz) {
        for (String key : metaMap.keySet()) {
            if (metaMap.get(key).equals(clazz)) {
                return key;
            }
        }
        return null;
    }
}
```

59:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\model\MessageEntity.java

```
package com.mindata.blockchain.core.model;

import com.mindata.blockchain.core.model.base.BaseEntity;

import javax.persistence.Entity;
```

```
import javax.persistence.Table;

/**
 * @author wuweifeng wrote on 2017/10/25.
 */
@Entity
@Table(name = "message")
public class MessageEntity extends BaseEntity {
    /**
     *
     */
    private String content;
    /**
     *
     */
    private String target;
    /**
     *
     */
    private String origin;
    /**
     * id
     */
    private String messageId;

    @Override
    public String toString() {
        return "MessageEntity{" +
                "content='" + content + '\'' +
                ", target='" + target + '\'' +
                ", origin='" + origin + '\'' +
                ", messageId='" + messageId + '\'' +
                '}';
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }
```

```
    public String getMessageId() {
        return messageId;
    }

    public void setMessageId(String messageId) {
        this.messageId = messageId;
    }

    public String getTarget() {
        return target;
    }

    public void setTarget(String target) {
        this.target = target;
    }

    public String getOrigin() {
        return origin;
    }

    public void setOrigin(String origin) {
        this.origin = origin;
    }
}
```

60:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\model\SyncEntity.java

```
package com.mindata.blockchain.core.model;

import com.mindata.blockchain.common.CommonUtil;

import javax.persistence.*;

/**
 * @author wuweifeng wrote on 2017/10/25.
 */
@Entity
@Table(name = "sync")
public class SyncEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
```

```
private Long id;
/**
 * hash
 */
private String hash;
/**
 *
 */
private Long createTime = CommonUtil.getNow();

@Override
public String toString() {
    return "AsyncEntity{" +
            "id=" + id +
            ", hash='" + hash + '\'' +
            ", createTime=" + createTime +
            '}';
}

public Long getId() {
    return id;
}

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

public String getHash() {
    return hash;
}

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

public Long getCreateTime() {
    return createTime;
}

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

```
}
```

61:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\repository\BaseRepository.java

```
package com.mindata.blockchain.core.repository;

import com.mindata.blockchain.core.model.base.BaseEntity;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.NoRepositoryBean;

/**
 * @author wuweifeng wrote on 2017/10/25.
 */
@NoRepositoryBean
public interface BaseRepository<T extends BaseEntity> extends JpaRepository<T, Long> {

}
```

62:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\repository\MessageRepository.java

```
package com.mindata.blockchain.core.repository;

import com.mindata.blockchain.core.model.MessageEntity;

/**
 * @author wuweifeng wrote on 2017/10/25.
 */
public interface MessageRepository extends BaseRepository<MessageEntity> {
    /**
     *
     * @param messageId  messageId
     */
    void deleteByMessageId(String messageId);

    /**
     *
     * @param messageId messageId
     * @return MessageEntity
     */
    MessageEntity findByMessageId(String messageId);
}
```

63:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\repository\SyncRepository.java

```
package com.mindata.blockchain.core.repository;

import com.mindata.blockchain.core.model.SyncEntity;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * @author wuweifeng wrote on 2017/10/25.
 */
public interface SyncRepository extends JpaRepository<SyncEntity, Long> {
    SyncEntity findTopByOrderByIdDesc();
}
```

64:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\requestbody\BlockRequestBody.java

```
package com.mindata.blockchain.core.requestbody;

import com.mindata.blockchain.block.BlockBody;

/**
 * Block
 * @author wuweifeng wrote on 2018/3/8.
 */
public class BlockRequestBody {
    private String publicKey;
    private BlockBody blockBody;

    @Override
    public String toString() {
        return "BlockRequestBody{" +
                "publicKey='" + publicKey + '\'' +
                ", blockBody=" + blockBody +
                '}';
    }

    public String getPublicKey() {
        return publicKey;
    }
```

```
    public void setPublicKey(String publicKey) {
        this.publicKey = publicKey;
    }

    public BlockBody getBlockBody() {
        return blockBody;
    }

    public void setBlockBody(BlockBody blockBody) {
        this.blockBody = blockBody;
    }
}
```

65:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\requestbody\InstructionBody.java

```
package com.mindata.blockchain.core.requestbody;

/**
 * @author wuweifeng wrote on 2018/3/7.
 */
public class InstructionBody {
    /**
     *
     */
    private byte operation;
    /**
     *
     */
    private String table;
    /**
     *
     */
    private String json;
    /**
     *
     */
    private String oldJson;
    /**
     * id
     */
    private String instructionId;
    /**
```

```
 *
 */
private String privateKey;
/**
 *
 */
private String publicKey;

@Override
public String toString() {
    return "InstructionBody{" +
            "operation=" + operation +
            ", table='" + table + '\'' +
            ", json='" + json + '\'' +
            ", oldJson='" + oldJson + '\'' +
            ", instructionId='" + instructionId + '\'' +
            ", privateKey='" + privateKey + '\'' +
            ", publicKey='" + publicKey + '\'' +
            '}';
}

public String getOldJson() {
    return oldJson;
}

public void setOldJson(String oldJson) {
    this.oldJson = oldJson;
}

public String getInstructionId() {
    return instructionId;
}

public void setInstructionId(String instructionId) {
    this.instructionId = instructionId;
}

public String getPublicKey() {
    return publicKey;
}

public void setPublicKey(String publicKey) {
```

```
        this.publicKey = publicKey;
    }

    public byte getOperation() {
        return operation;
    }

    public void setOperation(byte operation) {
        this.operation = operation;
    }

    public String getTable() {
        return table;
    }

    public void setTable(String table) {
        this.table = table;
    }

    public String getJson() {
        return json;
    }

    public void setJson(String json) {
        this.json = json;
    }

    public String getPrivateKey() {
        return privateKey;
    }

    public void setPrivateKey(String privateKey) {
        this.privateKey = privateKey;
    }
}
```

66:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\resthttp\RestTemplateConfig.java

```
package com.mindata.blockchain.core.resthttp;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.http.client.ClientHttpRequestFactory;
import org.springframework.http.client.SimpleClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;

/**
 * @author wuweifeng wrote on 2018/3/19.
 */
@Configuration
public class RestTemplateConfig {

    @Bean
    public RestTemplate restTemplate(ClientHttpRequestFactory factory) {
        return new RestTemplate(factory);
    }

    @Bean
    public ClientHttpRequestFactory simpleClientHttpRequestFactory() {
        SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory();
        factory.setReadTimeout(5000);
        factory.setConnectTimeout(5000);
        return factory;
    }
}
```

67:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\service\BlockService.java

```
package com.mindata.blockchain.core.service;

import cn.hutool.core.collection.CollectionUtil;
import cn.hutool.core.util.StrUtil;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.BlockHeader;
import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.block.merkle.MerkleTree;
import com.mindata.blockchain.common.CommonUtil;
import com.mindata.blockchain.common.Sha256;
import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.core.manager.PermissionManager;
import com.mindata.blockchain.core.requestbody.BlockRequestBody;
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.client.PacketSender;
```

```
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.util.List;
import java.util.stream.Collectors;

/**
 * @author wuweifeng wrote on 2018/3/8.
 */
@Service
public class BlockService {
    @Resource
    private InstructionService instructionService;
    @Value("${version}")
    private int version;
    @Resource
    private PacketSender packetSender;
    @Resource
    private DbBlockManager dbBlockManager;
    @Resource
    private PermissionManager permissionManager;

    /**
     *
     *
     * @param blockRequestBody
     *
     * @return null
     */
    public String check(BlockRequestBody blockRequestBody) throws TrustSDKException {
        //TODO publicKey
        if (blockRequestBody == null || blockRequestBody.getBlockBody() == null ||
StrUtil.isEmpty(blockRequestBody
                .getPublicKey())) {
            return "";
        }
        List<Instruction> instructions = blockRequestBody.getBlockBody().getInstructions();
        if (CollectionUtil.isEmpty(instructions)) {
```

```
            return "";
        }

        for (Instruction instruction : instructions) {
            if (!StrUtil.equals(blockRequestBody.getPublicKey(), instruction.getPublicKey())) {
                return "";
            }
            if (!instructionService.checkSign(instruction)) {
                return "";
            }
            if (!instructionService.checkHash(instruction)) {
                return "Hash";
            }
        }

        if (!permissionManager.checkPermission(instructions)) {
            return "";
        }

        return null;
    }

    /**
     *
     * @param blockRequestBody blockRequestBody
     * @return Block
     */
    public Block addBlock(BlockRequestBody blockRequestBody) {
        com.mindata.blockchain.block.BlockBody blockBody = blockRequestBody.getBlockBody();
        List<Instruction> instructions = blockBody.getInstructions();
        List<String> hashList = instructions.stream().map(Instruction::getHash).collect(Collectors
                .toList());

        BlockHeader blockHeader = new BlockHeader();
        blockHeader.setHashList(hashList);

        //hashRoot
        blockHeader.setHashMerkleRoot(new MerkleTree(hashList).build().getRoot());
        blockHeader.setPublicKey(blockRequestBody.getPublicKey());
        blockHeader.setTimeStamp(CommonUtil.getNow());
        blockHeader.setVersion(version);
        blockHeader.setNumber(dbBlockManager.getLastBlockNumber() + 1);
```

```
        blockHeader.setHashPreviousBlock(dbBlockManager.getLastBlockHash());
        Block block = new Block();
        block.setBlockBody(blockBody);
        block.setBlockHeader(blockHeader);
        block.setHash(Sha256.sha256(blockHeader.toString() + blockBody.toString()));

        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.GENERATE_BLOCK_REQUEST).setBody(new
                RpcBlockBody(block)).build();

        //
        packetSender.sendGroup(blockPacket);

        return block;
    }

}
```

68:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\service\InstructionService.java

```
package com.mindata.blockchain.core.service;

import cn.hutool.core.bean.BeanUtil;
import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.block.InstructionReverse;
import com.mindata.blockchain.block.Operation;
import com.mindata.blockchain.common.CommonUtil;
import com.mindata.blockchain.common.Sha256;
import com.mindata.blockchain.common.TrustSDK;
import com.mindata.blockchain.common.exception.TrustSDKException;
import com.mindata.blockchain.core.requestbody.InstructionBody;
import org.springframework.stereotype.Service;

/**
 * service
 *
 * @author wuweifeng wrote on 2018/3/7.
 */
@Service
public class InstructionService {
    /**
     *
```

```
     *
     * @param instructionBody
     *        instructionBody
     * @return boolean
     * @throws TrustSDKException
     *         TrustSDKException
     */
    public boolean checkKeyPair(InstructionBody instructionBody) throws TrustSDKException {
        return TrustSDK.checkPairKey(instructionBody.getPrivateKey(),
instructionBody.getPublicKey());
    }

    /**
     *
     * @param instructionBody instructionBody
     * @return true false
     */
    public boolean checkContent(InstructionBody instructionBody) {
        byte operation = instructionBody.getOperation();
        if (operation != Operation.ADD && operation != Operation.DELETE && operation !=
Operation.UPDATE) {
            return false;
        }
        //addidjsonjson
        return Operation.UPDATE != operation && Operation.DELETE != operation ||
instructionBody.getInstructionId()
                != null && instructionBody.getJson() != null && instructionBody.getOldJson() != null;
    }

    /**
     * body
     *
     * @param instructionBody
     *        body
     * @return Instruction
     */
    public Instruction build(InstructionBody instructionBody) throws Exception {
        Instruction instruction = new Instruction();
        BeanUtil.copyProperties(instructionBody, instruction);
        if (Operation.ADD == instruction.getOperation()) {
            instruction.setInstructionId(CommonUtil.generateUuid());
        }
```

```
    instruction.setTimeStamp(CommonUtil.getNow());
    String buildStr = getSignString(instruction);
    //
    instruction.setSign(TrustSDK.signString(instructionBody.getPrivateKey(), buildStr));
    //hash
    instruction.setHash(Sha256.sha256(buildStr));

    return instruction;
}

private String getSignString(Instruction instruction) {
return instruction.getOperation() + instruction.getTable() + instruction
.getInstructionId() + (instruction.getJson()==null?"":instruction.getJson());
}

/**
 * <p>
 * add table1 {id:xxx, name:"123"}delete table1 {id:xxx}
 * delete table2 id2 oldJson:{id:xxx, name:"123"}add table2 {id:xxx, name:"123"}
 * update table3 id3 json:{id:xxx, name:"123"} oldJson:{id:xxx, name:"456"}
 * json
 *
 * @param instruction
 *         instruction
 * @return
 */
public InstructionReverse buildReverse(Instruction instruction) {
    InstructionReverse instructionReverse = new InstructionReverse();
    BeanUtil.copyProperties(instruction, instructionReverse);

    if (Operation.ADD == instruction.getOperation()) {
        instructionReverse.setOperation(Operation.DELETE);
    } else if (Operation.DELETE == instruction.getOperation()) {
        instructionReverse.setOperation(Operation.ADD);
    }

    return instructionReverse;
}

public boolean checkSign(Instruction instruction) throws TrustSDKException {
    String buildStr = getSignString(instruction);
    return TrustSDK.verifyString(instruction.getPublicKey(), buildStr, instruction.getSign());
```

```
    }

    public boolean checkHash(Instruction instruction) {
        String buildStr = getSignString(instruction);
        return Sha256.sha256(buildStr).equals(instruction.getHash());
    }
}
```

69:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\service\PairKeyService.java

```
package com.mindata.blockchain.core.service;

import com.mindata.blockchain.block.PairKey;
import com.mindata.blockchain.common.TrustSDK;
import com.mindata.blockchain.common.exception.TrustSDKException;
import org.springframework.stereotype.Service;

/**
 * @author wuweifeng wrote on 2018/3/7.
 */
@Service
public class PairKeyService {

    /**
     *
     * @return PairKey
     * @throws TrustSDKException TrustSDKException
     */
    public PairKey generate() throws TrustSDKException {
        return TrustSDK.generatePairKey(true);
    }
}
```

70:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\DataSourceConfiguration.java

```
package com.mindata.blockchain.core.sqlite.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.jdbc.DataSourceBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
import org.sqlite.SQLiteDataSource;

import javax.sql.DataSource;


/**
 * sqliteDataSource
 * @author wuweifeng wrote on 2018/3/2.
 */
@Configuration
public class DataSourceConfiguration {
    @Value("${sqlite.dbName}")
    private String dbName;

    @Bean(destroyMethod = "", name = "EmbeddeddataSource")
    public DataSource dataSource() {
        DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
        dataSourceBuilder.driverClassName("org.sqlite.JDBC");
        dataSourceBuilder.url("jdbc:sqlite:" + dbName);
        dataSourceBuilder.type(SQLiteDataSource.class);
        return dataSourceBuilder.build();
    }

}
```

71:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\identity\SQLiteDialectIdentityColumnSupport.java

```
package com.mindata.blockchain.core.sqlite.config.identity;

import org.hibernate.dialect.Dialect;
import org.hibernate.dialect.identity.IdentityColumnSupportImpl;

/**
 * @author wuweifeng wrote on 2018/3/2.
 */
public class SQLiteDialectIdentityColumnSupport extends IdentityColumnSupportImpl {
    public SQLiteDialectIdentityColumnSupport(Dialect dialect) {
        super(dialect);
    }

    @Override
```

```
    public boolean supportsIdentityColumns() {
        return true;
    }

  /*
public boolean supportsInsertSelectIdentity() {
    return true; // As specified in NHibernate dialect
  }
  */

    @Override
    public boolean hasDataTypeInIdentityColumn() {
        // As specified in NHibernate dialect
        // FIXME true
        return false;
    }

  /*
public String appendIdentitySelectToInsert(String insertString) {
    return new StringBuffer(insertString.length()+30). // As specified in NHibernate dialect
      append(insertString).
      append("; ").append(getIdentitySelectString()).
      toString();
  }
  */

    @Override
    public String getIdentitySelectString(String table, String column, int type) {
        return "select last_insert_rowid()";
    }

    @Override
    public String getIdentityColumnString(int type) {
        // return "integer primary key autoincrement";
        // FIXME "autoincrement"
        return "integer";
    }
}
```

72:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\JpaConfiguration.java

```
package com.mindata.blockchain.core.sqlite.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.autoconfigure.orm.jpa.JpaProperties;
import org.springframework.boot.orm.jpa.EntityManagerFactoryBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.annotation.Resource;
import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;
import java.util.Map;

/**
 * @author wuweifeng wrote on 2018/3/2.
 */
@Configuration
@EnableJpaRepositories(
        basePackages = "com.mindata.blockchain.core.repository",
        transactionManagerRef = "jpaTransactionManager",
        entityManagerFactoryRef = "localContainerEntityManagerFactoryBean"
)
@EnableTransactionManagement
public class JpaConfiguration {
    @Resource
    private JpaProperties jpaProperties;

    @Autowired
    @Bean
    public JpaTransactionManager jpaTransactionManager(@Qualifier(value =
"EmbeddeddataSource") DataSource
                                               dataSource, EntityManagerFactory
                                               entityManagerFactory) {
        JpaTransactionManager jpaTransactionManager
                = new JpaTransactionManager();
        jpaTransactionManager.setEntityManagerFactory(entityManagerFactory);
        jpaTransactionManager.setDataSource(dataSource);
```

```
        return jpaTransactionManager;
    }

    @Autowired
    @Bean
    LocalContainerEntityManagerFactoryBean
localContainerEntityManagerFactoryBean(@Qualifier(value =
            "EmbeddeddataSource") DataSource dataSource, EntityManagerFactoryBuilder builder) {
        return builder.dataSource(dataSource)
                .packages("com.mindata.blockchain.core.model")
                .properties(getVendorProperties(dataSource))
                .build();
    }

    private Map<String, String> getVendorProperties(DataSource dataSource) {
        return jpaProperties.getHibernateProperties(dataSource);
    }

}
```

73:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\ModelMetaData.java

```
package com.mindata.blockchain.core.sqlite.config;

import org.hibernate.SessionFactory;
import org.hibernate.metadata.ClassMetadata;
import org.hibernate.persister.entity.AbstractEntityPersister;
import org.springframework.boot.autoconfigure.AutoConfigureAfter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.persistence.EntityManagerFactory;
import java.util.HashMap;
import java.util.Map;

/**
 * modelclassaccount_entity:AccountEntity.class
 *
 * @author wuweifeng wrote on 2018/3/2.
 */
@Configuration
```

```
@AutoConfigureAfter(JpaConfiguration.class)
public class ModelMetaData {

    @Bean(name = "metaMap")
    public Map<String, Class> metaMap(EntityManagerFactory factory) throws
ClassNotFoundException {
        if (factory.unwrap(SessionFactory.class) == null) {
            throw new NullPointerException("factory is not a hibernate factory");
        }
        SessionFactory sessionFactory = factory.unwrap(SessionFactory.class);
        Map<String, ClassMetadata> metaMap = sessionFactory.getAllClassMetadata();
        Map<String, Class> map = new HashMap<>(metaMap.size());
        for (String key : metaMap.keySet()) {
            AbstractEntityPersister classMetadata = (AbstractEntityPersister) metaMap
                    .get(key);
            String tableName = classMetadata.getTableName().toLowerCase();
            int index = tableName.indexOf(".");
            if (index >= 0) {
                tableName = tableName.substring(index + 1);
            }
            map.put(tableName, Class.forName(key));
        }
        return map;
    }

}
```

74:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\SQLiteDialect.java

```
package com.mindata.blockchain.core.sqlite.config;

import com.mindata.blockchain.core.sqlite.config.identity.SQLiteDialectIdentityColumnSupport;
import org.hibernate.JDBCException;
import org.hibernate.ScrollMode;
import org.hibernate.dialect.Dialect;
import org.hibernate.dialect.function.*;
import org.hibernate.dialect.identity.IdentityColumnSupport;
import org.hibernate.dialect.pagination.AbstractLimitHandler;
import org.hibernate.dialect.pagination.LimitHandler;
import org.hibernate.dialect.pagination.LimitHelper;
import org.hibernate.dialect.unique.DefaultUniqueDelegate;
import org.hibernate.dialect.unique.UniqueDelegate;
```

```
import org.hibernate.engine.spi.RowSelection;
import org.hibernate.exception.DataException;
import org.hibernate.exception.JDBCConnectionException;
import org.hibernate.exception.LockAcquisitionException;
import org.hibernate.exception.spi.SQLExceptionConversionDelegate;
import org.hibernate.exception.spi.TemplatedViolatedConstraintNameExtracter;
import org.hibernate.exception.spi.ViolatedConstraintNameExtracter;
import org.hibernate.internal.util.JdbcExceptionHelper;
import org.hibernate.mapping.Column;
import org.hibernate.type.StandardBasicTypes;

import java.sql.SQLException;
import java.sql.Types;

/**
 * sqliteHibernate jpaSQLite
 * @author wuweifeng wrote on 2018/3/2.
 */
public class SQLiteDialect extends Dialect {
    private final UniqueDelegate uniqueDelegate;

    public SQLiteDialect() {
        registerColumnType(Types.BIT, "boolean");
        //registerColumnType(Types.FLOAT, "float");
        //registerColumnType(Types.DOUBLE, "double");
        registerColumnType(Types.DECIMAL, "decimal");
        registerColumnType(Types.CHAR, "char");
        registerColumnType(Types.LONGVARCHAR, "longvarchar");
        registerColumnType(Types.TIMESTAMP, "datetime");
        registerColumnType(Types.BINARY, "blob");
        registerColumnType(Types.VARBINARY, "blob");
        registerColumnType(Types.LONGVARBINARY, "blob");

        registerFunction("concat", new VarArgsSQLFunction(StandardBasicTypes.STRING, "", "||",
""));
        registerFunction("mod", new SQLFunctionTemplate(StandardBasicTypes.INTEGER, "?1 %
?2"));
        registerFunction("quote", new StandardSQLFunction("quote",
StandardBasicTypes.STRING));
        registerFunction("random", new NoArgSQLFunction("random",
StandardBasicTypes.INTEGER));
        registerFunction("round", new StandardSQLFunction("round"));
```

```
        registerFunction("substr", new StandardSQLFunction("substr",
StandardBasicTypes.STRING));
        registerFunction("trim", new AbstractAnsiTrimEmulationFunction() {
            @Override
            protected SQLFunction resolveBothSpaceTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "trim(?1)");
            }

            @Override
            protected SQLFunction resolveBothSpaceTrimFromFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "trim(?2)");
            }

            @Override
            protected SQLFunction resolveLeadingSpaceTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "ltrim(?1)");
            }

            @Override
            protected SQLFunction resolveTrailingSpaceTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "rtrim(?1)");
            }

            @Override
            protected SQLFunction resolveBothTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "trim(?1, ?2)");
            }

            @Override
            protected SQLFunction resolveLeadingTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "ltrim(?1, ?2)");
            }

            @Override
            protected SQLFunction resolveTrailingTrimFunction() {
                return new SQLFunctionTemplate(StandardBasicTypes.STRING, "rtrim(?1, ?2)");
            }
        });
        uniqueDelegate = new SQLiteUniqueDelegate(this);
    }

    // database type mapping support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
/*@Override
public String getCastTypeName(int code) {
// http://sqlite.org/lang_expr.html#castexpr
return super.getCastTypeName( code );
}*/

    // IDENTITY support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    private static final SQLiteDialectIdentityColumnSupport IDENTITY_COLUMN_SUPPORT = new
            SQLiteDialectIdentityColumnSupport(new SQLiteDialect());



    @Override
    public IdentityColumnSupport getIdentityColumnSupport() {
        return IDENTITY_COLUMN_SUPPORT;
    }

    // limit/offset support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    private static final AbstractLimitHandler LIMIT_HANDLER = new AbstractLimitHandler() {
        @Override
        public String processSql(String sql, RowSelection selection) {
            final boolean hasOffset = LimitHelper.hasFirstRow(selection);
            return sql + (hasOffset ? " limit ? offset ?" : " limit ?");
        }

        @Override
        public boolean supportsLimit() {
            return true;
        }

        @Override
        public boolean bindLimitParametersInReverseOrder() {
            return true;
        }
    };

    @Override
    public LimitHandler getLimitHandler() {
        return LIMIT_HANDLER;
    }
```

```
// lock acquisition support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
@Override
public boolean supportsLockTimeouts() {
    // may be http://sqlite.org/c3ref/db_mutex.html ?
    return false;
}

@Override
public String getForUpdateString() {
    return "";
}

@Override
public boolean supportsOuterJoinForUpdate() {
    return false;
}

// current timestamp support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

@Override
public boolean supportsCurrentTimestampSelection() {
    return true;
}

@Override
public boolean isCurrentTimestampSelectStringCallable() {
    return false;
}

@Override
public String getCurrentTimestampSelectString() {
    return "select current_timestamp";
}

// SQLException support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

private static final int SQLITE_BUSY = 5;
private static final int SQLITE_LOCKED = 6;
private static final int SQLITE_IOERR = 10;
private static final int SQLITE_CORRUPT = 11;
private static final int SQLITE_NOTFOUND = 12;
private static final int SQLITE_FULL = 13;
```

```
    private static final int SQLITE_CANTOPEN = 14;
    private static final int SQLITE_PROTOCOL = 15;
    private static final int SQLITE_TOOBIG = 18;
    private static final int SQLITE_CONSTRAINT = 19;
    private static final int SQLITE_MISMATCH = 20;
    private static final int SQLITE_NOTADB = 26;

    @Override
    public SQLExceptionConversionDelegate buildSQLExceptionConversionDelegate() {
        return new SQLExceptionConversionDelegate() {
            @Override
            public JDBCException convert(SQLException sqlException, String message, String sql) {
                final int errorCode = JdbcExceptionHelper.extractErrorCode(sqlException) & 0xFF;
                if (errorCode == SQLITE_TOOBIG || errorCode == SQLITE_MISMATCH) {
                    return new DataException(message, sqlException, sql);
                } else if (errorCode == SQLITE_BUSY || errorCode == SQLITE_LOCKED) {
                    return new LockAcquisitionException(message, sqlException, sql);
                } else if ((errorCode >= SQLITE_IOERR && errorCode <= SQLITE_PROTOCOL) ||
errorCode == SQLITE_NOTADB) {
                    return new JDBCConnectionException(message, sqlException, sql);
                }

                // returning null allows other delegates to operate
                return null;
            }
        };
    }

    @Override
    public ViolatedConstraintNameExtracter getViolatedConstraintNameExtracter() {
        return EXTRACTER;
    }

    private static final ViolatedConstraintNameExtracter EXTRACTER = new
TemplatedViolatedConstraintNameExtracter() {
        @Override
        protected String doExtractConstraintName(SQLException sqle) throws
NumberFormatException {
            final int errorCode = JdbcExceptionHelper.extractErrorCode(sqle) & 0xFF;
            if (errorCode == SQLITE_CONSTRAINT) {
                return extractUsingTemplate("constraint ", " failed", sqle.getMessage());
            }
```

```
        return null;
    }
};

// union subclass support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

@Override
public boolean supportsUnionAll() {
    return true;
}

// DDL support ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

@Override
public boolean canCreateSchema() {
    return false;
}

@Override
public boolean hasAlterTable() {
    // As specified in NHibernate dialect
    return false;
}

@Override
public boolean dropConstraints() {
    return false;
}

@Override
public boolean qualifyIndexName() {
    return false;
}

@Override
public String getAddColumnString() {
    return "add column";
}

@Override
public String getDropForeignKeyString() {
    throw new UnsupportedOperationException("No drop foreign key syntax supported by
```

```
SQLiteDialect");
    }

    @Override
    public String getAddForeignKeyConstraintString(String constraintName,
                                                   String[] foreignKey, String referencedTable, String[] primaryKey,
                                                   boolean referencesPrimaryKey) {
        throw new UnsupportedOperationException("No add foreign key syntax supported by
SQLiteDialect");
    }

    @Override
    public String getAddPrimaryKeyConstraintString(String constraintName) {
        throw new UnsupportedOperationException("No add primary key syntax supported by
SQLiteDialect");
    }

    @Override
    public boolean supportsCommentOn() {
        return true;
    }

    @Override
    public boolean supportsIfExistsBeforeTableName() {
        return true;
    }

  /* not case insensitive for unicode characters by default (ICU extension needed)
public boolean supportsCaseInsensitiveLike() {
    return true;
  }
  */

    @Override
    public boolean doesReadCommittedCauseWritersToBlockReaders() {
        // TODO Validate (WAL mode...)
        return true;
    }

    @Override
    public boolean doesRepeatableReadCauseReadersToBlockWriters() {
        return true;
```

```
    }

    @Override
    public boolean supportsTupleDistinctCounts() {
        return false;
    }

    @Override
    public int getInExpressionCountLimit() {
        // Compile/runtime time option: http://sqlite.org/limits.html#max_variable_number
        return 1000;
    }

    @Override
    public UniqueDelegate getUniqueDelegate() {
        return uniqueDelegate;
    }

    private static class SQLiteUniqueDelegate extends DefaultUniqueDelegate {
        public SQLiteUniqueDelegate(Dialect dialect) {
            super(dialect);
        }

        @Override
        public String getColumnDefinitionUniquenessFragment(Column column) {
            return " unique";
        }
    }

    @Override
    public String getSelectGUIDString() {
        return "select hex(randomblob(16))";
    }

    @Override
    public ScrollMode defaultScrollMode() {
        return ScrollMode.FORWARD_ONLY;
    }
}
```

75:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\config\SQLiteMetadataBuild

erInitializer.java

```
package com.mindata.blockchain.core.sqlite.config;

import org.hibernate.boot.MetadataBuilder;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.spi.MetadataBuilderInitializer;
import org.hibernate.engine.jdbc.dialect.internal.DialectResolverSet;
import org.hibernate.engine.jdbc.dialect.spi.DialectResolver;
import org.jboss.logging.Logger;

/**
 * SQLite
 */
public class SQLiteMetadataBuilderInitializer implements MetadataBuilderInitializer {

    private final static Logger logger = Logger.getLogger(SQLiteMetadataBuilderInitializer.class);

    @Override
    public void contribute(MetadataBuilder metadataBuilder, StandardServiceRegistry
serviceRegistry) {
        DialectResolver dialectResolver = serviceRegistry.getService(DialectResolver.class);

        if (!(dialectResolver instanceof DialectResolverSet)) {
            logger.warnf("DialectResolver '%s' is not an instance of DialectResolverSet, not registering
SQLiteDialect",
                    dialectResolver);
            return;
        }

        ((DialectResolverSet) dialectResolver).addResolver(resolver);
    }

    static private final SQLiteDialect dialect = new SQLiteDialect();

    static private final DialectResolver resolver = (DialectResolver) info -> {
        if (info.getDatabaseName().equals("SQLite")) {
            return dialect;
        }

        return null;
    };
}
```

76:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlite\SqliteManager.java
```
package com.mindata.blockchain.core.sqlite;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.block.InstructionBase;
import com.mindata.blockchain.block.InstructionReverse;
import com.mindata.blockchain.core.event.DbSyncEvent;
import com.mindata.blockchain.core.manager.SyncManager;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.core.model.SyncEntity;
import com.mindata.blockchain.core.service.InstructionService;
import com.mindata.blockchain.core.sqlparser.InstructionParser;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import javax.annotation.Resource;
import java.util.ArrayList;
import java.util.List;

/**
 * sqlitesql
 *
 * @author wuweifeng wrote on 2018/3/15.
 */
@Component
public class SqliteManager {
    @Resource
    private InstructionParser instructionParser;
    @Resource
    private SyncManager syncManager;
    @Resource
    private DbBlockManager dbBlockManager;
    @Resource
    private InstructionService instructionService;
```

```
private Logger logger = LoggerFactory.getLogger(getClass());

/**
 * sqlitecheck
 */
@EventListener(DbSyncEvent.class)
public void dbSync() {
    logger.info("Sqlite");
    //
    SyncEntity syncEntity = syncManager.findLastOne();

    Block block;
    if (syncEntity == null) {
        //
        block = dbBlockManager.getFirstBlock();
        logger.info("hash" + block.getHash());
    } else {
        Block lastBlock = dbBlockManager.getLastBlock();
        //
        if (lastBlock.getHash().equals(syncEntity.getHash())) {
            logger.info("");
            return;
        }
        logger.info("hash" + lastBlock.getHash());
        String hash = syncEntity.getHash();
        block = dbBlockManager.getNextBlock(dbBlockManager.getBlockByHash(hash));
    }
    execute(block);
    ApplicationContextProvider.publishEvent(new DbSyncEvent(""));
}

/**
 * blocksql
 * block
 *
 * @param block
 *         block
 */
@Transactional(rollbackFor = Exception.class)
public void execute(Block block) {
    List<Instruction> instructions = block.getBlockBody().getInstructions();
    //InstructionParserImplInstructionBaseInstructionBase
```

```
    for (Instruction instruction : instructions) {
        instruction.setOldJson(instruction.getJson());
    }
    doSqlParse(instructions);

    //
    SyncEntity syncEntity = new SyncEntity();
    syncEntity.setHash(block.getHash());
    syncManager.save(syncEntity);
}

/**
 * block
 *
 * @param block
 *         block
 */
public void rollBack(Block block) {
    List<Instruction> instructions = block.getBlockBody().getInstructions();
    int size = instructions.size();
    //execute
    List<InstructionReverse> instructionReverses = new ArrayList<>(size);
    for (int i = size - 1; i >= 0; i--) {
        instructionReverses.add(instructionService.buildReverse(instructions.get(i)));
    }
    doSqlParse(instructionReverses);
}

private <T extends InstructionBase> void doSqlParse(List<T> instructions) {
    for (InstructionBase instruction : instructions) {
        instructionParser.parse(instruction);
    }
}

/**
 * block
 *
 * @param block block
 */
@Transactional(rollbackFor = Exception.class)
public void tryExecute(Block block) {
execute(block);
```

```
    }
}
```

77:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlparser\AbstractSqlParser.java

```
package com.mindata.blockchain.core.sqlparser;

import com.mindata.blockchain.core.model.base.BaseEntity;

/**
 * @author wuweifeng wrote on 2018/3/21.
 */
public abstract class AbstractSqlParser<T extends BaseEntity> {
    /**
     * sql
     * @param operation
     * @param id
     * @param entity entity
     */
    abstract void parse(byte operation, String id, T entity);

    /**
     *
     *
     * @return Class
     */
    abstract Class getEntityClass();
}
```

78:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlparser\InstructionParser.java

```
package com.mindata.blockchain.core.sqlparser;

import com.mindata.blockchain.block.InstructionBase;

/**
 * @author wuweifeng wrote on 2018/3/21.
 */
public interface InstructionParser {
    boolean parse(InstructionBase instructionBase);
}
```

79:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlparser\InstructionParserImpl.java

```
package com.mindata.blockchain.core.sqlparser;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.mindata.blockchain.block.Instruction;
import com.mindata.blockchain.block.InstructionBase;
import com.mindata.blockchain.common.FastJsonUtil;
import com.mindata.blockchain.core.model.base.BaseEntity;
import com.mindata.blockchain.core.model.convert.ConvertTableName;

/**
 *
 * @author wuweifeng wrote on 2018/3/21.
 */
@Service
public class InstructionParserImpl<T extends BaseEntity> implements InstructionParser {
    @Resource
    private ConvertTableName<T> convertTableName;
    @Resource
    private AbstractSqlParser<T>[] sqlParsers;

    @Override
    public boolean parse(InstructionBase instructionBase) {
        byte operation = instructionBase.getOperation();
        String table = instructionBase.getTable();
        String json = instructionBase.getOldJson();
        //MessageEntity.class
        Class<T> clazz = convertTableName.convertOf(table);
        T object = FastJsonUtil.toBean(json, clazz);
        for (AbstractSqlParser<T> sqlParser : sqlParsers) {
            if (clazz.equals(sqlParser.getEntityClass())) {
            if(instructionBase instanceof Instruction){
            object.setPublicKey(((Instruction)instructionBase).getPublicKey());
            }
                sqlParser.parse(operation, instructionBase.getInstructionId(), object);
                break;
            }
```

```
        }

        return true;
    }
}
```

80:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\core\sqlparser\MessageSqlParser.java

```
package com.mindata.blockchain.core.sqlparser;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;

import com.mindata.blockchain.block.Operation;
import com.mindata.blockchain.common.CommonUtil;
import com.mindata.blockchain.core.model.MessageEntity;
import com.mindata.blockchain.core.repository.MessageRepository;

import cn.hutool.core.bean.BeanUtil;
import cn.hutool.core.bean.copier.CopyOptions;

/**
 * Message
 * @author wuweifeng wrote on 2018/3/21.
 */
@Service
public class MessageSqlParser extends AbstractSqlParser<MessageEntity> {
    @Resource
    private MessageRepository messageRepository;

    @Override
    public void parse(byte operation, String messageId, MessageEntity entity) {
        if (Operation.ADD == operation) {
        entity.setCreateTime(CommonUtil.getNow());
            entity.setMessageId(messageId);
            messageRepository.save(entity);
        } else if (Operation.DELETE == operation) {
            messageRepository.deleteByMessageId(messageId);
        } else if (Operation.UPDATE == operation) {
            MessageEntity messageEntity = messageRepository.findByMessageId(messageId);
            BeanUtil.copyProperties(entity, messageEntity,
```

```
CopyOptions.create().setIgnoreNullValue(true).setIgnoreProperties("id", "createTime"));
            messageRepository.save(messageEntity);
        }
    }

    @Override
    public Class getEntityClass() {
        return MessageEntity.class;
    }

}
```

81:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\MdBlockchainApplication.java

```
package com.mindata.blockchain;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
@EnableAsync
public class MdBlockchainApplication {

public static void main(String[] args) {
SpringApplication.run(MdBlockchainApplication.class, args);
}
}
```

82:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\base\AbstractAioHandler.java

```
package com.mindata.blockchain.socket.base;

import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.core.ChannelContext;
import org.tio.core.GroupContext;
import org.tio.core.exception.AioDecodeException;
import org.tio.core.intf.AioHandler;
import org.tio.core.intf.Packet;
```

```
import java.nio.ByteBuffer;

/**
 * @author tanyaowu
 * 2017327 12:14:12
 */
public abstract class AbstractAioHandler implements AioHandler {
    /**
     * ByteBuffer
     * type + bodyLength
     * byte[]
     */
    @Override
    public BlockPacket decode(ByteBuffer buffer, ChannelContext channelContext) throws
AioDecodeException {
        int readableLength = buffer.limit() - buffer.position();
        if (readableLength < BlockPacket.HEADER_LENGTH) {
            return null;
        }

        //
        byte type = buffer.get();

        int bodyLength = buffer.getInt();

        if (bodyLength < 0) {
            throw new AioDecodeException("bodyLength [" + bodyLength + "] is not right, remote:" +
channelContext
.getClientNode());
        }

        int neededLength = BlockPacket.HEADER_LENGTH + bodyLength;
        int test = readableLength - neededLength;
        // (buffer)
        if (test < 0) {
            return null;
        }
        BlockPacket imPacket = new BlockPacket();
        imPacket.setType(type);
        if (bodyLength > 0) {
            byte[] dst = new byte[bodyLength];
            buffer.get(dst);
```

```
            imPacket.setBody(dst);
        }
        return imPacket;
    }

    /**
     * ByteBuffer
     * type + bodyLength
     * byte[]
     */
    @Override
    public ByteBuffer encode(Packet packet, GroupContext groupContext, ChannelContext
channelContext) {
        BlockPacket showcasePacket = (BlockPacket) packet;
        byte[] body = showcasePacket.getBody();
        int bodyLen = 0;
        if (body != null) {
            bodyLen = body.length;
        }

        //+
        int allLen = BlockPacket.HEADER_LENGTH + bodyLen;

        ByteBuffer buffer = ByteBuffer.allocate(allLen);
        buffer.order(groupContext.getByteOrder());

        //
        buffer.put(showcasePacket.getType());
        //
        buffer.putInt(bodyLen);

        //
        if (body != null) {
            buffer.put(body);
        }
        return buffer;
    }
}
```

83:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\base\AbstractBlockHandler.java

```
package com.mindata.blockchain.socket.base;
```

```
import com.mindata.blockchain.socket.body.BaseBody;
import com.mindata.blockchain.socket.common.Const;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.core.ChannelContext;
import org.tio.utils.json.Json;

/**
 * handler
 * @author tanyaowu
 * 2017327 9:56:16
 */
public abstract class AbstractBlockHandler<T extends BaseBody> implements HandlerInterface {

public AbstractBlockHandler() {
}

public abstract Class<T> bodyClass();

@Override
public Object handler(BlockPacket packet, ChannelContext channelContext) throws Exception {
String jsonStr;
T bsBody = null;
if (packet.getBody() != null) {
jsonStr = new String(packet.getBody(), Const.CHARSET);
bsBody = Json.toBean(jsonStr, bodyClass());
}

return handler(packet, bsBody, channelContext);
}

/**
 * handler
 * @param packet packet
 * @param bsBody
 * @param channelContext channelContext
 * @return
 * @throws Exception Exception
 */
public abstract Object handler(BlockPacket packet, T bsBody, ChannelContext channelContext)
throws Exception;
```

```
}
```

84:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\base\HandlerInterface.java

```
package com.mindata.blockchain.socket.base;

import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.core.ChannelContext;

/**
 *
 * @author wuweifeng
 */
public interface HandlerInterface {

/**
 * handler
 * @param packet packet
 * @param channelContext channelContext
 * @return Object
 * @throws Exception  Exception
 */
Object handler(BlockPacket packet, ChannelContext channelContext) throws Exception;

}
```

85:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\BaseBody.java

```
package com.mindata.blockchain.socket.body;

import com.mindata.blockchain.common.AppId;
import com.mindata.blockchain.common.CommonUtil;

/**
 *
 * @author tanyaowu
 * 2017327 12:12:17
 */
public class BaseBody {

/**
 *
```

```
 */
private Long time = System.currentTimeMillis();
    /**
     * id
     */
private String messageId = CommonUtil.generateUuid();
    /**
     *
     */
private String responseMsgId;
    /**
     *
     */
private String appId = AppId.value;

    public BaseBody() {
    }

    /**
 * @return the time
 */
public Long getTime() {
return time;
}

/**
 * @param time the time to set
 */
public void setTime(Long time) {
this.time = time;
}

public String getMessageId() {
return messageId;
}

public void setMessageId(String messageId) {
this.messageId = messageId;
}

    public String getResponseMsgId() {
        return responseMsgId;
```

```
    }

    public void setResponseMsgId(String responseMsgId) {
        this.responseMsgId = responseMsgId;
    }

    public String getAppId() {
        return appId;
    }

    public void setAppId(String appId) {
        this.appId = appId;
    }

    @Override
    public String toString() {
        return "BaseBody{" +
                "time=" + time +
                ", messageId='" + messageId + '\'' +
                ", responseMsgId='" + responseMsgId + '\'' +
                '}';
    }
}
```

86:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\BlockHash.java

```
package com.mindata.blockchain.socket.body;

/**
 * @author wuweifeng wrote on 2018/4/26.
 */
public class BlockHash {
    private String hash;
    private String prevHash;
    private String appId;

    public BlockHash() {
    }

    public BlockHash(String hash, String prevHash, String appId) {
        this.hash = hash;
        this.prevHash = prevHash;
```

```
        this.appId = appId;
    }

    public String getPrevHash() {
        return prevHash;
    }

    public void setPrevHash(String prevHash) {
        this.prevHash = prevHash;
    }

    public String getHash() {
        return hash;
    }

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

    public String getAppId() {
        return appId;
    }

    public void setAppId(String appId) {
        this.appId = appId;
    }
}
```

87:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\HeartBeatBody.java

```
package com.mindata.blockchain.socket.body;

/**
 * @author wuweifeng wrote on 2018/3/12.
 */
@Deprecated
public class HeartBeatBody extends BaseBody {
    /**
     * text
     */
    private String text;
```

```
    public HeartBeatBody() {
        super();
    }

    public HeartBeatBody(String text) {
        super();
        this.text = text;
    }

    public String getText() {
        return text;
    }

    public void setText(String text) {
        this.text = text;
    }

    @Override
    public String toString() {
        return "HeartBeatBody{" +
                "text='" + text + '\'' +
                '}';
    }
}
```

88:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\RpcBlockBody.java

```
package com.mindata.blockchain.socket.body;

import com.mindata.blockchain.block.Block;

/**
 * bodyblock
 * @author wuweifeng wrote on 2018/3/12.
 */
public class RpcBlockBody extends BaseBody {
    /**
     * blockJson
     */
    private Block block;

    public RpcBlockBody() {
```

```
        super();
    }

    public RpcBlockBody(Block block) {
        super();
        this.block = block;
    }

    public Block getBlock() {
        return block;
    }

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

    @Override
    public String toString() {
        return "BlockBody{" +
                "block=" + block +
                '}';
    }
}
```

89:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\RpcCheckBlockBody.java

```
package com.mindata.blockchain.socket.body;

import com.mindata.blockchain.block.Block;

/**
 * block
 * @author wuweifeng wrote on 2018/3/12.
 */
public class RpcCheckBlockBody extends RpcBlockBody {
    /**
     * 0-1number-2-3hash-4-10next block
     */
    private int code;
    /**
     * message
     */
```

```
    private String message;

    public RpcCheckBlockBody() {
    }

    public RpcCheckBlockBody(int code, String message) {
        this(code, message, null);
    }

    public RpcCheckBlockBody(int code, String message, Block block) {
        super(block);
        this.code = code;
        this.message = message;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    @Override
    public String toString() {
        return "RpcCheckBlockBody{" +
                "code=" + code +
                ", message='" + message + '\'' +
                '}';
    }
}
```

90:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\RpcNextBlockBody.java

```
package com.mindata.blockchain.socket.body;

/**
 * next block
 * @author wuweifeng wrote on 2018/4/25.
 */
public class RpcNextBlockBody extends BaseBody {
    /**
     * blockHash
     */
    private String hash;
    /**
     * hash
     */
    private String prevHash;

    public RpcNextBlockBody() {
        super();
    }

    public RpcNextBlockBody(String hash, String prevHash) {
        super();
        this.hash = hash;
        this.prevHash = prevHash;
    }

    public String getPrevHash() {
        return prevHash;
    }

    public void setPrevHash(String prevHash) {
        this.prevHash = prevHash;
    }

    public String getHash() {
        return hash;
    }

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

```
    @Override
    public String toString() {
        return "RpcNextBlockBody{" +
                "hash='" + hash + '\'' +
                ", prevHash='" + prevHash + '\'' +
                '}';
    }
}
```

91:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\RpcSimpleBlockBody.java

```
package com.mindata.blockchain.socket.body;

/**
 * @author wuweifeng wrote on 2018/4/25.
 */
public class RpcSimpleBlockBody extends BaseBody {
    /**
     * blockHash
     */
    private String hash;

    public RpcSimpleBlockBody() {
        super();
    }

    public RpcSimpleBlockBody(String hash) {
        super();
        this.hash = hash;
    }

    public String getHash() {
        return hash;
    }

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

    @Override
    public String toString() {
        return "RpcSimpleBlockBody{" +
```

```
                "hash='" + hash + '\'' +
                '}';
    }
}
```

92:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\body\VoteBody.java

```
package com.mindata.blockchain.socket.body;

import com.mindata.blockchain.socket.pbft.msg.VoteMsg;

/**
 * pbft
 * @author wuweifeng wrote on 2018/4/25.
 */
public class VoteBody extends BaseBody {
    private VoteMsg voteMsg;

    public VoteBody() {
        super();
    }

    public VoteBody(VoteMsg voteMsg) {
        super();
        this.voteMsg = voteMsg;
    }

    public VoteMsg getVoteMsg() {
        return voteMsg;
    }

    public void setVoteMsg(VoteMsg voteMsg) {
        this.voteMsg = voteMsg;
    }
}
```

93:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\BlockClientAioHandler.java

```
package com.mindata.blockchain.socket.client;

import com.mindata.blockchain.ApplicationContextProvider;
```

```
import com.mindata.blockchain.socket.base.AbstractAioHandler;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.MessageProducer;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.client.intf.ClientAioHandler;
import org.tio.core.ChannelContext;
import org.tio.core.intf.Packet;

/**
 * @author wuweifeng wrote on 2018/3/12.
 */
public class BlockClientAioHandler extends AbstractAioHandler implements ClientAioHandler {

    @Override
    public BlockPacket heartbeatPacket() {
        //Blocknext Block
        //return NextBlockPacketBuilder.build();
        return null;
    }

    /**
     * serverDisruptor
     */
    @Override
    public void handler(Packet packet, ChannelContext channelContext)  {
        BlockPacket blockPacket = (BlockPacket) packet;

        //DisruptorpublishDisruptorBlockServerAioHandler
        ApplicationContextProvider.getBean(MessageProducer.class).publish(new
BaseEvent(blockPacket, channelContext));
    }
}
```

94:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\BlockClientAioListener.java

```
package com.mindata.blockchain.socket.client;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.client.intf.ClientAioListener;
import org.tio.core.Aio;
```

```
import org.tio.core.ChannelContext;
import org.tio.core.intf.Packet;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.core.event.NodesConnectedEvent;

/**
 * clientserver</p>
 * server2minAiogroupremoveconnectgroup
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class BlockClientAioListener implements ClientAioListener {
    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    public void onAfterConnected(ChannelContext channelContext, boolean isConnected, boolean
isReconnect) throws Exception {
//        if (isConnected) {
//            logger.info("server-" + channelContext.getServerNode());
//            Aio.bindGroup(channelContext, Const.GROUP_NAME);
//        } else {
//            logger.info("server-" + channelContext.getServerNode());
//        }
        ApplicationContextProvider.publishEvent(new NodesConnectedEvent(channelContext));
    }

    @Override
    public void onBeforeClose(ChannelContext channelContext, Throwable throwable, String s,
boolean b) {
        logger.info("server-" + channelContext.getServerNode());
        Aio.unbindGroup(channelContext);
    }

    @Override
    public void onAfterDecoded(ChannelContext channelContext, Packet packet, int i) throws
Exception {

    }

    @Override
    public void onAfterReceivedBytes(ChannelContext channelContext, int i) throws Exception {
```

```
    }

    @Override
    public void onAfterSent(ChannelContext channelContext, Packet packet, boolean b) throws
Exception {

    }

    @Override
    public void onAfterHandled(ChannelContext channelContext, Packet packet, long l) throws
Exception {

    }

}
```

95:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\BlockGeneratedListener.java

```
package com.mindata.blockchain.socket.client;

import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 * group
 * @author wuweifeng wrote on 2018/3/21.
 */
@Component
public class BlockGeneratedListener {
    @Resource
    private PacketSender packetSender;
```

```
    @Order(2)
    @EventListener(AddBlockEvent.class)
    public void blockGenerated(AddBlockEvent addBlockEvent) {
        Block block = (Block) addBlockEvent.getSource();
        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.GENERATE_COMPLETE_REQUEST).setBody(new
                RpcSimpleBlockBody(block.getHash())).build();

        //
        packetSender.sendGroup(blockPacket);
    }
}
```

96:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\ClientContextConfig.java

```
package com.mindata.blockchain.socket.client;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.tio.client.ClientGroupContext;
import org.tio.client.ReconnConf;
import org.tio.client.intf.ClientAioHandler;
import org.tio.client.intf.ClientAioListener;

/**
 * ClientGroupContext
 * @author wuweifeng wrote on 2018/3/12.
 */
@Configuration
public class ClientContextConfig {

    /**
     * context
     * @return
     * ClientGroupContext
     */
    @Bean
    public ClientGroupContext clientGroupContext() {
        //handler,
        ClientAioHandler clientAioHandler = new BlockClientAioHandler();
        //null
```

```
        ClientAioListener clientAioListener = new BlockClientAioListener();
        //null
        ReconnConf reconnConf = new ReconnConf(5000L, 20);
        ClientGroupContext clientGroupContext = new ClientGroupContext(clientAioHandler,
clientAioListener,
                reconnConf);

        //clientGroupContext.setHeartbeatTimeout(Const.TIMEOUT);
        return clientGroupContext;
    }

}
```

97:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\ClientStarter.java

```
package com.mindata.blockchain.socket.client;

import com.google.common.collect.Maps;
import com.mindata.blockchain.common.AppId;
import com.mindata.blockchain.common.CommonUtil;
import com.mindata.blockchain.core.bean.Member;
import com.mindata.blockchain.core.bean.MemberData;
import com.mindata.blockchain.core.bean.Permission;
import com.mindata.blockchain.core.bean.PermissionData;
import com.mindata.blockchain.core.event.NodesConnectedEvent;
import com.mindata.blockchain.core.manager.PermissionManager;
import com.mindata.blockchain.socket.common.Const;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.NextBlockPacketBuilder;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.event.EventListener;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import org.tio.client.AioClient;
import org.tio.client.ClientGroupContext;
import org.tio.core.Aio;
import org.tio.core.ChannelContext;
import org.tio.core.Node;
import org.tio.utils.lock.SetWithLock;
```

```
import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.locks.Lock;
import java.util.stream.Collectors;

import static com.mindata.blockchain.socket.common.Const.GROUP_NAME;

/**
 * @author wuweifeng wrote on 2018/3/18.
 */
@Component
public class ClientStarter {
    @Resource
    private ClientGroupContext clientGroupContext;
    @Resource
    private PacketSender packetSender;
    @Resource
    private RestTemplate restTemplate;
    @Resource
    private PermissionManager permissionManager;
    @Value("${managerUrl}")
    private String managerUrl;
    @Value("${appId}")
    private String appId;
    @Value("${name}")
    private String name;
    @Value("${singleNode:false}")
    private Boolean singleNode;

    private Logger logger = LoggerFactory.getLogger(getClass());

    private Set<Node> nodes = new HashSet<>();

    //
    private Map<String,Integer> nodesStatus = Maps.newConcurrentMap();
    private volatile boolean isNodesReady = false; //
```

```
    /**
     *
     *
     */
    @PostConstruct
    public void initPermission() {
        fetchPermission();
    }

    /**
     * ip
     * 5
     */
    @Scheduled(fixedRate = 300000)
    public void fetchOtherServer() {
        String localIp = CommonUtil.getLocalIp();
        logger.info("IP{}",localIp);
        try {
            //
            MemberData memberData = restTemplate.getForEntity(managerUrl + "member?name=" +
name + "&appId=" + AppId
                            .value +
                            "&ip=" +
                            localIp,
                    MemberData.class).getBody();
            //
            if (memberData.getCode() == 0) {
                List<Member> memberList = memberData.getMembers();
                logger.info("" + memberList.size() + "" + memberList.toString());

                nodes.clear();
                for (Member member : memberList) {
                    Node node = new Node(member.getIp(), Const.PORT);
                    nodes.add(node);
                }
                //server
                bindServerGroup(nodes);

            } else {
                logger.error("");
                System.exit(0);
            }
```

```
        } catch (Exception e) {
            logger.error("md_blockchain_managerappId");
            System.exit(0);
        }

    }

    /**
     *
     */
    @Scheduled(fixedRate = 1000 * 60 * 60 * 24, initialDelay = 2000)
    public void fetchPermission() {
        try {
            //
            PermissionData permissionData = restTemplate.getForEntity(managerUrl +
"permission?name=" + name,
                    PermissionData.class).getBody();
            //
            if (permissionData.getCode() == 0) {
                List<Permission> permissionList = permissionData.getPermissions();
                permissionManager.savePermissionList(permissionList);
            } else {
                logger.error("");
                System.exit(0);
            }
        } catch (Exception e) {
            logger.error("md_blockchain_managerappId");
            System.exit(0);
        }

    }

    /**
     * 30Block
     */
    @Scheduled(fixedRate = 30000)
    public void heartBeat() {
    if(!isNodesReady) {
        return;
        }
        logger.info("-----------------");
        BlockPacket blockPacket = NextBlockPacketBuilder.build();
```

```
        packetSender.sendGroup(blockPacket);
    }

    public void onNodesReady() {
        logger.info("next Block");
        //group
        BlockPacket nextBlockPacket = NextBlockPacketBuilder.build();
        packetSender.sendGroup(nextBlockPacket);
    }

    /**
     * clientgroupgroup
     * serveripGroup
     */
    private void bindServerGroup(Set<Node> serverNodes) {
        //
        SetWithLock<ChannelContext> setWithLock =
Aio.getAllChannelContexts(clientGroupContext);
        Lock lock2 = setWithLock.getLock().readLock();
        lock2.lock();
        try {
            Set<ChannelContext> set = setWithLock.getObj();
            //
            Set<Node> connectedNodes =
set.stream().map(ChannelContext::getServerNode).collect(Collectors.toSet());

            //
            for (Node node : serverNodes) {
                if (!connectedNodes.contains(node)) {
                    connect(node);
                }
            }
            //
            for (ChannelContext channelContext : set) {
                Node node = channelContext.getServerNode();
                if (!serverNodes.contains(node)) {
                    Aio.remove(channelContext, "" + node.getIp());
                }

            }
        } finally {
            lock2.unlock();
```

```
        }

    }

    private void connect(Node serverNode) {
        try {
            AioClient aioClient = new AioClient(clientGroupContext);
            logger.info("" + ":" + serverNode.toString());
            aioClient.asynConnect(serverNode);
        } catch (Exception e) {
            logger.info("");
        }
    }

    @EventListener(NodesConnectedEvent.class)
    public void onConnected(NodesConnectedEvent connectedEvent){
    ChannelContext channelContext = connectedEvent.getSource();
    Node node = channelContext.getServerNode();
    if (channelContext.isClosed()) {
            logger.info("" + node.toString() + "");
            nodesStatus.put(node.getIp(), -1);
            return;
        }else{
        logger.info("" + node.toString() + "");
        nodesStatus.put(node.getIp(), 1);
        //groupgroup
        Aio.bindGroup(channelContext, GROUP_NAME);

        int csize = Aio.getAllChannelContexts(clientGroupContext).size();
        if(csize >= pbftAgreeCount()){
        synchronized (nodesStatus) {
        if(!isNodesReady){
        isNodesReady = true;
        onNodesReady();
        }
}
        }
        }
    }

    public int halfGroupSize() {
        SetWithLock<ChannelContext> setWithLock =
```

```
clientGroupContext.groups.clients(clientGroupContext, Const.GROUP_NAME);
        return setWithLock.getObj().size() / 2;
    }

    /**
     * pbftf3f+1
     *
     * @return f
     */
    public int pbftSize() {
        //Group
        int total = nodes.size();
        int pbft = (total - 1) / 3;
        if (pbft <= 0) {
            pbft = 1;
        }
        //0
        if(singleNode) {
            return 0;
        }
        return pbft;
    }

    public int pbftAgreeCount() {
        return pbftSize() * 2 + 1;
    }
}
```

98:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\client\PacketSender.java

```
package com.mindata.blockchain.socket.client;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.core.event.ClientRequestEvent;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.springframework.stereotype.Component;
import org.tio.client.ClientGroupContext;
import org.tio.core.Aio;

import javax.annotation.Resource;

import static com.mindata.blockchain.socket.common.Const.GROUP_NAME;
```

```
/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
@Component
public class PacketSender {
    @Resource
    private ClientGroupContext clientGroupContext;

    public void sendGroup(BlockPacket blockPacket) {
        //client
        ApplicationContextProvider.publishEvent(new ClientRequestEvent(blockPacket));
        //group
        Aio.sendToGroup(clientGroupContext, GROUP_NAME, blockPacket);
    }

}
```

99:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\common\Const.java

```
package com.mindata.blockchain.socket.common;

/**
 * @author wuweifeng wrote on 2018/3/9.
 */
public interface Const {
    /**
     *
     */
    String SERVER = "127.0.0.1";
    /**
     *
     */
    String GROUP_NAME = "block_group";
    /**
     *
     */
    int PORT = 6789;

    /**
     *
```

```
     */
    int TIMEOUT = 5000;

    String CHARSET = "utf-8";
}
```

100:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\base\BaseEvent.java

```
package com.mindata.blockchain.socket.distruptor.base;

import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.core.ChannelContext;

import java.io.Serializable;

/**
 * event
 *
 * @author wuweifeng wrote on 2018/4/20.
 */
public class BaseEvent implements Serializable {
    private BlockPacket blockPacket;
    private ChannelContext channelContext;

    public BaseEvent(BlockPacket blockPacket, ChannelContext channelContext) {
        this.blockPacket = blockPacket;
        this.channelContext = channelContext;
    }

    public BaseEvent(BlockPacket blockPacket) {
        this.blockPacket = blockPacket;
    }

    public BaseEvent() {
    }

    public ChannelContext getChannelContext() {
        return channelContext;
    }

    public void setChannelContext(ChannelContext channelContext) {
        this.channelContext = channelContext;
```

```
    }

    public BlockPacket getBlockPacket() {
        return blockPacket;
    }

    public void setBlockPacket(BlockPacket blockPacket) {
        this.blockPacket = blockPacket;
    }
}
```

101:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\base\BaseEventFactory.java

```
package com.mindata.blockchain.socket.distruptor.base;

import com.lmax.disruptor.EventFactory;

/**
 * @author wuweifeng wrote on 2018/4/20.
 */
public class BaseEventFactory implements EventFactory<BaseEvent> {
    @Override
    public BaseEvent newInstance() {
        return new BaseEvent();
    }

}
```

102:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\base\MessageConsumer.java

```
package com.mindata.blockchain.socket.distruptor.base;

/**
 * @author wuweifeng wrote on 2018/4/20.
 */
public interface MessageConsumer {
    void receive(BaseEvent baseEvent) throws Exception;
}
```

103:F:\git\coin\blockchain-

java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\base\MessageProducer.java

```
package com.mindata.blockchain.socket.distruptor.base;

/**
 * @author wuweifeng wrote on 2018/4/20.
 */
public interface MessageProducer {
    void publish(BaseEvent baseEvent);
}
```

104:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorClientConsumer.java

```
package com.mindata.blockchain.socket.distruptor;

import cn.hutool.core.util.StrUtil;
import com.mindata.blockchain.common.AppId;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.MessageConsumer;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.BaseBody;
import com.mindata.blockchain.socket.handler.client.FetchBlockResponseHandler;
import com.mindata.blockchain.socket.handler.client.NextBlockResponseHandler;
import com.mindata.blockchain.socket.handler.client.TotalBlockInfoResponseHandler;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketType;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;
import org.tio.utils.json.Json;

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

/**
 * server
 * @author wuweifeng wrote on 2018/4/20.
 */
@Component
public class DisruptorClientConsumer implements MessageConsumer {
    private static Map<Byte, AbstractBlockHandler<?>> handlerMap = new HashMap<>();
```

```
    private Logger logger = LoggerFactory.getLogger(getClass());

    static {
        handlerMap.put(PacketType.TOTAL_BLOCK_INFO_RESPONSE, new
TotalBlockInfoResponseHandler());
        handlerMap.put(PacketType.NEXT_BLOCK_INFO_RESPONSE, new
NextBlockResponseHandler());
        handlerMap.put(PacketType.FETCH_BLOCK_INFO_RESPONSE, new
FetchBlockResponseHandler());
    }

    @Override
    public void receive(BaseEvent baseEvent) throws Exception {
        BlockPacket blockPacket = baseEvent.getBlockPacket();
        Byte type = blockPacket.getType();
        AbstractBlockHandler<?> blockHandler = handlerMap.get(type);
        if (blockHandler == null) {
            return;
        }

        //
        BaseBody baseBody = Json.toBean(new String(blockPacket.getBody()), BaseBody.class);
        //logger.info("<" + baseBody.getAppId() + ">msg<" + baseBody.getResponseMsgId() + ">");

        String appId = baseBody.getAppId();
        if (StrUtil.equals(AppId.value, appId)) {
            //
            //return;
        }

        blockHandler.handler(blockPacket, baseEvent.getChannelContext());
    }
}
```

105:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorClientHandler.java

```
package com.mindata.blockchain.socket.distruptor;

import com.lmax.disruptor.EventHandler;
import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
```

```
/**
 * @author wuweifeng wrote on 2018/4/20.
 */
public class DisruptorClientHandler implements EventHandler<BaseEvent> {

    @Override
    public void onEvent(BaseEvent baseEvent, long sequence, boolean endOfBatch) throws
Exception {
        ApplicationContextProvider.getBean(DisruptorClientConsumer.class).receive(baseEvent);
    }
}
```

106:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorConfig.java

```
package com.mindata.blockchain.socket.distruptor;

import com.lmax.disruptor.BlockingWaitStrategy;
import com.lmax.disruptor.dsl.Disruptor;
import com.lmax.disruptor.dsl.ProducerType;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.BaseEventFactory;
import com.mindata.blockchain.socket.distruptor.base.MessageProducer;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.util.concurrent.Executors;
import java.util.concurrent.ThreadFactory;

/**
 * @author wuweifeng wrote on 2018/4/20.
 */
@Configuration
public class DisruptorConfig {

    private Disruptor<BaseEvent> disruptor() {
        ThreadFactory producerFactory = Executors.defaultThreadFactory();
        BaseEventFactory eventFactory = new BaseEventFactory();
        int bufferSize = 1024;
        Disruptor<BaseEvent> disruptor = new Disruptor<>(eventFactory, bufferSize,
producerFactory,
                ProducerType.SINGLE, new BlockingWaitStrategy());
```

```
        //type
        disruptor.handleEventsWith(new DisruptorServerHandler(), new DisruptorClientHandler());

        disruptor.start();

        return disruptor;
    }

    @Bean
    public MessageProducer messageProducer() {
        return new DisruptorProducer(disruptor());
    }
}
```

107:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorProducer.java

```
package com.mindata.blockchain.socket.distruptor;

import com.lmax.disruptor.RingBuffer;
import com.lmax.disruptor.dsl.Disruptor;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.MessageProducer;

/**
 * serverpublish
 * @author wuweifeng wrote on 2018/4/20.
 */
public class DisruptorProducer implements MessageProducer {
    private Disruptor<BaseEvent> disruptor;

    public DisruptorProducer(Disruptor<BaseEvent> disruptor) {
        this.disruptor = disruptor;
    }

    @Override
    public void publish(BaseEvent baseEvent) {
        RingBuffer<BaseEvent> ringBuffer = disruptor.getRingBuffer();
        long sequence = ringBuffer.next();
        try {
            // Get the entry in the Disruptor
            BaseEvent event = ringBuffer.get(sequence);
```

```
            // for the sequence   // Fill with data
            event.setBlockPacket(baseEvent.getBlockPacket());
            event.setChannelContext(baseEvent.getChannelContext());
        } finally {
            ringBuffer.publish(sequence);
        }
    }
}
```

108:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorServerConsumer.java

```
package com.mindata.blockchain.socket.distruptor;

import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.MessageConsumer;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.handler.server.*;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketType;
import org.springframework.stereotype.Component;

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

/**
 * client
 * @author wuweifeng wrote on 2018/4/20.
 */
@Component
public class DisruptorServerConsumer implements MessageConsumer {

    private static Map<Byte, AbstractBlockHandler<?>> handlerMap = new HashMap<>();

    static {
        handlerMap.put(PacketType.GENERATE_COMPLETE_REQUEST, new GenerateCompleteRequestHandler());
        handlerMap.put(PacketType.GENERATE_BLOCK_REQUEST, new GenerateBlockRequestHandler());
        handlerMap.put(PacketType.TOTAL_BLOCK_INFO_REQUEST, new TotalBlockInfoRequestHandler());
        handlerMap.put(PacketType.FETCH_BLOCK_INFO_REQUEST, new
```

```
FetchBlockRequestHandler());
        handlerMap.put(PacketType.HEART_BEAT, new HeartBeatHandler());
        handlerMap.put(PacketType.NEXT_BLOCK_INFO_REQUEST, new
NextBlockRequestHandler());
        handlerMap.put(PacketType.PBFT_VOTE, new PbftVoteHandler());
    }

    @Override
    public void receive(BaseEvent baseEvent) throws Exception {
        BlockPacket blockPacket = baseEvent.getBlockPacket();
        Byte type = blockPacket.getType();
        AbstractBlockHandler<?> handler = handlerMap.get(type);
        if (handler == null) {
            return;
        }
        handler.handler(blockPacket, baseEvent.getChannelContext());
    }
}
```

109:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\distruptor\DisruptorServerHandler.java

```
package com.mindata.blockchain.socket.distruptor;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.lmax.disruptor.EventHandler;
import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.handler.server.PbftVoteHandler;

/**
 * @author wuweifeng wrote on 2018/4/20.
 */
public class DisruptorServerHandler implements EventHandler<BaseEvent> {

private Logger logger = LoggerFactory.getLogger(DisruptorServerHandler.class);

    @Override
    public void onEvent(BaseEvent baseEvent, long sequence, boolean endOfBatch) throws
Exception {
```

```
    try {
    ApplicationContextProvider.getBean(DisruptorServerConsumer.class).receive(baseEvent);
} catch (Exception e) {
logger.error("Disruptor",e);
}
    }
}
```

110:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\client\FetchBlockResponseHandler.java

```
package com.mindata.blockchain.socket.handler.client;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.check.CheckerManager;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.body.RpcCheckBlockBody;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.NextBlockPacketBuilder;
import com.mindata.blockchain.socket.pbft.queue.NextBlockQueue;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;
import org.tio.utils.json.Json;

/**
 * hashblock
 *
 * @author wuweifeng wrote on 2018/3/16.
 */
public class FetchBlockResponseHandler extends AbstractBlockHandler<RpcBlockBody> {
    private Logger logger = LoggerFactory.getLogger(TotalBlockInfoResponseHandler.class);

    @Override
    public Class<RpcBlockBody> bodyClass() {
        return RpcBlockBody.class;
    }
```

```
    @Override
    public Object handler(BlockPacket packet, RpcBlockBody rpcBlockBody, ChannelContext
channelContext) {
        logger.info("<" + rpcBlockBody.getAppId() + ">Block" + Json.toJson(rpcBlockBody));

        Block block = rpcBlockBody.getBlock();
        //nullBlock
        if (block == null) {
            logger.info("Block");
        } else {
            //blocknext
        if(ApplicationContextProvider.getBean(NextBlockQueue.class).pop(block.getHash()) == null)
return null;

            CheckerManager checkerManager =
ApplicationContextProvider.getBean(CheckerManager.class);
            RpcCheckBlockBody rpcCheckBlockBody = checkerManager.check(block);
            //DB
            if (rpcCheckBlockBody.getCode() == 0) {
                ApplicationContextProvider.publishEvent(new AddBlockEvent(block));
                //
                BlockPacket blockPacket = NextBlockPacketBuilder.build();
                ApplicationContextProvider.getBean(PacketSender.class).sendGroup(blockPacket);
            }
        }

        return null;
    }
}
```

111:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\client\NextBlockResponseHandler.java

```
package com.mindata.blockchain.socket.handler.client;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.BlockHash;
import com.mindata.blockchain.socket.body.RpcNextBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.pbft.queue.NextBlockQueue;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;

/**
 * hashnext block
 *
 * @author wuweifeng wrote on 2018/3/16.
 */
public class NextBlockResponseHandler extends AbstractBlockHandler<RpcNextBlockBody> {
    private Logger logger = LoggerFactory.getLogger(TotalBlockInfoResponseHandler.class);

    @Override
    public Class<RpcNextBlockBody> bodyClass() {
        return RpcNextBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcNextBlockBody rpcBlockBody, ChannelContext
channelContext) {
        logger.info("<" + rpcBlockBody.getAppId() + ">Block hash" + rpcBlockBody.getHash());

        String hash = rpcBlockBody.getHash();
        //nullhashnext blockblock
        if (hash == null) {
            logger.info("<" + rpcBlockBody.getAppId() + ">");
        } else {
            BlockHash blockHash = new BlockHash(hash, rpcBlockBody.getPrevHash(),
rpcBlockBody.getAppId());
            //next blockhashhash2f+1
            ApplicationContextProvider.getBean(NextBlockQueue.class).push(blockHash);
        }

        return null;
    }
}
```

112:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\client\TotalBlockInfoResponseHandler.java

```
package com.mindata.blockchain.socket.handler.client;
```

```
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;
import org.tio.utils.json.Json;

/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class TotalBlockInfoResponseHandler extends AbstractBlockHandler<RpcBlockBody> {
    private Logger logger = LoggerFactory.getLogger(TotalBlockInfoResponseHandler.class);

    @Override
    public Class<RpcBlockBody> bodyClass() {
        return RpcBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcBlockBody rpcBlockBody, ChannelContext
channelContext) throws Exception {
        logger.info("<Block>", Json.toJson(rpcBlockBody));

        //TODO check
        //TODO response

        return null;
    }
}
```

113:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\FetchBlockRequestHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcBlockBody;
```

```
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.Aio;
import org.tio.core.ChannelContext;

/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class FetchBlockRequestHandler extends AbstractBlockHandler<RpcSimpleBlockBody> {
    private Logger logger = LoggerFactory.getLogger(FetchBlockRequestHandler.class);

    @Override
    public Class<RpcSimpleBlockBody> bodyClass() {
        return RpcSimpleBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcSimpleBlockBody rpcBlockBody,
ChannelContext channelContext) {
        logger.info("<" + rpcBlockBody.getAppId() + "><Block>block hash[" +
rpcBlockBody.getHash() + "]");
        Block block =
ApplicationContextProvider.getBean(DbBlockManager.class).getBlockByHash(rpcBlockBody.getH
ash());

        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.FETCH_BLOCK_INFO_RESPONSE).setBody(new
                RpcBlockBody(block)).build();
        Aio.send(channelContext, blockPacket);

        return null;
    }
}
```

114:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\GenerateBlockRequestHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.block.check.CheckerManager;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.body.RpcCheckBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.pbft.VoteType;
import com.mindata.blockchain.socket.pbft.msg.VotePreMsg;
import com.mindata.blockchain.socket.pbft.queue.MsgQueueManager;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;

/**
 * PrePre
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class GenerateBlockRequestHandler extends AbstractBlockHandler<RpcBlockBody> {
    private Logger logger = LoggerFactory.getLogger(GenerateBlockRequestHandler.class);


    @Override
    public Class<RpcBlockBody> bodyClass() {
        return RpcBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcBlockBody rpcBlockBody, ChannelContext
channelContext) {
        Block block = rpcBlockBody.getBlock();
        logger.info("<" + rpcBlockBody.getAppId() + "><Block>block[" + block + "]");

        CheckerManager checkerManager =
ApplicationContextProvider.getBean(CheckerManager.class);
        //pbftPre
        RpcCheckBlockBody rpcCheckBlockBody = checkerManager.check(block);
        logger.info(":" + rpcCheckBlockBody.toString());
        if (rpcCheckBlockBody.getCode() == 0) {
```

```
            VotePreMsg votePreMsg = new VotePreMsg();
            votePreMsg.setBlock(block);
            votePreMsg.setVoteType(VoteType.PREPREPARE);
            votePreMsg.setNumber(block.getBlockHeader().getNumber());
            votePreMsg.setAppId(rpcBlockBody.getAppId());
            votePreMsg.setHash(block.getHash());
            votePreMsg.setAgree(true);
            //PrePrepare
            ApplicationContextProvider.getBean(MsgQueueManager.class).pushMsg(votePreMsg);
        }

        return null;
    }
}
```

115:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\GenerateCompleteRequestHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.common.timer.TimerManager;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.NextBlockPacketBuilder;

/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class GenerateCompleteRequestHandler extends
AbstractBlockHandler<RpcSimpleBlockBody> {
    private Logger logger = LoggerFactory.getLogger(GenerateCompleteRequestHandler.class);
```

```
    @Override
    public Class<RpcSimpleBlockBody> bodyClass() {
        return RpcSimpleBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcSimpleBlockBody rpcBlockBody,
ChannelContext channelContext) {
        logger.info("<" + rpcBlockBody.getAppId() + "><Block>block hash[" +
rpcBlockBody.getHash() +
                "]");

        //2Block
        //Block
        TimerManager.schedule(() -> {
            Block block =
ApplicationContextProvider.getBean(DbBlockManager.class).getBlockByHash(rpcBlockBody
                    .getHash());
            //
            if (block == null) {
                logger.info("");
                //group
                BlockPacket nextBlockPacket = NextBlockPacketBuilder.build();
                ApplicationContextProvider.getBean(PacketSender.class).sendGroup(nextBlockPacket);
            }
            return null;
        },2000);

        return null;
    }
}
```

116:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\HeartBeatHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.HeartBeatBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import org.tio.core.ChannelContext;

/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
@Deprecated
public class HeartBeatHandler extends AbstractBlockHandler<HeartBeatBody> {
    private Logger logger = LoggerFactory.getLogger(HeartBeatHandler.class);

    @Override
    public Class<HeartBeatBody> bodyClass() {
        return HeartBeatBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, HeartBeatBody heartBeatBody, ChannelContext
channelContext) throws Exception {
        logger.info("<>", heartBeatBody.getText());

        return null;
    }
}
```

117:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\NextBlockRequestHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcNextBlockBody;
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.Aio;
import org.tio.core.ChannelContext;
```

```
import org.tio.utils.json.Json;

/**
 * lastBlock hashhash<p>
 * A354
 * @author wuweifeng wrote on 2018/3/16.
 */
public class NextBlockRequestHandler extends AbstractBlockHandler<RpcSimpleBlockBody> {
    private Logger logger = LoggerFactory.getLogger(TotalBlockInfoRequestHandler.class);

    @Override
    public Class<RpcSimpleBlockBody> bodyClass() {
        return RpcSimpleBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcSimpleBlockBody rpcBlockBody,
ChannelContext channelContext) {
        logger.info("<" + rpcBlockBody.getAppId() + "><Block>block hash" + Json.toJson
                (rpcBlockBody.getHash()));
        //BlocknullBlock
        String hash = rpcBlockBody.getHash();

        //next block hash2f+1
        Block nextBlock =
ApplicationContextProvider.getBean(DbBlockManager.class).getNextBlockByHash(hash);
        String nextHash = null;
        if (nextBlock != null) {
            nextHash = nextBlock.getHash();
        }
        RpcNextBlockBody respBody = new RpcNextBlockBody(nextHash, hash);
        respBody.setResponseMsgId(rpcBlockBody.getMessageId());
        BlockPacket blockPacket = new PacketBuilder<RpcNextBlockBody>().setType(PacketType
                .NEXT_BLOCK_INFO_RESPONSE).setBody(respBody).build();
        Aio.send(channelContext, blockPacket);
        logger.info("<" + rpcBlockBody.getAppId() + ">nextBlock" + respBody.toString());

        return null;
    }
}
```

java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\PbftVoteHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.VoteBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import com.mindata.blockchain.socket.pbft.queue.MsgQueueManager;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;

/**
 * pbft
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class PbftVoteHandler extends AbstractBlockHandler<VoteBody> {
    private Logger logger = LoggerFactory.getLogger(PbftVoteHandler.class);


    @Override
    public Class<VoteBody> bodyClass() {
        return VoteBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, VoteBody voteBody, ChannelContext channelContext) {
        VoteMsg voteMsg = voteBody.getVoteMsg();
        logger.info("<" + voteMsg.getAppId() + "><>[" + voteMsg + "]");

        ApplicationContextProvider.getBean(MsgQueueManager.class).pushMsg(voteMsg);
        return null;
    }
}
```

119:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\handler\server\TotalBlockInfoRequestHandler.java

```
package com.mindata.blockchain.socket.handler.server;

import com.mindata.blockchain.socket.base.AbstractBlockHandler;
import com.mindata.blockchain.socket.body.RpcBlockBody;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;
import org.tio.utils.json.Json;

/**
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
public class TotalBlockInfoRequestHandler extends AbstractBlockHandler<RpcBlockBody> {
    private Logger logger = LoggerFactory.getLogger(TotalBlockInfoRequestHandler.class);

    @Override
    public Class<RpcBlockBody> bodyClass() {
        return RpcBlockBody.class;
    }

    @Override
    public Object handler(BlockPacket packet, RpcBlockBody rpcBlockBody, ChannelContext
channelContext) throws Exception {
        logger.info("<Block>", Json.toJson(rpcBlockBody));

        //TODO check
        //TODO response

        return null;
    }
}

120:F:\git\coin\blockchain-
java\md_blockchain\src\main\java\com\mindata\blockchain\socket\packet\BlockPacket.java
package com.mindata.blockchain.socket.packet;

import com.mindata.blockchain.socket.common.Const;
import org.tio.core.intf.Packet;

import java.io.UnsupportedEncodingException;
```

```
/**
 * @author wuweifeng wrote on 2018/3/9.
 */
public class BlockPacket extends Packet {
    /**
     *   1+4
     */
    public static final int HEADER_LENGTH = 5;
    /**
     * Type
     */
    private byte type;

    private byte[] body;

    public BlockPacket() {
        super();
    }

    /**
     * @param type type
     * @param body body
     * @author tanyaowu
     */
    public BlockPacket(byte type, byte[] body) {
        super();
        this.type = type;
        this.body = body;
    }

    public BlockPacket(byte type, String body) {
        super();
        this.type = type;
        setBody(body);
    }

    /**
     * @return the body
     */
    public byte[] getBody() {
        return body;
```

```
    }

    /**
     * @return the type
     */
    public byte getType() {
        return type;
    }

    @Override
    public String logstr() {
        return "" + type;
    }

    /**
     * @param body
     *         the body to set
     */
    public void setBody(byte[] body) {
        this.body = body;
    }

    public void setBody(String body) {
        try {
            this.body = body.getBytes(Const.CHARSET);
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

    /**
     * @param type
     *         the type to set
     */
    public void setType(byte type) {
        this.type = type;
    }
}
```

121:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\packet\NextBlockPacketBuilder.java

```
package com.mindata.blockchain.socket.packet;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.core.event.ClientRequestEvent;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;

/**
 * next blockbuilder.blockhash
 * @author wuweifeng wrote on 2018/3/20.
 */
public class NextBlockPacketBuilder {
    public static BlockPacket build() {
        return build(null);
    }

    public static BlockPacket build(String responseId) {
        String hash =
ApplicationContextProvider.getBean(DbBlockManager.class).getLastBlockHash();

        RpcSimpleBlockBody rpcBlockBody = new RpcSimpleBlockBody(hash);
        rpcBlockBody.setResponseMsgId(responseId);
        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.NEXT_BLOCK_INFO_REQUEST).setBody
                (rpcBlockBody).build();
        //client
        ApplicationContextProvider.publishEvent(new ClientRequestEvent(blockPacket));
        return blockPacket;
    }

}
```

122:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\packet\PacketBuilder.java

```
package com.mindata.blockchain.socket.packet;

import com.mindata.blockchain.socket.body.BaseBody;
import org.tio.utils.json.Json;

/**
 * @author wuweifeng wrote on 2018/3/12.
 */
```

```
public class PacketBuilder<T extends BaseBody> {
    /**
     * Type
     */
    private byte type;

    private T body;

    public PacketBuilder<T> setType(byte type) {
        this.type = type;
        return this;
    }

    public PacketBuilder<T> setBody(T body) {
        this.body = body;
        return this;
    }

    public BlockPacket build() {
        return new BlockPacket(type, Json.toJson(body));
    }
}
```

123:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\packet\PacketType.java

```
package com.mindata.blockchain.socket.packet;

/**
 * packetType00
 * @author wuweifeng wrote on 2018/3/9.
 */
public interface PacketType {
    /**
     *
     */
    byte HEART_BEAT = 0;
    /**
     *
     */
    byte GENERATE_COMPLETE_REQUEST = 1;
    /**
     *
```

```
     */
    byte GENERATE_COMPLETE_RESPONSE = -1;
    /**
     * block
     */
    byte GENERATE_BLOCK_REQUEST = 2;
    /**
     *
     */
    byte GENERATE_BLOCK_RESPONSE = -2;
    /**
     * block
     */
    byte TOTAL_BLOCK_INFO_REQUEST = 3;
    /**
     *
     */
    byte TOTAL_BLOCK_INFO_RESPONSE = -3;
    /**
     * block
     */
    byte FETCH_BLOCK_INFO_REQUEST = 4;
    /**
     *
     */
    byte FETCH_BLOCK_INFO_RESPONSE = -4;
    /**
     *
     */
    byte NEXT_BLOCK_INFO_REQUEST = 5;
    /**
     *
     */
    byte NEXT_BLOCK_INFO_RESPONSE = -5;
    /**
     * pbft
     */
    byte PBFT_VOTE = 10;
}
```

124:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\event\MsgCommitEvent.jav

a
```
package com.mindata.blockchain.socket.pbft.event;

import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.springframework.context.ApplicationEvent;

/**
 * Commit
 * @author wuweifeng wrote on 2018/4/25.
 */
public class MsgCommitEvent extends ApplicationEvent {
    public MsgCommitEvent(VoteMsg source) {
        super(source);
    }
}
```

125:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\event\MsgPrepareEvent.java
```
package com.mindata.blockchain.socket.pbft.event;

import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.springframework.context.ApplicationEvent;

/**
 * Prepare
 * @author wuweifeng wrote on 2018/4/25.
 */
public class MsgPrepareEvent extends ApplicationEvent {
    public MsgPrepareEvent(VoteMsg source) {
        super(source);
    }
}
```

126:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\listener\CommitEventListener.java
```
package com.mindata.blockchain.socket.pbft.listener;

import com.mindata.blockchain.socket.body.VoteBody;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
```

```
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import com.mindata.blockchain.socket.pbft.event.MsgCommitEvent;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 * blockcommit
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public class CommitEventListener {
    @Resource
    private PacketSender packetSender;

    /**
     * blockcommit
     *
     * @param msgCommitEvent
     *         msgCommitEvent
     */
    @EventListener
    public void msgIsCommit(MsgCommitEvent msgCommitEvent) {
        VoteMsg voteMsg = (VoteMsg) msgCommitEvent.getSource();

        //Block Prepare
        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.PBFT_VOTE).setBody(new
                VoteBody(voteMsg)).build();

        //commit
        packetSender.sendGroup(blockPacket);
    }
}
```

127:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\listener\PrepareEventListener.java

```
package com.mindata.blockchain.socket.pbft.listener;
```

```
import javax.annotation.Resource;

import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;

import com.mindata.blockchain.socket.body.VoteBody;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import com.mindata.blockchain.socket.pbft.event.MsgPrepareEvent;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;

/**
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public class PrepareEventListener {
    @Resource
    private PacketSender packetSender;

    /**
     * blockPrepare
     *
     * @param msgPrepareEvent
     *        msgIsPrepareEvent
     */
    @EventListener
    public void msgIsPrepare(MsgPrepareEvent msgPrepareEvent) {
        VoteMsg voteMsg = (VoteMsg) msgPrepareEvent.getSource();

        //Block Prepare
        BlockPacket blockPacket = new
PacketBuilder<>().setType(PacketType.PBFT_VOTE).setBody(new
                VoteBody(voteMsg)).build();

        //Prepare
        packetSender.sendGroup(blockPacket);
    }
}
```

128:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\msg\VoteMsg.java

```
package com.mindata.blockchain.socket.pbft.msg;

/**
 * pbftpreparecommit
 * @author wuweifeng wrote on 2018/4/23.
 */
public class VoteMsg {
    /**
     * Preparecommit
     */
    private byte voteType;
    /**
     * hash
     */
    private String hash;
    /**
     * number
     */
    private int number;
    /**
     *
     */
    private String appId;
    /**
     *
     */
    private boolean agree;

    @Override
    public String toString() {
        return "VoteMsg{" +
                "voteType=" + voteType +
                ", hash='" + hash + '\'' +
                ", number=" + number +
                ", appId='" + appId + '\'' +
                ", agree=" + agree +
                '}';
    }

    public byte getVoteType() {
```

```
        return voteType;
    }

    public void setVoteType(byte voteType) {
        this.voteType = voteType;
    }

    public String getHash() {
        return hash;
    }

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

    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public String getAppId() {
        return appId;
    }

    public void setAppId(String appId) {
        this.appId = appId;
    }

    public boolean isAgree() {
        return agree;
    }

    public void setAgree(boolean agree) {
        this.agree = agree;
    }
}
```

129:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\msg\VotePreMsg.java

```
package com.mindata.blockchain.socket.pbft.msg;

import com.mindata.blockchain.block.Block;

/**
 * @author wuweifeng wrote on 2018/4/25.
 */
public class VotePreMsg extends VoteMsg {
    private Block block;

    public Block getBlock() {
        return block;
    }

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

130:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\AbstractVoteMsgQueue.java

```
package com.mindata.blockchain.socket.pbft.queue;

import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ConcurrentHashMap;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.mindata.blockchain.common.timer.TimerManager;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;

import cn.hutool.core.collection.CollectionUtil;

/**
 * @author wuweifeng wrote on 2018/4/26.
 */
public abstract class AbstractVoteMsgQueue extends BaseMsgQueue {
    /**
     * hash
```

```
     */
    protected ConcurrentHashMap<String, List<VoteMsg>> voteMsgConcurrentHashMap = new
ConcurrentHashMap<>();
    /**
     * hashcommitcommit
     */
    protected ConcurrentHashMap<String, Boolean> voteStateConcurrentHashMap = new
ConcurrentHashMap<>();

    private Logger logger = LoggerFactory.getLogger(getClass());

    abstract void deal(VoteMsg voteMsg, List<VoteMsg> voteMsgs);

    @Override
    protected void push(VoteMsg voteMsg) {
        String hash = voteMsg.getHash();
        List<VoteMsg> voteMsgs = voteMsgConcurrentHashMap.get(hash);
        if (CollectionUtil.isEmpty(voteMsgs)) {
            voteMsgs = new ArrayList<>();
            voteMsgConcurrentHashMap.put(hash, voteMsgs);
        } else {
            //voteMsg
            for (VoteMsg temp : voteMsgs) {
                if (temp.getAppId().equals(voteMsg.getAppId())) {
                    return;
                }
            }
        }

        //
        voteMsgs.add(voteMsg);
        //hash
        if (voteStateConcurrentHashMap.get(hash) != null) {
            return;
        }

        deal(voteMsg, voteMsgs);
    }

    /**
     * pushBlock <p>
     * 5PrepareCommitnumber>=5
```

```
     *
     * @param hash
     *         hash
     * @return
     */
    public boolean hasOtherConfirm(String hash, int number) {
        //
        for (String key : voteMsgConcurrentHashMap.keySet()) {
            //hash
            if (hash.equals(key)) {
                continue;
            }
            //number
            if (voteMsgConcurrentHashMap.get(key).get(0).getNumber() < number) {
                continue;
            }
            //>=numberBlocktruehash
            if (voteStateConcurrentHashMap.get(key) != null &&
voteStateConcurrentHashMap.get(key)) {
                return true;
            }
        }
        return false;
    }

    /**
     * blockhash
     */
    protected void clearOldBlockHash(int number) {
    TimerManager.schedule(() -> {
            for (String key : voteMsgConcurrentHashMap.keySet()) {
                if (voteMsgConcurrentHashMap.get(key).get(0).getNumber() <= number) {
                    voteMsgConcurrentHashMap.remove(key);
                    voteStateConcurrentHashMap.remove(key);
                }
            }
            return null;
        },2000);
    }
}

131:F:\git\coin\blockchain-
```

java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\BaseMsgQueue.java

```
package com.mindata.blockchain.socket.pbft.queue;

import com.mindata.blockchain.socket.client.ClientStarter;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;

/**
 *
 *
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public abstract class BaseMsgQueue {
    @Resource
    private ClientStarter clientStarter;

    public int pbftSize() {
        return clientStarter.pbftSize();
    }

    public int pbftAgreeSize() {
        return clientStarter.pbftAgreeCount();
    }
    /**
     *
     *
     * @param voteMsg
     *         voteMsg
     */
    protected abstract void push(VoteMsg voteMsg);
}
```

132:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\CommitMsgQueue.java

```
package com.mindata.blockchain.socket.pbft.queue;

import java.util.List;
```

```
import javax.annotation.Resource;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;

/**
 * Confirm
 * 2f+1commitcommitted
 *
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public class CommitMsgQueue extends AbstractVoteMsgQueue {
    @Resource
    private PreMsgQueue preMsgQueue;

    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    protected void deal(VoteMsg voteMsg, List<VoteMsg> voteMsgs) {
        String hash = voteMsg.getHash();

        //agreeBlock
        long count = voteMsgs.stream().filter(VoteMsg::isAgree).count();
        logger.info("committrue:"+ count);
        if (count >= pbftAgreeSize()) {
            Block block = preMsgQueue.findByHash(hash);
            if (block == null) {
                return;
            }
            //
            voteStateConcurrentHashMap.put(hash, true);
            ApplicationContextProvider.publishEvent(new AddBlockEvent(block));
```

```
        }
    }

    /**
     * clearmapnumber
     */
    @Order(3)
    @EventListener(AddBlockEvent.class)
    public void blockGenerated(AddBlockEvent addBlockEvent) {
        Block block = (Block) addBlockEvent.getSource();
        clearOldBlockHash(block.getBlockHeader().getNumber());
    }

}
```

133:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\MsgQueueManager.java

```
package com.mindata.blockchain.socket.pbft.queue;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.pbft.VoteType;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.springframework.stereotype.Component;

/**
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public class MsgQueueManager {

    public void pushMsg(VoteMsg voteMsg) {
    BaseMsgQueue baseMsgQueue = null;
        switch (voteMsg.getVoteType()) {
            case VoteType
                    .PREPREPARE:
                baseMsgQueue = ApplicationContextProvider.getBean(PreMsgQueue.class);
                break;
            case VoteType.PREPARE:
                baseMsgQueue = ApplicationContextProvider.getBean(PrepareMsgQueue.class);
                break;
            case VoteType.COMMIT:
```

```
                baseMsgQueue = ApplicationContextProvider.getBean(CommitMsgQueue.class);
                break;
            default:
                break;
        }
        if(baseMsgQueue != null) {
        baseMsgQueue.push(voteMsg);
        }
    }
}
```

134:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\NextBlockQueue.java

```
package com.mindata.blockchain.socket.pbft.queue;

import cn.hutool.core.util.StrUtil;

import com.google.common.collect.Lists;
import com.mindata.blockchain.core.manager.DbBlockManager;
import com.mindata.blockchain.socket.body.BlockHash;
import com.mindata.blockchain.socket.body.RpcSimpleBlockBody;
import com.mindata.blockchain.socket.client.ClientStarter;
import com.mindata.blockchain.socket.client.PacketSender;
import com.mindata.blockchain.socket.packet.BlockPacket;
import com.mindata.blockchain.socket.packet.PacketBuilder;
import com.mindata.blockchain.socket.packet.PacketType;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;
import java.util.stream.Collectors;

/**
 * next block
 *
 * @author wuweifeng wrote on 2018/3/26.
 */
@Component
```

```
public class NextBlockQueue {
    @Resource
    private DbBlockManager dbBlockManager;
    @Resource
    private ClientStarter clientStarter;
    @Resource
    private PacketSender packetSender;

    private Logger logger = LoggerFactory.getLogger(getClass());

    /**
     * prevHash->hashhashhash
     */
    private ConcurrentHashMap<String, List<BlockHash>> requestMap = new
ConcurrentHashMap<>();

    /**
     * hash,
     */
    private List<String> wantHashs = Lists.newCopyOnWriteArrayList();

    public String pop(String hash) {
    if(wantHashs.remove(hash)) {
    return hash;
    }
    return null;
    }

    public List<BlockHash> get(String key) {
        return requestMap.get(key);
    }

    public void put(String key, List<BlockHash> responses) {
        requestMap.put(key, responses);
    }

    private void add(String key, BlockHash blockHash) {
        List<BlockHash> baseResponses = get(key);

        if (baseResponses == null) {
            baseResponses = new ArrayList<>();
        }
```

```
        //
        for (BlockHash oldResponse : baseResponses) {
            if (StrUtil.equals(oldResponse.getAppId(), blockHash.getAppId())) {
                return;
            }
        }
        baseResponses.add(blockHash);
        put(key, baseResponses);
    }

    /**
     * keyBlockHashhash
     *
     * @param key
     *         key
     * @return hash
     */
    public List<BlockHash> findMaxHash(String key) {
        List<BlockHash> blockHashes = get(key);
        //hash
        Map<String, Integer> map = new HashMap<>();
        for (BlockHash blockHash : blockHashes) {
            String hash = blockHash.getHash();
            map.merge(hash, 1, (a, b) -> a + b);
        }
        //valuekeyhash
        String hash = getMaxKey(map);
        return blockHashes.stream().filter(blockHash ->
hash.equals(blockHash.getHash())).collect(Collectors.toList());
    }

    private String getMaxKey(Map<String, Integer> hashMap) {
        int value, flagValue = 0;
        String key, flagKey = null;
        Set<Map.Entry<String, Integer>> entrySet = hashMap.entrySet();
        for (Map.Entry<String, Integer> entry : entrySet) {
            key = entry.getKey();
            value = entry.getValue();

            if (flagValue < value) {
                //flagKey flagValue
                flagKey = key;
```

```
            flagValue = value;
        }
    }
    return flagKey;
}


public void remove(String key) {
    requestMap.remove(key);
}

/**
 * nextBlock
 *
 * @param blockHash
 *         blockHash
 */
public void push(BlockHash blockHash) {
    String wantHash = blockHash.getHash();
    String prevHash = blockHash.getPrevHash();
    //
    if (prevHash == null) {
        prevHash = "first_block_hash";
    }
    //hash
    if (dbBlockManager.getBlockByHash(wantHash) != null) {
        remove(prevHash);
        return;
    }
    add(prevHash, blockHash);

    int agreeCount = clientStarter.pbftAgreeCount();

    //hash
    int maxCount = findMaxHash(prevHash).size();

    //
    if (maxCount >= agreeCount - 1) {
        logger.info("<" + maxCount + ">next block hash" + wantHash);
        wantHashs.add(wantHash);
        //hashBlock
        BlockPacket blockPacket = new
```

```
PacketBuilder<RpcSimpleBlockBody>().setType(PacketType
                .FETCH_BLOCK_INFO_REQUEST).setBody(new
RpcSimpleBlockBody(wantHash)).build();
            packetSender.sendGroup(blockPacket);
            //removeagreeCountblock
            remove(prevHash);
        }

    }

}
```

135:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\PreMsgQueue.java

```
package com.mindata.blockchain.socket.pbft.queue;

import cn.hutool.core.bean.BeanUtil;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.common.AppId;
import com.mindata.blockchain.common.timer.TimerManager;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.core.sqlite.SqliteManager;
import com.mindata.blockchain.socket.pbft.VoteType;
import com.mindata.blockchain.socket.pbft.event.MsgPrepareEvent;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import com.mindata.blockchain.socket.pbft.msg.VotePreMsg;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.concurrent.ConcurrentHashMap;

/**
 * preprepareBlock
 *
 * @author wuweifeng wrote on 2018/4/23.
 */
@Component
```

```
public class PreMsgQueue extends BaseMsgQueue {
    @Resource
    private SqliteManager sqliteManager;
    @Resource
    private PrepareMsgQueue prepareMsgQueue;
    @Resource
    private ApplicationEventPublisher eventPublisher;

    private ConcurrentHashMap<String, VotePreMsg> blockConcurrentHashMap = new
ConcurrentHashMap<>();

    private Logger logger = LoggerFactory.getLogger(getClass());

    @Override
    protected void push(VoteMsg voteMsg) {
        //votePreMsg
        VotePreMsg votePreMsg = (VotePreMsg) voteMsg;
        String hash = votePreMsg.getHash();
        //
        if (blockConcurrentHashMap.get(hash) != null) {
            return;
        }
        //pushnumberblock
        //VotenumberprepreOKnumbervote
        if (prepareMsgQueue.otherConfirm(hash, voteMsg.getNumber())) {
            logger.info("Preparehash" + hash);
            return;
        }
        //
        try {
            sqliteManager.tryExecute(votePreMsg.getBlock());
        } catch (Exception e) {
            //
            logger.info("sql");
            return;
        }

        //Pre
        blockConcurrentHashMap.put(hash, votePreMsg);

        //Prepare
        VoteMsg prepareMsg = new VoteMsg();
```

```
        BeanUtil.copyProperties(voteMsg, prepareMsg);
        prepareMsg.setVoteType(VoteType.PREPARE);
        prepareMsg.setAppId(AppId.value);
        eventPublisher.publishEvent(new MsgPrepareEvent(prepareMsg));
    }

    /**
     * hashBlock
     *
     * @param hash
     *         hash
     * @return Block
     */
    public Block findByHash(String hash) {
        VotePreMsg votePreMsg = blockConcurrentHashMap.get(hash);
        if (votePreMsg != null) {
            return votePreMsg.getBlock();
        }
        return null;
    }

    /**
     * clearmapnumber
     */
    @Order(3)
    @EventListener(AddBlockEvent.class)
    public void blockGenerated(AddBlockEvent addBlockEvent) {
        Block block = (Block) addBlockEvent.getSource();
        int number = block.getBlockHeader().getNumber();
        TimerManager.schedule(() -> {
            for (String key : blockConcurrentHashMap.keySet()) {
                if (blockConcurrentHashMap.get(key).getNumber() <= number) {
                    blockConcurrentHashMap.remove(key);
                }
            }
            return null;
        }, 2000);
    }
}
```

136:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\queue\PrepareMsgQueue.j

ava

```
package com.mindata.blockchain.socket.pbft.queue;

import cn.hutool.core.bean.BeanUtil;
import com.mindata.blockchain.block.Block;
import com.mindata.blockchain.common.AppId;
import com.mindata.blockchain.core.event.AddBlockEvent;
import com.mindata.blockchain.socket.pbft.VoteType;
import com.mindata.blockchain.socket.pbft.event.MsgCommitEvent;
import com.mindata.blockchain.socket.pbft.msg.VoteMsg;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.event.EventListener;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

import javax.annotation.Resource;
import java.util.List;

/**
 * Prepare
 *
 * @author wuweifeng wrote on 2018/4/25.
 */
@Component
public class PrepareMsgQueue extends AbstractVoteMsgQueue {
    @Resource
    private CommitMsgQueue commitMsgQueue;
    @Resource
    private ApplicationEventPublisher eventPublisher;
    private Logger logger = LoggerFactory.getLogger(getClass());

    /**
     * BlockPrepare
     *
     * @param voteMsg
     *        voteMsg
     */
    @Override
    protected void deal(VoteMsg voteMsg, List<VoteMsg> voteMsgs) {
        String hash = voteMsg.getHash();
```

```
    VoteMsg commitMsg = new VoteMsg();
    BeanUtil.copyProperties(voteMsg, commitMsg);
    commitMsg.setVoteType(VoteType.COMMIT);
    commitMsg.setAppId(AppId.value);
    //commit
    //vote
    if (commitMsgQueue.hasOtherConfirm(hash, voteMsg.getNumber())) {
         agree(commitMsg, false);
    } else {
        //agree2f + 1commit
        long agreeCount = voteMsgs.stream().filter(VoteMsg::isAgree).count();
        long unAgreeCount = voteMsgs.size() - agreeCount;

        //commitor
        if (agreeCount >= pbftAgreeSize()) {
            agree(commitMsg, true);
        } else if (unAgreeCount >= pbftSize() + 1) {
            agree(commitMsg, false);
        }
    }

}

private void agree(VoteMsg commitMsg, boolean flag) {
    logger.info("Preparecommit" + flag);
    //commit
    commitMsg.setAgree(flag);
    voteStateConcurrentHashMap.put(commitMsg.getHash(), flag);
    eventPublisher.publishEvent(new MsgCommitEvent(commitMsg));
}

/**
 * BlocktrueBlock
 *
 * @param hash
 *         hash
 * @return
 */
public boolean otherConfirm(String hash, int number) {
    if (commitMsgQueue.hasOtherConfirm(hash, number)) {
        return true;
    }
```

```
        return hasOtherConfirm(hash, number);
    }

    /**
     * clearmapnumber
     *
     * @param addBlockEvent  addBlockEvent
     */
    @Order(3)
    @EventListener(AddBlockEvent.class)
    public void blockGenerated(AddBlockEvent addBlockEvent) {
        Block block = (Block) addBlockEvent.getSource();
        clearOldBlockHash(block.getBlockHeader().getNumber());
    }
}
```

137:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\pbft\VoteType.java

```
package com.mindata.blockchain.socket.pbft;

/**
 * pbft
 *
 *
 *
 * block
 * blockblockprepare<h, d, s>hblockdblocks
 * preparef+1preparepreparedcommit<h, d, s>
 * 2f+1commitcommitted
 * hblock
 * @author wuweifeng wrote on 2018/4/23.
 */
public class VoteType {
    /**
     * Block
     */
    public static final byte PREPREPARE = 1;
    /**
     * block
     */
    public static final byte PREPARE = 2;
    /**
```

```
     * 2f+1commitcommitted
     */
    public static final byte COMMIT = 3;
}
```

138:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\server\BlockServerAioHandler.java

```
package com.mindata.blockchain.socket.server;

import com.mindata.blockchain.ApplicationContextProvider;
import com.mindata.blockchain.socket.distruptor.base.BaseEvent;
import com.mindata.blockchain.socket.distruptor.base.MessageProducer;
import com.mindata.blockchain.socket.base.AbstractAioHandler;
import com.mindata.blockchain.socket.packet.BlockPacket;
import org.tio.core.ChannelContext;
import org.tio.core.intf.Packet;
import org.tio.server.intf.ServerAioHandler;

/**
 * serverclient
 * @author wuweifeng wrote on 2018/3/12.
 */
public class BlockServerAioHandler extends AbstractAioHandler implements ServerAioHandler {



    /**
     * server
     */
    @Override
    public void handler(Packet packet, ChannelContext channelContext) {
        BlockPacket blockPacket = (BlockPacket) packet;

        //DisruptorpublishDisruptorBlockClientAioHandler
        ApplicationContextProvider.getBean(MessageProducer.class).publish(new BaseEvent(blockPacket, channelContext));
    }
}
```

139:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\server\BlockServerAioListener.java

```
package com.mindata.blockchain.socket.server;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.tio.core.ChannelContext;
import org.tio.core.intf.Packet;
import org.tio.server.intf.ServerAioListener;
import org.tio.utils.json.Json;

/**
 * @author wuweifeng wrote on 2018/3/12.
 *
 * 2017326 8:22:31
 */
public class BlockServerAioListener implements ServerAioListener {
private static Logger log = LoggerFactory.getLogger(BlockServerAioListener.class);

@Override
public void onAfterConnected(ChannelContext channelContext, boolean isConnected, boolean
isReconnect) {
log.info("onAfterConnected channelContext:{}, isConnected:{}, isReconnect:{}", channelContext,
isConnected, isReconnect);

//channelContext
//channelContext.setAttribute(new ShowcaseSessionContext());
}

@Override
public void onAfterDecoded(ChannelContext channelContext, Packet packet, int i) throws
Exception {

}

@Override
public void onAfterReceivedBytes(ChannelContext channelContext, int i) throws Exception {
log.info("onAfterReceived channelContext:{}, packet:{}, packetSize:{}");
}


@Override
public void onAfterSent(ChannelContext channelContext, Packet packet, boolean isSentSuccess)
{
```

```
log.info("onAfterSent channelContext:{}, packet:{}, isSentSuccess:{}", channelContext,
Json.toJson(packet), isSentSuccess);
}

@Override
public void onAfterHandled(ChannelContext channelContext, Packet packet, long l) throws
Exception {

}

@Override
public void onBeforeClose(ChannelContext channelContext, Throwable throwable, String remark,
boolean isRemove) {
}
}
```

140:F:\git\coin\blockchain-java\md_blockchain\src\main\java\com\mindata\blockchain\socket\server\BlockServerStarter.java

```
package com.mindata.blockchain.socket.server;

import com.mindata.blockchain.socket.common.Const;
import org.springframework.stereotype.Component;
import org.tio.server.AioServer;
import org.tio.server.ServerGroupContext;
import org.tio.server.intf.ServerAioHandler;
import org.tio.server.intf.ServerAioListener;

import javax.annotation.PostConstruct;
import java.io.IOException;

/**
 * server
 *
 * @author wuweifeng wrote on 2018/3/12.
 */
@Component
public class BlockServerStarter {

    @PostConstruct
    public void serverStart() throws IOException {
        ServerAioHandler serverAioHandler = new BlockServerAioHandler();
        ServerAioListener serverAioListener = new BlockServerAioListener();
```

```
        ServerGroupContext serverGroupContext = new ServerGroupContext(serverAioHandler,
serverAioListener);
        AioServer aioServer = new AioServer(serverGroupContext);
        //
        aioServer.start(null, Const.PORT);
    }
}
```