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

0:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\cache\src\main\java\io\nuls\cache\CacheMap.java
```
 */
package io.nuls.cache;

import io.nuls.cache.listener.intf.NulsCacheListener;
import io.nuls.cache.manager.EhCacheManager;
import io.nuls.cache.model.CacheMapParams;
import org.ehcache.Cache;
import org.ehcache.spi.copy.Copier;

import java.io.Serializable;
import java.util.*;

/**
 *
 * The key values used by the cache provide some basic methods for storing the structure.
 *
 * @author Niels
 */
public class CacheMap<K, V> {

    private EhCacheManager cacheManager = EhCacheManager.getInstance();

    private final String cacheName;

    public CacheMap(String cacheName, int heapMb, Class keyType, Class<? extends
Serializable> valueType, Copier<V> valueCopier) {
        this(cacheName, heapMb, keyType, valueType, 0, 0, valueCopier);
    }

    public CacheMap(String cacheName, int heapMb, Class keyType, Class<? extends
Serializable> valueType) {
        this(cacheName, heapMb, keyType, valueType, 0, 0, null);
    }

    public CacheMap(String cacheName, int heapMb, Class keyType, Class<? extends
Serializable> valueType, int timeToLiveSeconds, int timeToIdleSeconds, Copier<V> valueCopier)
{
        this(cacheName, heapMb, keyType, valueType, timeToLiveSeconds, timeToIdleSeconds,
null, valueCopier);
```

```
    }

    public CacheMap(String cacheName, int heapMb, Class keyType, Class<? extends
Serializable> valueType, int timeToLiveSeconds, int timeToIdleSeconds) {
        this(cacheName, heapMb, keyType, valueType, timeToLiveSeconds, timeToIdleSeconds,
null, null);
    }

    public CacheMap(String cacheName, int heapMb, Class keyType, Class<? extends
Serializable> valueType, int timeToLiveSeconds, int timeToIdleSeconds, NulsCacheListener
listener, Copier<V> valueCopier) {
        this(cacheName, new CacheMapParams(heapMb, keyType, valueType, timeToLiveSeconds,
timeToIdleSeconds, listener, valueCopier));
    }

    public CacheMap(String cacheName, CacheMapParams params) {
        this.cacheName = cacheName;
        this.cacheManager.createCache(cacheName, params);
    }

    @Deprecated
    public int size() {
        return this.keySet().size();
    }

    public boolean isEmpty() {
        return this.keySet().isEmpty();
    }

    public boolean containsKey(K key) {
        Cache cache = this.cacheManager.getCache(cacheName);
        if (cache == null) {
            return false;
        }
        return cache.containsKey(key);
    }

    public boolean containsValue(V value) {
        List<V> vlist = this.values();
        return vlist.contains(value);
    }
```

```
public V get(K key) {
    if (null == cacheManager.getCache(cacheName) || null == key) {
        return null;
    }
    return ((V) cacheManager.getCache(cacheName).get(key));
}


public void put(K key, V value) {
    Object valueObj = value;
    if (null == cacheManager.getCache(cacheName)) {
        throw new RuntimeException("Cache not exist!");
    }
    cacheManager.getCache(cacheName).put(key, valueObj);
}

public void remove(K key) {
    if (null == cacheManager.getCache(cacheName)) {
        return;
    }
    cacheManager.getCache(cacheName).remove(key);
}

public void clear() {
    if (null == cacheManager.getCache(cacheName)) {
        return;
    }
    cacheManager.getCache(cacheName).clear();
}

public Set<K> keySet() {
    Cache cache = this.cacheManager.getCache(cacheName);
    if (null == cache) {
        return new HashSet<>();
    }
    Iterator it = cache.iterator();
    Set<K> set = new HashSet<>();
    while (it.hasNext()) {
        Cache.Entry<K, V> entry = (Cache.Entry<K, V>) it.next();
        set.add((K) entry.getKey());
    }
```

```
        return set;
    }

    public List<V> values() {
        if (cacheManager == null || null == cacheManager.getCache(cacheName)) {
            return new ArrayList<>();
        }
        Iterator it = cacheManager.getCache(cacheName).iterator();
        List<V> list = new ArrayList<>();
        while (it.hasNext()) {
            Cache.Entry<K, V> entry = (Cache.Entry<K, V>) it.next();
            V t = entry.getValue();
            list.add(t);
        }
        return list;
    }

    public void destroy() {
        this.cacheManager.removeCache(cacheName);
    }


}

1:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\LimitHashMap.java
 */

package io.nuls.cache;

import java.util.Collection;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.LinkedBlockingDeque;

/**
 * @author: Niels Wang
 * @date: 2018/7/5
 */
public class LimitHashMap<K, V> {
```

```
private final int maxSize;
private Map<K, V> map = new ConcurrentHashMap<>();

private LinkedBlockingDeque<K> queue = new LinkedBlockingDeque<>();

public LimitHashMap(int maxSize) {
    this.maxSize = maxSize;
}

public boolean put(K k, V v) {
    V other = map.put(k, v);
    if(other != null) {
        return false;
    }
    queue.offer(k);
    if (maxSize > queue.size()) {
        return true;
    }
    int count = maxSize / 2;
    for (int i = 0; i < count; i++) {
        K key = queue.poll();
        if (null == key) {
            return true;
        }
        map.remove(key);
        if (count % 100 == 0 && count > queue.size()) {
            break;
        }
    }
    return true;
}

public void remove(K k) {
    map.remove(k);
    queue.remove(k);
}

public V get(K k) {
    return map.get(k);
}
```

```
    public void clear() {
        queue.clear();
        map.clear();
    }

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

    public boolean containsKey(K key) {
        return map.containsKey(key);
    }

    public Collection<V> values() {
        return map.values();
    }

    public Map<K, V> getMap() {
        return map;
    }

    public LinkedBlockingDeque<K> getQueue() {
        return queue;
    }
}

2:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\listener\intf\NulsCacheListener.java
 */

package io.nuls.cache.listener.intf;


import io.nuls.cache.model.CacheListenerItem;

/**
 * @author Niels
 */
public interface NulsCacheListener<K, V> {

    void onCreate(CacheListenerItem<K, V> item);
```

```
    void onEvict(CacheListenerItem<K, V> item);

    void onRemove(CacheListenerItem<K, V> item);

    void onUpdate(CacheListenerItem<K, V> item);

    void onExpire(CacheListenerItem<K, V> item);

}

3:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\manager\EhCacheManager.java
 */
package io.nuls.cache.manager;

import io.nuls.cache.model.CacheMapParams;
import io.nuls.cache.utils.EhcacheListener;
import io.nuls.core.tools.param.AssertUtil;
import org.ehcache.Cache;
import org.ehcache.CacheManager;
import org.ehcache.config.builders.*;
import org.ehcache.config.units.MemoryUnit;
import org.ehcache.event.EventType;

import java.time.Duration;
import java.util.*;
import java.util.concurrent.ConcurrentHashMap;


/**
 * @author Niels
 */
public class EhCacheManager {
    private static final EhCacheManager INSTANCE = new EhCacheManager();
    private static final Map<String, Class> KEY_TYPE_MAP = new ConcurrentHashMap<>();
    private static final Map<String, Class> VALUE_TYPE_MAP = new ConcurrentHashMap<>();
    private static final long MAX_SIZE_OF_CACHE_OBJ_GRAPH = 5 * 1024 * 1024;

    private CacheManager cacheManager;

    private EhCacheManager() {
        init();
```

```
    }

    public static EhCacheManager getInstance() {
        return INSTANCE;
    }

    /**
     * ehcache
     * Initialize method, to create the ehcache manager.
     */
    private void init() {
        cacheManager = CacheManagerBuilder.newCacheManagerBuilder().build(true);
    }

    /**
     *
     * Create a cache container.
     *
     * @param title     ,cache name
     * @param params    ,init parameters
     */
    public void createCache(String title, CacheMapParams params) {
        AssertUtil.canNotEmpty(params.getHeapMb());
        AssertUtil.canNotEmpty(params.getKeyType());
        AssertUtil.canNotEmpty(params.getValueType());
        CacheConfigurationBuilder builder =
CacheConfigurationBuilder.newCacheConfigurationBuilder(params.getKeyType(),
params.getValueType(),
                ResourcePoolsBuilder.newResourcePoolsBuilder().heap(params.getHeapMb(),
MemoryUnit.MB)
        );
        if (params.getListener() != null) {
            Set<EventType> types = new HashSet<>();
            types.add(EventType.CREATED);
            types.add(EventType.UPDATED);
            types.add(EventType.EVICTED);
            types.add(EventType.EXPIRED);
            types.add(EventType.REMOVED);
            CacheEventListenerConfigurationBuilder cacheEventListenerConfiguration =
CacheEventListenerConfigurationBuilder
                    .newEventListenerConfiguration(new EhcacheListener(params.getListener()), types)
                    .unordered().asynchronous();
```

```
            builder = builder.add(cacheEventListenerConfiguration);
        }
        builder = builder.withSizeOfMaxObjectGraph(MAX_SIZE_OF_CACHE_OBJ_GRAPH);
//        builder = builder.withValueSerializer(new
CacheObjectSerializer(params.getValueType())).withKeySerializer(new
CacheObjectSerializer(params.getKeyType()));
        if (params.getTimeToLiveSeconds() > 0) {
            builder =
builder.withExpiry(ExpiryPolicyBuilder.timeToLiveExpiration(Duration.ofSeconds(params.getTime
ToLiveSeconds())));
        }
        if (params.getTimeToIdleSeconds() > 0) {
            builder =
builder.withExpiry(ExpiryPolicyBuilder.timeToIdleExpiration(Duration.ofSeconds(params.getTimeT
oIdleSeconds())));
        }
        if (null != params.getValueCopier()) {
            builder = builder.withValueCopier( params.getValueCopier());
        }
        cacheManager.createCache(title, builder.build());
        KEY_TYPE_MAP.put(title, params.getKeyType());
        VALUE_TYPE_MAP.put(title, params.getValueType());
    }

    /**
     * ehcachecache
     *
     * @param title cache name
     */
    public Cache getCache(String title) {
        Class keyType = KEY_TYPE_MAP.get(title);
        Class valueType = VALUE_TYPE_MAP.get(title);
        if (null == cacheManager || null == keyType || valueType == null) {
            return null;
        }
        return cacheManager.getCache(title, keyType, valueType);
    }

    public void close() {
        cacheManager.close();
    }
```

```
    public void removeCache(String title) {
        cacheManager.removeCache(title);
    }

    public List<String> getCacheTitleList() {
        return new ArrayList<String>(KEY_TYPE_MAP.keySet());
    }

}
```

4:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\cache\src\main\java\io\nuls\cache\model\CacheElement.java

```
package io.nuls.cache.model;

/**
 *
 * @author Niels
 *
 */
public class CacheElement<T> {
    private String cacheTitle;
    private String key;
    private T value;

    public CacheElement() {
    }

    public CacheElement(String key, T value) {
        this.key = key;
        this.value = value;
    }

    public String getCacheTitle() {
        return cacheTitle;
    }

    public void setCacheTitle(String cacheTitle) {
        this.cacheTitle = cacheTitle;
    }

    public String getKey() {
        return key;
```

```
    }

    public void setKey(String key) {
        this.key = key;
    }

    public T getValue() {
        return value;
    }

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

5:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\model\CacheListenerItem.java
 */

package io.nuls.cache.model;

/**
 * @author Niels
 */
public class CacheListenerItem<K, V> {

    public CacheListenerItem() {
    }

    public CacheListenerItem(K k, V newValue, V oldValue) {
        this.key = k;
        this.newValue = newValue;
        this.oldValue = oldValue;
    }

    private K key;
    private V newValue;

    public K getKey() {
        return key;
    }
```

```
    public void setKey(K key) {
        this.key = key;
    }

    public V getNewValue() {
        return newValue;
    }

    public void setNewValue(V newValue) {
        this.newValue = newValue;
    }

    public V getOldValue() {
        return oldValue;
    }

    public void setOldValue(V oldValue) {
        this.oldValue = oldValue;
    }

    private V oldValue;
}

6:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\model\CacheMapParams.java
 */

package io.nuls.cache.model;

import io.nuls.cache.listener.intf.NulsCacheListener;
import org.ehcache.spi.copy.Copier;

import java.io.Serializable;

/**
 * @author: Niels Wang
 */
public class CacheMapParams {
    private int heapMb;
    private int timeToLiveSeconds;
    private int timeToIdleSeconds;
    private NulsCacheListener listener;
```

```
    private Copier valueCopier;
    private Class keyType;
    private Class<? extends Serializable> valueType;

    public CacheMapParams (){}

    public CacheMapParams(int heapMb,Class keyType,Class<? extends Serializable> valueType,
int timeToLiveSeconds, int timeToIdleSeconds, NulsCacheListener listener, Copier valueCopier){
        this.heapMb = heapMb;
        this.keyType = keyType;
        this.valueCopier = valueCopier;
        this.valueType = valueType;
        this.timeToIdleSeconds = timeToIdleSeconds;
        this.timeToLiveSeconds = timeToLiveSeconds;
        this.listener = listener;
    }

    public int getHeapMb() {
        return heapMb;
    }

    public void setHeapMb(int heapMb) {
        this.heapMb = heapMb;
    }

    public int getTimeToLiveSeconds() {
        return timeToLiveSeconds;
    }

    public void setTimeToLiveSeconds(int timeToLiveSeconds) {
        this.timeToLiveSeconds = timeToLiveSeconds;
    }

    public int getTimeToIdleSeconds() {
        return timeToIdleSeconds;
    }

    public void setTimeToIdleSeconds(int timeToIdleSeconds) {
        this.timeToIdleSeconds = timeToIdleSeconds;
    }

    public NulsCacheListener getListener() {
```

```
        return listener;
    }

    public void setListener(NulsCacheListener listener) {
        this.listener = listener;
    }

    public Copier getValueCopier() {
        return valueCopier;
    }

    public Class getKeyType() {
        return keyType;
    }

    public void setKeyType(Class keyType) {
        this.keyType = keyType;
    }

    public Class<? extends Serializable> getValueType() {
        return valueType;
    }

    public void setValueType(Class<? extends Serializable> valueType) {
        this.valueType = valueType;
    }

    public void setValueCopier(Copier valueCopier) {
        this.valueCopier = valueCopier;
    }
}

7:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\utils\CacheObjectSerializer.java
 */

package io.nuls.cache.utils;

import io.nuls.core.tools.log.Log;
import io.protostuff.LinkedBuffer;
import io.protostuff.ProtostuffIOUtil;
import io.protostuff.runtime.RuntimeSchema;
```

```
import org.ehcache.spi.serialization.Serializer;
import org.ehcache.spi.serialization.SerializerException;

import java.nio.ByteBuffer;

/**
 * @author: Niels Wang
 */
public class CacheObjectSerializer<T> implements Serializer<T> {

    private final RuntimeSchema<T> schema;
    private final Class clazz;

    public CacheObjectSerializer(Class clazz) {
//        System.out.println("init++++" + clazz);
        schema = RuntimeSchema.createFrom(clazz);
        this.clazz = clazz;
    }

    @Override
    public ByteBuffer serialize(T o) throws SerializerException {
//        System.out.println("serialize+++" + o);
        byte[] bytes = ProtostuffIOUtil.toByteArray(o, schema,
LinkedBuffer.allocate(LinkedBuffer.DEFAULT_BUFFER_SIZE));
//        System.out.println("hahaha==========" + bytes.length);
        return ByteBuffer.wrap(bytes);
    }

    @Override
    public T read(ByteBuffer byteBuffer) throws SerializerException {
//        System.out.println("read+++" + byteBuffer);
        T t = null;
        try {
            t = (T) clazz.newInstance();
        } catch (InstantiationException e) {
            Log.error(e);
            return null;
        } catch (IllegalAccessException e) {
            Log.error(e);
            return null;
        }
        ProtostuffIOUtil.mergeFrom(byteBuffer.array(), t, schema);
```

```
        return t;
    }

    @Override
    public boolean equals(T o, ByteBuffer byteBuffer) throws SerializerException {
//        System.out.println("equals+++" + byteBuffer);
        if (o == null) {
            return false;
        }
        return o.equals(this.read(byteBuffer));
    }
}

8:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\main\java\io\nuls\cache\utils\EhcacheListener.java
 */

package io.nuls.cache.utils;

import io.nuls.cache.listener.intf.NulsCacheListener;
import io.nuls.cache.model.CacheListenerItem;
import org.ehcache.event.CacheEvent;
import org.ehcache.event.CacheEventListener;

/**
 * @author Niels
 */
public class EhcacheListener implements CacheEventListener {

    private final NulsCacheListener listener;

    public EhcacheListener(NulsCacheListener listener) {
        this.listener = listener;
    }

    @Override
    public void onEvent(CacheEvent event) {
        CacheListenerItem item = new CacheListenerItem(event.getKey(), event.getNewValue(),
event.getOldValue());
        switch (event.getType()) {
            case CREATED:
                listener.onCreate(item);
```

```
                break;
            case EVICTED:
                listener.onEvict(item);
                break;
            case EXPIRED:
                listener.onExpire(item);
                break;
            case REMOVED:
                listener.onRemove(item);
                break;
            case UPDATED:
                listener.onUpdate(item);
                break;
            default:
                return;
        }
    }
}

9:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\cache\src\test\java\io\nuls\cache\CacheMapTest.java
 */
package io.nuls.cache;

import io.nuls.cache.listener.intf.NulsCacheListener;
import io.nuls.cache.model.CacheListenerItem;
import org.ehcache.spi.copy.Copier;
import org.junit.Before;
import org.junit.Test;

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

import static org.junit.Assert.*;

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

```
    private CacheMap<String, ValueData> cacheMap;
    private NulsCacheListener<String, ValueData> listener;
    private Copier<ValueData> valueCopier;
    protected Map<String, Boolean> map = new HashMap<>();

    @Before
    public void before() {
        listener = new NulsCacheListener<String, ValueData>() {
            @Override
            public void onCreate(CacheListenerItem<String, ValueData> item) {
                map.put(item.getKey() + "_create", true);
                System.out.println("create");
            }

            @Override
            public void onEvict(CacheListenerItem<String, ValueData> item) {
                map.put(item.getKey() + "_evict", true);
                System.out.println("evict");
            }

            @Override
            public void onRemove(CacheListenerItem<String, ValueData> item) {
                map.put(item.getKey() + "_remove", true);
                System.out.println("remove");
            }

            @Override
            public void onUpdate(CacheListenerItem<String, ValueData> item) {
                map.put(item.getKey() + "_update", true);
                System.out.println("update");
            }

            @Override
            public void onExpire(CacheListenerItem<String, ValueData> item) {
                map.put(item.getKey() + "_expire", true);
                System.out.println("expire");
            }
        };
        valueCopier = new Copier<ValueData>() {
            @Override
            public ValueData copyForRead(ValueData valueData) {
//                return valueData.copy();
```

```
            return valueData;
        }

        @Override
        public ValueData copyForWrite(ValueData valueData) {
            return valueData.copy();
        }
    };
    this.cacheMap = new CacheMap("test-cache", 1, String.class, ValueData.class, 10, 10,
listener, valueCopier);
  }

  @Test
  public void test() {
    ValueData data1 = new ValueData();
    data1.setTime(1000L);
    data1.setName("test1");
    data1.setCode(1);
    cacheMap.put(data1.getName(), data1);
    ValueData data_get = cacheMap.get(data1.getName());
    assertNotNull(data_get);
    assertNotEquals(data1, data_get);
    long start = System.currentTimeMillis();
    while (null == map.get(data1.getName() + "_create")) {
        try {
            Thread.sleep(10);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    System.out.println("create late:" + (System.currentTimeMillis() - start));

    assertTrue(map.get(data1.getName() + "_create"));



    data1.setTime(10001L);
    cacheMap.put(data1.getName(), data1);
//      assertTrue(map.get(data1.getName() + "_update"));

    try {
        Thread.sleep(12000L);
    } catch (InterruptedException e) {
```

```
            e.printStackTrace();
            assertTrue(false);
        }
        assertNull(cacheMap.get(data1.getName()));

        start = System.currentTimeMillis();
        while (null == map.get(data1.getName() + "_expire")) {
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("expire late:" + (System.currentTimeMillis() - start));
        assertTrue(map.get(data1.getName() + "_expire"));

        cacheMap.put(data1.getName(), data1);
        cacheMap.remove(data1.getName());
        start = System.currentTimeMillis();
        while (null == map.get(data1.getName() + "_remove")) {
            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("remove late:" + (System.currentTimeMillis() - start));
        assertTrue(map.get(data1.getName() + "_remove"));
    }

    public void testSpeed(){
        CacheMap<String,ValueData> cacheMap = new CacheMap("test-cache-speed", 1024,
String.class, ValueData.class, 0, 0, null, null);
        Map<String,ValueData> map = new HashMap<>();
        List<ValueData> list = new ArrayList<>();
        for (int i = 0; i < 1000000; i++) {
            ValueData data = new ValueData();
            data.setTime(1000L);
            data.setName("test" + i);
            data.setCode(i);
            list.add(data);
        }
```

```
        long start = System.currentTimeMillis();
        for(int i=0;i<1000000;i++){
            ValueData valueData =list.get(i);
            cacheMap.put(valueData.getName(),valueData);
        }
        System.out.println("cache put use:"+(System.currentTimeMillis()-start));
        start = System.currentTimeMillis();
        for(int i=0;i<1000000;i++){
            ValueData valueData =cacheMap.get("test"+i);
        }
        System.out.println("cache get use:"+(System.currentTimeMillis()-start));


//=================================================================


        start = System.currentTimeMillis();
        for(int i=0;i<1000000;i++){
            ValueData valueData = list.get(i);
            map.put(valueData.getName(),valueData);
        }
        System.out.println("map put use:"+(System.currentTimeMillis()-start));
        start = System.currentTimeMillis();
        for(int i=0;i<1000000;i++){
            ValueData valueData =map.get("test"+i);
        }
        System.out.println("map get use:"+(System.currentTimeMillis()-start));
        System.out.println();
    }

    static class ValueData implements Serializable {
        private int code;
        private String name;
        private long time;

        public int getCode() {
            return code;
        }

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

```
        public String getName() {
            return name;
        }

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

        public long getTime() {
            return time;
        }

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

        public ValueData copy() {
            ValueData data = new ValueData();
            data.setCode(code);
            data.setName(name);
            data.setTime(time);
            return data;
        }
    }
}

10:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\aop\AopUtils.java
 */
package io.nuls.core.tools.aop;

import net.sf.cglib.proxy.Enhancer;
import net.sf.cglib.proxy.MethodInterceptor;

/**
 * Created by Niels on 2017/10/13.
 *
 */
public class AopUtils {

    public static final <T> T createProxy(Class<T> clazz,MethodInterceptor interceptor) {
```

```
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(clazz);
        enhancer.setCallback(new NulsMethodInterceptor(interceptor));

        return (T) enhancer.create();
    }

    public static final <T> T createProxy(Class<T> clazz,Class[] paramsClass,Object[]
params,MethodInterceptor interceptor) {
        Enhancer enhancer = new Enhancer();
        enhancer.setSuperclass(clazz);
        enhancer.setCallback(new NulsMethodInterceptor(interceptor));
        return (T) enhancer.create(paramsClass,params);
    }

//    public static final<T> T createObjProxy(T obj, MethodInterceptor interceptor) {
//        Enhancer enhancer = new Enhancer();
//        enhancer.setSuperclass(obj.getClass());
//        enhancer.setCallback(new ObjectProxyInterceptor(obj, interceptor));
//        return (T) enhancer.create();
//    }
}

11:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\aop\NulsMethodInterceptor.java
 */
package io.nuls.core.tools.aop;

import net.sf.cglib.proxy.MethodInterceptor;
import net.sf.cglib.proxy.MethodProxy;

import java.lang.reflect.Method;

/**
 * @author Niels
 */
public final class NulsMethodInterceptor implements MethodInterceptor {

    private MethodInterceptor interceptor;

    public NulsMethodInterceptor(MethodInterceptor interceptor) {
        this.interceptor = interceptor;
```

```
    }

    @Override
    public Object intercept(Object o, Method method, Object[] objects, MethodProxy methodProxy)
throws Throwable {
        if (method.isBridge()) {
            return methodProxy.invokeSuper(o, objects);
        }
        return interceptor.intercept(o, method, objects, methodProxy);
    }
}

12:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\array\ArraysTool.java
 */

package io.nuls.core.tools.array;

import java.util.Arrays;

/**
 * @author: Niels Wang
 */
public class ArraysTool {
    /**
     *
     * Splices the array into a large array containing all of the arrays in the incoming order.
     *
     * @param arrays A collection of arrays that you want to concatenate.
     * @return  the result of the Joining together
     */
    public static final byte[] concatenate(byte[]... arrays) {
        int length = 0;
        for (byte[] array : arrays) {
            length += array.length;
        }
        byte[] t = new byte[length];
        int offset = 0;
        for (byte[] array : arrays) {
            System.arraycopy(array, 0, t, offset, array.length);
            offset += array.length;
        }
```

```
        return t;
    }

    public static final boolean isEmptyOrNull(byte[] bytes) {
        return (bytes == null || bytes.length == 0);
    }

    public static boolean arrayEquals(byte[] array1, byte[] array2) {
        return Arrays.equals(array1, array2);
    }
}

13:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\BloomFilter\BloomFilter.java
 */
package io.nuls.core.tools.BloomFilter;

import static java.lang.Math.E;
import static java.lang.Math.log;
import static java.lang.Math.max;
import static java.lang.Math.min;
import static java.lang.Math.pow;

import java.util.Arrays;

public class BloomFilter {

    private static final int MAX_FILTER_SIZE = 1000000;
    private static final int MAX_HASH_FUNCS = 50;

    private byte[] data;
    private long hashFuncs;
    private long nTweak;

    private int elements;
    private double falsePositiveRate;
    private long randomNonce;

    public BloomFilter(int elements, double falsePositiveRate, long randomNonce) {
        this.elements = elements;
        this.falsePositiveRate = falsePositiveRate;
        this.randomNonce = randomNonce;
```

```
    init();
}

public void init() {
    int size = (int) (-1 / (pow(log(2), 2)) * elements * log(falsePositiveRate));
    size = max(1, min(size, (int) MAX_FILTER_SIZE * 8) / 8);
    data = new byte[size];
    hashFuncs = (int) (data.length * 8 / (double) elements * log(2));
    hashFuncs = max(1, min(hashFuncs, MAX_HASH_FUNCS));
    this.nTweak = randomNonce;
}

public double getFalsePositiveRate(int elements) {
    return pow(1 - pow(E, -1.0 * (hashFuncs * elements) / (data.length * 8)), hashFuncs);
}

@Override
public String toString() {
    return "Bloom Filter of size " + data.length + " with " + hashFuncs + " hash functions.";
}

private static int rotateLeft32(int x, int r) {
    return (x << r) | (x >>> (32 - r));
}

public static int murmurHash3(byte[] data, long nTweak, int hashNum, byte[] object) {
    int h1 = (int) (hashNum * 0xFBA4C795L + nTweak);
    final int c1 = 0xcc9e2d51;
    final int c2 = 0x1b873593;

    int numBlocks = (object.length / 4) * 4;
    // body
    for (int i = 0; i < numBlocks; i += 4) {
        int k1 = (object[i] & 0xFF) |
                ((object[i + 1] & 0xFF) << 8) |
                ((object[i + 2] & 0xFF) << 16) |
                ((object[i + 3] & 0xFF) << 24);

        k1 *= c1;
        k1 = rotateLeft32(k1, 15);
        k1 *= c2;
```

```
            h1 ^= k1;
            h1 = rotateLeft32(h1, 13);
            h1 = h1 * 5 + 0xe6546b64;
        }

        int k1 = 0;
        switch (object.length & 3) {
            case 3:
                k1 ^= (object[numBlocks + 2] & 0xff) << 16;
                // Fall through.
            case 2:
                k1 ^= (object[numBlocks + 1] & 0xff) << 8;
                // Fall through.
            case 1:
                k1 ^= (object[numBlocks] & 0xff);
                k1 *= c1;
                k1 = rotateLeft32(k1, 15);
                k1 *= c2;
                h1 ^= k1;
                // Fall through.
            default:
                // Do nothing.
                break;
        }

        // finalization
        h1 ^= object.length;
        h1 ^= h1 >>> 16;
        h1 *= 0x85ebca6b;
        h1 ^= h1 >>> 13;
        h1 *= 0xc2b2ae35;
        h1 ^= h1 >>> 16;

        return (int) ((h1 & 0xFFFFFFFFL) % (data.length * 8));
    }

    public synchronized boolean contains(byte[] object) {
        for (int i = 0; i < hashFuncs; i++) {
            if (!checkBitLE(data, murmurHash3(data, nTweak, i, object))) {
                return false;
            }
        }
```

```
        return true;
    }

    public synchronized void insert(byte[] object) {
        for (int i = 0; i < hashFuncs; i++) {
            setBitLE(data, murmurHash3(data, nTweak, i, object));
        }
    }

    public synchronized boolean matchesAll() {
        for (byte b : data) {
            if (b != (byte) 0xff) {
                return false;
            }
        }
        return true;
    }

    @Override
    public synchronized boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        BloomFilter other = (BloomFilter) o;
        return hashFuncs == other.hashFuncs && nTweak == other.nTweak && Arrays.equals(data,
other.data);
    }

    private static final int[] bitMask = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};

    public static boolean checkBitLE(byte[] data, int index) {
        return (data[index >>> 3] & bitMask[7 & index]) != 0;
    }

    public static void setBitLE(byte[] data, int index) {
        data[index >>> 3] |= bitMask[7 & index];
    }

    public byte[] getData() {
```

```
        return data;
    }

    public long getHashFuncs() {
        return hashFuncs;
    }
}

14:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\calc\DoubleUtils.java
 */

package io.nuls.core.tools.calc;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.text.DecimalFormat;

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

    public static final int DEFAULT_SCALE = 8;

    public static BigDecimal createBigDecimal(double value) {
        return BigDecimal.valueOf(value);
    }

    public static double round(double value, int scale, int roundingMode) {
        BigDecimal bd = createBigDecimal(value);
        bd = bd.setScale(scale, roundingMode);
        return bd.doubleValue();
    }

    public static double round(double value, int scale) {
        return round(value, scale, BigDecimal.ROUND_HALF_UP);
    }

    public static double round(double value) {
        return round(value, DEFAULT_SCALE);
    }
```

```
public static String getRoundStr(Double value, int scale, boolean hasThousands) {
    if (null == value) {
        return "";
    }
    String suffix = "";
    for (int i = 0; i < scale; i++) {
        if (i == 0) {
            suffix += ".";
        }
        suffix += "0";
    }

    if (hasThousands) {
        return new DecimalFormat("###,##0" + suffix).format(round(value, scale));
    } else {
        return new DecimalFormat("##0" + suffix).format(round(value, scale));
    }
}

public static String getRoundStr(Double value, int scale) {
    return getRoundStr(value, scale, false);
}

public static String getRoundStr(Double value) {
    return getRoundStr(value, DEFAULT_SCALE, false);
}

public static Double parseDouble(String value) {
    if (null == value || "".equals(value.trim())) {
        return null;
    }
    return Double.parseDouble(value.replaceAll(",", "").trim());
}

public static Double parseDouble(String value, int scale) {
    if (null == value || "".equals(value.trim())) {
        return null;
    }
    return round(Double.parseDouble(value.replaceAll(",", "").trim()), scale);
}
```

```
public static double sum(double d1, double d2) {
    return round(sum(createBigDecimal(d1), createBigDecimal(d2)).doubleValue());
}

public static double sub(double d1, double d2) {
    return round(sub(createBigDecimal(d1), createBigDecimal(d2)).doubleValue());
}

public static double mul(double d1, double d2) {
    return mul(createBigDecimal(d1), createBigDecimal(d2)).doubleValue();
}

public static double mul(double d1, double d2, int scale) {
    return round(mul(createBigDecimal(d1), createBigDecimal(d2)).doubleValue(), scale);
}

public static double div(double d1, double d2, int scale) {
    return round(div(createBigDecimal(d1), createBigDecimal(d2)).doubleValue(), scale);
}

public static double div(double d1, double d2) {
    return div(d1, d2, DEFAULT_SCALE);
}

public static BigDecimal sum(BigDecimal bd1, BigDecimal bd2) {
    return bd1.add(bd2);
}

public static BigDecimal sub(BigDecimal bd1, BigDecimal bd2) {
    return bd1.subtract(bd2);
}

public static BigDecimal mul(BigDecimal bd1, BigDecimal bd2) {
    return bd1.multiply(bd2);
}

public static BigDecimal div(BigDecimal bd1, BigDecimal bd2) {
    if (bd2.equals(BigDecimal.ZERO)) {
        throw new IllegalArgumentException("0");
    }
    return bd1.divide(bd2, 12, RoundingMode.HALF_UP);
}
```

```
    public static BigDecimal sum(BigDecimal bd1, double d2) {
        return sum(bd1, createBigDecimal(d2));
    }

    public static BigDecimal sub(BigDecimal bd1, double d2) {
        return sub(bd1, createBigDecimal(d2));
    }

    public static BigDecimal mul(BigDecimal bd1, double d2) {
        return mul(bd1, createBigDecimal(d2));
    }

    public static BigDecimal div(BigDecimal bd1, double d2) {
        return div(bd1, createBigDecimal(d2));
    }

    public static double abs(double d1) {
        return Math.abs(d1);
    }

    public static long longValue(double val) {
        return createBigDecimal(val).longValue();
    }
}

15:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\calc\LongUtils.java
 */

package io.nuls.core.tools.calc;

import java.math.BigInteger;

/**
 *
 * Long integer computing utility class.
 *
 * @author: Niels Wang
 */
public class LongUtils {
```

```
    public static long add(long val1, long val2) {
        BigInteger value1 = BigInteger.valueOf(val1);
        BigInteger value2 = BigInteger.valueOf(val2);
        return value1.add(value2).longValue();
    }

    public static long sub(long val1, long val2) {
        BigInteger value1 = BigInteger.valueOf(val1);
        BigInteger value2 = BigInteger.valueOf(val2);
        return value1.subtract(value2).longValue();
    }

    public static long mul(long val1, long val2) {
        BigInteger value1 = BigInteger.valueOf(val1);
        BigInteger value2 = BigInteger.valueOf(val2);
        return value1.multiply(value2).longValue();
    }

    public static double exactDiv(long val1, long val2) {
        return DoubleUtils.div(val1, val2);
    }

    public static long div(long val1, long val2) {
        BigInteger value1 = BigInteger.valueOf(val1);
        BigInteger value2 = BigInteger.valueOf(val2);
        return value1.divide(value2).longValue();
    }

    public static long mod(long val1, long val2) {
        BigInteger value1 = BigInteger.valueOf(val1);
        BigInteger value2 = BigInteger.valueOf(val2);
        return value1.mod(value2).longValue();
    }
}

16:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\cfg\ConfigLoader.java
 */
package io.nuls.core.tools.cfg;

import org.ini4j.Config;
import org.ini4j.Ini;
```

```
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Properties;

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

    public static Properties loadProperties(String fileName) throws IOException {
        InputStream is = ConfigLoader.class.getClassLoader().getResourceAsStream(fileName);
        Properties prop = new Properties();
        prop.load(is);
        is.close();
        return prop;
    }

    public static IniEntity loadIni(String fileName) throws IOException {
        Config cfg = new Config();
        URL url = ConfigLoader.class.getClassLoader().getResource(fileName);
        cfg.setMultiSection(true);
        Ini ini = new Ini();
        ini.setConfig(cfg);
        ini.load(url);
        return new IniEntity(ini);
    }

}

17:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\cfg\IniEntity.java
 */
package io.nuls.core.tools.cfg;

import io.nuls.core.tools.str.StringUtils;
import org.ini4j.Ini;
import org.ini4j.Profile;

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

```
import java.util.Map;
import java.util.Set;

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

    private final Ini ini;

    public IniEntity(Ini ini) {
        this.ini = ini;
    }


    public String getCfgValue(String section, String key) throws Exception {
        Profile.Section ps = ini.get(section);
        if (null == ps) {
            throw new Exception("CONFIGURATION_ITEM_DOES_NOT_EXIST");
        }
        String value = ps.get(key);
        if (StringUtils.isBlank(value)) {
            throw new Exception("CONFIGURATION_ITEM_DOES_NOT_EXIST");
        }
        return value;
    }

    public <T> T getCfgValue(String section, String key, T defaultValue) {
        Profile.Section ps = ini.get(section);
        if (null == ps) {
            return defaultValue;
        }
        String value = ps.get(key);
        if (StringUtils.isBlank(value)) {
            return defaultValue;
        }
        return getValueByType(value, defaultValue);
    }

    protected static <T> T getValueByType(String value, T defaultValue) {
        if (defaultValue instanceof Integer) {
            return (T) ((Integer) Integer.parseInt(value));
```

```
        } else if (defaultValue instanceof Long) {
            return (T) ((Long) Long.parseLong(value));
        } else if (defaultValue instanceof Float) {
            return (T) ((Float) Float.parseFloat(value));
        } else if (defaultValue instanceof Double) {
            return (T) ((Double) Double.parseDouble(value));
        } else if (defaultValue instanceof Boolean) {
            return (T) ((Boolean) Boolean.parseBoolean(value));
        }
        return (T) value;
    }

    public Profile.Section getSection(String section) throws Exception {
        Profile.Section ps = ini.get(section);
        if (null == ps) {
            throw new Exception("CONFIGURATION_ITEM_DOES_NOT_EXIST");
        }
        return ps;
    }

    public List<String> getSectionList() {
        Set<Map.Entry<String, Profile.Section>> entrySet = ini.entrySet();
        List<String> list = new ArrayList<>();
        for (Map.Entry<String, Profile.Section> entry : entrySet) {
            list.add(entry.getKey());
        }
        return list;
    }

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

18:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\cfg\PropertiesEntity.java
 */
package io.nuls.core.tools.cfg;

import java.util.Properties;
```

```
/**
 * @author Niels
 */
public class PropertiesEntity {

    private final Properties prop;

    public PropertiesEntity(Properties prop) {
        this.prop = prop;
    }

    public <T> T getPropValue(String name, T defaultValue) {
        if (prop.getProperty(name) == null) {
            return defaultValue;
        }
        String value = prop.getProperty(name);
        return IniEntity.getValueByType(value, defaultValue);
    }
}

19:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\AESEncrypt.java
 */
package io.nuls.core.tools.crypto;

import io.nuls.core.tools.crypto.Exception.CryptoException;
import org.spongycastle.crypto.BufferedBlockCipher;
import org.spongycastle.crypto.engines.AESFastEngine;
import org.spongycastle.crypto.modes.CBCBlockCipher;
import org.spongycastle.crypto.paddings.PaddedBufferedBlockCipher;
import org.spongycastle.crypto.params.KeyParameter;
import org.spongycastle.crypto.params.ParametersWithIV;

import java.io.UnsupportedEncodingException;
import java.security.SecureRandom;
import java.util.Arrays;

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

```
//todo
public class AESEncrypt {

    private static final SecureRandom SECURE_RANDOM = new SecureRandom();

    public static byte[] encrypt(byte[] plainBytes, String password) {
        EncryptedData ed = encrypt(plainBytes, new
KeyParameter(Sha256Hash.hash(password.getBytes())));
        return ed.getEncryptedBytes();

    }

    public static EncryptedData encrypt(byte[] plainBytes, KeyParameter aesKey) {
        return encrypt(plainBytes, null, aesKey);
    }

    public static EncryptedData encrypt(byte[] plainBytes, byte[] iv, KeyParameter aesKey) throws
RuntimeException {
        Util.checkNotNull(plainBytes);
        Util.checkNotNull(aesKey);

        try {
            if (iv == null) {
                iv = EncryptedData.DEFAULT_IV;
                //SECURE_RANDOM.nextBytes(iv);
            }

            ParametersWithIV keyWithIv = new ParametersWithIV(aesKey, iv);

            // Encrypt using AES.
            BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new CBCBlockCipher(new
AESFastEngine()));
            cipher.init(true, keyWithIv);
            byte[] encryptedBytes = new byte[cipher.getOutputSize(plainBytes.length)];
            final int length1 = cipher.processBytes(plainBytes, 0, plainBytes.length, encryptedBytes,
0);
            final int length2 = cipher.doFinal(encryptedBytes, length1);

            return new EncryptedData(iv, Arrays.copyOf(encryptedBytes, length1 + length2));
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
```

```
    }

    public static byte[] decrypt(byte[] dataToDecrypt, String password) throws CryptoException{
        byte[] defaultiv = new byte[16];
        EncryptedData data = new EncryptedData(defaultiv, dataToDecrypt);
        return decrypt(data, new KeyParameter(Sha256Hash.hash(password.getBytes())));
    }

    public static byte[] decrypt(byte[] dataToDecrypt, String password, String charset) throws
CryptoException,UnsupportedEncodingException {
        byte[] defaultiv = new byte[16];
        EncryptedData data = new EncryptedData(defaultiv, dataToDecrypt);
        return decrypt(data, new KeyParameter(Sha256Hash.hash(password.getBytes(charset))));
    }

    public static byte[] decrypt(EncryptedData dataToDecrypt, KeyParameter aesKey) throws
CryptoException {
        Util.checkNotNull(dataToDecrypt);
        Util.checkNotNull(aesKey);

        try {
            ParametersWithIV keyWithIv = new ParametersWithIV(new
KeyParameter(aesKey.getKey()), dataToDecrypt.getInitialisationVector());

            // Decrypt the validator.
            BufferedBlockCipher cipher = new PaddedBufferedBlockCipher(new CBCBlockCipher(new
AESFastEngine()));
            cipher.init(false, keyWithIv);

            byte[] cipherBytes = dataToDecrypt.getEncryptedBytes();
            byte[] decryptedBytes = new byte[cipher.getOutputSize(cipherBytes.length)];
            final int length1 = cipher.processBytes(cipherBytes, 0, cipherBytes.length, decryptedBytes,
0);
            final int length2 = cipher.doFinal(decryptedBytes, length1);

            return Arrays.copyOf(decryptedBytes, length1 + length2);
        } catch (Exception e) {
            throw new CryptoException();
        }
    }

}
```

```
20:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Base58.java
 */
package io.nuls.core.tools.crypto;

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

public class Base58 {
    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 Exception if the given string is not a valid base58 string
     */
    public static byte[] decode(String input) throws Exception {
        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 Exception("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 Exception {
    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)
 * @return byte
 * @throws Exception if the input is not base 58 or the checksum does not validate.
 */
public static byte[] decodeChecked(String input) throws Exception {
    byte[] decoded  = decode(input);
    if (decoded.length < 4) {
        throw new Exception( "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 Exception( "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;
    }

}

21:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\ByteStreams.java
 */
package io.nuls.core.tools.crypto;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

public class ByteStreams {

    private static final int BUF_SIZE = 0x1000; // 4K
```

```
    public static byte[] toByteArray(InputStream in) throws IOException {
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        copy(in, out);
        return out.toByteArray();
    }

    public static long copy(InputStream from, OutputStream to)
            throws IOException {
        Util.checkNotNull(from);
        Util.checkNotNull(to);
        byte[] buf = new byte[BUF_SIZE];
        long total = 0;
        while (true) {
            int r = from.read(buf);
            if (r == -1) {
                break;
            }
            to.write(buf, 0, r);
            total += r;
        }
        return total;
    }
}
```

22:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\crypto\Cipher.java

```
 */
package io.nuls.core.tools.crypto;

import org.bouncycastle.crypto.AsymmetricCipherKeyPair;
import org.bouncycastle.crypto.params.ECPrivateKeyParameters;
import org.bouncycastle.crypto.params.ECPublicKeyParameters;
import org.bouncycastle.math.ec.ECPoint;

import java.math.BigInteger;

/**
 * Created by facjas on 2017/11/20.
 */
public class Cipher {
    private int ct;
    private ECPoint p2;
```

```
private SM3Digest sm3keybase;
private SM3Digest sm3c3;
private byte key[];
private byte keyOff;

public Cipher() {
    this.ct = 1;
    this.key = new byte[32];
    this.keyOff = 0;
}

private void reset() {
    this.sm3keybase = new SM3Digest();
    this.sm3c3 = new SM3Digest();

    byte p[] = Util.byteConvert32Bytes(p2.getX().toBigInteger());
    this.sm3keybase.update(p, 0, p.length);
    this.sm3c3.update(p, 0, p.length);

    p = Util.byteConvert32Bytes(p2.getY().toBigInteger());
    this.sm3keybase.update(p, 0, p.length);
    this.ct = 1;
    nextkey();
}

private void nextkey() {
    SM3Digest sm3keycur = new SM3Digest(this.sm3keybase);
    sm3keycur.update((byte) (ct >> 24 & 0xff));
    sm3keycur.update((byte) (ct >> 16 & 0xff));
    sm3keycur.update((byte) (ct >> 8 & 0xff));
    sm3keycur.update((byte) (ct & 0xff));
    sm3keycur.doFinal(key, 0);
    this.keyOff = 0;
    this.ct++;
}

public ECPoint initEnc(SM2 sm2, ECPoint userKey) {
    AsymmetricCipherKeyPair key = sm2.ecc_key_pair_generator.generateKeyPair();
    ECPrivateKeyParameters ecpriv = (ECPrivateKeyParameters) key.getPrivate();
    ECPublicKeyParameters ecpub = (ECPublicKeyParameters) key.getPublic();
    BigInteger k = ecpriv.getD();
    ECPoint c1 = ecpub.getQ();
```

```
        this.p2 = userKey.multiply(k);
        reset();
        return c1;
    }

    public void encrypt(byte data[]) {
        this.sm3c3.update(data, 0, data.length);
        for (int i = 0; i < data.length; i++) {
            if (keyOff == key.length) {
                nextkey();
            }
            data[i] ^= key[keyOff++];
        }
    }

    public void initDec(BigInteger userD, ECPoint c1) {
        this.p2 = c1.multiply(userD);
        reset();
    }

    public void decrypt(byte data[]) {
        for (int i = 0; i < data.length; i++) {
            if (keyOff == key.length) {
                nextkey();
            }
            data[i] ^= key[keyOff++];
        }
        this.sm3c3.update(data, 0, data.length);
    }

    public void dofinal(byte c3[]) {
        byte p[] = Util.byteConvert32Bytes(p2.getY().toBigInteger());
        this.sm3c3.update(p, 0, p.length);
        this.sm3c3.doFinal(c3, 0);
        reset();
    }
}

23:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\ECKey.java
 */
package io.nuls.core.tools.crypto;
```

```
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.google.common.primitives.UnsignedBytes;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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.AsymmetricCipherKeyPair;
import org.spongycastle.crypto.digests.SHA256Digest;
import org.spongycastle.crypto.ec.CustomNamedCurves;
import org.spongycastle.crypto.generators.ECKeyPairGenerator;
import org.spongycastle.crypto.params.ECDomainParameters;
import org.spongycastle.crypto.params.ECKeyGenerationParameters;
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.ECPoint;
import org.spongycastle.math.ec.FixedPointCombMultiplier;
import org.spongycastle.math.ec.FixedPointUtil;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Comparator;

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

    private static final Logger log = LoggerFactory.getLogger(ECKey.class);

    private static final X9ECParameters CURVE_PARAMS =
```

```
CustomNamedCurves.getByName("secp256k1");
    public static final ECDomainParameters CURVE;
    public static final BigInteger HALF_CURVE_ORDER;

    private static final SecureRandom SECURE_RANDOM;    //

    static {
        if (Util.isAndroidRuntime()) {
            new LinuxSecureRandom();
        }

        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);
        SECURE_RANDOM = new SecureRandom();
    }

    protected final BigInteger priv;    //
    private final ECPoint pub;           //

    protected EncryptedData encryptedPrivateKey;

    protected long creationTimeSeconds;

    public ECKey() {
        this(SECURE_RANDOM);
    }

    public ECKey(SecureRandom secureRandom) {
        ECKeyPairGenerator generator = new ECKeyPairGenerator();
        ECKeyGenerationParameters keygenParams = new ECKeyGenerationParameters(CURVE,
secureRandom);
        generator.init(keygenParams);
        AsymmetricCipherKeyPair keypair = generator.generateKeyPair();
        ECPrivateKeyParameters privParams = (ECPrivateKeyParameters) keypair.getPrivate();
        ECPublicKeyParameters pubParams = (ECPublicKeyParameters) keypair.getPublic();
        priv = privParams.getD();
        pub = pubParams.getQ();
        creationTimeSeconds = System.currentTimeMillis();
    }
```

```
/**
 *
 * @param priv
 * @param pub
 */
private ECKey(BigInteger priv, ECPoint pub) {
    if (priv != null) {
        //01
        Util.checkState(!priv.equals(BigInteger.ZERO));
        Util.checkState(!priv.equals(BigInteger.ONE));
    }
    this.priv = priv;
    this.pub = Util.checkNotNull(pub);
    creationTimeSeconds = System.currentTimeMillis();
}

public static ECKey fromPrivate(BigInteger privKey) {
    return fromPrivate(privKey, true);
}

/**
 *
 * @param privKey private key
 * @param compressed compressed
 * @return ECKey
 */
public static ECKey fromPrivate(BigInteger privKey, boolean compressed) {

    ECPoint point = publicPointFromPrivate(privKey);
    return new ECKey(privKey, getPointWithCompression(point, compressed));
}



public static ECKey fromPublicOnly(byte[] pubKey) {
    return new ECKey(null, CURVE.getCurve().decodePoint(pubKey));
}

public static ECKey fromPublicOnly(ECPoint pub) {
    return new ECKey(null, pub);
}
```

```
public static ECPoint publicPointFromPrivate(BigInteger privKey) {
    if (privKey.bitLength() > CURVE.getN().bitLength()) {
        privKey = privKey.mod(CURVE.getN());
    }
    return new FixedPointCombMultiplier().multiply(CURVE.getG(), privKey);
}

public static ECKey fromEncrypted(EncryptedData encryptedPrivateKey, byte[] pubKey) {
    ECKey key = fromPublicOnly(pubKey);
    AssertUtil.canNotEmpty(encryptedPrivateKey, "encryptedPrivateKey can not null!");
    key.encryptedPrivateKey = encryptedPrivateKey;
    return key;
}

protected byte[] getPubKey(boolean compressed) {
    return pub.getEncoded(compressed);
}

/**
 * ,
 *
 * @return byte[]
 */
public byte[] getPubKey() {
    return getPubKey(true);
}

/**
 *
 *
 * @return BigInteger
 */
@JsonIgnore
public BigInteger getPrivKey() {
    if (priv == null) {
        throw new MissingPrivateKeyException();
    }
    return priv;
}

/**
 *
```

```
     *
     * @return byte[]
     */
    @JsonIgnore
    public byte[] getPrivKeyBytes() {
        return getPrivKey().toByteArray();
    }

    /**
     * 16
     *
     * @return String
     */
    @JsonIgnore
    public String getPrivateKeyAsHex() {
        return Hex.encode(getPrivKeyBytes());
    }

    /**
     * 16
     *
     * @return String
     */
    public String getPublicKeyAsHex() {
        return getPublicKeyAsHex(false);
    }

    public String getPublicKeyAsHex(boolean compressed) {
        return Hex.encode(getPubKey(compressed));
    }

    /*
     *
     */
    @SuppressWarnings("deprecation")
    private static ECPoint getPointWithCompression(ECPoint point, boolean compressed) {
        if (point.isCompressed() == compressed) {
            return point;
        }
        point = point.normalize();
        BigInteger x = point.getAffineXCoord().toBigInteger();
        BigInteger y = point.getAffineYCoord().toBigInteger();
```

```
        return CURVE.getCurve().createPoint(x, y, compressed);
    }

    public static boolean verify(byte[] data, ECDSASignature signature, byte[] pub) {
        ECDSASigner signer = new ECDSASigner();
        ECPublicKeyParameters params = new
ECPublicKeyParameters(CURVE.getCurve().decodePoint(pub), CURVE);
        signer.init(false, params);
        try {
            return signer.verifySignature(data, signature.r, signature.s);
        } catch (NullPointerException e) {
            log.error("Caught NPE inside bouncy castle", e);
            return false;
        }
    }

    public static boolean verify(byte[] data, byte[] signature, byte[] pub) {
        return verify(data, ECDSASignature.decodeFromDER(signature), pub);
    }

    public boolean verify(byte[] hash, byte[] signature) {
        return ECKey.verify(hash, signature, getPubKey());
    }

    public static class MissingPrivateKeyException extends RuntimeException {
        private static final long serialVersionUID = 2789844760773725676L;
    }

    public static class ECDSASignature {
        public final BigInteger r, s;

        public ECDSASignature(BigInteger r, BigInteger s) {
            this.r = r;
            this.s = s;
        }

        public byte[] encodeToDER() {
            try {
                return derByteStream().toByteArray();
            } catch (IOException e) {
                throw new RuntimeException(e);  // Cannot happen.
            }
```

```
    }

    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) {
            Log.error(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;
        }

        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 validator. 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.
         * @return ECDSASignature
         */
        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;
            }
        }
    }

    public byte[] sign(byte[] hash) {
        return sign(hash, null);
    }

    public byte[] sign(Sha256Hash hash, BigInteger aesKey) {
        return doSign(hash.getBytes(), priv);
    }
```

```
    public byte[] sign(byte[] hash, BigInteger aesKey) {
        return doSign(hash, priv);
    }

    protected byte[] doSign(byte[] input, BigInteger privateKeyForSigning) {
        Util.checkNotNull(privateKeyForSigning);
        ECDSASigner signer = new ECDSASigner(new HMacDSAKCalculator(new
SHA256Digest()));
        ECPrivateKeyParameters privKey = new ECPrivateKeyParameters(privateKeyForSigning,
CURVE);
        signer.init(true, privKey);
        BigInteger[] components = signer.generateSignature(input);
        return new ECDSASignature(components[0],
components[1]).toCanonicalised().encodeToDER();
    }

    public boolean hasPrivKey() {
        return priv != null;
    }

    public boolean isCompressed() {
        return pub.isCompressed();
    }

    public void setCreationTimeSeconds(long creationTimeSeconds) {
        this.creationTimeSeconds = creationTimeSeconds;
    }

    public long getCreationTimeSeconds() {
        return creationTimeSeconds;
    }

    public EncryptedData getEncryptedPrivateKey() {
        return encryptedPrivateKey;
    }

    public void setEncryptedPrivateKey(EncryptedData encryptedPrivateKey) {
        this.encryptedPrivateKey = encryptedPrivateKey;
    }

    public static boolean isValidPrivteHex(String privateHex) {
```

```
        int len = privateHex.length();
        if (len % 2 == 1) {
            return false;
        }

        if (len < 60 || len > 66) {
            return false;
        }
        return true;
    }

    public static final Comparator<ECKey> PUBKEY_COMPARATOR = new
Comparator<ECKey>() {
        private Comparator<byte[]> comparator = UnsignedBytes.lexicographicalComparator();

        @Override
        public int compare(ECKey k1, ECKey k2) {
            return comparator.compare(k1.getPubKey(), k2.getPubKey());
        }
    };
}

24:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\EncryptedData.java
 */
package io.nuls.core.tools.crypto;

import java.util.Arrays;
import java.util.Objects;

/**
 *
 * @author ln
 *
 */
public final class EncryptedData {
    //todo  maybe the default iv is not proper
    //todo  need to support mutiple algs
    public static byte[] DEFAULT_IV = new byte[16];
    private byte[] initialisationVector;//iv
    private byte[] encryptedBytes;//
```

```
    public EncryptedData(byte[] initialisationVector, byte[] encryptedBytes) {
        this.initialisationVector = Arrays.copyOf(initialisationVector, initialisationVector.length);
        this.encryptedBytes = Arrays.copyOf(encryptedBytes, encryptedBytes.length);
    }

    public EncryptedData(byte[] encryptedBytes) {
        this(DEFAULT_IV,encryptedBytes);
    }



    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        EncryptedData other = (EncryptedData) o;
        return Arrays.equals(encryptedBytes, other.encryptedBytes) &&
Arrays.equals(initialisationVector, other.initialisationVector);
    }

    @Override
    public int hashCode() {
        return Objects.hash(Arrays.hashCode(encryptedBytes),
Arrays.hashCode(initialisationVector));
    }

    @Override
    public String toString() {
        return "EncryptedData [initialisationVector=" + Arrays.toString(initialisationVector)
            + ", encryptedPrivateKey=" + Arrays.toString(encryptedBytes) + "]";
    }

public byte[] getInitialisationVector() {
return initialisationVector;
}

public byte[] getEncryptedBytes() {
return encryptedBytes;
}
```

```
public void setInitialisationVector(byte[] initialisationVector) {
this.initialisationVector = initialisationVector;
}

public void setEncryptedBytes(byte[] encryptedBytes) {
this.encryptedBytes = encryptedBytes;
}
}

25:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Exception\CryptoException.java
 */
package io.nuls.core.tools.crypto.Exception;

/**
 * author Facjas
 * date 2018/6/13.
 */
public class CryptoException extends Exception{
}

26:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Hex.java
 */
package io.nuls.core.tools.crypto;

/**
 * 16
 */
public class Hex {
/**
 * 16
 *
 * @param src
 * @return String
 */
public static String encode(byte[] src) {
StringBuffer strbuf = new StringBuffer(src.length * 2);
int i;

for (i = 0; i < src.length; i++) {
```

```
if (((int) src[i] & 0xff) < 0x10) {
                strbuf.append("0");
            }

strbuf.append(Long.toString((int) src[i] & 0xff, 16));
}

return strbuf.toString();
}

/**
 * 16
 *
 * @param hexString
 * @return byte[]
 */
public static byte[] decode(String hexString) {
byte[] bts = new byte[hexString.length() / 2];
for (int i = 0; i < bts.length; i++) {
bts[i] = (byte) Integer.parseInt(hexString.substring(2 * i, 2 * i + 2), 16);
}
return bts;
}

}

27:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Ints.java
 */
package io.nuls.core.tools.crypto;

public class Ints {

public static int fromBytes(byte b1, byte b2, byte b3, byte b4) {
return b1 << 24 | (b2 & 0xFF) << 16 | (b3 & 0xFF) << 8 | (b4 & 0xFF);
}
}

28:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\LinuxSecureRandom.java
 */
package io.nuls.core.tools.crypto;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.*;
import java.security.Provider;
import java.security.SecureRandomSpi;
import java.security.Security;

/**
 * A SecureRandom implementation that is able to override the standard JVM provided
implementation, and which simply
 * serves random numbers by reading /dev/U_RANDOM. That is, it delegates to the kernel on
UNIX systems and is unusable on
 * other platforms. Attempts to manually set the seed are ignored. There is no difference between
seed bytes and
 * non-seed bytes, they are all from the same source.
 */
public class LinuxSecureRandom extends SecureRandomSpi {
private static final long serialVersionUID = -1223766068997859131L;

private static final FileInputStream U_RANDOM;

    private static class LinuxSecureRandomProvider extends Provider {
private static final long serialVersionUID = 2559382307871869793L;

public LinuxSecureRandomProvider() {
            super("LinuxSecureRandom", 1.0, "A Linux specific random number provider that uses
/dev/U_RANDOM");
            put("SecureRandom.LinuxSecureRandom", LinuxSecureRandom.class.getName());
        }
    }

    private static final Logger log = LoggerFactory.getLogger(LinuxSecureRandom.class);

    static {
        try {
            File file = new File("/dev/U_RANDOM");
            // This stream is deliberately leaked.
            U_RANDOM = new FileInputStream(file);
            if (U_RANDOM.read() == -1) {
                throw new RuntimeException("/dev/U_RANDOM not readable?");
```

```
        }
        // Now override the default SecureRandom implementation with this one.
        int position = Security.insertProviderAt(new LinuxSecureRandomProvider(), 1);

        if (position != -1) {
            log.info("Secure randomness will be read from {} only.", file);
        } else {
            log.info("Randomness is already secure.");
        }
    } catch (FileNotFoundException e) {
        // Should never happen.
        log.error("/dev/U_RANDOM does not appear to exist or is not openable");
        throw new RuntimeException(e);
    } catch (IOException e) {
        log.error("/dev/U_RANDOM does not appear to be readable");
        throw new RuntimeException(e);
    }
}

private final DataInputStream dis;

public LinuxSecureRandom() {
    // DataInputStream is not thread safe, so each random object has its own.
    dis = new DataInputStream(U_RANDOM);
}

@Override
protected void engineSetSeed(byte[] bytes) {
    // Ignore.
}

@Override
protected void engineNextBytes(byte[] bytes) {
    try {
        dis.readFully(bytes); // This will block until all the bytes can be read.
    } catch (IOException e) {
        throw new RuntimeException(e); // Fatal error. Do not attempt to recover from this.
    }
}

@Override
protected byte[] engineGenerateSeed(int i) {
```

```
        byte[] bits = new byte[i];
        engineNextBytes(bits);
        return bits;
    }
}

29:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Sha256Hash.java
 */



package io.nuls.core.tools.crypto;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.Serializable;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;

/**
 * A Sha256Hash just wraps a byte[] so that equals and hashcode work correctly, allowing it to be
used as keys in a
 * map. It also checks that the length is correct and provides a bit more type safety.
 */
public class Sha256Hash implements Serializable, Comparable<Sha256Hash> {
private static final long serialVersionUID = 3986948258337764647L;

public static final int LENGTH = 32; // bytes
    public static final Sha256Hash ZERO_HASH = wrap(new byte[LENGTH]);

    private final byte[] bytes;

    @Deprecated
    public Sha256Hash(byte[] rawHashBytes) {
        Util.checkState(rawHashBytes.length == LENGTH);
        this.bytes = rawHashBytes;
    }
```

```
@Deprecated
public Sha256Hash(String hexString) {
Util.checkState(hexString.length() == LENGTH * 2);
    this.bytes = Hex.decode(hexString);
}

/**
 * Creates a new instance that wraps the given hash value.
 *
 * @param rawHashBytes the raw hash bytes to wrap
 * @return a new instance
 * @throws IllegalArgumentException if the given array length is not exactly 32
 */
public static Sha256Hash wrap(byte[] rawHashBytes) {
    return new Sha256Hash(rawHashBytes);
}

/**
 * Creates a new instance that wraps the given hash value (represented as a hex string).
 *
 * @param hexString a hash value represented as a hex string
 * @return a new instance
 * @throws IllegalArgumentException if the given string is not a valid
 *         hex string, or if it does not represent exactly 32 bytes
 */
public static Sha256Hash wrap(String hexString) {
    return wrap(Hex.decode(hexString));
}

/**
 * Creates a new instance that wraps the given hash value, but with byte order reversed.
 *
 * @param rawHashBytes the raw hash bytes to wrap
 * @return a new instance
 * @throws IllegalArgumentException if the given array length is not exactly 32
 */
public static Sha256Hash wrapReversed(byte[] rawHashBytes) {
    return wrap(Util.reverseBytes(rawHashBytes));
}

@Deprecated
public static Sha256Hash create(byte[] contents) {
```

```
        return of(contents);
    }

    /**
     * Creates a new instance containing the calculated (one-time) hash of the given bytes.
     *
     * @param contents the bytes on which the hash value is calculated
     * @return a new instance containing the calculated (one-time) hash
     */
    public static Sha256Hash of(byte[] contents) {
        return wrap(hash(contents));
    }


    @Deprecated
    public static Sha256Hash createDouble(byte[] contents) {
        return twiceOf(contents);
    }

    /**
     * Creates a new instance containing the hash of the calculated hash of the given bytes.
     *
     * @param contents the bytes on which the hash value is calculated
     * @return a new instance containing the calculated (two-time) hash
     */
    public static Sha256Hash twiceOf(byte[] contents) {
        return wrap(hashTwice(contents));
    }

    /**
     * Creates a new instance containing the calculated (one-time) hash of the given file's contents.
     *
     * The file contents are read fully into memory, so this method should only be used with small
files.
     *
     * @param file the file on which the hash value is calculated
     * @return a new instance containing the calculated (one-time) hash
     * @throws IOException if an error occurs while reading the file
     */
    public static Sha256Hash of(File file) throws IOException {
        FileInputStream in = new FileInputStream(file);
        try {
```

```
        return of(ByteStreams.toByteArray(in));
    } finally {
        in.close();
    }
}

/**
 * Returns a new SHA-256 MessageDigest instance.
 *
 * This is a convenience method which wraps the checked
 * exception that can never occur with a RuntimeException.
 *
 * @return a new SHA-256 MessageDigest instance
 */
public static MessageDigest newDigest() {
    try {
        return MessageDigest.getInstance("SHA-256");
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);  // Can't happen.
    }
}

/**
 * Calculates the SHA-256 hash of the given bytes.
 *
 * @param input the bytes to hash
 * @return the hash (in big-endian order)
 */
public static byte[] hash(byte[] input) {
    return hash(input, 0, input.length);
}

/**
 * Calculates the SHA-256 hash of the given byte range.
 *
 * @param input the array containing the bytes to hash
 * @param offset the offset within the array of the bytes to hash
 * @param length the number of bytes to hash
 * @return the hash (in big-endian order)
 */
public static byte[] hash(byte[] input, int offset, int length) {
    MessageDigest digest = newDigest();
```

```
    digest.update(input, offset, length);
    return digest.digest();
}

/**
 * Calculates the SHA-256 hash of the given bytes,
 * and then hashes the resulting hash again.
 *
 * @param input the bytes to hash
 * @return the double-hash (in big-endian order)
 */
public static byte[] hashTwice(byte[] input) {
    return hashTwice(input, 0, input.length);
}

/**
 * Calculates the SHA-256 hash of the given byte range,
 * and then hashes the resulting hash again.
 *
 * @param input the array containing the bytes to hash
 * @param offset the offset within the array of the bytes to hash
 * @param length the number of bytes to hash
 * @return the double-hash (in big-endian order)
 */
public static byte[] hashTwice(byte[] input, int offset, int length) {
    MessageDigest digest = newDigest();
    digest.update(input, offset, length);
    return digest.digest(digest.digest());
}

public static byte[] hashTwice(byte[] input1, int offset1, int length1,
                               byte[] input2, int offset2, int length2) {
    MessageDigest digest = newDigest();
    digest.update(input1, offset1, length1);
    digest.update(input2, offset2, length2);
    return digest.digest(digest.digest());
}

@Override
public boolean equals(Object o) {
    if (this == o) {
        return true;
```

```
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        return Arrays.equals(bytes, ((Sha256Hash)o).bytes);
    }

    /**
     * Returns the last four bytes of the wrapped hash. This should be unique enough to be a suitable hash code even for
     * blocks, where the goal is to try and get the first bytes to be zeros (i.e. the value as a big integer lower
     * than the target value).
     * @return int
     */
    @Override
    public int hashCode() {
        // Use the last 4 bytes, not the first 4 which are often zeros in Bitcoin.
        return Ints.fromBytes(bytes[LENGTH - 4], bytes[LENGTH - 3], bytes[LENGTH - 2], bytes[LENGTH - 1]);
    }

    @Override
    public String toString() {
        return Hex.encode(bytes);
    }

    /**
     * Returns the bytes interpreted as a positive integer.
     * @return BigInteger
     */
    public BigInteger toBigInteger() {
        return new BigInteger(1, bytes);
    }

    /**
     * Returns the internal byte array, without defensively copying. Therefore do NOT modify the returned array.
     * @return byte[]
     */
    public byte[] getBytes() {
        return bytes;
```

```
    }

    /**
     * Returns a reversed copy of the internal byte array.
     * @return byte[]
     */
    public byte[] getReversedBytes() {
        return Util.reverseBytes(bytes);
    }

    @Override
    public int compareTo(final Sha256Hash other) {
        for (int i = LENGTH - 1; i >= 0; i--) {
            final int thisByte = this.bytes[i] & 0xff;
            final int otherByte = other.bytes[i] & 0xff;
            if (thisByte > otherByte) {
                return 1;
            }
            if (thisByte < otherByte) {
                return -1;
            }
        }
        return 0;
    }
}

30:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\SM2.java
 */
package io.nuls.core.tools.crypto;

import org.bouncycastle.crypto.generators.ECKeyPairGenerator;
import org.bouncycastle.crypto.params.ECDomainParameters;
import org.bouncycastle.crypto.params.ECKeyGenerationParameters;
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.math.ec.ECFieldElement;
import org.bouncycastle.math.ec.ECFieldElement.Fp;
import org.bouncycastle.math.ec.ECPoint;

import java.math.BigInteger;
import java.security.SecureRandom;
```

```
/**
 * Created by facjas on 2017/11/20.
 */
public class SM2 {
    public static final String[] ECC_PARAM = {
            "8542D69E4C044F18E8B92435BF6FF7DE457283915C45517D722EDB8B08F1DFC3",
            "787968B4FA32C3FD2417842E73BBFEFF2F3C848B6831D7E0EC65228B3937E498",
            "63E4C6D3B23B0C849CF84241484BFE48F61D59A5B16BA06E6E12D1DA27C5249A",
            "8542D69E4C044F18E8B92435BF6FF7DD297720630485628D5AE74EE7C32E79B7",
"421DEBD61B62EAB6746434EBC3CC315E32220B3BADD50BDC4C4E6C147FEDD43D",
 "0680512BCBB42C07D47349D2153B70C4E5D7FDFCBFA36EA1A85841B9E46E09A2"
    };

    public static SM2 Instance() {
        return new SM2();
    }

    public final BigInteger ecc_p;
    public final BigInteger ecc_a;
    public final BigInteger ecc_b;
    public final BigInteger ecc_n;
    public final BigInteger ecc_gx;
    public final BigInteger ecc_gy;
    public final ECCurve ecc_curve;
    public final ECPoint ecc_point_g;
    public final ECDomainParameters ecc_bc_spec;
    public final ECKeyPairGenerator ecc_key_pair_generator;
    public final ECFieldElement ecc_gx_fieldelement;
    public final ECFieldElement ecc_gy_fieldelement;

    public SM2() {
        this.ecc_p = new BigInteger(ECC_PARAM[0], 16);
        this.ecc_a = new BigInteger(ECC_PARAM[1], 16);
        this.ecc_b = new BigInteger(ECC_PARAM[2], 16);
        this.ecc_n = new BigInteger(ECC_PARAM[3], 16);
        this.ecc_gx = new BigInteger(ECC_PARAM[4], 16);
        this.ecc_gy = new BigInteger(ECC_PARAM[5], 16);

        this.ecc_gx_fieldelement = new Fp(this.ecc_p, this.ecc_gx);
        this.ecc_gy_fieldelement = new Fp(this.ecc_p, this.ecc_gy);

        this.ecc_curve = new ECCurve.Fp(this.ecc_p, this.ecc_a, this.ecc_b);
```

```
        this.ecc_point_g = new ECPoint.Fp(this.ecc_curve, this.ecc_gx_fieldelement,
this.ecc_gy_fieldelement);

        this.ecc_bc_spec = new ECDomainParameters(this.ecc_curve, this.ecc_point_g, this.ecc_n);

        ECKeyGenerationParameters ecc_ecgenparam;
        ecc_ecgenparam = new ECKeyGenerationParameters(this.ecc_bc_spec, new
SecureRandom());

        this.ecc_key_pair_generator = new ECKeyPairGenerator();
        this.ecc_key_pair_generator.init(ecc_ecgenparam);
    }

    public byte[] sm2GetZ(byte[] userId, ECPoint userKey) {
        SM3Digest sm3 = new SM3Digest();

        int len = userId.length * 8;
        sm3.update((byte) (len >> 8 & 0xFF));
        sm3.update((byte) (len & 0xFF));
        sm3.update(userId, 0, userId.length);

        byte[] p = Util.byteConvert32Bytes(ecc_a);
        sm3.update(p, 0, p.length);

        p = Util.byteConvert32Bytes(ecc_b);
        sm3.update(p, 0, p.length);

        p = Util.byteConvert32Bytes(ecc_gx);
        sm3.update(p, 0, p.length);

        p = Util.byteConvert32Bytes(ecc_gy);
        sm3.update(p, 0, p.length);

        p = Util.byteConvert32Bytes(userKey.getX().toBigInteger());
        sm3.update(p, 0, p.length);

        p = Util.byteConvert32Bytes(userKey.getY().toBigInteger());
        sm3.update(p, 0, p.length);

        byte[] md = new byte[sm3.getDigestSize()];
        sm3.doFinal(md, 0);
        return md;
```

```
    }

    public void sm2Sign(byte[] md, BigInteger userD, ECPoint userKey, SM2Result sm2Result) {
        BigInteger e = new BigInteger(1, md);
        BigInteger k = null;
        ECPoint kp = null;
        BigInteger r = null;
        BigInteger s = null;
        do {
            do {
                String kS =
"6CB28D99385C175C94F94E934817663FC176D925DD72B727260DBAAE1FB2F96F";
                k = new BigInteger(kS, 16);
                kp = this.ecc_point_g.multiply(k);

                // r
                r = e.add(kp.getX().toBigInteger());
                r = r.mod(ecc_n);
            } while (r.equals(BigInteger.ZERO) || r.add(k).equals(ecc_n));

            // (1 + dA)~-1
            BigInteger da_1 = userD.add(BigInteger.ONE);
            da_1 = da_1.modInverse(ecc_n);

            // s
            s = r.multiply(userD);
            s = k.subtract(s).mod(ecc_n);
            s = da_1.multiply(s).mod(ecc_n);
        } while (s.equals(BigInteger.ZERO));

        sm2Result.r = r;
        sm2Result.s = s;
    }

    public void sm2Verify(byte md[], ECPoint userKey, BigInteger r, BigInteger s, SM2Result
sm2Result) {
        sm2Result.R = null;
        BigInteger e = new BigInteger(1, md);
        BigInteger t = r.add(s).mod(ecc_n);
        if (t.equals(BigInteger.ZERO)) {
            return;
        } else {
```

```
            ECPoint x1y1 = ecc_point_g.multiply(sm2Result.s);

            x1y1 = x1y1.add(userKey.multiply(t));
            sm2Result.R = e.add(x1y1.getX().toBigInteger()).mod(ecc_n);
            return;
        }
    }
}
```

31:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\crypto\SM2Result.java

```
 */
package io.nuls.core.tools.crypto;

import org.bouncycastle.math.ec.ECPoint;

import java.math.BigInteger;

/**
 * Created by facjas on 2017/11/20.
 */
public class SM2Result {
    public SM2Result() {
    }

    public BigInteger r;
    public BigInteger s;
    public BigInteger R;

    public byte[] sa;
    public byte[] sb;
    public byte[] s1;
    public byte[] s2;

    public ECPoint keyra;
    public ECPoint keyrb;
}
```

32:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\crypto\SM2Utils.java

```
 */
package io.nuls.core.tools.crypto;
```

```
import org.bouncycastle.asn1.*;
import org.bouncycastle.math.ec.ECPoint;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Enumeration;

/**
 * Created by facjas on 2017/11/20.
 */
public class SM2Utils {
    public static byte[] encrypt(byte[] publicKey, byte[] data) throws IOException {
        if (publicKey == null || publicKey.length == 0) {
            return null;
        }

        if (data == null || data.length == 0) {
            return null;
        }

        byte[] source = new byte[data.length];
        System.arraycopy(data, 0, source, 0, data.length);

        Cipher cipher = new Cipher();
        SM2 sm2 = SM2.Instance();
        ECPoint userKey = sm2.ecc_curve.decodePoint(publicKey);

        ECPoint c1 = cipher.initEnc(sm2, userKey);
        cipher.encrypt(source);
        byte[] c3 = new byte[32];
        cipher.dofinal(c3);

        DERInteger x = new DERInteger(c1.getX().toBigInteger());
        DERInteger y = new DERInteger(c1.getY().toBigInteger());
        DEROctetString derDig = new DEROctetString(c3);
        DEROctetString derEnc = new DEROctetString(source);
        ASN1EncodableVector v = new ASN1EncodableVector();
        v.add(x);
        v.add(y);
```

```
    v.add(derDig);
    v.add(derEnc);
    DERSequence seq = new DERSequence(v);
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    DEROutputStream dos = new DEROutputStream(bos);
    dos.writeObject(seq);
    return bos.toByteArray();
}

public static byte[] decrypt(byte[] privateKey, byte[] encryptedData) throws IOException {
    if (privateKey == null || privateKey.length == 0) {
        return null;
    }

    if (encryptedData == null || encryptedData.length == 0) {
        return null;
    }

    byte[] enc = new byte[encryptedData.length];
    System.arraycopy(encryptedData, 0, enc, 0, encryptedData.length);

    SM2 sm2 = SM2.Instance();
    BigInteger userD = new BigInteger(1, privateKey);

    ByteArrayInputStream bis = new ByteArrayInputStream(enc);
    ASN1InputStream dis = new ASN1InputStream(bis);
    DERObject derObj = dis.readObject();
    ASN1Sequence asn1 = (ASN1Sequence) derObj;
    DERInteger x = (DERInteger) asn1.getObjectAt(0);
    DERInteger y = (DERInteger) asn1.getObjectAt(1);
    ECPoint c1 = sm2.ecc_curve.createPoint(x.getValue(), y.getValue(), true);

    Cipher cipher = new Cipher();
    cipher.initDec(userD, c1);
    DEROctetString data = (DEROctetString) asn1.getObjectAt(3);
    enc = data.getOctets();
    cipher.decrypt(enc);
    byte[] c3 = new byte[32];
    cipher.dofinal(c3);
    return enc;
}
```

```
    public static byte[] sign(byte[] userId, byte[] privateKey, byte[] sourceData) throws IOException {
        if (privateKey == null || privateKey.length == 0) {
            return null;
        }

        if (sourceData == null || sourceData.length == 0) {
            return null;
        }

        SM2 sm2 = SM2.Instance();
        BigInteger userD = new BigInteger(privateKey);



        ECPoint userKey = sm2.ecc_point_g.multiply(userD);

        SM3Digest sm3 = new SM3Digest();
        byte[] z = sm2.sm2GetZ(userId, userKey);

        sm3.update(z, 0, z.length);
        sm3.update(sourceData, 0, sourceData.length);
        byte[] md = new byte[32];
        sm3.doFinal(md, 0);



        SM2Result sm2Result = new SM2Result();
        sm2.sm2Sign(md, userD, userKey, sm2Result);

        DERInteger dR = new DERInteger(sm2Result.r);
        DERInteger dS = new DERInteger(sm2Result.s);
        ASN1EncodableVector v2 = new ASN1EncodableVector();
        v2.add(dR);
        v2.add(dS);
        DERObject sign = new DERSequence(v2);
        byte[] signdata = sign.getDEREncoded();
        return signdata;
    }

    public static boolean verifySign(byte[] userId, byte[] publicKey, byte[] sourceData, byte[]
signData) throws IOException {
        if (publicKey == null || publicKey.length == 0) {
            return false;
        }
```

```
        if (sourceData == null || sourceData.length == 0) {
            return false;
        }

        SM2 sm2 = SM2.Instance();
        ECPoint userKey = sm2.ecc_curve.decodePoint(publicKey);

        SM3Digest sm3 = new SM3Digest();
        byte[] z = sm2.sm2GetZ(userId, userKey);
        sm3.update(z, 0, z.length);
        sm3.update(sourceData, 0, sourceData.length);
        byte[] md = new byte[32];
        sm3.doFinal(md, 0);

        ByteArrayInputStream bis = new ByteArrayInputStream(signData);
        ASN1InputStream dis = new ASN1InputStream(bis);
        DERObject derObj = dis.readObject();
        Enumeration<DERInteger> e = ((ASN1Sequence) derObj).getObjects();
        BigInteger r = ((DERInteger) e.nextElement()).getValue();
        BigInteger s = ((DERInteger) e.nextElement()).getValue();
        SM2Result sm2Result = new SM2Result();
        sm2Result.r = r;
        sm2Result.s = s;
        sm2.sm2Verify(md, userKey, sm2Result.r, sm2Result.s, sm2Result);
        return sm2Result.r.equals(sm2Result.R);
    }

}

33:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\SM3.java
 */
package io.nuls.core.tools.crypto;

/**
 *
 * @author facjas
 */

public class SM3 {
    public static final byte[] IV = {0x73, (byte) 0x80, 0x16, 0x6f, 0x49,
```

```
        0x14, (byte) 0xb2, (byte) 0xb9, 0x17, 0x24, 0x42, (byte) 0xd7,
        (byte) 0xda, (byte) 0x8a, 0x06, 0x00, (byte) 0xa9, 0x6f, 0x30,
        (byte) 0xbc, (byte) 0x16, 0x31, 0x38, (byte) 0xaa, (byte) 0xe3,
        (byte) 0x8d, (byte) 0xee, 0x4d, (byte) 0xb0, (byte) 0xfb, 0x0e,
        0x4e};

public static int[] Tj = new int[64];

static {
    for (int i = 0; i < 16; i++) {
        Tj[i] = 0x79cc4519;
    }

    for (int i = 16; i < 64; i++) {
        Tj[i] = 0x7a879d8a;
    }
}

public static byte[] cf(byte[] v1, byte[] b1) {
    int[] v, b;
    v = convert(v1);
    b = convert(b1);
    return convert(cf(v, b));
}

private static int[] convert(byte[] arr) {
    int[] out = new int[arr.length / 4];
    byte[] tmp = new byte[4];
    for (int i = 0; i < arr.length; i += 4) {
        System.arraycopy(arr, i, tmp, 0, 4);
        out[i / 4] = bigEndianByteToInt(tmp);
    }
    return out;
}

private static byte[] convert(int[] arr) {
    byte[] out = new byte[arr.length * 4];
    byte[] tmp = null;
    for (int i = 0; i < arr.length; i++) {
        tmp = bigEndianIntToByte(arr[i]);
        System.arraycopy(tmp, 0, out, i * 4, 4);
    }
```

```
    return out;
}

public static int[] cf(int[] vArray, int[] bArray) {
    int a, b, c, d, e, f, g, h;
    int ss1, ss2, tt1, tt2;
    a = vArray[0];
    b = vArray[1];
    c = vArray[2];
    d = vArray[3];
    e = vArray[4];
    f = vArray[5];
    g = vArray[6];
    h = vArray[7];

    int[][] arr = expand(bArray);
    int[] w = arr[0];
    int[] w1 = arr[1];

    for (int j = 0; j < 64; j++) {
        ss1 = (bitCycleLeft(a, 12) + e + bitCycleLeft(Tj[j], j));
        ss1 = bitCycleLeft(ss1, 7);
        ss2 = ss1 ^ bitCycleLeft(a, 12);
        tt1 = ffj(a, b, c, j) + d + ss2 + w1[j];
        tt2 = ggj(e, f, g, j) + h + ss1 + w[j];
        d = c;
        c = bitCycleLeft(b, 9);
        b = a;
        a = tt1;
        h = g;
        g = bitCycleLeft(f, 19);
        f = e;
        e = p0(tt2);
    }


    int[] out = new int[8];
    out[0] = a ^ vArray[0];
    out[1] = b ^ vArray[1];
    out[2] = c ^ vArray[2];
    out[3] = d ^ vArray[3];
    out[4] = e ^ vArray[4];
```

```
    out[5] = f ^ vArray[5];
    out[6] = g ^ vArray[6];
    out[7] = h ^ vArray[7];

    return out;
}

private static int[][] expand(int[] bArray) {
    int[] wArray = new int[68];
    int[] w1Array = new int[64];
    for (int i = 0; i < bArray.length; i++) {
        wArray[i] = bArray[i];
    }

    for (int i = 16; i < 68; i++) {
        wArray[i] = p1(wArray[i - 16] ^ wArray[i - 9] ^ bitCycleLeft(wArray[i - 3], 15))
                ^ bitCycleLeft(wArray[i - 13], 7) ^ wArray[i - 6];
    }

    for (int i = 0; i < 64; i++) {
        w1Array[i] = wArray[i] ^ wArray[i + 4];
    }

    int arr[][] = new int[][]{wArray, w1Array};
    return arr;
}

private static byte[] bigEndianIntToByte(int num) {
    return back(Util.intToBytes(num));
}

private static int bigEndianByteToInt(byte[] bytes) {
    return Util.byteToInt(back(bytes));
}

private static int ffj(int x, int y, int z, int j) {
    if (j >= 0 && j <= 15) {
        return ff1j(x, y, z);
    } else {
        return ff2j(x, y, z);
    }
}
```

```
private static int ggj(int x, int y, int z, int j) {
    if (j >= 0 && j <= 15) {
        return gg1j(x, y, z);
    } else {
        return gg2j(x, y, z);
    }
}

private static int ff1j(int x, int y, int z) {
    int tmp = x ^ y ^ z;
    return tmp;
}

private static int ff2j(int x, int y, int z) {
    int tmp = ((x & y) | (x & z) | (y & z));
    return tmp;
}

private static int gg1j(int x, int y, int z) {
    int tmp = x ^ y ^ z;
    return tmp;
}

private static int gg2j(int x, int y, int z) {
    int tmp = (x & y) | (~x & z);
    return tmp;
}

private static int p0(int x) {
    int y = rotateLeft(x, 9);
    y = bitCycleLeft(x, 9);
    int z = rotateLeft(x, 17);
    z = bitCycleLeft(x, 17);
    int t = x ^ y ^ z;
    return t;
}

private static int p1(int x) {
    int t = x ^ bitCycleLeft(x, 15) ^ bitCycleLeft(x, 23);
    return t;
}
```

```
public static byte[] padding(byte[] in, int bLen) {
    int k = 448 - (8 * in.length + 1) % 512;
    if (k < 0) {
        k = 960 - (8 * in.length + 1) % 512;
    }
    k += 1;
    byte[] padd = new byte[k / 8];
    padd[0] = (byte) 0x80;
    long n = in.length * 8 + bLen * 512;
    byte[] out = new byte[in.length + k / 8 + 64 / 8];
    int pos = 0;
    System.arraycopy(in, 0, out, 0, in.length);
    pos += in.length;
    System.arraycopy(padd, 0, out, pos, padd.length);
    pos += padd.length;
    byte[] tmp = back(Util.longToBytes(n));
    System.arraycopy(tmp, 0, out, pos, tmp.length);
    return out;
}

private static byte[] back(byte[] in) {
    byte[] out = new byte[in.length];
    for (int i = 0; i < out.length; i++) {
        out[i] = in[out.length - i - 1];
    }
    return out;
}

public static int rotateLeft(int x, int n) {
    return (x << n) | (x >> (32 - n));
}

private static int bitCycleLeft(int n, int bitLen) {
    bitLen %= 32;
    byte[] tmp = bigEndianIntToByte(n);
    int byteLen = bitLen / 8;
    int len = bitLen % 8;
    if (byteLen > 0) {
        tmp = byteCycleLeft(tmp, byteLen);
    }
```

```
        if (len > 0) {
            tmp = bitSmall8CycleLeft(tmp, len);
        }
        return bigEndianByteToInt(tmp);
    }

    private static byte[] bitSmall8CycleLeft(byte[] in, int len) {
        byte[] tmp = new byte[in.length];
        int t1, t2, t3;
        for (int i = 0; i < tmp.length; i++) {
            t1 = (byte) ((in[i] & 0x000000ff) << len);
            t2 = (byte) ((in[(i + 1) % tmp.length] & 0x000000ff) >> (8 - len));
            t3 = (byte) (t1 | t2);
            tmp[i] = (byte) t3;
        }
        return tmp;
    }

    private static byte[] byteCycleLeft(byte[] in, int byteLen) {
        byte[] tmp = new byte[in.length];
        System.arraycopy(in, byteLen, tmp, 0, in.length - byteLen);
        System.arraycopy(in, 0, tmp, in.length - byteLen, byteLen);
        return tmp;
    }
}

34:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\SM3Digest.java
 */
package io.nuls.core.tools.crypto;

/**
 * Created by facjas on 2017/11/20.
 */
public class SM3Digest {

    private static final int BYTE_LENGTH = 32;
    private static final int BLOCK_LENGTH = 64;
    private static final int BUFFER_LENGTH = BLOCK_LENGTH * 1;
    private byte[] xBuf = new byte[BUFFER_LENGTH];
    private int xBufOff;
```

```
private byte[] V = SM3.IV.clone();

private int cntBlock = 0;

public SM3Digest() {
}

public SM3Digest(SM3Digest t) {
    System.arraycopy(t.xBuf, 0, this.xBuf, 0, t.xBuf.length);
    this.xBufOff = t.xBufOff;
    System.arraycopy(t.V, 0, this.V, 0, t.V.length);
}

public int doFinal(byte[] out, int outOff) {
    byte[] tmp = doFinal();
    System.arraycopy(tmp, 0, out, 0, tmp.length);
    return BYTE_LENGTH;
}

public void reset() {
    xBufOff = 0;
    cntBlock = 0;
    V = SM3.IV.clone();
}

public void update(byte[] in, int inOff, int len) {
    int partLen = BUFFER_LENGTH - xBufOff;
    int inputLen = len;
    int dPos = inOff;
    if (partLen < inputLen) {
        System.arraycopy(in, dPos, xBuf, xBufOff, partLen);
        inputLen -= partLen;
        dPos += partLen;
        doUpdate();
        while (inputLen > BUFFER_LENGTH) {
            System.arraycopy(in, dPos, xBuf, 0, BUFFER_LENGTH);
            inputLen -= BUFFER_LENGTH;
            dPos += BUFFER_LENGTH;
            doUpdate();
        }
    }
```

```
        System.arraycopy(in, dPos, xBuf, xBufOff, inputLen);
        xBufOff += inputLen;
    }

    private void doUpdate() {
        byte[] B = new byte[BLOCK_LENGTH];
        for (int i = 0; i < BUFFER_LENGTH; i += BLOCK_LENGTH) {
            System.arraycopy(xBuf, i, B, 0, B.length);
            doHash(B);
        }
        xBufOff = 0;
    }

    private void doHash(byte[] B) {
        byte[] tmp = SM3.cf(V, B);
        System.arraycopy(tmp, 0, V, 0, V.length);
        cntBlock++;
    }

    private byte[] doFinal() {
        byte[] B = new byte[BLOCK_LENGTH];
        byte[] buffer = new byte[xBufOff];
        System.arraycopy(xBuf, 0, buffer, 0, buffer.length);
        byte[] tmp = SM3.padding(buffer, cntBlock);
        for (int i = 0; i < tmp.length; i += BLOCK_LENGTH) {
            System.arraycopy(tmp, i, B, 0, B.length);
            doHash(B);
        }
        return V;
    }

    public void update(byte in) {
        byte[] buffer = new byte[]{in};
        update(buffer, 0, 1);
    }

    public int getDigestSize() {
        return BYTE_LENGTH;
    }
}
```

```
module\tools\src\main\java\io\nuls\core\tools\crypto\SM4.java
 */
package io.nuls.core.tools.crypto;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

/**
 * Created by facjas on 2017/11/20.
 */

public class SM4 {
    public static final int SM4_ENCRYPT = 1;

    public static final int SM4_DECRYPT = 0;

    private long GET_ULONG_BE(byte[] b, int i) {
        long n = (long) (b[i] & 0xff) << 24 | (long) ((b[i + 1] & 0xff) << 16) | (long) ((b[i + 2] & 0xff) << 8)
| (long) (b[i + 3] & 0xff) & 0xffffffffL;
        return n;
    }

    private void PUT_ULONG_BE(long n, byte[] b, int i) {
        b[i] = (byte) (int) (0xFF & n >> 24);
        b[i + 1] = (byte) (int) (0xFF & n >> 16);
        b[i + 2] = (byte) (int) (0xFF & n >> 8);
        b[i + 3] = (byte) (int) (0xFF & n);
    }

    private long SHL(long x, int n) {
        return (x & 0xFFFFFFFF) << n;
    }

    private long ROTL(long x, int n) {
        return SHL(x, n) | x >> (32 - n);
    }

    private void SWAP(long[] sk, int i) {
        long t = sk[i];
        sk[i] = sk[(31 - i)];
        sk[(31 - i)] = t;
    }
```

```
public static final byte[] SBOX_TABLE = {
        (byte) 0xd6, (byte) 0x90, (byte) 0xe9, (byte) 0xfe,
        (byte) 0xcc, (byte) 0xe1, 0x3d, (byte) 0xb7, 0x16, (byte) 0xb6,
        0x14, (byte) 0xc2, 0x28, (byte) 0xfb, 0x2c, 0x05, 0x2b, 0x67,
        (byte) 0x9a, 0x76, 0x2a, (byte) 0xbe, 0x04, (byte) 0xc3,
        (byte) 0xaa, 0x44, 0x13, 0x26, 0x49, (byte) 0x86, 0x06,
        (byte) 0x99, (byte) 0x9c, 0x42, 0x50, (byte) 0xf4, (byte) 0x91,
        (byte) 0xef, (byte) 0x98, 0x7a, 0x33, 0x54, 0x0b, 0x43,
        (byte) 0xed, (byte) 0xcf, (byte) 0xac, 0x62, (byte) 0xe4,
        (byte) 0xb3, 0x1c, (byte) 0xa9, (byte) 0xc9, 0x08, (byte) 0xe8,
        (byte) 0x95, (byte) 0x80, (byte) 0xdf, (byte) 0x94, (byte) 0xfa,
        0x75, (byte) 0x8f, 0x3f, (byte) 0xa6, 0x47, 0x07, (byte) 0xa7,
        (byte) 0xfc, (byte) 0xf3, 0x73, 0x17, (byte) 0xba, (byte) 0x83,
        0x59, 0x3c, 0x19, (byte) 0xe6, (byte) 0x85, 0x4f, (byte) 0xa8,
        0x68, 0x6b, (byte) 0x81, (byte) 0xb2, 0x71, 0x64, (byte) 0xda,
        (byte) 0x8b, (byte) 0xf8, (byte) 0xeb, 0x0f, 0x4b, 0x70, 0x56,
        (byte) 0x9d, 0x35, 0x1e, 0x24, 0x0e, 0x5e, 0x63, 0x58, (byte) 0xd1,
        (byte) 0xa2, 0x25, 0x22, 0x7c, 0x3b, 0x01, 0x21, 0x78, (byte) 0x87,
        (byte) 0xd4, 0x00, 0x46, 0x57, (byte) 0x9f, (byte) 0xd3, 0x27,
        0x52, 0x4c, 0x36, 0x02, (byte) 0xe7, (byte) 0xa0, (byte) 0xc4,
        (byte) 0xc8, (byte) 0x9e, (byte) 0xea, (byte) 0xbf, (byte) 0x8a,
        (byte) 0xd2, 0x40, (byte) 0xc7, 0x38, (byte) 0xb5, (byte) 0xa3,
        (byte) 0xf7, (byte) 0xf2, (byte) 0xce, (byte) 0xf9, 0x61, 0x15,
        (byte) 0xa1, (byte) 0xe0, (byte) 0xae, 0x5d, (byte) 0xa4,
        (byte) 0x9b, 0x34, 0x1a, 0x55, (byte) 0xad, (byte) 0x93, 0x32,
        0x30, (byte) 0xf5, (byte) 0x8c, (byte) 0xb1, (byte) 0xe3, 0x1d,
        (byte) 0xf6, (byte) 0xe2, 0x2e, (byte) 0x82, 0x66, (byte) 0xca,
        0x60, (byte) 0xc0, 0x29, 0x23, (byte) 0xab, 0x0d, 0x53, 0x4e, 0x6f,
        (byte) 0xd5, (byte) 0xdb, 0x37, 0x45, (byte) 0xde, (byte) 0xfd,
        (byte) 0x8e, 0x2f, 0x03, (byte) 0xff, 0x6a, 0x72, 0x6d, 0x6c, 0x5b,
        0x51, (byte) 0x8d, 0x1b, (byte) 0xaf, (byte) 0x92, (byte) 0xbb,
        (byte) 0xdd, (byte) 0xbc, 0x7f, 0x11, (byte) 0xd9, 0x5c, 0x41,
        0x1f, 0x10, 0x5a, (byte) 0xd8, 0x0a, (byte) 0xc1, 0x31,
        (byte) 0x88, (byte) 0xa5, (byte) 0xcd, 0x7b, (byte) 0xbd, 0x2d,
        0x74, (byte) 0xd0, 0x12, (byte) 0xb8, (byte) 0xe5, (byte) 0xb4,
        (byte) 0xb0, (byte) 0x89, 0x69, (byte) 0x97, 0x4a, 0x0c,
        (byte) 0x96, 0x77, 0x7e, 0x65, (byte) 0xb9, (byte) 0xf1, 0x09,
        (byte) 0xc5, 0x6e, (byte) 0xc6, (byte) 0x84, 0x18, (byte) 0xf0,
        0x7d, (byte) 0xec, 0x3a, (byte) 0xdc, 0x4d, 0x20, 0x79,
        (byte) 0xee, 0x5f, 0x3e, (byte) 0xd7, (byte) 0xcb, 0x39, 0x48};
```

```
public static final int[] FK = {0xa3b1bac6, 0x56aa3350, 0x677d9197, 0xb27022dc};

public static final int[] CK = {
        0x00070e15, 0x1c232a31, 0x383f464d, 0x545b6269,
        0x70777e85, 0x8c939aa1, 0xa8afb6bd, 0xc4cbd2d9,
        0xe0e7eef5, 0xfc030a11, 0x181f262d, 0x343b4249,
        0x50575e65, 0x6c737a81, 0x888f969d, 0xa4abb2b9,
        0xc0c7ced5, 0xdce3eaf1, 0xf8ff060d, 0x141b2229,
        0x30373e45, 0x4c535a61, 0x686f767d, 0x848b9299,
        0xa0a7aeb5, 0xbcc3cad1, 0xd8dfe6ed, 0xf4fb0209,
        0x10171e25, 0x2c333a41, 0x484f565d, 0x646b7279};

private byte sm4Sbox(byte inch) {
    int i = inch & 0xFF;
    byte retVal = SBOX_TABLE[i];
    return retVal;
}

private long sm4Lt(long ka) {
    long bb = 0L;
    long c = 0L;
    byte[] a = new byte[4];
    byte[] b = new byte[4];
    PUT_ULONG_BE(ka, a, 0);
    b[0] = sm4Sbox(a[0]);
    b[1] = sm4Sbox(a[1]);
    b[2] = sm4Sbox(a[2]);
    b[3] = sm4Sbox(a[3]);
    bb = GET_ULONG_BE(b, 0);
    c = bb ^ ROTL(bb, 2) ^ ROTL(bb, 10) ^ ROTL(bb, 18) ^ ROTL(bb, 24);
    return c;
}

private long sm4F(long x0, long x1, long x2, long x3, long rk) {
    return x0 ^ sm4Lt(x1 ^ x2 ^ x3 ^ rk);
}

private long sm4CalciRK(long ka) {
    long bb = 0L;
    long rk = 0L;
    byte[] a = new byte[4];
    byte[] b = new byte[4];
```

```
    PUT_ULONG_BE(ka, a, 0);
    b[0] = sm4Sbox(a[0]);
    b[1] = sm4Sbox(a[1]);
    b[2] = sm4Sbox(a[2]);
    b[3] = sm4Sbox(a[3]);
    bb = GET_ULONG_BE(b, 0);
    rk = bb ^ ROTL(bb, 13) ^ ROTL(bb, 23);
    return rk;
}

private void sm4_setkey(long[] SK, byte[] key) {
    long[] MK = new long[4];
    long[] k = new long[36];
    int i = 0;
    MK[0] = GET_ULONG_BE(key, 0);
    MK[1] = GET_ULONG_BE(key, 4);
    MK[2] = GET_ULONG_BE(key, 8);
    MK[3] = GET_ULONG_BE(key, 12);
    k[0] = MK[0] ^ (long) FK[0];
    k[1] = MK[1] ^ (long) FK[1];
    k[2] = MK[2] ^ (long) FK[2];
    k[3] = MK[3] ^ (long) FK[3];
    for (; i < 32; i++) {
        k[(i + 4)] = (k[i] ^ sm4CalciRK(k[(i + 1)] ^ k[(i + 2)] ^ k[(i + 3)] ^ (long) CK[i]));
        SK[i] = k[(i + 4)];
    }
}

private void sm4_one_round(long[] sk, byte[] input, byte[] output) {
    int i = 0;
    long[] ulbuf = new long[36];
    ulbuf[0] = GET_ULONG_BE(input, 0);
    ulbuf[1] = GET_ULONG_BE(input, 4);
    ulbuf[2] = GET_ULONG_BE(input, 8);
    ulbuf[3] = GET_ULONG_BE(input, 12);
    while (i < 32) {
        ulbuf[(i + 4)] = sm4F(ulbuf[i], ulbuf[(i + 1)], ulbuf[(i + 2)], ulbuf[(i + 3)], sk[i]);
        i++;
    }
    PUT_ULONG_BE(ulbuf[35], output, 0);
    PUT_ULONG_BE(ulbuf[34], output, 4);
    PUT_ULONG_BE(ulbuf[33], output, 8);
```

```
    PUT_ULONG_BE(ulbuf[32], output, 12);
}

private byte[] padding(byte[] input, int mode) {
    if (input == null) {
        return null;
    }

    byte[] ret = (byte[]) null;
    if (mode == SM4_ENCRYPT) {
        int p = 16 - input.length % 16;
        ret = new byte[input.length + p];
        System.arraycopy(input, 0, ret, 0, input.length);
        for (int i = 0; i < p; i++) {
            ret[input.length + i] = (byte) p;
        }
    } else {
        int p = input[input.length - 1];
        ret = new byte[input.length - p];
        System.arraycopy(input, 0, ret, 0, input.length - p);
    }
    return ret;
}

public void sm4_setkey_enc(SM4_Context ctx, byte[] key) throws Exception {
    if (ctx == null) {
        throw new Exception("ctx is null!");
    }

    if (key == null || key.length != 16) {
        throw new Exception("key error!");
    }

    ctx.mode = SM4_ENCRYPT;
    sm4_setkey(ctx.sk, key);
}

public void sm4_setkey_dec(SM4_Context ctx, byte[] key) throws Exception {
    if (ctx == null) {
        throw new Exception("ctx is null!");
    }
```

```
    if (key == null || key.length != 16) {
        throw new Exception("key error!");
    }

    int i = 0;
    ctx.mode = SM4_DECRYPT;
    sm4_setkey(ctx.sk, key);
    for (i = 0; i < 16; i++) {
        SWAP(ctx.sk, i);
    }
}

public byte[] sm4_crypt_ecb(SM4_Context ctx, byte[] input) throws Exception {
    if (input == null) {
        throw new Exception("input is null!");
    }

    if ((ctx.isPadding) && (ctx.mode == SM4_ENCRYPT)) {
        input = padding(input, SM4_ENCRYPT);
    }

    int length = input.length;
    ByteArrayInputStream bins = new ByteArrayInputStream(input);
    ByteArrayOutputStream bous = new ByteArrayOutputStream();
    for (; length > 0; length -= 16) {
        byte[] in = new byte[16];
        byte[] out = new byte[16];
        bins.read(in);
        sm4_one_round(ctx.sk, in, out);
        bous.write(out);
    }

    byte[] output = bous.toByteArray();
    if (ctx.isPadding && ctx.mode == SM4_DECRYPT) {
        output = padding(output, SM4_DECRYPT);
    }
    bins.close();
    bous.close();
    return output;
}

public byte[] sm4_crypt_cbc(SM4_Context ctx, byte[] iv, byte[] input) throws Exception {
```

```
if (iv == null || iv.length != 16) {
    throw new Exception("iv error!");
}

if (input == null) {
    throw new Exception("input is null!");
}

if (ctx.isPadding && ctx.mode == SM4_ENCRYPT) {
    input = padding(input, SM4_ENCRYPT);
}

int i = 0;
int length = input.length;
ByteArrayInputStream bins = new ByteArrayInputStream(input);
ByteArrayOutputStream bous = new ByteArrayOutputStream();
if (ctx.mode == SM4_ENCRYPT) {
    for (; length > 0; length -= 16) {
        byte[] in = new byte[16];
        byte[] out = new byte[16];
        byte[] out1 = new byte[16];

        bins.read(in);
        for (i = 0; i < 16; i++) {
            out[i] = ((byte) (in[i] ^ iv[i]));
        }
        sm4_one_round(ctx.sk, out, out1);
        System.arraycopy(out1, 0, iv, 0, 16);
        bous.write(out1);
    }
} else {
    byte[] temp = new byte[16];
    for (; length > 0; length -= 16) {
        byte[] in = new byte[16];
        byte[] out = new byte[16];
        byte[] out1 = new byte[16];

        bins.read(in);
        System.arraycopy(in, 0, temp, 0, 16);
        sm4_one_round(ctx.sk, in, out);
        for (i = 0; i < 16; i++) {
            out1[i] = ((byte) (out[i] ^ iv[i]));
```

```
                }
                System.arraycopy(temp, 0, iv, 0, 16);
                bous.write(out1);
            }
        }

        byte[] output = bous.toByteArray();
        if (ctx.isPadding && ctx.mode == SM4_DECRYPT) {
            output = padding(output, SM4_DECRYPT);
        }
        bins.close();
        bous.close();
        return output;
    }
}

36:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\SM4Utils.java
 */
package io.nuls.core.tools.crypto;

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

import java.util.Base64;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * Created by facjas on 2017/11/20.
 */

public class SM4Utils {

    private static final Pattern PATTERN_1 = Pattern.compile("\\s*|\t|\r|\n");

    private String secretKey = "";
    private String iv = "";
    private boolean hexString = false;

    public SM4Utils() {
    }
```

```
public String encryptData_ECB(String plainText) {
    try {
        SM4_Context ctx = new SM4_Context();
        ctx.isPadding = true;
        ctx.mode = SM4.SM4_ENCRYPT;

        byte[] keyBytes;
        if (hexString) {
            keyBytes = Util.hexStringToBytes(secretKey);
        } else {
            keyBytes = secretKey.getBytes();
        }

        SM4 sm4 = new SM4();
        sm4.sm4_setkey_enc(ctx, keyBytes);
        byte[] encrypted = sm4.sm4_crypt_ecb(ctx, plainText.getBytes("GBK"));
        Base64.Encoder encoder = Base64.getEncoder();
        String cipherText = encoder.encodeToString(encrypted);
        if (cipherText != null && cipherText.trim().length() > 0) {

            Matcher m = PATTERN_1.matcher(cipherText);
            cipherText = m.replaceAll("");
        }
        return cipherText;
    } catch (Exception e) {
        Log.error(e);
        return null;
    }
}

public String decryptData_ECB(String cipherText) {
    try {
        SM4_Context ctx = new SM4_Context();
        ctx.isPadding = true;
        ctx.mode = SM4.SM4_DECRYPT;

        byte[] keyBytes;
        if (hexString) {
            keyBytes = Util.hexStringToBytes(secretKey);
        } else {
            keyBytes = secretKey.getBytes();
        }
```

```
        SM4 sm4 = new SM4();
        sm4.sm4_setkey_dec(ctx, keyBytes);
        Base64.Decoder decoder = Base64.getDecoder();
        byte[] decrypted = sm4.sm4_crypt_ecb(ctx, decoder.decode(cipherText));
        return new String(decrypted, "UTF-8");
    } catch (Exception e) {
        Log.error(e);
        return null;
    }
}

public String encryptData_CBC(String plainText) {
    try {
        SM4_Context ctx = new SM4_Context();
        ctx.isPadding = true;
        ctx.mode = SM4.SM4_ENCRYPT;

        byte[] keyBytes;
        byte[] ivBytes;
        if (hexString) {
            keyBytes = Util.hexStringToBytes(secretKey);
            ivBytes = Util.hexStringToBytes(iv);
        } else {
            keyBytes = secretKey.getBytes();
            ivBytes = iv.getBytes();
        }

        SM4 sm4 = new SM4();
        sm4.sm4_setkey_enc(ctx, keyBytes);
        byte[] encrypted = sm4.sm4_crypt_cbc(ctx, ivBytes, plainText.getBytes("UTF-8"));
        Base64.Encoder encoder = Base64.getEncoder();
        String cipherText = encoder.encodeToString(encrypted);
        if (cipherText != null && cipherText.trim().length() > 0) {
            Matcher m = PATTERN_1.matcher(cipherText);
            cipherText = m.replaceAll("");
        }
        return cipherText;
    } catch (Exception e) {
        Log.error(e);
        return null;
    }
```

```
    }

    public String decryptData_CBC(String cipherText) {
        try {
            SM4_Context ctx = new SM4_Context();
            ctx.isPadding = true;
            ctx.mode = SM4.SM4_DECRYPT;

            byte[] keyBytes;
            byte[] ivBytes;
            if (hexString) {
                keyBytes = Util.hexStringToBytes(secretKey);
                ivBytes = Util.hexStringToBytes(iv);
            } else {
                keyBytes = secretKey.getBytes();
                ivBytes = iv.getBytes();
            }
            SM4 sm4 = new SM4();
            sm4.sm4_setkey_dec(ctx, keyBytes);
            Base64.Decoder decoder = Base64.getDecoder();
            byte[] decrypted = sm4.sm4_crypt_cbc(ctx, ivBytes, decoder.decode(cipherText));
            return new String(decrypted, "UTF-8");
        } catch (Exception e) {
            Log.error(e);
            return null;
        }
    }
}

37:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\SM4_Context.java
 */
package io.nuls.core.tools.crypto;

/**
 * Created by facjas on 2017/11/20.
 */
public class SM4_Context {
    public int mode;

    public long[] sk;
```

```
    public boolean isPadding;

    public SM4_Context() {
        this.mode = 1;
        this.isPadding = true;
        this.sk = new long[32];
    }
}

38:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\UnsafeByteArrayOutputStream.java
 */

package io.nuls.core.tools.crypto;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.OutputStream;

/**
 * An unsynchronized implementation of ByteArrayOutputStream that will return the backing byte
array if its length == size().
 * This avoids unneeded array copy where the BOS is simply being used to extract a byte array of
known length from a
 * 'serialized to stream' method.
 * Unless the final length can be accurately predicted the only performance this will yield is due to
unsynchronized
 * methods.
 *
 * @author git
 */
public class UnsafeByteArrayOutputStream extends ByteArrayOutputStream {

    public UnsafeByteArrayOutputStream() {
        super(32);
    }

    public UnsafeByteArrayOutputStream(int size) {
        super(size);
    }

    /**
```

```
 * Writes the specified byte to this byte array output stream.
 *
 * @param b the byte to be written.
 */
@Override
public void write(int b) {
    int newcount = count + 1;
    if (newcount > buf.length) {
        buf = Util.copyOf(buf, Math.max(buf.length << 1, newcount));
    }
    buf[count] = (byte) b;
    count = newcount;
}

/**
 * Writes <code>len</code> bytes from the specified byte array
 * starting at offset <code>off</code> to this byte array output stream.
 *
 * @param b   the data.
 * @param off the start offset in the data.
 * @param len the number of bytes to write.
 */
@Override
public void write(byte[] b, int off, int len) {
    if ((off < 0) || (off > b.length) || (len < 0) ||
            ((off + len) > b.length) || ((off + len) < 0)) {
        throw new IndexOutOfBoundsException();
    } else if (len == 0) {
        return;
    }
    int newcount = count + len;
    if (newcount > buf.length) {
        buf = Util.copyOf(buf, Math.max(buf.length << 1, newcount));
    }
    System.arraycopy(b, off, buf, count, len);
    count = newcount;
}

/**
 * Writes the complete contents of this byte array output stream to
 * the specified output stream argument, as if by calling the output
 * stream's write method using <code>out.write(buf, 0, count)</code>.
```

```
     *
     * @param out the output stream to which to write the data.
     * @throws IOException if an I/O error occurs.
     */
    @Override
    public void writeTo(OutputStream out) throws IOException {
        out.write(buf, 0, count);
    }

    /**
     * Resets the <code>count</code> field of this byte array output
     * stream to zero, so that all currently accumulated output in the
     * output stream is discarded. The output stream can be used again,
     * reusing the already allocated buffer space.
     *
     * @see java.io.ByteArrayInputStream#count
     */
    @Override
    public void reset() {
        count = 0;
    }

    /**
     * Creates a newly allocated byte array. Its size is the current
     * size of this output stream and the valid contents of the buffer
     * have been copied into it.
     *
     * @return the current contents of this output stream, as a byte array.
     * @see ByteArrayOutputStream#size()
     */
    @Override
    public byte toByteArray()[] {
        return count == buf.length ? buf : Util.copyOf(buf, count);
    }

    /**
     * Returns the current size of the buffer.
     *
     * @return the value of the <code>count</code> field, which is the number
     *         of valid bytes in this output stream.
     * @see ByteArrayOutputStream#count
     */
```

```
    @Override
    public int size() {
        return count;
    }

}

39:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\crypto\Util.java
 */
package io.nuls.core.tools.crypto;

import java.math.BigInteger;

/**
 * Created by facjas on 2017/11/20.
 */

public class Util {
    public static byte[] intToBytes(int num) {
        byte[] bytes = new byte[4];
        bytes[0] = (byte) (0xff & (num >> 0));
        bytes[1] = (byte) (0xff & (num >> 8));
        bytes[2] = (byte) (0xff & (num >> 16));
        bytes[3] = (byte) (0xff & (num >> 24));
        return bytes;
    }

    public static int byteToInt(byte[] bytes) {
        int num = 0;
        int temp;
        temp = (0x000000ff & (bytes[0])) << 0;
        num = num | temp;
        temp = (0x000000ff & (bytes[1])) << 8;
        num = num | temp;
        temp = (0x000000ff & (bytes[2])) << 16;
        num = num | temp;
        temp = (0x000000ff & (bytes[3])) << 24;
        num = num | temp;
        return num;
    }
```

```
    public static byte[] longToBytes(long num) {
        byte[] bytes = new byte[8];
        for (int i = 0; i < 8; i++) {
            bytes[i] = (byte) (0xff & (num >> (i * 8)));
        }

        return bytes;
    }

    public static byte[] byteConvert32Bytes(BigInteger n) {
        byte tmpd[] = (byte[]) null;
        if (n == null) {
            return null;
        }

        if (n.toByteArray().length == 33) {
            tmpd = new byte[32];
            System.arraycopy(n.toByteArray(), 1, tmpd, 0, 32);
        } else if (n.toByteArray().length == 32) {
            tmpd = n.toByteArray();
        } else {
            tmpd = new byte[32];
            for (int i = 0; i < 32 - n.toByteArray().length; i++) {
                tmpd[i] = 0;
            }
            System.arraycopy(n.toByteArray(), 0, tmpd, 32 - n.toByteArray().length,
n.toByteArray().length);
        }
        return tmpd;
    }

    public static BigInteger byteConvertInteger(byte[] b) {
        if (b[0] < 0) {
            byte[] temp = new byte[b.length + 1];
            temp[0] = 0;
            System.arraycopy(b, 0, temp, 1, b.length);
            return new BigInteger(temp);
        }
        return new BigInteger(b);
    }

    public static String getHexString(byte[] bytes) {
```

```
    return getHexString(bytes, true);
}


public static String getHexString(byte[] bytes, boolean upperCase) {
    String ret = "";
    for (int i = 0; i < bytes.length; i++) {
        ret += Integer.toString((bytes[i] & 0xff) + 0x100, 16).substring(1);
    }
    return upperCase ? ret.toUpperCase() : ret;
}

public static void printHexString(byte[] bytes) {
    for (int i = 0; i < bytes.length; i++) {
        String hex = Integer.toHexString(bytes[i] & 0xFF);
        if (hex.length() == 1) {
            hex = '0' + hex;
        }
    }
}

public static byte[] hexStringToBytes(String hexString) {
    if (hexString == null || "".equals(hexString)) {
        return null;
    }

    hexString = hexString.toUpperCase();
    int length = hexString.length() / 2;
    char[] hexChars = hexString.toCharArray();
    byte[] d = new byte[length];
    for (int i = 0; i < length; i++) {
        int pos = i * 2;
        d[i] = (byte) (charToByte(hexChars[pos]) << 4 | charToByte(hexChars[pos + 1]));
    }
    return d;
}

public static byte charToByte(char c) {
    return (byte) "0123456789ABCDEF".indexOf(c);
}

private static final char[] DIGITS_LOWER = {'0', '1', '2', '3', '4', '5',
```

```
        '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

private static final char[] DIGITS_UPPER = {'0', '1', '2', '3', '4', '5',
        '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};

public static char[] encodeHex(byte[] data) {
    return encodeHex(data, true);
}



public static char[] encodeHex(byte[] data, boolean toLowerCase) {
    return encodeHex(data, toLowerCase ? DIGITS_LOWER : DIGITS_UPPER);
}

protected static char[] encodeHex(byte[] data, char[] toDigits) {
    int l = data.length;
    char[] out = new char[l << 1];
    for (int i = 0, j = 0; i < l; i++) {
        out[j++] = toDigits[(0xF0 & data[i]) >>> 4];
        out[j++] = toDigits[0x0F & data[i]];
    }
    return out;
}

public static String encodeHexString(byte[] data) {
    return encodeHexString(data, true);
}

public static String encodeHexString(byte[] data, boolean toLowerCase) {
    return encodeHexString(data, toLowerCase ? DIGITS_LOWER : DIGITS_UPPER);
}

protected static String encodeHexString(byte[] data, char[] toDigits) {
    return new String(encodeHex(data, toDigits));
}

public static byte[] decodeHex(char[] data) {
    int len = data.length;

    if ((len & 0x01) != 0) {
        throw new RuntimeException("Odd number of characters.");
    }
```

```
    byte[] out = new byte[len >> 1];

    for (int i = 0, j = 0; j < len; i++) {
        int f = toDigit(data[j], j) << 4;
        j++;
        f = f | toDigit(data[j], j);
        j++;
        out[i] = (byte) (f & 0xFF);
    }

    return out;
}

protected static int toDigit(char ch, int index) {
    int digit = Character.digit(ch, 16);
    if (digit == -1) {
        throw new RuntimeException("Illegal hexadecimal character " + ch
                + " at index " + index);
    }
    return digit;
}

public static String stringToAsciiString(String content) {
    String result = "";
    int max = content.length();
    for (int i = 0; i < max; i++) {
        char c = content.charAt(i);
        String b = Integer.toHexString(c);
        result = result + b;
    }
    return result;
}

public static String hexStringToString(String hexString, int encodeType) {
    String result = "";
    int max = hexString.length() / encodeType;
    for (int i = 0; i < max; i++) {
        char c = (char) hexStringToAlgorism(hexString
                .substring(i * encodeType, (i + 1) * encodeType));
        result += c;
    }
```

```
    return result;
}

public static int hexStringToAlgorism(String hex) {
    hex = hex.toUpperCase();
    int max = hex.length();
    int result = 0;
    for (int i = max; i > 0; i--) {
        char c = hex.charAt(i - 1);
        int algorism = 0;
        if (c >= '0' && c <= '9') {
            algorism = c - '0';
        } else {
            algorism = c - 55;
        }
        result += Math.pow(16, max - i) * algorism;
    }
    return result;
}

public static String hexStringToBinary(String hex) {
    hex = hex.toUpperCase();
    String result = "";
    int max = hex.length();
    for (int i = 0; i < max; i++) {
        char c = hex.charAt(i);
        switch (c) {
            case '0':
                result += "0000";
                break;
            case '1':
                result += "0001";
                break;
            case '2':
                result += "0010";
                break;
            case '3':
                result += "0011";
                break;
            case '4':
                result += "0100";
                break;
```

```
            case '5':
                result += "0101";
                break;
            case '6':
                result += "0110";
                break;
            case '7':
                result += "0111";
                break;
            case '8':
                result += "1000";
                break;
            case '9':
                result += "1001";
                break;
            case 'A':
                result += "1010";
                break;
            case 'B':
                result += "1011";
                break;
            case 'C':
                result += "1100";
                break;
            case 'D':
                result += "1101";
                break;
            case 'E':
                result += "1110";
                break;
            case 'F':
                result += "1111";
                break;
            default:
                break;
        }
    }
    return result;
}

public static String asciiStringToString(String content) {
    String result = "";
```

```
    int length = content.length() / 2;
    for (int i = 0; i < length; i++) {
        String c = content.substring(i * 2, i * 2 + 2);
        int a = hexStringToAlgorism(c);
        char b = (char) a;
        String d = String.valueOf(b);
        result += d;
    }
    return result;
}

public static String algorismToHexString(int algorism, int maxLength) {
    String result = "";
    result = Integer.toHexString(algorism);

    if (result.length() % 2 == 1) {
        result = "0" + result;
    }
    return patchHexString(result.toUpperCase(), maxLength);
}

public static String byteToString(byte[] bytearray) {
    String result = "";
    char temp;

    int length = bytearray.length;
    for (int i = 0; i < length; i++) {
        temp = (char) bytearray[i];
        result += temp;
    }
    return result;
}

public static int binaryToAlgorism(String binary) {
    int max = binary.length();
    int result = 0;
    for (int i = max; i > 0; i--) {
        char c = binary.charAt(i - 1);
        int algorism = c - '0';
        result += Math.pow(2, max - i) * algorism;
    }
    return result;
```

```
}

public static String algorismToHEXString(int algorism) {
    String result = "";
    result = Integer.toHexString(algorism);

    if (result.length() % 2 == 1) {
        result = "0" + result;

    }
    result = result.toUpperCase();

    return result;
}

static public String patchHexString(String str, int maxLength) {
    String temp = "";
    for (int i = 0; i < maxLength - str.length(); i++) {
        temp = "0" + temp;
    }
    str = (temp + str).substring(0, maxLength);
    return str;
}

public static int parseToInt(String s, int defaultInt, int radix) {
    int i = 0;
    try {
        i = Integer.parseInt(s, radix);
    } catch (NumberFormatException ex) {
        i = defaultInt;
    }
    return i;
}

public static int parseToInt(String s, int defaultInt) {
    int i = 0;
    try {
        i = Integer.parseInt(s);
    } catch (NumberFormatException ex) {
        i = defaultInt;
    }
    return i;
```

```
}

public static byte[] hexToByte(String hex)
        throws IllegalArgumentException {
    if (hex.length() % 2 != 0) {
        throw new IllegalArgumentException();
    }
    char[] arr = hex.toCharArray();
    byte[] b = new byte[hex.length() / 2];
    for (int i = 0, j = 0, l = hex.length(); i < l; i++, j++) {
        String swap = "" + arr[i++] + arr[i];
        int byteint = Integer.parseInt(swap, 16) & 0xFF;
        b[j] = new Integer(byteint).byteValue();
    }
    return b;
}

public static String byteToHex(byte b[]) {
    if (b == null) {
        throw new IllegalArgumentException(
                "Argument b ( byte array ) is null! ");
    }
    String hs = "";
    String stmp = "";
    for (int n = 0; n < b.length; n++) {
        stmp = Integer.toHexString(b[n] & 0xff);
        if (stmp.length() == 1) {
            hs = hs + "0" + stmp;
        } else {
            hs = hs + stmp;
        }
    }
    return hs.toUpperCase();
}

public static byte[] subByte(byte[] input, int startIndex, int length) {
    byte[] bt = new byte[length];
    for (int i = 0; i < length; i++) {
        bt[i] = input[i + startIndex];
    }
    return bt;
}
```

```
    public static <T> T checkNotNull(T t) {
        if (t == null) {
            throw new NullPointerException();
        }
        return t;
    }

    private static int isAndroid = -1;

    public static boolean isAndroidRuntime() {
        if (isAndroid == -1) {
            final String runtime = System.getProperty("java.runtime.name");
            isAndroid = (runtime != null && "Android Runtime".equals(runtime)) ? 1 : 0;
        }
        return isAndroid == 1;
    }


    public static void checkState(boolean status) {
        if (status) {
            return;
        } else {
            throw new RuntimeException();
        }
    }

    public static byte[] reverseBytes(byte[] bytes) {
        // We could use the XOR trick here but it's easier to understand if we don't. If we find this is
really a
        // performance issue the matter can be revisited.
        byte[] buf = new byte[bytes.length];
        for (int i = 0; i < bytes.length; i++) {
            buf[i] = bytes[bytes.length - 1 - i];
        }
        return buf;
    }


    public static byte[] copyOf(byte[] in, int length) {
        byte[] out = new byte[length];
        System.arraycopy(in, 0, out, 0, Math.min(length, in.length));
```

```
        return out;
    }
}

40:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\date\DateUtil.java
 */
package io.nuls.core.tools.date;


import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.*;

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

    public final static String EMPTY_SRING = "";
    public final static String DEFAULT_PATTERN = "yyyy-MM-dd HH:mm:ss";


    public final static long DATE_TIME = 1000 * 24 * 60 * 60;
    public final static long HOUR_TIME = 1000 * 60 * 60;
    public final static long MINUTE_TIME = 1000 * 60;

    public static String toGMTString(Date date) {
        SimpleDateFormat df = new SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z", Locale.UK);
        df.setTimeZone(new SimpleTimeZone(0, "GMT"));
        return df.format(date);
    }

//    /**
//     * yyyy-MM-dd HH:mm:ss
//     *
//     * @param date
//     * @return String
//     */
    public static String convertDate(Date date) {
```

```
        if (date == null) {
            return EMPTY_SRING;
        }
        return new SimpleDateFormat(DEFAULT_PATTERN).format(date);
    }

//    /**
//     * pattern
//     *
//     * @param date
//     * @param pattern
//     * @return String
//     */
    public static String convertDate(Date date, String pattern) {
        if (date == null) {
            return EMPTY_SRING;
        }
        return new SimpleDateFormat(pattern).format(date);
    }

//    /**
//     * @param date
//     * @return Date
//     */
    public static Date convertStringToDate(String date) {
        try {
            return new SimpleDateFormat(DEFAULT_PATTERN).parse(date);
        } catch (ParseException e) {
        }
        return new Date();
    }

//    /**
//     * @param date
//     * @param pattern
//     * @return Date
//     */
    public static Date convertStringToDate(String date, String pattern) {
        try {
            return new SimpleDateFormat(pattern).parse(date);
        } catch (ParseException e) {
            throw new RuntimeException(e.getMessage());
```

```
        }
    }
//
//    /**
//     *
//     *
//     * @param date
//     * @return boolean
//     */
    public static boolean isFirstDayInMonth(Date date) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        return calendar.get(Calendar.DAY_OF_MONTH) == 1;
    }

//    /**
//     *
//     *
//     * @param date
//     * @return boolean
//     */
    public static boolean isFirstDayInYear(Date date) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        return calendar.get(Calendar.DAY_OF_YEAR) == 1;
    }

//    /**
//     *
//     *
//     * @param date
//     * @return Date
//     */
    public static Date rounding(Date date) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.set(Calendar.HOUR_OF_DAY, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
        return calendar.getTime();
    }
```

```
//    /**
//     * dayday
//     *
//     * @param date
//     * @param day
//     * @return Date
//     */
    public static Date dateAdd(Date date, int day) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.set(Calendar.DATE, calendar.get(Calendar.DATE) + day);
        return calendar.getTime();
    }

//    /**
//     *
//     *
//     * @param date
//     * @param month
//     * @return Date
//     */
    public static Date dateAddMonth(Date date, int month) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.set(Calendar.MONTH, calendar.get(Calendar.MONTH) + month);
        return calendar.getTime();
    }

//    /**
//     *
//     *
//     * @return Date
//     */
    public static Date getFirstDayOfPreviousMonth() {
        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.MONTH, calendar.get(Calendar.MONTH) - 1);
        calendar.set(Calendar.DATE, 1);
        calendar.set(Calendar.HOUR_OF_DAY, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
        return calendar.getTime();
    }
```

```
//    /**
//     *
//     *
//     * @return Date
//     */
    public static Date getFirstDayOfMonth() {
        return getFirstDayOfMonth(new Date());
    }

//    /**
//     *
//     * @param date
//     * @return Date
//     */
    public static Date getFirstDayOfMonth(Date date) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(date);
        calendar.set(Calendar.DATE, 1);
        calendar.set(Calendar.HOUR_OF_DAY, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
        return calendar.getTime();
    }

//    /**
//     *
//     *
//     * @return Date
//     */
    public static Date getFirstDayOfPreviousYear() {
        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.YEAR, calendar.get(Calendar.YEAR) - 1);
        calendar.set(Calendar.MONTH, 0);
        calendar.set(Calendar.DATE, 1);
        calendar.set(Calendar.HOUR_OF_DAY, 0);
        calendar.set(Calendar.MINUTE, 0);
        calendar.set(Calendar.SECOND, 0);
        return calendar.getTime();
    }

    public static List<String> getDateRange(String beginDate, String endDate,
```

```
                                    int type) {
    List<String> list = new ArrayList<String>();
    if (isEmpty(beginDate) || isEmpty(endDate)) {
        return list;
    }
    if (type == 1) {
        Date begin = convertStringToDate(beginDate, "yyyy-MM-dd");
        Date end = convertStringToDate(endDate, "yyyy-MM-dd");
        if (begin == null || end == null) {
            return list;
        }
        while (begin.equals(end) || begin.before(end)) {
            list.add(convertDate(begin, "MM-dd"));
            begin = dateAdd(begin, 1);
        }
    } else if (type == 2) {
        Date begin = convertStringToDate(beginDate, "yyyy-MM-dd");
        Date end = convertStringToDate(endDate, "yyyy-MM-dd");
        if (begin == null || end == null) {
            return list;
        }
        Calendar beginCalendar = Calendar.getInstance();
        beginCalendar.setTime(begin);
        beginCalendar.set(Calendar.DAY_OF_MONTH, 1);
        Calendar endCalendar = Calendar.getInstance();
        endCalendar.setTime(end);
        endCalendar.set(Calendar.DAY_OF_MONTH, 1);

        while (beginCalendar.getTime().equals(endCalendar.getTime())
                || beginCalendar.getTime().before(endCalendar.getTime())) {
            list.add(convertDate(beginCalendar.getTime(), "yyyy-MM"));
            beginCalendar.set(Calendar.MONTH,
                    beginCalendar.get(Calendar.MONTH) + 1);
        }
    }
    return list;
}

public static boolean isEmpty(Object obj) {
    return obj == null || EMPTY_SRING.equals(obj);
}
```

```
//    /**
//     *
//     *
//     * @param c
//     * @return String
//     */
    public static String getWeekDay(Calendar c) {
        if (c == null) {
            return "";
        }
        if (Calendar.MONDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.TUESDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.WEDNESDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.THURSDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.FRIDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.SATURDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        if (Calendar.SUNDAY == c.get(Calendar.DAY_OF_WEEK)) {
            return "";
        }
        return "";
    }

//    /**
//     *
//     *
//     * @param startTime
//     * @param endTime
//     * @param showSuffix
//     * @return String
//     */
```

```
    public static String convertLebalFull(Date startTime, Date endTime,
                                          boolean showSuffix) {
        if (startTime == null || endTime == null) {
            return EMPTY_SRING;
        }
        //
        long time = (startTime.getTime() - endTime.getTime()) / 1000;
        String label = analyzeTime(time, true);
        if (showSuffix) {
            label += (time > 0) ? "" : "";
        }
        return label;
    }

//    /**
//     *
//     *
//     * @param startTime
//     * @param endTime
//     * @param showSuffix
//     * @return String
//     */
    public static String convertLebal(Date startTime, Date endTime,
                                      boolean showSuffix) {
        if (startTime == null || endTime == null) {
            return EMPTY_SRING;
        }
        //
        long time = (startTime.getTime() - endTime.getTime()) / 1000;
        String label = analyzeTime(time, false);
        if (showSuffix) {
            label += (time > 0) ? "" : "";
        }
        return label;
    }

    public static String analyzeTime(long time, boolean showFull) {
        String remark = EMPTY_SRING;
        long tempTime = Math.abs(time);
        if (tempTime < 60) {
            remark = String.format("%s", tempTime);
        } else if (tempTime < 3600) {
```

```
            remark = String.format("%s%s", tempTime / 60, tempTime % 60);
        } else if (tempTime / 3600 < 24) {
            if (showFull) {
                remark = String.format("%s%s%s", tempTime / 3600,
                        (tempTime / 60) % 60, tempTime % 60);
            } else {
                remark = String.format("%s%s", tempTime / 3600,
                        (tempTime / 60) % 60);
            }
        } else if (tempTime / (3600 * 24L) < 30) {
            if (showFull) {
                remark = String.format("%s%s%s%s",
                        tempTime / (3600 * 24L), (tempTime / 3600) % 24,
                        (tempTime / 60) % 60, tempTime % 60);
            } else {
                remark = String.format("%s%s", tempTime / (3600 * 24L),
                        (tempTime / 3600) % 24);
            }
        } else if (tempTime / (3600 * 24 * 30L) <= 12) {
            if (showFull) {
                remark = String.format("%%s%s", tempTime
                            / (3600 * 24 * 30L), tempTime / (3600 * 24L),
                        (tempTime / 3600) % 24);
            } else {
                remark = tempTime / (3600 * 24 * 30L) + "" + tempTime
                        / (3600 * 24L) % 30 + "";
            }
        } else if (tempTime / (3600 * 24 * 30L) < 12) {

        }
        return remark;
    }

    public static Date getToday() {
        return rounding(new Date());
    }

    public static Date getYesterday() {
        return rounding(dateAdd(new Date(), -1));
    }

//    /**
```

```
//     *
//     *
//     * @param startTime
//     * @param endTime
//     * @return int
//     */
    public static int getBetweenDateDays(Date startTime, Date endTime) {
        if (startTime == null || endTime == null) {
            return 0;
        }
        long to = startTime.getTime();
        long from = endTime.getTime();

        return (int) ((from - to) / (1000L * 60 * 60 * 24));
    }

    public static Date getTomorrow() {
        return rounding(dateAdd(new Date(), 1));
    }

//    /**
//     *
//     *
//     * @param date
//     * @param time
//     * @return boolean
//     */
    public static boolean checkAfterTime(Date date, String time) {
        Date dateTime = convertStringToDate(convertDate(date, "yyyy-MM-dd").concat("
").concat(time));
        return dateTime.before(date);
    }

    public static String getOffsetStringDate(long offsetTime) {
        int p = offsetTime > 0 ? 1 : -1;

        offsetTime = Math.abs(offsetTime);
        if (offsetTime < 1000) {
            return p * offsetTime + "ms";
        } else if (offsetTime < MINUTE_TIME) {
            long sec = offsetTime % DATE_TIME % HOUR_TIME % MINUTE_TIME / 1000;
            return p * sec + "s";
```

```
        } else if (offsetTime < HOUR_TIME) {
            long minute = offsetTime % DATE_TIME % HOUR_TIME / MINUTE_TIME;
            long sec = offsetTime % DATE_TIME % HOUR_TIME % MINUTE_TIME / 1000;
            if (minute >= 10) {
                return p * minute + "m";
            } else {
                return p * minute + "m" + sec + "s";
            }
        } else {
            long hour = offsetTime % DATE_TIME / HOUR_TIME;
            long minute = offsetTime % DATE_TIME % HOUR_TIME / MINUTE_TIME;
            if (hour >= 5) {
                return p * hour + "h";
            } else {
                return p * hour + "h" + minute + "m";
            }
        }
    }

}

41:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\disruptor\DisruptorData.java
 */
package io.nuls.core.tools.disruptor;

/**
 *
 * @author Niels
 */
public class DisruptorData<T> {

    private String name;

    private T data;

    private boolean stoped = false;

    public boolean isStoped() {
        return stoped;
    }
```

```
    public void setStoped(boolean stoped) {
        this.stoped = stoped;
    }

    public String getName() {
        return name;
    }

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

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }
}

42:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\disruptor\DisruptorUtil.java
 */
package io.nuls.core.tools.disruptor;

import com.lmax.disruptor.*;
import com.lmax.disruptor.dsl.Disruptor;
import com.lmax.disruptor.dsl.EventHandlerGroup;
import com.lmax.disruptor.dsl.ProducerType;
import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ThreadFactory;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * @author Niels
 */
```

```
public class DisruptorUtil<T extends DisruptorData> {

    private Lock locker = new ReentrantLock();

    private static final DisruptorUtil INSTANCE = new DisruptorUtil();
    private static final Map<String, Disruptor<DisruptorData>> DISRUPTOR_MAP = new
HashMap<>();

    public static DisruptorUtil getInstance() {
        return INSTANCE;
    }

    private DisruptorUtil() {
    }

    private static final EventFactory EVENT_FACTORY = new EventFactory() {
        @Override
        public Object newInstance() {
            return new DisruptorData();
        }
    };

//    /**
//     * create a disruptor
//     *
//     * @param name           The title of the disruptor
//     * @param ringBufferSize The size of ringBuffer
//     */
    public Disruptor<DisruptorData> createDisruptor(String name, int ringBufferSize, ThreadFactory
factory) {
        if (DISRUPTOR_MAP.keySet().contains(name)) {
            throw new RuntimeException("create disruptor faild,the name is repetitive!");
        }

        Disruptor<DisruptorData> disruptor = new Disruptor<DisruptorData>(EVENT_FACTORY,
                ringBufferSize, factory, ProducerType.MULTI,
                new BlockingWaitStrategy());
        disruptor.setDefaultExceptionHandler(new NulsExceptionHandler());
        //SleepingWaitStrategy
//        disruptor.handleEventsWith(new EventHandler<DisruptorData>() {
//            @Override
//            public void onEvent(DisruptorData DisruptorData, long l, boolean b) throws Exception {
```

```
//                Log.debug(DisruptorData.getData() + "");
//            }
//        });
        DISRUPTOR_MAP.put(name, disruptor);

        return disruptor;
    }

//    /**
//     * start a disruptor service
//     */
    public void start(String name) {
        Disruptor<DisruptorData> disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        disruptor.start();
    }

//    /**
//     * start a disruptor service
//     */
    public void shutdown(String name) {
        Disruptor<DisruptorData> disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        disruptor.shutdown();
    }



//    /**
//     * add the data obj to the disruptor named the field name
//     */
    public void offer(String name, Object obj) {

        Disruptor<DisruptorData> disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        RingBuffer<DisruptorData> ringBuffer = disruptor.getRingBuffer();

//        locker.lock();
        try {
            //
            long sequence = ringBuffer.next();
            try {
                //
```

```
                DisruptorData event = ringBuffer.get(sequence);
                event.setData(obj);
            } catch (Exception e) {
                Log.error(e);
            } finally {
                //
                ringBuffer.publish(sequence);
            }
        } finally {
//            locker.unlock();
        }
    }

//    /**
//     * add some message to worker pool of the disruptor
//     */
    public EventHandlerGroup<T> handleEventsWithWorkerPool(String name,
WorkHandler<DisruptorData>... handler) {
        Disruptor disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        return disruptor.handleEventsWithWorkerPool(handler);
    }

    public EventHandlerGroup<T> handleEventWith(String name, EventHandler<T> eventHandler)
{
        Disruptor disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        return disruptor.handleEventsWith(eventHandler);
    }

    public EventHandlerGroup<T> after(String name, EventHandler<T> eventHandler) {
        Disruptor disruptor = DISRUPTOR_MAP.get(name);
        AssertUtil.canNotEmpty(disruptor, "the disruptor is not exist!name:" + name);
        return disruptor.after(eventHandler);
    }
}

43:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\disruptor\NulsExceptionHandler.java
 */

package io.nuls.core.tools.disruptor;
```

```
import com.lmax.disruptor.BatchEventProcessor;
import com.lmax.disruptor.EventHandler;
import com.lmax.disruptor.ExceptionHandler;
import com.lmax.disruptor.LifecycleAware;
import io.nuls.core.tools.log.Log;

/**
 * @author: Niels Wang
 */
public class NulsExceptionHandler implements ExceptionHandler<Object> {
    /**
     * <p>Strategy for handling uncaught exceptions when processing an event.</p>
     * <p>If the strategy wishes to terminate further processing by the {@link BatchEventProcessor}
     * then it should throw a {@link RuntimeException}.</p>
     *
     * @param ex       the exception that propagated from the {@link EventHandler}.
     * @param sequence of the event which cause the exception.
     * @param event    being processed when the exception occurred.  This can be null.
     */
    @Override
    public void handleEventException(Throwable ex, long sequence, Object event) {
        Log.error(ex);

    }

    /**
     * Callback to notify of an exception during {@link LifecycleAware#onStart()}
     *
     * @param ex throw during the starting process.
     */
    @Override
    public void handleOnStartException(Throwable ex) {
        Log.error(ex);
    }

    /**
     * Callback to notify of an exception during {@link LifecycleAware#onShutdown()}
     *
     * @param ex throw during the shutdown process.
     */
    @Override
```

```
    public void handleOnShutdownException(Throwable ex) {
        Log.error(ex);
    }
}

44:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\io\HttpDownloadUtils.java
 */

package io.nuls.core.tools.io;

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

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;

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

    public static byte[] download(String urlStr) throws IOException {
        Log.info("Get the version info file from " + urlStr);
        URL url = new URL(urlStr);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setConnectTimeout(60 * 1000);
        conn.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible; MSIE 5.0; Windows NT;
DigExt)");
        InputStream inputStream = conn.getInputStream();
        byte[] getData = readInputStream(inputStream);
        return getData;
    }



//    /**
//     *
//     */
    public static byte[] readInputStream(InputStream inputStream) throws IOException {
        byte[] buffer = new byte[inputStream.available()];
```

```
        int len = 0;
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        while ((len = inputStream.read(buffer)) != -1) {
            bos.write(buffer, 0, len);
        }
        bos.close();
        return bos.toByteArray();
    }
}

45:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\io\StringFileLoader.java
 */
package io.nuls.core.tools.io;

import io.nuls.core.tools.log.Log;
import io.nuls.core.tools.param.AssertUtil;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.net.URLDecoder;

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

    public static String read(String path) throws Exception {
        AssertUtil.canNotEmpty(path, "null parameter");
        String filePath = StringFileLoader.class.getClassLoader().getResource(path).getPath();
        return readRealPath(filePath, false);
    }

    public static String readRealPath(String realPath, boolean format) throws Exception {
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(URLDecoder.decode(realPath, "UTF-8")));
        } catch (FileNotFoundException e) {
            Log.error(e);
            throw new Exception(e);
```

```
        }
        StringBuilder str = new StringBuilder();
        String line;
        try {
            while ((line = br.readLine()) != null) {

                if (format) {
                    str.append(line);
                    str.append("\n");
                } else {
                    str.append(line.trim());
                }
            }
        } catch (IOException e) {
            Log.error(e);
            throw new Exception(e);
        } finally {
            try {
                br.close();
            } catch (IOException e) {
                Log.error(e);
            }
        }
        return str.toString();
    }
}

46:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\json\JSONUtils.java
 */
package io.nuls.core.tools.json;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.JavaType;
import com.fasterxml.jackson.databind.ObjectMapper;

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

```
/**
 * @author Niels
 */
public final class JSONUtils {

    private static final ObjectMapper OBJECT_MAPPER = new ObjectMapper();


    public static ObjectMapper getInstance() {
        return OBJECT_MAPPER;
    }

//    /**
//     * javaBean,list,array convert to json string
//     */
    public static String obj2json(Object obj) throws Exception {
        return OBJECT_MAPPER.writeValueAsString(obj);
    }

    public static String obj2PrettyJson(Object obj) throws Exception {
        return OBJECT_MAPPER.writerWithDefaultPrettyPrinter().writeValueAsString(obj);
    }

//    /**
//     * json string convert to javaBean
//     */
    public static <T> T json2pojo(String jsonStr, Class<T> clazz)
            throws Exception {
        return OBJECT_MAPPER.readValue(jsonStr, clazz);
    }

    public static <T> T json2pojo(String json, Class<T> entityClass, Class... itemClass) throws
IOException {
        JavaType javaType =
OBJECT_MAPPER.getTypeFactory().constructParametricType(entityClass, itemClass);
        return OBJECT_MAPPER.readValue(json, javaType);
    }

//    /**
//     * json string convert to map
//     */
    public static <T> Map<String, Object> json2map(String jsonStr)
```

```
            throws Exception {
        return OBJECT_MAPPER.readValue(jsonStr, Map.class);
    }

//    /**
//     * json string convert to map with javaBean
//     */
    public static <T> Map<String, T> json2map(String jsonStr, Class<T> clazz)
            throws Exception {
        Map<String, Map<String, Object>> map = OBJECT_MAPPER.readValue(jsonStr,
                new TypeReference<Map<String, T>>() {
                });
        Map<String, T> result = new HashMap<String, T>();
        for (Map.Entry<String, Map<String, Object>> entry : map.entrySet()) {
            result.put(entry.getKey(), map2pojo(entry.getValue(), clazz));
        }
        return result;
    }
//
//    /**
//     * json array string convert to list with javaBean
//     */
    public static <T> List<T> json2list(String jsonArrayStr, Class<T> clazz)
            throws Exception {
        List<Map<String, Object>> list = OBJECT_MAPPER.readValue(jsonArrayStr,
                new TypeReference<List<T>>() {
                });
        List<T> result = new ArrayList<T>();
        for (Map<String, Object> map : list) {
            result.add(map2pojo(map, clazz));
        }
        return result;
    }

//    /**
//     * map convert to javaBean
//     */
    public static <T> T map2pojo(Map map, Class<T> clazz) {
        return OBJECT_MAPPER.convertValue(map, clazz);
    }

}
```

47:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\log\BlockLog.java
 */
package io.nuls.core.tools.log;

import ch.qos.logback.classic.Level;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

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

/**
 *
 *
 * slf4j
 *
 * @author Niels
 */
public final class BlockLog {
    /**
     *
     */
    private static final Logger LOG = LoggerFactory.getLogger("blockLog");

    /**
     *
     */
    private static final Map<String, Level> LOG_LEVELS = new HashMap<>();

    /**
     * deviceId
     */
    private static final ThreadLocal<String> THREAD_LOCAL = new ThreadLocal<>();

    /**
     *
     */
    private BlockLog() {

```
}

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

/**
 * DEBUG/INFO/WARN/ERROR/FATAL 5
 */
static {
    LOG_LEVELS.put("DEBUG", Level.DEBUG);
    LOG_LEVELS.put("INFO", Level.INFO);
    LOG_LEVELS.put("WARN", Level.WARN);
    LOG_LEVELS.put("ERROR", Level.ERROR);
}

/**
 * debug
 *
 * @param msg
 */
public static void debug(String msg) {
    if (LOG.isDebugEnabled()) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.debug(logContent);
    }
}

public static void debug(String msg, Object... objs) {
    if (LOG.isDebugEnabled()) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.debug(logContent, objs);
    }
}

/**
 * debug
 *
 * @param msg
```

```
     * @param throwable
     */
    public static void debug(String msg, Throwable throwable) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//            if (!(throwable instanceof NulsException) || !(throwable instanceof
NulsRuntimeException)) {
//                throwable = new NulsException(ErrorCode.FAILED, throwable);
//            }
            LOG.debug(logContent, throwable);
        }
    }

    private static boolean isStringBlank(String val) {
        return null == val || val.trim().isEmpty();
    }

    /**
     *
     *
     * @return
     */
    private static String getLogTrace() {
        StringBuilder logTrace = new StringBuilder();
        StackTraceElement stack[] = Thread.currentThread().getStackTrace();
        if (stack.length > 1) {
            // index3index12Logindex0
            StackTraceElement ste = stack[3];
            if (ste != null) {
//                logTrace.append("[" + DateUtil.convertDate(new
Date(TimeService.currentTimeMillis())) + "]");
                //
                logTrace.append(ste.getClassName());
                logTrace.append('.');
                logTrace.append(ste.getMethodName());
                logTrace.append('(');
                logTrace.append(ste.getFileName());
                logTrace.append(':');
                logTrace.append(ste.getLineNumber());
                logTrace.append(')');
            }
```

```
        }
        logTrace.append("\n");
        return logTrace.toString();
    }

//    /**
//     *
//     *
//     * @param level
//     */
//    public static void setLevel(String level) {
//        if (LOG_LEVELS.containsKey(level.toUpperCase())) {
//            LOG.setLevel(LOG_LEVELS.get(level.toUpperCase()));
//        }
//    }

    /**
     * 16
     *
     * @return 16
     */
    private static String getId() {
        return THREAD_LOCAL.get();
    }

    /**
     *
     *
     * @param id
     */
    public static void setId(String id) {
        THREAD_LOCAL.set(id);
    }

    public static void removeId() {
        THREAD_LOCAL.remove();
    }
}

48:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\log\ChainLog.java
 */
```

```
package io.nuls.core.tools.log;


import ch.qos.logback.classic.Level;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

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

/**
 *  <br>
 *
 * slf4j
 *
 * @author Niels
 */
public final class ChainLog {
    /**
     *
     */
    private static final Logger LOG = LoggerFactory.getLogger("chainLog");

    /**
     *
     */
    private static final Map<String, Level> LOG_LEVELS = new HashMap<>();

    /**
     * deviceId
     */
    private static final ThreadLocal<String> THREAD_LOCAL = new ThreadLocal<>();

    /**
     *
     */
    private ChainLog() {
    }

    @Override
    public String toString() {
        return super.toString();
```

```
}

/**
 * DEBUG/INFO/WARN/ERROR/FATAL 5
 */
static {
    LOG_LEVELS.put("DEBUG", Level.DEBUG);
    LOG_LEVELS.put("INFO", Level.INFO);
    LOG_LEVELS.put("WARN", Level.WARN);
    LOG_LEVELS.put("ERROR", Level.ERROR);
}

/**
 * debug
 *
 * @param msg
 */
public static void debug(String msg) {
    if (LOG.isDebugEnabled()) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.debug(logContent);
    }
}

public static void debug(String msg, Object... objs) {
    if (LOG.isDebugEnabled()) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.debug(logContent, objs);
    }
}

/**
 * debug
 *
 * @param msg
 * @param throwable
 */
public static void debug(String msg, Throwable throwable) {
    if (LOG.isDebugEnabled()) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
```

```
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//            if (!(throwable instanceof NulsException) || !(throwable instanceof
NulsRuntimeException)) {
//                throwable = new NulsException(ErrorCode.FAILED, throwable);
//            }
            LOG.debug(logContent, throwable);
        }
    }

    private static boolean isStringBlank(String val) {
        return null == val || val.trim().isEmpty();
    }

    /**
     *
     *
     * @return
     */
    private static String getLogTrace() {
        StringBuilder logTrace = new StringBuilder();
        StackTraceElement stack[] = Thread.currentThread().getStackTrace();
        if (stack.length > 1) {
            // index3index12Logindex0
            StackTraceElement ste = stack[3];
            if (ste != null) {
//                logTrace.append("[" + DateUtil.convertDate(new
Date(TimeService.currentTimeMillis())) + "]");
                //
                logTrace.append(ste.getClassName());
                logTrace.append('.');
                logTrace.append(ste.getMethodName());
                logTrace.append('(');
                logTrace.append(ste.getFileName());
                logTrace.append(':');
                logTrace.append(ste.getLineNumber());
                logTrace.append(')');
            }
        }
        logTrace.append("\n");
        return logTrace.toString();
    }
```

```
//    /**
//     *
//     *
//     * @param level
//     */
//    public static void setLevel(String level) {
//        if (LOG_LEVELS.containsKey(level.toUpperCase())) {
//            LOG.setLevel(LOG_LEVELS.get(level.toUpperCase()));
//        }
//    }

    /**
     * 16
     *
     * @return 16
     */
    private static String getId() {
        return THREAD_LOCAL.get();
    }

    /**
     *
     *
     * @param id
     */
    public static void setId(String id) {
        THREAD_LOCAL.set(id);
    }

    public static void removeId() {
        THREAD_LOCAL.remove();
    }
}

49:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\log\ConsensusLog.java
 */
package io.nuls.core.tools.log;


import ch.qos.logback.classic.Level;
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

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

/**
 *  <br>
 *
 * slf4j
 *
 * @author Niels
 */
public final class ConsensusLog {
    /**
     *
     */
    private static final Logger LOG = LoggerFactory.getLogger("consensusLog");

    /**
     *
     */
    private static final Map<String, Level> LOG_LEVELS = new HashMap<>();

    /**
     * deviceId
     */
    private static final ThreadLocal<String> THREAD_LOCAL = new ThreadLocal<>();

    /**
     *
     */
    private ConsensusLog() {
    }

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

    /**
     * DEBUG/INFO/WARN/ERROR/FATAL 5
     */
```

```
    static {
        LOG_LEVELS.put("DEBUG", Level.DEBUG);
        LOG_LEVELS.put("INFO", Level.INFO);
        LOG_LEVELS.put("WARN", Level.WARN);
        LOG_LEVELS.put("ERROR", Level.ERROR);
    }

    /**
     * debug
     *
     * @param msg
     */
    public static void debug(String msg) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
            LOG.debug(logContent);
        }
    }

    public static void debug(String msg, Object... objs) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
            LOG.debug(logContent, objs);
        }
    }

    /**
     * debug
     *
     * @param msg
     * @param throwable
     */
    public static void debug(String msg, Throwable throwable) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//            if (!(throwable instanceof NulsException) || !(throwable instanceof
NulsRuntimeException)) {
//                throwable = new Exception( throwable);
//            }
```

```
            LOG.debug(logContent, throwable);
        }
    }

    private static boolean isStringBlank(String val) {
        return null == val || val.trim().isEmpty();
    }

    /**
     *
     *
     * @return
     */
    private static String getLogTrace() {
        StringBuilder logTrace = new StringBuilder();
        StackTraceElement stack[] = Thread.currentThread().getStackTrace();
        if (stack.length > 1) {
            // index3index12Logindex0
            StackTraceElement ste = stack[3];
            if (ste != null) {
//                logTrace.append("[" + DateUtil.convertDate(new
Date(TimeService.currentTimeMillis())) + "]");
                //
                logTrace.append(ste.getClassName());
                logTrace.append('.');
                logTrace.append(ste.getMethodName());
                logTrace.append('(');
                logTrace.append(ste.getFileName());
                logTrace.append(':');
                logTrace.append(ste.getLineNumber());
                logTrace.append(')');
            }
        }
        return logTrace.toString();
    }

//    /**
//     *
//     *
//     * @param level
//     */
//    public static void setLevel(String level) {
```

```
//        if (LOG_LEVELS.containsKey(level.toUpperCase())) {
//            LOG.setLevel(LOG_LEVELS.get(level.toUpperCase()));
//        }
//    }

    /**
     * 16
     *
     * @return 16
     */
    private static String getId() {
        return THREAD_LOCAL.get();
    }

    /**
     *
     *
     * @param id
     */
    public static void setId(String id) {
        THREAD_LOCAL.set(id);
    }

    public static void removeId() {
        THREAD_LOCAL.remove();
    }
}
```

50:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\log\Log.java

```
 */
package io.nuls.core.tools.log;



import ch.qos.logback.classic.Level;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

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

/**
 *  <br>
```

```
 *
 * slf4j
 *
 * @author Niels
 */
public final class Log {
    /**
     *
     */
    private static final Logger LOG = LoggerFactory.getLogger(Log.class.getName());

    /**
     *
     */
    private static final Map<String, Level> LOG_LEVELS = new HashMap<>();

    /**
     * deviceId
     */
    private static final ThreadLocal<String> THREAD_LOCAL = new ThreadLocal<>();

    /**
     *
     */
    private Log() {
    }

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

    /**
     * DEBUG/INFO/WARN/ERROR/FATAL 5
     */
    static {
        LOG_LEVELS.put("DEBUG", Level.DEBUG);
        LOG_LEVELS.put("INFO", Level.INFO);
        LOG_LEVELS.put("WARN", Level.WARN);
        LOG_LEVELS.put("ERROR", Level.ERROR);
    }
```

```
    /**
     * debug
     *
     * @param msg
     */
    public static void debug(String msg) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
            LOG.debug(logContent);
        }
    }

    public static void debug(String msg, Object... objs) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
            LOG.debug(logContent, objs);
        }
    }

    /**
     * debug
     *
     * @param msg
     * @param throwable
     */
    public static void debug(String msg, Throwable throwable) {
        if (LOG.isDebugEnabled()) {
            String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                    : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//            if (!(throwable instanceof NulsException) || !(throwable instanceof
NulsRuntimeException)) {
//                throwable = new NulsException(ErrorCode.FAILED, throwable);
//            }
            LOG.debug(logContent, throwable);
        }
    }

    /**
     * info
     *
```

```
     * @param msg
     */
    public static void info(String msg) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.info(logContent);
    }

    public static void info(String msg, Object... objs) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.info(logContent, objs);
    }

    /**
     * info
     *
     * @param msg
     * @param throwable
     */
    public static void info(String msg, Throwable throwable) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//        if (!(throwable instanceof NulsException) || !(throwable instanceof NulsRuntimeException)) {
//            throwable = new NulsException(ErrorCode.FAILED, throwable);
//        }
        LOG.info(logContent, throwable);
    }

    /**
     * warn
     *
     * @param msg
     */
    public static void warn(String msg) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.warn(logContent);
    }

    public static void warn(String msg, Object... objs) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
```

```
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.warn(logContent, objs);
    }

    /**
     * warn
     *
     * @param msg
     * @param throwable
     */
    public static void warn(String msg, Throwable throwable) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//        if (!(throwable instanceof NulsException) || !(throwable instanceof NulsRuntimeException)) {
//            throwable = new NulsException(ErrorCode.FAILED, throwable);
//        }
        LOG.warn(logContent, throwable);
    }

    /**
     * error
     *
     * @param msg
     */
    public static void error(String msg) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.error(logContent);
    }


    public static void error(String msg, Object... objs) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.error(logContent, objs);
    }

    /**
     * error
     *
     * @param msg
     * @param throwable
```

```
     */
    public static void error(String msg, Throwable throwable) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//        if (!(throwable instanceof NulsException) || !(throwable instanceof NulsRuntimeException)) {
//            throwable = new NulsException(ErrorCode.FAILED, throwable);
//        }
        LOG.error(logContent, throwable);
    }

    public static void error(Throwable throwable) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":")
                : (getLogTrace() + "[" + getId() + "]" + ":");
//        if (!(throwable instanceof NulsException) || !(throwable instanceof NulsRuntimeException)) {
//            throwable = new NulsException(ErrorCode.FAILED, throwable);
//        }
        LOG.error(logContent, throwable);
    }

    /**
     * trace
     *
     * @param msg
     */
    public static void trace(String msg) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
        LOG.trace(logContent);
    }

    /**
     * trace
     *
     * @param msg
     * @param throwable
     */
    public static void trace(String msg, Throwable throwable) {
        String logContent = isStringBlank(getId()) ? (getLogTrace() + ":" + msg)
                : (getLogTrace() + "[" + getId() + "]" + ":" + msg);
//        if (!(throwable instanceof NulsException) || !(throwable instanceof NulsRuntimeException)) {
//            throwable = new NulsException(ErrorCode.FAILED, throwable);
//        }
```

```
        LOG.trace(logContent, throwable);
    }


    private static boolean isStringBlank(String val) {
        return null == val || val.trim().isEmpty();
    }

    /**
     *
     *
     * @return
     */
    private static String getLogTrace() {
        StringBuilder logTrace = new StringBuilder();
        StackTraceElement stack[] = Thread.currentThread().getStackTrace();
        if (stack.length > 1) {
            // index3index12Logindex0
            StackTraceElement ste = stack[3];
            if (ste != null) {
                //
                logTrace.append(ste.getClassName());
                logTrace.append('.');
                logTrace.append(ste.getMethodName());
                logTrace.append('(');
                logTrace.append(ste.getFileName());
                logTrace.append(':');
                logTrace.append(ste.getLineNumber());
                logTrace.append(')');
            }
        }
        return logTrace.toString();
    }

//    /**
//     *
//     *
//     * @param level
//     */
//    public static void setLevel(String level) {
//        if (LOG_LEVELS.containsKey(level.toUpperCase())) {
//            LOG.setLevel(LOG_LEVELS.get(level.toUpperCase()));
```

```
//        }
//    }

    /**
     * 16
     *
     * @return 16
     */
    private static String getId() {
        return THREAD_LOCAL.get();
    }

    /**
     *
     *
     * @param id
     */
    public static void setId(String id) {
        THREAD_LOCAL.set(id);
    }

    public static void removeId() {
        THREAD_LOCAL.remove();
    }

    public static boolean isDebugEnabled() {
        return LOG.isDebugEnabled();
    }
}
```

51:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\map\MapUtil.java

```
package io.nuls.core.tools.map;

import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedHashMap;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

/**
 * @author: PierreLuo
```

```
 */
public class MapUtil{

    private static final int MAXIMUM_CAPACITY = 1 << 30;

    public static int tableSizeFor(int cap) {
        int n = cap - 1;
        n |= n >>> 1;
        n |= n >>> 2;
        n |= n >>> 4;
        n |= n >>> 8;
        n |= n >>> 16;
        return (n < 0) ? 1 : (n >= MAXIMUM_CAPACITY) ? MAXIMUM_CAPACITY : n + 1;
    }

    public static HashMap createHashMap(int cap) {
        int capacity = tableSizeFor(cap) << 1;
        return new HashMap<>(capacity);
    }

    public static LinkedHashMap createLinkedHashMap(int cap) {
        int capacity = tableSizeFor(cap) << 1;
        return new LinkedHashMap<>(capacity);
    }

    public static ConcurrentHashMap createConcurrentHashMap(int cap) {
        int capacity = tableSizeFor(cap) << 1;
        return new ConcurrentHashMap<>(capacity);
    }

    public static HashSet createHashSet(int cap) {
        int capacity = tableSizeFor(cap) << 1;
        return new HashSet<>(capacity);
    }

    public static Set createConcurrentHashSet(int cap) {
        int capacity = tableSizeFor(cap) << 1;
        return ConcurrentHashMap.newKeySet(capacity);
    }

}
```

```
52:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\network\IpUtil.java
 */
package io.nuls.core.tools.network;

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

import java.net.*;
import java.util.*;
import java.util.regex.Pattern;

/**
 * @author vivi
 */
public class IpUtil {
    private static final Pattern pattern = Pattern.compile("\\<dd class\\=\"fz24\">(.*?)\\<\\/dd>");

    private static final Set<String> ips = new HashSet<>();

    static {
        List<String> localIPs = getLocalIP();
        for (String ip : localIPs) {
            ips.add(ip);
        }
    }

    public static Set<String> getIps() {
        return ips;
    }

    private static ArrayList<String> getLocalIP() {
        ArrayList<String> iplist = new ArrayList<>();
        boolean loop = false;
        String bindip;
        Enumeration<?> network;
        List<NetworkInterface> netlist = new ArrayList<>();
        try {
            network = NetworkInterface.getNetworkInterfaces();
            while (network.hasMoreElements()) {
                loop = true;
                NetworkInterface ni = (NetworkInterface) network.nextElement();
                if (ni.isLoopback()) {
```

```
    continue;
}
netlist.add(0, ni);
InetAddress ip;
for (NetworkInterface list : netlist) {
    if (loop == false) {
        break;
    }
    Enumeration<?> card = list.getInetAddresses();
    while (card.hasMoreElements()) {
        while (true) {
            ip = null;
            try {
                ip = (InetAddress) card.nextElement();
            } catch (Exception e) {

            }
            if (ip == null) {
                break;
            }
            if (!ip.isLoopbackAddress()) {
                if ("127.0.0.1".equalsIgnoreCase(ip.getHostAddress())) {
                    continue;
                }
            }
            if (ip instanceof Inet6Address) {
                continue;
            }
            if (ip instanceof Inet4Address) {
                bindip = ip.getHostAddress();
                boolean addto = true;
                for (int n = 0; n < iplist.size(); n++) {
                    if (bindip.equals(iplist.get(n))) {
                        addto = false;
                        break;
                    }
                }
                if (addto) {
                    iplist.add(bindip);
                }
                break;
            }
```

```
                }
            }
        }
    }
    } catch (SocketException e) {
        // skip
        Log.error("Get local IP error: " + e.getMessage());
    }

    return iplist;
}

public static boolean judgeLocalIsServer(String localIP, String remoteIP) {
    long local = ipToLong(localIP);
    long remote = ipToLong(remoteIP);
    if (local < remote) {
        return true;
    }
    return false;
}

public static long ipToLong(String ipAddress) {
    long result = 0;
    String[] ipAddressInArray = ipAddress.split("\\.");
    for (int i = 3; i >= 0; i--) {
        long ip = Long.parseLong(ipAddressInArray[3 - i]);
        //left shifting 24,16,8,0 and bitwise OR
        //1. 192 << 24
        //1. 168 << 16
        //1. 1   << 8
        //1. 2   << 0
        result |= ip << (i * 8);
    }
    return result;
}

public static String getNodeId(InetSocketAddress socketAddress) {
    if (socketAddress == null) {
        return null;
    }
    return socketAddress.getHostString() + ":" + socketAddress.getPort();
}
```

```
}

53:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\network\RequestUtil.java
 */
package io.nuls.core.tools.network;

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

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

/**
 * @author win10
 */
public class RequestUtil {

    public static byte[] get(String url) {

        InputStream is = null;
        try {
            URL u = new URL(url);
            is = u.openStream();
            byte[] content = readInputStream(is);
            return content;
        } catch (Exception ex) {
            System.err.println(ex);
        } finally {
            if (is != null) {
                try {
                    is.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }

        return null;
    }
```

```
//    /**
//     *
//     *
//     * @return byte[]
//     */
    public static byte[] readInputStream(InputStream inputStream) throws IOException {
        byte[] buffer = new byte[1024];
        int len = 0;
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        while ((len = inputStream.read(buffer)) != -1) {
            bos.write(buffer, 0, len);
        }
        bos.close();
        return bos.toByteArray();
    }

    public static String post(String url, final String param) {
        return post(url, param, null);
    }

    public static String post(String url, final String param, String encoding) {
        StringBuffer sb = new StringBuffer();
        OutputStream os = null;
        InputStream is = null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        // UTF-8
        if (StringUtils.isNull(encoding)) {
            encoding = "UTF-8";
        }
        try {
            URL u = new URL(url);
            URLConnection connection = u.openConnection();
            connection.setDoOutput(true);
            os = connection.getOutputStream();
            os.write(param.getBytes(encoding));
            os.flush();
            is = connection.getInputStream();
            isr = new InputStreamReader(is, encoding);
            br = new BufferedReader(isr);
            String line;
            while ((line = br.readLine()) != null) {
```

```
                sb.append(line);
                sb.append("\n");
            }
        } catch (Exception ex) {
            System.err.println(ex);
        } finally {
            if (is != null) {
                try {
                    is.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (os != null) {
                try {
                    os.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (isr != null) {
                try {
                    isr.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }
        return sb.toString();
    }

//    /**
//     * get
//     *
//     * @return String
```

```
//     */
    public static String doGet(String url, String encoding) {
        StringBuffer sb = new StringBuffer();
        InputStreamReader is = null;
        BufferedReader br = null;
        // NulsConstant.D
        if (StringUtils.isNull(encoding)) {
            encoding = "UTF-8";
        }
        try {
            URL u = new URL(url);
            is = new InputStreamReader(u.openStream(), encoding);
            br = new BufferedReader(is);
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line);
                sb.append("\n");
            }
        } catch (Exception ex) {
            System.err.println(ex);
        } finally {
            if (is != null) {
                try {
                    is.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }
        return sb.toString();
    }

    public static String postCustomer(String url, final String param) {
        StringBuffer sb = new StringBuffer();
        OutputStream os = null;
```

```
InputStream is = null;
InputStreamReader isr = null;
BufferedReader br = null;
String encoding = null;

// NulsConstant.D
if (StringUtils.isNull(encoding)) {
    encoding = "NulsConstant.D";
}
try {
    URL u = new URL(url);
    URLConnection connection = u.openConnection();
    connection.setRequestProperty("Content-Type", "text/html");
    connection.setDoOutput(true);
    os = connection.getOutputStream();
    os.write(param.getBytes(encoding));
    os.flush();
    is = connection.getInputStream();
    isr = new InputStreamReader(is, encoding);
    br = new BufferedReader(isr);
    String line;
    while ((line = br.readLine()) != null) {
        sb.append(line);
        sb.append("\n");
    }
} catch (Exception ex) {
    System.err.println(ex);
} finally {
    if (is != null) {
        try {
            is.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
    if (os != null) {
        try {
            os.close();
        } catch (IOException e) {
            Log.error(e);
        }
    }
```

```
            if (isr != null) {
                try {
                    isr.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
            if (br != null) {
                try {
                    br.close();
                } catch (IOException e) {
                    Log.error(e);
                }
            }
        }
        return sb.toString();
    }
}

54:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\page\Page.java
 */

package io.nuls.core.tools.page;

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

public class Page<T> {

    private int pageNumber;

    private int pageSize;

    private long total;

    private int pages;

    private List<T> list;

    public Page() {
        this.list = new ArrayList<>();
```

```
}

public Page(int pageNumber, int pageSize) {
    this();
    this.pageNumber = pageNumber;
    this.pageSize = pageSize;
}

public Page(int pageNumber, int pageSize, int total) {
    this(pageNumber, pageSize);
    this.total = total;

    int p = total / pageSize;
    if (total % pageSize > 0) {
        p++;
    }
    this.pages = p;
}

public Page(Page page) {
    this.pageNumber = page.getPageNumber();
    this.pageSize = page.getPageSize();
    this.total = page.getTotal();
    this.pages = page.getPages();
    this.list = new ArrayList<>();
}

public int getPageNumber() {
    return pageNumber;
}

public void setPageNumber(int pageNumber) {
    this.pageNumber = pageNumber;
}

public int getPageSize() {
    return pageSize;
}

public void setPageSize(int pageSize) {
    this.pageSize = pageSize;
}
```

```
    public long getTotal() {
        return total;
    }

    public void setTotal(long total) {
        this.total = total;
        if (pageSize != 0) {
            this.pages = (int) (total / pageSize);
            if (total % pageSize != 0) {
                this.pages++;
            }
        }
    }

    public int getPages() {
        return pages;
    }

    public void setPages(int pages) {
        this.pages = pages;
    }

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

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

55:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\param\AssertUtil.java
 */
package io.nuls.core.tools.param;


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

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

```
/**
 * @author Niels
 */
public final class AssertUtil {

    public static void isEquals(Object val1, Object val2, String msg) {
        if (val1 == val2 || (val1 != null && val1.equals(val2))) {
            return;
        }
        throw new RuntimeException(msg);
    }

    public static void canNotEmpty(Object val) {
        canNotEmpty(val, "null parameter");
    }

    public static void canNotEmpty(Object val, String msg) {
        boolean b = false;
        do {
            if (null == val) {
                b = true;
                break;
            }
            if (val instanceof String) {
                b = StringUtils.isBlank(val + "");
                break;
            }
            if (val instanceof List) {
                b = ((List) val).isEmpty();
                break;
            }
            if (val instanceof Map) {
                b = ((Map) val).isEmpty();
                break;
            }
            if (val instanceof String[]) {
                b = ((String[]) val).length == 0;
                break;
            }
            if (val instanceof byte[]) {
                b = ((byte[]) val).length == 0;
```

```
                break;
            }
        } while (false);
        if (b) {
            throw new RuntimeException(msg);
        }
    }
}
```

56:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\main\java\io\nuls\core\tools\str\StringUtils.java

```
 */
package io.nuls.core.tools.str;

import java.io.UnsupportedEncodingException;
import java.util.UUID;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import static java.nio.charset.StandardCharsets.UTF_8;

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

    /**
     * NULS
     */
    private static final String NULS = "NULS";

    public static final String EMPTY = "";

    public static boolean isBlank(String str) {
        return null == str || str.trim().length() == 0;
    }

    public static boolean isNull(String str) {
        return null == str || str.trim().length() == 0 || "null".equalsIgnoreCase(str.trim());
    }

    public static boolean isNotBlank(String str) {
```

```
    return !isBlank(str);
}

public static boolean isNotNull(String str) {
    return !isNull(str);
}

public static String getNewUUID() {
    return UUID.randomUUID().toString().replaceAll("-", "");
}

public static String formatStringPara(String para) {
    return (isNull(para)) ? null : para.trim();
}

/**
 *  Check the difficulty of the password
 *  length between 8 and 20, the combination of characters and numbers
 *
 * @return boolean
 */
public static boolean validPassword(String password) {
    if (isBlank(password)) {
        return false;
    }
    if (password.length() < 8 || password.length() > 20) {
        return false;
    }
    if (password.matches("(.*)[a-zA-z](.*)")
            && password.matches("(.*)\\d+(.*)")
            && !password.matches("(.*)\\s+(.*)")
            && !password.matches("(.*)[\u4e00-\u9fa5\u3000]+(.*)")) {
        return true;
    } else {
        return false;
    }
}

/**
 * :1~20
 * @param alias
 * @return
```

```
 */
public static boolean validAlias(String alias) {
    try {
        if (isBlank(alias)) {
            return false;
        }
        alias = alias.trim();
        byte[] aliasBytes = alias.getBytes("UTF-8");
        if (aliasBytes.length < 1 || aliasBytes.length > 20) {
            return false;
        }
        if (alias.matches("^([a-z0-9]+[a-z0-9_]*[a-z0-9]+)|[a-z0-9]+${1,20}")) {
            return true;
        } else {
            return false;
        }
    } catch (UnsupportedEncodingException e) {
        return false;
    }
}

/**
 * token:1~20
 * @param name
 * @return
 */
public static boolean validTokenNameOrSymbol(String name) {
    try {
        if (isBlank(name)) {
            return false;
        }

        String upperCaseName = name.toUpperCase();
        if(upperCaseName.contains(NULS)) {
            return false;
        }

        byte[] aliasBytes = name.getBytes("UTF-8");
        if (aliasBytes.length < 1 || aliasBytes.length > 20) {
            return false;
        }
        if (name.matches("^([a-zA-Z0-9]+[a-zA-Z0-9_]*[a-zA-Z0-9]+)|[a-zA-Z0-9]+${1,20}")) {
```

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

/**
 * : ,60
 * @param remark
 * @return
 */
public static boolean validRemark(String remark) {
    try {
        if (null == remark) {
            return true;
        }
        remark = remark.trim();
        byte[] aliasBytes = remark.getBytes("UTF-8");
        if (aliasBytes.length < 0 || aliasBytes.length > 60) {
            return false;
        }
        return true;
    } catch (UnsupportedEncodingException e) {
        return false;
    }
}

public static byte caculateXor(byte[] data) {
    byte xor = 0x00;
    if (data == null || data.length == 0) {
        return xor;
    }
    for (int i = 0; i < data.length; i++) {
        xor ^= data[i];
    }
    return xor;
}

public static boolean validAddressSimple(String address) {
```

```
        if (isBlank(address)) {
            return false;
        }
        if (address.length() > 40) {
            return false;
        }
        return true;
    }

    public static boolean isNumeric(String str) {
        for (int i = 0, len = str.length(); i < len; i++) {
            if (!Character.isDigit(str.charAt(i))) {
                return false;
            }
        }
        return true;
    }

    private static final Pattern NUMBER_PATTERN = Pattern.compile("-?[0-9]+(\\.[0-9]+)?");

    public static boolean isNumber(String str) {
        if (StringUtils.isBlank(str)) {
            return false;
        }
        Matcher isNum = NUMBER_PATTERN.matcher(str);
        if (!isNum.matches()) {
            return false;
        }
        return true;
    }


    private static final Pattern GT_ZERO_NUMBER_PATTERN = Pattern.compile("([1-9][0-
9]*(\\.\\d+)?)|(0\\.\\d*[1-9]+0*)");

//    /**
//     * 0(,)
//     * @param str
//     * @return
//     */
    public static boolean isNumberGtZero(String str) {
        if (StringUtils.isBlank(str)) {
```

```
            return false;
        }
        Matcher isNum = GT_ZERO_NUMBER_PATTERN.matcher(str);
        if (!isNum.matches()) {
            return false;
        }
        return true;
    }

//    /**
//     * .0
//     * @param s
//     * @return
//     */
    private static String subZeroAndDot(String s){
        if(s.indexOf(".") > 0){
            s = s.replaceAll("0+?$", "");
            s = s.replaceAll("[.]$", "");
        }
        return s;
    }

    private static final Pattern NULS_PATTERN = Pattern.compile("([1-
9]\\d*(\\.\\d{1,8})?)|(0\\.\\d{1,8})");

//    /**
//     * nuls
//     * 0(, 8)
//     * @param str
//     * @return
//     */
    public static boolean isNuls(String str) {
        if (StringUtils.isBlank(str)) {
            return false;
        }
        str = subZeroAndDot(str);
        Matcher isNum = NULS_PATTERN.matcher(str);
        if (!isNum.matches()) {
            return false;
        }
        return true;
    }
```

```
    private static final Pattern GT_ZERO_NUMBER_LIMIT_2_PATTERN = Pattern.compile("([1-9]\\d*(\\.\\d{1,2})?)|(0\\.\\d{1,2})");

//    /**
//     * 0(, 2)
//     * @param str
//     * @return
//     */
    public static boolean isNumberGtZeroLimitTwo(String str) {
        if (StringUtils.isBlank(str)) {
            return false;
        }
        str = subZeroAndDot(str);
        Matcher isNum = GT_ZERO_NUMBER_LIMIT_2_PATTERN.matcher(str);
        if (!isNum.matches()) {
            return false;
        }
        return true;
    }

    public static byte[] bytes(String value) {
        return (value == null) ? null : value.getBytes(UTF_8);
    }

    public static String asString(byte[] value) {
        return (value == null) ? null : new String(value, UTF_8);
    }

    public static Long parseLong(Object obj) {
        if (obj == null) {
            return 0L;
        }
        String value = obj.toString();
        if (value.trim().length() == 0) {
            return 0L;
        }
        try {
            return Long.valueOf(value);
        } catch (Exception e) {
            return 0L;
        }
```

```
    }

    public static boolean validPubkeys(String pubkeys,String m){
        if(StringUtils.isBlank(pubkeys)){
            return  false;
        }
        if(m == null || Integer.parseInt(m) <= 0) {
            return false;
        }
        //
        String[] dataList = pubkeys.split(",");
        if(dataList == null || dataList.length == 0 || dataList.length < Integer.parseInt(m)){
            return false;
        }
        return true;
    }

    public static boolean validSign(String args[]){
        if(args.length != 3){
            return false;
        }
        if (StringUtils.isBlank(args[1])) {
            return false;
        }
        if(args[2] == null || args[2].length() == 0){
            return false;
        }
        return true;
    }
}

57:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-
module\tools\src\main\java\io\nuls\core\tools\str\VersionUtils.java
 */

package io.nuls.core.tools.str;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

```
/**
 * @author Niels
 */
public class VersionUtils {

    /**
     * @param version0 main version
     * @param version1 other version
     * @return boolean
     */
    public static boolean higherThan(String version0, String version1) {
        if (StringUtils.isBlank(version0) || StringUtils.isBlank(version1)) {
            throw new RuntimeException("version is null");
        }
        Integer[] intArr0 = strArrayToInt(version0);
        Integer[] intArr1 = strArrayToInt(version1);
        boolean result = false;
        for (int i = 0; i < intArr0.length; i++) {
            Integer val1 = intArr0[i];
            if (intArr1.length <= i && val1 > 0) {
                result = true;
                break;
            }
            Integer val2 = intArr1[i];
            if (val1 > val2) {
                result = true;
                break;
            }
        }
        //version1
        if (!result && intArr1.length > intArr0.length) {
            result = intArr1[intArr0.length] > 0;
        }
        return result;
    }

    public static boolean equalsWith(String version0, String version1) {
        if (StringUtils.isBlank(version0) || StringUtils.isBlank(version1)) {
            throw new RuntimeException("version is null");
        }
        Integer[] intArr0 = strArrayToInt(version0);
        Integer[] intArr1 = strArrayToInt(version1);
```

```
    boolean result = intArr0.length == intArr1.length;
    if (!result) {
        return false;
    }
    for (int i = 0; i < intArr0.length; i++) {
        Integer val1 = intArr0[i];
        Integer val2 = intArr1[i];
        if (val1 != val2) {
            return false;
        }
    }
    return true;
}

/**
 * @param version0 main version
 * @param version1 other version
 * @return boolean
 */
public static boolean lowerThan(String version0, String version1) {
    if (StringUtils.isBlank(version0) || StringUtils.isBlank(version1)) {
        throw new RuntimeException("version is null");
    }
    Integer[] intArr0 = strArrayToInt(version0);
    Integer[] intArr1 = strArrayToInt(version1);
    boolean result = false;
    for (int i = 0; i < intArr0.length; i++) {
        Integer val1 = intArr0[i];
        if (intArr1.length <= i && val1 > 0) {
            break;
        }
        Integer val2 = intArr1[i];
        if (val1 < val2) {
            result = true;
            break;
        }
    }
    if (!result && intArr1.length > intArr0.length && intArr1[intArr0.length] > 0) {
        result = true;
    }
    return result;
}
```

```
    private static Integer[] strArrayToInt(String version) {
        if (StringUtils.isBlank(version)) {
            return null;
        }
        Pattern pattern = Pattern.compile("\\d+");
        Matcher matcher = pattern.matcher(version);
        List<Integer> list = new ArrayList<>();
        while (matcher.find()) {
            list.add(Integer.parseInt(matcher.group(0)));
        }
        return list.toArray(new Integer[list.size()]);
    }

}
```

58:F:\git\coin\nuls\nuls-1.1.3\nuls\tools-module\tools\src\test\java\io\nuls\core\tools\str\VersionUtilsTest.java

```
 *  *
 *
 */

package io.nuls.core.tools.str;

import org.junit.Test;

import static org.junit.Assert.*;

/**
 * @author: Niels Wang
 * @date: 2018/10/17
 */
public class VersionUtilsTest {

    @Test
    public void higherThan() {
        String v1 = "1.1.0";
        String v2 = "1.1.0-beta";
        String v3 = "1.1.1";
        String v4 = "1.1.1-beta1";
        String v5 = "1.1.1-beta2";
        String v6 = "1.1.2-beta1";
```

```
        String v7 = "1.2.0";

        assertTrue(!VersionUtils.higherThan(v2, v1));
        assertTrue(VersionUtils.higherThan(v3, v2));
        assertTrue(VersionUtils.higherThan(v4, v3));
        assertTrue(VersionUtils.higherThan(v5, v4));
        assertTrue(VersionUtils.higherThan(v6, v5));
        assertTrue(VersionUtils.higherThan(v7, v1));
        assertTrue(VersionUtils.higherThan(v7, v2));
        assertTrue(VersionUtils.higherThan(v7, v6));


        assertTrue(VersionUtils.lowerThan(v2, v3));
        assertTrue(VersionUtils.lowerThan(v3, v4));
        assertTrue(VersionUtils.lowerThan(v4, v5));
        assertTrue(VersionUtils.lowerThan(v5, v6));
        assertTrue(VersionUtils.lowerThan(v1, v7));
        assertTrue(VersionUtils.lowerThan(v2, v7));
        assertTrue(VersionUtils.lowerThan(v6, v7));
        assertTrue(!VersionUtils.lowerThan(v2, v1));


        assertTrue(VersionUtils.equalsWith(v1, v2));
    }
}
```

59:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\constant\UtxoAccountsConstant.java

```
package io.nuls.utxo.accounts.constant;

import io.nuls.kernel.constant.NulsConstant;


public interface UtxoAccountsConstant extends NulsConstant {

    short MODULE_ID_UTXOACCOUNTS = 11;

    int TX_TYPE_REGISTER_AGENT = 4;
    int TX_TYPE_JOIN_CONSENSUS = 5;
    int TX_TYPE_CANCEL_DEPOSIT = 6;
    int TX_TYPE_YELLOW_PUNISH = 7;
    int TX_TYPE_RED_PUNISH = 8;
```

```
    int TX_TYPE_STOP_AGENT = 9;


    /**
     * CONTRACT
     */
    int TX_TYPE_CREATE_CONTRACT = 100;
    int TX_TYPE_CALL_CONTRACT = 101;
    int TX_TYPE_DELETE_CONTRACT = 102;

    /**
     * contract transfer
     */
    int TX_TYPE_CONTRACT_TRANSFER = 103;
}
```

60:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\constant\UtxoAccountsErrorCode.java

```
 */

package io.nuls.utxo.accounts.constant;


import io.nuls.kernel.constant.KernelErrorCode;

/**
 * @author: cody
 */
public interface UtxoAccountsErrorCode extends KernelErrorCode {
}
```

61:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\locker\Lockers.java

```
 */
package io.nuls.utxo.accounts.locker;

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

public class Lockers {
    public final static Lock SYN_UTXO_ACCOUNTS_LOCK = new ReentrantLock();
}
```

62:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\module\AbstractUtxoAccountsModule.java

```
 */
package io.nuls.utxo.accounts.module;


import io.nuls.kernel.module.BaseModuleBootstrap;
import io.nuls.utxo.accounts.constant.UtxoAccountsConstant;

public abstract class AbstractUtxoAccountsModule extends BaseModuleBootstrap {
    public AbstractUtxoAccountsModule() {
        super(UtxoAccountsConstant.MODULE_ID_UTXOACCOUNTS);
    }
}
```

63:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\module\impl\UtxoAccountsModuleBootstrap.java

```
 */
package io.nuls.utxo.accounts.module.impl;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.thread.manager.NulsThreadFactory;
import io.nuls.kernel.thread.manager.TaskManager;
import io.nuls.protocol.service.BlockService;
import io.nuls.utxo.accounts.constant.UtxoAccountsConstant;
import io.nuls.utxo.accounts.module.AbstractUtxoAccountsModule;
import io.nuls.utxo.accounts.service.UtxoAccountsService;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;
import io.nuls.utxo.accounts.task.UtxoAccountsThread;

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

public class UtxoAccountsModuleBootstrap extends AbstractUtxoAccountsModule {
    @Override
    public void init() throws Exception {
        Log.info("init utxoAccountsModule");
```

```
    }

    @Override
    public void start() {
        Log.info("start utxoAccountsModule");
        //
        try {
            UtxoAccountsService utxoAccountsService =
NulsContext.getServiceBean(UtxoAccountsService.class);
            UtxoAccountsStorageService utxoAccountsStorageService =
NulsContext.getServiceBean(UtxoAccountsStorageService.class);
            BlockService blockService = NulsContext.getServiceBean(BlockService.class);
            long hadSynBlockHeight = utxoAccountsStorageService.getHadSynBlockHeight();
            //
           if(!utxoAccountsService.validateIntegrityBootstrap(hadSynBlockHeight)){
               Log.error("start utxoAccountsModule fail."+hadSynBlockHeight);
               return;
           }
        }catch (Exception e){
            Log.error(e);
            Log.error("start utxoAccountsModule fail.");
            return;

        }
        ScheduledThreadPoolExecutor executor = TaskManager.createScheduledThreadPool(1,
new NulsThreadFactory(UtxoAccountsConstant.MODULE_ID_UTXOACCOUNTS,
"utxoAccountsThread"));
        executor.scheduleAtFixedRate(NulsContext.getServiceBean(UtxoAccountsThread.class), 15,
1, TimeUnit.SECONDS);

    }

    @Override
    public void shutdown() {
        Log.info("shutdown utxoAccountsModule");
    }

    @Override
    public void destroy() {
        Log.info("destroy utxoAccountsModule");
    }
```

```
    @Override
    public String getInfo() {
        Log.info("getInfo utxoAccountsModule");
        return null;
    }
}

64:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\service\impl\UtxoAccountsBalanceServiceImpl.java
 */
package io.nuls.utxo.accounts.service.impl;


import io.nuls.account.constant.AccountErrorCode;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.utxo.accounts.constant.UtxoAccountsErrorCode;
import io.nuls.utxo.accounts.model.UtxoAccountsBalance;
import io.nuls.utxo.accounts.service.UtxoAccountsBalanceService;
import io.nuls.utxo.accounts.storage.po.LockedBalance;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsBalancePo;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;

import java.math.BigDecimal;
import java.util.List;

public class UtxoAccountsBalanceServiceImpl implements UtxoAccountsBalanceService {
    @Autowired
    UtxoAccountsStorageService utxoAccountsStorageService;
    @Override
    public Result<UtxoAccountsBalance> getUtxoAccountsBalance(byte[] owner){
        if (!AddressTool.validAddress(AddressTool.getStringAddressByBytes(owner))) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR);
        }
        try {
            Result<UtxoAccountsBalancePo>
utxoAccountsBalance=utxoAccountsStorageService.getUtxoAccountsBalanceByAddress(owner);
```

```
            long synBlockHeight=utxoAccountsStorageService.getHadSynBlockHeight();
            if(null==utxoAccountsBalance || null==utxoAccountsBalance.getData()){
                return Result.getFailed(UtxoAccountsErrorCode.DATA_NOT_FOUND);
            }
            UtxoAccountsBalancePo dbAccountsBalance =utxoAccountsBalance.getData();
            UtxoAccountsBalance accountBalance=new UtxoAccountsBalance();
            accountBalance.setOwner(dbAccountsBalance.getOwner());
            long totalNa=dbAccountsBalance.getOutputBalance()-
(dbAccountsBalance.getInputBalance());
            totalNa+=dbAccountsBalance.getContractToBalance();
            totalNa-=dbAccountsBalance.getContractFromBalance();
            accountBalance.setBalance(Na.valueOf(totalNa));
            long permanentLockedNa=dbAccountsBalance.getLockedPermanentBalance()-
(dbAccountsBalance.getUnLockedPermanentBalance());
            long lockedNa=permanentLockedNa;
            List<LockedBalance> timeLockedBalance=dbAccountsBalance.getLockedTimeList();
            long currentTime=TimeService.currentTimeMillis();
            for(LockedBalance balance:timeLockedBalance){
                if(balance.getLockedTime()>currentTime){
                    lockedNa+=balance.getLockedBalance();
                }else{
                    break;
                }
            }
            List<LockedBalance> heightLockedBalance=dbAccountsBalance.getLockedHeightList();
            for(LockedBalance balance:heightLockedBalance){
                if(balance.getLockedTime()>synBlockHeight){
                    lockedNa+=balance.getLockedBalance();
                }else{
                    break;
                }
            }
            accountBalance.setHadLocked(Na.valueOf(lockedNa));
            return Result.getSuccess().setData(accountBalance);
        } catch (NulsException e) {
            Log.error(e);
        }
        return Result.getFailed(UtxoAccountsErrorCode.SYS_UNKOWN_EXCEPTION);
    }
}

65:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
```

base\src\main\java\io\nuls\utxo\accounts\service\impl\UtxoAccountsServiceImpl.java
 */
```
package io.nuls.utxo.accounts.service.impl;

import io.nuls.contract.dto.ContractResult;
import io.nuls.contract.dto.ContractTransfer;
import io.nuls.contract.service.ContractService;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.utxo.accounts.constant.UtxoAccountsConstant;
import io.nuls.utxo.accounts.service.UtxoAccountsService;
import io.nuls.utxo.accounts.storage.constant.UtxoAccountsStorageConstant;
import io.nuls.utxo.accounts.storage.po.LocalCacheBlockBalance;
import io.nuls.utxo.accounts.storage.po.LockedBalance;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsBalancePo;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;
import io.nuls.utxo.accounts.util.UtxoAccountsUtil;

import java.util.*;

@Service
public class UtxoAccountsServiceImpl implements UtxoAccountsService {

    @Autowired
    UtxoAccountsStorageService utxoAccountsStorageService;
    @Autowired
    ContractService contractService;

    private boolean isPermanentLocked(int txType) {
        if (txType == UtxoAccountsConstant.TX_TYPE_REGISTER_AGENT || txType ==
UtxoAccountsConstant.TX_TYPE_JOIN_CONSENSUS) {
            return true;
        }
```

```
        return false;
    }

    private boolean isPermanentUnLocked(int txType) {
        if (txType == UtxoAccountsConstant.TX_TYPE_CANCEL_DEPOSIT || txType ==
UtxoAccountsConstant.TX_TYPE_STOP_AGENT) {
            return true;
        }
        return false;
    }

    private byte[] getInputAddress(Coin from) {
        byte[] fromHash;
        int fromIndex;
        byte[] owner = from.getOwner();
        // ownertxHashindex
        fromHash = UtxoAccountsUtil.getTxHashBytes(owner);
        fromIndex = UtxoAccountsUtil.getIndex(owner);
        NulsDigestData fromHashObj = new NulsDigestData();
        try {
            fromHashObj.parse(fromHash, 0);
            Transaction outPutTx = utxoAccountsStorageService.getTx(fromHashObj);
            return outPutTx.getCoinData().getTo().get(fromIndex).getOwner();
        } catch (NulsException e) {
            Log.error(e);
            return null;
        }
    }

    private boolean buildUtxoAccountsMap(Map<String, UtxoAccountsBalancePo>
utxoAccountsMap, Block block) {
        List<Transaction> txs = block.getTxs();
        int txIndex = 0;
        for (Transaction tx : txs) {
            if (tx.getCoinData() == null) {
                return true;
            }
            List<Coin> from = tx.getCoinData().getFrom();
            List<Coin> to = tx.getCoinData().getTo();

            for (Coin inputCoin : from) {
                byte[] inputOwner = getInputAddress(inputCoin);
```

```
                inputCoin.setOwner(inputOwner);
                buildUtxoAccountsBalance(utxoAccountsMap, inputCoin, tx, txIndex, true);
            }
            for (Coin outputCoin : to) {
                buildUtxoAccountsBalance(utxoAccountsMap, outputCoin, tx, txIndex, false);
            }
            //()
            //
//            if (tx.getType() == UtxoAccountsConstant.TX_TYPE_CALL_CONTRACT) {
//                ContractResult contractExecuteResult =
contractService.getContractExecuteResult(tx.getHash());
//                List<ContractTransfer> transferList = contractExecuteResult.getTransfers();
//                buildContractTranfersBalance(utxoAccountsMap, transferList,
block.getHeader().getHeight(), txIndex);
//            }
            txIndex++;
        }
        return true;
    }

    /**
     * build utxoAccounts Map(address,balance)
     *
     * @param utxoAccountsMap
     * @param coin
     * @param tx
     * @param txIndex
     * @param isInput
     */
    private void buildUtxoAccountsBalance(Map<String, UtxoAccountsBalancePo>
utxoAccountsMap, Coin coin, Transaction tx, int txIndex, boolean isInput) {
        long netBlockHeight = NulsContext.getInstance().getNetBestBlockHeight();
        //change coin.getOwner() to coin.getAddress() for support multiSig
        String address = AddressTool.getStringAddressByBytes(coin.getAddress());
        UtxoAccountsBalancePo balance = utxoAccountsMap.get(address);
        if (null == balance) {
            balance = new UtxoAccountsBalancePo();
            balance.setOwner(coin.getOwner());
            utxoAccountsMap.put(address, balance);
        }
        if (isInput) {
            if (isPermanentUnLocked(tx.getType())) {
```

```
                //remove balance
                balance.setUnLockedPermanentBalance(balance.getUnLockedPermanentBalance() +
(coin.getNa().getValue()));
            }
            balance.setInputBalance(balance.getInputBalance() + (coin.getNa().getValue()));
        } else {
            if (isPermanentLocked(tx.getType())) {
                //add locked balance
                if (coin.getLockTime() == -1) {
                    balance.setLockedPermanentBalance(balance.getLockedPermanentBalance() +
(coin.getNa().getValue()));
                }
            } else {
                //add lockedTime output
                if (coin.getLockTime() > 0) {
                    //by time
                    if (coin.getLockTime() > TimeService.currentTimeMillis()) {
                        LockedBalance lockedBalance = new LockedBalance();
                        lockedBalance.setLockedBalance(coin.getNa().getValue());
                        lockedBalance.setLockedTime(coin.getLockTime());
                        balance.getLockedTimeList().add(lockedBalance);
                    } else {
                        //by height
                        if (coin.getLockTime() > netBlockHeight) {
                            LockedBalance lockedBalance = new LockedBalance();
                            lockedBalance.setLockedBalance(coin.getNa().getValue());
                            lockedBalance.setLockedTime(coin.getLockTime());
                            balance.getLockedHeightList().add(lockedBalance);
                        }
                    }

                }
            }
            balance.setOutputBalance(balance.getOutputBalance() + (coin.getNa()).getValue());
        }
//        balance.setOwner(coin.getOwner());
        balance.setBlockHeight(tx.getBlockHeight());
        balance.setTxIndex(txIndex);
    }

    /**
     * @param utxoAccountsMap
```

```
     * @param transferList
     * @param blockHeight
     * @param txIndex
     */
    private void buildContractTranfersBalance(Map<String, UtxoAccountsBalancePo>
utxoAccountsMap,
                                              List<ContractTransfer> transferList, long blockHeight, int txIndex) {
        for (ContractTransfer contractTransfer : transferList) {
            byte[] from = contractTransfer.getFrom();
            byte[] to = contractTransfer.getTo();
            String addressFrom = AddressTool.getStringAddressByBytes(from);
            String addressTo = AddressTool.getStringAddressByBytes(to);
            UtxoAccountsBalancePo balanceFrom = utxoAccountsMap.get(addressFrom);
            UtxoAccountsBalancePo balanceTo = utxoAccountsMap.get(addressTo);
            if (null == balanceFrom) {
                balanceFrom = new UtxoAccountsBalancePo();
                balanceFrom.setOwner(from);
                utxoAccountsMap.put(addressFrom, balanceFrom);
            }
            balanceFrom.setBlockHeight(blockHeight);
            balanceFrom.setTxIndex(txIndex);
            balanceFrom.setContractFromBalance(balanceFrom.getContractFromBalance() +
contractTransfer.getValue().getValue());
            if (null == balanceTo) {
                balanceTo = new UtxoAccountsBalancePo();
                balanceTo.setOwner(to);
                utxoAccountsMap.put(addressTo, balanceTo);
            }
            balanceTo.setBlockHeight(blockHeight);
            balanceTo.setTxIndex(txIndex);
            balanceTo.setContractToBalance(balanceTo.getContractToBalance() +
contractTransfer.getValue().getValue());
        }
    }

    /**
     * build utxoAccounts  to list
     *
     * @param utxoAccountsMap
     * @return
     * @throws NulsException
     */
```

```
    private List<UtxoAccountsBalancePo> utxoAccountsMapToList(Map<String,
UtxoAccountsBalancePo> utxoAccountsMap, LocalCacheBlockBalance preSnapshot)
            throws NulsException {
        List<UtxoAccountsBalancePo> list = new ArrayList<>();
        List<UtxoAccountsBalancePo> preList = new ArrayList<>();
        preSnapshot.setBalanceList(preList);
        Collection<UtxoAccountsBalancePo> utxoAccountsBalances = utxoAccountsMap.values();
        for (UtxoAccountsBalancePo balance : utxoAccountsBalances) {
            UtxoAccountsBalancePo localBalance =
utxoAccountsStorageService.getUtxoAccountsBalanceByAddress(balance.getOwner()).getData();
            if (localBalance == null) {
                list.add(balance);
                UtxoAccountsBalancePo preBalance = new UtxoAccountsBalancePo();
                preBalance.setOwner(balance.getOwner());
                preList.add(preBalance);
            } else {
                preList.add(localBalance);
                UtxoAccountsBalancePo newBalance = new UtxoAccountsBalancePo();
                newBalance.setOwner(localBalance.getOwner());
                newBalance.setBlockHeight(balance.getBlockHeight());
                newBalance.setTxIndex(balance.getTxIndex());
                newBalance.setOutputBalance(localBalance.getOutputBalance() +
(balance.getOutputBalance()));
                newBalance.setInputBalance(localBalance.getInputBalance() +
(balance.getInputBalance()));
                newBalance.setContractFromBalance(localBalance.getContractFromBalance() +
(balance.getContractFromBalance()));
                newBalance.setContractToBalance(localBalance.getContractToBalance() +
(balance.getContractToBalance()));
                newBalance.setLockedPermanentBalance(localBalance.getLockedPermanentBalance()
+ (balance.getLockedPermanentBalance()));
newBalance.setUnLockedPermanentBalance(localBalance.getUnLockedPermanentBalance() + (b
alance.getUnLockedPermanentBalance()));
                clearLockedBalance(localBalance, balance, newBalance);
                list.add(newBalance);
            }
        }
        return list;
    }

    private void clearLockedBalance(UtxoAccountsBalancePo dbBalance, UtxoAccountsBalancePo
addBalance, UtxoAccountsBalancePo newBalance) {
```

```
        List<LockedBalance> newTimeList = new ArrayList<>();
        List<LockedBalance> newHeightList = new ArrayList<>();
        List<LockedBalance> heightList = dbBalance.getLockedHeightList();
        List<LockedBalance> timeList = dbBalance.getLockedTimeList();
        long netBlockHeight = NulsContext.getInstance().getNetBestBlockHeight();
        for (LockedBalance heightBalance : heightList) {
            if (heightBalance.getLockedTime() > netBlockHeight) {
                newHeightList.add(heightBalance);
            } else {
                break;
            }
        }
        newHeightList.addAll(addBalance.getLockedHeightList());
        newHeightList.sort(LockedBalance::compareByLockedTime);
        newBalance.setLockedHeightList(newHeightList);
        long serverTime = TimeService.currentTimeMillis();
        for (LockedBalance timeBalance : timeList) {
            if (timeBalance.getLockedTime() > serverTime) {
                newTimeList.add(timeBalance);
            } else {
                break;
            }
        }
        newTimeList.addAll(addBalance.getLockedTimeList());
        newTimeList.sort(LockedBalance::compareByLockedTime);
        newBalance.setLockedTimeList(newTimeList);

    }


    public boolean rollbackBlock(LocalCacheBlockBalance block) throws NulsException {
        Log.info("rollbackBlock:" + block.getBlockHeight());
        if (block.getBlockHeight() == 0) {
            utxoAccountsStorageService.deleteLocalCacheBlock(block.getBlockHeight());
            return true;
        }
        //
        utxoAccountsStorageService.saveHadSynBlockHeight(block.getBlockHeight() - 1);
        LocalCacheBlockBalance localPreBlock =
utxoAccountsStorageService.getLocalCacheBlock(block.getBlockHeight()).getData();
        //
        List<UtxoAccountsBalancePo> list = localPreBlock.getBalanceList();
```

```
        if (list.size() > 0) {
            utxoAccountsStorageService.batchSaveByteUtxoAcountsInfo(list);
        }
        //
        utxoAccountsStorageService.deleteLocalCacheBlock(block.getBlockHeight());
        return true;
    }

    @Override
    public boolean validateIntegrityBootstrap(long hadSynBlockHeight) throws NulsException {
        Log.info("utxoAccountsModule validateIntegrityBootstrap hadSynBlockHeight:" +
hadSynBlockHeight);
        LocalCacheBlockBalance localCacheNextBlock = null;
        try {
            localCacheNextBlock =
utxoAccountsStorageService.getLocalCacheBlock(hadSynBlockHeight + 1).getData();
        } catch (NulsException e) {
            Log.error(e);
            return false;
        }
        if (localCacheNextBlock == null) {
            //
            return true;
        }
        List<UtxoAccountsBalancePo> utxoAccountsBalances =
localCacheNextBlock.getBalanceList();
        if (utxoAccountsBalances == null) {
            //
            return true;
        }
        return rollbackBlock(localCacheNextBlock);
    }



    /**
     * @param blockHeight
     * @return
     */
    @Override
    public boolean synBlock(long blockHeight) {
        Log.debug("synBlock begin===blockHeight:" + blockHeight);
        Block nodeBlock = utxoAccountsStorageService.getBlock(blockHeight).getData();
```

```
        if (nodeBlock == null) {
            Log.error("utxoAccounts getBlock faile,blockHeight:" + blockHeight);
            return false;
        }
        boolean hadRoll = false;
        try {
            //get local pre block info /
            LocalCacheBlockBalance localLatestCacheBlock =
utxoAccountsStorageService.getLocalCacheBlock(blockHeight - 1).getData();
            //rollback judge /
            while (localLatestCacheBlock != null &&
!nodeBlock.getHeader().getPreHash().equals(localLatestCacheBlock.getHash())) {
                //roll back info /
                rollbackBlock(localLatestCacheBlock);
                blockHeight--;
                //get pre block
                localLatestCacheBlock = utxoAccountsStorageService.getLocalCacheBlock(blockHeight
- 1).getData();
                nodeBlock = utxoAccountsStorageService.getBlock(blockHeight).getData();
                hadRoll = true;
            }
            if (hadRoll) {
                return false;
            }
        } catch (NulsException e) {
            Log.error(e);
            Log.error("block syn error======blockHeight:" + blockHeight);
            return false;
        }
        //begin syn block/
        //analysis block/
        Map<String, UtxoAccountsBalancePo> utxoAccountsMap = new HashMap<>();
        if (!buildUtxoAccountsMap(utxoAccountsMap, nodeBlock)) {
            return false;
        }
        List<UtxoAccountsBalancePo> list = new ArrayList<>();

        LocalCacheBlockBalance localCacheBlockBalance = new LocalCacheBlockBalance();
//        LocalCacheBlockBalance preSnapshot=new LocalCacheBlockBalance();
        try {
            list = utxoAccountsMapToList(utxoAccountsMap, localCacheBlockBalance);
        } catch (NulsException e) {
```

```
            Log.info("utxoAccountsMapToList error======blockHeight:" + blockHeight);
            return false;
        }
        localCacheBlockBalance.setHash(nodeBlock.getHeader().getHash());
        localCacheBlockBalance.setPreHash(nodeBlock.getHeader().getPreHash());
//        localCacheBlockBalance.setBalanceList(list);
        localCacheBlockBalance.setBlockHeight(blockHeight);
        //save cache block info/
        utxoAccountsStorageService.saveLocalCacheBlock(blockHeight, localCacheBlockBalance);
//
utxoAccountsStorageService.saveLocalCacheChangeSnapshot(blockHeight,localCacheBlockBala
nce);



        utxoAccountsStorageService.batchSaveByteUtxoAcountsInfo(list);
        //update latest block height/
        utxoAccountsStorageService.saveHadSynBlockHeight(blockHeight);

        //delete overdue cache data/
        if (blockHeight > UtxoAccountsStorageConstant.MAX_CACHE_BLOCK_NUM) {
            utxoAccountsStorageService.deleteLocalCacheBlock(blockHeight -
UtxoAccountsStorageConstant.MAX_CACHE_BLOCK_NUM);
        }
        Log.debug("utxoAccounts synBlock success==blockHeight:" + blockHeight);
        return true;
    }

}

66:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
base\src\main\java\io\nuls\utxo\accounts\service\UtxoAccountsService.java
 */
package io.nuls.utxo.accounts.service;

import io.nuls.kernel.exception.NulsException;

public interface UtxoAccountsService {
    boolean validateIntegrityBootstrap(long hadSynBlockHeight) throws NulsException;
//    public boolean validateBlock();
//    public boolean rollbackBlock();
    boolean  synBlock(long blockHeight);
}
```

```
67:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\task\UtxoAccountsThread.java
 */
package io.nuls.utxo.accounts.task;

import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.utxo.accounts.locker.Lockers;
import io.nuls.utxo.accounts.service.UtxoAccountsService;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;

@Component
public class UtxoAccountsThread implements Runnable {
    @Autowired
    private UtxoAccountsService utxoAccountsService;
    @Autowired
    private UtxoAccountsStorageService utxoAccountsStorageService;

    @Override
    public void run() {
        Lockers.SYN_UTXO_ACCOUNTS_LOCK.lock();
        try {
            long hadSynBlockHeight = utxoAccountsStorageService.getHadSynBlockHeight();
            long end = NulsContext.getInstance().getBestHeight();
            for (long i = hadSynBlockHeight + 1; i <= end; i++) {
                if (!utxoAccountsService.synBlock(i)) {
                    Log.error("utxoAccounts block syn fail!");
                    break;
                }
            }
        } catch (Exception e) {
            Log.error(e);
        } finally {
            Lockers.SYN_UTXO_ACCOUNTS_LOCK.unlock();
        }
    }

}
```

68:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\main\java\io\nuls\utxo\accounts\util\UtxoAccountsUtil.java

```
 */
package io.nuls.utxo.accounts.util;

import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.utils.VarInt;

public class UtxoAccountsUtil {
    private final static int TX_HASH_LENGTH = NulsDigestData.HASH_LENGTH;
    public static byte[] getTxHashBytes(byte[] fromBytes) {
        if(fromBytes == null || fromBytes.length < TX_HASH_LENGTH) {
            return null;
        }
        byte[] txBytes = new byte[TX_HASH_LENGTH];
        System.arraycopy(fromBytes, 0, txBytes, 0, TX_HASH_LENGTH);
        return txBytes;
    }

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

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

}
```

69:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-base\src\test\java\Test.java

```
70:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-rpc\src\main\java\io\nuls\utxo\accounts\rpc\cmd\GetUtxoAccountsProcessor.java
 */

package io.nuls.utxo.accounts.rpc.cmd;

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

/**
 * @author: cody
 */
public class GetUtxoAccountsProcessor implements CommandProcessor {

    private RestFulUtils restFul = RestFulUtils.getInstance();

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

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

    @Override
    public String getCommandDescription() {
        return "getutxoaccount <address> --get utxo account asset";
    }

    @Override
    public boolean argsValidate(String[] args) {
```

```
        if (args.length != 2) {
            return false;
        }
        if (!CommandHelper.checkArgsIsNull(args)) {
            return false;
        }
        return true;
    }

    @Override
    public CommandResult execute(String[] args) {
        String address = args[1];
        RpcClientResult result = restFul.get("/utxoAccounts/" + address, null);
        if(result.isFailed()){
            return CommandResult.getFailed(result);
        }
        return CommandResult.getResult(result);
    }
}



71:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
rpc\src\main\java\io\nuls\utxo\accounts\rpc\dto\AccountBalanceDto.java
 */
package io.nuls.utxo.accounts.rpc.dto;

import io.nuls.utxo.accounts.storage.po.LockedBalance;

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

public class AccountBalanceDto {
    private String address;
    private String synBlockHeight;
    private String netBlockHeight;
    private String nuls;
    private String locked;
    private String permanentLocked;
    private String timeLocked;
    private String heightLocked;
    private String contractIn;
    private String contractOut;
```

```
private List<LockedBalance> lockedTimeList=new ArrayList<>();
private List<LockedBalance> lockedHeightList=new ArrayList<>();


public String getAddress() {
    return address;
}

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

public String getNuls() {
    return nuls;
}

public void setNuls(String nuls) {
    this.nuls = nuls;
}

public String getLocked() {
    return locked;
}

public void setLocked(String locked) {
    this.locked = locked;
}

public String getSynBlockHeight() {
    return synBlockHeight;
}

public void setSynBlockHeight(String synBlockHeight) {
    this.synBlockHeight = synBlockHeight;
}

public String getNetBlockHeight() {
    return netBlockHeight;
}

public void setNetBlockHeight(String netBlockHeight) {
    this.netBlockHeight = netBlockHeight;
```

```
}

public String getPermanentLocked() {
    return permanentLocked;
}

public void setPermanentLocked(String permanentLocked) {
    this.permanentLocked = permanentLocked;
}

public String getTimeLocked() {
    return timeLocked;
}

public void setTimeLocked(String timeLocked) {
    this.timeLocked = timeLocked;
}

public String getHeightLocked() {
    return heightLocked;
}

public void setHeightLocked(String heightLocked) {
    this.heightLocked = heightLocked;
}

public String getContractIn() {
    return contractIn;
}

public void setContractIn(String contractIn) {
    this.contractIn = contractIn;
}

public String getContractOut() {
    return contractOut;
}

public void setContractOut(String contractOut) {
    this.contractOut = contractOut;
}
```

```
    public List<LockedBalance> getLockedTimeList() {
        return lockedTimeList;
    }

    public void setLockedTimeList(List<LockedBalance> lockedTimeList) {
        this.lockedTimeList = lockedTimeList;
    }

    public List<LockedBalance> getLockedHeightList() {
        return lockedHeightList;
    }

    public void setLockedHeightList(List<LockedBalance> lockedHeightList) {
        this.lockedHeightList = lockedHeightList;
    }
}

72:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
rpc\src\main\java\io\nuls\utxo\accounts\rpc\resource\UtxoAccountsResource.java
 */
package io.nuls.utxo.accounts.rpc.resource;

import io.nuls.account.constant.AccountErrorCode;
import io.nuls.core.tools.log.Log;
import io.nuls.kernel.context.NulsContext;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.func.TimeService;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Component;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.RpcClientResult;
import io.nuls.kernel.utils.AddressTool;
import io.nuls.utxo.accounts.constant.UtxoAccountsErrorCode;
import io.nuls.utxo.accounts.rpc.dto.AccountBalanceDto;
import io.nuls.utxo.accounts.storage.po.LockedBalance;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsBalancePo;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;
import io.swagger.annotations.*;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
```

```
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import java.math.BigDecimal;
import java.util.List;


@Path("/utxoAccounts")
@Api(value = "utxoAccounts", description = "utxoAccounts")
@Component
public class UtxoAccountsResource {
    @Autowired
    private UtxoAccountsStorageService utxoAccountsStorageService;
    @GET
    @Path("/{address}")
    @Produces(MediaType.APPLICATION_JSON)
    @ApiOperation("[] ")
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "success", response = RpcClientResult.class)
    })
    public RpcClientResult get(@ApiParam(name = "address", value = "", required = true)
                               @PathParam("address") String address) {
        if (!AddressTool.validAddress(address)) {
            return Result.getFailed(AccountErrorCode.ADDRESS_ERROR).toRpcClientResult();
        }
        try {
            Result<UtxoAccountsBalancePo>
utxoAccountsBalance=utxoAccountsStorageService.getUtxoAccountsBalanceByAddress(Address
Tool.getAddress(address));
            long synBlockHeight=utxoAccountsStorageService.getHadSynBlockHeight();
            if(null==utxoAccountsBalance || null==utxoAccountsBalance.getData()){
                return
Result.getFailed(UtxoAccountsErrorCode.DATA_NOT_FOUND).toRpcClientResult();
            }
            UtxoAccountsBalancePo dbAccountsBalance =utxoAccountsBalance.getData();
            AccountBalanceDto accountBalance=new AccountBalanceDto();
            accountBalance.setAddress(address);
            long totalNa=dbAccountsBalance.getOutputBalance()-
(dbAccountsBalance.getInputBalance());
            totalNa+=dbAccountsBalance.getContractToBalance();
            totalNa-=dbAccountsBalance.getContractFromBalance();
            accountBalance.setNuls( new BigDecimal(totalNa).toPlainString());
            long timeLockedNa=0;
```

```
        long heightLockedNa=0;
        long permanentLockedNa=dbAccountsBalance.getLockedPermanentBalance()-
(dbAccountsBalance.getUnLockedPermanentBalance());
        long lockedNa=permanentLockedNa;
        List<LockedBalance> timeLockedBalance=dbAccountsBalance.getLockedTimeList();
        long currentTime=TimeService.currentTimeMillis();
        for(LockedBalance balance:timeLockedBalance){
            if(balance.getLockedTime()>currentTime){
                lockedNa+=balance.getLockedBalance();
                timeLockedNa+=balance.getLockedBalance();
                accountBalance.getLockedTimeList().add(balance);
            }else{
                break;
            }
        }
        List<LockedBalance> heightLockedBalance=dbAccountsBalance.getLockedHeightList();
        for(LockedBalance balance:heightLockedBalance){
            if(balance.getLockedTime()>synBlockHeight){
                lockedNa+=balance.getLockedBalance();
                heightLockedNa+=balance.getLockedBalance();
                accountBalance.getLockedHeightList().add(balance);
            }else{
                break;
            }
        }
        accountBalance.setPermanentLocked(new
BigDecimal(permanentLockedNa).toPlainString());
        accountBalance.setLocked(new BigDecimal(lockedNa).toPlainString());
        accountBalance.setTimeLocked(new BigDecimal(timeLockedNa).toPlainString());
        accountBalance.setHeightLocked(new BigDecimal(heightLockedNa).toPlainString());
        accountBalance.setSynBlockHeight(String.valueOf(synBlockHeight));
        long netHeight= NulsContext.getInstance().getNetBestBlockHeight();
        accountBalance.setNetBlockHeight(String.valueOf(netHeight));
accountBalance.setContractIn(String.valueOf(dbAccountsBalance.getContractToBalance()));
  accountBalance.setContractOut(String.valueOf(dbAccountsBalance.getContractFromBalance()));
        return Result.getSuccess().setData(accountBalance).toRpcClientResult();
      } catch (NulsException e) {
        Log.error(e);
      }
      return
Result.getFailed(UtxoAccountsErrorCode.SYS_UNKOWN_EXCEPTION).toRpcClientResult();
    }
```

```
}
```

73:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-storage\src\main\java\io\nuls\utxo\accounts\storage\constant\UtxoAccountsStorageConstant.java

```
 */
package io.nuls.utxo.accounts.storage.constant;

/**
 * @desription:
 * @author: cody
 */
public interface UtxoAccountsStorageConstant {
    byte []DB_NAME_UTXO_ACCOUNTS_BLOCK_SYN_KEY =
"utxo_accounts_block_syn_key".getBytes();
    String DB_NAME_UTXO_ACCOUNTS_CHANGE_SUFFIX_KEY = "suffix";
//    String DB_NAME_UTXO_ACCOUNTS_BLOCK_INDEX="utxo_accounts_block_index";
    String DB_NAME_UTXO_ACCOUNTS_BLOCK_CACHE = "utxo_accounts_block_cache";
    String DB_NAME_UTXO_ACCOUNTS_CONFIRMED_BALANCE =
"utxo_accounts_confirmed_balance";
//    String DB_NAME_UTXO_ACCOUNTS_LOCKEDTIME_BALANCE =
"utxo_accounts_lockedtime_balance";
//    String DB_NAME_UTXO_ACCOUNTS_LOCKEDHEIGHT_BALANCE =
"utxo_accounts_lockedheight_balance";
    final int MAX_CACHE_BLOCK_NUM=1100;

}
```

74:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-storage\src\main\java\io\nuls\utxo\accounts\storage\po\LocalCacheBlockBalance.java

```
 */
package io.nuls.utxo.accounts.storage.po;

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

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

```
public class LocalCacheBlockBalance extends BaseNulsData{
    private long blockHeight;
    private NulsDigestData hash;
    private NulsDigestData preHash;
    private List<UtxoAccountsBalancePo> balanceList=new ArrayList<>();

    public NulsDigestData getHash() {
        return hash;
    }

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

    public NulsDigestData getPreHash() {
        return preHash;
    }

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

    public long getBlockHeight() {
        return blockHeight;
    }

    public void setBlockHeight(long blockHeight) {
        this.blockHeight = blockHeight;
    }

    public List<UtxoAccountsBalancePo> getBalanceList() {
        return balanceList;
    }

    public void setBalanceList(List<UtxoAccountsBalancePo> balanceList) {
        this.balanceList = balanceList;
    }

    @Override
    protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
        stream.writeInt64(blockHeight);
```

```
        stream.write(this.getHash().serialize());
        stream.write(this.getPreHash().serialize());
        int balanceListCount = balanceList == null ? 0 : balanceList.size();
        stream.writeVarInt(balanceListCount);
        if (null != balanceList) {
            for (UtxoAccountsBalancePo balance : balanceList) {
                stream.writeBytesWithLength(balance.getOwner());
                stream.writeInt64(balance.getInputBalance());
                stream.writeInt64(balance.getOutputBalance());
                stream.writeInt64(balance.getLockedPermanentBalance());
                stream.writeInt64(balance.getUnLockedPermanentBalance());
                stream.writeInt64(balance.getContractFromBalance());
                stream.writeInt64(balance.getContractToBalance());
                stream.writeInt64(balance.getBlockHeight());
                stream.writeUint32(balance.getTxIndex());
                int lockedTimeListSize = balance.getLockedTimeList() == null ? 0 :
balance.getLockedTimeList() .size();
                stream.writeVarInt(lockedTimeListSize);
                if (null !=  balance.getLockedTimeList() ) {
                    for (LockedBalance lockedBalance :  balance.getLockedTimeList() ) {
                        stream.writeInt64(lockedBalance.getLockedTime());
                        stream.writeInt64(lockedBalance.getLockedBalance());
                    }
                }
                int lockedHeightListSize = balance.getLockedHeightList() == null ? 0 :
balance.getLockedHeightList().size();
                stream.writeVarInt(lockedHeightListSize);
                if (null !=  balance.getLockedHeightList()) {
                    for (LockedBalance lockedBalance :  balance.getLockedHeightList()) {
                        stream.writeInt64(lockedBalance.getLockedTime());
                        stream.writeInt64(lockedBalance.getLockedBalance());
                    }
                }
            }
        }

    }

    @Override
    public void parse(NulsByteBuffer byteBuffer) throws NulsException {
        this.blockHeight=byteBuffer.readInt64();
        this.hash=new NulsDigestData( byteBuffer.readByte(),byteBuffer.readByLengthByte());
```

```
        this.preHash=new NulsDigestData( byteBuffer.readByte(),byteBuffer.readByLengthByte());
        int listCount = (int) byteBuffer.readVarInt();
        if (0 < listCount) {
            List<UtxoAccountsBalancePo> list = new ArrayList<>();
            for (int i = 0; i < listCount; i++) {
                list.add(byteBuffer.readNulsData(new UtxoAccountsBalancePo()));
            }
            this.balanceList = list;
        }
    }

    @Override
    public int size() {
        int size=0;
        size += SerializeUtils.sizeOfInt64();
        size+=this.hash.size();
        size+=this.preHash.size();
        size+= SerializeUtils.sizeOfVarInt(balanceList == null ? 0 : balanceList.size());
        if (null != balanceList) {
            for (UtxoAccountsBalancePo balance : balanceList) {
                size += balance.size();
            }
        }
        return size;
    }
}

75:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
storage\src\main\java\io\nuls\utxo\accounts\storage\po\LockedBalance.java
 */
package io.nuls.utxo.accounts.storage.po;

import java.io.Serializable;

public class LockedBalance  implements Serializable {
    private long lockedTime=0;
    private long lockedBalance=0;

    public long getLockedTime() {
        return lockedTime;
    }
```

```
    public void setLockedTime(long lockedTime) {
        this.lockedTime = lockedTime;
    }

    public long getLockedBalance() {
        return lockedBalance;
    }

    public void setLockedBalance(long lockedBalance) {
        this.lockedBalance = lockedBalance;
    }

    public static int compareByLockedTime(LockedBalance b1, LockedBalance b2) {
        return (b1.getLockedTime()< b2.getLockedTime()) ? 1 : ((b1.getLockedTime() ==
b2.getLockedTime()) ? 0 : -1);
    }
}

76:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
storage\src\main\java\io\nuls\utxo\accounts\storage\po\UtxoAccountsBalancePo.java
 */
package io.nuls.utxo.accounts.storage.po;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.Coin;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

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

public class UtxoAccountsBalancePo extends BaseNulsData {
    private byte[] owner;
    private Long inputBalance;
    private Long outputBalance;
    private Long lockedPermanentBalance;
    private Long unLockedPermanentBalance;
    private Long contractFromBalance;
    private Long contractToBalance;
```

```
private Long blockHeight=0L;
private int txIndex=0;
private List<LockedBalance> lockedTimeList=new ArrayList<>();
private List<LockedBalance> lockedHeightList=new ArrayList<>();
public UtxoAccountsBalancePo() {
    this.inputBalance=0L;
    this.outputBalance=0L;
    this.lockedPermanentBalance=0L;
    this.unLockedPermanentBalance=0L;
    this.contractFromBalance=0L;
    this.contractToBalance=0L;
}



public byte[] getOwner() {
    return owner;
}

public void setOwner(byte[] owner) {
    this.owner = owner;
}
public Long getInputBalance() {
    return inputBalance;
}

public void setInputBalance(Long inputBalance) {
    this.inputBalance = inputBalance;
}

public Long getOutputBalance() {
    return outputBalance;
}

public void setOutputBalance(Long outputBalance) {
    this.outputBalance = outputBalance;
}

public Long getContractFromBalance() {
    return contractFromBalance;
}
```

```
public void setContractFromBalance(Long contractFromBalance) {
    this.contractFromBalance = contractFromBalance;
}

public Long getContractToBalance() {
    return contractToBalance;
}

public void setContractToBalance(Long contractToBalance) {
    this.contractToBalance = contractToBalance;
}

public Long getBlockHeight() {
    return blockHeight;
}

public void setBlockHeight(Long blockHeight) {
    this.blockHeight = blockHeight;
}

public int getTxIndex() {
    return txIndex;
}

public void setTxIndex(int txIndex) {
    this.txIndex = txIndex;
}

public Long getLockedPermanentBalance() {
    return lockedPermanentBalance;
}

public void setLockedPermanentBalance(Long lockedPermanentBalance) {
    this.lockedPermanentBalance = lockedPermanentBalance;
}

public Long getUnLockedPermanentBalance() {
    return unLockedPermanentBalance;
}

public void setUnLockedPermanentBalance(Long unLockedPermanentBalance) {
```

```
    this.unLockedPermanentBalance = unLockedPermanentBalance;
}

public List<LockedBalance> getLockedTimeList() {
    return lockedTimeList;
}

public void setLockedTimeList(List<LockedBalance> lockedTimeList) {
    this.lockedTimeList = lockedTimeList;
}

public List<LockedBalance> getLockedHeightList() {
    return lockedHeightList;
}

public void setLockedHeightList(List<LockedBalance> lockedHeightList) {
    this.lockedHeightList = lockedHeightList;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeBytesWithLength(owner);
    stream.writeInt64(inputBalance);
    stream.writeInt64(outputBalance);
    stream.writeInt64(lockedPermanentBalance);
    stream.writeInt64(unLockedPermanentBalance);
    stream.writeInt64(contractFromBalance);
    stream.writeInt64(contractToBalance);
    stream.writeInt64(blockHeight);
    stream.writeUint32(txIndex);
    int lockedTimeListSize = lockedTimeList == null ? 0 : lockedTimeList.size();
    stream.writeVarInt(lockedTimeListSize);
    if (null != lockedTimeList) {
        for (LockedBalance balance : lockedTimeList) {
            stream.writeInt64(balance.getLockedTime());
            stream.writeInt64(balance.getLockedBalance());
        }
    }
    int lockedHeightListSize = lockedHeightList == null ? 0 : lockedHeightList.size();
    stream.writeVarInt(lockedHeightListSize);
    if (null != lockedHeightList) {
        for (LockedBalance balance : lockedHeightList) {
```

```
            stream.writeInt64(balance.getLockedTime());
            stream.writeInt64(balance.getLockedBalance());
        }
    }
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
    this.owner = byteBuffer.readByLengthByte();
    this.inputBalance=byteBuffer.readInt64();
    this.outputBalance=byteBuffer.readInt64();
    this.lockedPermanentBalance=byteBuffer.readInt64();
    this.unLockedPermanentBalance=byteBuffer.readInt64();
    this.contractFromBalance=byteBuffer.readInt64();
    this.contractToBalance=byteBuffer.readInt64();
    this.blockHeight=byteBuffer.readInt64();
    this.txIndex=byteBuffer.readInt32();
    int lockedTimeCount = (int) byteBuffer.readVarInt();
    if (0 < lockedTimeCount) {
        List<LockedBalance> timeBalanceList = new ArrayList<>();
        for (int i = 0; i < lockedTimeCount; i++) {
            LockedBalance balance=new LockedBalance();
            balance.setLockedTime(byteBuffer.readInt64());
            balance.setLockedBalance(byteBuffer.readInt64());
            timeBalanceList.add(balance);
        }
        this.lockedTimeList = timeBalanceList;
    }
    int lockedHeightCount = (int) byteBuffer.readVarInt();
    if (0 < lockedHeightCount) {
        List<LockedBalance> heightBalanceList = new ArrayList<>();
        for (int i = 0; i < lockedHeightCount; i++) {
            LockedBalance balance=new LockedBalance();
            balance.setLockedTime(byteBuffer.readInt64());
            balance.setLockedBalance(byteBuffer.readInt64());
            heightBalanceList.add(balance);
        }
        this.lockedHeightList = heightBalanceList;
    }
}
```

```
    @Override
    public int size() {
        int size = 0;
        size += SerializeUtils.sizeOfBytes(owner);
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfInt64();
        size += SerializeUtils.sizeOfUint32();

        size+= SerializeUtils.sizeOfVarInt(lockedTimeList == null ? 0 : lockedTimeList.size());
        if (null != lockedTimeList) {
            for (LockedBalance balance : lockedTimeList) {
                size += SerializeUtils.sizeOfInt64();
                size += SerializeUtils.sizeOfInt64();
            }
        }
        size += SerializeUtils.sizeOfVarInt(lockedHeightList == null ? 0 : lockedHeightList.size());
        if (null != lockedHeightList) {
            for (LockedBalance balance : lockedHeightList) {
                size += SerializeUtils.sizeOfInt64();
                size += SerializeUtils.sizeOfInt64();
            }
        }

        return size;

    }
}

77:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
storage\src\main\java\io\nuls\utxo\accounts\storage\po\UtxoAccountsSynInfo.java
 */
package io.nuls.utxo.accounts.storage.po;

import java.io.Serializable;

public class UtxoAccountsSynInfo implements Serializable {
    private Long hadSynBlockHeight=-1L;
```

```
    private Long updateTimeMillion=0L;

    public UtxoAccountsSynInfo(long hadSynBlockHeight){
        this.hadSynBlockHeight=hadSynBlockHeight;
    }
    public Long getHadSynBlockHeight() {
        return hadSynBlockHeight;
    }

    public void setHadSynBlockHeight(Long hadSynBlockHeight) {
        this.hadSynBlockHeight = hadSynBlockHeight;
    }

    public Long getUpdateTimeMillion() {
        return updateTimeMillion;
    }

    public void setUpdateTimeMillion(Long updateTimeMillion) {
        this.updateTimeMillion = updateTimeMillion;
    }
}

78:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-
storage\src\main\java\io\nuls\utxo\accounts\storage\service\impl\UtxoAccountsStorageServiceImpl
.java
 */
package io.nuls.utxo.accounts.storage.service.impl;

import io.nuls.db.service.BatchOperation;
import io.nuls.db.service.DBService;
import io.nuls.kernel.constant.KernelErrorCode;
import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.lite.annotation.Autowired;
import io.nuls.kernel.lite.annotation.Service;
import io.nuls.kernel.lite.core.bean.InitializingBean;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.kernel.script.Script;
import io.nuls.kernel.script.SignatureUtil;
import io.nuls.utxo.accounts.storage.constant.UtxoAccountsStorageConstant;
```

```
import io.nuls.utxo.accounts.storage.po.LocalCacheBlockBalance;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsBalancePo;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsSynInfo;
import io.nuls.utxo.accounts.storage.service.UtxoAccountsStorageService;
import io.nuls.ledger.service.LedgerService;
import io.nuls.protocol.service.BlockService;

import java.util.List;

@Service
public class UtxoAccountsStorageServiceImpl implements UtxoAccountsStorageService,
InitializingBean {
    @Autowired
    private DBService dbService;
    @Autowired
    BlockService blockService;
    @Autowired
    LedgerService ledgerService;
    @Override
    public Result saveUtxoAccountsInfo(byte[] addressBytes, UtxoAccountsBalancePo balance) {
        try {
            return
dbService.putModel(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CONFIRME
D_BALANCE, addressBytes, balance);
        } catch (Exception e) {
            return Result.getFailed();
        }
    }

    @Override
    public Result saveByteUtxoAccountsInfo(byte[] addressBytes, UtxoAccountsBalancePo
balance) {
        try {
            return
dbService.put(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CONFIRMED_BA
LANCE, addressBytes, balance.serialize());
        } catch (Exception e) {
            return Result.getFailed();
        }
    }

    @Override
```

```
    public Result batchSaveByteUtxoAcountsInfo(List<UtxoAccountsBalancePo> list) {
        BatchOperation batch =
dbService.createWriteBatch(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CO
NFIRMED_BALANCE);
        try {
            for (UtxoAccountsBalancePo balance : list) {
                batch.put(balance.getOwner(), balance.serialize());
            }
            Result batchResult = batch.executeBatch();
            return batchResult;
        }catch (Exception e){
            return Result.getFailed();
        }
    }

    @Override
    public Result deleteUtxoAcountsInfo(byte[] addressBytes) {
        return
dbService.delete(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CONFIRMED_
BALANCE,addressBytes);

    }


    @Override
    public long getHadSynBlockHeight() {
        //
        //-1
        UtxoAccountsSynInfo utxoAccountsSynInfo=
dbService.getModel(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_CA
CHE, UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_SYN_KEY,
UtxoAccountsSynInfo.class);
        if(utxoAccountsSynInfo==null) {
            return -1;
        }
        return utxoAccountsSynInfo.getHadSynBlockHeight();
    }

    @Override
    public Result saveHadSynBlockHeight(long height) {
        UtxoAccountsSynInfo utxoAccountsSynInfo=new UtxoAccountsSynInfo(height);
        utxoAccountsSynInfo.setUpdateTimeMillion(System.currentTimeMillis());
```

```
        Result
result=dbService.putModel(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLO
CK_CACHE,UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_SYN_KEY,
utxoAccountsSynInfo);
        return result;
    }

    @Override
    public Result<UtxoAccountsBalancePo> getUtxoAccountsBalanceByAddress(byte[]
addressBytes) throws NulsException {
        //
        if (addressBytes == null) {
            return Result.getFailed(KernelErrorCode.NULL_PARAMETER);
        }
        byte[] bytes = null;
        if(addressBytes[2] == 3){
            Script scriptPubkey = SignatureUtil.createOutputScript(addressBytes);
            bytes = scriptPubkey.getProgram();
        }else{
         bytes = addressBytes;
        }
        byte []balance =
dbService.get(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CONFIRMED_BA
LANCE, bytes);
        UtxoAccountsBalancePo b=new UtxoAccountsBalancePo();
        b.parse(balance,0);
        if (b.getOwner()==null){
            return Result.getSuccess().setData(null);
        }
        return Result.getSuccess().setData(b);
    }

    @Override
    public Result<LocalCacheBlockBalance> getLocalCacheBlock(long height)  throws
NulsException {
//        LocalCacheBlockBalance
balance=dbService.getModel(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BL
OCK_CACHE,String.valueOf(height).getBytes(),LocalCacheBlockBalance.class);
        byte []block =
dbService.get(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_CACHE,
String.valueOf(height).getBytes());
        LocalCacheBlockBalance localCacheBlockBalance=new LocalCacheBlockBalance();
```

```
        localCacheBlockBalance.parse(block,0);
        if(localCacheBlockBalance.getHash()==null){
            return Result.getSuccess().setData(null);
        }
        return Result.getSuccess().setData(localCacheBlockBalance);
    }

    @Override
    public Result saveLocalCacheBlock(long height,LocalCacheBlockBalance
localCacheBlockBalance) {
//        Result
result=dbService.putModel(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLO
CK_CACHE,String.valueOf(height).getBytes(),localCacheBlockBalance);
        try {
            return
dbService.put(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_CACHE,
String.valueOf(height).getBytes(), localCacheBlockBalance.serialize());
        } catch (Exception e) {
            e.printStackTrace();
            return Result.getFailed();
        }
    }


    @Override
    public Result deleteLocalCacheBlock(long height) {
        return
dbService.delete(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_CACH
E,String.valueOf(height).getBytes());
    }

    @Override
    public Transaction getTx(NulsDigestData hash) {
        return ledgerService.getTx(hash);
    }

    @Override
    public Result<Block> getBlock(long height) {
        return blockService.getBlock(height,true);
    }
```

```
    @Override
    public void afterPropertiesSet() throws NulsException {
dbService.createArea(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_CONFIRMED_BALANCE);
dbService.createArea(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_BLOCK_CACHE);
//
dbService.createArea(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_LOCKEDTIME_BALANCE);
//
dbService.createArea(UtxoAccountsStorageConstant.DB_NAME_UTXO_ACCOUNTS_LOCKEDHEIGHT_BALANCE);

    }
}

79:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\base\utxo-accounts-storage\src\main\java\io\nuls\utxo\accounts\storage\service\UtxoAccountsStorageService.java
 */
package io.nuls.utxo.accounts.storage.service;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.Block;
import io.nuls.kernel.model.NulsDigestData;
import io.nuls.kernel.model.Result;
import io.nuls.kernel.model.Transaction;
import io.nuls.utxo.accounts.storage.po.LocalCacheBlockBalance;
import io.nuls.utxo.accounts.storage.po.UtxoAccountsBalancePo;


import java.util.List;

public interface UtxoAccountsStorageService {

    Result saveUtxoAccountsInfo(byte[] addressBytes, UtxoAccountsBalancePo balance);
//    public Result<UtxoAccountsBalance> getUtxoAccountBalanceByAddress(byte[] addressBytes);
    Result saveByteUtxoAccountsInfo(byte[] addressBytes, UtxoAccountsBalancePo balance);
    Result batchSaveByteUtxoAcountsInfo(List<UtxoAccountsBalancePo> list);
    Result deleteUtxoAcountsInfo(byte[] addressBytes);
    long getHadSynBlockHeight();
    Result saveHadSynBlockHeight(long height);
```

```
    Result<UtxoAccountsBalancePo> getUtxoAccountsBalanceByAddress(byte[] addressBytes)
throws NulsException;

    /**
     *
     * Get the block (from storage) according to the block height
     *
     * @param height /block height
     * @return /block
     */
    Result<LocalCacheBlockBalance> getLocalCacheBlock(long height) throws NulsException;

    Result saveLocalCacheBlock(long height, LocalCacheBlockBalance localCacheBlockBalance);

    Result deleteLocalCacheBlock(long height);

    Transaction getTx(NulsDigestData hash);
    Result<Block> getBlock(long height);


}

80:F:\git\coin\nuls\nuls-1.1.3\nuls\utxo-accounts-module\utxo-
accounts\src\main\java\io\nuls\utxo\accounts\model\UtxoAccountsBalance.java
 */
package io.nuls.utxo.accounts.model;

import io.nuls.kernel.exception.NulsException;
import io.nuls.kernel.model.BaseNulsData;
import io.nuls.kernel.model.Na;
import io.nuls.kernel.model.NulsData;
import io.nuls.kernel.utils.NulsByteBuffer;
import io.nuls.kernel.utils.NulsOutputStreamBuffer;
import io.nuls.kernel.utils.SerializeUtils;

import java.io.IOException;

public class UtxoAccountsBalance extends BaseNulsData {
    private byte[] owner;
    private Na balance;
    private Na hadLocked;
```

```
public byte[] getOwner() {
    return owner;
}

public void setOwner(byte[] owner) {
    this.owner = owner;
}

public Na getBalance() {
    return balance;
}

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

public Na getHadLocked() {
    return hadLocked;
}

public void setHadLocked(Na hadLocked) {
    this.hadLocked = hadLocked;
}

@Override
public int size() {
    int size=0;
    size += SerializeUtils.sizeOfBytes(owner);
    size += SerializeUtils.sizeOfInt64();
    size += SerializeUtils.sizeOfInt64();
    return size;
}

@Override
protected void serializeToStream(NulsOutputStreamBuffer stream) throws IOException {
    stream.writeBytesWithLength(owner);
    stream.writeInt64(balance.getValue());
    stream.writeInt64(hadLocked.getValue());
}

@Override
public void parse(NulsByteBuffer byteBuffer) throws NulsException {
```

```
        this.owner = byteBuffer.readByLengthByte();
        this.balance = Na.valueOf(byteBuffer.readInt64());
        this.hadLocked = Na.valueOf(byteBuffer.readInt64());
    }
}
```