目　录

JT701D SDK

Java

Java

1.Jar package download

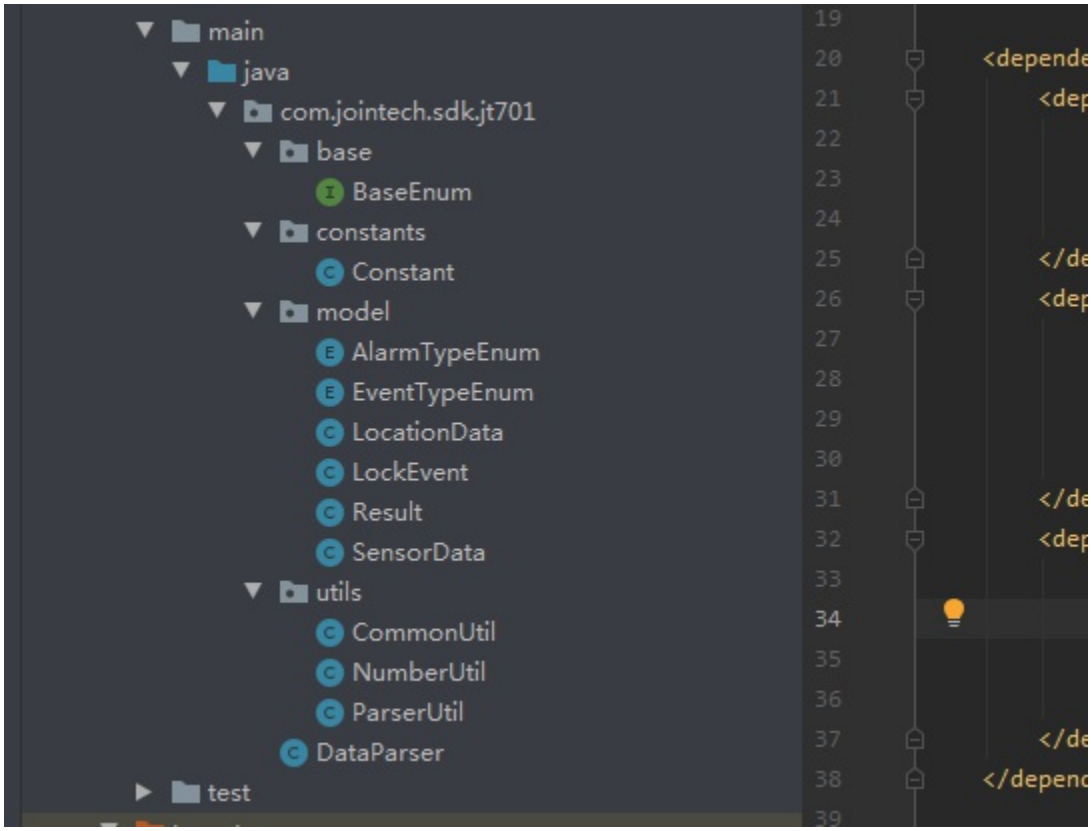
jt701-sdk-1.0.0.jar Download
If you need Jar package development source code, please contact the business application

---

2.Integrated Development Instructions

2.1.Integrated development language and framework description

jt701-sdk-1.0.0.jar is based on the Java language，SpringBoot2.x frame，use netty，fastjson，lombok



BaseEnum：base enumeration
Constant：custom constant
AlarmTypeEnum：Alarm enumeration
EventTypeEnum：event enumeration
LocationData：Location entity class
LockEvent：lock event entity class
Result：result entity class

SensorData：Slave data entity class
CommonUtil：public method class
NumberUtil：digital manipulation tools
ParserUtil：Parser method tool class
DataParser：Parser main method

2.2.Integration Instructions

Introduce jt701-sdk-1.0.0.jar into your gateway program as follows：
Introduce the jar package in pom.xml

```
        <dependency>
            <groupId>com.jointech.sdk</groupId>
            <artifactId>jt701-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>
```

Call jt701-sdk-1.0.0.jar, the receiveData() method in the DataParser class
receiveData() method is overloaded

```
    /**
     * Parse Hex string raw data
     * @param strData hexadecimal string
     * @return
     */
    public static Object receiveData(String strData)
    {
        ByteBuf msgBodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(strData.le
ngth()/2);
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse byte[] raw data
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =ByteBufAllocator.DEFAULT.heapBuffer(bytes.lengt
h);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }
```

```
    /**
     * Parse ByteBuf raw data
     * @param in
     * @return
     */
    private static Object receiveData(ByteBuf in)
    {
        Object decoded = null;
        in.markReaderIndex();
        int header = in.readByte();
        if (header == Constant.TEXT_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeTextMessage(in);
        } else if (header == Constant.BINARY_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeBinaryMessage(in);
        } else {
            return null;
        }
        return JSONArray.toJSON(decoded).toString();
    }
```

2.3.core code

Parsing method tool class ParserUtil

```
package com.jointech.sdk.jt701.utils;

import com.jointech.sdk.jt701.constants.Constant;
import com.jointech.sdk.jt701.model.*;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
```

```java
/**
 * <p>Description: Analysis method tool class</p>
 * @author HyoJung
 * @date 20210526
 */
public class ParserUtil {
    private ParserUtil()
    {}

    /**
     * Parse command response data
     * @param in rawdata
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in)
    {
        //Define the location data entity class
        Result model = new Result();
        //packet head(
        in.readByte();
        //Field list
        List<String> itemList = new ArrayList<String>();
        //Transparent binary data
        ByteBuf msgBody = null;
        while (in.readableBytes() > 0) {
            //The 7th field of wireless gateway data upload (WLNET5, WLNET7
) and the following are binary data
            if (itemList.size() >= 6 && Objects.equals("WLNET", itemList.ge
t(3)) && Constant.WLNET_TYPE_LIST.contains(itemList.get(4))) {
                //Length up to the end ")"
                int lastItemLen = in.readableBytes() - 1;
                //unescape
                msgBody = Unpooled.buffer(lastItemLen);
                CommonUtil.unescape(in, msgBody, lastItemLen);
                in.readByte();
            } else {
                //Query the subscript of comma to intercept data
                int index = in.bytesBefore(Constant.TEXT_MSG_SPLITER);
                int itemLen = index > 0 ? index : in.readableBytes() - 1;
                byte[] byteArr = new byte[itemLen];
                in.readBytes(byteArr);
                in.readByte();
                itemList.add(new String(byteArr));
            }
        }
        //WLNET message type is composite
```

```
        String msgType = itemList.get(1);
        if (itemList.size() >= 5 && (Objects.equals("WLNET", itemList.get(3
))||Objects.equals("OTA", itemList.get(3)))) {
            msgType = itemList.get(3) + itemList.get(4);
        }
        Object dataBody=null;
        if(msgType.equals("WLNET5")) {
            SensorData sensorData=parseWlnet5(msgBody);
            dataBody=sensorData;
            model.setReplyMsg(replyMessage(msgType,sensorData.getIndex()));
        }else if(msgType.equals("P45")) {
            dataBody=parseP45(itemList);
            model.setReplyMsg(replyMessage(msgType,itemList));
        }else {
            if(itemList.size()>0)
            {
                dataBody="(";
                for(String item :itemList) {
                    dataBody+=item+",";
                }
                dataBody=CommonUtil.trimEnd(dataBody.toString(),",");
                dataBody += ")";
            }
        }
        model.setDeviceID(itemList.get(0));
        model.setMsgType(msgType);
        model.setDataBody(dataBody);
        return model;
    }

    /**
     * Parse slave data
     * @param byteBuf
     * @return
     */
    private static SensorData parseWlnet5(ByteBuf byteBuf) {
        SensorData sensorData = new SensorData();
        //positioning time
        byte[] timeArr = new byte[6];
        byteBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = parseBcdTime(bcdTimeStr);
        //latitude
        byte[] latArr = new byte[4];
        byteBuf.readBytes(latArr);
        String latHexStr = ByteBufUtil.hexDump(latArr);
```

```
        BigDecimal latFloat = new BigDecimal(latHexStr.substring(2, 4) + ".
" + latHexStr.substring(4)).divide(new BigDecimal("60"), 6, RoundingMode.HA
LF_UP);
        double lat = new BigDecimal(latHexStr.substring(0, 2)).add(latFloat
).doubleValue();
        //Longitude
        byte[] lngArr = new byte[5];
        byteBuf.readBytes(lngArr);
        String lngHexStr = ByteBufUtil.hexDump(lngArr);
        BigDecimal lngFloat = new BigDecimal(lngHexStr.substring(3, 5) + ".
" + lngHexStr.substring(5, 9)).divide(new BigDecimal("60"), 6, RoundingMode
.HALF_UP);
        double lng = new BigDecimal(lngHexStr.substring(0, 3)).add(lngFloat
).doubleValue();
        //bit indication
        int bitFlag = Byte.parseByte(lngHexStr.substring(9, 10), 16);
        //Positioning status
        int locationType = (bitFlag & 0x01) > 0 ? 1 : 0;
        //north latitude, south latitude
        if ((bitFlag & 0b0010) == 0) {
            lat = -lat;
        }
        //East longitude and West longitude
        if ((bitFlag & 0b0100) == 0) {
            lng = -lng;
        }
        //Speed
        int speed = (int) (byteBuf.readUnsignedByte() * 1.85);
        //Header(direction)
        int direction = byteBuf.readUnsignedByte() * 2;

        //slave time
        byte[] slaveMachineTimeArr = new byte[6];
        byteBuf.readBytes(slaveMachineTimeArr);
        String slaveMachineBcdTimeStr = ByteBufUtil.hexDump(slaveMachineTim
eArr);
        ZonedDateTime slaveMachineZonedDateTime = parseBcdTime(slaveMachine
BcdTimeStr);
        //slave sensor ID
        byte[] slaveMachineIdArr = new byte[5];
        byteBuf.readBytes(slaveMachineIdArr);
        String slaveMachineId = ByteBufUtil.hexDump(slaveMachineIdArr).toUp
perCase();
        //slave data serial number
        int flowId = byteBuf.readUnsignedByte();
        //slave sensor battery level
```

```
        String voltage = new BigDecimal(byteBuf.readUnsignedShort()).divide
(new BigDecimal("100"), 2, RoundingMode.HALF_UP).toString();
        //slave sensor battery percertage
        int power = byteBuf.readUnsignedByte();
        //RSSI
        int rssi = byteBuf.readUnsignedByte();
        //sensor type
        int sensorType = byteBuf.readUnsignedByte();
        //temperature value
        double temperature = -1000.0;
        //Humidity value
        int humidity = 0;
        //Event type
        int eventType = -1;
        //Device status
        int terminalStatus = -1;
        //unlock&lock times
        int lockTimes = -1;
        if (sensorType == 1) {
            //temperature
            temperature = parseTemperature(byteBuf.readShort());
            //Humidity
            humidity = byteBuf.readUnsignedByte();
            //Gateway saves the number of data
            int itemCount = byteBuf.readUnsignedShort();
            //gateway status
            int gatewayStatus = byteBuf.readUnsignedByte();
        } else if (sensorType == 4) {
            //event
            int event = byteBuf.readUnsignedShort();
            //Judgment event
            if (NumberUtil.getBitValue(event, 0) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_0.getV
alue());
            } else if (NumberUtil.getBitValue(event, 1) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_1.getV
alue());
            } else if (NumberUtil.getBitValue(event, 2) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_2.getV
alue());
            } else if (NumberUtil.getBitValue(event, 3) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_3.getV
alue());
            } else if (NumberUtil.getBitValue(event, 4) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_4.getV
alue());
```

```
            } else if (NumberUtil.getBitValue(event, 5) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_5.getV
alue());
            } else if (NumberUtil.getBitValue(event, 6) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_6.getV
alue());
            } else if (NumberUtil.getBitValue(event, 7) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_7.getV
alue());
            } else if (NumberUtil.getBitValue(event, 8) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_8.getV
alue());
            } else if (NumberUtil.getBitValue(event, 9) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_9.getV
alue());
            }else if (NumberUtil.getBitValue(event, 14) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_14.get
Value());
            }
            //Device status
            terminalStatus = byteBuf.readUnsignedShort();
            //unlock times
            lockTimes = byteBuf.readUnsignedShort();
            //gateway status
            int gatewayStatus = byteBuf.readUnsignedByte();
        }else if (sensorType == 5||sensorType == 6)
        {
            //event
            int event = byteBuf.readUnsignedShort();
            //Judgment event
            if (NumberUtil.getBitValue(event, 0) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_15.get
Value());
            } else if (NumberUtil.getBitValue(event, 1) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_16.get
Value());
            } else if (NumberUtil.getBitValue(event, 2) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_17.get
Value());
            } else if (NumberUtil.getBitValue(event, 3) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_6.getV
alue());
            } else if (NumberUtil.getBitValue(event, 4) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_18.get
Value());
            } else if (NumberUtil.getBitValue(event, 5) == 1) {
```

```
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_19.get
Value());
            } else if (NumberUtil.getBitValue(event, 6) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_3.getV
alue());
            } else if (NumberUtil.getBitValue(event, 7) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_0.getV
alue());
            } else if (NumberUtil.getBitValue(event, 8) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_1.getV
alue());
            } else if (NumberUtil.getBitValue(event, 9) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_20.get
Value());
            }else if (NumberUtil.getBitValue(event, 10) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_21.get
Value());
            }else if (NumberUtil.getBitValue(event, 11) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_22.get
Value());
            }else if (NumberUtil.getBitValue(event, 12) == 1) {
                eventType = Integer.parseInt(EventTypeEnum.LockEvent_5.getV
alue());
            }
            //Device status
            terminalStatus = byteBuf.readUnsignedShort();
            //unlock times
            lockTimes = byteBuf.readUnsignedShort();
            //gateway status
            int gatewayStatus = byteBuf.readUnsignedByte();
        }
        sensorData.setGpsTime(gpsZonedDateTime.toString());
        sensorData.setLatitude(lat);
        sensorData.setLongitude(lng);
        sensorData.setLocationType(locationType);
        sensorData.setSpeed(speed);
        sensorData.setDirection(direction);
        sensorData.setSensorID(slaveMachineId);
        sensorData.setLockStatus(NumberUtil.getBitValue(terminalStatus, 0))
;
        sensorData.setLockRope(NumberUtil.getBitValue(terminalStatus, 0));
        sensorData.setLockTimes(lockTimes);
        sensorData.setIndex(flowId);
        sensorData.setVoltage(voltage);
        sensorData.setPower(power);
        sensorData.setRSSI(rssi);
```

```
            sensorData.setDateTime(slaveMachineZonedDateTime.toString());
            sensorData.setSensorType(sensorType);
            sensorData.setTemperature(temperature);
            sensorData.setHumidity(humidity);
            sensorData.setEvent(eventType);
            return sensorData;
        }

        /**
         * Parse P45
         * @param itemList
         * @return
         */
        private static LockEvent parseP45(List<String> itemList)
        {
            LockEvent model = new LockEvent();
            model.DateTime= parseBcdTime(itemList.get(2) + itemList.get(3)).toS
    tring();
            model.Latitude = Double.valueOf(itemList.get(4));
            if (itemList.get(5).equals("S"))
            {
                model.Latitude = -model.Latitude;
            }
            model.Longitude = Double.valueOf(itemList.get(6));
            if (itemList.get(5).equals("W"))
            {
                model.Longitude = -model.Longitude;
            }
            model.LocationType= itemList.get(8).equals("V") ? 0 : 1;
            model.Speed= Double.valueOf(itemList.get(9)).intValue();
            model.Direction = Integer.valueOf(itemList.get(10));
            model.Event = Integer.valueOf(itemList.get(11));
            //unlock verification
            int status= Integer.valueOf(itemList.get(12));
            model.RFIDNo = itemList.get(13);
            //Dynamic password unlock
            if (model.Event == 6)
            {
                if (status == 0)
                {
                    //Incorrect unlock code
                    model.Status = 0;
                }
                else if (status > 0 && status <= 10)
                {
                    //normal unlock
```

```
                model.Status = 1;
                //Fence ID when unlocking inside the fence
                model.UnlockFenceID = status;
            }
            else if (status == 98)
            {
                //normal unlock
                model.Status = 1;
            }
            else if (status == 99)
            {
                //The device has enabled unlocking within the fence, and th
 e current unlocking is not within the fence, refusing to unlock
                model.Status = 3;
            }
        }
        else if (model.Event == 4)
        {
            if (Integer.valueOf(itemList.get(14)) == 0)
            {
                //Incorrect unlock code
                model.Status = 0;
            }
            else
            {
                //normal unlock
                model.Status = 1;
            }
        }
        model.PsdErrorTimes = Integer.valueOf(itemList.get(15));
        model.Index = Integer.valueOf(itemList.get(16));
        if (itemList.size() > 17)
        {
            model.Mileage = Integer.valueOf(itemList.get(16));
        }
        return model;
    }

    /**
     * Parse slave data temperature
     *
     * @param temperatureInt
     * @return
     */
    private static double parseTemperature(int temperatureInt) {
        if (temperatureInt == 0xFFFF) {
```

```
            return 9999.9;
        }
        double temperature = ((short) (temperatureInt << 4) >> 4) * 0.1;
        if ((temperatureInt >> 12) > 0) {
            temperature = -temperature;
        }
        return temperature;
    }

    /**
     * Convert GPS time
     *
     * @param bcdTimeStr
     * @return
     */
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        if(bcdTimeStr.equals("000000000000"))
        {
            //The default time given is January 1, 2000 00:00:00
            bcdTimeStr="010100000000";
        }
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("ddMMyyHH
mmss");
        LocalDateTime localDateTime = LocalDateTime.parse(bcdTimeStr, forma
tter);
        ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, ZoneO
ffset.UTC);
        return zonedDateTime;
    }

    /**
     * Parse location data
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in)
    {
        //protocol header
        in.readByte();
        //DeviceID
        byte[] terminalNumArr = new byte[5];
        in.readBytes(terminalNumArr);
        String terminalNum = ByteBufUtil.hexDump(terminalNumArr);
        //Protocol version
        int version = in.readUnsignedByte();
        short tempByte = in.readUnsignedByte();
```

```
        //Device type
        int terminalType = tempByte >> 4;
        //data type
        int dataType = tempByte & 0b00001111;
        //Data length
        int dataLen = in.readUnsignedShort();
        //GPS g time
        byte[] timeArr = new byte[6];
        in.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = parseBcdTime(bcdTimeStr);
        //Latitude
        byte[] latArr = new byte[4];
        in.readBytes(latArr);
        String latHexStr = ByteBufUtil.hexDump(latArr);
        double lat = 0.0;
        BigDecimal latFloat = new BigDecimal(latHexStr.substring(2, 4) + ".
" + latHexStr.substring(4)).divide(new BigDecimal("60"), 6, RoundingMode.HA
LF_UP);
        lat = new BigDecimal(latHexStr.substring(0, 2)).add(latFloat).doubl
eValue();
        //Longitude
        byte[] lngArr = new byte[5];
        in.readBytes(lngArr);
        String lngHexStr = ByteBufUtil.hexDump(lngArr);
        double lng=0.0;
        BigDecimal lngFloat = new BigDecimal(lngHexStr.substring(3, 5) + ".
" + lngHexStr.substring(5, 9)).divide(new BigDecimal("60"), 6, RoundingMode
.HALF_UP);
        lng = new BigDecimal(lngHexStr.substring(0, 3)).add(lngFloat).doubl
eValue();
        //bit indication
        int bitFlag = Byte.parseByte(lngHexStr.substring(9, 10), 16);
        //Positioning status
        int locationType = (bitFlag & 0x01) > 0 ? 1 : 0;
        //north latitude, south latitude
        if ((bitFlag & 0b0010) == 0) {
            lat = -lat;
        }
        //East longitude and West longitude
        if ((bitFlag & 0b0100) == 0) {
            lng = -lng;
        }
        //Speed
        int speed = (int) (in.readUnsignedByte() * 1.85);
        //Header(direction)
```

```java
        int direction = in.readUnsignedByte() * 2;
        //Mileage
        long mileage = in.readUnsignedInt();
        //Number of GPS satellites
        int gpsSignal = in.readByte();
        //Bind vehicle ID
        long vehicleId = in.readUnsignedInt();
        //Device status
        int terminalStatus = in.readUnsignedShort();
        //Whether the base station is located
        if (NumberUtil.getBitValue(terminalStatus, 0) == 1) {
            locationType = 2;
        }
        //Battery indicator
        int batteryPercent = in.readUnsignedByte();
        //2G CELL ID
        int cellId2G = in.readUnsignedShort();
        //LAC
        int lac = in.readUnsignedShort();
        //GSM Signal quality
        int cellSignal = in.readUnsignedByte();
        //fence Alarm ID
        int regionAlarmId = in.readUnsignedByte();
        //Device status3
        int terminalStatus3 = in.readUnsignedByte();
        //Wakeup source
        int fWakeSource=(terminalStatus3 & 0b0000_1111);
        //reserved
        in.readShort();
        //IMEI No.
        byte[] imeiArr = new byte[8];
        in.readBytes(imeiArr);
        String imei = ByteBufUtil.hexDump(imeiArr);
        //3G CELL ID High 16 bits
        int cellId3G = in.readUnsignedShort();
        int cellId=0;
        if(cellId3G>0){
            cellId=(cellId3G<<16)+cellId2G;
        }else{
            cellId=cellId2G;
        }
        //MCC
        int mcc = in.readUnsignedShort();
        //MNC
        int mnc = in.readUnsignedByte();
        //Data serial number
```

```java
        int flowId = in.readUnsignedByte();
        //Parse alarm
        int fAlarm=parseLocationAlarm(terminalStatus);

        LocationData location=new LocationData();
        location.setProtocolType(version);
        location.setDeviceType(terminalType);
        location.setDataType(dataType);
        location.setDataLength(dataLen);
        location.setGpsTime(gpsZonedDateTime.toString());
        location.setLatitude(lat);
        location.setLongitude(lng);
        location.setLocationType(locationType);
        location.setSpeed(speed);
        location.setDirection(direction);
        location.setMileage(mileage);
        location.setGpsSignal(gpsSignal);
        location.setGSMSignal(cellSignal);
        location.setAlarm(fAlarm);
        location.setAlarmArea(regionAlarmId);
        location.setBattery(batteryPercent);
        location.setLockStatus(NumberUtil.getBitValue(terminalStatus, 7) ==
1 ? 0 : 1);
        location.setLockRope(NumberUtil.getBitValue(terminalStatus, 6) == 1
? 0 : 1);
        location.setBackCover(NumberUtil.getBitValue(terminalStatus, 13));
        location.setMCC(mcc);
        location.setMNC(mnc);
        location.setLAC(lac);
        location.setCELLID(cellId);
        location.setIMEI(imei);
        location.setAlarm(fWakeSource);
        location.setIndex(flowId);
        //Define the location data entity class
        Result model = new Result();
        model.setDeviceID(terminalNum);
        model.setMsgType("Location");
        model.setDataBody(location);
        if (version < 0x19) {
            model.setReplyMsg("(P35)");
        } else {
            model.setReplyMsg(String.format("(P69,0,%s)",flowId));
        }
        return model;
    }
```

```java
    /**
     * Analyze and locate alarms
     * @param terminalStatus
     * @return
     */
    private static int parseLocationAlarm(int terminalStatus)
    {
        //Trigger alarm or not
        int fAlarm = -1;
        //Acknowledge or not
        if (NumberUtil.getBitValue(terminalStatus, 5) == 1) {
            //Judgment alarm
            if (NumberUtil.getBitValue(terminalStatus, 1) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_9.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 2) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_10.getVa
lue());
            } else if (NumberUtil.getBitValue(terminalStatus, 3) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_1.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 4) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_2.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 8) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_3.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 9) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_4.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 10) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_5.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 11) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_6.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 12) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_7.getVal
ue());
            } else if (NumberUtil.getBitValue(terminalStatus, 14) == 1) {
                fAlarm = Integer.parseInt(AlarmTypeEnum.LOCK_ALARM_8.getVal
ue());
            } else {
                fAlarm = -1;
            }
        }
```

```
        return fAlarm;
    }

    /**
     * Command response reply
     * @param msgType
     * @param itemList
     * @return
     */
    private static String replyMessage(String msgType,List<String> itemList
)
    {
        String replyContent = null;
        switch (msgType)
        {
            case "P22":
                ZonedDateTime currentDateTime = ZonedDateTime.now(ZoneOffse
t.UTC);
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern("
ddMMyyHHmmss");
                replyContent = String.format("(P22,%s)", currentDateTime.fo
rmat(formatter));
                break;
            case "P43":
                if (itemList.get(2).equals("0")) {
                    //reset Password
                    replyContent = String.format("(P44,1,888888)");
                }
                break;
            case "P45":
                replyContent = String.format("(P69,0,%s)", itemList.get(16)
);
                break;
            case "P52":
                if (itemList.get(2).equals("2")) {
                    replyContent = String.format("(P52,2,%s)", itemList.get
(3));
                }
                break;
            default:
                break;
        }
        return replyContent;
    }

    /**
```

```
     * Command response reply
     * @param msgType
     * @param index
     * @return
     */
    public static String replyMessage(String msgType, int index)
    {
        String replyContent = null;
        switch (msgType)
        {
            case "WLNET5":
            case "WLNET7":
                replyContent = String.format("(P69,0,{0})", index);
                break;
            default:
                break;
        }
        return replyContent;
    }
}
```

public method class CommonUtil

```
package com.jointech.sdk.jt701.utils;

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;

/**
 * <p>Description:Used to store some public methods encountered in parsing
</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    private CommonUtil()
    {

    }

    /**
     * Unescaped text to transparently transmit data
```

```
     *
     * @param in
     * @param frame
     * @param bodyLen
     */
    public static void unescape(ByteBuf in, ByteBuf frame, int bodyLen) {
        int i = 0;
        while (i < bodyLen) {
            int b = in.readUnsignedByte();
            if (b == 0x3D) {
                int nextByte = in.readUnsignedByte();
                if (nextByte == 0x14) {
                    frame.writeByte(0x3D ^ 0x14);
                } else if (nextByte == 0x15) {
                    frame.writeByte(0x3D ^ 0x15);
                } else if (nextByte == 0x00) {
                    frame.writeByte(0x3D ^ 0x00);
                } else if (nextByte == 0x11) {
                    frame.writeByte(0x3D ^ 0x11);
                } else {
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * remove last character from string
     * @param inStr input string
     * @param suffix characters to remove
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * Hexadecimal to byte[]
```

```
     * @param hex
     * @return
     */
    public static byte[] hexStr2Byte(String hex) {
        ByteBuffer bf = ByteBuffer.allocate(hex.length() / 2);
        for (int i = 0; i < hex.length(); i++) {
            String hexStr = hex.charAt(i) + "";
            i++;
            hexStr += hex.charAt(i);
            byte b = (byte) Integer.parseInt(hexStr, 16);
            bf.put(b);
        }
        return bf.array();
    }
}
```

Parse constants

```
package com.jointech.sdk.jt701.constants;

import java.util.Arrays;
import java.util.List;

/**
 * constant definition
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * binary message header
     */
    public static final byte BINARY_MSG_HEADER = '$';

    /**
     * text message header
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * text message trailer
     */
    public static final byte TEXT_MSG_TAIL = ')';
```

```
    /**
     * text message separator
     */
    public static final byte TEXT_MSG_SPLITER = ',';

    /**
     * Instructions for transparently transmitting binary data
     */
    public static final List<String> WLNET_TYPE_LIST = Arrays.asList("5", "
7");
}
```

2.4.Return message and description

（1）positioning data

Raw data：

```
2480405002251911003426032118530329532416031008941d0000000018070c0000000020e
04f8b0c56001f00020f0f0f0f0f0f0f0f0f0f00f2028f0157
```

Return message：

```
{
    "DeviceID": "8040500225",
    "DataBody": {
        "GpsTime": "2021-03-26T18:53:03Z",
        "MNC": 1,
        "BackCover": 1,
        "Index": 87,
        "Latitude": -29.88736,
        "Awaken": 0,
        "Direction": 0,
        "Battery": 79,
        "GpsSignal": 12,
        "DataType": 1,
        "AlarmArea": 0,
        "Speed": 0,
        "LockStatus": 0,
        "Mileage": 6151,
        "IMEI": "0f0f0f0f0f0f0f0f",
        "MCC": 655,
        "Longitude": 31.014902,
        "LAC": 22016,
        "DeviceType": 1,
```

```
        "ProtocolType": 25,
        "Alarm": 2,
        "DataLength": 52,
        "CELLID": 15895308,
        "LockRope": 0,
        "LocationType": 1,
        "GSMSignal": 31
    },
    "ReplyMsg": "(P69,0,87)",
    "MsgType": "Location"
}
```

Return message description

```
{
    "DeviceID" : DeviceID
    "MsgType" : message type ,  Here is: Location, which means location data,
    "DataBody" : message body
    {
        "ProtocolType": Protocol version,
        "DeviceType": Device type,
        "DataType": Data type number (1 means the real-time binary position
ing data, 2 means alarm data, 3 means blind area regular binary positioning
 data, 4 means second-new binary positioning data)
        "DataLength": Datalength,
        "GpsTime": Positioning time (GMT time),
        "Latitude": Latitude (dd.dddd format),
        "Longitude": Longitude (dd.dddd format),
        "LocationType": Positioning type (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Speed": Speed (unit: km/h),
        "Direction": Direction (0~360; 0 and 360 indicate true north direct
ion),
        "Mileage": Current mileage value (unit: km),
        "GpsSignal": Number of GPS satellites,
        "GSMSignal": GSM signal value,
        "Alarm": Alarm type ( - 1: No alarm information; 1: Lock rope cut;
2: Vibration; 3: Long time unlocking; 4: Unlock password is wrong for 5 con
secutive times; 5: Swipe illegal card; 6: Low battery; 7: Back cover openin
g ; 8: Motor struck; 9: Enter fence alarm; 10: Exit fence alarm),
        "AlarmArea": If the alarm is related to the area, the value here is
 the area ID,
        "Battery": Battery value ( 0~100; 255: charging),
        "LockStatus": Lock motor status (1: ON; 0: OFF),
        "LockRope": Lock rope status (1: unplugged; 0: inserted),
```

```
        "BackCover": Back cover status (1: closed; 0: open),
        "MCC": country code,
        "MNC": carrier code,
        "LAC": location area code,
        "CELLID": base station number,
        "IMEI": IMEI number, all 0F is invalid,
        "Awaken": wakeup source ( 0 restart,
         1: RTC wake-up, 2: Vibrate, 3: Open back cover, 4: insert/pull out
 lock rope, 5: Connect external power, 6: Swipe card, 7: Door sensor, 8: VI
P SMS, 9: Non-VIP SMS or spam),
        "Index": Data serial number
    },
    "ReplyMsg" : Reply content (if it is an empty string, the platform no ne
ed to reply to this message)
}
```

(2) Sensors collect data (WLNET5)

Raw data(SensorType=1) :

```
283830353035303037332c312c3134312c574c4e45542c352c322c2603211847096496729
53949673d1408ff00260321184718102011098666013362 3c0100d71b00000029
```

Return message(SensorType=1) :

```
{
    "DeviceID": "8050500073",
    "DataBody": {
        "SensorID": "1020110986",
        "SensorType": 1,
        "Speed": 471,
        "GpsTime": "2021-03-26T18:47:09Z",
        "Temperature": 21.5,
        "LockStatus": 1,
        "Index": 102,
        "Latitude": -65.612158,
        "Direction": 0,
        "Longitude": -395.61215,
        "DateTime": "2021-03-26T18:47:18Z",
        "RSSI": 60,
        "Humidity": 27,
        "Voltage": "3.07",
        "LockTimes": -1,
        "Event": -1,
        "LockRope": 1,
        "LocationType": 0,
```

```
            "Power": 98
        },
        "ReplyMsg": "(P69,0,102)",
        "MsgType": "WLNET5"
    }
```

Raw data(SensorType=4)：

```
28373030303331333330392C312C3038312C574C4E45542C352C322C05082115430722348250113550300F0000050821154304E0171E086925018D4E690400400000002A0029
```

Return message(SensorType=4)：

```
{
    "DeviceID": "7000313309",
    "MsgType": "WLNET5",
    "DataBody": {
        "GpsTime": "2021-08-05T15:43:07",
        "Latitude": 22.580416666666668,
        "Longitude": 113.91716666666667,
        "LocationType": 1,
        "Speed": 0,
        "Direction": 0,
        "SensorID": "E0171E0869",
        "LockStatus": 0,
        "LockRope": 0,
        "LockTimes": 42,
        "Index": 37,
        "Voltage": "3.97",
        "Power": 78,
        "RSSI": -105,
        "DateTime": "2021-08-05T15:43:04",
        "SensorType": 4,
        "Temperature": 0.0,
        "Humidity": 0,
        "Event": 6
    },
    "ReplyMsg": "(P69,0,37)"
}
```

Return message description

```
{
    "DeviceID": DeviceID,
```

```
    "MsgType": message type ,  Here is: WLNET5, which means that the sensor
transparently transmits data,
    "DataBody": message body
    {
        "GpsTime": Positioning time (GMT time),
        "Latitude": Latitude (dd.dddd format),
        "Longitude": Longitude (dd.dddd format),
        "LocationType": Positioning type (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Speed": Speed (unit: km/h); 0xFF means invalid speed,
        "Direction": direction (0~360; 0 and 360 indicate true north direct
ion),
        "SensorID":sensor ID,
        "LockStatus": This is only valid if the slave type SensorType=4; 1:
 unlocked; 0: locked,
        "LockRope": This is only valid if the slave type SensorType=4; 1: t
he lock rope is pulled out; 0: the lock rope is inserted,
        "LockTimes": Number of unlocks (this value is only valid if SensorT
ype=4),
        "Index": data serial number,
        "Voltage": Voltage value (unit: V),
        "Power": Sensor power (0~100; 255: charging),
        "RSSI":RSSI signal strength, the closer the negative number is to 0
, the better the signal,
        "DateTime": Data collection time (GMT time),
        "SensorType": Sensor type (1: temperature and humidity sensor (JT12
6); 4: slave JT709;),
        "Temperature":Temperature value (only valid if SensorType=1),
        "Humidity": Humidity value (only valid if SensorType=1),
        "Event": Event type (-1: no slave event; 0: lock event; 1: Bluetoot
h unlock event; 2: NFC unlock event; 3: Lora unlock event; 4: slave lock cl
ipping alarm event; 5: key-press wake-up event; 6 : Timing reporting event;
 7: Charging reporting event)
    },
    "ReplyMsg": Reply content (if it is an empty string,  no need to reply
to the message from platform) )
}
```

(3) Lock event reporting data (P45)

Rawdata :

```
28373839303632393238342c5034352c3236303332312c3139343933392c32362e323732303
3352c4e2c35302e3632313433352c452c412c302e30352c302c342c312c3030303030303030
30302c312c302c3129
```

Return messsage：

```
{
    "DeviceID": "7890629284",
    "MsgType": "P45",
    "DataBody": {
        "DateTime": "2021-03-26T19:49:39",
        "Latitude": 26.272035,
        "Longitude": 50.621435,
        "LocationType": 1,
        "Speed": 0,
        "Direction": 0,
        "Event": 4,
        "Status": 1,
        "UnlockFenceID": -1,
        "RFIDNo": "0000000000",
        "PsdErrorTimes": 0,
        "Index": 1,
        "Mileage": 0
    },
    "ReplyMsg": "(P69,0,1)"
}
```

Return message description：

```
{
    "DeviceID": Device ID,
    "MsgType": The message type, here is: P45, which means the switch lock
event data upload,
    "DataBody": message body
    {
        "DateTime": event time (GMT time),
        "Latitude": Latitude (dd.dddd format),
        "Longitude": Longitude (dd.dddd format),
        "LocationType": Positioning type (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Speed": Speed (unit: km/h),
        "Direction": direction (0~360; 0 and 360 indicate true north direct
ion),
        "Event": Event type (1: means swiping authorization card; 2: means
swiping illegal card; 3: means swiping vehicle ID card binding; 4: means un
locking with password; 5: means terminal automatic lock record; 6: in dynam
ic password fence unlock; 7: Bluetooth unlock),
        "Status": Unlock verification (0: the unlock password is incorrect;
 1: normal unlock; 2: because the fence is unlocked, it is not unlocked in
```

```
the fence, and the unlock is rejected),
        "UnlockFenceID": (-1: Unlocking has nothing to do with the fence; 1
~10: Identifies the corresponding unlocking fence ID),
        "RFIDNo": Swipe card number; if not "0000000000", it is invalid,
        "PsdErrorTimes": Number of incorrect unlock passwords,
        "Index": data serial number,
        "Mileage": Current mileage value (unit: km)
    },
    "ReplyMsg": Reply content (if it is an empty string, there is no need t
o reply to the message)
}
```

(4) Other commands reply data

Raw data：

```
28373839303632393238342c50333529
```

return message：

```
{
    "DeviceID": "7890629284",
    "MsgType": "P35",
    "DataBody": "(7890629284,P35)",
    "ReplyMsg": ""
}
```

Return message description：

```
{
    "DeviceID": Device ID,
    "MsgType": Message type (see more message types and descriptions, pleas
e refer to 3. Message types and message body content description),
    "DataBody": The content of the message body (except for the location da
ta: Location; the sensor transparent transmission data: WLNET5; the lock ev
ent report: P45; the ASCII strings of the command content are directly retu
rned here),
    "ReplyMsg": (If it is an empty string, there is no need to reply to the
 message)
}
```

JT704&JT706 SDK

Java

1.Jar package download

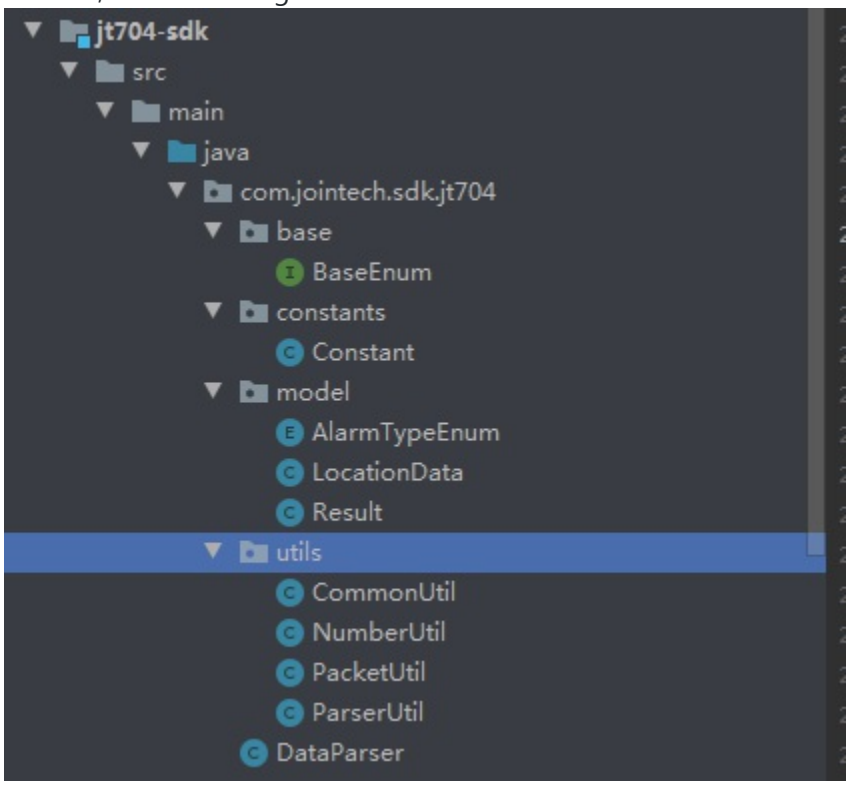
jt704-sdk-1.0.0.jar Download
If you need Jar package development source code, please contact the business application

---

2.Integrated Development Instructions

2.1.Integrated development language and framework description

jt704-sdk-1.0.0.jar is based on the Java language，SpringBoot2.x frame，use netty，fastjson，lombok,commons-lang3



BaseEnum：base enumeration
Constant：custom constant
AlarmTypeEnum：Alarm enumeration
EventTypeEnum：event enumeration
LocationData：Location entity class
LockEvent：lock event entity class
Result：result entity class
SensorData：Slave data entity class

CommonUtil：public method class
NumberUtil：digital manipulation tools
ParserUtil：Parser method tool class
DataParser：Parser main method

2.2.Integration Instructions

Introduce jt704-sdk-1.0.0.jar into your gateway program as follows：
Introduce the jar package in pom.xml

```
        <dependency>
            <groupId>com.jointech.sdk</groupId>
            <artifactId>jt704-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>
```

Call jt704-sdk-1.0.0.jar, the receiveData() method in the DataParser class
receiveData() method is overloaded

```
    /**
     * Parse Hex string raw data
     * @param strData hexadecimal string
     * @return
     */
    public static Object receiveData(String strData) {
        int length=strData.length()%2>0?strData.length()/2+1:strData.length
()/2;
        ByteBuf msgBodyBuf = Unpooled.buffer(length);
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse byte[] raw data
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
```

```
     * Parse ByteBuf raw data
     * @param in
     * @return
     */
    private static Object receiveData(ByteBuf in)
    {
        Object decoded = null;
        in.markReaderIndex();
        int header = in.readByte();
        if (header == Constant.TEXT_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeTextMessage(in);
        } else if (header == Constant.BINARY_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeBinaryMessage(in);
        } else {
            return null;
        }
        return JSONArray.toJSON(decoded).toString();
    }
```

2.3.Core code

ParserUtil：Parsing method tool class ParserUtil

```
package com.jointech.sdk.jt704.utils;

import com.jointech.sdk.jt704.constants.Constant;
import com.jointech.sdk.jt704.model.AlarmTypeEnum;
import com.jointech.sdk.jt704.model.LocationData;
import com.jointech.sdk.jt704.model.Result;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;
import org.apache.commons.lang3.StringUtils;

import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Description: Analysis method tool class</p>
 * @author HyoJung
 * @date 20210526
 */
```

```java
public class ParserUtil {
    /**
     * Parse Positioning data 0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //unexcape rawdata
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //message length
        int msgLen = msg.readableBytes();
        //packet header
        msg.readByte();
        //message ID
        int msgId = msg.readUnsignedShort();
        //message body properties
        int msgBodyAttr = msg.readUnsignedShort();
        //message body length
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //Whether to subcontract
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //Remove the base length of the message body
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //The following packet length is obtained according to the length o
f the message body and whether it is subcontracted
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
        //array of deviceID
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //Device ID (remove leading 0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //message serial number
        int msgFlowId = msg.readUnsignedShort();
        //total number of message packets
        int packetTotalCount = 0;
        //package serial number
        int packetOrder = 0;
        //subcontract
        if (multiPacket) {
            packetTotalCount = msg.readShort();
```

```
            packetOrder = msg.readShort();
        }
        //message body
        byte[] msgBodyArr = new byte[msgBodyLen];
        msg.readBytes(msgBodyArr);
        if(msgId==0x0200) {
            //Parse the message body
            LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
            locationData.setIndex(msgFlowId);
            locationData.setDataLength(msgBodyLen);
            //check code
            int checkCode = msg.readUnsignedByte();
            //packet end
            msg.readByte();
            //Get message response content
            String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
            //Define the location data entity class
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("Location");
            model.setDataBody(locationData);
            model.setReplyMsg(replyMsg);
            return model;
        }else {
            //Define the location data entity class
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("heartbeat");
            model.setDataBody(null);
            return model;
        }
    }

    /**
     * Parse instruction data
     * @param in raw data
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

        //The read pointer is set to the message header
        in.markReaderIndex();
        //Look for the end of the message, if not found, continue to wait f
or the next packet
```

```
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //Define the location data entity class
        Result model = new Result();
        //packet header(
        in.readByte();
        //Field list
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //Query the subscript of comma to intercept data
            int index = in.bytesBefore(Constant.TEXT_MSG_SPLITER);
            int itemLen = index > 0 ? index : in.readableBytes() - 1;
            byte[] byteArr = new byte[itemLen];
            in.readBytes(byteArr);
            in.readByte();
            itemList.add(new String(byteArr));
        }
        String msgType = "";
        if (itemList.size() >= 5) {
            msgType = itemList.get(3) + itemList.get(4);
        }
        Object dataBody=null;
        if(itemList.size()>0){
            dataBody="(";
            for(String item :itemList) {
                dataBody+=item+",";
            }
            dataBody=CommonUtil.trimEnd(dataBody.toString(),",");
            dataBody += ")";
        }
        String replyMsg="";
        if(msgType.equals("BASE2")&&itemList.get(5).toUpperCase().equals("T
IME")) {
            replyMsg=PacketUtil.replyBASE2Message(itemList);
        }
        model.setDeviceID(itemList.get(0));
        model.setMsgType(msgType);
        model.setDataBody(dataBody);
        model.setReplyMsg(replyMsg);
        return model;
    }

    /**
```

```java
     * Parse and locate message body
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
        //alarm sign
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //Device status
        long status = msgBodyBuf.readUnsignedInt();
        //Latitude
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Longitude
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Altitude, in meters
        int altitude = msgBodyBuf.readShort();
        //Speed
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //directioin
        int direction = msgBodyBuf.readShort();
        //GPS time
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
        //Determine whether the south latitude and west longitude are based
 on the value of the status bit
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //Positioning status
        int locationType=NumberUtil.getBitValue(status, 18);
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 1);
        }
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 6) > 0 ? 2 : 0;
        }
```

```java
        //Alarm status
        int alarm=-1;
        if(NumberUtil.getBitValue(alarmFlag, 16) == 1)
        {
            alarm=Integer.parseInt(AlarmTypeEnum.ALARM_1.getValue());
        }
        //Back cover status
        int backCover=NumberUtil.getBitValue(status, 7);
        //wake-up source
        long awaken = (status>>24)&0b00001111;

        LocationData locationData=new LocationData();
        locationData.setGpsTime(gpsZonedDateTime.toString());
        locationData.setLatitude(lat);
        locationData.setLongitude(lon);
        locationData.setLocationType(locationType);
        locationData.setSpeed((int)speed);
        locationData.setDirection(direction);
        locationData.setAlarm(alarm);
        locationData.setBackCover(backCover);
        locationData.setAwaken((int)awaken);
        //Handling additional information
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }

    /**
     * Parse additional information
     *
     * @param msgBody
     * @param location
     */
    private static void parseExtraInfo(ByteBuf msgBody, LocationData locati
on) {
        ByteBuf extraInfoBuf = null;
        while (msgBody.readableBytes() > 1) {
            int extraInfoId = msgBody.readUnsignedByte();
            int extraInfoLen = msgBody.readUnsignedByte();
            if (msgBody.readableBytes() < extraInfoLen) {
                break;
            }
            extraInfoBuf = msgBody.readSlice(extraInfoLen);
            switch (extraInfoId) {
                case 0x0F:
```

```
                    //Parsing temperature data
                    double temperature = -1000.0;
                    temperature = parseTemperature(extraInfoBuf.readShort()
 );
                    location.setTemperature((int)temperature);
                    break;
                //Wireless communication network signal strength
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //number of satellites
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //battery percentage
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //battery voltage
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setVoltage(fVoltage*0.01);
                    break;
                case 0xFD:
                    //Cell code information
                    int mcc=extraInfoBuf.readUnsignedShort();
                    location.setMCC(mcc);
                    int mnc=extraInfoBuf.readUnsignedByte();
                    location.setMNC(mnc);
                    long cellId=extraInfoBuf.readUnsignedInt();
                    location.setCELLID((int)cellId);
                    int lac=extraInfoBuf.readUnsignedShort();
                    location.setLAC(lac);
                    break;
                case 0xFE:
                    long mileage = extraInfoBuf.readUnsignedInt();
                    location.setMileage(mileage);
                    break;
                default:
                    ByteBufUtil.hexDump(extraInfoBuf);
                    break;
            }
        }
```

```
    }

    /**
     * Parse temperature
     * @param temperatureInt
     * @return
     */
    private static double parseTemperature(int temperatureInt) {
        if (temperatureInt == 0xFFFF) {
            return -1000;
        }
        double temperature = ((short) (temperatureInt << 4) >> 4) * 0.1;
        if ((temperatureInt >> 12) > 0) {
            temperature = -temperature;
        }
        return temperature;
    }
}
```

PacketUtil：Used to process preprocessed data and restore method encapsulation class

```
package com.jointech.sdk.jt704.utils;

import com.jointech.sdk.jt704.constants.Constant;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.ByteBufUtil;
import io.netty.util.ReferenceCountUtil;

import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Random;

/**
 * Parse package preprocessing (do unescaping)
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * Parse message packets
     *
     * @param in
     * @return
```

```java
     */
    public static Object decodePacket(ByteBuf in) {
        //The readable length cannot be less than the base length
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //Prevent illegal code stream attacks, the data is too large to be abnormal data
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //Look for the end of the message, if not found, continue to wait for the next packet
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //Create a ByteBuf to store the reversed data
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //The data between the header and the end of the message is escaped
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //The length after the reverse escape cannot be less than the base length
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
        return frame;
    }

    /**
     * In the message header, message body and check code, 0x7D 0x02 is reversed to 0x7E, and 0x7D 0x01 is reversed to 0x7D
     *
     * @param in
```

```
     * @param frame
     * @param bodyLen
     */
    public static void unescape(ByteBuf in, ByteBuf frame, int bodyLen) {
        int i = 0;
        while (i < bodyLen) {
            int b = in.readUnsignedByte();
            if (b == 0x7D) {
                int nextByte = in.readUnsignedByte();
                if (nextByte == 0x01) {
                    frame.writeByte(0x7D);
                } else if (nextByte == 0x02) {
                    frame.writeByte(0x7E);
                } else {
                    //abnormal data
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * In the message header, message body and check code, 0x7E is escaped
as 0x7D 0x02, and 0x7D is escaped as 0x7D 0x01
     *
     * @param out
     * @param bodyBuf
     */
    public static void escape(ByteBuf out, ByteBuf bodyBuf) {
        while (bodyBuf.readableBytes() > 0) {
            int b = bodyBuf.readUnsignedByte();
            if (b == 0x7E) {
                out.writeShort(0x7D02);
            } else if (b == 0x7D) {
                out.writeShort(0x7D01);
            } else {
                out.writeByte(b);
            }
        }
    }
```

```java
    /**
     * reply content
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //Remove the length of the head and tail
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //message ID
            bodyBuf.writeShort(0x8001);
            //data length
            bodyBuf.writeShort(0x0005);
            //Device ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //Generate random numbers from 1-65534
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //current message serial number
            bodyBuf.writeShort(index);
            //response message serial number
            bodyBuf.writeShort(msgFlowId);
            //response message ID
            bodyBuf.writeShort(0x0200);
            //response result
            bodyBuf.writeByte(0x00);
            //check code
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //packet header
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //The read pointer is reset to the starting position
            bodyBuf.readerIndex(0);
            //escape
            PacketUtil.escape(replyBuf, bodyBuf);
            //packet end
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
```

```
        }
    }

    /**
     * Timing command reply
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //set date format
            ZonedDateTime currentDateTime = ZonedDateTime.now(ZoneOffset.UT
C);
            String strBase2Reply = String.format("(%s,%s,%s,%s,%s,%s)", ite
mList.get(0), itemList.get(1)
                    , itemList.get(2), itemList.get(3), itemList.get(4), Da
teTimeFormatter.ofPattern("yyyyMMddHHmmss").format(currentDateTime));
            return strBase2Reply;
        }catch (Exception e) {
            return "";
        }
    }
}
```

NumberUtil：digital manipulation tools

```
package com.jointech.sdk.jt704.utils;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

/**
 * digital tools
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     * Coordinate accuracy
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     * Coordinate factor
```

```java
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * One decimal place precision
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     * Format message ID (convert to 0xXXXX)
     *
     * @param msgId Message ID
     * @return format string
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * format short numbers
     *
     * @param num number
     * @return format string
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
     * Convert 4-digit hexadecimal string
     *
     * @param num digits
     * @return format string
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * Parse the value of type short and get the number of digits whose val
ue is 1
     *
```

```java
     * @param number
     * @return
     */
    public static List<Integer> parseShortBits(int number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 16; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Parse the value of type int and get the number of digits whose value
  is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseIntegerBits(long number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 32; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Get the value of the index-th bit in binary
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

    /**
     * bit list to int
     *
     * @param bits
     * @param len
```

```java
     * @return
     */
    public static int bitsToInt(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        int result = Integer.parseInt(new String(chars), 2);
        return result;
    }

    /**
     * bit list to long
     *
     * @param bits
     * @param len
     * @return
     */
    public static long bitsToLong(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0L;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        long result = Long.parseLong(new String(chars), 2);
        return result;
    }

    /**
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(long longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
```

```
doubleValue();
    }

    /**
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(int longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
doubleValue();
    }
}
```

CommonUtil：public method class

```
package com.jointech.sdk.jt704.utils;

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

/**
 * <p>Description: Used to store some public methods encountered in parsing
</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * remove last character from string
     * @param inStr input string
     * @param suffix characters to remove
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
```

```
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * Hexadecimal to byte[]
     * @param hex
     * @return
     */
    public static byte[] hexStr2Byte(String hex) {
        ByteBuffer bf = ByteBuffer.allocate(hex.length() / 2);
        for (int i = 0; i < hex.length(); i++) {
            String hexStr = hex.charAt(i) + "";
            i++;
            hexStr += hex.charAt(i);
            byte b = (byte) Integer.parseInt(hexStr, 16);
            bf.put(b);
        }
        return bf.array();
    }

    /**
     *Convert GPS time
     *
     * @param bcdTimeStr
     * @return
     */
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyMMddHH
mmss");
        LocalDateTime localDateTime = LocalDateTime.parse(bcdTimeStr, forma
tter);
        ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, ZoneO
ffset.UTC);
        return zonedDateTime;
    }

    /**
     * XOR evaluation of each byte
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
```

```
            while (buf.readableBytes() > 0) {
                checksum ^= buf.readUnsignedByte();
            }
            return checksum;
        }
    }
```

LocationData：Location entity class

```
package com.jointech.sdk.jt704.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * <p>Description: Location entity class</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
@Data
public class LocationData implements Serializable {
    /**
     * Messagebody
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * positioning time
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * Latitude
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * Longitude
     */
    @JSONField(name = "Longitude")
```

```java
    public double Longitude;
    /**
     * Positioning type
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * Speed
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * Direction
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * Mileage
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * GpsSignal
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSMSignal
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * Battery
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * Voltage
     */
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * BackCover status
     */
    @JSONField(name = "BackCover")
    public int BackCover;
```

```
    /**
     * MCC
     */
    @JSONField(name = "MCC")
    public int MCC;
    /**
     * MNC
     */
    @JSONField(name = "MNC")
    public int MNC;
    /**
     * LAC
     */
    @JSONField(name = "LAC")
    public int LAC;
    /**
     * CELLID
     */
    @JSONField(name = "CELLID")
    public long CELLID;
    /**
     * Awaken
     */
    @JSONField(name = "Awaken")
    public int Awaken;
    /**
     * Alarm type
     */
    @JSONField(name = "Alarm")
    public int Alarm = -1;
    /**
     * Data serial number
     */
    @JSONField(name = "Index")
    public int Index;
    /**
     * Temperature
     */
    @JSONField(name = "Temperature")
    public int Temperature=-1000;
}
```

Result：result entity class

```
package com.jointech.sdk.jt704.model;
```

```java
import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * result entity class
 * @author HyoJung
 * @date 20210526
 */
@Data
public class Result implements Serializable {
    @JSONField(name = "DeviceID")
    private String DeviceID;
    @JSONField(name = "MsgType")
    private String MsgType;
    @JSONField(name = "DataBody")
    private Object DataBody;
    @JSONField(name = "ReplyMsg")
    private String ReplyMsg;
}
```

AlarmTypeEnum：Terminal alarm type enumeration

```java
package com.jointech.sdk.jt704.model;

import com.jointech.sdk.jt704.base.BaseEnum;
import lombok.Getter;

/**
 * Terminal alarm type
 * @author HyoJung
 */
public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("Main unit disassembly alarm", "1");

    @Getter
    private String desc;

    private String value;

    AlarmTypeEnum(String desc, String value) {
        this.desc = desc;
```

Java

```
            this.value = value;
        }

        @Override
        public String getValue() {
            return value;
        }

        public static AlarmTypeEnum fromValue(Integer value) {
            String valueStr = String.valueOf(value);
            for (AlarmTypeEnum alarmTypeEnum : values()) {
                if (alarmTypeEnum.getValue().equals(valueStr)) {
                    return alarmTypeEnum;
                }
            }
            return null;
        }
    }
```

2.4.Return message and description

（1）Heartbeat data
Raw data：

```
7E000200007501804283100001267E
```

return message：

```
{"DeviceID":"750180428310","MsgType":"heartbeat"}
```

Return message description

```
{"DeviceID":DeviceID,"MsgType":messagetype（heartbeat：heartbeat）}
```

（2）positioning data
Rawdata：

```
7E0200003B750180925004010200000000000400020158F0F06CA3C5200270000000018110
6123212D4015AD502015C30011431010CFD0901CC00000010922866EF014A0F0201406E7E
```

Return content：

```
{
    "DeviceID": "750180925004",
    "DataBody": {
        "Speed": 0,
        "GpsTime": "2018-11-06T12:32:12Z",
        "Temperature": 32,
        "MNC": 0,
        "Mileage": 0,
        "BackCover": 0,
        "Index": 258,
        "Latitude": 22.581007,
        "Awaken": 0,
        "MCC": 460,
        "Direction": 0,
        "Longitude": 113.91701,
        "LAC": 10342,
        "Alarm": -1,
        "Battery": 90,
        "GpsSignal": 12,
        "Voltage": 3.48,
        "DataLength": 59,
        "CELLID": 4242,
        "LocationType": 1,
        "GSMSignal": 20
    },
    "ReplyMsg": "7e8001000575018092500 40dbd0102020000077e",
    "MsgType": "Location"
}
```


Return message Description

```
{
    "DeviceID": DeviceID,
    "DataBody": {
        "Speed": Speed,
        "GpsTime": GPStime(UTC),
        "Temperature": Temperature,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "Mileage": Mileage(km),
        "BackCover": BackCover(1：开；0：关),
        "Index": Data serial number,
        "Latitude": Latitude(WGS84),
```

```
        "Longitude": Longitude ( WGS84 ) ,
        "Awaken": Wake-up source ( 0:RTC reporting 3 : Back Cover openning re
porting  4 : Close Cover reporting ) ,
        "Direction": Direction ( 0~360,0 means true north ) ,
        "Alarm": Alarm type ( 1 : Main unit disassembly alarm ; -1 : No alarm ) ,
        "Battery": BattteryLevel ( 0~100% ) ,
        "GpsSignal": number of satellites,
        "Voltage": Battery Voltage ( unit : V ) ,
        "DataLength": Data length,
        "LocationType": Positioning type ( 1 : GPS positioning ; 0 : No position
ing ) ,
        "GSMSignal": GSM signal strength
    },
    "ReplyMsg": Need to reply to the command of the device,
    "MsgType": Data type ( Location : Position data )
}
```

(3) Command data parsing

Rawdata :

```
28373030313630383138303030 2c312c3030312c424153452c312c312c32303135303431385
f473330302c302c4265694875616e2c313133374230335349 4d3930304d36345f53545f4d4d
532c383938363030343231393131333032373235343 92c303132323037303035363230393333
22c3436302c30302c343234332c3638373729
```

Return message :

```
{
    "DeviceID": "700160818000",
    "DataBody": "(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B
03SIM900M64_ST_MMS,898600421911302725 49,012207005620932,460,00,4243,6877)",
    "ReplyMsg": "",
    "MsgType": "BASE1"
}
```

Return message description

```
{
    "DeviceID": DeviceID,
    "DataBody": Message Content,
    "ReplyMsg": Reply to the device message (empty means no reply is requir
ed),
    "MsgType": command type
}
```

Java

JT705A SDK

Java

1.Jar package download

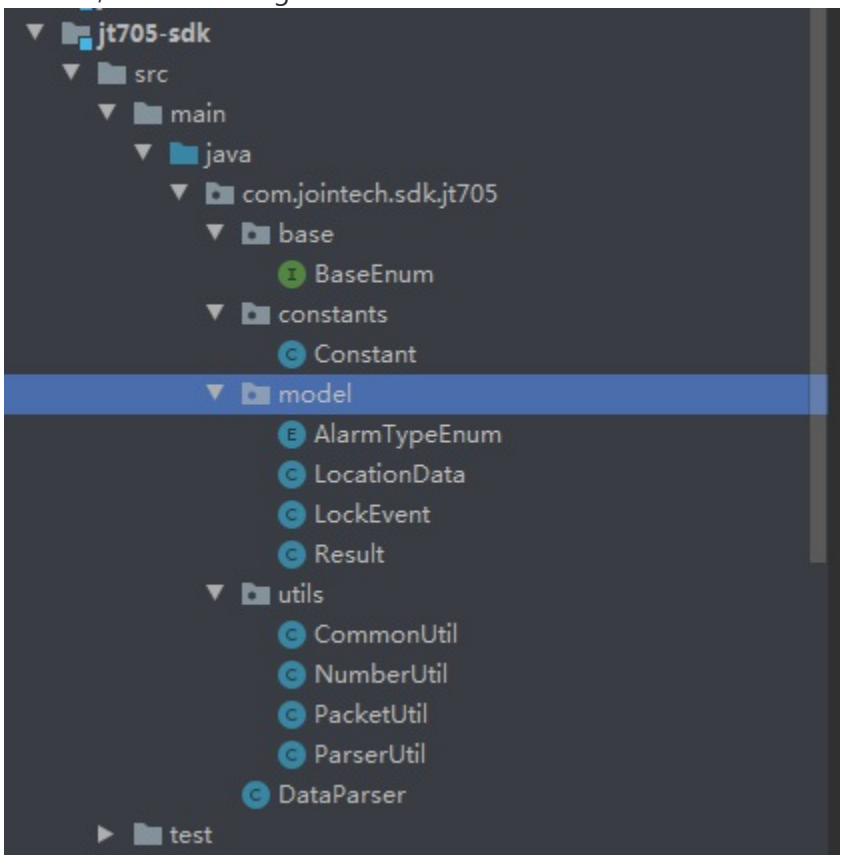
jt705-sdk-1.0.0.jar Download
If you need Jar package development source code, please contact the business application

---

2.Integrated Development Instruction

2.1.Integrated development language and framework description

jt705-sdk-1.0.0.jar is based on the Java language，SpringBoot2.x frame，use netty，fastjson，lombok,commons-lang3



BaseEnum：base enumeration
Constant：custom constant
AlarmTypeEnum：Alarm enumeration
EventTypeEnum：event enumeration
LocationData：Location entity class
LockEvent：lock event entity class
Result：result entity class

SensorData：Slave data entity class
CommonUtil：public method class
NumberUtil：digital manipulation tools
ParserUtil：Parser method tool class
DataParser：Parser main method

2.2.Integration Instructions

Introduce jt705-sdk-1.0.0.jar into your gateway program as follows：
Introduce the jar package in pom.xml

```
        <dependency>
            <groupId>com.jointech.sdk</groupId>
            <artifactId>jt705-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>
```

Call jt705-sdk-1.0.0.jar， the receiveData() method in the DataParser class
receiveData() method is overloaded

```
package com.jointech.sdk.jt705;

import com.alibaba.fastjson.JSONArray;
import com.jointech.sdk.jt705.constants.Constant;
import com.jointech.sdk.jt705.utils.CommonUtil;
import com.jointech.sdk.jt705.utils.ParserUtil;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;

/**
 * <p>Description: The body of the parsing method</p>
 *
 * @author lenny
 * @version 1.0.1
 */
public class DataParser {
    /**
     * Parse Hex string raw data
     * @param strData hexadecimal string
     * @return
     */
    public static Object receiveData(String strData) {
        int length=strData.length()%2>0?strData.length()/2+1:strData.length
()/2;
        ByteBuf msgBodyBuf = Unpooled.buffer(length);
```

```
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse byte[] raw data
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse ByteBuf raw data
     * @param in
     * @return
     */
    private static Object receiveData(ByteBuf in)
    {
        Object decoded = null;
        in.markReaderIndex();
        int header = in.readByte();
        if (header == Constant.TEXT_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeTextMessage(in);
        } else if (header == Constant.BINARY_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeBinaryMessage(in);
        } else {
            return null;
        }
        return JSONArray.toJSON(decoded).toString();
    }
}
```

2.3.Core code

ParserUtil：Parsing method tool class ParserUtil

```
package com.jointech.sdk.jt705.utils;
```

```java
import com.jointech.sdk.jt705.constants.Constant;
import com.jointech.sdk.jt705.model.AlarmTypeEnum;
import com.jointech.sdk.jt705.model.LocationData;
import com.jointech.sdk.jt705.model.LockEvent;
import com.jointech.sdk.jt705.model.Result;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;
import org.apache.commons.lang3.StringUtils;

import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Description:  Analysis method tool class</p>
 * @author HyoJung
 */
public class ParserUtil {
    /**
     * Parse Positioning data 0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //unexcape rawdata
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //message length
        int msgLen = msg.readableBytes();
        //packet header
        msg.readByte();
        //message ID
        int msgId = msg.readUnsignedShort();
        //message body properties
        int msgBodyAttr = msg.readUnsignedShort();
        //message body length
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //Whether to subcontract
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //Remove the base length of the message body
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //The following packet length is obtained according to the length o
f the message body and whether it is subcontracted
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
```

```
        if (msgLen != ensureLen) {
            return null;
        }
        //array of deviceID
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //Device ID (remove leading 0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //message serial number
        int msgFlowId = msg.readUnsignedShort();
        //total number of message packets
        int packetTotalCount = 0;
        //package serial number
        int packetOrder = 0;
        //subcontract
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //message body
        byte[] msgBodyArr = new byte[msgBodyLen];
        msg.readBytes(msgBodyArr);
        if(msgId==0x0200) {
            //Parse the message body
            LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
            locationData.setIndex(msgFlowId);
            locationData.setDataLength(msgBodyLen);
            //check code
            int checkCode = msg.readUnsignedByte();
            //packet end
            msg.readByte();
            //Get message response content
            String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
            //Define the location data entity class
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("Location");
            model.setDataBody(locationData);
            model.setReplyMsg(replyMsg);
            return model;
        }else {
            //Define the location data entity class
            Result model = new Result();
```

```
            model.setDeviceID(terminalNumber);
            model.setMsgType("heartbeat");
            model.setDataBody(null);
            return model;
        }
    }

    /**
     * Parse instruction data
     * @param in  raw data
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

        //The read pointer is set to the message header
        in.markReaderIndex();
        //Look for the end of the message, if not found, continue to wait f
or the next packet
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //Define the location data entity class
        Result model = new Result();
        //packet header(
        in.readByte();
        //Field list
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //Query the subscript of comma to intercept data
            int index = in.bytesBefore(Constant.TEXT_MSG_SPLITER);
            int itemLen = index > 0 ? index : in.readableBytes() - 1;
            byte[] byteArr = new byte[itemLen];
            in.readBytes(byteArr);
            in.readByte();
            itemList.add(new String(byteArr));
        }
        String msgType = "";
        if (itemList.size() >= 5) {
            msgType = itemList.get(3) + itemList.get(4);
        }
        Object dataBody=null;
        if(itemList.size()>0){
            dataBody="(";
            for(String item :itemList) {
```

```
                dataBody+=item+",";
            }
            dataBody=CommonUtil.trimEnd(dataBody.toString(),",");
            dataBody += ")";
        }
        String replyMsg="";
        if(msgType.equals("BASE2")&&itemList.get(5).toUpperCase().equals("T
IME")) {
            replyMsg=PacketUtil.replyBASE2Message(itemList);
        }
        model.setDeviceID(itemList.get(0));
        model.setMsgType(msgType);
        model.setDataBody(dataBody);
        model.setReplyMsg(replyMsg);
        return model;
    }

    /**
     * Parse and locate message body
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
        //alarm sign
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //Device status
        long status = msgBodyBuf.readUnsignedInt();
        //Latitude
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Longitude
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Altitude, in meters
        int altitude = msgBodyBuf.readShort();
        //Speed
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //directioin
        int direction = msgBodyBuf.readShort();
        //GPS time
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
```

```
        //Determine whether the south latitude and west longitude are based
  on the value of the status bit
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //Positioning status
        int locationType=NumberUtil.getBitValue(status, 18);
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 1);
        }
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 6) > 0 ? 2 : 0;
        }
        //Alarm status
        int lockRope=NumberUtil.getBitValue(status, 20);
        //Lock status
        int lockMotor=NumberUtil.getBitValue(status, 21);
        //The lock state (judged by the combination of the lock button + mo
  tor;when lock button open (1) or motor unlock(1),lock state is unlocking(1)
  ;when lock button close(0) and motor locked(0),the lock state is locked
        int lockStatus=0;
        if(lockRope==1||lockMotor==1) {
            lockStatus=1;
        }
        int backCover=NumberUtil.getBitValue(status, 7);
        //Wake-up source
        long awaken = (status>>24)&0b00001111;
        int alarm=parseAlarm(alarmFlag);
        LocationData locationData=new LocationData();
        locationData.setGpsTime(gpsZonedDateTime.toString());
        locationData.setLatitude(lat);
        locationData.setLongitude(lon);
        locationData.setLocationType(locationType);
        locationData.setSpeed((int)speed);
        locationData.setDirection(direction);
        locationData.setAltitude(altitude);
        locationData.setLockStatus(lockStatus);
        locationData.setLockRope(lockRope);
        locationData.setAwaken((int)awaken);
        locationData.setBackCover(backCover);
        locationData.setAlarm(alarm);
```

```
        //Handling additional information
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }

    /**
     * Parse additional information
     *
     * @param msgBody
     * @param location
     */
    private static void parseExtraInfo(ByteBuf msgBody, LocationData locati
on) {
        ByteBuf extraInfoBuf = null;
        while (msgBody.readableBytes() > 1) {
            int extraInfoId = msgBody.readUnsignedByte();
            int extraInfoLen = msgBody.readUnsignedByte();
            if (msgBody.readableBytes() < extraInfoLen) {
                break;
            }
            extraInfoBuf = msgBody.readSlice(extraInfoLen);
            switch (extraInfoId) {
                //Lock event
                case 0x0B:
                    LockEvent event=new LockEvent();
                    //Lock event
                    int type=extraInfoBuf.readUnsignedByte();
                    event.setType(type);
                    if(type==0x01||type==0x02||type==0x03||type==0x05||type
==0x1E||type==0x1F){
                        //Unlock by password
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
                        event.setPassword(password);
                        int unlockStatus=extraInfoBuf.readUnsignedByte();
                        if(unlockStatus==0xff){
                            event.setUnLockStatus(0);
                        }else{
                            if(unlockStatus>0&&unlockStatus<100){
                                event.setFenceId(unlockStatus);
                            }
                            event.setUnLockStatus(1);
                        }
```

```
                    }else if(type==0x06||type==0x07||type==0x08||type==0x10
||type==0x11||type==0x18||type==0x19||type==0x20||type==0x28||type==0x29){
                        //Unlock by password
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
                        event.setPassword(password);
                        event.setUnLockStatus(0);
                    }else if(type==0x22){
                        //RFID card No.
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                        if(extraInfoBuf.readableBytes()>0) {
                            int unlockStatus = extraInfoBuf.readUnsignedByt
e();
                            if (unlockStatus == 0xff) {
                                event.setUnLockStatus(0);
                            } else {
                                if (unlockStatus > 0 && unlockStatus < 100)
 {
                                    event.setFenceId(unlockStatus);
                                }
                                event.setUnLockStatus(1);
                            }
                        }else{
                            event.setUnLockStatus(1);
                        }
                    }else if(type==0x23||type==0x2A||type==0x2B){
                        //RFID card No.
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                    }
                    location.setLockEvent(event);
                    break;
                //Gyroscope three-axis data
                case 0x0C:
                    int x=extraInfoBuf.readUnsignedShort();
                    int y=extraInfoBuf.readUnsignedShort();
                    int z=extraInfoBuf.readUnsignedShort();
                    x=x > 32768 ? (x - 65536) : x;
```

```
                    y=y > 32768 ? (y - 65536) : y;
                    z=z > 32768 ? (z - 65536) : z;
                    location.setPosture("x:"+x+";y:"+y+";z:"+z);
                    break;
                //Wireless communication network signal strength
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //number of satellites
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //battery percentage
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //battery voltage
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setVoltage(fVoltage*0.01);
                    break;
                case 0xF9:
                    //Protocol version
                    int version=extraInfoBuf.readUnsignedShort();
                    location.setProtocolVersion(version);
                    break;
                case 0xFD:
                    //LBS information
                    int mcc=extraInfoBuf.readUnsignedShort();
                    location.setMCC(mcc);
                    int mnc=extraInfoBuf.readUnsignedByte();
                    location.setMNC(mnc);
                    long cellId=extraInfoBuf.readUnsignedInt();
                    location.setCELLID((int)cellId);
                    int lac=extraInfoBuf.readUnsignedShort();
                    location.setLAC(lac);
                    break;
                case 0xFC:
                    int fenceId = extraInfoBuf.readUnsignedByte();
                    location.setFenceId(fenceId);
                    break;
                case 0xFE:
                    long mileage = extraInfoBuf.readUnsignedInt();
```

```
                    location.setMileage(mileage);
                    break;
                default:
                    ByteBufUtil.hexDump(extraInfoBuf);
                    break;
            }
        }
    }

    /**
     * Alarm Parsing
     * @param alarmFlag
     * @return
     */
    private static int parseAlarm(long alarmFlag) {
        int alarm=-1;
        //Single trigger alarm
        if(NumberUtil.getBitValue(alarmFlag, 1) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_1.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 7) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_2.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 16) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_3.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 17) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_4.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 18) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_5.getValue());
        }
        return alarm;
    }
}
```

PacketUtil：Used to process preprocessed data and restore method encapsulation class

```
package com.jointech.sdk.jt705.utils;

import com.jointech.sdk.jt705.constants.Constant;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.ByteBufUtil;
```

```java
import io.netty.util.ReferenceCountUtil;

import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Random;

/**
 * Parse package preprocessing (do unescaping)
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * Parse message packets
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //The readable length cannot be less than the base length
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //Prevent illegal code stream attacks, the data is too large to be
abnormal data
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //Look for the end of the message, if not found, continue to wait f
or the next packet
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //Create a ByteBuf to store the reversed data
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //The data between the header and the end of the message is escaped
```

```
            unescape(in, frame, bodyLen);
            in.readByte();
            frame.writeByte(Constant.BINARY_MSG_HEADER);

            //The length after the reverse escape cannot be less than the base
 length
            if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
                ReferenceCountUtil.release(frame);
                return null;
            }
            return frame;
        }

        /**
         * In the message header, message body and check code, 0x7D 0x02 is rev
 ersed to 0x7E, and 0x7D 0x01 is reversed to 0x7D
         *
         * @param in
         * @param frame
         * @param bodyLen
         */
        public static void unescape(ByteBuf in, ByteBuf frame, int bodyLen) {
            int i = 0;
            while (i < bodyLen) {
                int b = in.readUnsignedByte();
                if (b == 0x7D) {
                    int nextByte = in.readUnsignedByte();
                    if (nextByte == 0x01) {
                        frame.writeByte(0x7D);
                    } else if (nextByte == 0x02) {
                        frame.writeByte(0x7E);
                    } else {
                        //abnormal data
                        frame.writeByte(b);
                        frame.writeByte(nextByte);
                    }
                    i += 2;
                } else {
                    frame.writeByte(b);
                    i++;
                }
            }
        }

        /**
         * In the message header, message body and check code, 0x7E is escaped
```

```
as 0x7D 0x02, and 0x7D is escaped as 0x7D 0x01
     *
     * @param out
     * @param bodyBuf
     */
    public static void escape(ByteBuf out, ByteBuf bodyBuf) {
        while (bodyBuf.readableBytes() > 0) {
            int b = bodyBuf.readUnsignedByte();
            if (b == 0x7E) {
                out.writeShort(0x7D02);
            } else if (b == 0x7D) {
                out.writeShort(0x7D01);
            } else {
                out.writeByte(b);
            }
        }
    }

    /**
     * Reply content
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //Remove the length of the head and tail
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //Message ID
            bodyBuf.writeShort(0x8001);
            //Data Length
            bodyBuf.writeShort(0x0005);
            //Device ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //Generate random numbers from 1-65534
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //Current message serial number
            bodyBuf.writeShort(index);
            //Reply message serial number
            bodyBuf.writeShort(msgFlowId);
            //Reply message ID
```

```
            bodyBuf.writeShort(0x0200);
            //Response result
            bodyBuf.writeByte(0x00);
            //check code
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //packet header
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //The read pointer is reset to the starting position
            bodyBuf.readerIndex(0);
            //escape
            PacketUtil.escape(replyBuf, bodyBuf);
            //packet end
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }

    /**
     * Timing command reply
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //set date format
            ZonedDateTime currentDateTime = ZonedDateTime.now(ZoneOffset.UT
C);
            String strBase2Reply = String.format("(%s,%s,%s,%s,%s,%s)", ite
mList.get(0), itemList.get(1)
                    , itemList.get(2), itemList.get(3), itemList.get(4), Da
teTimeFormatter.ofPattern("yyyyMMddHHmmss").format(currentDateTime));
            return strBase2Reply;
        }catch (Exception e) {
            return "";
        }
    }
}
```

NumberUtil：digital manipulation tools

```
package com.jointech.sdk.jt705.utils;
```

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

/**
 * digital tools
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     *  Coordinate accuracy
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     *  Coordinate factor
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * One decimal place precision
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     *  Format message ID (convert to 0xXXXX)
     *
     * @param msgId Message ID
     * @return return format string
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * format short numbers
     *
     * @param num number
     * @return format string
     */
```

```java
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
     * Convert 4-digit hexadecimal string
     *
     * @param num digits
     * @return format string
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * Parse the value of type short and get the number of digits whose val
ue is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseShortBits(int number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 16; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Parse the value of type int and get the number of digits whose value
 is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseIntegerBits(long number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 32; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
```

```
    }

    /**
     *  Get the value of the index-th bit in binary
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

    /**
     *  bit list to int
     *
     * @param bits
     * @param len
     * @return
     */
    public static int bitsToInt(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        int result = Integer.parseInt(new String(chars), 2);
        return result;
    }

    /**
     * bit list to long
     *
     * @param bits
     * @param len
     * @return
     */
    public static long bitsToLong(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0L;
        }
```

```
            char[] chars = new char[len];
            for (int i = 0; i < len; i++) {
                char value = bits.contains(i) ? '1' : '0';
                chars[len - 1 - i] = value;
            }
            long result = Long.parseLong(new String(chars), 2);
            return result;
        }

        /**
         * BigDecimal Multiplication
         *
         * @param longNum
         * @param precision
         * @return
         */
        public static double multiply(long longNum, BigDecimal precision) {
            return new BigDecimal(String.valueOf(longNum)).multiply(precision).
    doubleValue();
        }

        /**
         * BigDecimal Multiplication
         *
         * @param longNum
         * @param precision
         * @return
         */
        public static double multiply(int longNum, BigDecimal precision) {
            return new BigDecimal(String.valueOf(longNum)).multiply(precision).
    doubleValue();
        }
    }
```

CommonUtil：public method class

```
package com.jointech.sdk.jt705.utils;

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
```

```java
/**
 * <p>Description: Used to store some public methods encountered in parsing
</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * remove the last character of a string
     * @param inStr Input string
     * @param suffix characters to remove
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * Hexadecimal to byte[]
     * @param hex
     * @return
     */
    public static byte[] hexStr2Byte(String hex) {
        ByteBuffer bf = ByteBuffer.allocate(hex.length() / 2);
        for (int i = 0; i < hex.length(); i++) {
            String hexStr = hex.charAt(i) + "";
            i++;
            hexStr += hex.charAt(i);
            byte b = (byte) Integer.parseInt(hexStr, 16);
            bf.put(b);
        }
        return bf.array();
    }

    /**
     * Convert GPS time
     *
     * @param bcdTimeStr
     * @return
     */
```

```
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyMMddHH
mmss");
        LocalDateTime localDateTime = LocalDateTime.parse(bcdTimeStr, forma
tter);
        ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, ZoneO
ffset.UTC);
        return zonedDateTime;
    }

    /**
     * Each byte is XORed
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

BaseEnum：base enumeration

```
package com.jointech.sdk.jt705.base;

import java.io.Serializable;

/**
 * base enumeration
 * @author HyoJung
 */
public interface BaseEnum  <T> extends Serializable {
    T getValue();
}
```

Constant：custom constant

```
package com.jointech.sdk.jt705.constants;

/**
 * constant definition
```

Java

```
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * binary message header
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
     * Base length without message body
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;

    /**
     * message length
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * text message header
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * text message tail
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * text message delimiter
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

AlarmTypeEnum：Device alarm type enumeration

```
package com.jointech.sdk.jt705.model;

import com.jointech.sdk.jt705.base.BaseEnum;
import lombok.Getter;

/**
 * alarm type enumeration
 * @author HyoJung
```

Java

```
 */

public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("Speeding alarm", "1"),
    ALARM_2("Low Battery alarm", "2"),
    ALARM_3("Main unit Cover opening alarm", "3"),
    ALARM_4("Enter fence alarm", "4"),
    ALARM_5("Exit fence alarm", "5");

    @Getter
    private String desc;

    private String value;

    AlarmTypeEnum(String desc, String value) {
        this.desc = desc;
        this.value = value;
    }

    @Override
    public String getValue() {
        return value;
    }

    public static AlarmTypeEnum fromValue(Integer value) {
        String valueStr = String.valueOf(value);
        for (AlarmTypeEnum alarmTypeEnum : values()) {
            if (alarmTypeEnum.getValue().equals(valueStr)) {
                return alarmTypeEnum;
            }
        }
        return null;
    }
}
```

LocationData：Positiondata Entity Classes

```
package com.jointech.sdk.jt705.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;
```

```java
/**
 * <p>Description: Positiondata Entity Classes</p>
 *
 * @author lenny
 * @version 1.0.1
 */
@Data
public class LocationData implements Serializable {
    /**
     * Message body
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * Posintioning time
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * Latitude
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * Longitude
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * Positioning type
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * Speed
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * Direction(Header)
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * Mileage
     */
```

```java
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * Altitude
     */
    @JSONField(name = "Altitude")
    public int Altitude;
    /**
     * GPS signal
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM signal quatity
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * Battery level
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * Batttery voltage
     */
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * Lock status
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * Lock rope status
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * Back Cover status
     */
    @JSONField(name = "BackCover")
    public int BackCover;
    /**
     * Protocol version
     */
    @JSONField(name = "ProtocolVersion")
```

```
    public int ProtocolVersion;
    /**
     * Fence ID
     */
    @JSONField(name = "FenceId")
    public int FenceId;
    /**
     * MCC
     */
    @JSONField(name = "MCC")
    public int MCC;
    /**
     * MNC
     */
    @JSONField(name = "MNC")
    public int MNC;
    /**
     * LAC
     */
    @JSONField(name = "LAC")
    public int LAC;
    /**
     * CELLID
     */
    @JSONField(name = "CELLID")
    public long CELLID;
    /**
     * Awaken
     */
    @JSONField(name = "Awaken")
    public int Awaken;
    /**
     * Alarm
     */
    @JSONField(name = "Alarm")
    public int Alarm;
    /**
     * Lock&unlock event
     */
    @JSONField(name = "LockEvent")
    public LockEvent LockEvent;
    /**
     * attitude
     */
    @JSONField(name = "Posture")
    public String Posture;
```

```
    /**
     * Data Serial number
     */
    @JSONField(name = "Index")
    public int Index;
}
```

LockEvent：lock event entity class

```
package com.jointech.sdk.jt705.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

/**
 * lock event
 * @author HyoJung
 */
@Data
public class LockEvent {
    /**
     * Event type
     */
    @JSONField(name = "Type")
    public int Type;
    /**
     * Swipe RFID card number
     */
    @JSONField(name = "CardNo")
    public String CardNo;
    /**
     * Unlock password
     */
    @JSONField(name = "Password")
    public String Password;
    /**
     * Unlocking status(1:success；0：failed)
     */
    @JSONField(name = "UnLockStatus")
    public int UnLockStatus=0;
    /**
     * Fence ID related to unlocking
     */
    @JSONField(name = "FenceId")
    public int FenceId=-1;
```

```
}
```

Result：result entity class

```
package com.jointech.sdk.jt705.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * result entity class
 * @author HyoJung
 * @date 20210526
 */
@Data
public class Result implements Serializable {
    @JSONField(name = "DeviceID")
    private String DeviceID;
    @JSONField(name = "MsgType")
    private String MsgType;
    @JSONField(name = "DataBody")
    private Object DataBody;
    @JSONField(name = "ReplyMsg")
    private String ReplyMsg;
}
```

2.4.Return messages and instructions

（1）heartbeat data
Rawdata：

```
7E0002000075018042831000012 67E
```

Return message：

```
{"DeviceID":"750180428310","MsgType":"heartbeat"}
```

Return message description

```
{"DeviceID":DeviceID,"MsgType":Messagetype（heartbeat：heartbeat）}
```

（2）Position data

Rawdata（without lock&unlock event）：

```
7E0200004285600900101 40E560000000022340042015 88F9506CA3D93000B0012000020121
0073024D40127D502017B30011E310106FE040000001BFD0900000000000000000000C060000
0000FFDFD47E
```

Return message：

```
{
    "DeviceID": "856009001014",
    "DataBody": {
        "GpsTime": "2020-12-10T07:30:24Z",
        "MNC": 0,
        "FenceId": 0,
        "BackCover": 0,
        "Index": 3670,
        "Latitude": 22.581141,
        "Awaken": 2,
        "ProtocolVersion": 0,
        "Direction": 0,
        "Posture": "x:0;y:0;z:-33",
        "Battery": 39,
        "GpsSignal": 6,
        "Voltage": 3.79,
        "Speed": 1,
        "LockStatus": 1,
        "Mileage": 27,
        "MCC": 0,
        "Longitude": 113.917331,
        "LAC": 0,
        "Alarm": -1,
        "DataLength": 66,
        "CELLID": 0,
        "LockRope": 1,
        "LocationType": 1,
        "Altitude": 11,
        "GSMSignal": 30
    },
    "ReplyMsg": "7e8001000585600900101 4d81b0e56020000f57e",
    "MsgType": "Location"
}
```

Return message description

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": Positioning time ( UTC time ) ,
        "Latitude": latitude ( WGS84 ) ,
        "Longitude": longitude ( WGS84 ) ,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "FenceId": Geo-Fence ID,
        "BackCover": BackCover status ( 1 : Open ; 0 : Close ) ,
        "Index": Data Serial number,
        "Awaken":Wake-up source (1: RTC alarm wake-up, 2: Gsens vibration w
ake-up 3: open back cover wake-up 4: rope-cut wake-up 5: charging wake-up 6
: swipe card wake-up 7: Lora wake-up 8: VIP number wake-up 9: non-VIP wake-
up 10: Bluetooth wake-up )
        "ProtocolVersion":Protocol version,
        "Direction": True North is 0, clockwise0-360,
        "Battery": Battery percentage ( 0~100% ) ,
        "GpsSignal":Number of satellites currently received by GPS,
        "Voltage": Battery Voltage,unit: V,
        "Speed": Speed, unit:km/h,
        "LockStatus": DeviceLock Status ( 0 : locked ; 1 : unlock ) ,
        "LockRope": Lock rope status ( 0 : Inserted ; 1 : Pull out ) ,
        "Mileage": Mileage,Unit:km,
        "Alarm": Alarm type (1: Overspeed alarm; 2: Low power alarm; 3: Bac
k cover open alarm; 4: Entering the fence alarm; 5: Exiting the fence alarm
),
        "DataLength": Data length ( Bytes ) ,
        "LocationType": Positioning mode (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Altitude": Altitude,unit:km,
        "GSMSignal":GSM signal ,
        "Posture":Attitude (x: plus 180 degrees, minus 180 degrees; y: plus
 90 degrees, minus 90 degrees; z: plus 180 degrees, minus 180 degrees)
    },
    "ReplyMsg": The content of the device that needs to be replied to (empt
y means no reply is required)(platform response),
    "MsgType": Data type ( Location : Position data )
}
```

Rawdata (data with lock&lock event) ：

```
7E0200004C8560090010140E5400000000022400420158 8F9506CA3D93000B0012000020121
```

```
0073017D40127D502017B30011E310106FE040000001BFD0900000000000000000000B080538
3838383838650C0600000000FFC5A27E
```

Return message：

```
{
    "DeviceID": "856009001014",
    "DataBody": {
        "GpsTime": "2020-12-10T07:30:17Z",
        "MNC": 0,
        "FenceId": 0,
        "BackCover": 0,
        "Index": 3668,
        "Latitude": 22.581141,
        "Awaken": 2,
        "ProtocolVersion": 0,
        "Direction": 0,
        "Posture": "x:0;y:0;z:-59",
        "Battery": 39,
        "GpsSignal": 6,
        "Voltage": 3.79,
        "LockEvent": {
            "Type": 5,
            "FenceId": -1,
            "UnLockStatus": 1,
            "Password": "888888"
        },
        "Speed": 1,
        "LockStatus": 1,
        "Mileage": 27,
        "MCC": 0,
        "Longitude": 113.917331,
        "LAC": 0,
        "Alarm": -1,
        "DataLength": 76,
        "CELLID": 0,
        "LockRope": 0,
        "LocationType": 1,
        "Altitude": 11,
        "GSMSignal": 30
    },
    "ReplyMsg": "7e8001000585600900101 4c8750e54020000897e",
    "MsgType": "Location"
}
```

Return message description

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": Positioning time ( UTC time ) ,
        "Latitude": latitude ( WGS84 ) ,
        "Longitude": longitude ( WGS84 ) ,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "FenceId": Geo-Fence ID,
        "BackCover": BackCover status ( 1 : Open ; 0 : Close ) ,
        "Index": Data Serial number,
        "Awaken":Wake-up source (1: RTC alarm wake-up, 2: Gsens vibration w
ake-up 3: open back cover wake-up 4: rope-cut wake-up 5: charging wake-up 6
: swipe card wake-up 7: Lora wake-up 8: VIP number wake-up 9: non-VIP wake-
up 10: Bluetooth wake-up )
        "ProtocolVersion":Protocol version,
        "Direction": True North is 0, clockwise0-360,
        "Battery": Battery percentage ( 0~100% ) ,
        "GpsSignal":Number of satellites currently received by GPS,
        "Voltage": Battery Voltage,unit: V,
        "Speed": Speed, unit:km/h,
        "LockStatus": DeviceLock Status ( 0 : locked ; 1 : unlock ) ,
        "LockRope": Lock rope status ( 0 : Inserted ; 1 : Pull out ) ,
        "Mileage": Mileage,Unit:KM,
        "Alarm": Alarm type (1: Overspeed alarm; 2: Low power alarm; 3: Bac
k cover open alarm; 4: Entering the fence alarm; 5: Exiting the fence alarm
),
        "LockEvent": {
            "Type": Event type ( Refer to below Table1 ) ,
            "FenceId": Fence ID associated with the event (-1: no fence rel
ated),
            "UnLockStatus": Unlock status (1: unlock successful; 0: unlock
failed),
            "Password": Unlock password
            "CardNo":If the event type is swipe to unlock, here is the RFID
 card number
        },
        "DataLength": Data length ( Bytes ) ,
        "LocationType": Positioning mode (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Altitude": Altitude,unit:km,
        "GSMSignal":GSM signal ,
```

```
Java
```

```
        "Posture":Attitude (x: plus 180 degrees, minus 180 degrees; y: plus
 90 degrees, minus 90 degrees; z: plus 180 degrees, minus 180 degrees)
    },
    "ReplyMsg": The content of the device that needs to be replied to (empt
y means no reply is required)(platform response),
    "MsgType": Data type ( Location：Position data )
}
```

(3) Command data parse

Rawdata：

```
28383536303039303031303538 2C362C3030312C424153452C322C32303231303732343035353
531353529
```

Retrun message：

```
{
    "DeviceID": "856009001058",
    "DataBody": "(856009001058,6,001,BASE,2,20210724055155)",
    "ReplyMsg": "",
    "MsgType": "BASE2"
}
```

Retrun message description

```
{
    "DeviceID":DeviceID,
    "DataBody":Message content,
    "ReplyMsg": Reply to the device's message (empty means no reply is requ
ired)(platform response),
    "MsgType": command type
}
```

Table1

EventID（HEX）	EventID	Event description
0x01	1	Static password remote unlock
0x02	2	Dynamic password remote unlocking
0x03	3	Dynamic password on-site (Bluetooth or WIFI) APP unlock

0x05	5	Indicates static password on-site (Bluetooth or WIFI) APP unlocking
0x06	6	Wrong static password remote unlock
0x07	7	Wrong dynamic password remote unlock
0x08	8	Wrong dynamic password on-site (Bluetooth or WIFI) APP unlock
0x0B	11	Long unlock event
0x0C	12	Lock rope cutting event
0x0D	13	Lock events (automatic locked)
0x10	16	The remote execution of unlocking is abnormal, and the unlocking is not executed without positioning
0x11	17	The remote execution of unlocking is abnormal, and the unlocking will not be executed if it is positioned outside the fence
0x12	18	Abnormal motor
0x18	24	Unlocking is abnormal in Bluetooth execution, and unlocking is not performed without positioning
0x19	25	The Bluetooth unlocking is abnormal, and the unlocking is not performed if it is positioned outside the fence
0x1C	28	Unlock and pull out the lock rope
0x1E	30	SMS static password remote unlock
0x1F	31	SMS dynamic password remote unlock
0x20	32	Wrong SMS dynamic password
0x22	34	Swipe the authorization card to unlock the event
0x23	35	Swipe illegal card unlock event
0x28	40	Wrong static password on-site (Bluetooth or WIFI) APP unlock
0x29	41	Wrong SMS static password to unlock

0x2A	42	RFID performs unlocking abnormally, and does not perform unlocking without positioning
0x2B	43	RFID performs unlocking abnormally, if it is positioned outside the fence, it does not perform unlocking

JT707A&C SDK

Java

1.Jar package download

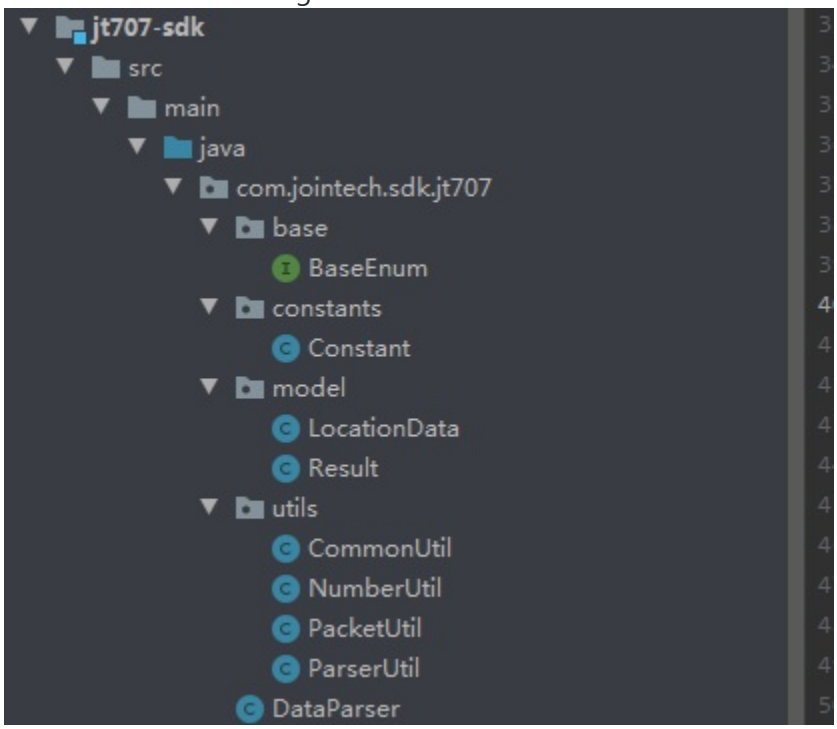
jt707-sdk-1.0.0.jar download
If you need Jar package development source code, please contact the business application

---

2.Integrated Development Instructions

2.1.Integrated development language and framework description

jt707-sdk-1.0.0.jar is based on the Java language，SpringBoot2.x frame,use netty，fastjson，lombok，commons-lang3



BaseEnum：base enumeration
Constant：custom constant
AlarmTypeEnum：Alarm enumeration
EventTypeEnum：event enumeration
LocationData：Location entity class
LockEvent：lock event entity class
Result：result entity class
SensorData：Slave data entity class
CommonUtil：public method class
NumberUtil：digital manipulation tools

ParserUtil：Parser method tool class
DataParser：Parser main method

2.2.Integration Instructions

Introduce jt707-sdk-1.0.0.jar into your gateway program as follows：
Introduce pom.xml package in pom.xml

```
        <dependency>
            <groupId>com.jointech.sdk</groupId>
            <artifactId>jt707-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>
```

Call jt707-sdk-1.0.0.jar，DataParser method in the DataParser class
receiveData() method is overloaded

```
    /**
     * Parse Hex string raw data
     * @param strData hexadecimal string
     * @return
     */
    public static Object receiveData(String strData) {
        int length=strData.length()%2>0?strData.length()/2+1:strData.length
()/2;
        ByteBuf msgBodyBuf = Unpooled.buffer(length);
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     *  Parse byte[] raw data
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse ByteBuf raw data
     * @param in
```

```
     * @return
     */
    private static Object receiveData(ByteBuf in)
    {
        Object decoded = null;
        in.markReaderIndex();
        int header = in.readByte();
        if (header == Constant.TEXT_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeTextMessage(in);
        } else if (header == Constant.BINARY_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeBinaryMessage(in);
        } else {
            return null;
        }
        return JSONArray.toJSON(decoded).toString();
    }
```

2.3.Core code

ParserUtil：Parsing method tool class ParserUtil

```
package com.jointech.sdk.jt707.utils;

import com.jointech.sdk.jt707.constants.Constant;
import com.jointech.sdk.jt707.model.LocationData;
import com.jointech.sdk.jt707.model.Result;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;
import org.apache.commons.lang3.StringUtils;

import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Description:Analysis method tool class</p>
 * @author HyoJung
 * @date 20210526
 */
public class ParserUtil {
    /**
     * Parse Positioning data 0x0200
```

```
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //unexcape rawdata
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //message length
        int msgLen = msg.readableBytes();
        //packet header
        msg.readByte();
        //message ID
        int msgId = msg.readUnsignedShort();
        //message body properties
        int msgBodyAttr = msg.readUnsignedShort();
        //message body length
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //Whether to subcontract
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //Remove the base length of the message body
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //The following packet length is obtained according to the length o
f the message body and whether it is subcontracted
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
         //array of deviceID
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //Device ID (remove leading 0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //message serial number
        int msgFlowId = msg.readUnsignedShort();
        //total number of message packets
        int packetTotalCount = 0;
        //package serial number
        int packetOrder = 0;
        //subcontract
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //message body
```

```
            byte[] msgBodyArr = new byte[msgBodyLen];
            msg.readBytes(msgBodyArr);
            if(msgId==0x0200) {
                //Parse the message body
                LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
                locationData.setIndex(msgFlowId);
                locationData.setDataLength(msgBodyLen);
                //check code
                int checkCode = msg.readUnsignedByte();
                //packet end
                msg.readByte();
                //Get message response content
                String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
                //Define the location data entity class
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("Location");
                model.setDataBody(locationData);
                model.setReplyMsg(replyMsg);
                return model;
            }else {
                //Define the location data entity class
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("heartbeat");
                model.setDataBody(null);
                return model;
            }
        }

        /**
         * Parse instruction data
         * @param in raw data
         * @return
         */
        public static Result decodeTextMessage(ByteBuf in) {

             //The read pointer is set to the message header
            in.markReaderIndex();
            //Look for the end of the message, if not found, continue to wait f
or the next packet
            int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
            if (tailIndex < 0) {
                in.resetReaderIndex();
```

```
            return null;
        }
        //Define the location data entity class
        Result model = new Result();
        //packet header(
        in.readByte();
        //Field list
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //Query the subscript of comma to intercept data
            int index = in.bytesBefore(Constant.TEXT_MSG_SPLITER);
            int itemLen = index > 0 ? index : in.readableBytes() - 1;
            byte[] byteArr = new byte[itemLen];
            in.readBytes(byteArr);
            in.readByte();
            itemList.add(new String(byteArr));
        }
        String msgType = "";
        if (itemList.size() >= 5) {
            msgType = itemList.get(3) + itemList.get(4);
        }
        Object dataBody=null;
        if(itemList.size()>0){
            dataBody="(";
            for(String item :itemList) {
                dataBody+=item+",";
            }
            dataBody=CommonUtil.trimEnd(dataBody.toString(),",");
            dataBody += ")";
        }
        String replyMsg="";
        if(msgType.equals("BASE2")&&itemList.get(5).toUpperCase().equals("T
IME")) {
            replyMsg=PacketUtil.replyBASE2Message(itemList);
        }
        model.setDeviceID(itemList.get(0));
        model.setMsgType(msgType);
        model.setDataBody(dataBody);
        model.setReplyMsg(replyMsg);
        return model;
    }

    /**
     * Parse and locate message body
     * @param msgBodyBuf
     * @return
```

```java
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
         //alarm sign
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //Device status
        long status = msgBodyBuf.readUnsignedInt();
        //Latitude
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Longitude
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Altitude, in meters
        int altitude = msgBodyBuf.readShort();
        //Speed
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //directioin
        int direction = msgBodyBuf.readShort();
        //GPS time
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
         //Determine whether the south latitude and west longitude are base
d on the value of the status bit
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //Positioning status
        int locationType=NumberUtil.getBitValue(status, 18);
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 1);
        }
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 6) > 0 ? 2 : 0;
        }

        LocationData locationData=new LocationData();
        locationData.setGpsTime(gpsZonedDateTime.toString());
```

```
        locationData.setLatitude(lat);
        locationData.setLongitude(lon);
        locationData.setLocationType(locationType);
        locationData.setSpeed((int)speed);
        locationData.setDirection(direction);
        //Handling additional information
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }

    /**
     * Parse additional information
     *
     * @param msgBody
     * @param location
     */
    private static void parseExtraInfo(ByteBuf msgBody, LocationData locati
on) {
        ByteBuf extraInfoBuf = null;
        while (msgBody.readableBytes() > 1) {
            int extraInfoId = msgBody.readUnsignedByte();
            int extraInfoLen = msgBody.readUnsignedByte();
            if (msgBody.readableBytes() < extraInfoLen) {
                break;
            }
            extraInfoBuf = msgBody.readSlice(extraInfoLen);
            switch (extraInfoId) {
                case 0x0F:
                    //Parsing temperature data
                    double temperature = -1000.0;
                    temperature = parseTemperature(extraInfoBuf.readShort()
);
                    location.setTemperature((int)temperature);
                    break;
                //Wireless communication network signal strength
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                 //number of satellites
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
```

```
                //battery percentage
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                 //battery voltage
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setBattery(fVoltage*0.01);
                    break;
                case 0xDA:
                    //Number of rope cuts
                    int fTimes =extraInfoBuf.readUnsignedShort();
                    //Status bit
                    int status =extraInfoBuf.readUnsignedByte();
                    //Lock status big
                    int fLockStatus=(status&0b0000_0001)==1?0:1;
                    //motion status
                    int fRunStatus=(status&0b0000_0010)>0?1:0;
                    //Sim card status
                    int fSimStatus=(status&0b0000_0100)>0?1:0;
                    //wake-up source
                    int fWakeSource=((status>>3)&0b0111);
                    location.setLockStatus(fLockStatus);
                    location.setLockRope(fLockStatus);
                    location.setUnLockTime(fTimes);
                    location.setRunStatus(fRunStatus);
                    location.setSimStatus(fSimStatus);
                    location.setAwaken(fWakeSource);
                    break;
                case 0xDB:
                    //Number of positioning data sent
                    int sendCount=extraInfoBuf.readUnsignedShort();
                    location.setSendDataCount(sendCount);
                    break;
                case 0xDC:
                    //Debug
                    byte[] debugArr = new byte[4];
                    extraInfoBuf.readBytes(debugArr);
                    String strDebug=ByteBufUtil.hexDump(debugArr);
                    break;
                case 0xF8:
                    //temperature value
                    int temp=extraInfoBuf.readUnsignedShort();
                    if(temp==0xffff){
                        location.setTemperature(-1000);
```

```
                    }else {
                        temp=(temp / 10)-50;
                        location.setTemperature(temp);
                    }
                    break;
                case 0xF9:
                    //Protocol version
                    int version=extraInfoBuf.readUnsignedShort();
                    break;
                case 0xFD:
                    //LBS info.
                    int mcc=extraInfoBuf.readUnsignedShort();
                    location.setMCC(mcc);
                    int mnc=extraInfoBuf.readUnsignedByte();
                    location.setMNC(mnc);
                    long cellId=extraInfoBuf.readUnsignedInt();
                    location.setCELLID((int)cellId);
                    int lac=extraInfoBuf.readUnsignedShort();
                    location.setLAC(lac);
                    break;
                case 0xFE:
                    long mileage = extraInfoBuf.readUnsignedInt();
                    location.setMileage(mileage);
                    break;
                default:
                    ByteBufUtil.hexDump(extraInfoBuf);
                    break;
            }
        }
    }

    /**
     * Parse temperature
     * @param temperatureInt
     * @return
     */
    private static double parseTemperature(int temperatureInt) {
        if (temperatureInt == 0xFFFF) {
            return -1000;
        }
        double temperature = ((short) (temperatureInt << 4) >> 4) * 0.1;
        if ((temperatureInt >> 12) > 0) {
            temperature = -temperature;
        }
        return temperature;
    }
```

```
}
```

PacketUtil：Used to process preprocessed data and restore method encapsulation class

```
package com.jointech.sdk.jt707.utils;

import com.jointech.sdk.jt707.constants.Constant;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.ByteBufUtil;
import io.netty.util.ReferenceCountUtil;

import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Random;

/**
 *  Parse package preprocessing (do unescaping)
 * @author HyoJung
 */
public class PacketUtil {
    /**
     *  Parse message packets
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //The readable length cannot be less than the base length
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //Prevent illegal code stream attacks, the data is too large to be
abnormal data
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //Look for the end of the message, if not found, continue to wait f
or the next packet
        in.readByte();
```

```
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
         //Create a ByteBuf to store the reversed data
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //The data between the header and the end of the message is escaped
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //The length after the reverse escape cannot be less than the base
length
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
        return frame;
    }

    /**
     * In the message header, message body and check code, 0x7D 0x02 is rev
ersed to 0x7E, and 0x7D 0x01 is reversed to 0x7D
     *
     * @param in
     * @param frame
     * @param bodyLen
     */
    public static void unescape(ByteBuf in, ByteBuf frame, int bodyLen) {
        int i = 0;
        while (i < bodyLen) {
            int b = in.readUnsignedByte();
            if (b == 0x7D) {
                int nextByte = in.readUnsignedByte();
                if (nextByte == 0x01) {
                    frame.writeByte(0x7D);
                } else if (nextByte == 0x02) {
                    frame.writeByte(0x7E);
                } else {
                     //abnormal data
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
```

```
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * In the message header, message body and check code, 0x7E is escaped
as 0x7D 0x02, and 0x7D is escaped as 0x7D 0x01
     *
     * @param out
     * @param bodyBuf
     */
    public static void escape(ByteBuf out, ByteBuf bodyBuf) {
        while (bodyBuf.readableBytes() > 0) {
            int b = bodyBuf.readUnsignedByte();
            if (b == 0x7E) {
                out.writeShort(0x7D02);
            } else if (b == 0x7D) {
                out.writeShort(0x7D01);
            } else {
                out.writeByte(b);
            }
        }
    }

    /**
     * reply content
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //Remove the length of the head and tail
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //message ID
            bodyBuf.writeShort(0x8001);
            //data length
```

```
            bodyBuf.writeShort(0x0005);
            //Device ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //Generate random numbers from 1-65534
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //current message serial number
            bodyBuf.writeShort(index);
            //response message serial number
            bodyBuf.writeShort(msgFlowId);
            //response message ID
            bodyBuf.writeShort(0x0200);
            //response result
            bodyBuf.writeByte(0x00);
            //check code
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //packet header
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //The read pointer is reset to the starting position
            bodyBuf.readerIndex(0);
            //escape
            PacketUtil.escape(replyBuf, bodyBuf);
            //packet end
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }

    /**
     *  Timing command reply
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //set date format
            ZonedDateTime currentDateTime = ZonedDateTime.now(ZoneOffset.UT
C);
            String strBase2Reply = String.format("(%s,%s,%s,%s,%s,%s)", ite
mList.get(0), itemList.get(1)
                    , itemList.get(2), itemList.get(3), itemList.get(4), Da
teTimeFormatter.ofPattern("yyyyMMddHHmmss").format(currentDateTime));
            return strBase2Reply;
```

```
            }catch (Exception e) {
                return "";
            }
        }
    }
```

NumberUtil：digital manipulation tools

```
package com.jointech.sdk.jt707.utils;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

/**
 * digital tools
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     * Coordinate accuracy
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     * Coordinate factor
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * One decimal place precision
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     * Format message ID (convert to 0xXXXX)
     *
     * @param msgId Message ID
     * @return format string
```

```java
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * format short numbers
     *
     * @param num number
     * @return format string
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

   /**
     * Convert 4-digit hexadecimal string
     *
     * @param num digits
     * @return format string
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * Parse the value of type short and get the number of digits whose val
ue is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseShortBits(int number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 16; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Parse the value of type int and get the number of digits whose value
 is 1
     *
```

```java
     * @param number
     * @return
     */
    public static List<Integer> parseIntegerBits(long number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 32; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Get the value of the index-th bit in binary
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

    /**
     * bit list to int
     *
     * @param bits
     * @param len
     * @return
     */
    public static int bitsToInt(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        int result = Integer.parseInt(new String(chars), 2);
        return result;
    }

    /**
```

```
     *  bit list to long
     *
     * @param bits
     * @param len
     * @return
     */
    public static long bitsToLong(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0L;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        long result = Long.parseLong(new String(chars), 2);
        return result;
    }

    /**
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(long longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
doubleValue();
    }

    /**
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(int longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
doubleValue();
    }
}
```

CommonUtil：public method class

```java
package com.jointech.sdk.jt707.utils;

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

/**
 * <p>Description: Used to store some public methods encountered in parsing
</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * remove last character from string
     * @param inStr input string
     * @param suffix characters to remove
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * Hexadecimal to byte[]
     * @param hex
     * @return
     */
    public static byte[] hexStr2Byte(String hex) {
        ByteBuffer bf = ByteBuffer.allocate(hex.length() / 2);
        for (int i = 0; i < hex.length(); i++) {
            String hexStr = hex.charAt(i) + "";
            i++;
            hexStr += hex.charAt(i);
            byte b = (byte) Integer.parseInt(hexStr, 16);
            bf.put(b);
        }
```

```
        return bf.array();
    }

    /**
     * Convert GPS time
     *
     * @param bcdTimeStr
     * @return
     */
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyMMddHH
mmss");
        LocalDateTime localDateTime = LocalDateTime.parse(bcdTimeStr, forma
tter);
        ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, ZoneO
ffset.UTC);
        return zonedDateTime;
    }

    /**
     *XOR evaluation of each byte
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

BaseEnum：base enumeration

```
package com.jointech.sdk.jt707.base;

import java.io.Serializable;

/**
 * base enumeration
 * @author HyoJung
 */
public interface BaseEnum  <T> extends Serializable {
```

```
    T getValue();
}
```

Constant：custom constant

```
package com.jointech.sdk.jt707.constants;

/**
 * constant definition
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * binary message header
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
     * Base length without message body
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;

    /**
     * message length
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * text message header
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * text message tail
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     *text message delimiter
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

LocationData：Location entity class

```java
package com.jointech.sdk.jt707.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * <p>Description: Location entity class</p>
 *
 * @author Lenny
 * @version 1.0.1
 * @date 20210328
 */
@Data
public class LocationData implements Serializable {
    /**
     * Messagebody
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * positioning time
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * Latitude
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * Longitude
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     *  Positioning type
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * Speed
     */
    @JSONField(name = "Speed")
    public int Speed;
```

```java
    /**
     * Direction
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * Mileage
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * GpsSignal
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSMSignal
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * Battery
     */
    @JSONField(name = "Battery")
    public double Battery;
    /**
     *  Voltage
     */
    @JSONField(name = "Voltage")
    public int Voltage;
    /**
     * Device lock status
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * Lock rope status
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * Number of rope cuts
     */
    @JSONField(name = "UnLockTime")
    public int UnLockTime;
    /**
```

```
     * Motion status
     */
    @JSONField(name = "RunStatus")
    public int RunStatus;
    /**
     * Sim card type
     */
    @JSONField(name = "SimStatus")
    public int SimStatus;
    /**
     * Number of positioning data sent
     */
    @JSONField(name = "SendDataCount")
    public int SendDataCount;
    /**
     * MCC
     */
    @JSONField(name = "MCC")
    public int MCC;
    /**
     * MNC
     */
    @JSONField(name = "MNC")
    public int MNC;
    /**
     * LAC
     */
    @JSONField(name = "LAC")
    public int LAC;
    /**
     * CELLID
     */
    @JSONField(name = "CELLID")
    public long CELLID;
    /**
     * Awaken
     */
    @JSONField(name = "Awaken")
    public int Awaken;
    /**
     * Data serial number
     */
    @JSONField(name = "Index")
    public int Index;
    /**
     * Temperature
```

Java

```
     */
    @JSONField(name = "Temperature")
    public int Temperature=-1000;
}
```

Result：result entity class

```
package com.jointech.sdk.jt707.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * result entity class
 * @author HyoJung
 * @date 20210526
 */
@Data
public class Result implements Serializable {
    @JSONField(name = "DeviceID")
    private String DeviceID;
    @JSONField(name = "MsgType")
    private String MsgType;
    @JSONField(name = "DataBody")
    private Object DataBody;
    @JSONField(name = "ReplyMsg")
    private String ReplyMsg;
}
```

2.4.Return message and description

（1）Heartbeat data
Raw data：

```
7E0002000077019120391 5FFF2E47E
```

return message：

```
{"DeviceID":"770191203915","MsgType":"heartbeat"}
```

Return message description

```
{"DeviceID":DeviceID,"MsgType":messagetype ( heartbeat : heartbeat ) }
```

(2) positioning data

Rawdata :

```
7E020000477701912039150000D0000000001041002015 88F7D0206CA3F580000001C0001210
72201060230011F310106D40164D5020050DA03000105DB02000CDC0400000000FD0901CC00
000010922866F80203421B7E
```

Return content :


```
{
    "DeviceID": "770191203915",
    "DataBody": {
        "GpsTime": "2021-07-22T01:06:02Z",
        "Temperature": 33,
        "MNC": 0,
        "UnLockTime": 1,
        "RunStatus": 0,
        "Index": 13,
        "Latitude": 22.581118,
        "Awaken": 0,
        "SimStatus": 1,
        "Direction": 1,
        "Battery": 100,
        "GpsSignal": 6,
        "Voltage": 0.8,
        "Speed": 2,
        "LockStatus": 0,
        "Mileage": 0,
        "MCC": 460,
        "Longitude": 113.917784,
        "LAC": 10342,
        "DataLength": 71,
        "CELLID": 4242,
        "LockRope": 0,
        "LocationType": 1,
        "SendDataCount": 12,
        "GSMSignal": 31
    },
    "ReplyMsg": "7e8001000577019120391 5ee25000d020000ab7e",
    "MsgType": "Location"
}
```

Return message Description

```
{
    "DeviceID":  DeviceID,
    "DataBody": {
        "Index": Data serial number,
        "DataLength": Data length(Bytes),
        "GpsTime": GPStime ( UTC ) ,
        "Latitude": Latitude ( WGS84 ) ,
        "Longitude": Longitude ( WGS84 ) ,
        "Temperature": temperature value , 33 indicates the current temperat
ure is 33°C,
        "UnLockTime": Number of rope cutting and unlocking,
        "RunStatus": Motion status (1: motion; 0: static),
        "Awaken": 0: RTC reporting 1: Rope cutting reporting, 2: Rope inser
tion reporting 3: Cover opening reporting 4: Cover closing reporting 5: Cha
rging/configuring/simulating rope cutting reporting,
        "SimStatus": SIM card type 0: eSIM card; 1: Micro SIM card slot,
        "Direction": True North is 0, clockwise 0-359,
        "Battery": Battery percentage ( 0~100% ) ,
        "GpsSignal": Number of satellites currently received by GPS,
        "Voltage": Battery voltage,unit:Volts,
        "Speed": Speed,unit:KM/H,
        "LockStatus": Device lock status ( 0 : locked ; 1 : unlock ) ,
        "LockRope": Lock rope status ( 0 : inserted ; 1 : pull out ) ,
        "Mileage": Mileage,unit:KM,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "LocationType": Positioning type ( 1 : GPS positioning ; 0 : No position
ing ) ,
        "SendDataCount": Number of sent data,
        "GSMSignal": GSM signal
    },
    "ReplyMsg": Need to reply to the command of the device,
    "MsgType": Data type ( Location : Position data )
}
```

(3) Command data parsing

Rawdata :

```
28373730313931323033393036 2c312c3030312c424153452c362c352c33302c3529
```

Return message :

```
{
    "DeviceID": "770191203906",
    "DataBody": "(770191203906,1,001,BASE,6,5,30,5)",
    "ReplyMsg": "",
    "MsgType": "BASE6"
}
```

Return message description

```
{
    "DeviceID": DeviceID,
    "DataBody": Message Content,
    "ReplyMsg": Reply to the device message (empty means no reply is requir
ed),
    "MsgType": command type
}
```

JT709A&B&C SDK

Java

1.Jar package download

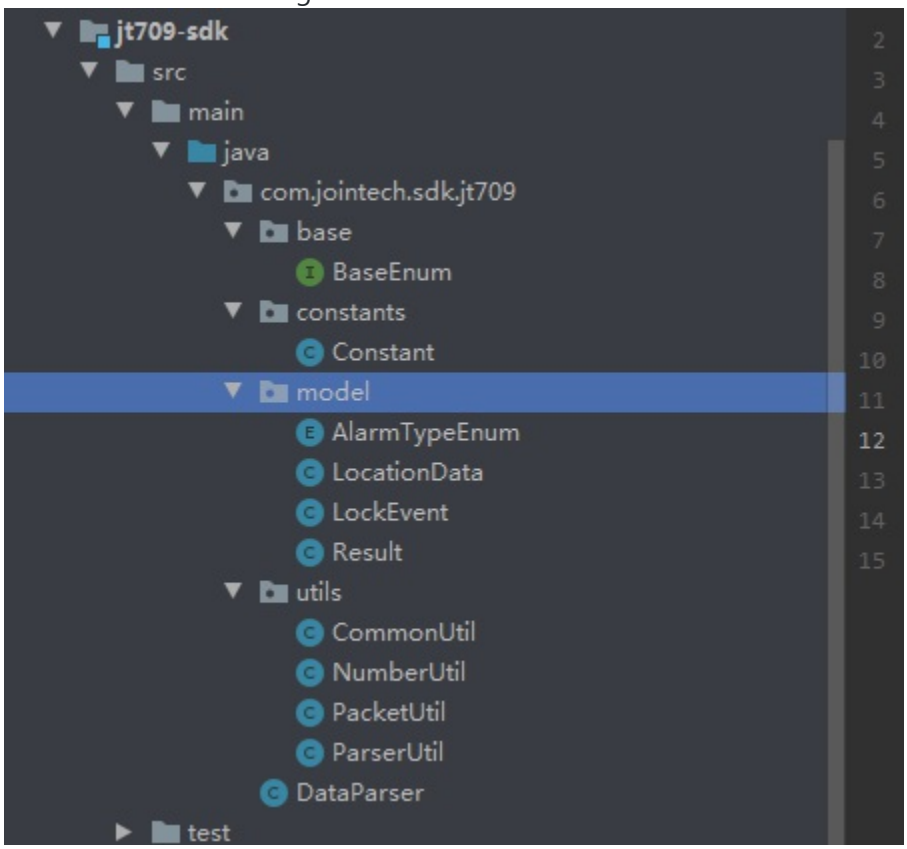
jt709-sdk-1.0.0.jar Download
If you need Jar package development source code, please contact the business application

---

2.Integrated Development Instructions

2.1.Integrated development language and framework description

jt709-sdk-1.0.0.jar is based on the Java language，SpringBoot2.x frame，usenetty，fastjson，lombok，commons-lang3



BaseEnum：base enumeration
Constant：custom constant
AlarmTypeEnum：Alarm enumeration
EventTypeEnum：event enumeration
LocationData：Location entity class
LockEvent：lock event entity class
Result：result entity class

SensorData：Slave data entity class
CommonUtil：public method class
NumberUtil：digital manipulation tools
ParserUtil：Parser method tool class
DataParser：Parser main method

2.2.Integration Instructions

Introduce jt709-sdk-1.0.0.jar into your gateway program as follows：
Introduce the jar package in pom.xml

```
        <dependency>
            <groupId>com.jointech.sdk</groupId>
            <artifactId>jt709-sdk</artifactId>
            <version>1.0.0</version>
        </dependency>
```

Call jt709-sdk-1.0.0.jar，the receiveData() method in the DataParser class
receiveData() method is overloaded

```
package com.jointech.sdk.jt709;

import com.alibaba.fastjson.JSONArray;
import com.jointech.sdk.jt709.constants.Constant;
import com.jointech.sdk.jt709.utils.CommonUtil;
import com.jointech.sdk.jt709.utils.ParserUtil;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.Unpooled;

/**
 * <p>Description: the body of the parse method</p>
 *
 * @author lenny
 * @version 1.0.1
 */
public class DataParser {
    /**
     * Parse Hex string raw data
     * @param strData hexadecimal string
     * @return
     */
    public static Object receiveData(String strData) {
        int length=strData.length()%2>0?strData.length()/2+1:strData.length
()/2;
        ByteBuf msgBodyBuf = Unpooled.buffer(length);
```

```
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     * Parse byte[] raw data
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     *  Parse ByteBuf raw data
     * @param in
     * @return
     */
    private static Object receiveData(ByteBuf in)
    {
        Object decoded = null;
        in.markReaderIndex();
        int header = in.readByte();
        if (header == Constant.TEXT_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeTextMessage(in);
        } else if (header == Constant.BINARY_MSG_HEADER) {
            in.resetReaderIndex();
            decoded = ParserUtil.decodeBinaryMessage(in);
        } else {
            return null;
        }
        return JSONArray.toJSON(decoded).toString();
    }
}
```

2.3.Core code

ParserUtil：Parsing method tool class ParserUtil

```
package com.jointech.sdk.jt709.utils;
```

```java
import com.jointech.sdk.jt709.constants.Constant;
import com.jointech.sdk.jt709.model.AlarmTypeEnum;
import com.jointech.sdk.jt709.model.LocationData;
import com.jointech.sdk.jt709.model.LockEvent;
import com.jointech.sdk.jt709.model.Result;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufUtil;
import io.netty.buffer.Unpooled;
import org.apache.commons.lang3.StringUtils;

import java.time.ZonedDateTime;
import java.util.ArrayList;
import java.util.List;

/**
 * <p>Description: Analysis method tool class</p>
 * @author HyoJung
 */
public class ParserUtil {
    /**
     *Parse Positioning data 0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
         //unexcape rawdata
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //message length
        int msgLen = msg.readableBytes();
        //packet header
        msg.readByte();
        //message ID
        int msgId = msg.readUnsignedShort();
        //message body properties
        int msgBodyAttr = msg.readUnsignedShort();
        //message body length
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //Whether to subcontract
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //Remove the base length of the message body
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //The following packet length is obtained according to the length o
f the message body and whether it is subcontracted
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
```

```
        if (msgLen != ensureLen) {
            return null;
        }
        //array of deviceID
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //Device ID (remove leading 0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //message serial number
        int msgFlowId = msg.readUnsignedShort();
        //total number of message packets
        int packetTotalCount = 0;
        //package serial number
        int packetOrder = 0;
        //subcontract
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //message body
        byte[] msgBodyArr = new byte[msgBodyLen];
        msg.readBytes(msgBodyArr);
        if(msgId==0x0200) {
            //Parse the message body
            LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
            locationData.setIndex(msgFlowId);
            locationData.setDataLength(msgBodyLen);
            //check code
            int checkCode = msg.readUnsignedByte();
            //packet end
            msg.readByte();
            //Get message response content
            String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
            //Define the location data entity class
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("Location");
            model.setDataBody(locationData);
            model.setReplyMsg(replyMsg);
            return model;
        }else {
            //Define the location data entity class
            Result model = new Result();
```

```
            model.setDeviceID(terminalNumber);
            model.setMsgType("heartbeat");
            model.setDataBody(null);
            return model;
        }
    }

    /**
     * Parse instruction data
     * @param in raw data
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

         //The read pointer is set to the message header
        in.markReaderIndex();
        //Look for the end of the message, if not found, continue to wait f
or the next packet
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //Define the location data entity class
        Result model = new Result();
        //packet header(
        in.readByte();
        //Field list
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //Query the subscript of comma to intercept data0) {
            int index = in.bytesBefore(Constant.TEXT_MSG_SPLITER);
            int itemLen = index > 0 ? index : in.readableBytes() - 1;
            byte[] byteArr = new byte[itemLen];
            in.readBytes(byteArr);
            in.readByte();
            itemList.add(new String(byteArr));
        }
        String msgType = "";
        if (itemList.size() >= 5) {
            msgType = itemList.get(3) + itemList.get(4);
        }
        Object dataBody=null;
        if(itemList.size()>0){
            dataBody="(";
            for(String item :itemList) {
```

```
                dataBody+=item+",";
            }
            dataBody=CommonUtil.trimEnd(dataBody.toString(),",");
            dataBody += ")";
        }
        String replyMsg="";
        if(msgType.equals("BASE2")&&itemList.get(5).toUpperCase().equals("T
IME")) {
            replyMsg=PacketUtil.replyBASE2Message(itemList);
        }
        model.setDeviceID(itemList.get(0));
        model.setMsgType(msgType);
        model.setDataBody(dataBody);
        model.setReplyMsg(replyMsg);
        return model;
    }

    /**
     * Parse and locate message body
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
         //alarm sign
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //Device status
        long status = msgBodyBuf.readUnsignedInt();
        //latitude
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //longitude
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //Altitude, in meters
        int altitude = msgBodyBuf.readShort();
        //Speed
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //directioin
        int direction = msgBodyBuf.readShort();
        //GPS time
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
```

```
        //Determine whether the south latitude and west longitude are based
  on the value of the status bit
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
         //Positioning status
        int locationType=NumberUtil.getBitValue(status, 18);
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 1);
        }
        if(locationType==0)
        {
            locationType = NumberUtil.getBitValue(status, 6) > 0 ? 2 : 0;
        }
        //Lock rope status
        int lockRope=NumberUtil.getBitValue(status, 20);
        //Lock Motor status
        int lockMotor=NumberUtil.getBitValue(status, 21);
         //The lock state (judged by the combination of the lock rope + mot
  or;when lock rope pull out (1) or motor unlock(1),lock state is unlocking(1
  );when lock rope inserted(0) and motor locked(0),the lock state is locked
        int lockStatus=0;
        if(lockRope==1||lockMotor==1) {
            lockStatus=1;
        }
        int backCover=NumberUtil.getBitValue(status, 7);
        //wake-up source
        long awaken = (status>>24)&0b00001111;
        int alarm=parseAlarm(alarmFlag);
        LocationData locationData=new LocationData();
        locationData.setGpsTime(gpsZonedDateTime.toString());
        locationData.setLatitude(lat);
        locationData.setLongitude(lon);
        locationData.setLocationType(locationType);
        locationData.setSpeed((int)speed);
        locationData.setDirection(direction);
        locationData.setAltitude(altitude);
        locationData.setLockStatus(lockStatus);
        locationData.setLockRope(lockRope);
        locationData.setAwaken((int)awaken);
        locationData.setBackCover(backCover);
        locationData.setAlarm(alarm);
```

```
        //Handling additional information
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }

    /**
     *  Parse additional information
     *
     * @param msgBody
     * @param location
     */
    private static void parseExtraInfo(ByteBuf msgBody, LocationData locati
on) {
        ByteBuf extraInfoBuf = null;
        while (msgBody.readableBytes() > 1) {
            int extraInfoId = msgBody.readUnsignedByte();
            int extraInfoLen = msgBody.readUnsignedByte();
            if (msgBody.readableBytes() < extraInfoLen) {
                break;
            }
            extraInfoBuf = msgBody.readSlice(extraInfoLen);
            switch (extraInfoId) {
                //Lock event
                case 0x0B:
                    LockEvent event=new LockEvent();
                    //Lock event
                    int type=extraInfoBuf.readUnsignedByte();
                    event.setType(type);
                    if(type==0x01||type==0x02||type==0x03||type==0x05||type
==0x1E||type==0x1F){
                        //Unlock by password
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
                        event.setPassword(password);
                        int unlockStatus=extraInfoBuf.readUnsignedByte();
                        if(unlockStatus==0xff){
                            event.setUnLockStatus(0);
                        }else{
                            if(unlockStatus>0&&unlockStatus<100){
                                event.setFenceId(unlockStatus);
                            }
                            event.setUnLockStatus(1);
                        }
```

```
                    }else if(type==0x06||type==0x07||type==0x08||type==0x10
||type==0x11||type==0x18||type==0x19||type==0x20||type==0x28||type==0x29){
                        //Unlock by password
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
                        event.setPassword(password);
                        event.setUnLockStatus(0);
                    }else if(type==0x22){
                        //RFID card No.
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                        if(extraInfoBuf.readableBytes()>0) {
                            int unlockStatus = extraInfoBuf.readUnsignedByt
e();
                            if (unlockStatus == 0xff) {
                                event.setUnLockStatus(0);
                            } else {
                                if (unlockStatus > 0 && unlockStatus < 100)
 {
                                    event.setFenceId(unlockStatus);
                                }
                                event.setUnLockStatus(1);
                            }
                        }else{
                            event.setUnLockStatus(1);
                        }
                    }else if(type==0x23||type==0x2A||type==0x2B){
                        //RFID card No.
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                    }
                    location.setLockEvent(event);
                    break;
                //Wireless communication network signal strength
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //number of satellites
```

```
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //battery percentage
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //battery voltage
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setVoltage(fVoltage*0.01);
                    break;
                case 0xF9:
                    //Protocol version
                    int version=extraInfoBuf.readUnsignedShort();
                    location.setProtocolVersion(version);
                    break;
                case 0xFD:
                    //LBS information
                    int mcc=extraInfoBuf.readUnsignedShort();
                    location.setMCC(mcc);
                    int mnc=extraInfoBuf.readUnsignedByte();
                    location.setMNC(mnc);
                    long cellId=extraInfoBuf.readUnsignedInt();
                    location.setCELLID((int)cellId);
                    int lac=extraInfoBuf.readUnsignedShort();
                    location.setLAC(lac);
                    break;
                case 0xFC:
                    int fenceId = extraInfoBuf.readUnsignedByte();
                    location.setFenceId(fenceId);
                    break;
                case 0xFE:
                    long mileage = extraInfoBuf.readUnsignedInt();
                    location.setMileage(mileage);
                    break;
                default:
                    ByteBufUtil.hexDump(extraInfoBuf);
                    break;
            }
        }
    }

    /**
```

```
     * Alarm Parsing
     * @param alarmFlag
     * @return
     */
    private static int parseAlarm(long alarmFlag) {
        int alarm=-1;
        //Single trigger alarm
        if(NumberUtil.getBitValue(alarmFlag, 1) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_1.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 7) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_2.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 16) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_3.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 17) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_4.getValue());
        }else if(NumberUtil.getBitValue(alarmFlag, 18) == 1)
        {
            alarm = Integer.parseInt(AlarmTypeEnum.ALARM_5.getValue());
        }
        return alarm;
    }
}
```

PacketUtil：Used to process preprocessed data and restore method encapsulation class

```
package com.jointech.sdk.jt709.utils;

import com.jointech.sdk.jt709.constants.Constant;
import io.netty.buffer.ByteBuf;
import io.netty.buffer.ByteBufAllocator;
import io.netty.buffer.ByteBufUtil;
import io.netty.util.ReferenceCountUtil;

import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Random;

/**
 * Parse package preprocessing (do unescaping)
```

```java
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * Parse message packets
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
         //The readable length cannot be less than the base length
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //Prevent illegal code stream attacks, the data is too large to be abnormal data
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //Look for the end of the message, if not found, continue to wait for the next packet
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
         //Create a ByteBuf to store the reversed data
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //The data between the header and the end of the message is escaped
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //The length after the reverse escape cannot be less than the base length
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
```

```
        return frame;
    }

    /**
     *  In the message header, message body and check code, 0x7D 0x02 is re
versed to 0x7E, and 0x7D 0x01 is reversed to 0x7D
     *
     * @param in
     * @param frame
     * @param bodyLen
     */
    public static void unescape(ByteBuf in, ByteBuf frame, int bodyLen) {
        int i = 0;
        while (i < bodyLen) {
            int b = in.readUnsignedByte();
            if (b == 0x7D) {
                int nextByte = in.readUnsignedByte();
                if (nextByte == 0x01) {
                    frame.writeByte(0x7D);
                } else if (nextByte == 0x02) {
                    frame.writeByte(0x7E);
                } else {
                    //abnormal data
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * In the message header, message body and check code, 0x7E is escaped
as 0x7D 0x02, and 0x7D is escaped as 0x7D 0x01
     *
     * @param out
     * @param bodyBuf
     */
    public static void escape(ByteBuf out, ByteBuf bodyBuf) {
        while (bodyBuf.readableBytes() > 0) {
            int b = bodyBuf.readUnsignedByte();
            if (b == 0x7E) {
                out.writeShort(0x7D02);
```

```
            } else if (b == 0x7D) {
                out.writeShort(0x7D01);
            } else {
                out.writeByte(b);
            }
        }
    }

    /**
     * Reply content
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
       //Remove the length of the head and tail
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //Message ID
            bodyBuf.writeShort(0x8001);
            //Data Length
            bodyBuf.writeShort(0x0005);
            //Device ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //Generate random numbers from 1-65534
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //Current message serial number
            bodyBuf.writeShort(index);
            //Reply message serial number
            bodyBuf.writeShort(msgFlowId);
            //Reply message ID
            bodyBuf.writeShort(0x0200);
            //Response result
            bodyBuf.writeByte(0x00);
            //check code
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //packet header
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //The read pointer is reset to the starting position
            bodyBuf.readerIndex(0);
```

```
            //escape
            PacketUtil.escape(replyBuf, bodyBuf);
            //packet end
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }

    /**
     * Timing command reply
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //set date format
            ZonedDateTime currentDateTime = ZonedDateTime.now(ZoneOffset.UT
C);
            String strBase2Reply = String.format("(%s,%s,%s,%s,%s,%s)", ite
mList.get(0), itemList.get(1)
                    , itemList.get(2), itemList.get(3), itemList.get(4), Da
teTimeFormatter.ofPattern("yyyyMMddHHmmss").format(currentDateTime));
            return strBase2Reply;
        }catch (Exception e) {
            return "";
        }
    }
}
```

NumberUtil：digital manipulation tools

```
package com.jointech.sdk.jt709.utils;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

/**
 * digital tools
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
```

```java
    /**
     *  Coordinate accuracy
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     *  Coordinate factor
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * One decimal place precision
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     *  Format message ID (convert to 0xXXXX)
     *
     * @param msgId Message ID
     * @return return format string
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * format short numbers
     *
     * @param num number
     * @return format string
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
      * Convert 4-digit hexadecimal string
     *
     * @param num digits
     * @return format string
     */
```

```java
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     *  Parse the value of type short and get the number of digits whose va
 lue is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseShortBits(int number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 16; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Parse the value of type int and get the number of digits whose value
  is 1
     *
     * @param number
     * @return
     */
    public static List<Integer> parseIntegerBits(long number) {
        List<Integer> bits = new ArrayList<>();
        for (int i = 0; i < 32; i++) {
            if (getBitValue(number, i) == 1) {
                bits.add(i);
            }
        }
        return bits;
    }

    /**
     * Get the value of the index-th bit in binary
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
```

```
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

    /**
     * bit list to int
     *
     * @param bits
     * @param len
     * @return
     */
    public static int bitsToInt(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        int result = Integer.parseInt(new String(chars), 2);
        return result;
    }

    /**
     * bit list to long
     *
     * @param bits
     * @param len
     * @return
     */
    public static long bitsToLong(List<Integer> bits, int len) {
        if (bits == null || bits.isEmpty()) {
            return 0L;
        }

        char[] chars = new char[len];
        for (int i = 0; i < len; i++) {
            char value = bits.contains(i) ? '1' : '0';
            chars[len - 1 - i] = value;
        }
        long result = Long.parseLong(new String(chars), 2);
        return result;
    }

    /**
```

```
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(long longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
doubleValue();
    }

    /**
     * BigDecimal Multiplication
     *
     * @param longNum
     * @param precision
     * @return
     */
    public static double multiply(int longNum, BigDecimal precision) {
        return new BigDecimal(String.valueOf(longNum)).multiply(precision).
doubleValue();
    }
}
```

CommonUtil：public method class

```
package com.jointech.sdk.jt709.utils;

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;
import java.time.LocalDateTime;
import java.time.ZoneOffset;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

/**
 * <p>Description:Used to store some public methods encountered in parsing<
/p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
```

```java
    /**
     * remove the last character of a string
     * @param inStr Input string
     * @param suffix characters to remove
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     *  Hexadecimal to byte[]
     * @param hex
     * @return
     */
    public static byte[] hexStr2Byte(String hex) {
        ByteBuffer bf = ByteBuffer.allocate(hex.length() / 2);
        for (int i = 0; i < hex.length(); i++) {
            String hexStr = hex.charAt(i) + "";
            i++;
            hexStr += hex.charAt(i);
            byte b = (byte) Integer.parseInt(hexStr, 16);
            bf.put(b);
        }
        return bf.array();
    }

    /**
     * Convert GPS time
     *
     * @param bcdTimeStr
     * @return
     */
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyMMddHH
mmss");
        LocalDateTime localDateTime = LocalDateTime.parse(bcdTimeStr, forma
tter);
        ZonedDateTime zonedDateTime = ZonedDateTime.of(localDateTime, ZoneO
ffset.UTC);
        return zonedDateTime;
    }
```

```
    /**
     * Each byte is XORed
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

BaseEnum：base enumeration

```
package com.jointech.sdk.jt709.base;

import java.io.Serializable;

/**
 * base enumeration
 * @author HyoJung
 */
public interface BaseEnum  <T> extends Serializable {
    T getValue();
}
```

Constant：custom constant

```
package com.jointech.sdk.jt709.constants;

/**
 * constant definition
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
     private Constant(){}
    /**
     * binary message header
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
```

```
     * Base length without message body
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;

    /**
     * message length
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * text message header
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * text message tail
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * text message delimiter
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

AlarmTypeEnum：Device alarm type enumeration

```
package com.jointech.sdk.jt709.model;

import com.jointech.sdk.jt709.base.BaseEnum;
import lombok.Getter;

/**
 * alarm type enumeration
 * @author HyoJung
 */

public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("Speeding alarm", "1"),
    ALARM_2("Low Battery alarm", "2"),
    ALARM_3("Main unit Cover opening alarm", "3"),
    ALARM_4("Enter fence alarm", "4"),
    ALARM_5("Exit fence alarm", "5");
```

```
    @Getter
    private String desc;

    private String value;

    AlarmTypeEnum(String desc, String value) {
        this.desc = desc;
        this.value = value;
    }

    @Override
    public String getValue() {
        return value;
    }

    public static AlarmTypeEnum fromValue(Integer value) {
        String valueStr = String.valueOf(value);
        for (AlarmTypeEnum alarmTypeEnum : values()) {
            if (alarmTypeEnum.getValue().equals(valueStr)) {
                return alarmTypeEnum;
            }
        }
        return null;
    }
}
```

LocationData：Positiondata Entity Classes

```
package com.jointech.sdk.jt709.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * <p>Description: Positiondata Entity Classes</p>
 *
 * @author lenny
 * @version 1.0.1
 */
@Data
public class LocationData implements Serializable {
    /**
     * Message body
```

```
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * Posintioning time
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * Latitude
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * Longitude
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * Positioning type
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * Speed
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * Direction(Header)
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * Mileage
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * Altitude
     */
    @JSONField(name = "Altitude")
    public int Altitude;
    /**
     * GPS signal
     */
```

```java
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM signal quatity
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * Battery level
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * Batttery voltage
     */
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * Lock status
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * Lock rope status
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * Back Cover status
     */
    @JSONField(name = "BackCover")
    public int BackCover;
    /**
     * Protocol version
     */
    @JSONField(name = "ProtocolVersion")
    public int ProtocolVersion;
    /**
     * Fence ID
     */
    @JSONField(name = "FenceId")
    public int FenceId;
    /**
     * MCC
     */
    @JSONField(name = "MCC")
```

```
    public int MCC;
    /**
     * MNC
     */
    @JSONField(name = "MNC")
    public int MNC;
    /**
     * LAC
     */
    @JSONField(name = "LAC")
    public int LAC;
    /**
     * CELLID
     */
    @JSONField(name = "CELLID")
    public long CELLID;
    /**
     * Awaken
     */
    @JSONField(name = "Awaken")
    public int Awaken;
    /**
     * Alarm
     */
    @JSONField(name = "Alarm")
    public int Alarm;
    /**
     * Lock&unlock event
     */
    @JSONField(name = "LockEvent")
    public LockEvent LockEvent;
    /**
     * Data Serial number
     */
    @JSONField(name = "Index")
    public int Index;
}
```

LockEvent：lock event entity class

```
package com.jointech.sdk.jt709.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;
```

```java
/**
 * lock event
 * @author HyoJung
 */
@Data
public class LockEvent {
    /**
     * Event type
     */
    @JSONField(name = "Type")
    public int Type;
     /**
     * Swipe RFID card number
     */
    @JSONField(name = "CardNo")
    public String CardNo;
    /**
     * Unlock password
     */
    @JSONField(name = "Password")
    public String Password;
    /**
     * Unlocking status(1:success；0：failed)
     */
    @JSONField(name = "UnLockStatus")
    public int UnLockStatus=0;
    /**
     * Fence ID related to unlocking
     */
    @JSONField(name = "FenceId")
    public int FenceId=-1;
}
```

Result：result entity class

```java
package com.jointech.sdk.jt709.model;

import com.alibaba.fastjson.annotation.JSONField;
import lombok.Data;

import java.io.Serializable;

/**
 * result entity class
 * @author HyoJung
```

```
 * @date 20210526
 */
@Data
public class Result implements Serializable {
    @JSONField(name = "DeviceID")
    private String DeviceID;
    @JSONField(name = "MsgType")
    private String MsgType;
    @JSONField(name = "DataBody")
    private Object DataBody;
    @JSONField(name = "ReplyMsg")
    private String ReplyMsg;
}
```

2.4.Return messages and instructions

（1）heartbeat data
Rawdata：

```
7E000200007501804283100001267E
```

Return message：

```
{"DeviceID":"750180428310","MsgType":"heartbeat"}
```

Return message description

```
{"DeviceID":DeviceID,"MsgType":Messagetype（heartbeat：heartbeat）}
```

（2）Position data
Rawdata（without lock&unlock event）：

```
7E0200003A790011000094180C0000000000234000201E807C80713B76A00780000000021080
4150651D4014AD502018C30011A310108FE0400000BCCFD0901CC000000308D51D0F27E
```

Return message：

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": "2021-08-04T15:06:51Z",
        "MNC": 0,
```

```
            "FenceId": 0,
            "BackCover": 0,
            "Index": 6156,
            "Latitude": 31.98356,
            "Awaken": 2,
            "ProtocolVersion": 0,
            "Direction": 0,
            "Battery": 74,
            "GpsSignal": 8,
            "Voltage": 3.96,
            "Speed": 0,
            "LockStatus": 1,
            "Mileage": 3020,
            "MCC": 460,
            "Longitude": 118.73265,
            "LAC": 20944,
            "Alarm": -1,
            "DataLength": 58,
            "CELLID": 12429,
            "LockRope": 1,
            "LocationType": 1,
            "Altitude": 120,
            "GSMSignal": 26
        },
        "ReplyMsg": "7e8001000579001100094dae5180c020000517e",
        "MsgType": "Location"
    }
```

Return message description

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": Positioning time ( UTC time ) ,
        "Latitude": latitude ( WGS84 ) ,
        "Longitude": longitude ( WGS84 ) ,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "FenceId": Geo-Fence ID,
        "BackCover": BackCover status ( 1 : Open ; 0 : Close ) ,
        "Index": Data Serial number,
        "Awaken":Wake-up source (1: RTC alarm wake-up, 2: Gsens vibration w
ake-up 3: open back cover wake-up 4: rope-cut wake-up 5: charging wake-up 6
```

```
: swipe card wake-up 7: Lora wake-up 8: VIP number wake-up 9: non-VIP wake-
up 10: Bluetooth wake-up )
        "ProtocolVersion":Protocol version,
        "Direction": True North is 0, clockwise0-360,
        "Battery": Battery percentage ( 0~100%, 255 indicates battery chargi
ng ) ,
        "GpsSignal":Number of satellites currently received by GPS,
        "Voltage": Battery Voltage,unit: V,
        "Speed": Speed, unit:km/h,
        "LockStatus": DeviceLock Status ( 0 : locked ; 1 : unlock ) ,
        "LockRope": Lock rope status ( 0 : Inserted ; 1 : Pull out ) ,
        "Mileage": Mileage,Unit:km,
        "Alarm": Alarm type (1: Overspeed alarm; 2: Low power alarm; 3: Bac
k cover open alarm; 4: Entering the fence alarm; 5: Exiting the fence alarm
),
        "DataLength": Data length ( Bytes ) ,
        "LocationType": Positioning mode (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Altitude": Altitude,unit:km,
        "GSMSignal":GSM signal ,
    },
    "ReplyMsg": The content of the device that needs to be replied to (empt
y means no reply is required)(platform response),
    "MsgType": Data type ( Location : Position data )
}
```

Rawdata (data with lock&lock event) :

```
7E020000477900110000941808000000000224000201E807AA0713B7EC00910000000021080
4150506D4014AD502018C30011E310108FE0400000BCCFD0901CC00000038CB51D0EF014A0B
0801383838383838654B7E
```

Return message :

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": "2021-08-04T15:05:06Z",
        "MNC": 0,
        "FenceId": 0,
        "BackCover": 0,
        "Index": 6152,
        "Latitude": 31.98353,
        "Awaken": 2,
        "ProtocolVersion": 0,
```

```
        "Direction": 0,
        "Battery": 74,
        "GpsSignal": 8,
        "Voltage": 3.96,
        "LockEvent": {
            "Type": 1,
            "FenceId": -1,
            "UnLockStatus": 1,
            "Password": "888888"
        },
        "Speed": 0,
        "LockStatus": 1,
        "Mileage": 3020,
        "MCC": 460,
        "Longitude": 118.73278,
        "LAC": 20944,
        "Alarm": -1,
        "DataLength": 71,
        "CELLID": 14539,
        "LockRope": 0,
        "LocationType": 1,
        "Altitude": 145,
        "GSMSignal": 30
    },
    "ReplyMsg": "7e8001000579001100094a8251808020000e77e",
    "MsgType": "Location"
}
```

Return message description

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": Positioning time ( UTC time ) ,
        "Latitude": latitude ( WGS84 ) ,
        "Longitude": longitude ( WGS84 ) ,
        "MCC": MCC,
        "MNC": MNC,
        "LAC": LAC,
        "CELLID": CELLID,
        "FenceId": Geo-Fence ID,
        "BackCover": BackCover status ( 1 : Open ; 0 : Close ) ,
        "Index": Data Serial number,
        "Awaken":Wake-up source (1: RTC alarm wake-up, 2: Gsens vibration w
ake-up 3: open back cover wake-up 4: rope-cut wake-up 5: charging wake-up 6
```

```
: swipe card wake-up 7: Lora wake-up 8: VIP number wake-up 9: non-VIP wake-
up 10: Bluetooth wake-up )
        "ProtocolVersion":Protocol version,
        "Direction": True North is 0, clockwise0-360,
        "Battery": Battery percentage ( 0~100%, 255 indicates battery chargi
ng ) ,
        "GpsSignal":Number of satellites currently received by GPS,
        "Voltage": Battery Voltage,unit: V,
        "Speed": Speed, unit:km/h,
        "LockStatus": DeviceLock Status ( 0 : locked ; 1 : unlock ) ,
        "LockRope": Lock rope status ( 0 : Inserted ; 1 : Pull out ) ,
        "Mileage": Mileage,Unit:KM,
        "Alarm": Alarm type (1: Overspeed alarm; 2: Low power alarm; 3: Bac
k cover open alarm; 4: Entering the fence alarm; 5: Exiting the fence alarm
),
        "LockEvent": {
            "Type": Event type ( Refer to below Table1 ) ,
            "FenceId": Fence ID associated with the event (-1: no fence rel
ated),
            "UnLockStatus": Unlock status (1: unlock successful; 0: unlock
failed),
            "Password": Unlock password
            "CardNo":If the event type is swipe to unlock, here is the RFID
 card number
        },
        "DataLength": Data length ( Bytes ) ,
        "LocationType": Positioning mode (0: no positioning; 1: GPS positio
ning; 2: base station positioning),
        "Altitude": Altitude,unit:km,
        "GSMSignal":GSM signal ,
    },
    ReplyMsg": The content of the device that needs to be replied to (empty
 means no reply is required)(platform response),
    "MsgType": Data type ( Location : Position data )
}
```

(3) Command data parse

Rawdata :

```
28373931303035303030303032c312c3030312c424153452c312c4a54373039415f3230323
0303932325f48572d56312e305f4e6f524649445f53494d434f4d373630305f56325f312c30
2c4c4531314230335349 4d373630304d31315f412c383938363034313231303138433037333
83535342c3836373538343033343631313131362c3436302c302c313830353534373634 2c31
3033343229
```

Retrun message：

```
{
    "DeviceID": "791005000003",
    "DataBody": "(791005000003,1,001,BASE,1,JT709A_20200922_HW-V1.0_NoRFID_
SIMCOM7600_V2_1,0,LE11B03SIM7600M11_A,898604121018C0738554,867584034611116,
460,0,180554764,10342)",
    "ReplyMsg": "",
    "MsgType": "BASE1"
}
```

Retrun message description

```
{
    "DeviceID":DeviceID,
    "DataBody":Message content,
    "ReplyMsg": Reply to the device's message (empty means no reply is requ
ired)(platform response),
    "MsgType": command type
}
```

Table1

EventID（HEX）	EventID	Event description
0x01	1	Static password remote unlock
0x02	2	Dynamic password remote unlocking
0x03	3	Dynamic password on-site (Bluetooth or WIFI) APP unlock
0x05	5	Indicates static password on-site (Bluetooth or WIFI) APP unlocking
0x06	6	Wrong static password remote unlock
0x07	7	Wrong dynamic password remote unlock
0x08	8	Wrong dynamic password on-site (Bluetooth or WIFI) APP unlock
0x0B	11	Long unlock event
0x0C	12	Lock rope cutting event

0x0D	13	Lock events (automatic locked)
0x10	16	The remote execution of unlocking is abnormal, and the unlocking is not executed without positioning
0x11	17	The remote execution of unlocking is abnormal, and the unlocking will not be executed if it is positioned outside the fence
0x12	18	Abnormal motor
0x18	24	Unlocking is abnormal in Bluetooth execution, and unlocking is not performed without positioning
0x19	25	The Bluetooth unlocking is abnormal, and the unlocking is not performed if it is positioned outside the fence
0x1C	28	Unlock and pull out the lock rope
0x1E	30	SMS static password remote unlock
0x1F	31	SMS dynamic password remote unlock
0x20	32	Wrong SMS dynamic password
0x22	34	Swipe the authorization card to unlock the event
0x23	35	Swipe illegal card unlock event
0x28	40	Wrong static password on-site (Bluetooth or WIFI) APP unlock
0x29	41	Wrong SMS static password to unlock
0x2A	42	RFID performs unlocking abnormally, and does not perform unlocking without positioning
0x2B	43	RFID performs unlocking abnormally, if it is positioned outside the fence, it does not perform unlocking