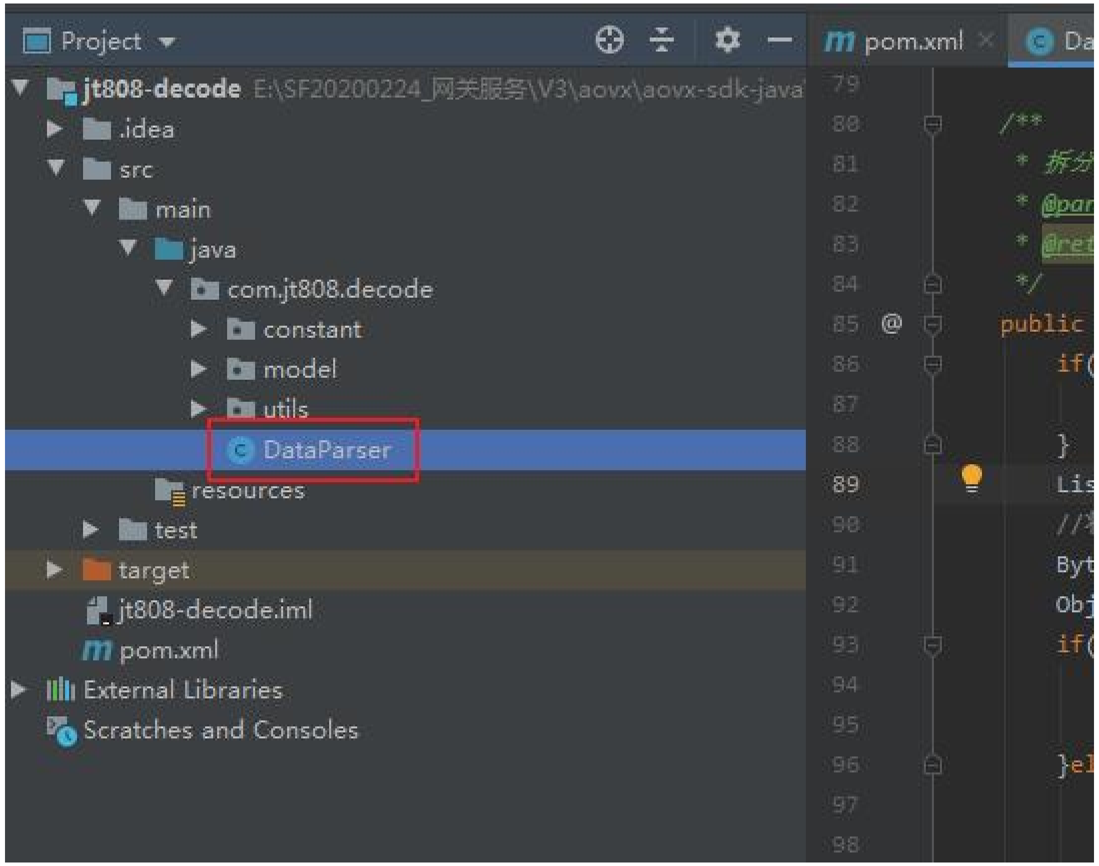
1.Introduction to Development Package

This development is mainly aimed at AOVX products, a toolkit for decoding device uplink data and packaging platform downlink control commands. Adopting the Java SpringBoot2. x framework.
The development mainly achieved the unpacking of device reported data, converting the binary stream data reported by the device into well-known JSON strings.

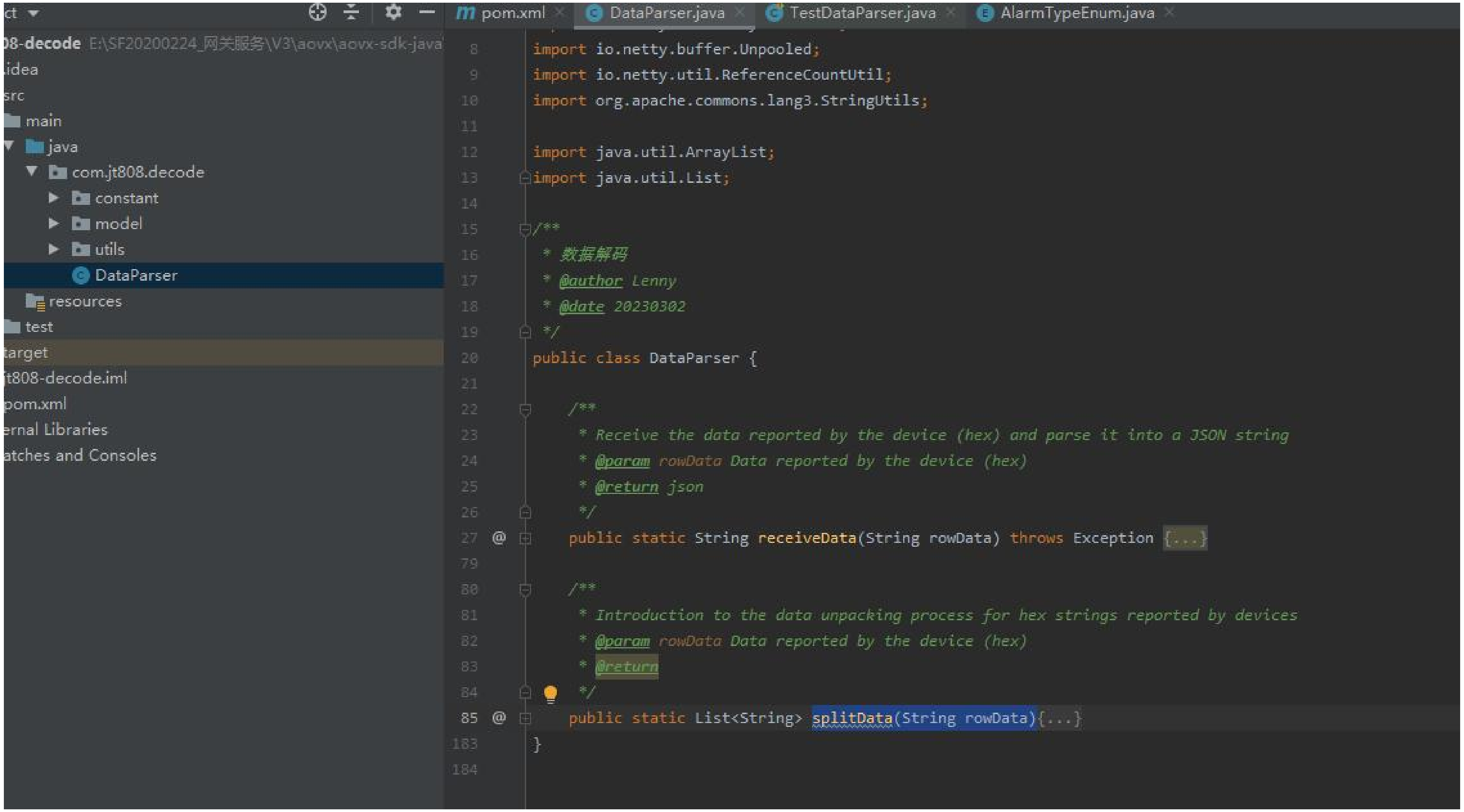
2.Introduction to Source Code

The project name is: jt808 decode, and the method entry class is DataParse.



This class includes two methods:
(1) receiveData(String rowData): Receives the raw data (hexadecimal) reported by the device and converts the data into a JSON string.

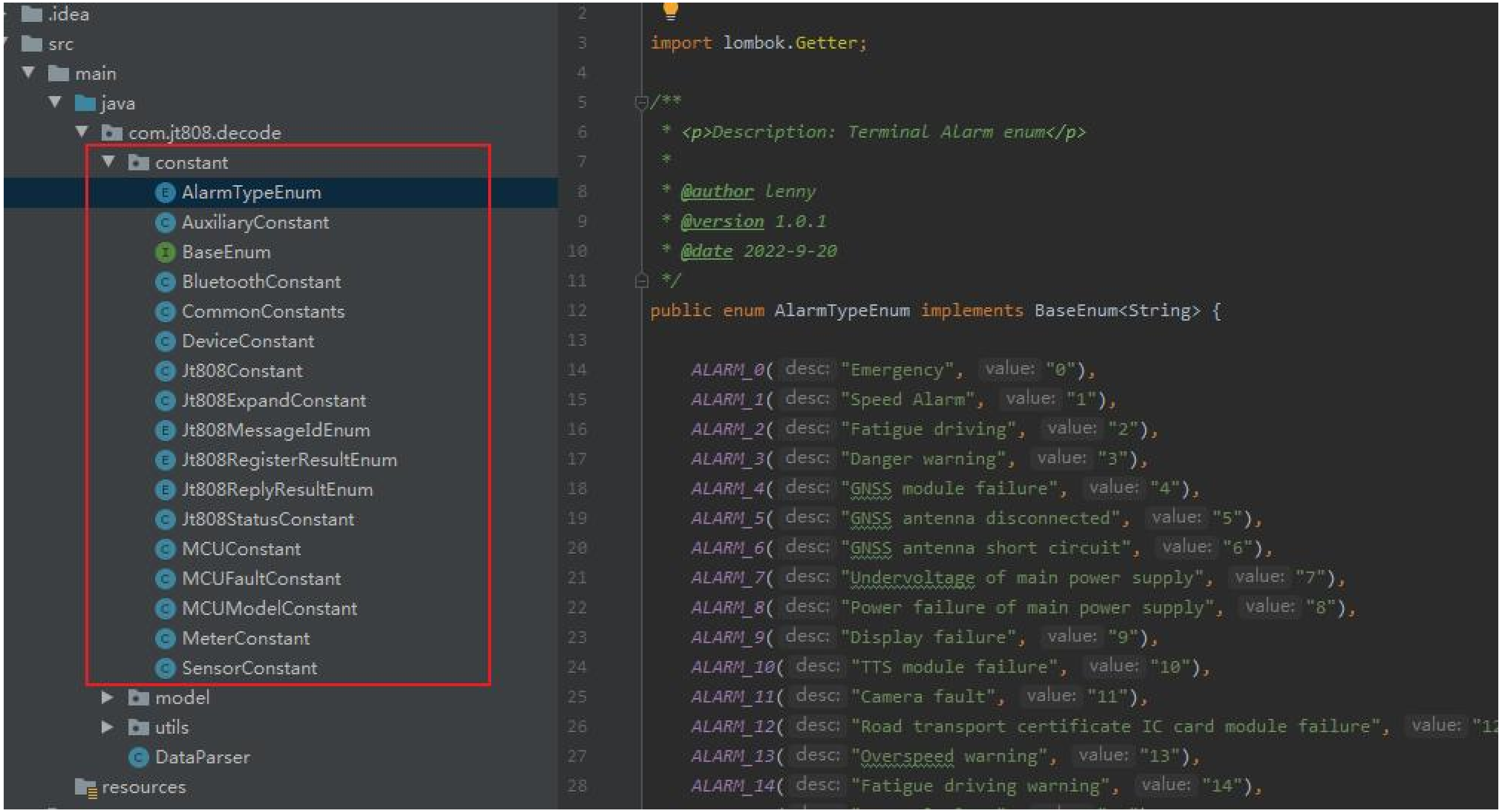
(2) splitData(String rowData): To unpack and explain the raw data reported by the device, and return an introduction to unpacking and segmentation.



```java
import io.netty.buffer.Unpooled;
import io.netty.util.ReferenceCountUtil;
import org.apache.commons.lang3.StringUtils;

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

/**
 * 数据解码
 * @author Lenny
 * @date 20230302
 */
public class DataParser {

    /**
     * Receive the data reported by the device (hex) and parse it into a JSON string
     * @param rowData Data reported by the device (hex)
     * @return json
     */
    public static String receiveData(String rowData) throws Exception {...}

    /**
     * Introduction to the data unpacking process for hex strings reported by devices
     * @param rowData Data reported by the device (hex)
     * @return
     */
    public static List<String> splitData(String rowData){...}
}
```

The project includes three package files:

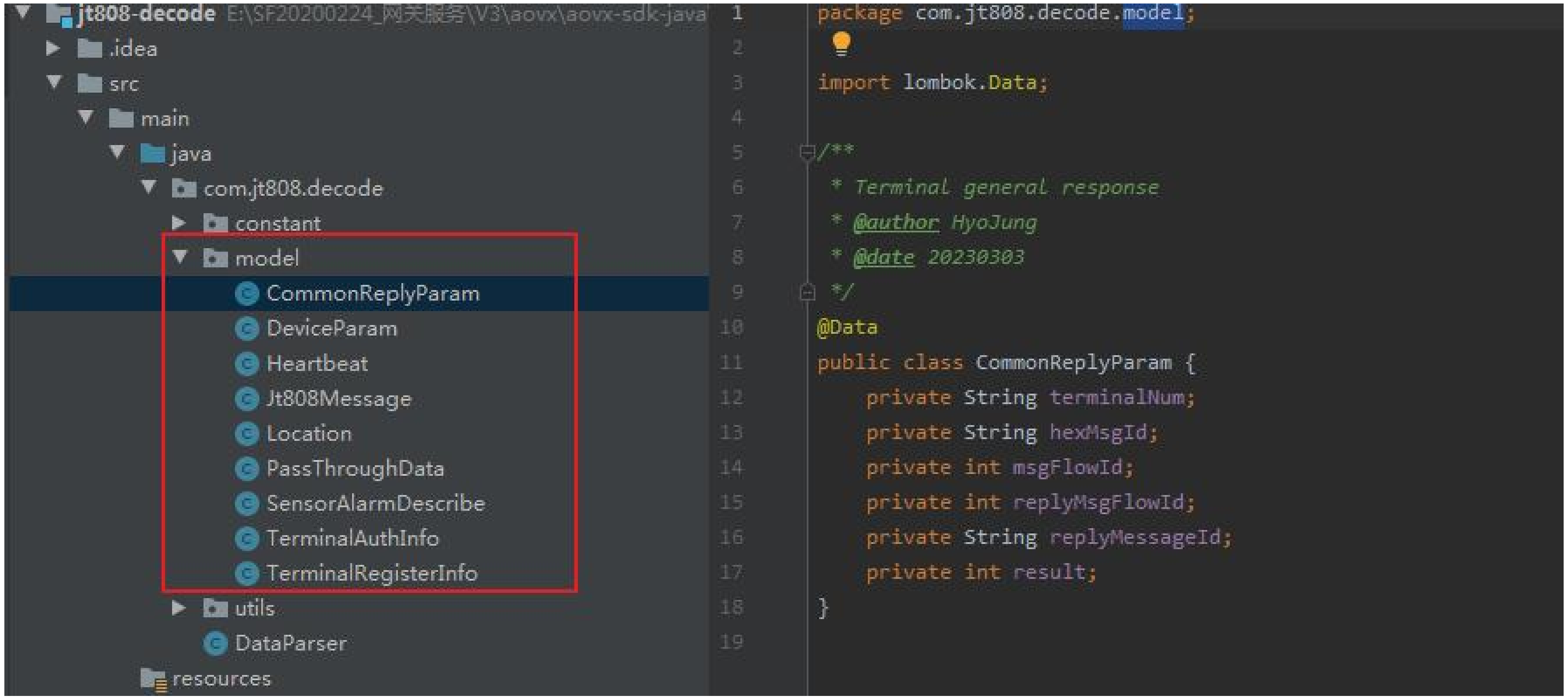
(1) constant: defines some constants or enumeration definitions used in the process of parsing device raw data.



```
.idea
src
  main
    java
      com.jt808.decode
        constant
          AlarmTypeEnum
          AuxiliaryConstant
          BaseEnum
          BluetoothConstant
          CommonConstants
          DeviceConstant
          Jt808Constant
          Jt808ExpandConstant
          Jt808MessageIdEnum
          Jt808RegisterResultEnum
          Jt808ReplyResultEnum
          Jt808StatusConstant
          MCUConstant
          MCUFaultConstant
          MCUModelConstant
          MeterConstant
          SensorConstant
        model
        utils
        DataParser
    resources
```

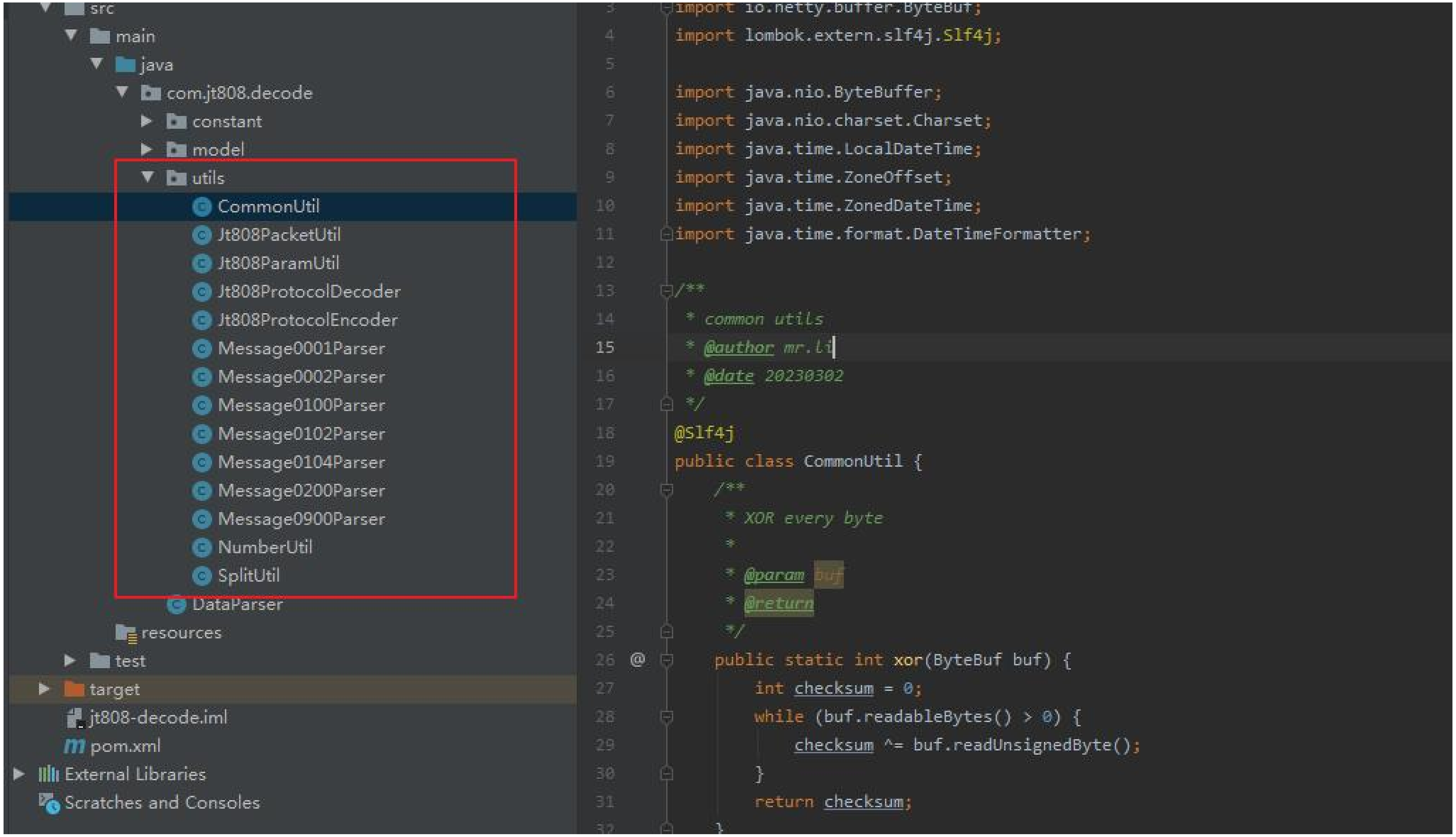
```java

import lombok.Getter;

/**
 * <p>Description: Terminal Alarm enum</p>
 *
 * @author Lenny
 * @version 1.0.1
 * @date 2022-9-20
 */
public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_0( desc: "Emergency",  value: "0"),
    ALARM_1( desc: "Speed Alarm",  value: "1"),
    ALARM_2( desc: "Fatigue driving",  value: "2"),
    ALARM_3( desc: "Danger warning",  value: "3"),
    ALARM_4( desc: "GNSS module failure",  value: "4"),
    ALARM_5( desc: "GNSS antenna disconnected",  value: "5"),
    ALARM_6( desc: "GNSS antenna short circuit",  value: "6"),
    ALARM_7( desc: "Undervoltage of main power supply",  value: "7"),
    ALARM_8( desc: "Power failure of main power supply",  value: "8"),
    ALARM_9( desc: "Display failure",  value: "9"),
    ALARM_10( desc: "TTS module failure",  value: "10"),
    ALARM_11( desc: "Camera fault",  value: "11"),
    ALARM_12( desc: "Road transport certificate IC card module failure",  value: "12
    ALARM_13( desc: "Overspeed warning",  value: "13"),
    ALARM_14( desc: "Fatigue driving warning",  value: "14"),
```

(2) model: defines some entity classes during the decoding process.



```java
package com.jt808.decode.model;

import lombok.Data;

/**
 * Terminal general response
 * @author HyoJung
 * @date 20230303
 */
@Data
public class CommonReplyParam {
    private String terminalNum;
    private String hexMsgId;
    private int msgFlowId;
    private int replyMsgFlowId;
    private String replyMessageId;
    private int result;
}
```

(3) utils: method toolkit, which contains some method classes used during the unpacking process.



```
▼ src
  ▼ main
    ▼ java
      ▼ com.jt808.decode
        ▶ constant
        ▶ model
        ▼ utils
            CommonUtil
            Jt808PacketUtil
            Jt808ParamUtil
            Jt808ProtocolDecoder
            Jt808ProtocolEncoder
            Message0001Parser
            Message0002Parser
            Message0100Parser
            Message0102Parser
            Message0104Parser
            Message0200Parser
            Message0900Parser
            NumberUtil
            SplitUtil
          DataParser
      resources
  ▶ test
▶ target
  jt808-decode.iml
  pom.xml
▶ External Libraries
  Scratches and Consoles
```

```java
import io.netty.buffer.ByteBuf;
import lombok.extern.slf4j.Slf4j;

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

/**
 * common utils
 * @author mr.li
 * @date 20230302
 */
@Slf4j
public class CommonUtil {
    /**
     * XOR every byte
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
```

2.1.receiveData(String rowData)

Call different parsing methods based on different message IDs for parsing:

```
/**
     * Receive the data reported by the device (hex) and parse it into a JSON string
     * @param rowData Data reported by the device (hex)
     * @return json
     */
    public static String receiveData(String rowData) throws Exception {
        if(StringUtils.isBlank(rowData)){
            return null;
        }
        //Convert Hex string to ByteBuf
        ByteBuf byteBuf = Unpooled.wrappedBuffer(ByteBufUtil.decodeHexDump(rowData));
        Object obj= Jt808PacketUtil.decodeJt808Packet(byteBuf);
        String resultJson="";
        if(obj==null){
            ReferenceCountUtil.release(byteBuf);
            return null;
        }else{
            Jt808Message jt808Msg= Jt808ProtocolDecoder.decode((ByteBuf)obj);
            switch (jt808Msg.getMsgId()){
                case 0x0001:
```

```
        CommonReplyParam commonReplyParam= Message0001Parser.parse(jt808Msg,jt808Msg.getMsgBody());

        resultJson=JSON.toJSONString(commonReplyParam);

        break;

    case 0x0002:

        Heartbeat heartbeat= Message0002Parser.parse(jt808Msg);

        heartbeat.setReplyMsg(Jt808PacketUtil.reply8001(jt808Msg));

        resultJson=JSON.toJSONString(heartbeat);

        break;

    case 0x0100:

        TerminalRegisterInfo registerInfo=Message0100Parser.parse(jt808Msg,jt808Msg.getMsgBody());

        registerInfo.setReplyMsg(Jt808PacketUtil.reply8100(jt808Msg,registerInfo.getAuthCode()));

        resultJson=JSON.toJSONString(registerInfo);

        break;

    case 0x0102:

        TerminalAuthInfo authInfo=Message0102Parser.parse(jt808Msg,jt808Msg.getMsgBody());

        authInfo.setReplyMsg(Jt808PacketUtil.reply8001(jt808Msg));

        resultJson=JSON.toJSONString(authInfo);

        break;

    case 0x0104:

        DeviceParam deviceParam=Message0104Parser.parse(jt808Msg,jt808Msg.getMsgBody());
```

```
                resultJson=JSON.toJSONString(deviceParam);

                break;

            case 0x0200:

                Location location= Message0200Parser.parse(jt808Msg,jt808Msg.getMsgBody());

                location.setReplyMsg(Jt808PacketUtil.reply8001(jt808Msg));

                resultJson=JSON.toJSONString(location);

                break;

            case 0x0900:

                PassThroughData throughData=Message0900Parser.parser(jt808Msg,jt808Msg.getMsgBody());

                resultJson=JSON.toJSONString(throughData);

                break;

            default:

                break;

        }

    }

    return resultJson;

}
```

2.2.splitData(String rowData)

Explanation of splitting data according to protocol content format.

```java
/**
     * Introduction to the data unpacking process for hex strings reported by devices
     * @param rowData Data reported by the device (hex)
     * @return
     */
    public static List<String> splitData(String rowData){
        if(StringUtils.isBlank(rowData)){
            return null;
        }
        List<String> dataArr=new ArrayList<>();
        //Convert Hex string to ByteBuf
        ByteBuf byteBuf = Unpooled.wrappedBuffer(ByteBufUtil.decodeHexDump(rowData));
        Object obj= Jt808PacketUtil.decodeJt808Packet(byteBuf);
        if(obj==null){
            ReferenceCountUtil.release(byteBuf);
            return null;
        }else{
            ByteBuf msgBuf= (ByteBuf) obj;
            dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedByte(),2),"Start Flag"));
            int msgId=msgBuf.readUnsignedShort();
```

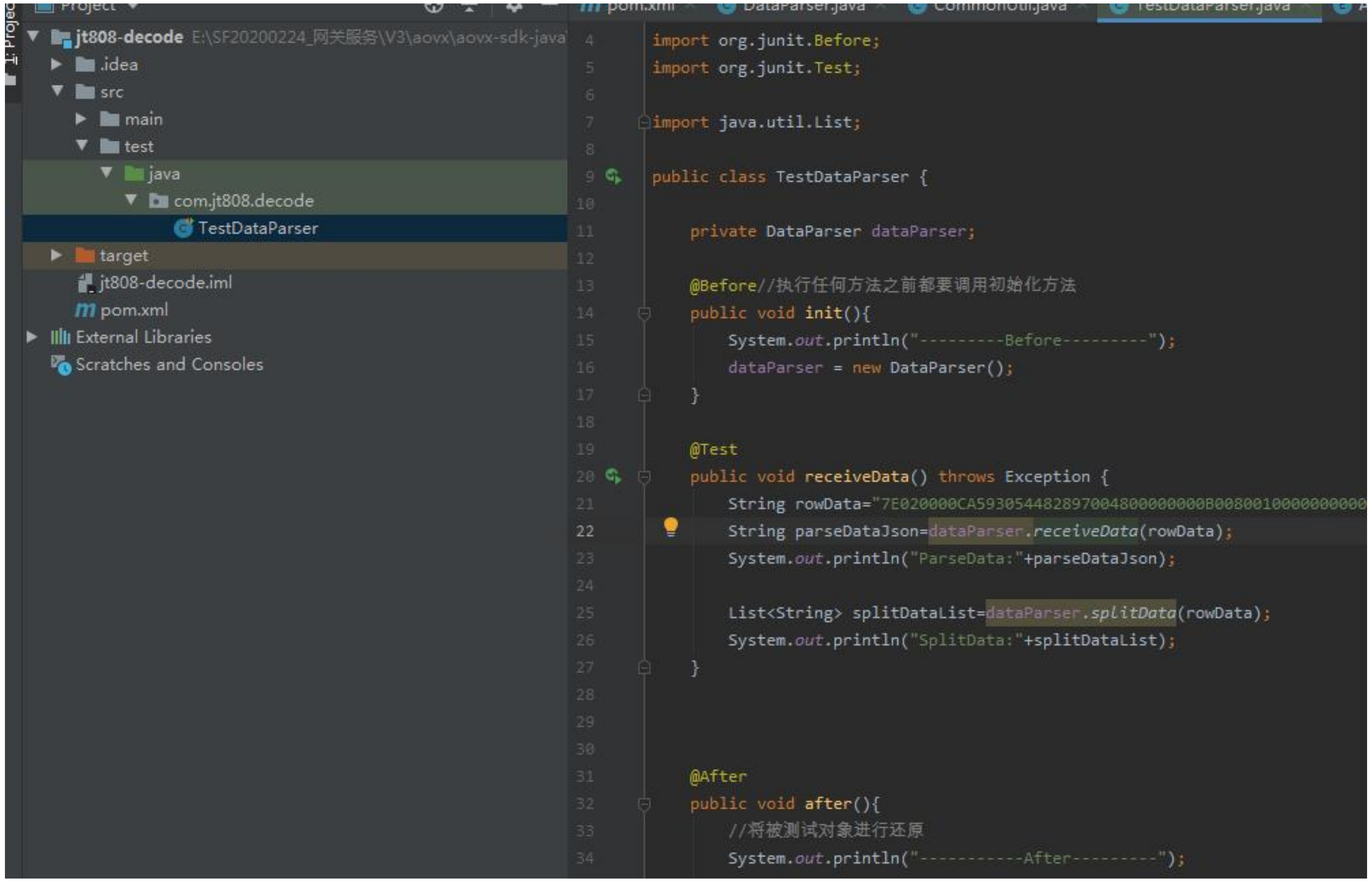
```
dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgId,4),"Message ID"));

//message body properties

short msgBodyAttr = msgBuf.readShort();

//Version ID (version ID 0 refers to the version in 2011 and 1 refers to the version in 2019)

int versionFlag = (msgBodyAttr & 0b01000000_00000000)>0?1:0;

//is multi packet?

boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;

dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBodyAttr,4),"Properties of Message Body"));

//Terminal phone number array,JT808-2019 is 10 bytes

byte[] phoneNumberArr;

if (versionFlag == 1) {

    dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedByte(),2),"Protocol Version"));

    phoneNumberArr = new byte[10];

} else {

    phoneNumberArr = new byte[6];

}

msgBuf.readBytes(phoneNumberArr);

dataArr.add(String.format("%s-->%s",ByteBufUtil.hexDump(phoneNumberArr),"Device Number"));

//Message serial number

int msgFlowId = msgBuf.readUnsignedShort();
```

```
        dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgFlowId,4),"Message serial number"));
        //multi packet?
        if (multiPacket) {
            dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedShort(),4),"Packet Total Count"));
            dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedShort(),4),"Packet Order"));
        }
        //message body length
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        if(msgBodyLen>msgBuf.readableBytes()-2){
            byte[] msgBodyArr=new byte[msgBuf.readableBytes()-2];
            msgBuf.readBytes(msgBodyArr);
            dataArr.add(String.format("%s-->%s",ByteBufUtil.hexDump(msgBodyArr),"Insufficient message body length!"));
        }else{
            ByteBuf msgBodyBuf =msgBuf.readSlice(msgBuf.readableBytes()-2);
            switch (msgId){
                case 0x0001:
                    dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBodyBuf.readShort(),4),"Response serial number"));
                    dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBodyBuf.readShort(),4),"Response message ID"));
                    dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBodyBuf.readShort(),2),"Results (0: success; 1: failu
re; 2: message error; 3: not supported)"));
```

```
    break;
case 0x0002:
    if(msgBodyBuf.readableBytes()>0){
        byte[] msgBodyArr=new byte[msgBodyBuf.readableBytes()];
        msgBodyBuf.readBytes(msgBodyArr);
        dataArr.add(String.format("%s-->%s",ByteBufUtil.hexDump(msgBodyArr),"Message Body"));
    }
    break;
case 0x0100:
    if(versionFlag == 1){
        SplitUtil.splitTerminalRegisterInfo(msgBodyBuf,dataArr);
    }else{
        SplitUtil.splitTerminalRegisterInfo2019(msgBodyBuf,dataArr);
    }
    break;
case 0x0102:
    SplitUtil.splitAuthInfo(msgBodyBuf,dataArr,versionFlag);
    break;
case 0x0104:
    SplitUtil.splitTerminalParameterResponse(msgBodyBuf,dataArr);
```

```
                    break;
                case 0x0200:
                    SplitUtil.splitLocationInfo(msgBodyBuf,dataArr);
                    break;
                case 0x0900:
                    dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBodyBuf.readUnsignedByte(),2),"Transparent message ty
pe"));
                    if(msgBodyBuf.readableBytes()>0){
                        byte [] msgContentArr=new byte[msgBodyBuf.readableBytes()];
                        msgBodyBuf.readBytes(msgContentArr);
                        dataArr.add(String.format("%s-->%s",ByteBufUtil.hexDump(msgContentArr),"Transparent message content"));
                    }
                    break;
                default:
                    byte[] msgBodyArr=new byte[msgBodyBuf.readableBytes()];
                    msgBodyBuf.readBytes(msgBodyArr);
                    dataArr.add(String.format("%s-->%s",ByteBufUtil.hexDump(msgBodyArr),"Message Body"));
                    break;
            }
        }
        dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedByte(),2),"Check Code"));

        dataArr.add(String.format("%s-->%s",NumberUtil.hexStr(msgBuf.readUnsignedByte(),2),"End Flag"));

    }

    return dataArr;

}
```

3.test



```
import org.junit.Before;
import org.junit.Test;

import java.util.List;

public class TestDataParser {

    private DataParser dataParser;

    @Before//执行任何方法之前都要调用初始化方法
    public void init(){
        System.out.println("---------Before---------");
        dataParser = new DataParser();
    }

    @Test
    public void receiveData() throws Exception {
        String rowData="7E020000CA593054482897004800000000B00800100000000000
        String parseDataJson=dataParser.receiveData(rowData);
        System.out.println("ParseData:"+parseDataJson);

        List<String> splitDataList=dataParser.splitData(rowData);
        System.out.println("SplitData:"+splitDataList);
    }



    @After
    public void after(){
        //将被测试对象进行还原
        System.out.println("-----------After---------");
```

3.1.Decoding method effect

Use Example：

```
    @Test

    public void receiveData() throws Exception {

        String rowData="7E020000CA5930544828970048000000000B00800100000000000000000000000000000000023030110180430019E310100F00E0136019A04C
14B10000083039F03F22C414F56585F414D3330302D474C5F48322E305F424739354D334C415230324130345F56322E302E343A763135F423542160033C41C354216
000E04DC33CB74B797D02B8C0F4C114700274BD3EB74B797D02BEBBF60A0009000000000000000000F70400000DE3F81D0408645930544828978932042000001221867
1414D3330302D474C0000F912000F000000000000000000000000000000008C7E";

        String parseDataJson=dataParser.receiveData(rowData);

        System.out.println("ParseData:"+parseDataJson);

    }
```

Output Content：

```
{

    "acc": 0,

    "alarmTypeList": [],

    "altitude": 0,

    "battery": 0,

    "direction": 0,

    "expandMap": {

        "wifi": "[{\"rssi\":-61,\"mac\":\"54:21:60:03:3c:41\"},{\"rssi\":-61,\"mac\":\"54:21:60:00:e0:4d\"},{\"rssi\":-64,\"mac\":\"3
c:b7:4b:79:7e:b8\"},{\"rssi\":-67,\"mac\":\"f4:c1:14:70:02:74\"},{\"rssi\":-69,\"mac\":\"3e:b7:4b:79:7e:be\"}]",
        "auxiliary": "{\"gnss_time\":\"000000000000\",\"acc_duration\":0,\"position_age\":0,\"hdop\":0}",

        "sensor": "[{\"light\":0,\"accelerometer\":\"x:0,y:0,z:0\",\"data_type\":0}]",

        "software_version": "\"AOVX_AM300-GL_H2.0_BG95M3LAR02A04_V2.0.4:v15\"",

        "device": "{\"iccid\":\"89320420000012218671\",\"imei\":\"0864593054482897\",\"device_type\":\"AM300-GL\\u0000\\u0000\",\"wor
k_model\":4}"

    },

    "gnssTime": "2023-03-01T10:18:04Z",

    "gnssValue": 0,

    "gsmValue": 158,

    "hexMsgId": "0x0200",

    "lat": 0.0,

    "lbsCells": "310,410,33539,79776528,159",

    "locationType": 0,

    "lon": 0.0,

    "mileage": 0.0,

    "msgFlowId": 72,

    "recvTime": "2023-04-27T08:34:11.902Z",

    "replyMsg": "7e80010005593054482897004800480200004c7e",

    "speed": 0.0,

    "statusMap": {
        "operate": 1,

        "load": 0,

        "realtime_data": 1,

        "oil_electric": 0,

        "normal_data": 0,

        "door_lock": 0,

        "oil_circuit": 0,

        "confidential": 0

    },

    "terminalNum": "593054482897",

    "voltage": 3.555

}
```

3.2.Effect of unpacking method

Use Example：

```
    @Test

    public void splitData() throws Exception {

        String rowData="7E020000CA593054482897004800000000B0080010000000000000000000000000000023030110180430019E310100F00E0136019A04C
14B10000083039F03F22C414F56585F414D3330302D474C5F48322E305F424739354D334C415230324130345F56322E302E343A763135F423542160033C41C354216
000E04DC33CB74B797D02B8C0F4C114700274BD3EB74B797D02BEBBF60A0009000000000000000000F70400000DE3F81D0408645930544828978932042000001221867
1414D3330302D474C0000F912000F0000000000000000000000000000000008C7E";
```

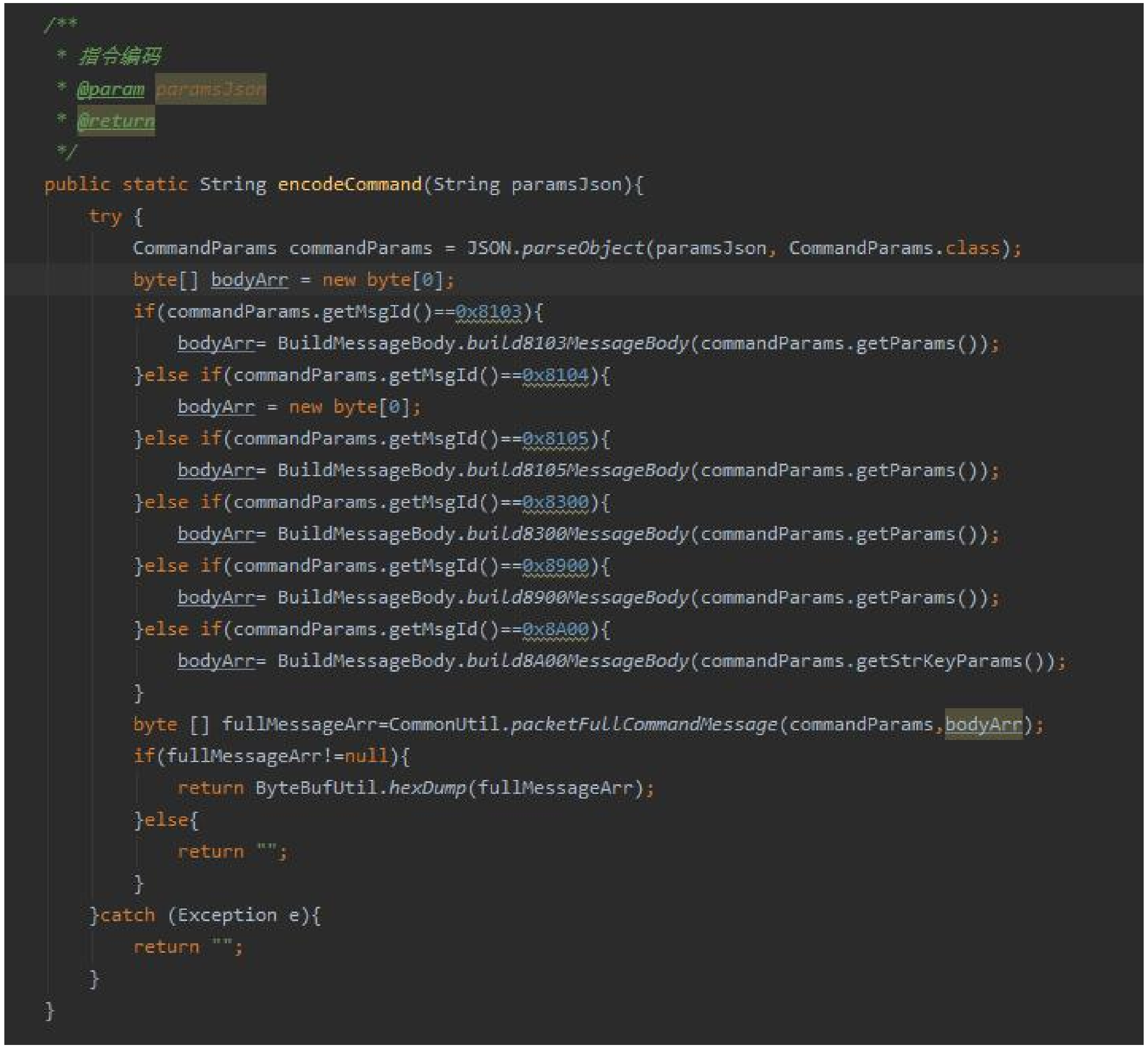
```
        List<String> splitDataList=dataParser.splitData(rowData);

        System.out.println("SplitData:"+splitDataList);

    }
```

Output Content:

```
[7E- - > Start Flag, 0200-- > Message ID, 00 CA-- > Properties of Message Body, 593054482897-- > Device Number, 0048-- > Message seria
l number, 00000000-- > Alarm flag,

B0080010-- > Terminal status, 00000000-- > Latitude, 00000000-- > Longitude, 0000-- > Altitude, 2303-- > Speed, 1101804-- > Direction,
 30019e310100-- > Datetime,

F0-- > Extension ID, 0E- - > Extension information length, 0136019a04c14b10000083039f03-- > Extension information,

F2-- > Extension ID, 2C-- > Extension information length, 414f56585f414d3330302d474c5f48322e305f424739354d334c415230324130345f56322e3
02e343a763135-- > Extension information,

F4-- > Extension ID, 23-- > Extension information length, 542160033c41c354216000e04dc33cb74b797eb8c0f4c114700274bd3eb74b797ebebb-- >
Extension information,

F6-- > Extension ID, 0A-- > Extension information length, 00090000000000000000-- > Extension information,

F7-- > Extension ID, 04-- > Extension information length, 00000de3-- > Extension information,

F8-- > Extension ID, 1D-- > Extension information length, 0408645930544828978932042000012218671414d3330302d474c0000-- > Extension in
formation,

F9-- > Extension ID, 12-- > Extension information length, 000f000000000000000000000000000000-- > Extension information, 8 C-- > Che
ck Code, 7E- - > End Flag

]
```

4.Protocol packaging

4.1.Packaging method encodeCommand(String paramsJson)



```
/**
 * 指令编码
 * @param paramsJson
 * @return
 */
public static String encodeCommand(String paramsJson){
    try {
        CommandParams commandParams = JSON.parseObject(paramsJson, CommandParams.class);
        byte[] bodyArr = new byte[0];
        if(commandParams.getMsgId()==0x8103){
            bodyArr= BuildMessageBody.build8103MessageBody(commandParams.getParams());
        }else if(commandParams.getMsgId()==0x8104){
            bodyArr = new byte[0];
        }else if(commandParams.getMsgId()==0x8105){
            bodyArr= BuildMessageBody.build8105MessageBody(commandParams.getParams());
        }else if(commandParams.getMsgId()==0x8300){
            bodyArr= BuildMessageBody.build8300MessageBody(commandParams.getParams());
        }else if(commandParams.getMsgId()==0x8900){
            bodyArr= BuildMessageBody.build8900MessageBody(commandParams.getParams());
        }else if(commandParams.getMsgId()==0x8A00){
            bodyArr= BuildMessageBody.build8A00MessageBody(commandParams.getStrKeyParams());
        }
        byte [] fullMessageArr=CommonUtil.packetFullCommandMessage(commandParams,bodyArr);
        if(fullMessageArr!=null){
            return ByteBufUtil.hexDump(fullMessageArr);
        }else{
            return "";
        }
    }catch (Exception e){
        return "";
    }
}
```

Parameter input json string

4.2.Parameter JSON description

```
{"msgFlowId":1,"msgId":33027,"params":{1:10,61488:"AT+QUERY?"},"terminalNum":"12345678901"}
```

Field Name	Field Type	Field Description
terminalNum	String	Device S/N
msgFlowId	int	serial number（1~65535）
msgId	int	Protocol defined message ID，such as：0x8103,0x8104,0x8105
params	Map	key:value;wherein,key:Command ID,value:Configured values

0x8103 (setting device parameters)

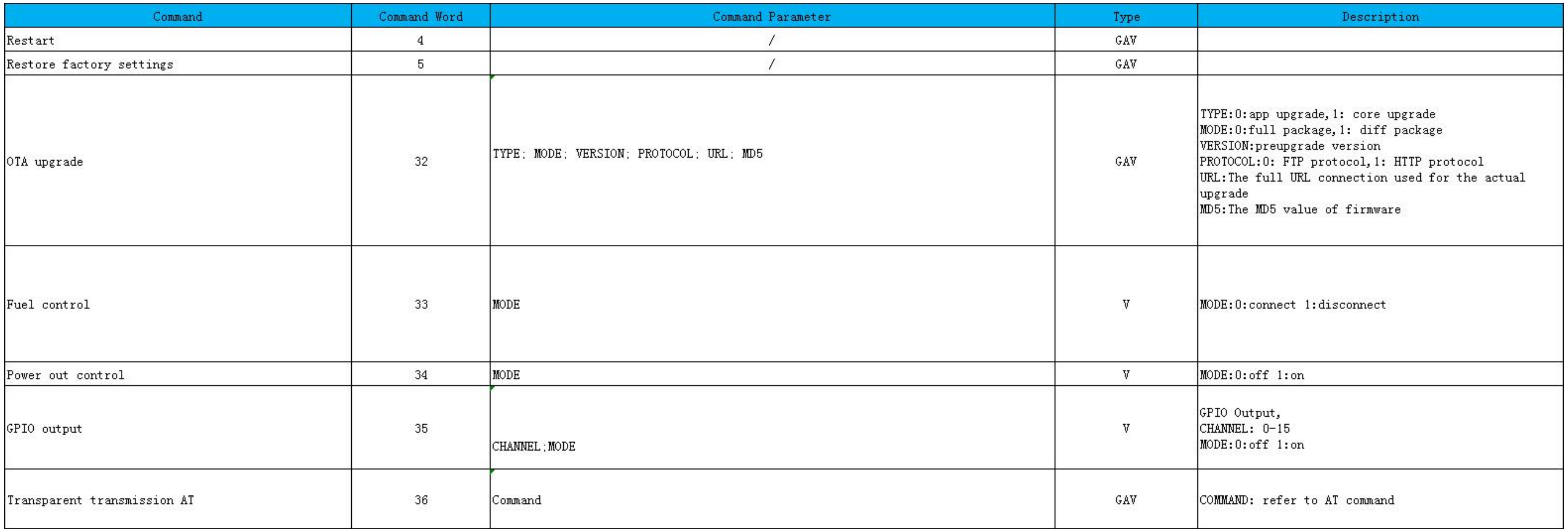
对应的 params，可以设置的命令 ID 以及传入的值说明。

Parameters	Parameter ID	Parameter Type	Parameter Length	Unit	Type	
Device heartbeat interval	0x0001	DWORD	4	second	GAV	
Server APN	0x0010	STRING	/	/	GAV	
APN username	0x0011	STRING	/	/	GAV	
APN password	0x0012	STRING	/	/	GAV	
Main server address	0x0013	STRING	/	/	GAV	Support server IP or domain name
Backup server address	0x0017	STRING	/	/	GAV	Support server IP or domain name
Main server port	0x0018	DWORD	4	/	GAV	TCP or UDP port
Report interval in sleep mode	0x0027	DWORD	4	s	V	
Report interval in run mode	0x0029	DWORD	4	s	V	
Inflection point angle	0x0030	DWORD	4	degree	V	Less than 180 degrees
The highest speed	0x0055	DWORD	4	km/h	V	
The time of overspeed duration	0x0056	DWORD	4	second	V	
The data of odometer	0x0080	DWORD	4	1/10km	V	For example:103 = 10.3km
Device ID	0xF000	STRING	/	/	GAV	The max BCD code is 12 by default.
The voltage in run mode	0xF001	WORD	2	millivolt	V	
Stop voltage	0xF002	WORD	2	millivolt	V	
The voltage in sleep mode	0xF003	WORD	2	millivolt	V	
NTP server address	0xF004	STRING	/	/	GAV	Support domain name and IP.
NTP server port	0xF005	DWORD	4	/	GAV	
Timezone	0xF006	BYTE	1	/	GAV	[-12,12]
The type of protocol	0xF007	BYTE	1	/	GAV	[0:JTT808 1:TAIP]
The encryption of protocol	0xF009	BYTE	1	/	GAV	[0:NULL 1:RSA 2:AES 3:XTEA]
Positioning Galaxy	0xF00A	BYTE	1	/	GAV	[0:GPS+BD 1:GPS+GLO 2:GPS+GAL]
WIFI enable	0xF00B	BYTE	1	/	GAV	[0:off 1:on]
WIFI work mode*	0xF00C	BYTE	1	/	GAV	[0:AP 1:STA]
The max AP of WIFI	0xF00D	BYTE	1	/	GAV	
WiFi single scan time	0xF00E	WORD	2	second	GA	
BT enable	0xF00F	BYTE	1	/	GAV	[0:off 1:open]
BT work mode	0xF010	BYTE	1	/	GAV	[0:host 1:slave]
Maximum number nodes of BT	0xF011	BYTE	1	/	GAV	
The timeout of BT nodes	0xF012	BYTE	1	minute	GAV	BT judges whether the node is offline through automatic scanning
BT single scan time	0xF013	WORD	2	second	GA	
BT report mask	0xF014	BYTE	1	/	GAV	
Input GPIO mode	0xF015	WORD	2	/	V	High byte [channel number], low byte [0: digital 1: analog]
GPIO direction*	0xF016	BYTE	1	/	V	
Transmit protocol	0xF017	BYTE	1	/	GAV	[0:TCP 1:UDP 2:MQTT]
Report mask	0xF018	DWORD	4	/	GAV	
Gsensor enable	0xF019	BYTE	1	/	GA	[0:off 1:on]
Gsensor sensitivity	0xF01A	BYTE	1	/	GAV	[0-255]
Gsensor range	0xF01B	BYTE	1	/	GAV	[0:2g 1:4g 2:8g 3:16g]
Gsensor times	0xF01C	WORD	2	/	GA	
Gsensor time	0xF01D	DWORD	4	second	GAV	
Gsensor trigger interval	0xF01E	DWORD	4	second	GA	
Gsensor report mask	0xF01F	BYTE	1	/	GAV	
Light enable	0xF020	BYTE	1	/	GA	[0:off 1:on]
Light hreshold	0xF021	WORD	2	/	GA	
Light trigger interval	0xF022	DWORD	4	second	GA	
Temp&hum enable	0xF023	BYTE	1	/	G	[0:off 1:on]
Upper temperature limit	0xF024	WORD	2	/	G	
Lower temperature limit	0xF025	WORD	2	/	G	
Upper humidity limit	0xF026	WORD	2	/	G	
Lower humidity limit	0xF027	WORD	2	/	G	
Temp&humi trigger interval	0xF028	DWORD	4	second	G	
GPS enable	0xF029	BYTE	1	/	GA	[0:off 1:on]
Work mode	0xF02A	BYTE	1	/	GA	
Backup server port	0xF02B	DWORD	4	/	GAV	TCP or UDP port
Buffer storage switch.*	0xF02C	BYTE	1	/	GAV	
Server ACK switch*	0xF02D	BYTE	1	/	GAV	
Reporting cycle	0xF02E	DWORD	4	second	GA	
Sampling period	0xF02F	DWORD	4	second	GA	
Transparent transmission AT command	0xF030	STRING	/	/	GAV	More details find in AT commands

0x8104 (query device parameters)

params is null

0x8105（Control commands）

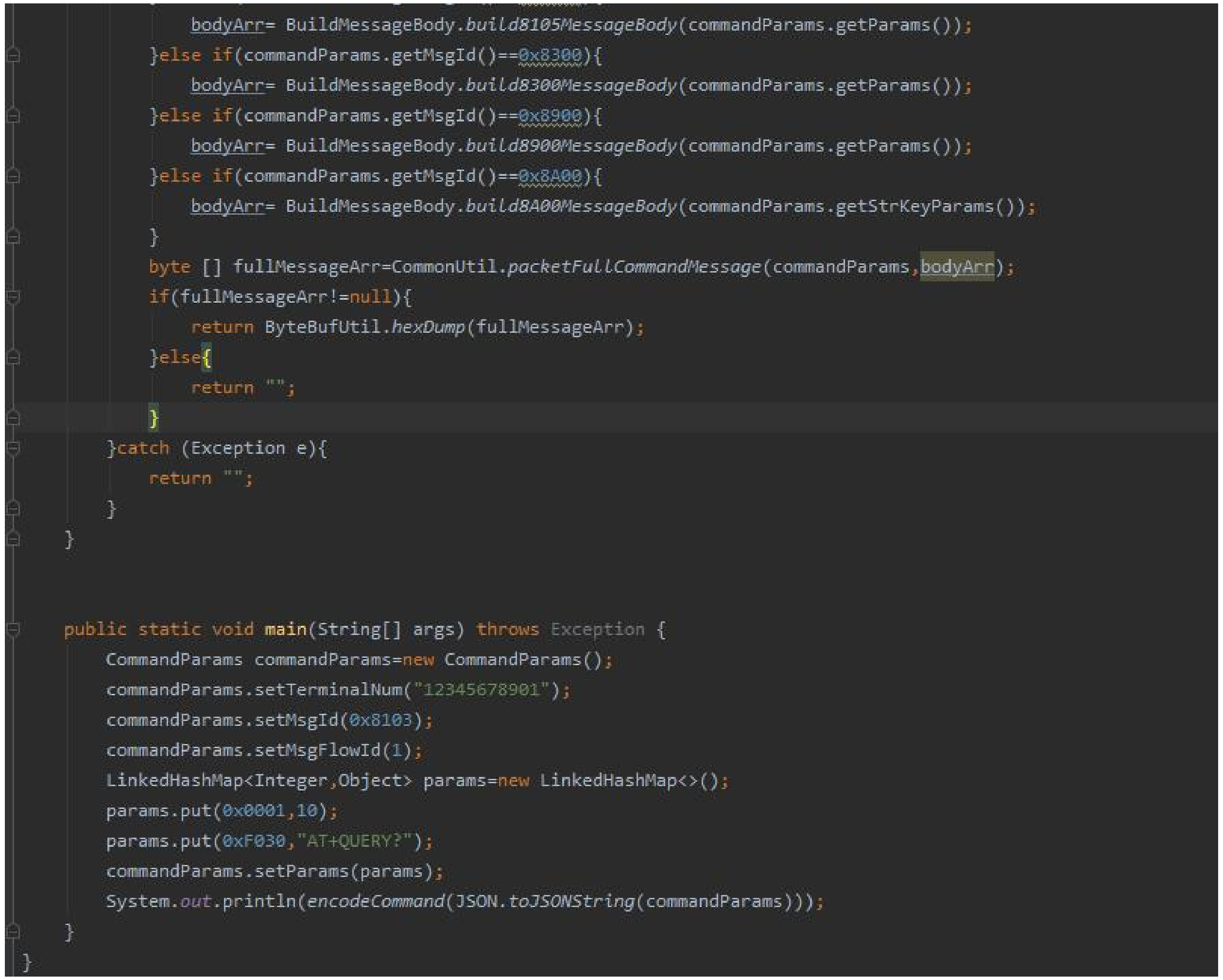


Command	Command Word	Command Parameter	Type	Description
Restart	4	/	GAV	
Restore factory settings	5	/	GAV	
OTA upgrade	32	TYPE; MODE; VERSION; PROTOCOL; URL; MD5	GAV	TYPE:0:app upgrade,1: core upgrade MODE:0:full package,1: diff package VERSION:preupgrade version PROTOCOL:0: FTP protocol,1: HTTP protocol URL:The full URL connection used for the actual upgrade MD5:The MD5 value of firmware
Fuel control	33	MODE	V	MODE:0:connect 1:disconnect
Power out control	34	MODE	V	MODE:0:off 1:on
GPIO output	35	CHANNEL;MODE	V	GPIO Output, CHANNEL: 0-15 MODE:0:off 1:on
Transparent transmission AT	36	Command	GAV	COMMAND: refer to AT command

4.3.Example of packaging method

This method takes 0x8103 as an example, while setting the heartbeat interval and sending AT commands
The corresponding JSON string is as follows:

```
{"msgFlowId":1,"msgId":33027,"params":{1:10,61488:"AT+QUERY?"},"terminalNum":"12345678901"}
```



```
                bodyArr= BuildMessageBody.build8105MessageBody(commandParams.getParams());
            }else if(commandParams.getMsgId()==0x8300){
                bodyArr= BuildMessageBody.build8300MessageBody(commandParams.getParams());
            }else if(commandParams.getMsgId()==0x8900){
                bodyArr= BuildMessageBody.build8900MessageBody(commandParams.getParams());
            }else if(commandParams.getMsgId()==0x8A00){
                bodyArr= BuildMessageBody.build8A00MessageBody(commandParams.getStrKeyParams());
            }
            byte [] fullMessageArr=CommonUtil.packetFullCommandMessage(commandParams,bodyArr);
            if(fullMessageArr!=null){
                return ByteBufUtil.hexDump(fullMessageArr);
            }else{
                return "";
            }
        }catch (Exception e){
            return "";
        }
    }


    public static void main(String[] args) throws Exception {
        CommandParams commandParams=new CommandParams();
        commandParams.setTerminalNum("12345678901");
        commandParams.setMsgId(0x8103);
        commandParams.setMsgFlowId(1);
        LinkedHashMap<Integer,Object> params=new LinkedHashMap<>();
        params.put(0x0001,10);
        params.put(0xF030,"AT+QUERY?");
        commandParams.setParams(params);
        System.out.println(encodeCommand(JSON.toJSONString(commandParams)));
    }
}
```