目　录

JT701D SDK 集成开发

.NET版本

Java版本

.NET 版本

1.SDK下载

JT701SDK(JT701SDK_V2.rar) 下载地址
Jt701SDK测试工具(JT701SDK_Test.rar) 下载地址
如果需要测试工具及SDK开发源码，请与商务申请

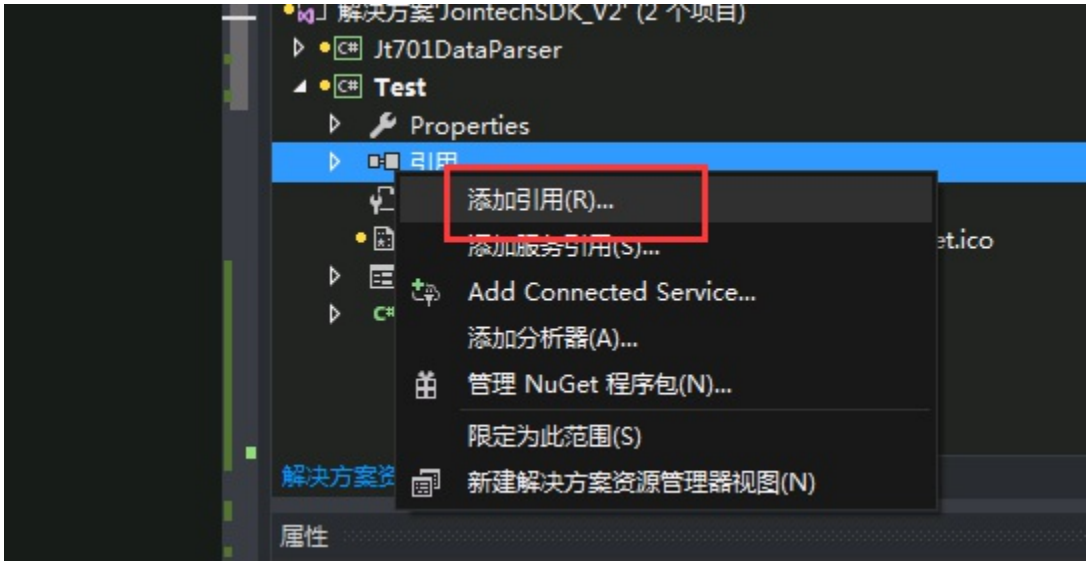
2.集成开发说明

2.1.集成开发语言及框架说明

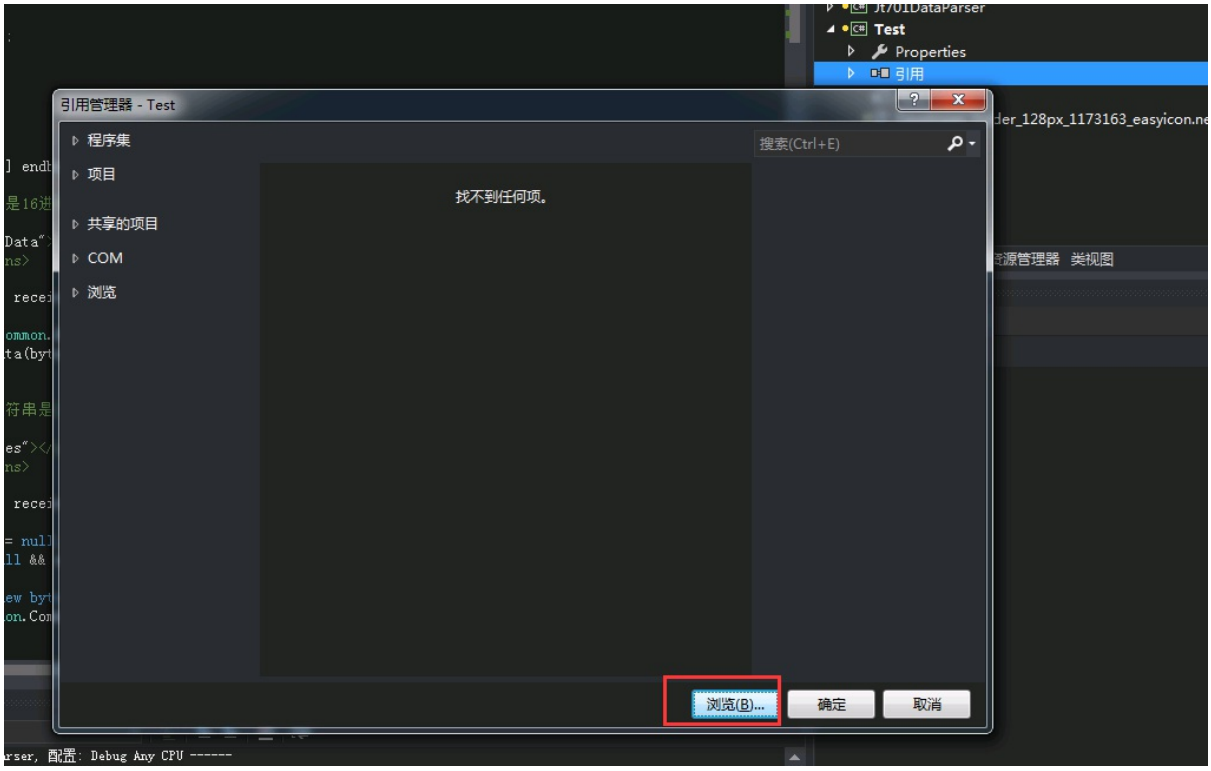
JT701SDK（Jt701DataParser.dll）及测试工具（Test.exe）是基于C#语言，.NET Framework 4.6.1目标框架开发的。该SDK的集成开发环境也是需要依赖于C#语言以及.NET Framework 4.6.1目标框架。

2.2.集成说明

（1）将Jt701DataParser.dll引入到自己的网关程序中，引入方法如下：
鼠标右键项目下的"引用"，选择"添加引用(R)..."



选择下图中"浏览(B)..."，找到你解压JT701SDK20210327.rar后的文件夹，选中Jt701DataParser.dll与对应的Json字符串序列化/反序列化包Newtonsoft.Json.dll即可完成对SDK的引用



（2）调用Jt701DataParser.dll中的解析方法receiveData

receiveData()是一个重载方法，你可以传入16进制字符串或者byte[],一般我们网关程序接收到的设备数据是以二进制流进行传输的，所以我们这里建议使用此重载方法receiveData(byte[] bytes);

当然如果你想通过16进制字符串进行测试，也可以使用receiveData(string strData)；

strData：就是我们接收到的设备数据转成16进制后的字符串，

例如定位数据：

24804050037019110034260321155955331972940200200850586000010e10c0000000020e043600a526d1700020f0f0f0f0f0f0f0f0f0eaa028f018b

指令数据：

283738393036323932383342c5034352c3236303332312c3138353334322c32362e3138313239362c4e2c35302e3332393738382c452c562c302e30302c302c352c302c303030303030303030302c302c302c3029

2.3.核心代码

Jt701DataParser.dll

（1）解析类DataParser.cs

```
public class DataParser
    {
        private static byte[] endbytes = null;// 上一次未处理的剩余字节

        /// <summary>
        /// 解析16进制原始数据
        /// </summary>
        /// <param name="strData"></param>
```

```
        /// <returns></returns>
        public static string receiveData(string strData)
        {
            byte[] bytes = Common.HexStringToBytes(strData);
            return receiveData(bytes);
        }

        /// <summary>
        /// 解析2进制原始数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static string receiveData(byte[] bytes)
        {
            byte[] newBytes = null;
            if (endbytes!=null && endbytes.Length > 0)
            {
                newBytes = new byte[endbytes.Length + bytes.Length];
                bytes = Common.CombineBytes(endbytes, bytes);
            }
            else
            {
                newBytes = new byte[bytes.Length];
                newBytes = bytes;
            }
            int i = 0;//初始化byte[]下标
            byte[] parserBytes = null;//去除正确包头之前的数据后的数据
            foreach (var item in newBytes)
            {
                if (item == '$')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserLocation(parserBytes);
                }
                else if (item == '(')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserCommand(parserBytes);
                }
                i++;
            }
            return null;
```

```
        }

        /// <summary>
        /// 解析定位数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static string parserLocation(byte[] bytes)
        {
            //定义定位数据实体类
            Result model = new Result();
            try
            {
                //跳过包头，然后解析设备ID
                model.DeviceID= Common.ByteToHexStr(bytes.Skip(1).Take(5).T
oArray());
                model.MsgType = "Location";
                LocationData location = new LocationData();
                location.ProtocolType = bytes[6];
                location.DeviceType = bytes[7] & 0x0F;
                location.DataType = (bytes[7]>>4) & 0x0F;
                //得到数据长度
                location.DataLength = Common.SwapUInt16(BitConverter.ToUInt
16(bytes, 8));
                //获取时间段，转成我们识别的"yyyy-MM-dd HH:mm:ss"格式
                location.GpsTime = Common.GetDataTime(bytes.Skip(10).Take(6
).ToArray());
                //纬度信息解析
                LatLon latLon = Common.GetLatLon(bytes.Skip(16).Take(9).ToA
rray());
                location.Latitude = latLon.Latitude;//纬度
                location.Longitude = latLon.Longitude;//经度
                location.LocationType = latLon.LocationType;//定位状态
                //解析速度
                location.Speed = (int)(bytes[25]*1.85);
                //解析方向
                location.Direction = bytes[26] * 2;
                //解析里程
                location.Mileage = Common.SwapUInt32(BitConverter.ToUInt32(
bytes, 27));//里程
                //解析GPS卫星个数
                location.GpsSignal = bytes[31];
                //解析设备状态
                DeviceStatus deviceStatus = Common.GetDeviceStatus(bytes.Sk
ip(36).Take(2).ToArray(), location.LocationType);
                //解析是否基站定位（GPS定位>基站定位>不定位）
```

```
                location.LocationType = deviceStatus.LocationType;
                //解析报警类型
                location.Alarm = deviceStatus.Alarm;
                //解析锁绳状态
                location.LockRope = deviceStatus.LockRope;
                //解析锁状态
                location.LockStatus = deviceStatus.LockStatus;
                //解析后盖状态
                location.BackCover = deviceStatus.BackCover;
                //获取电量（255为充电中）
                location.Battery = bytes[38];
                //解析小区码CellID低2位数据
                string strLowCellID = Common.ByteToHexStr(bytes.Skip(39).Ta
ke(2).ToArray());
                //解析小区码LAC
                location.LAC = Common.SwapUInt16(BitConverter.ToUInt16(byte
s, 41));
                //解析GSM信号值
                location.GSMSignal = bytes[43];
                //解析区域ID
                location.AlarmArea = bytes[44];
                //得到唤醒源
                location.Awaken = bytes[45];
                //得到IMEI号
                location.IMEI = Common.ByteToHexStr(bytes.Skip(48).Take(8).
ToArray());
                ////解析小区码CellID高2位数据
                string strHightCellID = Common.ByteToHexStr(bytes.Skip(56).
Take(2).ToArray());
                location.CELLID = Convert.ToInt64(strHightCellID + strLowCe
llID, 16);
                location.MCC = Common.SwapUInt16(BitConverter.ToUInt16(byte
s, 58));
                location.MNC = bytes[60];
                location.Index= bytes[61];
                model.DataBody = location;
                if (location.ProtocolType < 0x19)
                {
                    model.ReplyMsg= "(P35)";
                }
                else
                {
                    model.ReplyMsg = string.Format("(P69,0,{0})", location.
Index);
                }
            }
```

```
            catch (Exception ex)
            {
                return null;
            }
            return JsonConvert.SerializeObject(model);
        }

        /// <summary>
        /// 解析指令数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static string parserCommand(byte[] bytes)
        {
            Result result = new Result();
            byte[] newBytes = null;
            int tailIndex = Common.BytesIndexOf(bytes,0x29)+1;
            if (tailIndex > 0)
            {
                if (bytes.Length > tailIndex)
                {
                    endbytes = bytes.Skip(tailIndex).Take(bytes.Length - ta
ilIndex).ToArray();
                }
                newBytes= bytes.Skip(0).Take(tailIndex).ToArray();
            }
            else
            {
                endbytes = new byte[bytes.Length];
                endbytes = bytes;
            }
            if (newBytes!=null && newBytes.Length > 0)
            {
                //转换成（）带括号的数据
                string msgAscii = Common.ByteToASCII(newBytes);
                //按逗号拆分字符串
                char[] p = new char[] { ',' };
                //返回的数组中，包含空字符串
                string[] msgArrays = msgAscii.Replace("(", "").Replace(")",
 "").Split(p, StringSplitOptions.None);
                result.DeviceID = msgArrays[0];
                string msgType = string.Empty;
                if (msgArrays.Length > 5 && msgArrays[3] == "WLNET")
                {
                    msgType = msgArrays[3] + msgArrays[4];
                }
```

```
                else
                {
                    msgType = msgArrays[1];
                }
                result.MsgType = msgType;
                if (msgType.Equals("WLNET5"))
                {
                    SensorData sensorData = parserWLNET5(msgArrays, newByte
s);
                    result.DataBody = sensorData;
                    result.ReplyMsg = Common.replyMessage(msgType, sensorDa
ta.Index);
                }
                else
                {
                    if (msgType.Equals("P45"))
                    {
                        LockEvent lockEvent = parserP45(msgArrays);
                        result.DataBody = lockEvent;
                    }
                    else
                    {
                        result.DataBody = msgAscii;
                    }
                    result.ReplyMsg = Common.replyMessage(msgType, msgArray
s);
                }
                return JsonConvert.SerializeObject(result);
            }
            else
            {
                return null;
            }
        }

        /// <summary>
        /// WLNET5透传数据解析
        /// </summary>
        /// <param name="msgArrays"></param>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static SensorData parserWLNET5(string [] msgArrays, byte[]
inbytes)
        {
            byte[] bytes = Common.unescape(inbytes, inbytes.Length).ToArray
();
```

```
            if (bytes.Length > 30)
            {
                //透传数据之外的消息头长度
                int headLength = string.Format("({0},{1},{2},{3},{4},{5},",
 msgArrays[0], msgArrays[1], msgArrays[2], msgArrays[3], msgArrays[4], msgA
rrays[5]).Length;
                SensorData sensor = new SensorData();
                byte[] wlnetBytes = new byte[bytes.Length - headLength];
                //得到无限网关透传数据
                wlnetBytes = bytes.Skip(headLength).Take(bytes.Length - hea
dLength).ToArray();
                //获取时间段，转成我们识别的"yyyy-MM-dd HH:mm:ss"格式
                sensor.GpsTime = Common.GetDataTime(wlnetBytes.Skip(0).Take
(6).ToArray());
                //纬度信息解析
                LatLon latLon = Common.GetLatLon(wlnetBytes.Skip(6).Take(9)
.ToArray());
                sensor.Latitude = latLon.Latitude;//纬度
                sensor.Longitude = latLon.Longitude;//经度
                sensor.LocationType = latLon.LocationType;//定位状态
                //解析速度
                sensor.Speed = (int)(wlnetBytes[15]*1.85);
                //解析方向
                sensor.Direction = wlnetBytes[16] * 2;
                sensor.DateTime = Common.GetDataTime(wlnetBytes.Skip(17).Ta
ke(6).ToArray());
                sensor.SensorID = Common.ByteToHexStr(wlnetBytes.Skip(23).T
ake(5).ToArray());
                sensor.Index = wlnetBytes[28];
                sensor.Voltage = (Common.SwapUInt16(BitConverter.ToUInt16(w
lnetBytes, 29)) / 100.0).ToString("f2");
                sensor.Power = wlnetBytes[31];
                sensor.RSSI = -wlnetBytes[32];
                sensor.SensorType = wlnetBytes[33];
                int fEventType = -1;
                if (sensor.SensorType == 1)
                {
                    sensor.Temperature = Common.SwapUInt16(BitConverter.ToU
Int16(wlnetBytes, 34)) * 0.1;
                    sensor.Humidity = wlnetBytes[36];
                }
                else if (sensor.SensorType == 4)
                {
                    fEventType = Common.SwapUInt16(BitConverter.ToUInt16(wl
netBytes, 34));
                    if ((fEventType & 0x0001) > 0)
```

```
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_0;
                    }
                    else if ((fEventType & 0x0002) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_1;
                    }
                    else if ((fEventType & 0x0004) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_2;
                    }
                    else if ((fEventType & 0x0008) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_3;
                    }
                    else if ((fEventType & 0x0010) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_4;
                    }
                    else if ((fEventType & 0x0020) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_5;
                    }
                    else if ((fEventType & 0x0040) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_6;
                    }
                    else if ((fEventType & 0x0080) > 0)
                    {
                        fEventType = (int)EventTypeEnum.LockEvent_7;
                    }
                    else
                    {
                        fEventType = -1;
                    }
                    sensor.Event = fEventType;
                    sensor.LockRope = (Common.SwapUInt16(BitConverter.ToUIn
t16(wlnetBytes, 36)) & 0x0001);
                    sensor.LockStatus = sensor.LockRope;
                    sensor.LockTimes = Common.SwapUInt16(BitConverter.ToUIn
t16(wlnetBytes, 38));
                }
                return sensor;
            }
            else
            {
```

```
                return null;
            }
        }

        /// <summary>
        /// P45开关锁事件解析
        /// </summary>
        /// <param name="msgArrays"></param>
        /// <returns></returns>
        private static LockEvent parserP45(string[] msgArrays)
        {
            LockEvent model = new LockEvent();
            model.DateTime= Common.GetDataTime(msgArrays[2] + msgArrays[3])
;
            model.Latitude = double.Parse(msgArrays[4]);
            if (msgArrays[5].Equals("S"))
            {
                model.Latitude = -model.Latitude;
            }
            model.Longitude = double.Parse(msgArrays[6]);
            if (msgArrays[7].Equals("W"))
            {
                model.Longitude = -model.Longitude;
            }
            model.LocationType= msgArrays[8].Equals("V") ? 0 : 1;
            model.Speed= (int)double.Parse(msgArrays[9]);
            model.Direction = int.Parse(msgArrays[10]);
            model.Event = int.Parse(msgArrays[11]);
            //开锁验证
            int status= int.Parse(msgArrays[12]);
            model.RFIDNo = msgArrays[13];
            //动态密码开锁
            if (model.Event == 6)
            {
                if (status == 0)
                {
                    model.Status = 0;//开锁密码不正确
                }
                else if (status > 0 && status <= 10)
                {
                    model.Status = 1;//正常开锁
                    model.UnlockFenceID = status;//围栏内开锁时候的围栏ID
                }
                else if (status == 98)
                {
                    model.Status = 1;//正常开锁
```

```
                }
                else if (status == 99)
                {
                    model.Status = 3;//设备开启了围栏内开锁，且当前开锁并未在围
栏内，拒绝开锁
                }
            }
            else if (model.Event == 4)
            {
                if (int.Parse(msgArrays[14]) == 0)
                {
                    model.Status = 0;//开锁密码不正确
                }
                else
                {
                    model.Status = 1;//正常开锁
                }
            }
            model.PsdErrorTimes = int.Parse(msgArrays[15]);
            model.Index = int.Parse(msgArrays[16]);
            if (msgArrays.Length > 17)
            {
                model.Mileage = long.Parse(msgArrays[17]);
            }
            return model;
        }
    }
```

（2）工具类Common.cs

```
public static class Common
    {

        /// <summary>
        /// 16进制格式字符串转字节数组
        /// </summary>
        /// <param name="hexString"></param>
        /// <returns></returns>
        public static byte[] HexStringToBytes(string hexString)
        {
            hexString = Regex.Replace(hexString, @".{2}", "$0 ");
            //以 ' ' 分割字符串，并去掉空字符
            string[] chars = hexString.Split(new char[] { ' ' }, StringSpli
tOptions.RemoveEmptyEntries);
            byte[] returnBytes = new byte[chars.Length];
```

```
            //逐个字符变为16进制字节数据
            for (int i = 0; i < chars.Length; i++)
            {
                returnBytes[i] = Convert.ToByte(chars[i], 16);
            }
            return returnBytes;
        }

        /// <summary>
        /// 合并数组
        /// </summary>
        /// <param name="bytes1"></param>
        /// <param name="bytes2"></param>
        /// <returns></returns>
        public static byte[] CombineBytes(byte[] bytes1, byte[] bytes2)
        {
            List<byte> tmp = new List<byte>(bytes1.Length + bytes2.Length);
            tmp.AddRange(bytes1);
            tmp.AddRange(bytes2);
            byte[] merged = tmp.ToArray();
            return merged;
        }

        /// <summary>
        /// 报告指定的 System.Byte[] 在此实例中的第一个匹配项的索引。
        /// </summary>
        /// <param name="srcBytes">被执行查找的 System.Byte[]。</param>
        /// <param name="searchBytes">要查找的 System.Byte[]。</param>
        /// <returns>如果找到该字节数组，则为 searchBytes 的索引位置；如果未找
到该字节数组，则为 -1。如果 searchBytes 为 null 或者长度为0，则返回值为 -1。</re
turns>
        public static int BytesIndexOf(byte[] srcBytes, byte searchBytes)
        {
            if (srcBytes == null) { return -1; }
            if (srcBytes.Length == 0) { return -1; }
            for (int i = 0; i < srcBytes.Length; i++)
            {
                if (srcBytes[i] == searchBytes)
                {
                    return i;
                }
            }
            return -1;
        }

        /// <summary>
```

```
        /// 16进制字符串转ASCII
        /// </summary>
        /// <param name="Data"></param>
        /// <param name="istrans"></param>
        /// <returns></returns>
        public static string HexStrToAsciiString(string Data, bool istrans)
        {
            if (istrans)
            {
                Data = Regex.Replace(Data, @"(?is)(?<=^([0-9a-f]{2})+)(?!$)
", " ");
                Data = Data.Replace("3D 11", "2C ").Replace("3D 14", "28 ")
.Replace("3D 15", "29 ").Replace("3D 00", "3D ").Replace(" ", "");
            }
            int count = Data.Length / 2;
            string strContent = "";
            char[] num = new char[count];
            for (int i = 0; i < count; i++)
            {
                string str = Data.Substring(i * 2, 2);
                byte by = Convert.ToByte(Convert.ToInt32(str, 16));
                num[i] = Convert.ToChar(by);
                strContent += num[i].ToString();
            }
            return strContent;
        }

        /// <summary>
        /// 二进制转ASCII
        /// </summary>
        /// <param name="bt"></param>
        /// <returns></returns>
        public static string ByteToASCII(byte[] bt)
        {
            string lin = "";
            for (int i = 0; i < bt.Length; i++)
            {
                lin = lin + bt[i] + " ";
            }
            string[] ss = lin.Trim().Split(new char[] { ' ' });
            char[] c = new char[ss.Length];
            int a;
            for (int i = 0; i < c.Length; i++)
            {
                a = Convert.ToInt32(ss[i]);
                c[i] = Convert.ToChar(a);
```

```
            }

            string b = new string(c);
            return b;
        }

        /// <summary>
        /// 应答回复
        /// </summary>
        /// <param name="msgType"></param>
        /// <param name="strArry"></param>
        /// <returns></returns>
        public static string replyMessage(string msgType,string [] strArry)
        {
            string replyContent = string.Empty;
            switch (msgType)
            {
                case "P22":
                    replyContent = string.Format("(P22,{0})", DateTime.UtcN
ow.ToString("ddMMyyHHmmss"));
                    break;
                case "P43":
                    if (strArry[2].Equals("0"))
                    {
                        replyContent = string.Format("{0}", "P44,1,888888")
;//密码重置
                    }
                    break;
                case "P45":
                    replyContent = string.Format("(P69,0,{0})", strArry[16]
);
                    break;
                case "P52":
                    if (strArry[2].Equals("2"))
                    {
                        replyContent = string.Format("(P52,2,{0})", strArry
[3]);
                    }
                    break;
                default:
                    break;
            }
            return replyContent;
        }

        /// <summary>
```

```
        /// 应答回复
        /// </summary>
        /// <param name="msgType"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public static string replyMessage(string msgType, int index)
        {
            string replyContent = string.Empty;
            switch (msgType)
            {
                case "WLNET5":
                case "WLNET7":
                    replyContent = string.Format("(P69,0,{0})", index);
                    break;
                default:
                    break;
            }
            return replyContent;
        }

        /// <summary>
        /// 时间格式转换
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DateTime GetDataTime(byte[] bytes)
        {
            return DateTime.ParseExact(ByteToHexStr(bytes), "ddMMyyHHmmss",
 System.Globalization.CultureInfo.CurrentCulture);
        }
        public static DateTime GetDataTime(string strdate)
        {
            return DateTime.ParseExact(strdate, "ddMMyyHHmmss", System.Glob
alization.CultureInfo.CurrentCulture);
        }
        /// <summary>
        /// 字节数组转16进制字符串
        /// </summary>
        /// <param name="byteDatas"></param>
        /// <returns></returns>
        public static string ByteToHexStr(byte[] byteDatas)
        {
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < byteDatas.Length; i++)
            {
                builder.Append(string.Format("{0:X2}", byteDatas[i]));
```

```
            }
            return builder.ToString().Trim();
        }

        public static LatLon GetLatLon(byte[] bytes)
        {
            try
            {
                LatLon model = new LatLon();
                model.Latitude = GetLatLong60(bytes.Skip(0).Take(4).ToArray
()); 
                model.Longitude = GetLatLong60(bytes.Skip(4).Take(5).ToArra
y()); 
                model.LocationType = bytes[8] & 0x01;
                if ((bytes[8] & 0x02) == 0)
                {
                    model.Latitude = -model.Latitude;
                }
                if ((bytes[8] & 0x04) == 0)
                {
                    model.Longitude = -model.Longitude;
                }
                return model;
            }
            catch (Exception ex)
            {
                return null;
            }
        }

        /// <summary>
        /// 经纬度计算
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static double GetLatLong60(byte[] bytes)
        {
            try
            {
                string locStr = ByteToHexStr(bytes);
                if (locStr.Length < 9)
                {
                    locStr = locStr.PadLeft(9, '0');
                }
                else
                {
```

```
                    locStr = locStr.Substring(0, 9);
                }
                var head = Convert.ToDouble(locStr.Remove(3));
                var bodyStr = locStr.Substring(3, locStr.Length - 3);
                var body = Convert.ToDouble(bodyStr) / 10000;
                head += body / 60;
                return head;
            }
            catch (Exception ex)
            {
                return 0;
            }
        }

        public static ushort SwapUInt16(ushort v)
        {
            return (ushort)(((v & 0xff) << 8) | ((v >> 8) & 0xff));
        }

        public static uint SwapUInt32(uint v)
        {
            return (uint)(((SwapUInt16((ushort)v) & 0xffff) << 0x10) |
                          (SwapUInt16((ushort)(v >> 0x10)) & 0xffff));
        }

        /// <summary>
        /// 获取设备状态
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DeviceStatus GetDeviceStatus(byte[] bytes, int locati
onType)
        {
            try
            {
                DeviceStatus model = new DeviceStatus();
                //低8位
                if (locationType == 0)
                {
                    model.LocationType = bytes[0] & 0x01;
                }
                else
                {
                    model.LocationType = locationType;
                }
                if ((bytes[1] & 0x02)>0)
```

```
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_9;
                }
                if ((bytes[1] & 0x04)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_10;
                }
                if ((bytes[1] & 0x08)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_1;
                }
                if ((bytes[1] & 0x10)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_2;
                }
                else
                {
                    model.Alarm = -1;
                }
                model.LockRope = (bytes[1] & 0x40) > 0 ? 0 : 1;
                model.LockStatus = (bytes[1] & 0x80) > 0 ? 0 : 1;
                //高8位
                if ((bytes[0] & 0x01)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_3;
                }
                if ((bytes[0] & 0x02)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_4;
                }
                if ((bytes[0] & 0x04)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_5;
                }
                if ((bytes[0] & 0x08)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_6;
                }
                if ((bytes[0] & 0x10)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_7;
                }
                model.BackCover = (bytes[1] & 0x20) > 0 ? 1 : 0;
                if ((bytes[0] & 0x40)>0)
                {
                    model.Alarm = (int)AlarmTypeEnum.LOCK_ALARM_8;
```

```
                }

                return model;
            }
            catch (Exception ex)
            {
                return null;
            }
        }

        /// <summary>
        /// 转义
        /// </summary>
        /// <param name="inBytes"></param>
        /// <param name="bodyLen"></param>
        /// <returns></returns>
        public static List<byte> unescape(byte[] inBytes, int bodyLen)
        {
            List<byte> list = new List<byte>();
            int i = 0;
            while (i < bodyLen)
            {
                int b = inBytes[i];
                if (b == 0x3D)
                {
                    int nextByte = inBytes[i+1]; ;
                    if (nextByte == 0x14)
                    {
                        list.Add(0x3D ^ 0x14);
                    }
                    else if (nextByte == 0x15)
                    {
                        list.Add(0x3D ^ 0x15);
                    }
                    else if (nextByte == 0x00)
                    {
                        list.Add(0x3D ^ 0x00);
                    }
                    else if (nextByte == 0x11)
                    {
                        list.Add(0x3D ^ 0x11);
                    }
                    else
                    {
                        list.Add(BitConverter.GetBytes(b)[0]);
                        list.Add(BitConverter.GetBytes(nextByte)[0]);
```

```
                    }
                    i += 2;
                }
                else
                {
                    list.Add(BitConverter.GetBytes(b)[0]);
                    i++;
                }
            }
            return list;
        }
    }
```

2.4.返回消息及说明

（1）定位数据
原始数据：

```
24804050022519110034260321185303295324160310089 41d0000000018070c0000000020e
04f8b0c56001f00020f0f0f0f0f0f0f0f0f0f0f00f2028f0157
```

返回消息：

```
{
    "DeviceID": "8040500225",
    "MsgType": "Location",
    "DataBody": {
        "ProtocolType": 25,
        "DeviceType": 1,
        "DataType": 1,
        "DataLength": 52,
        "GpsTime": "2021-03-26T18:53:03",
        "Latitude": -29.88736,
        "Longitude": 31.014901666666667,
        "LocationType": 1,
        "Speed": 0,
        "Direction": 0,
        "Mileage": 6151,
        "GpsSignal": 12,
        "GSMSignal": 31,
        "Alarm": -1,
        "AlarmArea": 0,
        "Battery": 79,
        "LockStatus": 0,
```

```
        "LockRope": 0,
        "BackCover": 1,
        "MCC": 655,
        "MNC": 1,
        "LAC": 22016,
        "CELLID": 15895308,
        "IMEI": "0F0F0F0F0F0F0F0F",
        "Awaken": 2,
        "Index": 87
    },
    "ReplyMsg": "(P69,0,87)"
}
```

返回消息描述

```
{
    "DeviceID"：设备ID
    "MsgType"：消息类型， 此处为： Location,表示定位数据,
    "DataBody"：消息体内容
    {
        "ProtocolType": 协议版本号,
        "DeviceType": 终端类型号,
        "DataType": 数据类型号（ 1 表明最新二进制定位数据， 2 表示报警数据， 3 表示盲区常规二进制定位数据， 4 表示次新二进制定位数据）,
        "DataLength": 数据长度,
        "GpsTime": 定位时间（ GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位）,
        "Speed": 速度（ 单位： km / h）,
        "Direction": 方向（ 0~360； 0 与360表示正北方向）,
        "Mileage": 当前里程值（单位：km）,
        "GpsSignal": GPS卫星个数,
        "GSMSignal": GSM信号值,
        "Alarm": 报警类型（ - 1： 无告警信息； 1： 锁绳剪断； 2： 震动； 3： 长时间开锁； 4： 开锁密码连续5次错误； 5： 刷非法卡； 6： 低电量； 7： 开后盖； 8： 卡锁； 9： 进区域报警； 10： 出区域报警）,
        "AlarmArea": 如果告警与区域有关， 则此处值为区域的ID,
        "Battery": 电量值（ 0~100； 255： 充电中）,
        "LockStatus": 锁电机状态（ 1： 开； 0： 关）,
        "LockRope": 锁绳状态（ 1： 拔出； 0： 插入）,
        "BackCover": 后盖状态（ 1： 关闭； 0： 开启）,
        "MCC": 国家代码,
        "MNC": 运营商代码,
        "LAC": 位置区域码,
```

```
        "CELLID": 基站编号,
        "IMEI": IMEI号码， 全是0F无效,
        "Awaken": 唤醒源（ 0 重启,
        1: RTC唤醒， 2: 震动， 3: 开后盖， 4: 锁绳， 5: 接外电， 6: 刷卡， 7:
门磁， 8: VIP短信， 9: 非VIP短信或垃圾短信）,
        "Index": 数据流水号
    },
    "ReplyMsg"：回复内容（ 如果为空字符串， 则不需要给终端回复内容）
}
```

（2）传感器采集数据（WLNET5）

原始数据(SensorType=1)：

```
28383035303530303037332c312c3134312c574c4e45542c352c322c2603211847096496729
53949673d1408ff00260321184718102011098666013362 3c0100d71b00000029
```

返回消息(SensorType=1)：

```
{
    "DeviceID": "8050500073",
    "MsgType": "WLNET5",
    "DataBody": {
        "GpsTime": "2021-03-26T18:47:09",
        "Latitude": -65.612158333333326,
        "Longitude": -395.61215,
        "LocationType": 0,
        "Speed": 471,
        "Direction": 0,
        "SensorID": "1020110986",
        "LockStatus": 0,
        "LockRope": 0,
        "LockTimes": 0,
        "Index": 102,
        "Voltage": "3.07",
        "Power": 98,
        "RSSI": -60,
        "DateTime": "2021-03-26T18:47:18",
        "SensorType": 1,
        "Temperature": 21.5,
        "Humidity": 27,
        "Event": -1
    },
    "ReplyMsg": "(P69,0,102)"
}
```

原始数据(SensorType=4)：

```
28373030303331333330392C312C3038312C574C4E45542C352C322C0508211543072234825
0113550300F0000050821154304E0171E086925018D4E690400400000002A0029
```

返回消息(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)"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型， 此处为： WLNET5,表示传感器透传数据,
    "DataBody": 消息体内容
    {
        "GpsTime": 定位时间（GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
```

```
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位），
        "Speed": 速度（ 单位： km / h ）；0xFF表示速度无效，
        "Direction": 方向（ 0~360； 0 与360表示正北方向），
        "SensorID": 传感器ID，
        "LockStatus": 如果从机类型SensorType=4，此是才有效；1：开锁；0：关锁，
        "LockRope": 如果从机类型SensorType=4，此是才有效；1：锁绳拔出；0：锁绳
插入，
        "LockTimes": 开锁次数（如果SensorType=4此值才有效），
        "Index": 数据流水号，
        "Voltage": 电压值（单位：V），
        "Power": 传感器电量（0~100；255： 充电中），
        "RSSI": RSSI信号强度，是负数越接近0信号越好，
        "DateTime": 数据采集时间（GMT时间），
        "SensorType": 传感器类型（1：温湿度传感器(JT126)；4:从机JT709； ），
        "Temperature":温度值（如果SensorType=1此值才有效），
        "Humidity": 湿度值（如果SensorType=1此值才有效），
        "Event": 事件类型（-1：无从机事件；0：关锁事件；1：蓝牙开锁事件；2：NFC
开锁事件；3：Lora开锁事件；4：从机锁剪断报警事件；5：按键唤醒事件；6：定时上报事件
；7：充电上报事件）
    },
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

（3）锁事件上报数据（P45）

原始数据：

```
283738393036323932383432c5034352c3236303332312c3139343933392c32362e323732303
3352c4e2c35302e3632313433352c452c412c302e30352c302c342c312c3030303030303030
30302c312c302c3129
```

返回消息：

```
{
    "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)"
}
```

返回消息描述：

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型， 此处为： P45,表示开关锁事件数据上传,
    "DataBody": 消息体内容
    {
        "DateTime": 事件时间（GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位）,
        "Speed": 速度（ 单位： km / h）,
        "Direction": 方向（ 0~360； 0 与360表示正北方向）,
        "Event": 事件类型（1：表示刷授权卡；2：表示刷非法卡；3：表示刷车辆ID卡绑定；4：表示为凭密码开锁；5：表示终端自动关锁记录；6: 动态密码围栏内开锁；7: 蓝牙开锁）,
        "Status": 开锁验证（0：开锁密码不正确；1：正常开锁；2：因为开启了围栏开锁，未在围栏内开锁，开锁被拒绝）,
        "UnlockFenceID": （-1：开锁与围栏无关；1~10：标识对应的开锁围栏ID）,
        "RFIDNo": 刷卡卡号；如果未“0000000000”，则无效,
        "PsdErrorTimes": 开锁密码错误次数,
        "Index": 数据流水号,
        "Mileage": 当前里程值（单位：km）
    },
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

（4）其他指令回复数据
原始数据：

```
28373839303632393238342c50333529
```

返回消息：

```
{
    "DeviceID": "7890629284",
```

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

返回消息描述：

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型（参见更多的消息类型及其描述，请参阅3.消息类型及消息体内容描述），
    "DataBody": 消息体内容（除定位数据：Location；传感器透传数据：WLNET5;锁事件上报：P45外；其他此处均直接返回指令内容的ASCII字符串），
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

3.消息类型及消息体内容描述

3.1.P01:查询终端当前的版本号

消息体内容：（示例）
`(7591225008,P01,JT701D_20200720_China_Jointech_SIM7600,77%)`
消息体内容描述：
`7591225008：设备ID`
`P01：消息类型`
`JT701D_20200720_China_Jointech_SIM7600：协议版本`
`77%：当前设备电量`

P03:低电休眠控制

消息体内容：（示例）
`(7560704001,P03,1,30)`
消息体描述：
`7560704001：设备ID`
`P03：消息类型`
`1：生效；0：不生效`
`30：设置电量低于30的时候进入休眠，默认31%，可设定范围5%~90%`

P04：设置/查询数据上传间隔和休眠自动唤醒间隔

消息体内容：（示例）
`(7570101998,P04,30,30)`
消息体描述：

7570101998：设备ID
P04：消息类型
30：数据上传间隔，单位秒钟，默认30秒，取值范围5-600
30：休眠自动唤醒间隔，单位分钟，默认30分钟，取值范围30-1440

P06:设置/查询监控中心IP与端口、APN

消息体内容：（示例）
(7570101998,P06,211.162.111.225,10906,CMNET,user,password,1)
消息体描述：
7570101998：设备ID
P06：消息类型
211.162.111.225：监控中心的IP地址
10906：监控中心端口地址，最大65530
CMNET：接入点名称(最长50个字节)
User：APN用户名(最长50个字节)
Password：APN密码(最长50个字节)
1：0表示卡1，1表示卡2

P10:设置/查询终端使用地点与国际标准时间的时差

消息体内容：（示例）
(7570101998,P10,480)
消息体描述：
7570101998：设备ID
P10：消息类型
480：时差值,以分钟为单位.如北京时间与标准时时差为8小时,即为480分钟，取值范围-12*60-13*60，默认0

P11:设置/查询VIP手机号码

消息体内容：（示例）
(7570101998,P11,1,8613910102345)
消息体描述：
7570101998：设备ID
P11：消息类型
1：VIP手机号码索引,取值为1-5,允许有五组VIP手机号码
8613910102345：手机号码,不能超过15位数字，前面需加国际区号，中国为86或者+86.

P12:设置/查询VIP号码是否允许报警

消息体内容：（示例）
(7570101998,P12,1,1,1,1,1)
消息体描述：

`7570101998：设备ID`
`P12：消息类型`
`1,1,1,1,1:分别对应5个VIP号码是否允许报警；1表示允许对应的VIP号码报警，0表示不允许此VIP号码报警`

P13:恢复出厂设置

消息体内容：（示例）
`(7570101998,P13)`
消息体描述：
`7570101998：设备ID`
`P13：消息类型`

P14:读取终端的IMEI号

消息体内容：（示例）
`(7570101998,P14,012207004451636)`
消息体描述：
`7570101998：设备ID`
`P14：消息类型`
`012207004451636：终端的IMEI号`

P15:终端重启指令

消息体内容：（示例）
`(7570101998,P15)`
消息体描述：
`7570101998：设备ID`
`P15：消息类型`

P22:GPS无效的时候，监控中心对终端授时

消息体内容：（示例）
`(7570101998,P22,1)`
消息体描述：
`7570101998：设备ID`
`P22：消息类型`
`1：1表示授时成功，0表示失败，2表示主动请求授时`

P22:设置/取消短信、电话可唤醒工作模式

消息体内容：（示例）
`(7570101998,P23,1)`
消息体描述：

7570101998：设备ID
P23：消息类型
1：1表示设置成功，0表示设置失败.

P24:区域是否有效，及区域名称设置指令

消息体内容：（示例）
(7570101998,P24,10,1,area10)
消息体描述：
7570101998：设备ID
P24：消息类型
10：表示第10个区域.
1：表示有效 ，0表示无效
area10：表示区域名称，最大长度为16个字节.

P29:设置或者查询区域的详细节点信息

消息体内容：（示例）
(7570101998,P29,8,15,1,10,11323.1234…)
消息体描述：
7570101998：设备ID
P29：消息类型
8:表示第八个区域
15:表示总点数
1:表示当前页
10:表示当前页的点数.余下的为各个点的经度与纬度

P30:清除相关的区域

消息体内容：（示例）
(7570101998,P30,1)
消息体描述：
7570101998：设备ID
P30：消息类型
1:1表示清除成功，0表示清除失败.

P31:区域信息设置完毕

消息体内容：（示例）
(7570101998,P31)
消息体描述：
7570101998：设备ID
P31：消息类型

P32:主动进入休眠指令

消息体内容：（示例）

`(7570101998,P32)`

消息体描述：

`7570101998：设备ID`

`P32：消息类型`

P37:查询/设置G-sensor相关参数

消息体内容：（示例）

`(7570101998,P37,500)`

消息体描述：

`7570101998：设备ID`

`P37：消息类型`

`500：运动检测门限值，范围是63到500,单位是mg；如果设置为0则为关闭G-sensor相关全部功能；关闭G-sensor功能后，如需重新开启G-sensor功能只需重新设置有效的G-sensor参数即可.默认值126`

P38:开锁报警时间间隔设置指令

消息体内容：（示例）

`(7570101998,P38,120)`

消息体描述：

`7570101998：设备ID`

`P38：消息类型`

`120：开锁报警时间间隔设置，即从锁电机处于开锁状态时刻算起，如果该状态持续保持超过120分钟，则可触发开锁报警，范围是3到180,单位是分钟；默认为120分钟.`

P40:查询/设置GPRS通道和短消息通道的报警开关

消息体内容：（示例）

`(7570101998,P40,1,1,1,1,1,1,1,1,1,1,1)`

消息体描述：

`7570101998：设备ID`

`P40：消息类型`

`1,1,1,1,1,1,1,1,1,1,1：从左至右依次为锁挂绳剪断报警、刷非法卡报警、开锁状态保持一段时间报警、指令开锁密码连续输错5次报警、震动报警、进区域报警、出区域报警的开关、低电报警、开后盖报警、卡锁报警；每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.`

P41:增删开锁授权号指令

消息体内容：（示例1）

`(7570101998,P41,1,30)`

消息体描述：

`7570101998：设备ID`

`P41：消息类型`

`1：1表示增加授权卡号操作;2表示删除授权号，3表示删除所有的授权号`

`30：30表示当前已存授权卡号总个数;`

消息体内容：（示例2）

`(7570101998,P41,2,3,0013953759，0013953758，0013953757)`

消息体描述：

`7570101998：设备ID`

`P41：消息类型`

`2：查询第2组数据；一共分3组，分别为1~3，查询时每次最多返回20个ID号`

`3：3表示有3个ID号`

`0013953759，0013953758，0013953757表示该组存储的授权号列表`

P42:现场刷卡授权模式配置指令

消息体内容：（示例1）

`(7570101998,P42,0)`

消息体描述：

`7570101998：设备ID`

`P42：消息类型`

`1：1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。`

消息体内容：（示例2）

`(7570101998,P41,2,0013953759,0013953751)`

消息体描述：

`7570101998：设备ID`

`P42：消息类型`

`2：表示终端已存了2个开锁授权号.`

`0013953759,0013953751：授权卡号.`

P44:远程开锁密码修改指令

消息体内容：（示例1）

`(7570101998,P44,1)`

消息体描述：

`7570101998：设备ID`

`P44：消息类型`

`1：1：表示修改密码是否成功，1表示成功，0表示失败。`

消息体内容：（示例2）

`(7570101998,P44,888888)`

消息体描述：

`7570101998：设备ID`
`P44：消息类型`
`888888：当前的设备可开锁的动态密码`

P50:电源开关生效控制设置

消息体内容：（示例）
`(7570101998,P50,1)`
消息体描述：
`7570101998：设备ID`
`P44：消息类型`
`1：1表示开关有效，默认是1；0表示停用开关.`

P52:动态密码指令

消息体内容：（示例1）
`(7570101998,P52,0,405935,326387)`
消息体描述：
`7570101998：设备ID`
`P52：消息类型`
`0：指令操作；查询当前产生的动态密码和当前用来开锁的动态密码`
`405935：表示，还没有被系统确认的动态密码`
`326387：表示目前用来开锁的动态密码`

消息体内容：（示例2）
`(7570101998,P52,1,1,0)`
消息体描述：
`7570101998：设备ID`
`P52：消息类型`
`1：1表示设置动态密码功能`
`1：1查询是否开启动态密码功能；0：表示关闭动态密码功能`
`0：1表示动态密码开锁必须在区域内才能开锁，即表明如果想要开锁，必须设置区域，0表示动态密码开锁跟区域无关，只要符合动态密码开锁的其它条件就可以开锁`

消息体内容：（示例3）
`(7570101998,P52,2,405935)`
消息体描述：
`7570101998：设备ID`
`P52：消息类型`
`2：回复上传的动态密码确认指令`
`405935：需要被确认的动态密码`

消息体内容：（示例4）
`(7570101998,P52,3,1,0)`
消息体描述：

`7570101998：设备ID`
`P52：消息类型`
`3：发送动态密码开锁`
`1：1,代表成功，0是失败`
`0：代表失败次数`

P54:查询/设置是否关闭休眠模式指令(设备不休眠)

消息体内容：（示例）
`(7570101998,P54,0,0)`
消息体描述：
`7570101998：设备ID`
`P54：消息类型`
`0：0 查询；1 设置`
`0：0 需要休眠 ， 1 开启不休眠`

P58:查询和设置RFID卡是否关联电子围栏 (默认关联电子围栏)

消息体内容：（示例）
`(7570101998,P58,1,1)`
消息体描述：
`7570101998：设备ID`
`P58：消息类型`
`1：1表示设置，0表示查询`
`1：1：关联电子围栏，必须在区域内才能刷卡开锁；0表示刷卡开锁跟区域无关，只要符刷卡开锁的条件就可以开锁`

P61:设置或查询电量低报警提示的阀值

消息体内容：（示例）
`(7570101998,P61,30)`
消息体描述：
`7570101998：设备ID`
`P61：消息类型`
`30：当前阀值`

P62:设置或查询里程统计相关参数

消息体内容：（示例）
`(7570101998,P62,1,10)`
消息体描述：
`7570101998：设备ID`
`P62：消息类型`
`1：操作参数类型;1:设置一个速度值，低于这个速度里程不会统计；2：同步当前设备里程值`

10：当操作参数类型为1，此时为速度值，单位km/h；当操作参数类型为2，此时为里程值，单位km

P63:静态飘移处理功能设置

消息体内容：（示例）
(7570101998,P63,1)
消息体描述：
7570101998：设备ID
P63：消息类型
1：1开启，0关闭(默认)

P68:查询SIM卡的IMSI/ICCID号

消息体内容：（示例）
(7570101998,P65,2,898600220909A0206023)
消息体描述：
7570101998：设备ID
P65：消息类型
2：1：查询IMSI，2:查询ICCID
898600220909A0206023：SIM卡的IMSI/ICCID号

P70:启用/关闭VIP号码功能

消息体内容：（示例）
(7570101998,P70,1)
消息体描述：
7570101998：设备ID
P70：消息类型
1：0：关闭状态；1：开启状态

WLNET1:查询/设置要监听的从机设备ID号

消息体内容：（示例）
(700160818000,1,001,WLNET,1,20,0217270000,,,,,,,,)
消息体描述：
700160818000：设备ID
WLNET,1：组合消息类型WLNET1
20：表示当前配置20个从机设备ID号,如果为配置0个ID号表示清除所有配置的ID号
0217270000,,,,,,,,：从机ID号

WLNET 2 :查询/设置从机工作时间间隔

消息体内容：（示例）

(700160818000,1,001,WLNET,2,20)

消息体描述：

700160818000：设备ID

WLNET,2：组合消息类型WLNET2

20：20：从机工作时间间隔为20分钟，单位分钟，最小1分钟，最大1440一天，默认5分钟

WLNET3:查询/设置从机发送功率

消息体内容：（示例）

(700160818000,1,001,WLNET,3,2)

消息体描述：

700160818000：设备ID

WLNET,3：组合消息类型WLNET3

2：从机从机发送功率，1~3分别是低中高三个发送功率模式，默认是1低功率

WLNET4:查询/设置从机发送功率

消息体内容：（示例）

(700160818000,1,001,WLNET,4,170828,170829)

消息体描述：

700160818000：设备ID

WLNET,4：组合消息类型WLNET4

170828:网关版本

170829:传感器的软件版本

WLNET6:查询/设置从机温度门限参数

消息体内容：（示例）

(700160818000,1,001,WLNET,6,1,2,10,120,30,15,12)

消息体描述：

700160818000：设备ID

WLNET,6：组合消息类型WLNET6

1:指令功能：0表示查询;1:表示设置

2：参数类型：1表示温度相关参数，2表示设置数据上传方式

10：低温门限值：默认0值无效，温度以1度为单位10~175有效，-40～125度，门限值 - 50 = 实际值，如10- 50 = -40℃。

120：高温门限值：默认0值无效，温度以1度为单位10~175有效，-40～125度，门限值 - 50 = 实际值，如120- 50 = 70℃

30：温度变化时间值：多长时间内温度变化多少的时间值，单位分钟，默认0值无效，一般20~60参考值，最大240，4小时

15：温度变化值：默认0值无效，15表示1.5度，最大250、25度，也就是如30分钟内变化1.5度就报警

12：每天定点上传多少个点：默认0值当前一个点，12表示每天12个点2小时一个点，最多24个点，一般12或24个点参考值

Java 版本

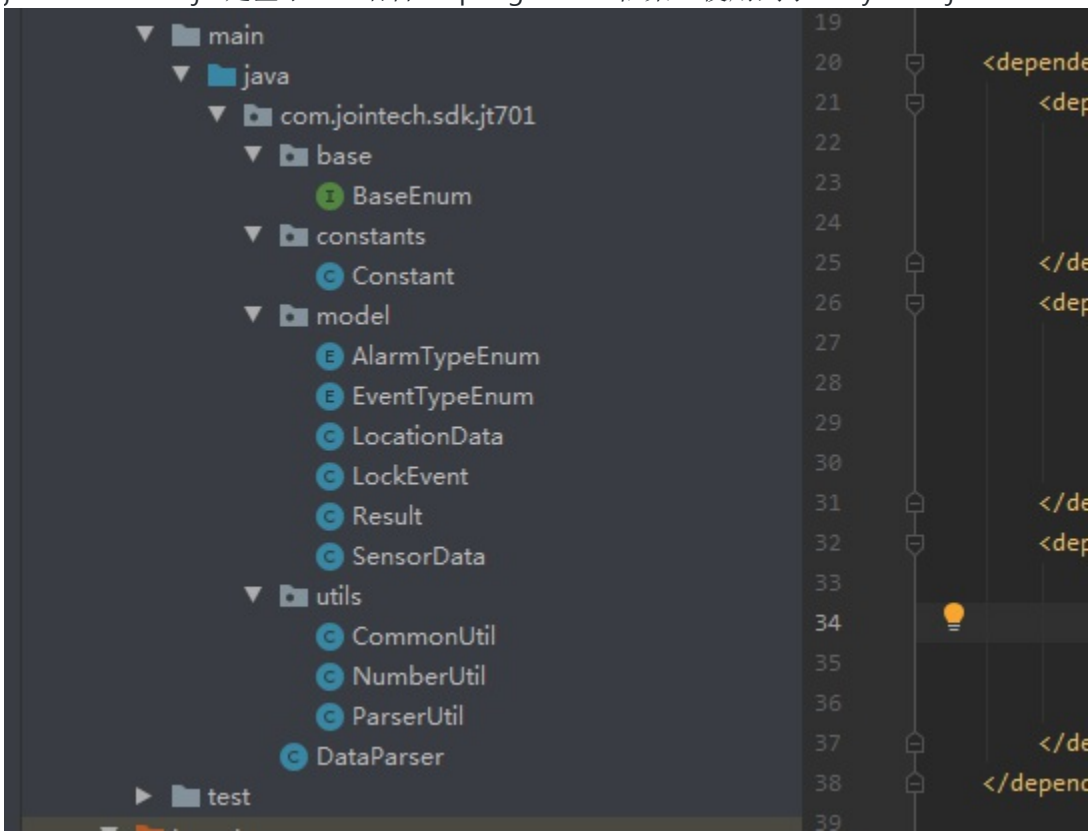
1.Jar包下载

jt701-sdk-1.0.0.jar 下载地址
如果需要Jar包开发源码，请与商务申请

2.集成开发说明

2.1.集成开发语言及框架说明

jt701-sdk-1.0.0.jar是基于Java语言，SpringBoot2.x框架，使用到了netty，fastjson，lombok



BaseEnum：基础枚举
Constant：自定义常量
AlarmTypeEnum：报警枚举
EventTypeEnum：事件枚举
LocationData：定位实体类
LockEvent：锁事件实体类
Result：结果实体类
SensorData：从机数据实体类
CommonUtil：公共方法类

NumberUtil：数字操作工具类
ParserUtil：解析方法工具类
DataParser：解析主方法

2.2.集成说明

将jt701-sdk-1.0.0.jar引入到自己的网关程序中，引入方法如下：
在pom.xml引入jar包

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

调用jt701-sdk-1.0.0.jar，DataParser类中receiveData()方法
receiveData()方法是重载方法

```
    /**
     * 解析Hex字符串原始数据
     * @param strData 16进制字符串
     * @return
     */
    public static Object receiveData(String strData)
    {
        ByteBuf msgBodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(strData.le
ngth()/2);
        msgBodyBuf.writeBytes(CommonUtil.hexStr2Byte(strData));
        return receiveData(msgBodyBuf);
    }

    /**
     * 解析byte[]原始数据
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =ByteBufAllocator.DEFAULT.heapBuffer(bytes.lengt
h);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
```

```
     * 解析ByteBuf原始数据
     * @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.核心代码

解析方法工具类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;

/**
 * <p>Description: 解析方法工具类</p>
```

```
 * @author HyoJung
 * @date 20210526
 */
public class ParserUtil {
    private ParserUtil()
    {}

    /**
     * 解析指令数据
     * @param in 原始数据
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in)
    {
        //定义定位数据实体类
        Result model = new Result();
        //包头(
        in.readByte();
        //字段列表
        List<String> itemList = new ArrayList<String>();
        //透传的二进制数据
        ByteBuf msgBody = null;
        while (in.readableBytes() > 0) {
            //无线网关数据上传(WLNET5、WLNET7)第7个字段以及后面为二进制数据
            if (itemList.size() >= 6 && Objects.equals("WLNET", itemList.ge
t(3)) && Constant.WLNET_TYPE_LIST.contains(itemList.get(4))) {
                //到结尾")"前的长度
                int lastItemLen = in.readableBytes() - 1;
                //反转义
                msgBody = Unpooled.buffer(lastItemLen);
                CommonUtil.unescape(in, msgBody, lastItemLen);
                in.readByte();
            } else {
                //查询逗号的下标截取数据
                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消息类型为组合
        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;
    }

    /**
     * 解析从机数据
     * @param byteBuf
     * @return
     */
    private static SensorData parseWlnet5(ByteBuf byteBuf) {
        SensorData sensorData = new SensorData();
        //定位时间
        byte[] timeArr = new byte[6];
        byteBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = parseBcdTime(bcdTimeStr);
        //纬度
        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();
        //经度
        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();
        //位指示
        int bitFlag = Byte.parseByte(lngHexStr.substring(9, 10), 16);
        //定位状态
        int locationType = (bitFlag & 0x01) > 0 ? 1 : 0;
        //北纬、南纬
        if ((bitFlag & 0b0010) == 0) {
            lat = -lat;
        }
        //东经、西经
        if ((bitFlag & 0b0100) == 0) {
            lng = -lng;
        }
        //速度
        int speed = (int) (byteBuf.readUnsignedByte() * 1.85);
        //方向
        int direction = byteBuf.readUnsignedByte() * 2;

        //从机时间
        byte[] slaveMachineTimeArr = new byte[6];
        byteBuf.readBytes(slaveMachineTimeArr);
        String slaveMachineBcdTimeStr = ByteBufUtil.hexDump(slaveMachineTim
eArr);
        ZonedDateTime slaveMachineZonedDateTime = parseBcdTime(slaveMachine
BcdTimeStr);
        //从机ID
        byte[] slaveMachineIdArr = new byte[5];
        byteBuf.readBytes(slaveMachineIdArr);
        String slaveMachineId = ByteBufUtil.hexDump(slaveMachineIdArr).toUp
perCase();
        //从机数据流水号
        int flowId = byteBuf.readUnsignedByte();
        //从机电压
        String voltage = new BigDecimal(byteBuf.readUnsignedShort()).divide
(new BigDecimal("100"), 2, RoundingMode.HALF_UP).toString();
        //从机电量
```

```
            int power = byteBuf.readUnsignedByte();
            //RSSI
            int rssi = byteBuf.readUnsignedByte();
            //传感器类型
            int sensorType = byteBuf.readUnsignedByte();
            //温度值
            double temperature = -1000.0;
            //湿度值
            int humidity = 0;
            //事件类型
            int eventType = -1;
            //设备状态
            int terminalStatus = -1;
            //开关锁次数
            int lockTimes = -1;
            if (sensorType == 1) {
                //温度
                temperature = parseTemperature(byteBuf.readShort());
                //湿度
                humidity = byteBuf.readUnsignedByte();
                //网关保存数据条数
                int itemCount = byteBuf.readUnsignedShort();
                //网关状态
                int gatewayStatus = byteBuf.readUnsignedByte();
            } else if (sensorType == 4) {
                //事件
                int event = byteBuf.readUnsignedShort();
                //判断事件
                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());
            }
            //设备状态
            terminalStatus = byteBuf.readUnsignedShort();
            //开关锁次数
            lockTimes = byteBuf.readUnsignedShort();
            //网关状态
            int gatewayStatus = byteBuf.readUnsignedByte();
        }else if (sensorType == 5||sensorType == 6)
        {
            //事件
            int event = byteBuf.readUnsignedShort();
            //判断事件
            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());
            }
            //设备状态
            terminalStatus = byteBuf.readUnsignedShort();
            //开关锁次数
            lockTimes = byteBuf.readUnsignedShort();
            //网关状态
            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;
        }

        /**
         * 解析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));
            //开锁验证
            int status= Integer.valueOf(itemList.get(12));
            model.RFIDNo = itemList.get(13);
            //动态密码开锁
            if (model.Event == 6)
            {
                if (status == 0)
                {
                    //开锁密码不正确
                    model.Status = 0;
                }
                else if (status > 0 && status <= 10)
                {
                    //正常开锁
                    model.Status = 1;
                    //围栏内开锁时候的围栏ID
                    model.UnlockFenceID = status;
```

```
            }
            else if (status == 98)
            {
                //正常开锁
                model.Status = 1;
            }
            else if (status == 99)
            {
                //设备开启了围栏内开锁，且当前开锁并未在围栏内，拒绝开锁
                model.Status = 3;
            }
        }
        else if (model.Event == 4)
        {
            if (Integer.valueOf(itemList.get(14)) == 0)
            {
                //开锁密码不正确
                model.Status = 0;
            }
            else
            {
                //正常开锁
                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;
    }

    /**
     * 解析从机数据温度
     *
     * @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;
    }

    /**
     * 转换GPS时间
     *
     * @param bcdTimeStr
     * @return
     */
    public static ZonedDateTime parseBcdTime(String bcdTimeStr) {
        if(bcdTimeStr.equals("000000000000"))
        {
            //默认给出时间为2000年1月1日 00时00分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;
    }

    /**
     * 解析定位数据
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in)
    {
        //协议头
        in.readByte();
        //终端号码
        byte[] terminalNumArr = new byte[5];
        in.readBytes(terminalNumArr);
        String terminalNum = ByteBufUtil.hexDump(terminalNumArr);
        //协议版本号
        int version = in.readUnsignedByte();
        short tempByte = in.readUnsignedByte();
        //终端类型号
        int terminalType = tempByte >> 4;
        //数据类型号
        int dataType = tempByte & 0b00001111;
```

```
        //数据长度
        int dataLen = in.readUnsignedShort();
        //定位时间
        byte[] timeArr = new byte[6];
        in.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = parseBcdTime(bcdTimeStr);
        //纬度
        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();
        //经度
        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();
        //位指示
        int bitFlag = Byte.parseByte(lngHexStr.substring(9, 10), 16);
        //定位状态
        int locationType = (bitFlag & 0x01) > 0 ? 1 : 0;
        //北纬、南纬
        if ((bitFlag & 0b0010) == 0) {
            lat = -lat;
        }
        //东经、西经
        if ((bitFlag & 0b0100) == 0) {
            lng = -lng;
        }
        //速度
        int speed = (int) (in.readUnsignedByte() * 1.85);
        //方向
        int direction = in.readUnsignedByte() * 2;
        //里程
        long mileage = in.readUnsignedInt();
        //GPS卫星个数
```

```
        int gpsSignal = in.readByte();
        //绑定车辆ID
        long vehicleId = in.readUnsignedInt();
        //终端状态
        int terminalStatus = in.readUnsignedShort();
        //是否基站定位
        if (NumberUtil.getBitValue(terminalStatus, 0) == 1) {
            locationType = 2;
        }
        //电量指示
        int batteryPercent = in.readUnsignedByte();
        //2G CELL ID
        int cellId2G = in.readUnsignedShort();
        //LAC
        int lac = in.readUnsignedShort();
        //GSM信号质量
        int cellSignal = in.readUnsignedByte();
        //区域报警ID
        int regionAlarmId = in.readUnsignedByte();
        //设备状态3
        int terminalStatus3 = in.readUnsignedByte();
        //唤醒源
        int fWakeSource=(terminalStatus3 & 0b0000_1111);
        //预留
        in.readShort();
        //IMEI号
        byte[] imeiArr = new byte[8];
        in.readBytes(imeiArr);
        String imei = ByteBufUtil.hexDump(imeiArr);
        //3G CELL ID 高16位
        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();
        //流水号
        int flowId = in.readUnsignedByte();
        //解析报警
        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);
        //定义定位数据实体类
        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;
    }

    /**
     * 解析定位报警
     * @param terminalStatus
     * @return
```

```
     */
    private static int parseLocationAlarm(int terminalStatus)
    {
        //是否报警
        int fAlarm = -1;
        //是否应答确认
        if (NumberUtil.getBitValue(terminalStatus, 5) == 1) {
            //判断报警
            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;
    }

    /**
```

```
     * 指令应答回复
     * @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")) {
                    //密码重置
                    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;
    }

    /**
     * 指令应答回复
     * @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;
    }
}
```

公共方法类CommonUtil

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

import io.netty.buffer.ByteBuf;

import java.nio.ByteBuffer;

/**
 * <p>Description: 用来存储一些解析中遇到的公共方法</p>
 *
 * @author Lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    private CommonUtil()
    {

    }

    /**
     * 反转义文本透传数据
     *
     * @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++;
            }
        }
    }

    /**
     * 去掉字符串最后一个字符
     * @param inStr 输入的字符串
     * @param suffix 需要去掉的字符
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * 16进制转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();
        }
    }
```

解析常量Constant

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

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

/**
 * 常量定义
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * 二进制消息包头
     */
    public static final byte BINARY_MSG_HEADER = '$';

    /**
     * 文本消息包头
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * 文本消息包尾
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * 文本消息分隔符
     */
    public static final byte TEXT_MSG_SPLITER = ',';
```

```
    /**
     * 透传二进制数据的指令
     */
    public static final List<String> WLNET_TYPE_LIST = Arrays.asList("5", "
7");
}
```

2.4.返回消息及说明

（1）定位数据

原始数据：

```
2480405002251911003426032118530329532416031008941d0000000018070c0000000020e
04f8b0c56001f00020f0f0f0f0f0f0f0f0f0f00f2028f0157
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID"：设备ID
    "MsgType"：消息类型， 此处为： Location,表示定位数据,
    "DataBody"：消息体内容
    {
        "ProtocolType": 协议版本号,
        "DeviceType": 终端类型号,
        "DataType": 数据类型号（ 1 表明最新二进制定位数据， 2 表示报警数据， 3
表示盲区常规二进制定位数据， 4 表示次新二进制定位数据）,
        "DataLength": 数据长度,
        "GpsTime": 定位时间（ GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位）,
        "Speed": 速度（ 单位： km / h）,
        "Direction": 方向（ 0~360； 0 与360表示正北方向）,
        "Mileage": 当前里程值（单位：km）,
        "GpsSignal": GPS卫星个数,
        "GSMSignal": GSM信号值,
        "Alarm": 报警类型（ - 1： 无告警信息； 1： 锁绳剪断； 2： 震动； 3： 长
时间开锁； 4： 开锁密码连续5次错误； 5： 刷非法卡； 6： 低电量； 7： 开后盖； 8：
 卡锁； 9： 进区域报警； 10： 出区域报警）,
        "AlarmArea": 如果告警与区域有关， 则此处值为区域的ID,
        "Battery": 电量值（ 0~100； 255： 充电中）,
        "LockStatus": 锁电机状态（ 1： 开； 0： 关）,
        "LockRope": 锁绳状态（ 1： 拔出； 0： 插入）,
        "BackCover": 后盖状态（ 1： 关闭； 0： 开启）,
        "MCC": 国家代码,
        "MNC": 运营商代码,
        "LAC": 位置区域码,
        "CELLID": 基站编号,
        "IMEI": IMEI号码， 全是0F无效,
        "Awaken": 唤醒源（ 0 重启,
        1: RTC唤醒， 2: 震动， 3: 开后盖， 4: 锁绳， 5: 接外电， 6: 刷卡， 7:
门磁， 8: VIP短信， 9: 非VIP短信或垃圾短信）,
        "Index": 数据流水号
```

```
    },
    "ReplyMsg"：回复内容（ 如果为空字符串， 则不需要给终端回复内容）
}
```

（2）传感器采集数据（WLNET5）

原始数据(SensorType=1)：

```
28383035303530303037332c312c3134312c574c4e45542c352c322c26032118470964967295394967 3d1408ff00260321184718102011098666013362 3c0100d71b00000029
```

返回消息(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"
}
```

原始数据(SensorType=4)：

```
28373030303331333330392C312C3038312C574C4E45542C352C322C05082115430722348250113550300F0000050821154304E0171E086925018D4E690400400000002A0029
```

返回消息(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)"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型， 此处为： WLNET5,表示传感器透传数据,
    "DataBody": 消息体内容
    {
        "GpsTime": 定位时间（GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位）,
        "Speed": 速度（ 单位： km / h）；0xFF表示速度无效,
        "Direction": 方向（ 0~360； 0 与360表示正北方向）,
        "SensorID": 传感器ID,
        "LockStatus": 如果从机类型SensorType=4，此是才有效；1：开锁；0：关锁,
        "LockRope": 如果从机类型SensorType=4，此是才有效；1：锁绳拔出；0：锁绳
```

```
插入，
        "LockTimes": 开锁次数（如果SensorType=4此值才有效），
        "Index": 数据流水号，
        "Voltage": 电压值（单位：V），
        "Power": 传感器电量（0~100；255： 充电中），
        "RSSI": RSSI信号强度，是负数越接近0信号越好，
        "DateTime": 数据采集时间（GMT时间），
        "SensorType": 传感器类型（1：温湿度传感器(JT126)；4:从机JT709； ），
        "Temperature":温度值（如果SensorType=1此值才有效），
        "Humidity": 湿度值（如果SensorType=1此值才有效），
        "Event": 事件类型（-1：无从机事件；0：关锁事件；1：蓝牙开锁事件；2：NFC
开锁事件；3：Lora开锁事件；4：从机锁剪断报警事件；5：按键唤醒事件；6：定时上报事件
；7：充电上报事件）
    },
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

（3）锁事件上报数据（P45）

原始数据：

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

返回消息：

```
{
    "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)"
```

```
}
```

返回消息描述：

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型， 此处为： P45,表示开关锁事件数据上传,
    "DataBody": 消息体内容
    {
        "DateTime": 事件时间（GMT时间）,
        "Latitude": 纬度(dd.dddd格式),
        "Longitude": 经度(dd.dddd格式),
        "LocationType": 定位类型（ 0： 不定位； 1： GPS定位； 2： 基站定位）,
        "Speed": 速度（ 单位： km / h）,
        "Direction": 方向（ 0~360； 0 与360表示正北方向）,
        "Event": 事件类型（1：表示刷授权卡；2：表示刷非法卡；3：表示刷车辆ID卡绑定；4：表示为凭密码开锁；5：表示终端自动关锁记录；6: 动态密码围栏内开锁；7: 蓝牙开锁）,
        "Status": 开锁验证（0：开锁密码不正确；1：正常开锁；2：因为开启了围栏开锁，未在围栏内开锁，开锁被拒绝）,
        "UnlockFenceID": （-1：开锁与围栏无关；1~10：标识对应的开锁围栏ID）,
        "RFIDNo": 刷卡卡号；如果未“0000000000”，则无效,
        "PsdErrorTimes": 开锁密码错误次数,
        "Index": 数据流水号,
        "Mileage": 当前里程值（单位：km）
    },
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

（4）其他指令回复数据
原始数据：

```
28373839303632393238342c50333529
```

返回消息：

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

返回消息描述：

```
{
    "DeviceID": 设备ID,
    "MsgType": 消息类型（参见更多的消息类型及其描述，请参阅3.消息类型及消息体内容描述）,
    "DataBody": 消息体内容（除定位数据：Location；传感器透传数据：WLNET5;锁事件上报：P45外；其他此处均直接返回指令内容的ASCII字符串）,
    "ReplyMsg": 回复内容（如果为空字符串，则不需要给终端回复内容）
}
```

3.消息类型及消息体内容描述

3.1.P01:查询终端当前的版本号

消息体内容：（示例）

`(7591225008,P01,JT701D_20200720_China_Jointech_SIM7600,77%)`

消息体内容描述：

`7591225008：设备ID`

`P01：消息类型`

`JT701D_20200720_China_Jointech_SIM7600：协议版本`

`77%：当前设备电量`

P03:低电休眠控制

消息体内容：（示例）

`(7560704001,P03,1,30)`

消息体描述：

`7560704001：设备ID`

`P03：消息类型`

`1：生效；0：不生效`

`30：设置电量低于30的时候进入休眠，默认31%，可设定范围5%~90%`

P04：设置/查询数据上传间隔和休眠自动唤醒间隔

消息体内容：（示例）

`(7570101998,P04,30,30)`

消息体描述：

`7570101998：设备ID`

`P04：消息类型`

`30：数据上传间隔，单位秒钟，默认30秒，取值范围5-600`

`30：休眠自动唤醒间隔，单位分钟，默认30分钟，取值范围30-1440`

P06:设置/查询监控中心IP与端口、APN

消息体内容：（示例）

`(7570101998,P06,211.162.111.225,10906,CMNET,user,password,1)`

消息体描述：

`7570101998：设备ID`

`P06：消息类型`

`211.162.111.225：监控中心的IP地址`

`10906：监控中心端口地址，最大65530`

`CMNET：接入点名称(最长50个字节)`

`User：APN用户名(最长50个字节)`

`Password：APN密码(最长50个字节)`

`1：0表示卡1，1表示卡2`

P10:设置/查询终端使用地点与国际标准时间的时差

消息体内容：（示例）

`(7570101998,P10,480)`

消息体描述：

`7570101998：设备ID`

`P10：消息类型`

`480：时差值,以分钟为单位.如北京时间与标准时时差为8小时,即为480分钟，取值范围-12*60-13*60，默认0`

P11:设置/查询VIP手机号码

消息体内容：（示例）

`(7570101998,P11,1,8613910102345)`

消息体描述：

`7570101998：设备ID`

`P11：消息类型`

`1：VIP手机号码索引,取值为1-5,允许有五组VIP手机号码`

`8613910102345：手机号码,不能超过15位数字，前面需加国际区号，中国为86或者+86.`

P12:设置/查询VIP号码是否允许报警

消息体内容：（示例）

`(7570101998,P12,1,1,1,1,1)`

消息体描述：

`7570101998：设备ID`

`P12：消息类型`

`1,1,1,1,1:分别对应5个VIP号码是否允许报警；1表示允许对应的VIP号码报警，0表示不允许此VIP号码报警`

P13:恢复出厂设置

消息体内容：（示例）

```
(7570101998,P13)
```

消息体描述：

```
7570101998：设备ID
P13：消息类型
```

P14:读取终端的IMEI号

消息体内容：（示例）

```
(7570101998,P14,012207004451636)
```

消息体描述：

```
7570101998：设备ID
P14：消息类型
012207004451636：终端的IMEI号
```

P15:终端重启指令

消息体内容：（示例）

```
(7570101998,P15)
```

消息体描述：

```
7570101998：设备ID
P15：消息类型
```

P22:GPS无效的时候，监控中心对终端授时

消息体内容：（示例）

```
(7570101998,P22,1)
```

消息体描述：

```
7570101998：设备ID
P22：消息类型
1：1表示授时成功，0表示失败，2表示主动请求授时
```

P22:设置/取消短信、电话可唤醒工作模式

消息体内容：（示例）

```
(7570101998,P23,1)
```

消息体描述：

```
7570101998：设备ID
P23：消息类型
1：1表示设置成功，0表示设置失败.
```

P24:区域是否有效，及区域名称设置指令

消息体内容：（示例）

`(7570101998,P24,10,1,area10)`

消息体描述：

`7570101998：设备ID`

`P24：消息类型`

`10：表示第10个区域.`

`1：表示有效 ，0表示无效`

`area10：表示区域名称，最大长度为16个字节.`

P29:设置或者查询区域的详细节点信息

消息体内容：（示例）

`(7570101998,P29,8,15,1,10,11323.1234…)`

消息体描述：

`7570101998：设备ID`

`P29：消息类型`

`8:表示第八个区域`

`15:表示总点数`

`1:表示当前页`

`10:表示当前页的点数.余下的为各个点的经度与纬度`

P30:清除相关的区域

消息体内容：（示例）

`(7570101998,P30,1)`

消息体描述：

`7570101998：设备ID`

`P30：消息类型`

`1:1表示清除成功，0表示清除失败.`

P31:区域信息设置完毕

消息体内容：（示例）

`(7570101998,P31)`

消息体描述：

`7570101998：设备ID`

`P31：消息类型`

P32:主动进入休眠指令

消息体内容：（示例）

`(7570101998,P32)`

消息体描述：

`7570101998：设备ID`

`P32：消息类型`

P37:查询/设置G-sensor相关参数

消息体内容：（示例）

`(7570101998,P37,500)`

消息体描述：

`7570101998：设备ID`

`P37：消息类型`

`500：运动检测门限值，范围是63到500,单位是mg；如果设置为0则为关闭G-sensor相关全部功能；关闭G-sensor功能后，如需重新开启G-sensor功能只需重新设置有效的G-sensor参数即可.默认值126`

P38:开锁报警时间间隔设置指令

消息体内容：（示例）

`(7570101998,P38,120)`

消息体描述：

`7570101998：设备ID`

`P38：消息类型`

`120：开锁报警时间间隔设置，即从锁电机处于开锁状态时刻算起，如果该状态持续保持超过120分钟，则可触发开锁报警，范围是3到180,单位是分钟；默认为120分钟.`

P40:查询/设置GPRS通道和短消息通道的报警开关

消息体内容：（示例）

`(7570101998,P40,1,1,1,1,1,1,1,1,1,1,1)`

消息体描述：

`7570101998：设备ID`

`P40：消息类型`

`1,1,1,1,1,1,1,1,1,1,1：从左至右依次为锁挂绳剪断报警、刷非法卡报警、开锁状态保持一段时间报警、指令开锁密码连续输错5次报警、震动报警、进区域报警、出区域报警的开关、低电报警、开后盖报警、卡锁报警；每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.`

P41:增删开锁授权号指令

消息体内容：（示例1）

`(7570101998,P41,1,30)`

消息体描述：

`7570101998：设备ID`

`P41：消息类型`

`1：1表示增加授权卡号操作;2表示删除授权号，3表示删除所有的授权号`

`30：30表示当前已存授权卡号总个数;`

消息体内容：（示例2）

`(7570101998,P41,2,3,0013953759, 0013953758, 0013953757)`

消息体描述：
`7570101998：设备ID`
`P41：消息类型`
`2：查询第2组数据；一共分3组，分别为1~3，查询时每次最多返回20个ID号`
`3：3表示有3个ID号`
`0013953759，0013953758，0013953757表示该组存储的授权号列表`

P42:现场刷卡授权模式配置指令

消息体内容：（示例1）
`(7570101998,P42,0)`
消息体描述：
`7570101998：设备ID`
`P42：消息类型`
`1：1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。`

消息体内容：（示例2）
`(7570101998,P41,2,0013953759,0013953751)`
消息体描述：
`7570101998：设备ID`
`P42：消息类型`
`2：表示终端已存了2个开锁授权号.`
`0013953759,0013953751：授权卡号.`

P44:远程开锁密码修改指令

消息体内容：（示例1）
`(7570101998,P44,1)`
消息体描述：
`7570101998：设备ID`
`P44：消息类型`
`1：1：表示修改密码是否成功，1表示成功，0表示失败。`

消息体内容：（示例2）
`(7570101998,P44,888888)`
消息体描述：
`7570101998：设备ID`
`P44：消息类型`
`888888：当前的设备可开锁的动态密码`

P50:电源开关生效控制设置

消息体内容：（示例）
`(7570101998,P50,1)`
消息体描述：

7570101998：设备ID
P44：消息类型
1：1表示开关有效，默认是1；0表示停用开关.

P52:动态密码指令

消息体内容：（示例1）
(7570101998,P52,0,405935,326387)
消息体描述：
7570101998：设备ID
P52：消息类型
0：指令操作；查询当前产生的动态密码和当前用来开锁的动态密码
405935：表示，还没有被系统确认的动态密码
326387：表示目前用来开锁的动态密码

消息体内容：（示例2）
(7570101998,P52,1,1,0)
消息体描述：
7570101998：设备ID
P52：消息类型
1：1表示设置动态密码功能
1：1查询是否开启动态密码功能；0：表示关闭动态密码功能
0：1表示动态密码开锁必须在区域内才能开锁，即表明如果想要开锁，必须设置区域，0表示动态密码开锁跟区域无关，只要符合动态密码开锁的其它条件就可以开锁

消息体内容：（示例3）
(7570101998,P52,2,405935)
消息体描述：
7570101998：设备ID
P52：消息类型
2：回复上传的动态密码确认指令
405935：需要被确认的动态密码

消息体内容：（示例4）
(7570101998,P52,3,1,0)
消息体描述：
7570101998：设备ID
P52：消息类型
3：发送动态密码开锁
1：1,代表成功，0是失败
0：代表失败次数

P54:查询/设置是否关闭休眠模式指令(设备不休眠)

消息体内容：（示例）

`(7570101998,P54,0,0)`
消息体描述：
`7570101998：设备ID`
`P54：消息类型`
`0：0 查询；1 设置`
`0：0 需要休眠 ， 1 开启不休眠`

P58:查询和设置RFID卡是否关联电子围栏 (默认关联电子围栏)

消息体内容：（示例）
`(7570101998,P58,1,1)`
消息体描述：
`7570101998：设备ID`
`P58：消息类型`
`1：1表示设置，0表示查询`
`1：1：关联电子围栏，必须在区域内才能刷卡开锁；0表示刷卡开锁跟区域无关，只要符刷卡开锁的条件就可以开锁`

P61:设置或查询电量低报警提示的阀值

消息体内容：（示例）
`(7570101998,P61,30)`
消息体描述：
`7570101998：设备ID`
`P61：消息类型`
`30：当前阀值`

P62:设置或查询里程统计相关参数

消息体内容：（示例）
`(7570101998,P62,1,10)`
消息体描述：
`7570101998：设备ID`
`P62：消息类型`
`1：操作参数类型;1:设置一个速度值，低于这个速度里程不会统计；2：同步当前设备里程值`
`10：当操作参数类型为1，此时为速度值，单位km/h；当操作参数类型为2，此时为里程值，单位km`

P63:静态飘移处理功能设置

消息体内容：（示例）
`(7570101998,P63,1)`
消息体描述：
`7570101998：设备ID`

P63：消息类型
1：1开启，0关闭(默认)

P68:查询SIM卡的IMSI/ICCID号

消息体内容：（示例）
(7570101998,P65,2,898600220909A0206023)
消息体描述：
7570101998：设备ID
P65：消息类型
2：1：查询IMSI，2:查询ICCID
898600220909A0206023：SIM卡的IMSI/ICCID号

P70:启用/关闭VIP号码功能

消息体内容：（示例）
(7570101998,P70,1)
消息体描述：
7570101998：设备ID
P70：消息类型
1：0：关闭状态；1：开启状态

WLNET1:查询/设置要监听的从机设备ID号

消息体内容：（示例）
(700160818000,1,001,WLNET,1,20,0217270000,,,,,,,,)
消息体描述：
700160818000：设备ID
WLNET,1：组合消息类型WLNET1
20：表示当前配置20个从机设备ID号,如果为配置0个ID号表示清除所有配置的ID号
0217270000,,,,,,,,：从机ID号

WLNET 2 :查询/设置从机工作时间间隔

消息体内容：（示例）
(700160818000,1,001,WLNET,2,20)
消息体描述：
700160818000：设备ID
WLNET,2：组合消息类型WLNET2
20：20：从机工作时间间隔为20分钟，单位分钟，最小1分钟，最大1440一天，默认5分钟

WLNET3:查询/设置从机发送功率

消息体内容：（示例）

(700160818000,1,001,WLNET,3,2)

消息体描述：

700160818000：设备ID

WLNET,3：组合消息类型WLNET3

2：从机从机发送功率，1~3分别是低中高三个发送功率模式，默认是1低功率

WLNET4:查询/设置从机发送功率

消息体内容：（示例）

(700160818000,1,001,WLNET,4,170828,170829)

消息体描述：

700160818000：设备ID

WLNET,4：组合消息类型WLNET4

170828:网关版本

170829:传感器的软件版本

WLNET6:查询/设置从机温度门限参数

消息体内容：（示例）

(700160818000,1,001,WLNET,6,1,2,10,120,30,15,12)

消息体描述：

700160818000：设备ID

WLNET,6：组合消息类型WLNET6

1:指令功能：0表示查询;1:表示设置

2：参数类型：1表示温度相关参数，2表示设置数据上传方式

10：低温门限值：默认0值无效，温度以1度为单位10~175有效，-40～125度，门限值 - 50 = 实际值，如10- 50 = -40℃。

120：高温门限值：默认0值无效，温度以1度为单位10~175有效，-40～125度，门限值 - 50 = 实际值，如120- 50 = 70℃

30：温度变化时间值：多长时间内温度变化多少的时间值，单位分钟，默认0值无效，一般20~60参考值，最大240，4小时

15：温度变化值：默认0值无效，15表示1.5度，最大250、25度，也就是如30分钟内变化1.5度就报警

12：每天定点上传多少个点：默认0值当前一个点，12表示每天12个点2小时一个点，最多24个点，一般12或24个点参考值

JT704&JT706 SDK集成开发

.NET版本

Java版本

.NET 版本

1.SDK下载

JT704SDK(JT704SDK_V2.rar) 下载地址
JT704SDK测试工具(JT704SDK_Test.rar) 下载地址
如果需要测试工具及SDK开发源码，请与商务申请

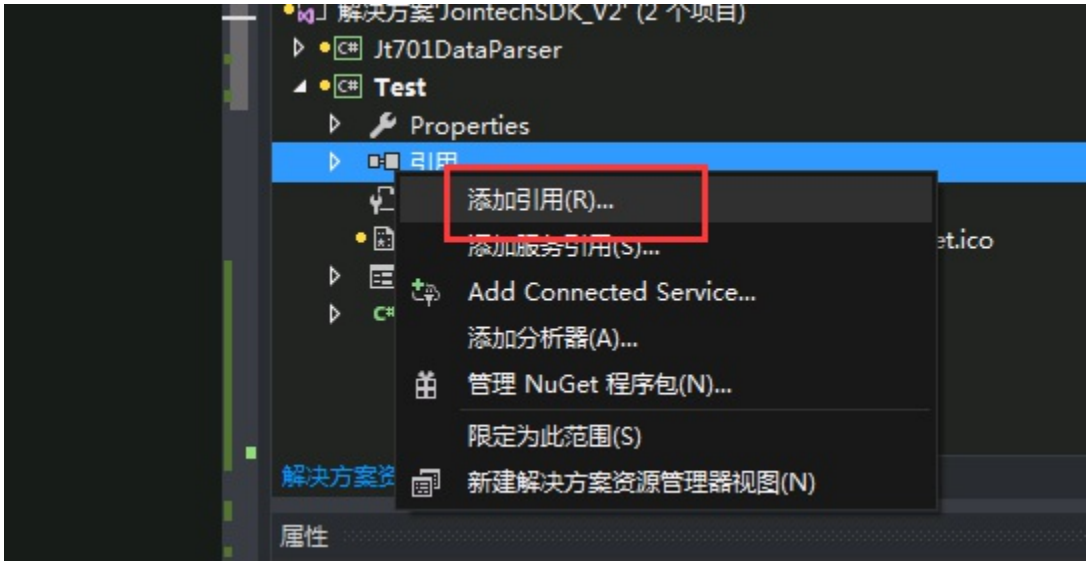
2.集成开发说明

2.1.集成开发语言及框架说明

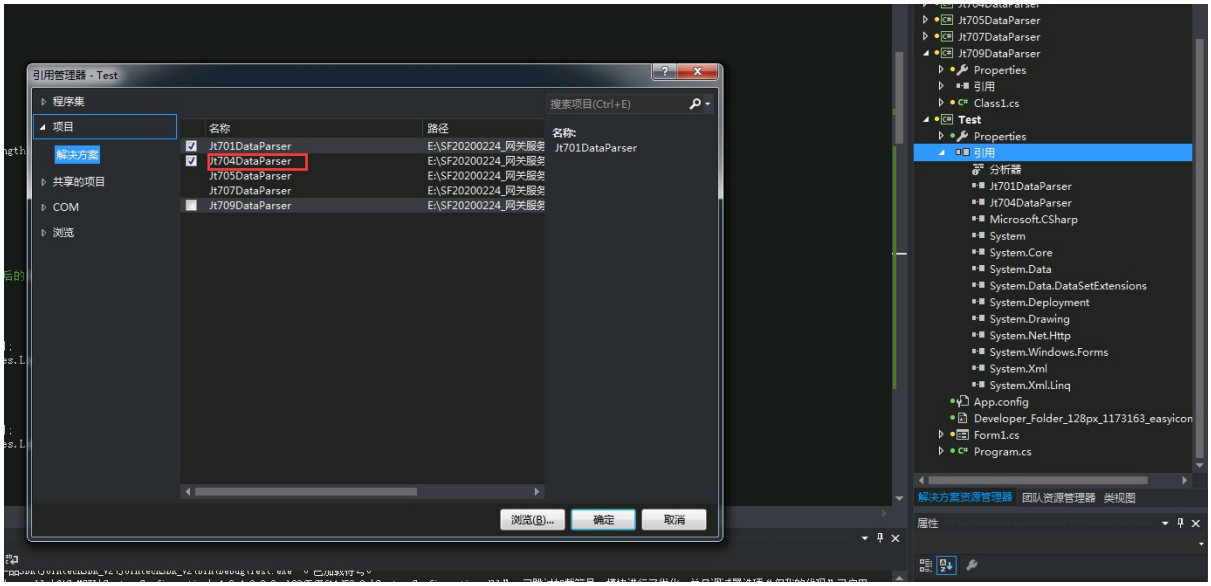
JT704SDK（Jt704DataParser.dll）及测试工具（Test.exe）是基于C#语言，.NET Framework 4.6.1目标框架开发的。该SDK的集成开发环境也是需要依赖于C#语言以及.NET Framework 4.6.1目标框架。

2.2.集成说明

（1）将Jt704DataParser.dll引入到自己的网关程序中，引入方法如下：
鼠标右键项目下的"引用"，选择"添加引用(R)..."



选择下图中"浏览(B)..."，找到你解压JT704SDK20210806.rar后的文件夹，选中Jt704DataParser.dll与对应的Json字符串序列化/反序列化包Newtonsoft.Json.dll即可完成对SDK的引用



（2）调用Jt704DataParser.dll中的解析方法receiveData

receiveData()是一个重载方法，你可以传入16进制字符串或者byte[],一般我们网关程序接收到的设备数据是以二进制流进行传输的，所以我们这里建议使用此重载方法receiveData(byte[] bytes);

当然如果你想通过16进制字符串进行测试，也可以使用receiveData(string strData)；

strData：就是我们接收到的设备数据转成16进制后的字符串

2.3.核心代码

Jt704DataParser.dll

(1)解析类DataParser.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt704DataParser
{
    public class DataParser
    {
        private static byte[] endbytes = null;// 上一次未处理的剩余字节

        /// <summary>
        /// 解析16进制原始数据
        /// </summary>
        /// <param name="strData"></param>
        /// <returns></returns>
        public static string receiveData(string strData)
        {
```

```
            byte[] bytes = Common.HexStringToBytes(strData);
            return receiveData(bytes);
        }

        /// <summary>
        /// 解析2进制原始数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static string receiveData(byte[] bytes)
        {
            byte[] newBytes = null;
            if (endbytes != null && endbytes.Length > 0)
            {
                newBytes = new byte[endbytes.Length + bytes.Length];
                bytes = Common.CombineBytes(endbytes, bytes);
            }
            else
            {
                newBytes = new byte[bytes.Length];
                newBytes = bytes;
            }
            int i = 0;//初始化byte[]下标
            byte[] parserBytes = null;//去除正确包头之前的数据后的数据
            foreach (var item in newBytes)
            {
                if (item == 0x7E)
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserLocation(parserBytes);
                }
                else if (item == '(')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserCommand(parserBytes);
                }
                i++;
            }
            return null;
        }

        private static string parserLocation(byte[] bytes) {
```

```
            byte[] frame = PacketUtil.decodePacket(bytes);
            if (frame == null) {
                endbytes = bytes;
                return null;
            }
            //定义定位数据实体类
            Result model = new Result();
            //消息ID
            int msgId = Common.SwapUInt16(BitConverter.ToUInt16(frame, 1));
            //消息体属性
            int msgBodyAttr = Common.SwapUInt16(BitConverter.ToUInt16(frame
, 3));
            //消息体长度
            int msgBodyLen = msgBodyAttr & 0x03FF;
            byte[] terminalNumArr = bytes.Skip(5).Take(6).ToArray();
            model.DeviceID = Common.ByteToHexStr(terminalNumArr).TrimStart(
'0');
            //消息流水号
            int msgFlowId = Common.SwapUInt16(BitConverter.ToUInt16(frame,
11));
            //消息体
            byte[] msgBodyArr = bytes.Skip(13).Take(msgBodyLen).ToArray();
            if (msgId == 0x0200)
            {
                //解析消息体
                LocationData locationData = PacketUtil.parseLocationBody(ms
gBodyArr);
                locationData.Index=msgFlowId;
                locationData.DataLength=msgBodyLen;
                model.MsgType="Location";
                model.DataBody=locationData;
                string replyMsg = PacketUtil.replyBinaryMessage(terminalNum
Arr, msgFlowId);
                model.ReplyMsg=replyMsg;
            }
            else
            {
                model.MsgType = "heartbeat";
            }
            return JsonConvert.SerializeObject(model);
        }

        /// <summary>
        /// 解析指令数据
        /// </summary>
        /// <param name="bytes"></param>
```

```
        /// <returns></returns>
        private static string parserCommand(byte[] bytes)
        {
            Result result = new Result();
            byte[] newBytes = null;
            int tailIndex = Common.BytesIndexOf(bytes, 0x29) + 1;
            if (tailIndex > 0)
            {
                if (bytes.Length > tailIndex)
                {
                    endbytes = bytes.Skip(tailIndex).Take(bytes.Length - ta
ilIndex).ToArray();
                }
                newBytes = bytes.Skip(0).Take(tailIndex).ToArray();
            }
            else
            {
                endbytes = new byte[bytes.Length];
                endbytes = bytes;
            }
            if (newBytes != null && newBytes.Length > 0)
            {
                //转换成（）带括号的数据
                string msgAscii = Common.ByteToASCII(newBytes);
                //按逗号拆分字符串
                char[] p = new char[] { ',' };
                //返回的数组中，包含空字符串
                string[] msgArrays = msgAscii.Replace("(", "").Replace(")",
 "").Split(p, StringSplitOptions.None);
                result.DeviceID = msgArrays[0];
                string msgType = string.Empty;
                if (msgArrays.Length > 5)
                {
                    msgType = msgArrays[3] + msgArrays[4];
                }
                else
                {
                    return null;
                }
                result.MsgType = msgType;
                result.DataBody = msgAscii;
                string replyMsg = string.Empty;
                if (msgType.Equals("BASE2") && msgArrays[5].ToUpper().Equal
s("TIME"))
                {
                    replyMsg = PacketUtil.replyBASE2Message(msgArrays);
```

```
                }
                result.ReplyMsg = replyMsg;
                return JsonConvert.SerializeObject(result);
            }
            else
            {
                return null;
            }
        }
    }
}
```

（2）公共方法类：Common.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Jt704DataParser
{
    public class Common
    {

        /// <summary>
        /// 16进制格式字符串转字节数组
        /// </summary>
        /// <param name="hexString"></param>
        /// <returns></returns>
        public static byte[] HexStringToBytes(string hexString)
        {
            hexString = Regex.Replace(hexString, @".{2}", "$0 ");
            //以 ' ' 分割字符串，并去掉空字符
            string[] chars = hexString.Split(new char[] { ' ' }, StringSpli
tOptions.RemoveEmptyEntries);
            byte[] returnBytes = new byte[chars.Length];
            //逐个字符变为16进制字节数据
            for (int i = 0; i < chars.Length; i++)
            {
                returnBytes[i] = Convert.ToByte(chars[i], 16);
            }
            return returnBytes;
        }
```

```
        /// <summary>
        /// 合并数组
        /// </summary>
        /// <param name="bytes1"></param>
        /// <param name="bytes2"></param>
        /// <returns></returns>
        public static byte[] CombineBytes(byte[] bytes1, byte[] bytes2)
        {
            List<byte> tmp = new List<byte>(bytes1.Length + bytes2.Length);
            tmp.AddRange(bytes1);
            tmp.AddRange(bytes2);
            byte[] merged = tmp.ToArray();
            return merged;
        }

        /// <summary>
        /// 报告指定的 System.Byte[] 在此实例中的第一个匹配项的索引。
        /// </summary>
        /// <param name="srcBytes">被执行查找的 System.Byte[]。</param>
        /// <param name="searchBytes">要查找的 System.Byte[]。</param>
        /// <returns>如果找到该字节数组，则为 searchBytes 的索引位置；如果未找
到该字节数组，则为 -1。如果 searchBytes 为 null 或者长度为0，则返回值为 -1。</re
turns>
        public static int BytesIndexOf(byte[] srcBytes, byte searchBytes)
        {
            if (srcBytes == null) { return -1; }
            if (srcBytes.Length == 0) { return -1; }
            for (int i = 0; i < srcBytes.Length; i++)
            {
                if (srcBytes[i] == searchBytes)
                {
                    return i;
                }
            }
            return -1;
        }

        /// <summary>
        /// 16进制字符串转ASCII
        /// </summary>
        /// <param name="Data"></param>
        /// <param name="istrans"></param>
        /// <returns></returns>
        public static string HexStrToAsciiString(string Data, bool istrans)
        {
```

```
            if (istrans)
            {
                Data = Regex.Replace(Data, @"(?is)(?<=^([0-9a-f]{2})+)(?!$)
", " ");
                Data = Data.Replace("3D 00", "3D ").Replace("3D 11", "2C ")
.Replace("3D 14", "28 ").Replace("3D 15", "29 ").Replace(" ", "");
            }
            int count = Data.Length / 2;
            string strContent = "";
            char[] num = new char[count];
            for (int i = 0; i < count; i++)
            {
                string str = Data.Substring(i * 2, 2);
                byte by = Convert.ToByte(Convert.ToInt32(str, 16));
                num[i] = Convert.ToChar(by);
                strContent += num[i].ToString();
            }
            return strContent;
        }

        /// <summary>
        /// 二进制转ASCII
        /// </summary>
        /// <param name="bt"></param>
        /// <returns></returns>
        public static string ByteToASCII(byte[] bt)
        {
            string lin = "";
            for (int i = 0; i < bt.Length; i++)
            {
                lin = lin + bt[i] + " ";
            }
            string[] ss = lin.Trim().Split(new char[] { ' ' });
            char[] c = new char[ss.Length];
            int a;
            for (int i = 0; i < c.Length; i++)
            {
                a = Convert.ToInt32(ss[i]);
                c[i] = Convert.ToChar(a);
            }

            string b = new string(c);
            return b;
        }

        /// <summary>
```

```
        /// 时间格式转换
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DateTime GetDataTime(byte[] bytes)
        {
            return DateTime.ParseExact(ByteToHexStr(bytes), "yyMMddHHmmss",
 System.Globalization.CultureInfo.CurrentCulture);
        }

        public static DateTime GetDataTime(string strdate)
        {
            return DateTime.ParseExact(strdate, "yyMMddHHmmss", System.Glob
alization.CultureInfo.CurrentCulture);
        }

        /// <summary>
        /// 获取二进制第index位的值
        /// </summary>
        /// <param name="number"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public static int getBitValue(long number, int index)
        {
            return (number & (1 << index)) > 0 ? 1 : 0;
        }

        /// <summary>
        /// 字节数组转16进制字符串
        /// </summary>
        /// <param name="byteDatas"></param>
        /// <returns></returns>
        public static string ByteToHexStr(byte[] byteDatas)
        {
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < byteDatas.Length; i++)
            {
                builder.Append(string.Format("{0:X2}", byteDatas[i]));
            }
            return builder.ToString().Trim();
        }

        public static ushort SwapUInt16(ushort v)
        {
            return (ushort)(((v & 0xff) << 8) | ((v >> 8) & 0xff));
        }
```

```
        public static uint SwapUInt32(uint v)
        {
            return (uint)(((SwapUInt16((ushort)v) & 0xffff) << 0x10) |
                          (SwapUInt16((ushort)(v >> 0x10)) & 0xffff));
        }

        /// <summary>
        /// Short转Bytes
        /// </summary>
        /// <param name="number"></param>
        /// <returns></returns>
        public static byte[] ShortToBytes(short number)
        {
            byte byte2 = (byte)(number >> 8);
            byte byte1 = (byte)(number & 255);
            byte[] bytes = new byte[2];
            bytes[0] = byte1;
            bytes[1] = byte2;
            return bytes.Reverse().ToArray();
        }

        /// <summary>
        /// 随机short
        /// </summary>
        /// <returns></returns>
        public static short RandomNumber(int min, int max)
        {
            Random rd = new Random();
            return (short)rd.Next(min, max);
        }

        /// <summary>
        /// 计算校验码
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static byte xor(List<byte> bytes)
        {
            int checksum = 0;
            foreach (byte b in bytes) {
                checksum ^= b;
            }
            return (byte)(checksum &0xff);
        }
```

```
        /// <summary>
        /// 转义
        /// </summary>
        /// <param name="inBytes"></param>
        /// <returns></returns>
        public static byte[] escape(List<byte> inBytes)
        {
            List<byte> outBytes = new List<byte>();
            foreach (byte b in inBytes) {
                if (b == 0x7E) {
                    outBytes.AddRange(HexStringToBytes("7D02"));
                }else if (b == 0x7D) {
                    outBytes.AddRange(HexStringToBytes("7D01"));
                } else {
                    outBytes.Add(b);
                }
            }
            return outBytes.ToArray();
        }
    }
}
```

(3)解包工具类PacketUtil.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt704DataParser
{
    public class PacketUtil
    {

        public static byte[] decodePacket(byte[] bytes) {
            //查找消息尾
            byte[] msgBodyNoHeader = bytes.Skip(1).Take(bytes.Length - 1).T
oArray();
            int tailIndex = Common.BytesIndexOf(msgBodyNoHeader, 0x7E);
            if (tailIndex <= 0) {
                return null;
            }
           return unescape(bytes, tailIndex).ToArray();
        }
```

```
        /// <summary>
        /// 反转义
        /// </summary>
        /// <param name="bytes"></param>
        /// <param name="bodyLen"></param>
        /// <returns></returns>
        public static List<byte> unescape(byte[] bytes, int bodyLen)
        {
            int i = 0;
            List<byte> frame = new List<byte>();
            while (i < bodyLen)
            {
                byte b = bytes[i];
                if (b == 0x7D)
                {
                    byte nextByte = bytes[i+1];
                    if (nextByte == 0x01)
                    {
                        frame.Add(0x7D);
                    }
                    else if (nextByte == 0x02)
                    {
                        frame.Add(0x7E);
                    }
                    else
                    {
                        //异常数据
                        frame.Add(b);
                        frame.Add(nextByte);
                    }
                    i += 2;
                }
                else
                {
                    frame.Add(b);
                    i++;
                }
            }
            return frame;
        }

        /// <summary>
        /// 解析定位消息体
        /// </summary>
        /// <param name="msgBodyBuf"></param>
        /// <returns></returns>
```

```
        public static LocationData parseLocationBody(byte[] msgBodyBuf)
        {
            //报警标志
            long alarmFlag = Common.SwapUInt32(BitConverter.ToUInt32(msgBod
yBuf, 0));
            //状态
            long status = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBu
f, 4));
            //纬度
            double lat = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 8))* 0.000001;
            //经度
            double lon = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 12)) * 0.000001;
            //海拔高度,单位为米
            int altitude = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 16));
            //速度
            double speed = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 18))*0.1;
            //方向
            int direction = Common.SwapUInt16(BitConverter.ToUInt16(msgBody
Buf, 20));
            //定位时间
            DateTime gpsZonedDateTime = Common.GetDataTime(msgBodyBuf.Skip(
22).Take(6).ToArray());
            //根据状态位的值判断是否南纬和西经
            if (Common.getBitValue(status, 2) == 1)
            {
                lat = -lat;
            }
            if (Common.getBitValue(status, 3) == 1)
            {
                lon = -lon;
            }
            //定位状态
            int locationType = Common.getBitValue(status, 18);
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 1);
            }
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 6) > 0 ? 2 : 0;
            }
            //报警状态
```

```
            int alarm = -1;
            if (Common.getBitValue(alarmFlag, 16) == 1)
            {
                alarm = (int)AlarmTypeEnum.ALARM_1;
            }
            //后盖状态
            int backCover = Common.getBitValue(status, 7);
            //唤醒源
            long awaken = (status >> 24) & 0x0f;

            LocationData locationData = new LocationData();
            locationData.GpsTime = gpsZonedDateTime;
            locationData.Latitude=lat;
            locationData.Longitude=lon;
            locationData.LocationType=locationType;
            locationData.Speed=(int)speed;
            locationData.Direction=direction;
            locationData.Alarm=alarm;
            locationData.BackCover=backCover;
            locationData.Awaken=(int)awaken;
            //处理附加信息
            if (msgBodyBuf.Length > 28)
            {
                byte[] extraBytes = msgBodyBuf.Skip(28).Take(msgBodyBuf.Len
gth - 28).ToArray();
                parseExtraInfo(extraBytes, locationData);
            }
            return locationData;
        }

        /// <summary>
        /// 解析附加信息
        /// </summary>
        /// <param name="msgBody"></param>
        /// <param name="location"></param>
        private static void parseExtraInfo(byte[] msgBody, LocationData loc
ation)
        {
            byte[] extraInfoBuf = null;
            while (msgBody.Length > 1)
            {
                int extraInfoId = msgBody[0];
                int extraInfoLen = msgBody[1];
                if (msgBody.Length-2 < extraInfoLen)
                {
                    break;
```

```
                }
                extraInfoBuf = msgBody.Skip(2).Take(extraInfoLen).ToArray()
;
                msgBody= msgBody.Skip(2+ extraInfoLen).Take(msgBody.Length-
(2 + extraInfoLen)).ToArray();
                switch (extraInfoId)
                {
                    case 0x0F:
                        //解析温度数据
                        double temperature = -1000.0;
                        temperature = parseTemperature(Common.SwapUInt16(Bi
tConverter.ToUInt16(extraInfoBuf, 0)));
                        location.Temperature=(int)temperature;
                        break;
                    //无线通信网络信号强度
                    case 0x30:
                        int fCellSignal = extraInfoBuf[0];
                        location.GSMSignal = fCellSignal;
                        break;
                    //卫星数
                    case 0x31:
                        int fGPSSignal = extraInfoBuf[0];
                        location.GpsSignal = fGPSSignal;
                        break;
                    //电池电量百分比
                    case 0xD4:
                        int fBattery = extraInfoBuf[0];
                        location.Battery=fBattery;
                        break;
                    //电池电压
                    case 0xD5:
                        int fVoltage = Common.SwapUInt16(BitConverter.ToUIn
t16(extraInfoBuf, 0));
                        location.Voltage=fVoltage * 0.01;
                        break;
                    case 0xFD:
                        //小区码信息
                        int mcc = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 0));
                        location.MCC=mcc;
                        int mnc = extraInfoBuf[2];
                        location.MNC=mnc;
                        long cellId = Common.SwapUInt32(BitConverter.ToUInt
32(extraInfoBuf, 3));
                        location.CELLID= cellId;
                        int lac = Common.SwapUInt16(BitConverter.ToUInt16(e
```

```
xtraInfoBuf, 7));
                        location.LAC=lac;
                        break;
                    case 0xFE:
                        long mileage = Common.SwapUInt32(BitConverter.ToUIn
t32(extraInfoBuf, 0));
                        location.Mileage=mileage;
                        break;
                    default:
                        break;
                }
            }
        }

        /// <summary>
        /// 温度解析
        /// </summary>
        /// <param name="temperatureInt"></param>
        /// <returns></returns>
        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;
        }

        /// <summary>
        /// 回复内容
        /// </summary>
        /// <param name="terminalNumArr"></param>
        /// <param name="msgFlowId"></param>
        /// <returns></returns>
        public static string replyBinaryMessage(byte[] terminalNumArr, int
msgFlowId) {
            List<byte> byteList = new List<byte>();
            byteList.AddRange(Common.HexStringToBytes("8001"));
            byteList.AddRange(Common.HexStringToBytes("0005"));
            byteList.AddRange(terminalNumArr);
            byteList.AddRange(Common.ShortToBytes(Common.RandomNumber(0, 65
```

```
534)));
            byteList.AddRange(Common.ShortToBytes((short)msgFlowId));
            byteList.AddRange(Common.HexStringToBytes("0200"));
            byteList.Add(0x00);
            byteList.Add(Common.xor(byteList));
            byte[] bytes = Common.escape(byteList);
            List<byte> replyList = new List<byte>();
            replyList.Add(0x7E);
            replyList.AddRange(bytes);
            replyList.Add(0x7E);
            return Common.ByteToHexStr(replyList.ToArray());
        }

        /// <summary>
        /// 授时
        /// </summary>
        /// <param name="itemList"></param>
        /// <returns></returns>
        public static string replyBASE2Message(string[] itemList)
        {
            try
            {
                string strBase2Reply = string.Format("({0},{1},{2},{3},{4},
{5})", itemList[0], itemList[1]
                        , itemList[2], itemList[3], itemList[4], DateTime.U
tcNow.ToString("yyyyMMddHHmmss"));
                return strBase2Reply;
            }
            catch (Exception e)
            {
                return "";
            }
        }
    }
}
```

2.4.返回消息及说明

（1）心跳数据
原始数据：

```
7E0002000075018042831000012 67E
```

返回消息：

```
{"DeviceID":"750180428310","MsgType":"heartbeat","DataBody":null,"ReplyMsg":
null}
```

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据：

```
7E0200003B750180925004010200000000000004000201588F0F06CA3C520027000000001811O
6123212D4015AD502015C30011431010CFD0901CC00000010922866EF014A0F0201406E7E
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": 终端号,
    "DataBody": {
        "Speed": 速度,
        "GpsTime": 定位时间（UTC）,
        "Temperature": 温度,
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "Mileage": 里程值（km）,
        "BackCover": 后盖状态（1：开；0：关）,
        "Index": 流水号,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "Awaken": 唤醒源（0:RTC上报 3：开盖上报  4：关盖上报）,
        "Direction": 方向（0~360,0表示正北）,
        "Alarm": 报警类型（1：主机拆卸报警；-1：无报警）,
        "Battery": 电量值（0~100%）,
        "GpsSignal": 卫星颗数,
        "Voltage": 电压值（单位：V）,
        "DataLength": 数据长度,
        "LocationType": 定位类型（1：GPS定位；0：不定位）,
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的指令,
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析
原始数据：

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

返回消息：

```
{
    "DeviceID": "700160818000",
```

```
    "DataBody": "(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B
03SIM900M64_ST_MMS,89860042191130272549,012207005620932,460,00,4243,6877)",
    "ReplyMsg": "",
    "MsgType": "BASE1"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复）,
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,8986004219113027254 9,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	20150418_G300	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	SIM卡的ICCID。
7	012207005620932	GSM模块的IMEI号

8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，保留字段，暂无意义。
4	30	设备正常工作间隔，单位分钟，默认1440分钟，取值范围（5~1440）。

3.6.设置/查询设备本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.7.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.8.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	1则表明当前有效闹钟个数，闹钟时间点最多4个；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.9.设置/调试选项

消息体内容（DataBody）：

(2310915002,1,001,DEBUG,2,2,1)

消息体内容描述：

序号	示例	说明
1	2310915002	设备ID
2	DEBUG,2	指令类型（DEBUG2）
3	2	2: 没意义，固定字段
4	1	1: 表示调试GPS模块原始输出信息 2表示调试GPRS原始输出AT指令信息

3.10.设置设备进入休眠的时间

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15,10)

消息体内容描述：

序号	示例	说明
1	2310915002	设备ID
2	DEBUG,15	指令类型（DEBUG15）
3	10	10: 表示设备10秒钟后进入休眠

3.11.查询设备还可以存储多少条数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,19,1000)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	DEBUG,19	指令类型（DEBUG19）
3	1000	1000表示设备可以存储1000条数据

3.12.查询/设置/启用基站授时

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,26,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,26	指令类型（DEBUG26）
3	1	返回1则表明启用基站授时，0表示关闭基站授时。

3.13.查询卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27,123456789,888888888)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,27	指令类型（DEBUG27）
3	123456789	123456789表示ICCID
4	888888888	888888888表示IMSI

3.14.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.15.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.16.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.17.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.18.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.

4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.19.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT704_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT704_V103R001	固件版本号

Java 版本

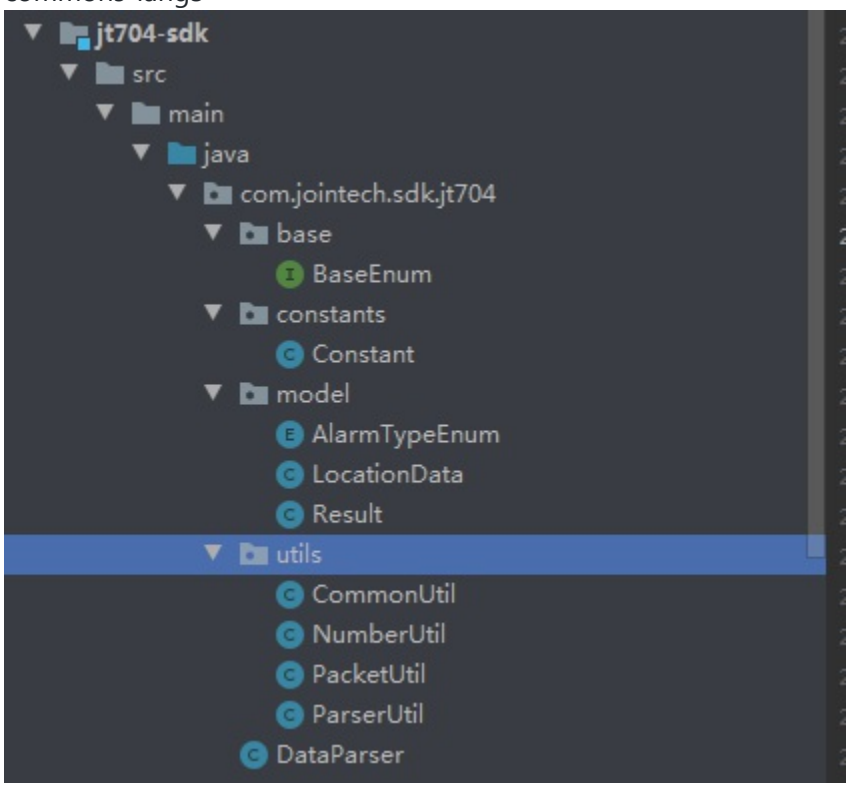
1.Jar包下载

jt704-sdk-1.0.0.jar 下载地址
如果需要Jar包开发源码，请与商务申请

2.集成开发说明

2.1.集成开发语言及框架说明

jt704-sdk-1.0.0.jar是基于Java语言，SpringBoot2.x框架，使用到了netty，fastjson，lombok，commons-lang3



BaseEnum：基础枚举
Constant：自定义常量
AlarmTypeEnum：终端报警类型枚举
LocationData：定位实体类
Result：结果实体类
CommonUtil：公共方法类
NumberUtil：数字操作工具类
PacketUtil：用来处理预处理数据以及分包方法封装类
ParserUtil：解析方法工具类

DataParser：解析主方法

2.2.集成说明

将jt704-sdk-1.0.0.jar引入到自己的网关程序中，引入方法如下：
在pom.xml引入jar包

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

调用jt704-sdk-1.0.0.jar，DataParser类中receiveData()方法
receiveData()方法是重载方法

```
    /**
     * 解析Hex字符串原始数据
     * @param strData 16进制字符串
     * @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);
    }

    /**
     * 解析byte[]原始数据
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * 解析ByteBuf原始数据
     * @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.核心代码

ParserUtil：解析方法工具类

```
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: 解析方法工具类</p>
 * @author HyoJung
 * @date 20210526
 */
public class ParserUtil {
    /**
     * 定位数据解析0x0200
```

```
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //对数据进行反转移处理
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //消息长度
        int msgLen = msg.readableBytes();
        //包头
        msg.readByte();
        //消息ID
        int msgId = msg.readUnsignedShort();
        //消息体属性
        int msgBodyAttr = msg.readUnsignedShort();
        //消息体长度
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //是否分包
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //去除消息体的基础长度
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //根据消息体长度和是否分包得出后面的包长
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
        //终端号数组
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //终端号(去除前面的0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //消息流水号
        int msgFlowId = msg.readUnsignedShort();
        //消息总包数
        int packetTotalCount = 0;
        //包序号
        int packetOrder = 0;
        //分包
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //消息体
        byte[] msgBodyArr = new byte[msgBodyLen];
```

```
            msg.readBytes(msgBodyArr);
            if(msgId==0x0200) {
                //解析消息体
                LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
                locationData.setIndex(msgFlowId);
                locationData.setDataLength(msgBodyLen);
                //校验码
                int checkCode = msg.readUnsignedByte();
                //包尾
                msg.readByte();
                //获取消息回复内容
                String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
                //定义定位数据实体类
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("Location");
                model.setDataBody(locationData);
                model.setReplyMsg(replyMsg);
                return model;
            }else {
                //定义定位数据实体类
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("heartbeat");
                model.setDataBody(null);
                return model;
            }
        }

        /**
         * 解析指令数据
         * @param in 原始数据
         * @return
         */
        public static Result decodeTextMessage(ByteBuf in) {

            //读指针设置到消息头
            in.markReaderIndex();
            //查找消息尾，如果未找到则继续等待下一包
            int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
            if (tailIndex < 0) {
                in.resetReaderIndex();
                return null;
            }
```

```
        //定义定位数据实体类
        Result model = new Result();
        //包头(
        in.readByte();
        //字段列表
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //查询逗号的下标截取数据
            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;
    }

    /**
     * 解析定位消息体
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
```

```
        //报警标志
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //状态
        long status = msgBodyBuf.readUnsignedInt();
        //纬度
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //经度
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //海拔高度,单位为米
        int altitude = msgBodyBuf.readShort();
        //速度
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //方向
        int direction = msgBodyBuf.readShort();
        //定位时间
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
        //根据状态位的值判断是否南纬和西经
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //定位状态
        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;
        }
        //报警状态
        int alarm=-1;
        if(NumberUtil.getBitValue(alarmFlag, 16) == 1)
        {
            alarm=Integer.parseInt(AlarmTypeEnum.ALARM_1.getValue());
        }
```

```
        //后盖状态
        int backCover=NumberUtil.getBitValue(status, 7);
        //唤醒源
        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);
        //处理附加信息
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }

    /**
     * 解析附加信息
     *
     * @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:
                    //解析温度数据
                    double temperature = -1000.0;
                    temperature = parseTemperature(extraInfoBuf.readShort()
);
                    location.setTemperature((int)temperature);
                    break;
```

```
                    //无线通信网络信号强度
                    case 0x30:
                        int fCellSignal=extraInfoBuf.readByte();
                        location.setGSMSignal(fCellSignal);
                        break;
                    //卫星数
                    case 0x31:
                        int fGPSSignal=extraInfoBuf.readByte();
                        location.setGpsSignal(fGPSSignal);
                        break;
                    //电池电量百分比
                    case 0xD4:
                        int fBattery=extraInfoBuf.readUnsignedByte();
                        location.setBattery(fBattery);
                        break;
                    //电池电压
                    case 0xD5:
                        int fVoltage=extraInfoBuf.readUnsignedShort();
                        location.setVoltage(fVoltage*0.01);
                        break;
                    case 0xFD:
                        //小区码信息
                        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;
                }
            }
        }

        /**
         * 温度解析
         * @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：用来处理预处理数据以及恢复方法封装类

```
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;

/**
 * 解析包预处理（进行反转义）
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * 解析消息包
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //可读长度不能小于基本长度
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }
```

```
        //防止非法码流攻击，数据太大为异常数据
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //查找消息尾，如果未找到则继续等待下一包
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //创建ByteBuf存放反转义后的数据
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //消息头尾之间的数据进行反转义
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //反转义后的长度不能小于基本长度
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
        return frame;
    }

    /**
     * 消息头、消息体、校验码中0x7D 0x02反转义为0x7E，0x7D 0x01反转义为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 {
                    //异常数据
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * 消息头、消息体、校验码中0x7E转义为0x7D 0x02，0x7D转义为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);
            }
        }
    }

    /**
     * 回复内容
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //去除包头包尾的长度
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
```

```
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //消息ID
            bodyBuf.writeShort(0x8001);
            //数据长度
            bodyBuf.writeShort(0x0005);
            //终端ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //生成1-65534内的随机数
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //当前消息流水号
            bodyBuf.writeShort(index);
            //回复消息流水号
            bodyBuf.writeShort(msgFlowId);
            //应答的消息ID
            bodyBuf.writeShort(0x0200);
            //应答结果
            bodyBuf.writeByte(0x00);
            //校验码
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //包头
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //读指针重置到起始位置
            bodyBuf.readerIndex(0);
            //转义
            PacketUtil.escape(replyBuf, bodyBuf);
            //包尾
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }

    /**
     * 授时指令回复
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //设置日期格式
            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：数字操作工具类

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

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

/**
 * 数字工具类
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     * 坐标精度
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     * 坐标因数
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * 小数点一位精度
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }
```

```java
    /**
     * 格式化消息ID(转成0xXXXX)
     *
     * @param msgId 消息ID
     * @return 格式化字符串
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * 格式化短数字
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
     * 转4位的十六进制字符串
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * 解析short类型的值,获取值为1的位数
     *
     * @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;
    }
```

```java
    /**
     * 解析int类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 获取二进制第index位的值
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

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

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

```
        }
    }
```

CommonUtil：公共方法类

```
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: 用来存储一些解析中遇到的公共方法</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * 去掉字符串最后一个字符
     * @param inStr 输入的字符串
     * @param suffix 需要去掉的字符
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * 16进制转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();
    }

    /**
     * 转换GPS时间
     *
     * @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;
    }

    /**
     * 每个字节进行异或求值
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

LocationData：定位实体类

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

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

```java
import lombok.Data;

import java.io.Serializable;

/**
 * <p>Description: 定位实体类</p>
 *
 * @author Lenny
 * @version 1.0.1
 * @date 20210328
 */
@Data
public class LocationData implements Serializable {
    /**
     * 消息体
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * 定位时间
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * 纬度
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * 经度
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * 定位方式
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * 速度
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * 方向
     */
```

```
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * 里程
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * GPS信号值
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM信号质量
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * 电量值
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * 电压
     */
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * 后盖状态
     */
    @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;
    /**
     * 报警类型
     */
    @JSONField(name = "Alarm")
    public int Alarm = -1;
    /**
     * 流水号
     */
    @JSONField(name = "Index")
    public int Index;
    /**
     * 温度值
     */
    @JSONField(name = "Temperature")
    public int Temperature=-1000;
}
```

Result：结果实体类

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

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

import java.io.Serializable;

/**
 * 结果实体类
 * @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：终端报警类型枚举

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

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

/**
 * 终端报警类型
 * @author HyoJung
 */
public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("主机拆卸报警", "1");

    @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;
        }
    }
```

2.4.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007501804283100001267E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据：

```
7E0200003B750180925004010200000000000400020158F0F06CA3C5200270000000018110
6123212D4015AD502015C30011431010CFD0901CC00000010922866EF014A0F0201406E7E
```

返回消息：

```
{
    "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"
    }
```

返回消息描述

```
{
    "DeviceID": 终端号,
    "DataBody": {
        "Speed": 速度,
        "GpsTime": 定位时间（UTC）,
        "Temperature": 温度,
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "Mileage": 里程值（km）,
        "BackCover": 后盖状态（1：开；0：关）,
        "Index": 流水号,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "Awaken": 唤醒源（0:RTC上报 3：开盖上报  4：关盖上报）,
        "Direction": 方向（0~360,0表示正北）,
        "Alarm": 报警类型（1：主机拆卸报警；-1：无报警）,
        "Battery": 电量值（0~100%）,
        "GpsSignal": 卫星颗数,
        "Voltage": 电压值（单位：V）,
        "DataLength": 数据长度,
        "LocationType": 定位类型（1：GPS定位；0：不定位）,
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的指令,
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析
原始数据：

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

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复）,
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,8986004
2191130272549,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	BASE,1	指令类型（BASE1）
3	20150418_G300	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	SIM卡的ICCID。
7	012207005620932	GSM模块的IMEI号
8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，保留字段，暂无意义。
4	30	设备正常工作间隔，单位分钟，默认1440分钟，取值范围（5~1440）。

3.6.设置/查询设备本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.7.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.8.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序	示例	说明

号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	1则表明当前有效闹钟个数，闹钟时间点最多4个；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.9.设置/调试选项

消息体内容（DataBody）：

(2310915002,1,001,DEBUG,2,2,1)

消息体内容描述：

序号	示例	说明
1	2310915002	设备ID
2	DEBUG,2	指令类型（DEBUG2）
3	2	2: 没意义，固定字段
4	1	1: 表示调试GPS模块原始输出信息 2表示调试GPRS原始输出AT指令信息

3.10.设置设备进入休眠的时间

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15,10)

消息体内容描述：

序号	示例	说明
1	2310915002	设备ID
2	DEBUG,15	指令类型（DEBUG15）

3.11.查询设备还可以存储多少条数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,19,1000)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,19	指令类型（DEBUG19）
3	1000	1000表示设备可以存储1000条数据

3.12.查询/设置/启用基站授时

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,26,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,26	指令类型（DEBUG26）
3	1	返回1则表明启用基站授时，0表示关闭基站授时。

3.13.查询卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27,123456789,888888888)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	DEBUG,27	指令类型（DEBUG27）
3	123456789	123456789表示ICCID
4	888888888	888888888表示IMSI

3.14.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.15.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.

5	100	下一条需要发送的数据包序号.

3.16.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.17.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.18.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.19.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT704_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT707A_V103R001	固件版本号

JT705A SDK集成开发

.NET版本

Java版本

.NET 版本

1.SDK下载

JT705SDK(JT705SDK_V2.rar) 下载地址
JT705SDK测试工具(JT705SDK_Test.rar) 下载地址
如果需要测试工具及SDK开发源码，请与商务申请

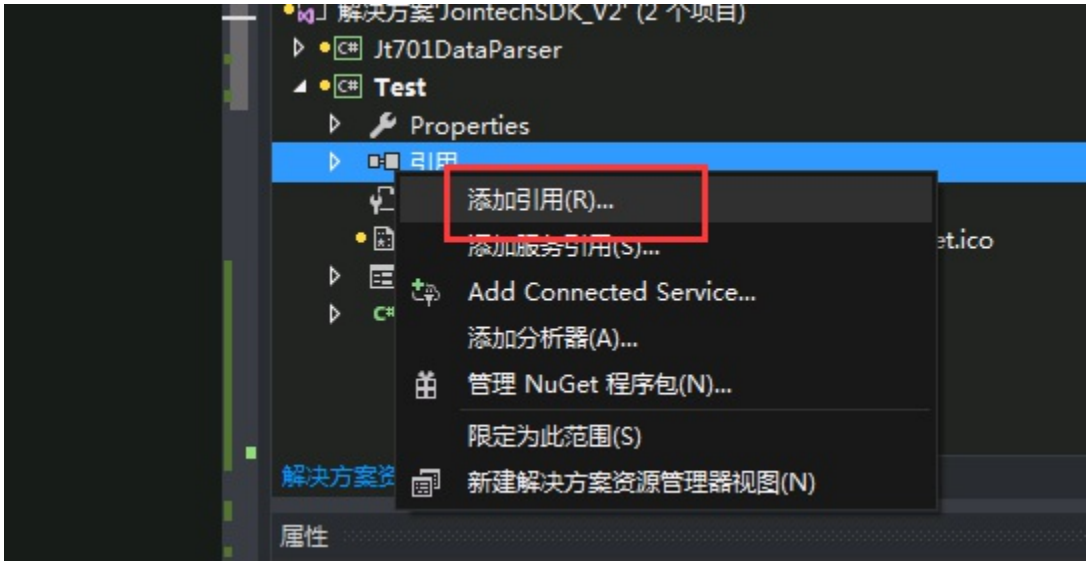
2.集成开发说明

2.1.集成开发语言及框架说明

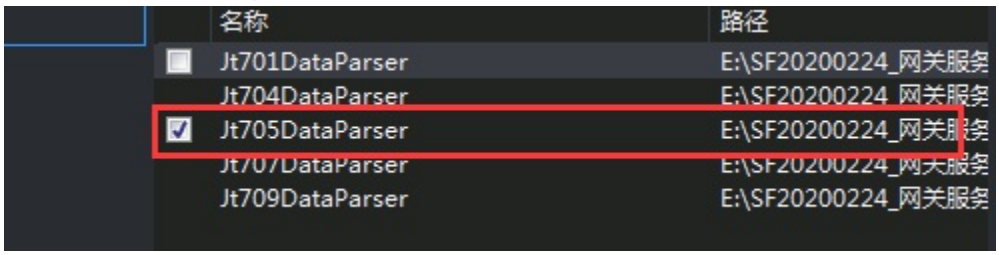
JT705SDK（Jt705DataParser.dll）及测试工具（Test.exe）是基于C#语言，.NET Framework 4.6.1目标框架开发的。该SDK的集成开发环境也是需要依赖于C#语言以及.NET Framework 4.6.1目标框架。

2.2.集成说明

（1）将Jt705DataParser.dll引入到自己的网关程序中，引入方法如下：
鼠标右键项目下的"引用"，选择"添加引用(R)..."



选择下图中"浏览(B)..."，找到你解压JT705SDK_V2.rar后的文件夹，选中Jt705DataParser.dll与对应的Json字符串序列化/反序列化包Newtonsoft.Json.dll即可完成对SDK的引用



（2）调用Jt705DataParser.dll中的解析方法receiveData
receiveData()是一个重载方法，你可以传入16进制字符串或者byte[],一般我们网关程序接收到的设备数据是以二进制流进行传输的，所以我们这里建议使用此重载方法receiveData(byte[] bytes);
当然如果你想通过16进制字符串进行测试，也可以使用receiveData(string strData)；
strData：就是我们接收到的设备数据转成16进制后的字符串

2.3.核心代码

Jt705DataParser.dll
(1)解析类DataParser.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt705DataParser
{
    public class DataParser
    {
        private static byte[] endbytes = null;// 上一次未处理的剩余字节

        /// <summary>
        /// 解析16进制原始数据
        /// </summary>
        /// <param name="strData"></param>
        /// <returns></returns>
        public static string receiveData(string strData)
        {
            byte[] bytes = Common.HexStringToBytes(strData);
            return receiveData(bytes);
        }

        /// <summary>
        /// 解析2进制原始数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static string receiveData(byte[] bytes)
        {
            byte[] newBytes = null;
            if (endbytes != null && endbytes.Length > 0)
            {
```

```
                newBytes = new byte[endbytes.Length + bytes.Length];
                bytes = Common.CombineBytes(endbytes, bytes);
            }
            else
            {
                newBytes = new byte[bytes.Length];
                newBytes = bytes;
            }
            int i = 0;//初始化byte[]下标
            byte[] parserBytes = null;//去除正确包头之前的数据后的数据
            foreach (var item in newBytes)
            {
                if (item == 0x7E)
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserLocation(parserBytes);
                }
                else if (item == '(')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserCommand(parserBytes);
                }
                i++;
            }
            return null;
        }

        private static string parserLocation(byte[] bytes)
        {
            byte[] frame = PacketUtil.decodePacket(bytes);
            if (frame == null)
            {
                endbytes = bytes;
                return null;
            }
            //定义定位数据实体类
            Result model = new Result();
            //消息ID
            int msgId = Common.SwapUInt16(BitConverter.ToUInt16(frame, 1));
            //消息体属性
            int msgBodyAttr = Common.SwapUInt16(BitConverter.ToUInt16(frame
, 3));
```

```
            //消息体长度
            int msgBodyLen = msgBodyAttr & 0x03FF;
            byte[] terminalNumArr = bytes.Skip(5).Take(6).ToArray();
            model.DeviceID = Common.ByteToHexStr(terminalNumArr).TrimStart(
'0');
            //消息流水号
            int msgFlowId = Common.SwapUInt16(BitConverter.ToUInt16(frame,
11));
            //消息体
            byte[] msgBodyArr = bytes.Skip(13).Take(msgBodyLen).ToArray();
            if (msgId == 0x0200)
            {
                //解析消息体
                LocationData locationData = PacketUtil.parseLocationBody(ms
gBodyArr);
                locationData.Index = msgFlowId;
                locationData.DataLength = msgBodyLen;
                model.MsgType = "Location";
                model.DataBody = locationData;
                string replyMsg = PacketUtil.replyBinaryMessage(terminalNum
Arr, msgFlowId);
                model.ReplyMsg = replyMsg;
            }
            else
            {
                model.MsgType = "heartbeat";
            }
            return JsonConvert.SerializeObject(model);
        }

        /// <summary>
        /// 解析指令数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static string parserCommand(byte[] bytes)
        {
            Result result = new Result();
            byte[] newBytes = null;
            int tailIndex = Common.BytesIndexOf(bytes, 0x29) + 1;
            if (tailIndex > 0)
            {
                if (bytes.Length > tailIndex)
                {
                    endbytes = bytes.Skip(tailIndex).Take(bytes.Length - ta
ilIndex).ToArray();
```

```
                }
                newBytes = bytes.Skip(0).Take(tailIndex).ToArray();
            }
            else
            {
                endbytes = new byte[bytes.Length];
                endbytes = bytes;
            }
            if (newBytes != null && newBytes.Length > 0)
            {
                //转换成（）带括号的数据
                string msgAscii = Common.ByteToASCII(newBytes);
                //按逗号拆分字符串
                char[] p = new char[] { ',' };
                //返回的数组中，包含空字符串
                string[] msgArrays = msgAscii.Replace("(", "").Replace(")",
 "").Split(p, StringSplitOptions.None);
                result.DeviceID = msgArrays[0];
                string msgType = string.Empty;
                if (msgArrays.Length > 5)
                {
                    msgType = msgArrays[3] + msgArrays[4];
                }
                else
                {
                    return null;
                }
                result.MsgType = msgType;
                result.DataBody = msgAscii;
                string replyMsg = string.Empty;
                if (msgType.Equals("BASE2") && msgArrays[5].ToUpper().Equal
s("TIME"))
                {
                    replyMsg = PacketUtil.replyBASE2Message(msgArrays);
                }
                result.ReplyMsg = replyMsg;
                return JsonConvert.SerializeObject(result);
            }
            else
            {
                return null;
            }
        }
    }
}
```

（2）公共方法类：Common.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Jt705DataParser
{
    public class Common
    {

        /// <summary>
        /// 16进制格式字符串转字节数组
        /// </summary>
        /// <param name="hexString"></param>
        /// <returns></returns>
        public static byte[] HexStringToBytes(string hexString)
        {
            hexString = Regex.Replace(hexString, @".{2}", "$0 ");
            //以 ' ' 分割字符串，并去掉空字符
            string[] chars = hexString.Split(new char[] { ' ' }, StringSpli
tOptions.RemoveEmptyEntries);
            byte[] returnBytes = new byte[chars.Length];
            //逐个字符变为16进制字节数据
            for (int i = 0; i < chars.Length; i++)
            {
                returnBytes[i] = Convert.ToByte(chars[i], 16);
            }
            return returnBytes;
        }

        /// <summary>
        /// 合并数组
        /// </summary>
        /// <param name="bytes1"></param>
        /// <param name="bytes2"></param>
        /// <returns></returns>
        public static byte[] CombineBytes(byte[] bytes1, byte[] bytes2)
        {
            List<byte> tmp = new List<byte>(bytes1.Length + bytes2.Length);
            tmp.AddRange(bytes1);
            tmp.AddRange(bytes2);
            byte[] merged = tmp.ToArray();
```

```
            return merged;
        }

        /// <summary>
        /// 报告指定的 System.Byte[] 在此实例中的第一个匹配项的索引。
        /// </summary>
        /// <param name="srcBytes">被执行查找的 System.Byte[]。</param>
        /// <param name="searchBytes">要查找的 System.Byte[]。</param>
        /// <returns>如果找到该字节数组，则为 searchBytes 的索引位置；如果未找到该字节数组，则为 -1。如果 searchBytes 为 null 或者长度为0，则返回值为 -1。</returns>
        public static int BytesIndexOf(byte[] srcBytes, byte searchBytes)
        {
            if (srcBytes == null) { return -1; }
            if (srcBytes.Length == 0) { return -1; }
            for (int i = 0; i < srcBytes.Length; i++)
            {
                if (srcBytes[i] == searchBytes)
                {
                    return i;
                }
            }
            return -1;
        }

        /// <summary>
        /// 16进制字符串转ASCII
        /// </summary>
        /// <param name="Data"></param>
        /// <param name="istrans"></param>
        /// <returns></returns>
        public static string HexStrToAsciiString(string Data, bool istrans)
        {
            if (istrans)
            {
                Data = Regex.Replace(Data, @"(?is)(?<=^([0-9a-f]{2})+)(?!$)", " ");
                Data = Data.Replace("3D 00", "3D ").Replace("3D 11", "2C ").Replace("3D 14", "28 ").Replace("3D 15", "29 ").Replace(" ", "");
            }
            int count = Data.Length / 2;
            string strContent = "";
            char[] num = new char[count];
            for (int i = 0; i < count; i++)
            {
                string str = Data.Substring(i * 2, 2);
```

```
                byte by = Convert.ToByte(Convert.ToInt32(str, 16));
                num[i] = Convert.ToChar(by);
                strContent += num[i].ToString();
            }
            return strContent;
        }

        /// <summary>
        /// 二进制转ASCII
        /// </summary>
        /// <param name="bt"></param>
        /// <returns></returns>
        public static string ByteToASCII(byte[] bt)
        {
            string lin = "";
            for (int i = 0; i < bt.Length; i++)
            {
                lin = lin + bt[i] + " ";
            }
            string[] ss = lin.Trim().Split(new char[] { ' ' });
            char[] c = new char[ss.Length];
            int a;
            for (int i = 0; i < c.Length; i++)
            {
                a = Convert.ToInt32(ss[i]);
                c[i] = Convert.ToChar(a);
            }

            string b = new string(c);
            return b;
        }

        /// <summary>
        /// 时间格式转换
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DateTime GetDataTime(byte[] bytes)
        {
            return DateTime.ParseExact(ByteToHexStr(bytes), "yyMMddHHmmss",
  System.Globalization.CultureInfo.CurrentCulture);
        }

        public static DateTime GetDataTime(string strdate)
        {
            return DateTime.ParseExact(strdate, "yyMMddHHmmss", System.Glob
```

```
alization.CultureInfo.CurrentCulture);
        }

        /// <summary>
        /// 获取二进制第index位的值
        /// </summary>
        /// <param name="number"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public static int getBitValue(long number, int index)
        {
            return (number & (1 << index)) > 0 ? 1 : 0;
        }

        /// <summary>
        /// 字节数组转16进制字符串
        /// </summary>
        /// <param name="byteDatas"></param>
        /// <returns></returns>
        public static string ByteToHexStr(byte[] byteDatas)
        {
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < byteDatas.Length; i++)
            {
                builder.Append(string.Format("{0:X2}", byteDatas[i]));
            }
            return builder.ToString().Trim();
        }

        public static ushort SwapUInt16(ushort v)
        {
            return (ushort)(((v & 0xff) << 8) | ((v >> 8) & 0xff));
        }

        public static uint SwapUInt32(uint v)
        {
            return (uint)(((SwapUInt16((ushort)v) & 0xffff) << 0x10) |
                          (SwapUInt16((ushort)(v >> 0x10)) & 0xffff));
        }

        /// <summary>
        /// Short转Bytes
        /// </summary>
        /// <param name="number"></param>
        /// <returns></returns>
        public static byte[] ShortToBytes(short number)
```

```
        {
            byte byte2 = (byte)(number >> 8);
            byte byte1 = (byte)(number & 255);
            byte[] bytes = new byte[2];
            bytes[0] = byte1;
            bytes[1] = byte2;
            return bytes.Reverse().ToArray();
        }

        /// <summary>
        /// 随机short
        /// </summary>
        /// <returns></returns>
        public static short RandomNumber(int min, int max)
        {
            Random rd = new Random();
            return (short)rd.Next(min, max);
        }

        /// <summary>
        /// 计算校验码
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static byte xor(List<byte> bytes)
        {
            int checksum = 0;
            foreach (byte b in bytes)
            {
                checksum ^= b;
            }
            return (byte)(checksum & 0xff);
        }

        /// <summary>
        /// 转义
        /// </summary>
        /// <param name="inBytes"></param>
        /// <returns></returns>
        public static byte[] escape(List<byte> inBytes)
        {
            List<byte> outBytes = new List<byte>();
            foreach (byte b in inBytes)
            {
                if (b == 0x7E)
                {
```

```
                    outBytes.AddRange(HexStringToBytes("7D02"));
                }
                else if (b == 0x7D)
                {
                    outBytes.AddRange(HexStringToBytes("7D01"));
                }
                else
                {
                    outBytes.Add(b);
                }
            }
            return outBytes.ToArray();
        }
    }
}
```

(3)解包工具类PacketUtil.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt705DataParser
{
    public class PacketUtil
    {

        public static byte[] decodePacket(byte[] bytes)
        {
            //查找消息尾
            byte[] msgBodyNoHeader = bytes.Skip(1).Take(bytes.Length - 1).T
oArray();
            int tailIndex = Common.BytesIndexOf(msgBodyNoHeader, 0x7E);
            if (tailIndex <= 0)
            {
                return null;
            }
            return unescape(bytes, tailIndex).ToArray();
        }
        /// <summary>
        /// 反转义
        /// </summary>
        /// <param name="bytes"></param>
```

```
        /// <param name="bodyLen"></param>
        /// <returns></returns>
        public static List<byte> unescape(byte[] bytes, int bodyLen)
        {
            int i = 0;
            List<byte> frame = new List<byte>();
            while (i < bodyLen)
            {
                byte b = bytes[i];
                if (b == 0x7D)
                {
                    byte nextByte = bytes[i + 1];
                    if (nextByte == 0x01)
                    {
                        frame.Add(0x7D);
                    }
                    else if (nextByte == 0x02)
                    {
                        frame.Add(0x7E);
                    }
                    else
                    {
                        //异常数据
                        frame.Add(b);
                        frame.Add(nextByte);
                    }
                    i += 2;
                }
                else
                {
                    frame.Add(b);
                    i++;
                }
            }
            return frame;
        }

        /// <summary>
        /// 解析定位消息体
        /// </summary>
        /// <param name="msgBodyBuf"></param>
        /// <returns></returns>
        public static LocationData parseLocationBody(byte[] msgBodyBuf)
        {
            //报警标志
            long alarmFlag = Common.SwapUInt32(BitConverter.ToUInt32(msgBod
```

```
yBuf, 0));
            //状态
            long status = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBu
f, 4));
            //纬度
            double lat = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 8)) * 0.000001;
            //经度
            double lon = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 12)) * 0.000001;
            //海拔高度,单位为米
            int altitude = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 16));
            //速度
            double speed = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 18)) * 0.1;
            //方向
            int direction = Common.SwapUInt16(BitConverter.ToUInt16(msgBody
Buf, 20));
            //定位时间
            DateTime gpsZonedDateTime = Common.GetDataTime(msgBodyBuf.Skip(
22).Take(6).ToArray());
            //根据状态位的值判断是否南纬和西经
            if (Common.getBitValue(status, 2) == 1)
            {
                lat = -lat;
            }
            if (Common.getBitValue(status, 3) == 1)
            {
                lon = -lon;
            }
            //定位状态
            int locationType = Common.getBitValue(status, 18);
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 1);
            }
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 6) > 0 ? 2 : 0;
            }
            //报警状态
            int alarm = -1;
            if (Common.getBitValue(alarmFlag, 16) == 1)
            {
                alarm = (int)AlarmTypeEnum.ALARM_1;
```

```
            }
            //后盖状态
            int backCover = Common.getBitValue(status, 7);
            //唤醒源
            long awaken = (status >> 24) & 0x0f;

            LocationData locationData = new LocationData();
            locationData.GpsTime = gpsZonedDateTime;
            locationData.Latitude = lat;
            locationData.Longitude = lon;
            locationData.LocationType = locationType;
            locationData.Speed = (int)speed;
            locationData.Direction = direction;
            locationData.Alarm = alarm;
            locationData.BackCover = backCover;
            locationData.Awaken = (int)awaken;
            //处理附加信息
            if (msgBodyBuf.Length > 28)
            {
                byte[] extraBytes = msgBodyBuf.Skip(28).Take(msgBodyBuf.Len
gth - 28).ToArray();
                parseExtraInfo(extraBytes, locationData);
            }
            return locationData;
        }

        /// <summary>
        /// 解析附加信息
        /// </summary>
        /// <param name="msgBody"></param>
        /// <param name="location"></param>
        private static void parseExtraInfo(byte[] msgBody, LocationData loc
ation)
        {
            byte[] extraInfoBuf = null;
            while (msgBody.Length > 1)
            {
                int extraInfoId = msgBody[0];
                int extraInfoLen = msgBody[1];
                if (msgBody.Length - 2 < extraInfoLen)
                {
                    break;
                }
                extraInfoBuf = msgBody.Skip(2).Take(extraInfoLen).ToArray()
;
                msgBody = msgBody.Skip(2 + extraInfoLen).Take(msgBody.Lengt
```

```
h - (2 + extraInfoLen)).ToArray();
                switch (extraInfoId)
                {
                    //锁事件
                    case 0x0B:
                        LockEvent lockEvent = new LockEvent();
                        //锁事件
                        int type = extraInfoBuf[0];
                        lockEvent.Type = type;
                        if (type == 0x01 || type == 0x02 || type == 0x03 ||
 type == 0x05 || type == 0x1E || type == 0x1F)
                        {
                            byte[] passwordBytes = extraInfoBuf.Skip(1).Tak
e(6).ToArray();
                            string password = Common.ByteToASCII(passwordBy
tes);
                            lockEvent.Password = password;
                            int unlockStatus = extraInfoBuf[7];
                            if (unlockStatus == 0xff)
                            {
                                lockEvent.UnLockStatus = 0;
                            }
                            else
                            {
                                if (unlockStatus > 0 && unlockStatus < 100)
                                {
                                    lockEvent.FenceId = unlockStatus;
                                }
                                lockEvent.UnLockStatus = 1;
                            }
                        }
                        else if (type == 0x06 || type == 0x07 || type == 0x
08 || type == 0x10 || type == 0x11 || type == 0x18 || type == 0x19 || type
== 0x20 || type == 0x28 || type == 0x29)
                        {
                            byte[] passwordBytes = extraInfoBuf.Skip(0).Tak
e(6).ToArray();
                            string password = Common.ByteToASCII(passwordBy
tes);
                            lockEvent.Password = password;
                            lockEvent.UnLockStatus = 0;
                        }
                        else if (type == 0x22)
                        {
                            long cardId = Common.SwapUInt32(BitConverter.To
UInt32(extraInfoBuf, 0));
```

```
                            if (cardId != 0)
                            {
                                lockEvent.CardNo = cardId.ToString().PadLef
t(10, '0');
                            }
                            if (extraInfoBuf.Length > 4)
                            {
                                int unlockStatus = extraInfoBuf[4];
                                if (unlockStatus == 0xff)
                                {
                                    lockEvent.UnLockStatus = 0;
                                }
                                else
                                {
                                    if (unlockStatus > 0 && unlockStatus <
100)
                                    {
                                        lockEvent.FenceId = unlockStatus;
                                    }
                                    lockEvent.UnLockStatus = 1;
                                }
                            }
                        }
                        else if (type == 0x23 || type == 0x2A || type == 0x
2B) {
                            long cardId = Common.SwapUInt32(BitConverter.To
UInt32(extraInfoBuf, 0));
                            if (cardId != 0)
                            {
                                lockEvent.CardNo = cardId.ToString().PadLef
t(10, '0');
                            }
                        }
                        location.LockEvent = lockEvent;
                        break;
                    //陀螺仪三轴数据
                    case 0x0C:
                        int x = Common.SwapUInt16(BitConverter.ToUInt16(ext
raInfoBuf, 0));
                        int y = Common.SwapUInt16(BitConverter.ToUInt16(ext
raInfoBuf, 2));
                        int z = Common.SwapUInt16(BitConverter.ToUInt16(ext
raInfoBuf, 4));
                        x = x > 32768 ? (x - 65536) : x;
                        y = y > 32768 ? (y - 65536) : y;
                        z = z > 32768 ? (z - 65536) : z;
```

```
                        location.Posture = string.Format("x:{0};y:{1};z:{2}
", x, y, z);
                        break;
                    //无线通信网络信号强度
                    case 0x30:
                        int fCellSignal = extraInfoBuf[0];
                        location.GSMSignal = fCellSignal;
                        break;
                    //卫星数
                    case 0x31:
                        int fGPSSignal = extraInfoBuf[0];
                        location.GpsSignal = fGPSSignal;
                        break;
                    //电池电量百分比
                    case 0xD4:
                        int fBattery = extraInfoBuf[0];
                        location.Battery = fBattery;
                        break;
                    //电池电压
                    case 0xD5:
                        int fVoltage = Common.SwapUInt16(BitConverter.ToUIn
t16(extraInfoBuf, 0));
                        location.Voltage = fVoltage * 0.01;
                        break;
                    case 0xF9:
                        //版本号
                        int version = Common.SwapUInt16(BitConverter.ToUInt
16(extraInfoBuf, 0));
                        location.ProtocolVersion=version;
                        break;
                    case 0xFD:
                        //小区码信息
                        int mcc = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 0));
                        location.MCC = mcc;
                        int mnc = extraInfoBuf[2];
                        location.MNC = mnc;
                        long cellId = Common.SwapUInt32(BitConverter.ToUInt
32(extraInfoBuf, 3));
                        location.CELLID = cellId;
                        int lac = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 7));
                        location.LAC = lac;
                        break;
                    case 0xFC:
                        int fenceId = extraInfoBuf[0];
```

```
                        location.FenceId=fenceId;
                        break;
                    case 0xFE:
                        long mileage = Common.SwapUInt32(BitConverter.ToUIn
t32(extraInfoBuf, 0));
                        location.Mileage = mileage;
                        break;
                    default:
                        break;
                }
            }
        }

        /// <summary>
        /// 温度解析
        /// </summary>
        /// <param name="temperatureInt"></param>
        /// <returns></returns>
        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;
        }

        /// <summary>
        /// 回复内容
        /// </summary>
        /// <param name="terminalNumArr"></param>
        /// <param name="msgFlowId"></param>
        /// <returns></returns>
        public static string replyBinaryMessage(byte[] terminalNumArr, int
msgFlowId)
        {
            List<byte> byteList = new List<byte>();
            byteList.AddRange(Common.HexStringToBytes("8001"));
            byteList.AddRange(Common.HexStringToBytes("0005"));
            byteList.AddRange(terminalNumArr);
            byteList.AddRange(Common.ShortToBytes(Common.RandomNumber(0, 65
```

```
534)));
            byteList.AddRange(Common.ShortToBytes((short)msgFlowId));
            byteList.AddRange(Common.HexStringToBytes("0200"));
            byteList.Add(0x00);
            byteList.Add(Common.xor(byteList));
            byte[] bytes = Common.escape(byteList);
            List<byte> replyList = new List<byte>();
            replyList.Add(0x7E);
            replyList.AddRange(bytes);
            replyList.Add(0x7E);
            return Common.ByteToHexStr(replyList.ToArray());
        }

        /// <summary>
        /// 授时
        /// </summary>
        /// <param name="itemList"></param>
        /// <returns></returns>
        public static string replyBASE2Message(string[] itemList)
        {
            try
            {
                string strBase2Reply = string.Format("({0},{1},{2},{3},{4},
{5})", itemList[0], itemList[1]
                        , itemList[2], itemList[3], itemList[4], DateTime.U
tcNow.ToString("yyyyMMddHHmmss"));
                return strBase2Reply;
            }
            catch (Exception e)
            {
                return "";
            }
        }
    }
}
```

2.4.返回消息及说明

（1）心跳数据
原始数据：

```
7E0002000075018042831000012 67E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据（不带锁事件）：

```
7E020000428560090010140E560000000022340042015 88F9506CA3D93000B0012000020121
0073024D40127D502017B30011E310106FE040000001BFD0900000000000000000000C060000
0000FFDFD47E
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间），
        "Latitude": 纬度（WGS84），
        "Longitude": 经度（WGS84），
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关），
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳
唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%），
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏
报警；5:出围栏报警），
        "DataLength": 数据长度（字节数），
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值，
        "Posture":姿态（x:正180度，负180度;y:正90度，负90度;z:正180度，负180度
）
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

原始数据（带锁事件）：

```
7E0200004C8560090010140E5400000000022400420158F9506CA3D93000B0012000020121
0073017D40127D502017B30011E310106FE040000001BFD0900000000000000000000B080538
3838383838650C0600000000FFC5A27E
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间），
        "Latitude": 纬度（WGS84），
        "Longitude": 经度（WGS84），
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关），
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%），
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏报警；5:出围栏报警），
        "LockEvent": {
            "Type": 事件类型（参见附件1），
            "FenceId": 事件关联的围栏ID（-1：与围栏无关），
            "UnLockStatus": 开锁状态（1：开锁成功；0：开锁失败），
            "Password": 开锁密码
            "CardNo":如果事件类型为刷卡开锁，这里表示的是卡号
        },
        "DataLength": 数据长度（字节数），
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值,
        "Posture":姿态（x:正180度，负180度;y:正90度，负90度;z:正180度，负180度）
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析
原始数据：

```
28383536303039303031303538 2C362C3030312C424153452C322C323032313037323430353
531353529
```

返回消息：

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

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复）,
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,898600421911302725 49,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）

3	JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	SIM卡的ICCID。
7	012207005620932	GSM模块的IMEI号。
8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,5, 1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,5	指令类型（BASE5）
3	1,1,1,1,1,1,1,1,1,1,1	从左至右依次为(1)锁挂绳剪断报警、(2)刷非法卡报警、(3)开锁状态保持一段时间报警、(4)指令开锁密码连续输错5次报警、(5)震动报警、(6)进区域报警、(7)出区域报警的开关、(8)低电报警（电量低于满电的30%）、(9)开后盖报警、(10)卡锁报警、(11)超速报警;(每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)

3.6.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，默认60。最长1小时，最短5秒
4	30	设备正常工作间隔，单位分钟，默认60分钟. 当该值为0时，启用持续追踪模式，设备不休眠。最长24小时，最短5分钟

3.7.查询/设置终端休眠模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,7	指令类型（BASE7）
3	1	0正常休眠模式任一中断、RTC可以唤醒，1表示在0模式上增加短信电话可以唤醒

3.8.查询/设置终端本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.9.查询/设置终端本地时差

消息体内容（DataBody）：

(700160818000,1,001,BASE,9,8613998765432,0,0,0,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,9	指令类型（BASE9）
3	8613998765432,0,0,0,0	5个监控手机号码,0表示没有设置

3.10.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名

9	123456	密码

3.11.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	则表明当前有效闹钟个数；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.12.查询/设置仅支持VIP号码

消息体内容（DataBody）：

(700160818000,1,001,BASE,40,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,40	指令类型（BASE40）
3	1	1表示仅支持VIP号码进行设备配置和查询，0表示非vip号码也可以配置和查询。默认0

3.13.查询/设置里程统计相关参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,43,10,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,43	指令类型（BASE43）
3	10	表示里程统计的速度限制值操作(即速度小于改值不统计里程,默认10km/h, 参数0-100)
4	10	表示设置里程同步值为10公里

3.14.查询/设置静态飘移处理功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,44,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,44	指令类型（BASE44）
3	1	1打开这个静态漂移处理功能，0表示关闭静态漂移处理功能

3.15.查询/设置GPS追踪定位模式功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,45,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,45	指令类型（BASE45）
3	1	1表示节能定时追踪，0 表示节能定位追踪，2表示全速追踪

3.16.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,46,5,25,6)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,46	指令类型（BASE46）
3	5	HDOP门限值
4	25	单颗卫星信噪比门限值
5	6	联合定位卫星个数门限值

3.17.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,47,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1	1: 启用监控IP服务器，0关闭监控IP服务器

3.18.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,48, 1,1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序		

号		
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1,1,1,1, 1,1,1,1, 1,1,1,1,1	从左至右依次为(1) 锁绳剪断报警、(2)刷非法卡报警、(3)长时间开锁报警、(4)开锁密码错误、(5)震动报警、(6)进区域报警、(7)出区域报警、(8)低电报警、(9)开后盖报警、(10)卡锁报警、(11)GSM信号低报警、(12)超速报警 (每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)，(13)倾斜报警，其中红色的没有相应报警

3.19.查询/设置震动检测阈值

消息体内容（DataBody）：

(700160818000,1,001,GSENS,1, 500)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GSENS,1	指令类型（GSENS1）
3	500	500：单位是mg（gsensor芯片63mg为阶梯配置，实际芯片配置值为500/63取整=7），最小63，最大值8000，默认500。当震动达到这个值，会检测到一次震动。如果值等于0，关闭该功能

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,7)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	DEBUG,7	指令类型（DEBUG7）

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15, 10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,15	指令类型（DEBUG15）
3	10	0表示马上进入睡眠，设置大于0的数表示相应时间后进入休眠

3.21.查询物联卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27, ICCID, IMSI)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,27	指令类型（DEBUG27）
3	ICCID	ICCID号码
4	IMSI	IMSI号码

3.22.查询/清除历史(盲区)数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,29, 0)

消息体内容描述：

1	700160818000	设备ID
2	DEBUG,29	指令类型（DEBUG29）
3	0	返回0则表明当前历史数据条数

3.23.查询/设置超速报警速度

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,30,120,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,30	指令类型（DEBUG30）
3	120	报警的速度临界值(默认值120)，单位为:公里
4	10	超速此时间后报警(默认值10),单位：秒

3.24.查询/设置低电报警数值

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,31,5)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,31	指令类型（DEBUG31）
3	5	报警的电量临界值；5表明当前电量百分比

3.25.查询/设置关闭看门狗作用

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,38,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,38	指令类型（DEBUG38）
3	0	返回0表明已接收该指令，并生效。

3.26.设置/查询区域详细节点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,7,8,15,1,10,11323.1234...)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,7	指令类型（GFCE7）
3	8	8表示第八个区域
4	15	15表示总点数
5	1	1表示当前页
6	10	10表示当前页的点数
7	11323.1234...	各个点的经度与纬度

3.27.清除设置的全部区域点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,8,1)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	GFCE,8	指令类型（GFCE8）
3	1	1表示删除围栏的ID

3.28.查询/设置没有关锁报警提醒时间间隔

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,2, 1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,2	指令类型（ELOCK2）
3	1	提醒时间，单位(分钟)。当开锁后没有关锁，如果此值不为0(为0无效)时间到就上报开锁提醒消息，最小值为1分钟，最大值为10分钟。默认为1分钟

3.29.终端上报动态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,3,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,3	指令类型（ELOCK3）
3	123456	上报的动态密码

3.30.设置/修改远程开锁静态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,4,666666)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,4	指令类型（ELOCK4）
3	666666	666666: 表示修改成功的密码。

3.31.凭静态或动态密码开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,5,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,5	指令类型（ELOCK5）
3	1	0: 表示密码正确，大于0表示密码错误开锁失败

3.32.查询/设置围栏外禁止开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1,2,3,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
		1表示设置了栏外禁止开锁，必须在定位状下围栏内才能开锁。2表示设置了栏外禁止开锁，有定位必须在围栏内开锁，没有定位可以

3	1	通过远程静态密码或动态密码开锁，如果设置了围栏外禁止开锁没有定位不能通过现场按键或蓝牙APP开锁。0表示没有开启围栏外禁止开锁功能，默认没有开启。
4	2	围栏ID个数：1~10有效，只支持区域围栏。其它类型的围栏不支持
5	3，4	围栏ID： 1~10有效，3,4表示只能在区域围栏ID为3,4的围栏可以开锁，1～10的ID存在对应的位置

3.33.动态密码权限处理(不对外公开)

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示使能产生动态密码；0表示关闭产生动态密码

3.34.查询/增删/开锁授权号

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,15,0,2,20,2,1,0013953759, 0013953758, 0013953757，，，，，，，，，，，，)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	0	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。

4	2	2为查询第2组数据，一共分3组，分别为1~3
5	20	20表示有20个ID号
6	2	2表示总共有2应答包
7	1	1表示当前是第1个应答包
8	0013953759, 0013953758, 0013953757，，，，，	表示该组存储的授权号列表

消息体内容（DataBody）实例2：

(700160818000,1,001,ELOCK,15,1,3,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	1	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	增加3个授权卡号成功
5	200	当前总共有200个卡

消息体内容（DataBody）实例3：

(700160818000,1,001,ELOCK,15,2,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	2	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权

3	2	号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	表示删除3个授权卡号成功。(每次最多一次性删除20)

消息体内容（DataBody）实例4：

(700160818000,1,001, ELOCK,15,3,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	3	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	0	表示删除全部授权卡号成功，目前剩余0个授权卡

3.35.现场刷卡授权模式配置指令

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,16,2,0013953759,0013953751)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	2	授权的卡个数
4	0013953759,0013953751	授权的卡号

消息体内容（DataBody）实例2：

(700160818000,1,001, ELOCK,16,0)

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	0	1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。

3.36.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.37.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）

3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.38.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.39.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.40.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.41.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT705A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT705A_V103R001	固件版本号

附件1

事件ID（16进制）	事件ID	事件描述
0x01	1	静态密码远程开锁
0x02	2	动态密码远程开锁

0x02	2	动态密码远程开锁
0x03	3	动态密码现场(蓝牙或WIFI)APP开锁
0x05	5	表示静态密码现场(蓝牙或WIFI)APP开锁
0x06	6	错误的静态密码远程开锁
0x07	7	错误的动态密码远程开锁
0x08	8	错误的动态密码现场(蓝牙或WIFI)APP开锁
0x0B	11	长时间开锁事件
0x0C	12	锁绳剪断事件
0x0D	13	关锁事件(包括自动关锁和远程关锁)
0x10	16	远程执行开锁异常，没有定位不执行开锁
0x11	17	远程执行开锁异常，有定位在围栏外不执行开锁
0x12	18	电机异常
0x18	24	蓝牙执行开锁异常，没有定位不执行开锁
0x19	25	蓝牙执行开锁异常，有定位在围栏外不执行开锁
0x1C	28	开锁并拔出锁绳
0x1E	30	短信静态密码远程开锁
0x1F	31	短信动态密码远程开锁
0x20	32	表示错误的短信动态密码
0x22	34	刷授权卡开锁事件
0x23	35	刷非法卡开锁事件
0x28	40	错误的静态密码现场(蓝牙或WIFI)APP开锁
0x29	41	错误的短信静态密码开锁
0x2A	42	RFID执行开锁异常，没有定位不执行开锁
0x2B	43	RFID执行开锁异常，有定位在围栏外不执行开锁

Java 版本

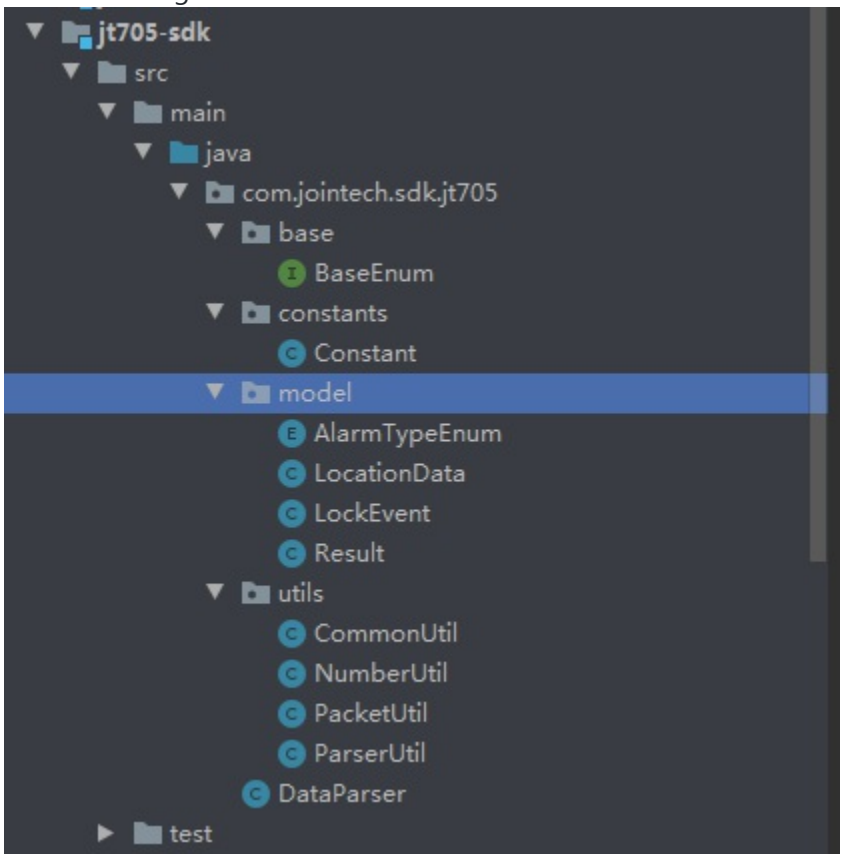
1.Jar包下载

jt705-sdk-1.0.0.jar 下载地址
如果需要Jar包开发源码，请与商务申请

2.集成开发说明

2.1.集成开发语言及框架说明

jt705-sdk-1.0.0.jar是基于Java语言，SpringBoot2.x框架，使用到了netty，fastjson，lombok，commons-lang3



BaseEnum：基础枚举
Constant：自定义常量
AlarmTypeEnum:终端报警类型枚举
LocationData：定位实体类
LockEvent：锁事件实体类
Result：结果实体类
CommonUtil：公共方法类
NumberUtil：数字操作工具类

PacketUtil：用来处理预处理数据以及分包方法封装类
ParserUtil：解析方法工具类
DataParser：解析主方法

2.2.集成说明

将jt705-sdk-1.0.0.jar引入到自己的网关程序中，引入方法如下：
在pom.xml引入jar包

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

调用jt705-sdk-1.0.0.jar，DataParser类中receiveData()方法
receiveData()方法是重载方法

```
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: 解析方法的主体</p>
 *
 * @author lenny
 * @version 1.0.1
 */
public class DataParser {
    /**
     * 解析Hex字符串原始数据
     * @param strData 16进制字符串
     * @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);
```

```
    }

    /**
     * 解析byte[]原始数据
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * 解析ByteBuf原始数据
     * @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.核心代码

ParserUtil：解析方法工具类

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

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: 解析方法工具类</p>
 * @author HyoJung
 */
public class ParserUtil {
    /**
     * 定位数据解析0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //对数据进行反转移处理
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //消息长度
        int msgLen = msg.readableBytes();
        //包头
        msg.readByte();
        //消息ID
        int msgId = msg.readUnsignedShort();
        //消息体属性
        int msgBodyAttr = msg.readUnsignedShort();
        //消息体长度
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //是否分包
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //去除消息体的基础长度
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //根据消息体长度和是否分包得出后面的包长
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
```

```
        //终端号数组
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //终端号(去除前面的0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //消息流水号
        int msgFlowId = msg.readUnsignedShort();
        //消息总包数
        int packetTotalCount = 0;
        //包序号
        int packetOrder = 0;
        //分包
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //消息体
        byte[] msgBodyArr = new byte[msgBodyLen];
        msg.readBytes(msgBodyArr);
        if(msgId==0x0200) {
            //解析消息体
            LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
            locationData.setIndex(msgFlowId);
            locationData.setDataLength(msgBodyLen);
            //校验码
            int checkCode = msg.readUnsignedByte();
            //包尾
            msg.readByte();
            //获取消息回复内容
            String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
            //定义定位数据实体类
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("Location");
            model.setDataBody(locationData);
            model.setReplyMsg(replyMsg);
            return model;
        }else {
            //定义定位数据实体类
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("heartbeat");
            model.setDataBody(null);
```

```
            return model;
        }
    }

    /**
     * 解析指令数据
     * @param in 原始数据
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

        //读指针设置到消息头
        in.markReaderIndex();
        //查找消息尾，如果未找到则继续等待下一包
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //定义定位数据实体类
        Result model = new Result();
        //包头(
        in.readByte();
        //字段列表
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //查询逗号的下标截取数据
            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;
    }

    /**
     * 解析定位消息体
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
        //报警标志
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //状态
        long status = msgBodyBuf.readUnsignedInt();
        //纬度
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //经度
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //海拔高度,单位为米
        int altitude = msgBodyBuf.readShort();
        //速度
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //方向
        int direction = msgBodyBuf.readShort();
        //定位时间
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
        //根据状态位的值判断是否南纬和西经
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
```

```
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //定位状态
        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;
        }
        //锁绳状态
        int lockRope=NumberUtil.getBitValue(status, 20);
        //锁电机状态
        int lockMotor=NumberUtil.getBitValue(status, 21);
        //锁状态(由锁绳/按钮+电机来组合判断，当锁绳/按钮为开（1）则锁状态必然为开（1）；或者电机状态为开（1）则锁状态也为开（1）；其他的则为关（0））
        int lockStatus=0;
        if(lockRope==1||lockMotor==1) {
            lockStatus=1;
        }
        int backCover=NumberUtil.getBitValue(status, 7);
        //唤醒源
        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);
        //处理附加信息
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }
```

```
    /**
     * 解析附加信息
     *
     * @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) {
                //锁事件
                case 0x0B:
                    LockEvent event=new LockEvent();
                    //锁事件
                    int type=extraInfoBuf.readUnsignedByte();
                    event.setType(type);
                    if(type==0x01||type==0x02||type==0x03||type==0x05||type
==0x1E||type==0x1F){
                        //凭密开锁
                        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){
                        //凭密开锁
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
```

```
                        event.setPassword(password);
                        event.setUnLockStatus(0);
                    }else if(type==0x22){
                        //卡号
                        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){
                        //卡号
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                    }
                    location.setLockEvent(event);
                    break;
                //陀螺仪三轴数据
                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;
                //无线通信网络信号强度
                case 0x30:
```

```
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //卫星数
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //电池电量百分比
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //电池电压
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setVoltage(fVoltage*0.01);
                    break;
                case 0xF9:
                    //版本号
                    int version=extraInfoBuf.readUnsignedShort();
                    location.setProtocolVersion(version);
                    break;
                case 0xFD:
                    //小区码信息
                    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;
            }
```

```
            }
        }

        /**
         * 报警解析
         * @param alarmFlag
         * @return
         */
        private static int parseAlarm(long alarmFlag) {
            int alarm=-1;
            //单次触发报警
            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：用来处理预处理数据以及恢复方法封装类

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

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

```java
import java.util.Random;

/**
 * 解析包预处理（进行反转义）
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * 解析消息包
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //可读长度不能小于基本长度
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //防止非法码流攻击，数据太大为异常数据
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //查找消息尾，如果未找到则继续等待下一包
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //创建ByteBuf存放反转义后的数据
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //消息头尾之间的数据进行反转义
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //反转义后的长度不能小于基本长度
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
```

```
        }
        return frame;
    }

    /**
     * 消息头、消息体、校验码中0x7D 0x02反转义为0x7E，0x7D 0x01反转义为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 {
                    //异常数据
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * 消息头、消息体、校验码中0x7E转义为0x7D 0x02，0x7D转义为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);
            }
        }
    }

    /**
     * 回复内容
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //去除包头包尾的长度
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //消息ID
            bodyBuf.writeShort(0x8001);
            //数据长度
            bodyBuf.writeShort(0x0005);
            //终端ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //生成1-65534内的随机数
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //当前消息流水号
            bodyBuf.writeShort(index);
            //回复消息流水号
            bodyBuf.writeShort(msgFlowId);
            //应答的消息ID
            bodyBuf.writeShort(0x0200);
            //应答结果
            bodyBuf.writeByte(0x00);
            //校验码
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //包头
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //读指针重置到起始位置
            bodyBuf.readerIndex(0);
            //转义
```

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

    /**
     * 授时指令回复
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //设置日期格式
            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：数字操作工具类

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

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

/**
 * 数字工具类
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
```

```
     * 坐标精度
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     * 坐标因数
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * 小数点一位精度
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     * 格式化消息ID(转成0xXXXX)
     *
     * @param msgId 消息ID
     * @return 格式化字符串
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * 格式化短数字
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
     * 转4位的十六进制字符串
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String hexStr(int num) {
```

```
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * 解析short类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 解析int类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 获取二进制第index位的值
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }
```

```java
    /**
     * bit列表转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列表转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乘法
     *
     * @param longNum
```

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

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

CommonUtil：公共方法类

```
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;

/**
 * <p>Description: 用来存储一些解析中遇到的公共方法</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * 去掉字符串最后一个字符
     * @param inStr 输入的字符串
     * @param suffix 需要去掉的字符
```

```
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * 16进制转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();
    }

    /**
     * 转换GPS时间
     *
     * @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;
    }

    /**
     * 每个字节进行异或求值
     *
     * @param buf
```

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

BaseEnum：基础枚举

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

import java.io.Serializable;

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

Constant：自定义常量

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

/**
 * 常量定义
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * 二进制消息包头
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
     * 不包含消息体的基本长度
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;
```

```
    /**
     * 消息最大长度
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * 文本消息包头
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * 文本消息包尾
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * 文本消息分隔符
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

AlarmTypeEnum：终端报警类型枚举

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

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

/**
 * 报警枚举
 * @author HyoJung
 */

public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("超速报警", "1"),
    ALARM_2("低电报警", "2"),
    ALARM_3("主机开盖报警", "3"),
    ALARM_4("进围栏报警", "4"),
    ALARM_5("出围栏报警", "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：定位实体类

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

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

import java.io.Serializable;

/**
 * <p>Description: 定位实体类</p>
 *
 * @author lenny
 * @version 1.0.1
 */
@Data
public class LocationData implements Serializable {
    /**
     * 消息体
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
```

```
     * 定位时间
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * 纬度
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * 经度
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * 定位方式
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * 速度
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * 方向
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * 里程
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * 海拔高度
     */
    @JSONField(name = "Altitude")
    public int Altitude;
    /**
     * GPS信号值
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM信号质量
```

```
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * 电量值
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * 电压
     */
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * 锁状态
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * 锁绳状态
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * 后盖状态
     */
    @JSONField(name = "BackCover")
    public int BackCover;
    /**
     * 协议版本号
     */
    @JSONField(name = "ProtocolVersion")
    public int ProtocolVersion;
    /**
     * 围栏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;
    /**
     * 锁事件
     */
    @JSONField(name = "LockEvent")
    public LockEvent LockEvent;
    /**
     * 姿态
     */
    @JSONField(name = "Posture")
    public String Posture;
    /**
     * 流水号
     */
    @JSONField(name = "Index")
    public int Index;
}
```

LockEvent：锁事件实体类

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

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

```
/**
 * 锁事件
 * @author HyoJung
 */
@Data
public class LockEvent {
    /**
     * 事件类型
     */
    @JSONField(name = "Type")
    public int Type;
    /**
     * 刷卡卡号
     */
    @JSONField(name = "CardNo")
    public String CardNo;
    /**
     * 开锁密码
     */
    @JSONField(name = "Password")
    public String Password;
    /**
     * 开锁状态(1:成功；0：失败)
     */
    @JSONField(name = "UnLockStatus")
    public int UnLockStatus=0;
    /**
     * 与开锁有关的围栏ID
     */
    @JSONField(name = "FenceId")
    public int FenceId=-1;
}
```

Result：结果实体类

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

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

import java.io.Serializable;

/**
 * 结果实体类
```

```
 * @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.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007501804283100001267E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据（不带锁事件）：

```
7E020000428560090010140E560000000022340042015 88F9506CA3D93000B0012000020121
0073024D40127D502017B30011E310106FE040000001BFD090000000000000000000C060000
0000FFDFD47E
```

返回消息：

```
{
    "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"
    }
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间）,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关）,
```

```
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%）,
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开）,
        "LockRope": 锁绳状态（0：插入；1：拔出）,
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏报警；5:出围栏报警）,
        "DataLength": 数据长度（字节数）,
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位）,
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值，
        "Posture":姿态（x:正180度，负180度;y:正90度，负90度;z:正180度，负180度）
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复）,
    "MsgType": 数据类型（Location：定位数据）
}
```

原始数据（带锁事件）：

```
7E0200004C8560090010140E5400000000022400420158F9506CA3D93000B0012000020121
0073017D40127D502017B30011E310106FE040000001BFD0900000000000000000000B080538
3838383838650C0600000000FFC5A27E
```

返回消息：

```
{
    "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"
    }
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间），
        "Latitude": 纬度（WGS84），
        "Longitude": 经度（WGS84），
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关），
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
```

```
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%）,
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开）,
        "LockRope": 锁绳状态（0：插入；1：拔出）,
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏
报警；5:出围栏报警）,
        "LockEvent": {
            "Type": 事件类型（参见附件1）,
            "FenceId": 事件关联的围栏ID（-1：与围栏无关）,
            "UnLockStatus": 开锁状态（1：开锁成功；0：开锁失败）,
            "Password": 开锁密码
            "CardNo":如果事件类型为刷卡开锁，这里表示的是卡号
        },
        "DataLength": 数据长度（字节数）,
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位）,
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值,
        "Posture":姿态（x:正180度，负180度;y:正90度，负90度;z:正180度，负180度
）
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复）,
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析

原始数据：

```
28383536303039303031303538 2C362C3030312C424153452C322C323032313037323430353
531353529
```

返回消息：

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

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复）,
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,898600421911302725 49,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	SIM卡的ICCID。
7	012207005620932	GSM模块的IMEI号。
8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为

		16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,5, 1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,5	指令类型（BASE5）
3	1,1,1,1,1,1,1,1,1,1	从左至右依次为(1)锁挂绳剪断报警、(2)刷非法卡报警、(3)开锁状态保持一段时间报警、(4)指令开锁密码连续输错5次报警、(5)震动报警、(6)进区域报警、(7)出区域报警的开关、(8)低电报警（电量低于满电的30%）、(9)开后盖报警、(10)卡锁报警、(11)超速报警;(每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)

3.6.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，默认60。最长1小时，最短5秒
4	30	设备正常工作间隔，单位分钟，默认60分钟. 当该值为0时，启用持续追踪模式，设备不休眠。最长24小时，最短5分钟

3.7.查询/设置终端休眠模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,7	指令类型（BASE7）
3	1	0正常休眠模式任一中断、RTC可以唤醒，1表示在0模式上增加短信电话可以唤醒

3.8.查询/设置终端本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.9.查询/设置终端本地时差

消息体内容（DataBody）：

(700160818000,1,001,BASE,9,8613998765432,0,0,0,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	BASE,9	指令类型（BASE9）
3	8613998765432,0,0,0,0	5个监控手机号码,0表示没有设置

3.10.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.11.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）

3	1	则表明当前有效闹钟个数；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.12.查询/设置仅支持VIP号码

消息体内容（DataBody）：

(700160818000,1,001,BASE,40,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,40	指令类型（BASE40）
3	1	1表示仅支持VIP号码进行设备配置和查询，0表示非vip号码也可以配置和查询。默认0

3.13.查询/设置里程统计相关参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,43,10,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,43	指令类型（BASE43）
3	10	表示里程统计的速度限制值操作(即速度小于改值不统计里程,默认10km/h, 参数0-100)
4	10	表示设置里程同步值为10公里

3.14.查询/设置静态飘移处理功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,44,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,44	指令类型（BASE44）
3	1	1打开这个静态漂移处理功能，0表示关闭静态漂移处理功能

3.15.查询/设置GPS追踪定位模式功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,45,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,45	指令类型（BASE45）
3	1	1表示节能定时追踪，0 表示节能定位追踪，2表示全速追踪

3.16.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,46,5,25,6)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,46	指令类型（BASE46）
3	5	HDOP门限值
4	25	单颗卫星信噪比门限值

5	6	联合定位卫星个数门限值

3.17.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,47,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1	1: 启用监控IP服务器，0关闭监控IP服务器

3.18.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,48, 1,1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1,1,1,1, 1,1,1,1, 1,1,1,1,1	从左至右依次为(1) 锁绳剪断报警、(2)刷非法卡报警、(3)长时间开锁报警、(4)开锁密码错误、(5)震动报警、(6)进区域报警、(7)出区域报警、(8)低电报警、(9)开后盖报警、(10)卡锁报警、(11)GSM信号低报警、(12)超速报警 (每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)，(13)倾斜报警，其中红色的没有相应报警

3.19.查询/设置震动检测阈值

消息体内容（DataBody）：

(700160818000,1,001,GSENS,1, 500)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GSENS,1	指令类型（GSENS1）
3	500	500：单位是mg（gsensor芯片63mg为阶梯配置，实际芯片配置值为500/63取整=7），最小63，最大值8000，默认500。当震动达到这个值，会检测到一次震动。如果值等于0，关闭该功能

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,7)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,7	指令类型（DEBUG7）

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15, 10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,15	指令类型（DEBUG15）

3	10	0表示马上进入睡眠，设置大于0的数表示相应时间后进入休眠

3.21.查询物联卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27, ICCID, IMSI)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,27	指令类型（DEBUG27）
3	ICCID	ICCID号码
4	IMSI	IMSI号码

3.22.查询/清除历史(盲区)数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,29, 0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,29	指令类型（DEBUG29）
3	0	返回0则表明当前历史数据条数

3.23.查询/设置超速报警速度

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,30,120,10)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	DEBUG,30	指令类型（DEBUG30）
3	120	报警的速度临界值(默认值120)，单位为:公里
4	10	超速此时间后报警(默认值10),单位：秒

3.24.查询/设置低电报警数值

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,31,5)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,31	指令类型（DEBUG31）
3	5	报警的电量临界值；5表明当前电量百分比

3.25.查询/设置关闭看门狗作用

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,38,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,38	指令类型（DEBUG38）
3	0	返回0表明已接收该指令，并生效。

3.26.设置/查询区域详细节点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,7,8,15,1,10,11323.1234...)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,7	指令类型（GFCE7）
3	8	8表示第八个区域
4	15	15表示总点数
5	1	1表示当前页
6	10	10表示当前页的点数
7	11323.1234...	各个点的经度与纬度

3.27.清除设置的全部区域点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,8,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,8	指令类型（GFCE8）
3	1	1表示删除围栏的ID

3.28.查询/设置没有关锁报警提醒时间间隔

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,2, 1)

消息体内容描述：

序	示例	说明

号	示例	说明
1	700160818000	设备ID
2	ELOCK,2	指令类型（ELOCK2）
3	1	提醒时间，单位(分钟)。当开锁后没有关锁，如果此值不为0(为0无效)时间到就上报开锁提醒消息，最小值为1分钟，最大值为10分钟。默认为1分钟

3.29.终端上报动态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,3,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,3	指令类型（ELOCK3）
3	123456	上报的动态密码

3.30.设置/修改远程开锁静态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,4,666666)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,4	指令类型（ELOCK4）
3	666666	666666: 表示修改成功的密码。

3.31.凭静态或动态密码开锁

(700160818000,1,001,ELOCK,5,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,5	指令类型（ELOCK5）
3	1	0: 表示密码正确，大于0表示密码错误开锁失败

3.32.查询/设置围栏外禁止开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1,2,3,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示设置了栏外禁止开锁，必须在定位状下围栏内才能开锁。2表示设置了栏外禁止开锁，有定位必须在围栏内开锁，没有定位可以通过远程静态密码或动态密码开锁，如果设置了围栏外禁止开锁没有定位不能通过现场按键或蓝牙APP开锁。0表示没有开启围栏外禁止开锁功能，默认没有开启。
4	2	围栏ID个数：1~10有效，只支持区域围栏。其它类型的围栏不支持
5	3，4	围栏ID： 1~10有效，3,4表示只能在区域围栏ID为3,4的围栏可以开锁，1～10的ID存在对应的位置

3.33.动态密码权限处理(不对外公开)

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1)

消息体内容描述：

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示使能产生动态密码；0表示关闭产生动态密码

3.34.查询/增删/开锁授权号

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,15,0,2,20,2,1,0013953759, 0013953758, 0013953757，，，，，，，，，，，，)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	0	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	2	2为查询第2组数据，一共分3组，分别为1~3
5	20	20表示有20个ID号
6	2	2表示总共有2应答包
7	1	1表示当前是第1个应答包
8	0013953759, 0013953758, 0013953757，，，，，	表示该组存储的授权号列表

消息体内容（DataBody）实例2：

(700160818000,1,001,ELOCK,15,1,3,200)

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	1	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	增加3个授权卡号成功
5	200	当前总共有200个卡

消息体内容（DataBody）实例3：

(700160818000,1,001,ELOCK,15,2,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	2	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	表示删除3个授权卡号成功。(每次最多一次性删除20)

消息体内容（DataBody）实例4：

(700160818000,1,001, ELOCK,15,3,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）

2	ELOCK,15	指令类型（ELOCK15）
3	3	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	0	表示删除全部授权卡号成功，目前剩余0个授权卡

3.35.现场刷卡授权模式配置指令

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,16,2,0013953759,0013953751)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	2	授权的卡个数
4	0013953759,0013953751	授权的卡号

消息体内容（DataBody）实例2：

(700160818000,1,001, ELOCK,16,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	0	1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。

3.36.通知设备MCU升级

消息体内容（DataBody）：

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.37.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.38.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序		

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.39.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.40.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）

3	1	和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.41.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT705A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT705A_V103R001	固件版本号

附件1

事件ID（16进制）	事件ID	事件描述
0x01	1	静态密码远程开锁
0x02	2	动态密码远程开锁
0x03	3	动态密码现场(蓝牙或WIFI)APP开锁
0x05	5	表示静态密码现场(蓝牙或WIFI)APP开锁
0x06	6	错误的静态密码远程开锁
0x07	7	错误的动态密码远程开锁
0x08	8	错误的动态密码现场(蓝牙或WIFI)APP开锁
0x0B	11	长时间开锁事件
0x0C	12	锁绳剪断事件
0x0D	13	关锁事件(包括自动关锁和远程关锁)

0x0D	13	关锁事件(包括自动关锁和远程关锁)
0x10	16	远程执行开锁异常，没有定位不执行开锁
0x11	17	远程执行开锁异常，有定位在围栏外不执行开锁
0x12	18	电机异常
0x18	24	蓝牙执行开锁异常，没有定位不执行开锁
0x19	25	蓝牙执行开锁异常，有定位在围栏外不执行开锁
0x1C	28	开锁并拔出锁绳
0x1E	30	短信静态密码远程开锁
0x1F	31	短信动态密码远程开锁
0x20	32	表示错误的短信动态密码
0x22	34	刷授权卡开锁事件
0x23	35	刷非法卡开锁事件
0x28	40	错误的静态密码现场(蓝牙或WIFI)APP开锁
0x29	41	错误的短信静态密码开锁
0x2A	42	RFID执行开锁异常，没有定位不执行开锁
0x2B	43	RFID执行开锁异常，有定位在围栏外不执行开锁

JT707A&C SDK集成开发

.NET版本

Java版本

.NET 版本

1.SDK下载

JT707SDK(JT707SDK_V2.rar) 下载地址
JT707SDK测试工具(JT707SDK_Test.rar) 下载地址
如果需要测试工具及SDK开发源码，请与商务申请

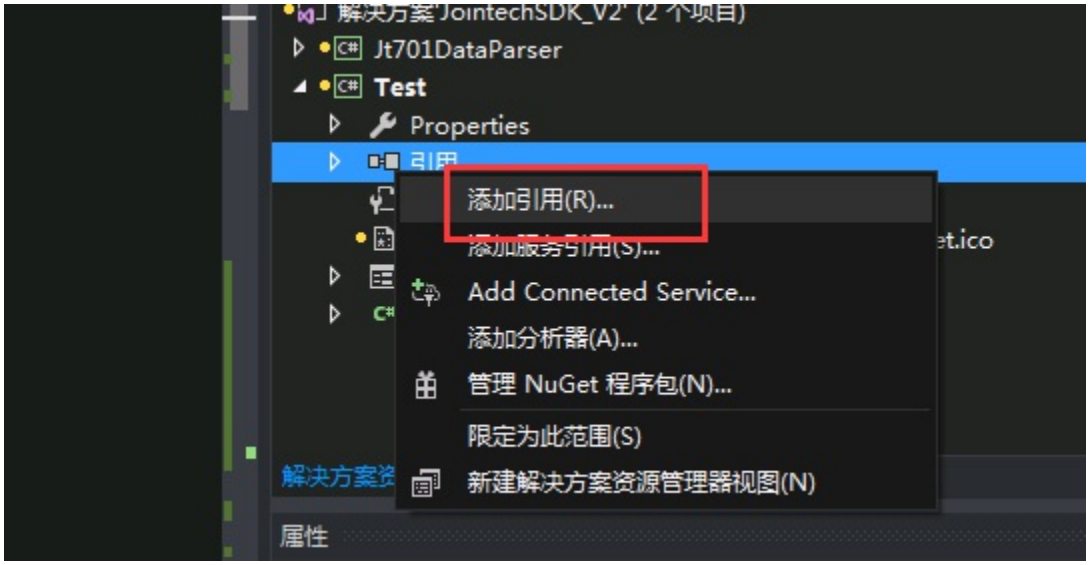
2.集成开发说明

2.1.集成开发语言及框架说明

JT707SDK（Jt707DataParser.dll）及测试工具（Test.exe）是基于C#语言，.NET Framework 4.6.1目标框架开发的。该SDK的集成开发环境也是需要依赖于C#语言以及.NET Framework 4.6.1目标框架。

2.2.集成说明

（1）将Jt707DataParser.dll引入到自己的网关程序中，引入方法如下：
鼠标右键项目下的"引用"，选择"添加引用(R)..."



选择下图中"浏览(B)..."，找到你解压JT707SDK_V2.rar后的文件夹，选中Jt707DataParser.dll与对应的Json字符串序列化/反序列化包Newtonsoft.Json.dll即可完成对SDK的引用
（2）调用Jt707DataParser.dll中的解析方法receiveData
receiveData()是一个重载方法，你可以传入16进制字符串或者byte[],一般我们网关程序接收到的设备数据是以二进制流进行传输的，所以我们这里建议使用此重载方法receiveData(byte[] bytes);
当然如果你想通过16进制字符串进行测试，也可以使用receiveData(string strData)；
strData：就是我们接收到的设备数据转成16进制后的字符串

2.3.核心代码

Jt707DataParser.dll
(1)解析类DataParser.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt707DataParser
{
    public class DataParser
    {
        private static byte[] endbytes = null;// 上一次未处理的剩余字节

        /// <summary>
        /// 解析16进制原始数据
        /// </summary>
        /// <param name="strData"></param>
        /// <returns></returns>
        public static string receiveData(string strData)
        {
            byte[] bytes = Common.HexStringToBytes(strData);
            return receiveData(bytes);
        }

        /// <summary>
        /// 解析2进制原始数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static string receiveData(byte[] bytes)
        {
            byte[] newBytes = null;
            if (endbytes != null && endbytes.Length > 0)
            {
                newBytes = new byte[endbytes.Length + bytes.Length];
                bytes = Common.CombineBytes(endbytes, bytes);
            }
            else
            {
                newBytes = new byte[bytes.Length];
                newBytes = bytes;
```

```
            }
            int i = 0;//初始化byte[]下标
            byte[] parserBytes = null;//去除正确包头之前的数据后的数据
            foreach (var item in newBytes)
            {
                if (item == 0x7E)
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserLocation(parserBytes);
                }
                else if (item == '(')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserCommand(parserBytes);
                }
                i++;
            }
            return null;
        }

        private static string parserLocation(byte[] bytes)
        {
            byte[] frame = PacketUtil.decodePacket(bytes);
            if (frame == null)
            {
                endbytes = bytes;
                return null;
            }
            string hexFrame = Common.ByteToHexStr(frame);
            //定义定位数据实体类
            Result model = new Result();
            //消息ID
            int msgId = Common.SwapUInt16(BitConverter.ToUInt16(frame, 1));
            //消息体属性
            int msgBodyAttr = Common.SwapUInt16(BitConverter.ToUInt16(frame
, 3));
            //消息体长度
            int msgBodyLen = msgBodyAttr & 0x03FF;
            byte[] terminalNumArr = frame.Skip(5).Take(6).ToArray();
            model.DeviceID = Common.ByteToHexStr(terminalNumArr).TrimStart(
'0');
            //消息流水号
```

```
            int msgFlowId = Common.SwapUInt16(BitConverter.ToUInt16(frame,
11));
            //消息体
            byte[] msgBodyArr = frame.Skip(13).Take(msgBodyLen).ToArray();
            if (msgId == 0x0200)
            {
                //解析消息体
                LocationData locationData = PacketUtil.parseLocationBody(ms
gBodyArr);
                locationData.Index = msgFlowId;
                locationData.DataLength = msgBodyLen;
                model.MsgType = "Location";
                model.DataBody = locationData;
                string replyMsg = PacketUtil.replyBinaryMessage(terminalNum
Arr, msgFlowId);
                model.ReplyMsg = replyMsg;
            }
            else
            {
                model.MsgType = "heartbeat";
            }
            return JsonConvert.SerializeObject(model);
        }

        /// <summary>
        /// 解析指令数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static string parserCommand(byte[] bytes)
        {
            Result result = new Result();
            byte[] newBytes = null;
            int tailIndex = Common.BytesIndexOf(bytes, 0x29) + 1;
            if (tailIndex > 0)
            {
                if (bytes.Length > tailIndex)
                {
                    endbytes = bytes.Skip(tailIndex).Take(bytes.Length - ta
ilIndex).ToArray();
                }
                newBytes = bytes.Skip(0).Take(tailIndex).ToArray();
            }
            else
            {
                endbytes = new byte[bytes.Length];
```

```
                endbytes = bytes;
            }
            if (newBytes != null && newBytes.Length > 0)
            {
                //转换成（）带括号的数据
                string msgAscii = Common.ByteToASCII(newBytes);
                //按逗号拆分字符串
                char[] p = new char[] { ',' };
                //返回的数组中，包含空字符串
                string[] msgArrays = msgAscii.Replace("(", "").Replace(")",
 "").Split(p, StringSplitOptions.None);
                result.DeviceID = msgArrays[0];
                string msgType = string.Empty;
                if (msgArrays.Length > 5)
                {
                    msgType = msgArrays[3] + msgArrays[4];
                }
                else
                {
                    return null;
                }
                result.MsgType = msgType;
                result.DataBody = msgAscii;
                string replyMsg = string.Empty;
                if (msgType.Equals("BASE2") && msgArrays[5].ToUpper().Equal
s("TIME"))
                {
                    replyMsg = PacketUtil.replyBASE2Message(msgArrays);
                }
                result.ReplyMsg = replyMsg;
                return JsonConvert.SerializeObject(result);
            }
            else
            {
                return null;
            }
        }
    }
}
```

（2）公共方法类：Common.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Jt707DataParser
{
    public class Common
    {

        /// <summary>
        /// 16进制格式字符串转字节数组
        /// </summary>
        /// <param name="hexString"></param>
        /// <returns></returns>
        public static byte[] HexStringToBytes(string hexString)
        {
            hexString = Regex.Replace(hexString, @".{2}", "$0 ");
            //以 ' ' 分割字符串，并去掉空字符
            string[] chars = hexString.Split(new char[] { ' ' }, StringSpli
tOptions.RemoveEmptyEntries);
            byte[] returnBytes = new byte[chars.Length];
            //逐个字符变为16进制字节数据
            for (int i = 0; i < chars.Length; i++)
            {
                returnBytes[i] = Convert.ToByte(chars[i], 16);
            }
            return returnBytes;
        }

        /// <summary>
        /// 合并数组
        /// </summary>
        /// <param name="bytes1"></param>
        /// <param name="bytes2"></param>
        /// <returns></returns>
        public static byte[] CombineBytes(byte[] bytes1, byte[] bytes2)
        {
            List<byte> tmp = new List<byte>(bytes1.Length + bytes2.Length);
            tmp.AddRange(bytes1);
            tmp.AddRange(bytes2);
            byte[] merged = tmp.ToArray();
            return merged;
        }

        /// <summary>
        /// 报告指定的 System.Byte[] 在此实例中的第一个匹配项的索引。
```

```
        /// </summary>
        /// <param name="srcBytes">被执行查找的 System.Byte[]。</param>
        /// <param name="searchBytes">要查找的 System.Byte[]。</param>
        /// <returns>如果找到该字节数组，则为 searchBytes 的索引位置；如果未找
到该字节数组，则为 -1。如果 searchBytes 为 null 或者长度为0，则返回值为 -1。</re
turns>
        public static int BytesIndexOf(byte[] srcBytes, byte searchBytes)
        {
            if (srcBytes == null) { return -1; }
            if (srcBytes.Length == 0) { return -1; }
            for (int i = 0; i < srcBytes.Length; i++)
            {
                if (srcBytes[i] == searchBytes)
                {
                    return i;
                }
            }
            return -1;
        }

        /// <summary>
        /// 16进制字符串转ASCII
        /// </summary>
        /// <param name="Data"></param>
        /// <param name="istrans"></param>
        /// <returns></returns>
        public static string HexStrToAsciiString(string Data, bool istrans)
        {
            if (istrans)
            {
                Data = Regex.Replace(Data, @"(?is)(?<=^([0-9a-f]{2})+)(?!$)
", " ");
                Data = Data.Replace("3D 00", "3D ").Replace("3D 11", "2C ")
.Replace("3D 14", "28 ").Replace("3D 15", "29 ").Replace(" ", "");
            }
            int count = Data.Length / 2;
            string strContent = "";
            char[] num = new char[count];
            for (int i = 0; i < count; i++)
            {
                string str = Data.Substring(i * 2, 2);
                byte by = Convert.ToByte(Convert.ToInt32(str, 16));
                num[i] = Convert.ToChar(by);
                strContent += num[i].ToString();
            }
            return strContent;
```

```
        }

        /// <summary>
        /// 二进制转ASCII
        /// </summary>
        /// <param name="bt"></param>
        /// <returns></returns>
        public static string ByteToASCII(byte[] bt)
        {
            string lin = "";
            for (int i = 0; i < bt.Length; i++)
            {
                lin = lin + bt[i] + " ";
            }
            string[] ss = lin.Trim().Split(new char[] { ' ' });
            char[] c = new char[ss.Length];
            int a;
            for (int i = 0; i < c.Length; i++)
            {
                a = Convert.ToInt32(ss[i]);
                c[i] = Convert.ToChar(a);
            }

            string b = new string(c);
            return b;
        }

        /// <summary>
        /// 时间格式转换
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DateTime GetDataTime(byte[] bytes)
        {
            return DateTime.ParseExact(ByteToHexStr(bytes), "yyMMddHHmmss",
 System.Globalization.CultureInfo.CurrentCulture);
        }

        public static DateTime GetDataTime(string strdate)
        {
            return DateTime.ParseExact(strdate, "yyMMddHHmmss", System.Glob
alization.CultureInfo.CurrentCulture);
        }

        /// <summary>
        /// 获取二进制第index位的值
```

```
        /// </summary>
        /// <param name="number"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public static int getBitValue(long number, int index)
        {
            return (number & (1 << index)) > 0 ? 1 : 0;
        }

        /// <summary>
        /// 字节数组转16进制字符串
        /// </summary>
        /// <param name="byteDatas"></param>
        /// <returns></returns>
        public static string ByteToHexStr(byte[] byteDatas)
        {
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < byteDatas.Length; i++)
            {
                builder.Append(string.Format("{0:X2}", byteDatas[i]));
            }
            return builder.ToString().Trim();
        }

        public static ushort SwapUInt16(ushort v)
        {
            return (ushort)(((v & 0xff) << 8) | ((v >> 8) & 0xff));
        }

        public static uint SwapUInt32(uint v)
        {
            return (uint)(((SwapUInt16((ushort)v) & 0xffff) << 0x10) |
                          (SwapUInt16((ushort)(v >> 0x10)) & 0xffff));
        }

        /// <summary>
        /// Short转Bytes
        /// </summary>
        /// <param name="number"></param>
        /// <returns></returns>
        public static byte[] ShortToBytes(short number)
        {
            byte byte2 = (byte)(number >> 8);
            byte byte1 = (byte)(number & 255);
            byte[] bytes = new byte[2];
            bytes[0] = byte1;
```

```
            bytes[1] = byte2;
            return bytes.Reverse().ToArray();
        }

        /// <summary>
        /// 随机short
        /// </summary>
        /// <returns></returns>
        public static short RandomNumber(int min, int max)
        {
            Random rd = new Random();
            return (short)rd.Next(min, max);
        }

        /// <summary>
        /// 计算校验码
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static byte xor(List<byte> bytes)
        {
            int checksum = 0;
            foreach (byte b in bytes)
            {
                checksum ^= b;
            }
            return (byte)(checksum & 0xff);
        }

        /// <summary>
        /// 转义
        /// </summary>
        /// <param name="inBytes"></param>
        /// <returns></returns>
        public static byte[] escape(List<byte> inBytes)
        {
            List<byte> outBytes = new List<byte>();
            foreach (byte b in inBytes)
            {
                if (b == 0x7E)
                {
                    outBytes.AddRange(HexStringToBytes("7D02"));
                }
                else if (b == 0x7D)
                {
                    outBytes.AddRange(HexStringToBytes("7D01"));
```

```
                }
                else
                {
                    outBytes.Add(b);
                }
            }
            return outBytes.ToArray();
        }
    }
}
```

(3)解包工具类PacketUtil.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt707DataParser
{
    public class PacketUtil
    {

        public static byte[] decodePacket(byte[] bytes)
        {
            //查找消息尾
            byte[] msgBodyNoHeader = bytes.Skip(1).Take(bytes.Length - 1).T
oArray();
            int tailIndex = Common.BytesIndexOf(msgBodyNoHeader, 0x7E);
            if (tailIndex <= 0)
            {
                return null;
            }
            return unescape(bytes, tailIndex).ToArray();
        }
        /// <summary>
        /// 反转义
        /// </summary>
        /// <param name="bytes"></param>
        /// <param name="bodyLen"></param>
        /// <returns></returns>
        public static List<byte> unescape(byte[] bytes, int bodyLen)
        {
            int i = 0;
```

```
            List<byte> frame = new List<byte>();
            while (i < bodyLen)
            {
                byte b = bytes[i];
                if (b == 0x7D)
                {
                    byte nextByte = bytes[i + 1];
                    if (nextByte == 0x01)
                    {
                        frame.Add(0x7D);
                    }
                    else if (nextByte == 0x02)
                    {
                        frame.Add(0x7E);
                    }
                    else
                    {
                        //异常数据
                        frame.Add(b);
                        frame.Add(nextByte);
                    }
                    i += 2;
                }
                else
                {
                    frame.Add(b);
                    i++;
                }
            }
            return frame;
        }

        /// <summary>
        /// 解析定位消息体
        /// </summary>
        /// <param name="msgBodyBuf"></param>
        /// <returns></returns>
        public static LocationData parseLocationBody(byte[] msgBodyBuf)
        {
            //报警标志
            long alarmFlag = Common.SwapUInt32(BitConverter.ToUInt32(msgBod
yBuf, 0));
            //状态
            long status = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBu
f, 4));
            //纬度
```

```
            double lat = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 8)) * 0.000001;
            //经度
            double lon = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 12)) * 0.000001;
            //海拔高度,单位为米
            int altitude = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 16));
            //速度
            double speed = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 18)) * 0.1;
            //方向
            int direction = Common.SwapUInt16(BitConverter.ToUInt16(msgBody
Buf, 20));
            //定位时间
            DateTime gpsZonedDateTime = Common.GetDataTime(msgBodyBuf.Skip(
22).Take(6).ToArray());
            //根据状态位的值判断是否南纬和西经
            if (Common.getBitValue(status, 2) == 1)
            {
                lat = -lat;
            }
            if (Common.getBitValue(status, 3) == 1)
            {
                lon = -lon;
            }
            //定位状态
            int locationType = Common.getBitValue(status, 18);
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 1);
            }
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 6) > 0 ? 2 : 0;
            }

            LocationData locationData = new LocationData();
            locationData.GpsTime = gpsZonedDateTime;
            locationData.Latitude = lat;
            locationData.Longitude = lon;
            locationData.LocationType = locationType;
            locationData.Speed = (int)speed;
            locationData.Direction = direction;
            //处理附加信息
            if (msgBodyBuf.Length > 28)
```

```
            {
                byte[] extraBytes = msgBodyBuf.Skip(28).Take(msgBodyBuf.Len
gth - 28).ToArray();
                parseExtraInfo(extraBytes, locationData);
            }
            return locationData;
        }

        /// <summary>
        /// 解析附加信息
        /// </summary>
        /// <param name="msgBody"></param>
        /// <param name="location"></param>
        private static void parseExtraInfo(byte[] msgBody, LocationData loc
ation)
        {
            byte[] extraInfoBuf = null;
            while (msgBody.Length > 1)
            {
                int extraInfoId = msgBody[0];
                int extraInfoLen = msgBody[1];
                if (msgBody.Length - 2 < extraInfoLen)
                {
                    break;
                }
                extraInfoBuf = msgBody.Skip(2).Take(extraInfoLen).ToArray()
;
                msgBody = msgBody.Skip(2 + extraInfoLen).Take(msgBody.Lengt
h - (2 + extraInfoLen)).ToArray();
                switch (extraInfoId)
                {
                    case 0x0F:
                        //解析温度数据
                        double temperature = -1000.0;
                        temperature = parseTemperature(Common.SwapUInt16(Bi
tConverter.ToUInt16(extraInfoBuf, 0)));
                        location.Temperature=(int)temperature;
                        break;
                    //无线通信网络信号强度
                    case 0x30:
                        int fCellSignal = extraInfoBuf[0];
                        location.GSMSignal = fCellSignal;
                        break;
                    //卫星数
                    case 0x31:
                        int fGPSSignal = extraInfoBuf[0];
```

```
                        location.GpsSignal = fGPSSignal;
                        break;
                    //电池电量百分比
                    case 0xD4:
                        int fBattery = extraInfoBuf[0];
                        location.Battery = fBattery;
                        break;
                    //电池电压
                    case 0xD5:
                        int fVoltage = Common.SwapUInt16(BitConverter.ToUIn
t16(extraInfoBuf, 0));
                        location.Voltage = fVoltage * 0.01;
                        break;
                    case 0xDA:
                        //剪绳次数
                        int fTimes = Common.SwapUInt16(BitConverter.ToUInt1
6(extraInfoBuf, 0));
                        //状态位
                        int status = extraInfoBuf[2];
                        //锁状态
                        int fLockStatus = (status & 0x01)== 1 ? 0 : 1;
                        //运动状态
                        int fRunStatus = (status & 0x02) > 0 ? 1 : 0;
                        //Sim卡状态
                        int fSimStatus = (status & 0x04) > 0 ? 1 : 0;
                        //唤醒源
                        int fWakeSource = ((status >> 3) & 0x07);
                        location.LockStatus =fLockStatus;
                        location.LockRope=fLockStatus;
                        location.UnLockTime=fTimes;
                        location.RunStatus=fRunStatus;
                        location.SimStatus=fSimStatus;
                        location.Awaken = fWakeSource;
                        break;
                    case 0xDB:
                        //定位数据发送条数
                        int sendCount = Common.SwapUInt16(BitConverter.ToUI
nt16(extraInfoBuf, 0));
                        location.SendDataCount=sendCount;
                        break;
                    case 0xF8:
                        //温度值
                        int temp = Common.SwapUInt16(BitConverter.ToUInt16(
extraInfoBuf, 0));
                        if (temp == 0xffff)
                        {
```

```
                            location.Temperature=-1000;
                        }
                        else
                        {
                            temp = (temp / 10) - 50;
                            location.Temperature=temp;
                        }
                        break;
                    case 0xFD:
                        //小区码信息
                        int mcc = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 0));
                        location.MCC = mcc;
                        int mnc = extraInfoBuf[2];
                        location.MNC = mnc;
                        long cellId = Common.SwapUInt32(BitConverter.ToUInt
32(extraInfoBuf, 3));
                        location.CELLID = cellId;
                        int lac = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 7));
                        location.LAC = lac;
                        break;
                    case 0xFE:
                        long mileage = Common.SwapUInt32(BitConverter.ToUIn
t32(extraInfoBuf, 0));
                        location.Mileage = mileage;
                        break;
                    default:
                        break;
                }
            }
        }

        /// <summary>
        /// 温度解析
        /// </summary>
        /// <param name="temperatureInt"></param>
        /// <returns></returns>
        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;
        }

        /// <summary>
        /// 回复内容
        /// </summary>
        /// <param name="terminalNumArr"></param>
        /// <param name="msgFlowId"></param>
        /// <returns></returns>
        public static string replyBinaryMessage(byte[] terminalNumArr, int
msgFlowId)
        {
            List<byte> byteList = new List<byte>();
            byteList.AddRange(Common.HexStringToBytes("8001"));
            byteList.AddRange(Common.HexStringToBytes("0005"));
            byteList.AddRange(terminalNumArr);
            byteList.AddRange(Common.ShortToBytes(Common.RandomNumber(0, 65
534)));
            byteList.AddRange(Common.ShortToBytes((short)msgFlowId));
            byteList.AddRange(Common.HexStringToBytes("0200"));
            byteList.Add(0x00);
            byteList.Add(Common.xor(byteList));
            byte[] bytes = Common.escape(byteList);
            List<byte> replyList = new List<byte>();
            replyList.Add(0x7E);
            replyList.AddRange(bytes);
            replyList.Add(0x7E);
            return Common.ByteToHexStr(replyList.ToArray());
        }

        /// <summary>
        /// 授时
        /// </summary>
        /// <param name="itemList"></param>
        /// <returns></returns>
        public static string replyBASE2Message(string[] itemList)
        {
            try
            {
                string strBase2Reply = string.Format("({0},{1},{2},{3},{4},
{5})", itemList[0], itemList[1]
                        , itemList[2], itemList[3], itemList[4], DateTime.U
tcNow.ToString("yyyyMMddHHmmss"));
```

```
                return strBase2Reply;
            }
            catch (Exception e)
            {
                return "";
            }
        }
    }
}
```

2.4.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007701912039 15FFF2E47E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据：

```
7E0200004777019120391500 0D000000000104100201588F7D0206CA3F580000001C0001210
722010602300 11F310106D40164D5020050DA03000105DB02000CDC0400000000FD0901CC00
000010922866F80203421B7E
```

返回消息：

```
{
    "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"
    }
```


返回消息描述

```
{
    "DeviceID": 终端ID,
    "DataBody": {
        "Index": 数据流水号,
        "DataLength": 数据长度，字节数,
        "GpsTime": 定位数据时间（UTC时间）,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "Temperature": 温度值，33标识的当前温度是33℃,
        "UnLockTime": 剪绳开锁次数,
        "RunStatus": 运动状态（1：运动；0：静止）,
        "Awaken": 0:RTC上报  1：剪绳上报，2：插绳上报  3：开盖上报  4：关盖上报
  5：充电/配置/模拟剪绳上报,
        "SimStatus": SIM卡类型 0：eSIM卡；1：Micro SIM卡槽,
        "Direction": 正北为0，顺时针0-359,
        "Battery": 电量值（0~100%）,
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
```

```
            "Speed": 速度，单位km/h,
            "LockStatus": 锁状态（0：关；1：开），
            "LockRope": 锁绳状态（0：插入；1：拔出），
            "Mileage": 里程值，单位km,
            "MCC": 小区码信息MCC,
            "MNC": 小区码信息MNC,
            "LAC": 小区码信息LAC,
            "CELLID": 小区码信息CI,
            "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
            "SendDataCount": 发送数据的条数,
            "GSMSignal": GSM信号值
        },
        "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
        "MsgType": 数据类型（Location：定位数据）
    }
```

（3）指令数据解析
原始数据：

```
28373730313931323033393036 2c312c3030312c424153452c362c352c33302c3529
```

返回消息：

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

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复），
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,
89860042191130272549,12345678965433224653,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	20150418_G300	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	eSIM卡的CCID。
7	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.恢复终端出厂默认值

消息体内容（DataBody）：

(700160818000,1,001,BASE,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）
3	1	1表示全部信息恢复为出厂默认值；2表示除通信主从IP外，其它信息全部恢复为出厂默认值；3表示除通信主从IP、VIP号码外，其它信息全部恢复为出厂默认值

3.4.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔(静止态)，以分钟为单位。此值默认为30分钟。最小值为30，最大值为1440（24小时）。
4	30	固定为30，未使用
5	30	上传间隔(运动态)，以分钟为单位。此值默认为30分钟。最小值为30，最大值为1440（24小时）。

3.5.时差设置

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.6.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.7.查询/设置关闭基站定位功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,29, 1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,29	指令类型（BASE29）
3	1	1表示开启基站定位功能，0关闭基站定位功能。

3.8.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001, BASE,32,1,8:9:10,12:13:14)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	1则表明当前闹钟有效，0表明当前闹钟无效。
4	8:9:10	8： 设置一个设备闹钟的小时；9： 设置一个设备闹钟的分钟； 10：设置一个设备闹钟的秒钟
5	12:13:14	12： 设置一个设备闹钟的小时；13： 设置一个设备闹钟的分钟； 14：设置一个设备闹钟的秒钟

3.9.查询/设置追踪模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,33,1,5,43200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	BASE,33	指令类型（BASE33）
3	1	1表示有效追踪模式，0表示失效追踪模式
4	5	5表示追踪上传间隔，分钟为单位
5	43200	43200表示追踪时间段，分钟为单位

3.10.查询/设置当前simCard

消息体内容（DataBody）示例1：

(700160818000,1,001,BASE,36,2)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,36	指令类型（BASE36）
3	2	表示当前卡为卡2

消息体内容（DataBody）示例2：
(700160818000,1,001,BASE,36,454030220234116,89852031600002341160,454030220234117,89852031600002341161)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,36	指令类型（BASE36）
3	454030220234116	当前卡1的IMSI
4	89852031600002341160	当前卡1的ICCID
5	454030220234117	当前卡2的IMSI
6	89852031600002341161	当前卡2的ICCID

3.11.查询/设置进行 OTA的FTP服务器的IP地址（或域名）和端口号

消息体内容（DataBody）：

(700160818000,1,001,BASE,37,1,ftpupdate.jointcontrols.cn,2021)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,37	指令类型（BASE37）
3	1	1表示设置，0表示查询(如果查询后面可以不接地址和断口号)
4	ftpupdate.jointcontrols.cn	OTA的FTP服务器的IP地址
5	2021	OTA的FTP服务器的端口

3.12.通知设备无线模块升级

消息体内容（DataBody）：

(2011091502,1,001,DEBUG,13,190508,180405)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,13	指令类型（DEBUG13）
3	190508	请求升级的软件日期版本
4	180405	软件当前正在运行的软件日期版本

3.13.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.14.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.15.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.16.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.17.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.

4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.18.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT707A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT707A_V103R001	固件版本号

Java 版本

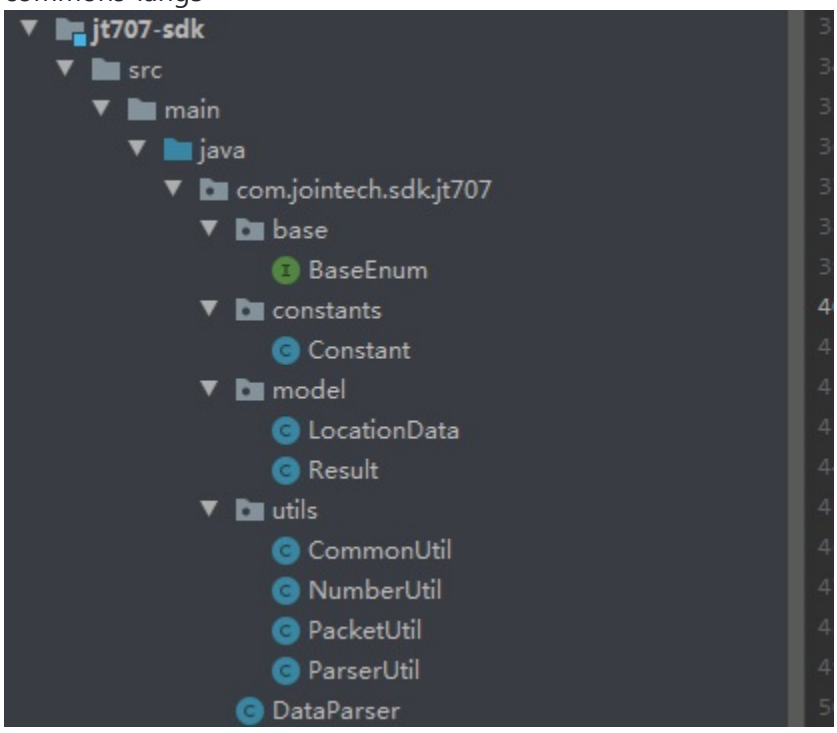
1.Jar包下载

jt707-sdk-1.0.0.jar 下载地址
如果需要Jar包开发源码，请与商务申请

2.集成开发说明

2.1.集成开发语言及框架说明

jt707-sdk-1.0.0.jar是基于Java语言，SpringBoot2.x框架，使用到了netty，fastjson，lombok，commons-lang3



BaseEnum：基础枚举
Constant：自定义常量
LocationData：定位实体类
Result：结果实体类
CommonUtil：公共方法类
NumberUtil：数字操作工具类
PacketUtil：用来处理预处理数据以及分包方法封装类
ParserUtil：解析方法工具类
DataParser：解析主方法

2.2.集成说明

将jt707-sdk-1.0.0.jar引入到自己的网关程序中，引入方法如下：
在pom.xml引入jar包

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

调用jt707-sdk-1.0.0.jar，DataParser类中receiveData()方法
receiveData()方法是重载方法

```
    /**
     * 解析Hex字符串原始数据
     * @param strData 16进制字符串
     * @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);
    }

    /**
     * 解析byte[]原始数据
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * 解析ByteBuf原始数据
     * @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.核心代码

ParserUtil：解析方法工具类

```
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: 解析方法工具类</p>
 * @author HyoJung
 * @date 20210526
 */
public class ParserUtil {
    /**
     * 定位数据解析0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //对数据进行反转移处理
```

```
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //消息长度
        int msgLen = msg.readableBytes();
        //包头
        msg.readByte();
        //消息ID
        int msgId = msg.readUnsignedShort();
        //消息体属性
        int msgBodyAttr = msg.readUnsignedShort();
        //消息体长度
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //是否分包
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //去除消息体的基础长度
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //根据消息体长度和是否分包得出后面的包长
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
        //终端号数组
        byte[] terminalNumArr = new byte[6];
        msg.readBytes(terminalNumArr);
        //终端号(去除前面的0)
        String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
        //消息流水号
        int msgFlowId = msg.readUnsignedShort();
        //消息总包数
        int packetTotalCount = 0;
        //包序号
        int packetOrder = 0;
        //分包
        if (multiPacket) {
            packetTotalCount = msg.readShort();
            packetOrder = msg.readShort();
        }
        //消息体
        byte[] msgBodyArr = new byte[msgBodyLen];
        msg.readBytes(msgBodyArr);
        if(msgId==0x0200) {
            //解析消息体
            LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
```

```
            locationData.setIndex(msgFlowId);
            locationData.setDataLength(msgBodyLen);
            //校验码
            int checkCode = msg.readUnsignedByte();
            //包尾
            msg.readByte();
            //获取消息回复内容
            String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
 sgFlowId);
            //定义定位数据实体类
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("Location");
            model.setDataBody(locationData);
            model.setReplyMsg(replyMsg);
            return model;
        }else {
            //定义定位数据实体类
            Result model = new Result();
            model.setDeviceID(terminalNumber);
            model.setMsgType("heartbeat");
            model.setDataBody(null);
            return model;
        }
    }

    /**
     * 解析指令数据
     * @param in 原始数据
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

        //读指针设置到消息头
        in.markReaderIndex();
        //查找消息尾，如果未找到则继续等待下一包
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //定义定位数据实体类
        Result model = new Result();
        //包头(
        in.readByte();
        //字段列表
```

```
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //查询逗号的下标截取数据
            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;
    }

    /**
     * 解析定位消息体
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
        //报警标志
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //状态
        long status = msgBodyBuf.readUnsignedInt();
        //纬度
```

```
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //经度
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //海拔高度,单位为米
        int altitude = msgBodyBuf.readShort();
        //速度
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //方向
        int direction = msgBodyBuf.readShort();
        //定位时间
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
        //根据状态位的值判断是否南纬和西经
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //定位状态
        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);
        //处理附加信息
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
```

```
        }
        return locationData;
    }

    /**
     * 解析附加信息
     *
     * @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:
                    //解析温度数据
                    double temperature = -1000.0;
                    temperature = parseTemperature(extraInfoBuf.readShort()
);
                    location.setTemperature((int)temperature);
                    break;
                //无线通信网络信号强度
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //卫星数
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //电池电量百分比
                case 0xD4:
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //电池电压
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
```

```
                    location.setBattery(fVoltage*0.01);
                    break;
                case 0xDA:
                    //剪绳次数
                    int fTimes =extraInfoBuf.readUnsignedShort();
                    //状态位
                    int status =extraInfoBuf.readUnsignedByte();
                    //锁状态
                    int fLockStatus=(status&0b0000_0001)==1?0:1;
                    //运动状态
                    int fRunStatus=(status&0b0000_0010)>0?1:0;
                    //Sim卡状态
                    int fSimStatus=(status&0b0000_0100)>0?1:0;
                    //唤醒源
                    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:
                    //定位数据发送条数
                    int sendCount=extraInfoBuf.readUnsignedShort();
                    location.setSendDataCount(sendCount);
                    break;
                case 0xDC:
                    //Debug上网状态
                    byte[] debugArr = new byte[4];
                    extraInfoBuf.readBytes(debugArr);
                    String strDebug=ByteBufUtil.hexDump(debugArr);
                    break;
                case 0xF8:
                    //温度值
                    int temp=extraInfoBuf.readUnsignedShort();
                    if(temp==0xffff){
                        location.setTemperature(-1000);
                    }else {
                        temp=(temp / 10)-50;
                        location.setTemperature(temp);
                    }
                    break;
                case 0xF9:
                    //版本号
                    int version=extraInfoBuf.readUnsignedShort();
```

```
                    break;
                case 0xFD:
                    //小区码信息
                    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;
            }
        }
    }

    /**
     * 温度解析
     * @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：用来处理预处理数据以及恢复方法封装类

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

import com.jointech.sdk.jt707.constants.Constant;
```

```java
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;

/**
 * 解析包预处理（进行反转义）
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * 解析消息包
     *
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //可读长度不能小于基本长度
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //防止非法码流攻击，数据太大为异常数据
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //查找消息尾，如果未找到则继续等待下一包
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //创建ByteBuf存放反转义后的数据
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
```

```
        //消息头尾之间的数据进行反转义
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //反转义后的长度不能小于基本长度
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
        return frame;
    }

    /**
     * 消息头、消息体、校验码中0x7D 0x02反转义为0x7E，0x7D 0x01反转义为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 {
                    //异常数据
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * 消息头、消息体、校验码中0x7E转义为0x7D 0x02，0x7D转义为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);
            }
        }
    }

    /**
     * 回复内容
     * @param terminalNumArr
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //去除包头包尾的长度
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //消息ID
            bodyBuf.writeShort(0x8001);
            //数据长度
            bodyBuf.writeShort(0x0005);
            //终端ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //生成1-65534内的随机数
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //当前消息流水号
            bodyBuf.writeShort(index);
            //回复消息流水号
            bodyBuf.writeShort(msgFlowId);
            //应答的消息ID
            bodyBuf.writeShort(0x0200);
            //应答结果
```

```
            bodyBuf.writeByte(0x00);
            //校验码
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //包头
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //读指针重置到起始位置
            bodyBuf.readerIndex(0);
            //转义
            PacketUtil.escape(replyBuf, bodyBuf);
            //包尾
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }

    /**
     * 授时指令回复
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //设置日期格式
            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：数字操作工具类

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

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

```java
import java.util.List;

/**
 * 数字工具类
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     * 坐标精度
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

    /**
     * 坐标因数
     */
    public static final BigDecimal COORDINATE_FACTOR = new BigDecimal("1000
000");

    /**
     * 小数点一位精度
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     * 格式化消息ID(转成0xXXXX)
     *
     * @param msgId 消息ID
     * @return 格式化字符串
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * 格式化短数字
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
```

```
    }

    /**
     * 转4位的十六进制字符串
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * 解析short类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 解析int类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 获取二进制第index位的值
```

```
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

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

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

CommonUtil：公共方法类

```
    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: 用来存储一些解析中遇到的公共方法</p>
     *
```

```
 * @author Lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * 去掉字符串最后一个字符
     * @param inStr 输入的字符串
     * @param suffix 需要去掉的字符
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
     * 16进制转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();
    }

    /**
     * 转换GPS时间
     *
     * @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;
    }

    /**
     * 每个字节进行异或求值
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

BaseEnum：基础枚举

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

import java.io.Serializable;

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

Constant：自定义常量

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

/**
 * 常量定义
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
```

```
    /**
     * 二进制消息包头
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
     * 不包含消息体的基本长度
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;

    /**
     * 消息最大长度
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * 文本消息包头
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
     * 文本消息包尾
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * 文本消息分隔符
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

LocationData：定位实体类

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

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

import java.io.Serializable;

/**
 * <p>Description: 定位实体类</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
```

```
 */
@Data
public class LocationData implements Serializable {
    /**
     * 消息体
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * 定位时间
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * 纬度
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * 经度
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * 定位方式
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * 速度
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * 方向
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * 里程
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * GPS信号值
     */
```

```
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM信号质量
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * 电量值
     */
    @JSONField(name = "Battery")
    public double Battery;
    /**
     * 电压
     */
    @JSONField(name = "Voltage")
    public int Voltage;
    /**
     * 锁状态
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * 锁绳状态
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * 剪绳次数
     */
    @JSONField(name = "UnLockTime")
    public int UnLockTime;
    /**
     * 运动状态
     */
    @JSONField(name = "RunStatus")
    public int RunStatus;
    /**
     * Sim卡类型
     */
    @JSONField(name = "SimStatus")
    public int SimStatus;
    /**
     * 定位数据发送条数
     */
    @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;
    /**
     * 流水号
     */
    @JSONField(name = "Index")
    public int Index;
    /**
     * 温度值
     */
    @JSONField(name = "Temperature")
    public int Temperature=-1000;
}
```

Result：结果实体类

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

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

```
import java.io.Serializable;

/**
 * 结果实体类
 * @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.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007701912039 15FFF2E47E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据：

```
7E0200004777019120391500 0D000000000104100201588F7D0206CA3F580000001C0001210
722010602300 11F310106D40164D5020050DA03000105DB02000CDC0400000000FD0901CC00
000010922866F80203421B7E
```

返回消息：

```
{
    "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"
}
```


返回消息描述

```
{
    "DeviceID": 终端ID,
    "DataBody": {
        "Index": 数据流水号,
        "DataLength": 数据长度，字节数,
        "GpsTime": 定位数据时间（UTC时间）,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "Temperature": 温度值，33标识的当前温度是33℃,
        "UnLockTime": 剪绳开锁次数,
```

```
        "RunStatus": 运动状态（1：运动；0：静止），
        "Awaken": 0:RTC上报  1：剪绳上报，2：插绳上报  3：开盖上报  4：关盖上报
    5：充电/配置/模拟剪绳上报，
        "SimStatus": SIM卡类型 0：eSIM卡；1：Micro SIM卡槽，
        "Direction": 正北为0，顺时针0-359，
        "Battery": 电量值（0~100%），
        "GpsSignal": GPS当前接收到的卫星颗数，
        "Voltage": 电压值，单位V，
        "Speed": 速度，单位km/h，
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
        "Mileage": 里程值，单位km，
        "MCC": 小区码信息MCC，
        "MNC": 小区码信息MNC，
        "LAC": 小区码信息LAC，
        "CELLID": 小区码信息CI，
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "SendDataCount": 发送数据的条数，
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析

原始数据：

```
28373730313931323033393036 2c312c3030312c424153452c362c352c33302c3529
```

返回消息：

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

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复），
```

```
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(700160818000,1,001,BASE,1,20150418_G300,0,BeiHuan,1137B03SIM900M64_ST_MMS,
89860042191130272549,123456789654332246 53,012207005620932,460,00,4243,6877)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	20150418_G300	当前设备的版本号。
4	0,BeiHuan	前面的0表示英文，1表示其它语言的Unicode码，ASCII码表示，如：报警62A5 8B66上传是8个字节。此处为英语，名称为BeiHuan
5	1137B03SIM900M64_ST_MMS	GSM模块版本。
6	89860042191130272549	eSIM卡的CCID。
7	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.恢复终端出厂默认值

消息体内容（DataBody）：

(700160818000,1,001,BASE,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）
3	1	1表示全部信息恢复为出厂默认值；2表示除通信主从IP外，其它信息全部恢复为出厂默认值；3表示除通信主从IP、VIP号码外，其它信息全部恢复为出厂默认值

3.4.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔(静止态)，以分钟为单位。此值默认为30分钟。最小值为30，最大值为1440（24小时）。

4	30	固定为30，未使用
5	30	上传间隔(运动态)，以分钟为单位。此值默认为30分钟。最小值为30，最大值为1440（24小时）。

3.5.时差设置

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.6.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名

9	123456	密码

3.7.查询/设置关闭基站定位功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,29, 1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,29	指令类型（BASE29）
3	1	1表示开启基站定位功能，0关闭基站定位功能。

3.8.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001, BASE,32,1,8:9:10,12:13:14)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	1则表明当前闹钟有效，0表明当前闹钟无效。
4	8:9:10	8： 设置一个设备闹钟的小时；9： 设置一个设备闹钟的分钟； 10：设置一个设备闹钟的秒钟
5	12:13:14	12： 设置一个设备闹钟的小时；13： 设置一个设备闹钟的分钟； 14：设置一个设备闹钟的秒钟

3.9.查询/设置追踪模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,33,1,5,43200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,33	指令类型（BASE33）
3	1	1表示有效追踪模式，0表示失效追踪模式
4	5	5表示追踪上传间隔，分钟为单位
5	43200	43200表示追踪时间段，分钟为单位

3.10.查询/设置当前simCard

消息体内容（DataBody）示例1：

(700160818000,1,001,BASE,36,2)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,36	指令类型（BASE36）
3	2	表示当前卡为卡2

消息体内容（DataBody）示例2：
(700160818000,1,001,BASE,36,454030220234116,89852031600002341160,454030220234117,89852031600002341161)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,36	指令类型（BASE36）
3	454030220234116	当前卡1的IMSI

4	89852031600002341160	当前卡1的ICCID
5	454030220234117	当前卡2的IMSI
6	89852031600002341161	当前卡2的ICCID

3.11.查询/设置进行 OTA的FTP服务器的IP地址（或域名）和端口号

消息体内容（DataBody）：

(700160818000,1,001,BASE,37,1,ftpupdate.jointcontrols.cn,2021)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,37	指令类型（BASE37）
3	1	1表示设置，0表示查询(如果查询后面可以不接地址和断口号)
4	ftpupdate.jointcontrols.cn	OTA的FTP服务器的IP地址
5	2021	OTA的FTP服务器的端口

3.12.通知设备无线模块升级

消息体内容（DataBody）：

(2011091502,1,001,DEBUG,13,190508,180405)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,13	指令类型（DEBUG13）
3	190508	请求升级的软件日期版本
4	180405	软件当前正在运行的软件日期版本

3.13.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.14.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.15.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.16.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.17.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.18.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT707A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT707A_V103R001	固件版本号

JT709A&B&C SDK集成开发

.NET版本

Java版本

.NET 版本

1.SDK下载

JT709SDK(JT709SDK_V2.rar) 下载地址
JT709SDK测试工具(JT709SDK_Test.rar) 下载地址
如果需要测试工具及SDK开发源码，请与商务申请

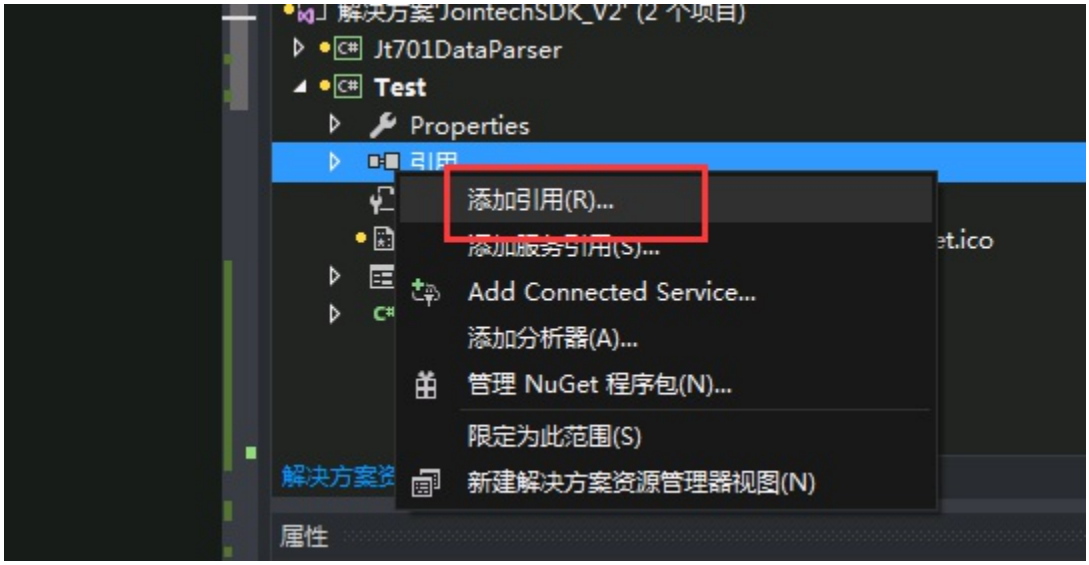
2.集成开发说明

2.1.集成开发语言及框架说明

JT709SDK（Jt709DataParser.dll）及测试工具（Test.exe）是基于C#语言，.NET Framework 4.6.1目标框架开发的。该SDK的集成开发环境也是需要依赖于C#语言以及.NET Framework 4.6.1目标框架。

2.2.集成说明

（1）将Jt709DataParser.dll引入到自己的网关程序中，引入方法如下：
鼠标右键项目下的"引用"，选择"添加引用(R)..."



选择下图中"浏览(B)..."，找到你解压JT709SDK_V2.rar后的文件夹，选中Jt709DataParser.dll与对应的Json字符串序列化/反序列化包Newtonsoft.Json.dll即可完成对SDK的引用
（2）调用Jt709DataParser.dll中的解析方法receiveData
receiveData()是一个重载方法，你可以传入16进制字符串或者byte[],一般我们网关程序接收到的设备数据是以二进制流进行传输的，所以我们这里建议使用此重载方法receiveData(byte[] bytes);
当然如果你想通过16进制字符串进行测试，也可以使用receiveData(string strData)；
strData：就是我们接收到的设备数据转成16进制后的字符串

2.3.核心代码

Jt709DataParser.dll
(1)解析类DataParser.cs

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt709DataParser
{
    public class DataParser
    {
        private static byte[] endbytes = null;// 上一次未处理的剩余字节

        /// <summary>
        /// 解析16进制原始数据
        /// </summary>
        /// <param name="strData"></param>
        /// <returns></returns>
        public static string receiveData(string strData)
        {
            byte[] bytes = Common.HexStringToBytes(strData);
            return receiveData(bytes);
        }

        /// <summary>
        /// 解析2进制原始数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static string receiveData(byte[] bytes)
        {
            byte[] newBytes = null;
            if (endbytes != null && endbytes.Length > 0)
            {
                newBytes = new byte[endbytes.Length + bytes.Length];
                bytes = Common.CombineBytes(endbytes, bytes);
            }
            else
            {
                newBytes = new byte[bytes.Length];
                newBytes = bytes;
```

```
            }
            int i = 0;//初始化byte[]下标
            byte[] parserBytes = null;//去除正确包头之前的数据后的数据
            foreach (var item in newBytes)
            {
                if (item == 0x7E)
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserLocation(parserBytes);
                }
                else if (item == '(')
                {
                    parserBytes = new byte[newBytes.Length - i];
                    parserBytes = newBytes.Skip(i).Take(newBytes.Length - i
).ToArray();
                    return parserCommand(parserBytes);
                }
                i++;
            }
            return null;
        }

        private static string parserLocation(byte[] bytes)
        {
            byte[] frame = PacketUtil.decodePacket(bytes);
            if (frame == null)
            {
                endbytes = bytes;
                return null;
            }
            //定义定位数据实体类
            Result model = new Result();
            //消息ID
            int msgId = Common.SwapUInt16(BitConverter.ToUInt16(frame, 1));
            //消息体属性
            int msgBodyAttr = Common.SwapUInt16(BitConverter.ToUInt16(frame
, 3));
            //消息体长度
            int msgBodyLen = msgBodyAttr & 0x03FF;
            byte[] terminalNumArr = frame.Skip(5).Take(6).ToArray();
            model.DeviceID = Common.ByteToHexStr(terminalNumArr).TrimStart(
'0');
            //消息流水号
            int msgFlowId = Common.SwapUInt16(BitConverter.ToUInt16(frame,
```

```
11));
            //消息体
            byte[] msgBodyArr = frame.Skip(13).Take(msgBodyLen).ToArray();
            if (msgId == 0x0200)
            {
                //解析消息体
                LocationData locationData = PacketUtil.parseLocationBody(ms
gBodyArr);
                locationData.Index = msgFlowId;
                locationData.DataLength = msgBodyLen;
                model.MsgType = "Location";
                model.DataBody = locationData;
                string replyMsg = PacketUtil.replyBinaryMessage(terminalNum
Arr, msgFlowId);
                model.ReplyMsg = replyMsg;
            }
            else
            {
                model.MsgType = "heartbeat";
            }
            return JsonConvert.SerializeObject(model);
        }

        /// <summary>
        /// 解析指令数据
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        private static string parserCommand(byte[] bytes)
        {
            Result result = new Result();
            byte[] newBytes = null;
            int tailIndex = Common.BytesIndexOf(bytes, 0x29) + 1;
            if (tailIndex > 0)
            {
                if (bytes.Length > tailIndex)
                {
                    endbytes = bytes.Skip(tailIndex).Take(bytes.Length - ta
ilIndex).ToArray();
                }
                newBytes = bytes.Skip(0).Take(tailIndex).ToArray();
            }
            else
            {
                endbytes = new byte[bytes.Length];
                endbytes = bytes;
```

```
            }
            if (newBytes != null && newBytes.Length > 0)
            {
                //转换成（）带括号的数据
                string msgAscii = Common.ByteToASCII(newBytes);
                //按逗号拆分字符串
                char[] p = new char[] { ',' };
                //返回的数组中，包含空字符串
                string[] msgArrays = msgAscii.Replace("(", "").Replace(")",
 "").Split(p, StringSplitOptions.None);
                result.DeviceID = msgArrays[0];
                string msgType = string.Empty;
                if (msgArrays.Length > 5)
                {
                    msgType = msgArrays[3] + msgArrays[4];
                }
                else
                {
                    return null;
                }
                result.MsgType = msgType;
                result.DataBody = msgAscii;
                string replyMsg = string.Empty;
                if (msgType.Equals("BASE2") && msgArrays[5].ToUpper().Equal
s("TIME"))
                {
                    replyMsg = PacketUtil.replyBASE2Message(msgArrays);
                }
                result.ReplyMsg = replyMsg;
                return JsonConvert.SerializeObject(result);
            }
            else
            {
                return null;
            }
        }
    }
}
```

（2）公共方法类：Common.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Jt709DataParser
{
    public class Common
    {

        /// <summary>
        /// 16进制格式字符串转字节数组
        /// </summary>
        /// <param name="hexString"></param>
        /// <returns></returns>
        public static byte[] HexStringToBytes(string hexString)
        {
            hexString = Regex.Replace(hexString, @".{2}", "$0 ");
            //以 ' ' 分割字符串，并去掉空字符
            string[] chars = hexString.Split(new char[] { ' ' }, StringSpli
tOptions.RemoveEmptyEntries);
            byte[] returnBytes = new byte[chars.Length];
            //逐个字符变为16进制字节数据
            for (int i = 0; i < chars.Length; i++)
            {
                returnBytes[i] = Convert.ToByte(chars[i], 16);
            }
            return returnBytes;
        }

        /// <summary>
        /// 合并数组
        /// </summary>
        /// <param name="bytes1"></param>
        /// <param name="bytes2"></param>
        /// <returns></returns>
        public static byte[] CombineBytes(byte[] bytes1, byte[] bytes2)
        {
            List<byte> tmp = new List<byte>(bytes1.Length + bytes2.Length);
            tmp.AddRange(bytes1);
            tmp.AddRange(bytes2);
            byte[] merged = tmp.ToArray();
            return merged;
        }

        /// <summary>
        /// 报告指定的 System.Byte[] 在此实例中的第一个匹配项的索引。
        /// </summary>
```

```
        /// <param name="srcBytes">被执行查找的 System.Byte[]。</param>
        /// <param name="searchBytes">要查找的 System.Byte[]。</param>
        /// <returns>如果找到该字节数组，则为 searchBytes 的索引位置；如果未找到该字节数组，则为 -1。如果 searchBytes 为 null 或者长度为0，则返回值为 -1。</returns>
        public static int BytesIndexOf(byte[] srcBytes, byte searchBytes)
        {
            if (srcBytes == null) { return -1; }
            if (srcBytes.Length == 0) { return -1; }
            for (int i = 0; i < srcBytes.Length; i++)
            {
                if (srcBytes[i] == searchBytes)
                {
                    return i;
                }
            }
            return -1;
        }

        /// <summary>
        /// 16进制字符串转ASCII
        /// </summary>
        /// <param name="Data"></param>
        /// <param name="istrans"></param>
        /// <returns></returns>
        public static string HexStrToAsciiString(string Data, bool istrans)
        {
            if (istrans)
            {
                Data = Regex.Replace(Data, @"(?is)(?<=^([0-9a-f]{2})+)(?!$)", " ");
                Data = Data.Replace("3D 00", "3D ").Replace("3D 11", "2C ").Replace("3D 14", "28 ").Replace("3D 15", "29 ").Replace(" ", "");
            }
            int count = Data.Length / 2;
            string strContent = "";
            char[] num = new char[count];
            for (int i = 0; i < count; i++)
            {
                string str = Data.Substring(i * 2, 2);
                byte by = Convert.ToByte(Convert.ToInt32(str, 16));
                num[i] = Convert.ToChar(by);
                strContent += num[i].ToString();
            }
            return strContent;
        }
```

```
        /// <summary>
        /// 二进制转ASCII
        /// </summary>
        /// <param name="bt"></param>
        /// <returns></returns>
        public static string ByteToASCII(byte[] bt)
        {
            string lin = "";
            for (int i = 0; i < bt.Length; i++)
            {
                lin = lin + bt[i] + " ";
            }
            string[] ss = lin.Trim().Split(new char[] { ' ' });
            char[] c = new char[ss.Length];
            int a;
            for (int i = 0; i < c.Length; i++)
            {
                a = Convert.ToInt32(ss[i]);
                c[i] = Convert.ToChar(a);
            }

            string b = new string(c);
            return b;
        }

        /// <summary>
        /// 时间格式转换
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static DateTime GetDataTime(byte[] bytes)
        {
            return DateTime.ParseExact(ByteToHexStr(bytes), "yyMMddHHmmss",
 System.Globalization.CultureInfo.CurrentCulture);
        }

        public static DateTime GetDataTime(string strdate)
        {
            return DateTime.ParseExact(strdate, "yyMMddHHmmss", System.Glob
alization.CultureInfo.CurrentCulture);
        }

        /// <summary>
        /// 获取二进制第index位的值
        /// </summary>
```

```
        /// <param name="number"></param>
        /// <param name="index"></param>
        /// <returns></returns>
        public static int getBitValue(long number, int index)
        {
            return (number & (1 << index)) > 0 ? 1 : 0;
        }

        /// <summary>
        /// 字节数组转16进制字符串
        /// </summary>
        /// <param name="byteDatas"></param>
        /// <returns></returns>
        public static string ByteToHexStr(byte[] byteDatas)
        {
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < byteDatas.Length; i++)
            {
                builder.Append(string.Format("{0:X2}", byteDatas[i]));
            }
            return builder.ToString().Trim();
        }

        public static ushort SwapUInt16(ushort v)
        {
            return (ushort)(((v & 0xff) << 8) | ((v >> 8) & 0xff));
        }

        public static uint SwapUInt32(uint v)
        {
            return (uint)(((SwapUInt16((ushort)v) & 0xffff) << 0x10) |
                          (SwapUInt16((ushort)(v >> 0x10)) & 0xffff));
        }

        /// <summary>
        /// Short转Bytes
        /// </summary>
        /// <param name="number"></param>
        /// <returns></returns>
        public static byte[] ShortToBytes(short number)
        {
            byte byte2 = (byte)(number >> 8);
            byte byte1 = (byte)(number & 255);
            byte[] bytes = new byte[2];
            bytes[0] = byte1;
            bytes[1] = byte2;
```

```
            return bytes.Reverse().ToArray();
        }

        /// <summary>
        /// 随机short
        /// </summary>
        /// <returns></returns>
        public static short RandomNumber(int min, int max)
        {
            Random rd = new Random();
            return (short)rd.Next(min, max);
        }

        /// <summary>
        /// 计算校验码
        /// </summary>
        /// <param name="bytes"></param>
        /// <returns></returns>
        public static byte xor(List<byte> bytes)
        {
            int checksum = 0;
            foreach (byte b in bytes)
            {
                checksum ^= b;
            }
            return (byte)(checksum & 0xff);
        }

        /// <summary>
        /// 转义
        /// </summary>
        /// <param name="inBytes"></param>
        /// <returns></returns>
        public static byte[] escape(List<byte> inBytes)
        {
            List<byte> outBytes = new List<byte>();
            foreach (byte b in inBytes)
            {
                if (b == 0x7E)
                {
                    outBytes.AddRange(HexStringToBytes("7D02"));
                }
                else if (b == 0x7D)
                {
                    outBytes.AddRange(HexStringToBytes("7D01"));
                }
```

```
                else
                {
                    outBytes.Add(b);
                }
            }
            return outBytes.ToArray();
        }
    }
}
```

(3)解包工具类PacketUtil.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Jt709DataParser
{
    public class PacketUtil
    {

        public static byte[] decodePacket(byte[] bytes)
        {
            //查找消息尾
            byte[] msgBodyNoHeader = bytes.Skip(1).Take(bytes.Length - 1).T
oArray();
            int tailIndex = Common.BytesIndexOf(msgBodyNoHeader, 0x7E);
            if (tailIndex <= 0)
            {
                return null;
            }
            return unescape(bytes, tailIndex).ToArray();
        }
        /// <summary>
        /// 反转义
        /// </summary>
        /// <param name="bytes"></param>
        /// <param name="bodyLen"></param>
        /// <returns></returns>
        public static List<byte> unescape(byte[] bytes, int bodyLen)
        {
            int i = 0;
            List<byte> frame = new List<byte>();
```

```
            while (i < bodyLen)
            {
                byte b = bytes[i];
                if (b == 0x7D)
                {
                    byte nextByte = bytes[i + 1];
                    if (nextByte == 0x01)
                    {
                        frame.Add(0x7D);
                    }
                    else if (nextByte == 0x02)
                    {
                        frame.Add(0x7E);
                    }
                    else
                    {
                        //异常数据
                        frame.Add(b);
                        frame.Add(nextByte);
                    }
                    i += 2;
                }
                else
                {
                    frame.Add(b);
                    i++;
                }
            }
            return frame;
        }

        /// <summary>
        /// 解析定位消息体
        /// </summary>
        /// <param name="msgBodyBuf"></param>
        /// <returns></returns>
        public static LocationData parseLocationBody(byte[] msgBodyBuf)
        {
            //报警标志
            long alarmFlag = Common.SwapUInt32(BitConverter.ToUInt32(msgBod
yBuf, 0));
            //状态
            long status = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBu
f, 4));
            //纬度
            double lat = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
```

```
, 8)) * 0.000001;
            //经度
            double lon = Common.SwapUInt32(BitConverter.ToUInt32(msgBodyBuf
, 12)) * 0.000001;
            //海拔高度,单位为米
            int altitude = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 16));
            //速度
            double speed = Common.SwapUInt16(BitConverter.ToUInt16(msgBodyB
uf, 18)) * 0.1;
            //方向
            int direction = Common.SwapUInt16(BitConverter.ToUInt16(msgBody
Buf, 20));
            //定位时间
            DateTime gpsZonedDateTime = Common.GetDataTime(msgBodyBuf.Skip(
22).Take(6).ToArray());
            //根据状态位的值判断是否南纬和西经
            if (Common.getBitValue(status, 2) == 1)
            {
                lat = -lat;
            }
            if (Common.getBitValue(status, 3) == 1)
            {
                lon = -lon;
            }
            //定位状态
            int locationType = Common.getBitValue(status, 18);
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 1);
            }
            if (locationType == 0)
            {
                locationType = Common.getBitValue(status, 6) > 0 ? 2 : 0;
            }
            //锁绳状态
            int lockRope = Common.getBitValue(status, 20);
            //锁电机状态
            int lockMotor = Common.getBitValue(status, 21);
            //锁状态(由锁绳/按钮+电机来组合判断，当锁绳/按钮为开（1）则锁状态必然
为开（1）；或者电机状态为开（1）则锁状态也为开（1）；其他的则为关（0））
            int lockStatus = 0;
            if (lockRope == 1 || lockMotor == 1)
            {
                lockStatus = 1;
            }
```

```
            //后盖状态
            int backCover = Common.getBitValue(status, 7);
            //唤醒源
            long awaken = (status >> 24) & 0x0f;
            int alarm = parseAlarm(alarmFlag);
            LocationData locationData = new LocationData();
            locationData.GpsTime = gpsZonedDateTime;
            locationData.Latitude = lat;
            locationData.Longitude = lon;
            locationData.LocationType = locationType;
            locationData.Speed = (int)speed;
            locationData.Direction = direction;
            locationData.Altitude = altitude;
            locationData.LockStatus = lockStatus;
            locationData.LockRope = lockRope;
            locationData.Alarm = alarm;
            locationData.BackCover = backCover;
            locationData.Awaken = (int)awaken;
            //处理附加信息
            if (msgBodyBuf.Length > 28)
            {
                byte[] extraBytes = msgBodyBuf.Skip(28).Take(msgBodyBuf.Len
gth - 28).ToArray();
                parseExtraInfo(extraBytes, locationData);
            }
            return locationData;
        }

        /// <summary>
        /// 解析附加信息
        /// </summary>
        /// <param name="msgBody"></param>
        /// <param name="location"></param>
        private static void parseExtraInfo(byte[] msgBody, LocationData loc
ation)
        {
            byte[] extraInfoBuf = null;
            while (msgBody.Length > 1)
            {
                int extraInfoId = msgBody[0];
                int extraInfoLen = msgBody[1];
                if (msgBody.Length - 2 < extraInfoLen)
                {
                    break;
                }
                extraInfoBuf = msgBody.Skip(2).Take(extraInfoLen).ToArray()
```

```
;
                msgBody = msgBody.Skip(2 + extraInfoLen).Take(msgBody.Lengt
h - (2 + extraInfoLen)).ToArray();
                switch (extraInfoId)
                {
                    //锁事件
                    case 0x0B:
                        LockEvent lockEvent = new LockEvent();
                        //锁事件
                        int type = extraInfoBuf[0];
                        lockEvent.Type = type;
                        if (type == 0x01 || type == 0x02 || type == 0x03 ||
 type == 0x05 || type == 0x1E || type == 0x1F)
                        {
                            byte[] passwordBytes = extraInfoBuf.Skip(1).Tak
e(6).ToArray();
                            string password = Common.ByteToASCII(passwordBy
tes);
                            lockEvent.Password = password;
                            int unlockStatus = extraInfoBuf[7];
                            if (unlockStatus == 0xff)
                            {
                                lockEvent.UnLockStatus = 0;
                            }
                            else
                            {
                                if (unlockStatus > 0 && unlockStatus < 100)
                                {
                                    lockEvent.FenceId = unlockStatus;
                                }
                                lockEvent.UnLockStatus = 1;
                            }
                        }
                        else if (type == 0x06 || type == 0x07 || type == 0x
08 || type == 0x10 || type == 0x11 || type == 0x18 || type == 0x19 || type
== 0x20 || type == 0x28 || type == 0x29)
                        {
                            byte[] passwordBytes = extraInfoBuf.Skip(0).Tak
e(6).ToArray();
                            string password = Common.ByteToASCII(passwordBy
tes);
                            lockEvent.Password = password;
                            lockEvent.UnLockStatus = 0;
                        }
                        else if (type == 0x22)
                        {
```

```
                            long cardId = Common.SwapUInt32(BitConverter.To
UInt32(extraInfoBuf, 0));
                            if (cardId != 0)
                            {
                                lockEvent.CardNo = cardId.ToString().PadLef
t(10, '0');
                            }
                            if (extraInfoBuf.Length > 4)
                            {
                                int unlockStatus = extraInfoBuf[4];
                                if (unlockStatus == 0xff)
                                {
                                    lockEvent.UnLockStatus = 0;
                                }
                                else
                                {
                                    if (unlockStatus > 0 && unlockStatus <
100)
                                    {
                                        lockEvent.FenceId = unlockStatus;
                                    }
                                    lockEvent.UnLockStatus = 1;
                                }
                            }
                        }
                        else if (type == 0x23 || type == 0x2A || type == 0x
2B)
                        {
                            long cardId = Common.SwapUInt32(BitConverter.To
UInt32(extraInfoBuf, 0));
                            if (cardId != 0)
                            {
                                lockEvent.CardNo = cardId.ToString().PadLef
t(10, '0');
                            }
                        }
                        location.LockEvent = lockEvent;
                        break;
                    //无线通信网络信号强度
                    case 0x30:
                        int fCellSignal = extraInfoBuf[0];
                        location.GSMSignal = fCellSignal;
                        break;
                    //卫星数
                    case 0x31:
                        int fGPSSignal = extraInfoBuf[0];
```

```
                        location.GpsSignal = fGPSSignal;
                        break;
                    //电池电量百分比
                    case 0xD4:
                        int fBattery = extraInfoBuf[0];
                        location.Battery = fBattery;
                        break;
                    //电池电压
                    case 0xD5:
                        int fVoltage = Common.SwapUInt16(BitConverter.ToUIn
t16(extraInfoBuf, 0));
                        location.Voltage = fVoltage * 0.01;
                        break;
                    case 0xF9:
                        //版本号
                        int version = Common.SwapUInt16(BitConverter.ToUInt
16(extraInfoBuf, 0));
                        location.ProtocolVersion = version;
                        break;
                    case 0xFD:
                        //小区码信息
                        int mcc = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 0));
                        location.MCC = mcc;
                        int mnc = extraInfoBuf[2];
                        location.MNC = mnc;
                        long cellId = Common.SwapUInt32(BitConverter.ToUInt
32(extraInfoBuf, 3));
                        location.CELLID = cellId;
                        int lac = Common.SwapUInt16(BitConverter.ToUInt16(e
xtraInfoBuf, 7));
                        location.LAC = lac;
                        break;
                    case 0xFC:
                        int fenceId = extraInfoBuf[0];
                        location.FenceId = fenceId;
                        break;
                    case 0xFE:
                        long mileage = Common.SwapUInt32(BitConverter.ToUIn
t32(extraInfoBuf, 0));
                        location.Mileage = mileage;
                        break;
                    default:
                        break;
                }
            }
```

```
        }

        /// <summary>
        /// 温度解析
        /// </summary>
        /// <param name="temperatureInt"></param>
        /// <returns></returns>
        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;
        }

        /// <summary>
        /// 回复内容
        /// </summary>
        /// <param name="terminalNumArr"></param>
        /// <param name="msgFlowId"></param>
        /// <returns></returns>
        public static string replyBinaryMessage(byte[] terminalNumArr, int
msgFlowId)
        {
            List<byte> byteList = new List<byte>();
            byteList.AddRange(Common.HexStringToBytes("8001"));
            byteList.AddRange(Common.HexStringToBytes("0005"));
            byteList.AddRange(terminalNumArr);
            byteList.AddRange(Common.ShortToBytes(Common.RandomNumber(0, 65
534)));
            byteList.AddRange(Common.ShortToBytes((short)msgFlowId));
            byteList.AddRange(Common.HexStringToBytes("0200"));
            byteList.Add(0x00);
            byteList.Add(Common.xor(byteList));
            byte[] bytes = Common.escape(byteList);
            List<byte> replyList = new List<byte>();
            replyList.Add(0x7E);
            replyList.AddRange(bytes);
            replyList.Add(0x7E);
            return Common.ByteToHexStr(replyList.ToArray());
```

```
        }

        /// <summary>
        /// 授时
        /// </summary>
        /// <param name="itemList"></param>
        /// <returns></returns>
        public static string replyBASE2Message(string[] itemList)
        {
            try
            {
                string strBase2Reply = string.Format("({0},{1},{2},{3},{4},
{5})", itemList[0], itemList[1]
                        , itemList[2], itemList[3], itemList[4], DateTime.U
tcNow.ToString("yyyyMMddHHmmss"));
                return strBase2Reply;
            }
            catch (Exception e)
            {
                return "";
            }
        }

        /// <summary>
        /// 报警解析
        /// </summary>
        /// <param name="alarmFlag"></param>
        /// <returns></returns>
        private static int parseAlarm(long alarmFlag)
        {
            int alarm = -1;
            //单次触发报警
            if (Common.getBitValue(alarmFlag, 1) == 1)
            {
                alarm = (int)AlarmTypeEnum.ALARM_1;
            }
            else if (Common.getBitValue(alarmFlag, 7) == 1)
            {
                alarm = (int)AlarmTypeEnum.ALARM_2;
            }
            else if (Common.getBitValue(alarmFlag, 16) == 1)
            {
                alarm = (int)AlarmTypeEnum.ALARM_3;
            }
            else if (Common.getBitValue(alarmFlag, 17) == 1)
            {
```

```
                    alarm = (int)AlarmTypeEnum.ALARM_4;
                }
                else if (Common.getBitValue(alarmFlag, 18) == 1)
                {
                    alarm = (int)AlarmTypeEnum.ALARM_5;
                }
                return alarm;
            }
        }
    }
```

2.4.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007501804283100001267E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据（不带锁事件）：

```
7E0200003A790011000094180C0000000000234000201E807C80713B76A00780000000021080
4150651D4014AD502018C30011A310108FE0400000BCCFD0901CC000000308D51D0F27E
```

返回消息：

```
{
    "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"
    }
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间）,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关）,
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
```

```
        "Battery": 电量值（0~100%,255 表示电池充电中），
        "GpsSignal": GPS当前接收到的卫星颗数，
        "Voltage": 电压值，单位V，
        "Speed": 速度，单位km/h，
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
        "Mileage": 里程值，单位km，
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏
报警；5:出围栏报警），
        "DataLength": 数据长度（字节数），
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "Altitude": 海拔高度km，
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

原始数据（带锁事件）：

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

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间），
        "Latitude": 纬度（WGS84），
        "Longitude": 经度（WGS84），
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关），
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳
唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%,255表示电池充电中），
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
```

```
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏报警；5:出围栏报警），
        "LockEvent": {
            "Type": 事件类型（参见附件1），
            "FenceId": 事件关联的围栏ID（-1：与围栏无关），
            "UnLockStatus": 开锁状态（1：开锁成功；0：开锁失败），
            "Password": 开锁密码
            "CardNo":如果事件类型为刷卡开锁，这里表示的是卡号
        },
        "DataLength": 数据长度（字节数），
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析

原始数据：

```
283739313030353030303030332c312c3030312c424153452c312c4a54373039415f3230323
0303932325f48572d56312e305f4e6f524649445f53494d434f4d373630305f56325f312c30
2c4c45313142303353494d373630304d31315f412c383938363034313231303138433037333
83535342c383637353834303334363131313131362c3436302c302c313830353535343736342c31
3033343229
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
```

```
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复），
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(791005000003,1,001,BASE,1,JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1,0,LE11B03SIM7600M11_A,898604121018C0738554,867584034611116,460,0,180554764,10342)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1	当前设备的版本号。
4	0	前面的0表示英文，1表示其它语言的Unicode码
5	LE11B03SIM7600M11_A	GSM模块版本。
6	898604121018C0738554	SIM卡的ICCID。
7	867584034611116	GSM模块的IMEI号。
8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,5, 1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,5	指令类型（BASE5）
3	1,1,1,1,1,1,1,1,1,1	从左至右依次为(1)锁挂绳剪断报警、(2)刷非法卡报警、(3)开锁状态保持一段时间报警、(4)指令开锁密码连续输错5次报警、(5)震动报警、(6)进区域报警、(7)出区域报警的开关、(8)低电报警（电量低于满电的30%）、(9)开后盖报警、(10)卡锁报警、(11)超速报警;(每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)

3.6.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，默认60。最长1小时，最短5秒
4	30	设备正常工作间隔，单位分钟，默认60分钟. 当该值为0时，启用持续追踪模式，设备不休眠。最长24小时，最短5分钟

3.7.查询/设置终端休眠模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,7	指令类型（BASE7）
3	1	0正常休眠模式任一中断、RTC可以唤醒，1表示在0模式上增加短信电话可以唤醒

3.8.查询/设置终端本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.9.查询/设置终端本地时差

消息体内容（DataBody）：

(700160818000,1,001,BASE,9,8613998765432,0,0,0,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,9	指令类型（BASE9）
3	8613998765432,0,0,0,0	5个监控手机号码,0表示没有设置

3.10.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）
4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.11.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	则表明当前有效闹钟个数；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.12.查询/设置仅支持VIP号码

消息体内容（DataBody）：

(700160818000,1,001,BASE,40,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,40	指令类型（BASE40）
3	1	1表示仅支持VIP号码进行设备配置和查询，0表示非vip号码也可以配置和查询。默认0

3.13.查询/设置里程统计相关参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,43,10,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,43	指令类型（BASE43）
3	10	表示里程统计的速度限制值操作(即速度小于改值不统计里程,默认10km/h, 参数0-100)
4	10	表示设置里程同步值为10公里

3.14.查询/设置静态飘移处理功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,44,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	BASE,44	指令类型（BASE44）
3	1	1打开这个静态漂移处理功能，0表示关闭静态漂移处理功能

3.15.查询/设置GPS追踪定位模式功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,45,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,45	指令类型（BASE45）
3	1	1表示节能定时追踪，0 表示节能定位追踪，2表示全速追踪

3.16.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,46,5,25,6)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,46	指令类型（BASE46）
3	5	HDOP门限值
4	25	单颗卫星信噪比门限值
5	6	联合定位卫星个数门限值

3.17.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,47,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1	1: 启用监控IP服务器，0关闭监控IP服务器

3.18.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,48, 1,1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1,1,1,1, 1,1,1,1, 1,1,1,1,1	从左至右依次为(1) 锁绳剪断报警、(2)刷非法卡报警、(3)长时间开锁报警、(4)开锁密码错误、(5)震动报警、(6)进区域报警、(7)出区域报警、(8)低电报警、(9)开后盖报警、(10)卡锁报警、(11)GSM信号低报警、(12)超速报警 (每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)，(13)倾斜报警，其中红色的没有相应报警

3.19.查询/设置震动检测阈值

消息体内容（DataBody）：

(700160818000,1,001,GSENS,1, 500)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	GSENS,1	指令类型（GSENS1）
3	500	500：单位是mg（gsensor芯片63mg为阶梯配置，实际芯片配置值为500/63取整=7），最小63，最大值8000，默认500。当震动达到这个值，会检测到一次震动。如果值等于0，关闭该功能

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,7)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,7	指令类型（DEBUG7）

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15, 10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,15	指令类型（DEBUG15）
3	10	0表示马上进入睡眠，设置大于0的数表示相应时间后进入休眠

3.21.查询物联卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27, ICCID, IMSI)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,27	指令类型（DEBUG27）
3	ICCID	ICCID号码
4	IMSI	IMSI号码

3.22.查询/清除历史(盲区)数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,29, 0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,29	指令类型（DEBUG29）
3	0	返回0则表明当前历史数据条数

3.23.查询/设置超速报警速度

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,30,120,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,30	指令类型（DEBUG30）
3	120	报警的速度临界值(默认值120)，单位为:公里
4	10	超速此时间后报警(默认值10),单位：秒

3.24.查询/设置低电报警数值

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,31,5)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,31	指令类型（DEBUG31）
3	5	报警的电量临界值；5表明当前电量百分比

3.25.查询/设置关闭看门狗作用

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,38,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,38	指令类型（DEBUG38）
3	0	返回0表明已接收该指令，并生效。

3.26.设置/查询区域详细节点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,7,8,15,1,10,11323.1234...)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,7	指令类型（GFCE7）
3	8	8表示第八个区域

4	15	15表示总点数
5	1	1表示当前页
6	10	10表示当前页的点数
7	11323.1234...	各个点的经度与纬度

3.27.清除设置的全部区域点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,8,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,8	指令类型（GFCE8）
3	1	1表示删除围栏的ID

3.28.查询/设置没有关锁报警提醒时间间隔

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,2, 1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,2	指令类型（ELOCK2）
3	1	提醒时间，单位(分钟)。当开锁后没有关锁，如果此值不为0(为0无效)时间到就上报开锁提醒消息，最小值为1分钟，最大值为10分钟。默认为1分钟

3.29.终端上报动态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,3,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,3	指令类型（ELOCK3）
3	123456	上报的动态密码

3.30.设置/修改远程开锁静态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,4,666666)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,4	指令类型（ELOCK4）
3	666666	666666: 表示修改成功的密码。

3.31.凭静态或动态密码开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,5,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,5	指令类型（ELOCK5）
3	1	0: 表示密码正确，大于0表示密码错误开锁失败

3.32.查询/设置围栏外禁止开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1,2,3,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示设置了栏外禁止开锁，必须在定位状下围栏内才能开锁。2表示设置了栏外禁止开锁，有定位必须在围栏内开锁，没有定位可以通过远程静态密码或动态密码开锁，如果设置了围栏外禁止开锁没有定位不能通过现场按键或蓝牙APP开锁。0表示没有开启围栏外禁止开锁功能，默认没有开启。
4	2	围栏ID个数：1~10有效，只支持区域围栏。其它类型的围栏不支持
5	3，4	围栏ID： 1~10有效，3,4表示只能在区域围栏ID为3,4的围栏可以开锁，1～10的ID存在对应的位置

3.33.动态密码权限处理(不对外公开)

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示使能产生动态密码；0表示关闭产生动态密码

3.34.查询/增删/开锁授权号

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,15,0,2,20,2,1,0013953759, 0013953758, 0013953757，，，，，，，，，，，，)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	0	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	2	2为查询第2组数据，一共分3组，分别为1~3
5	20	20表示有20个ID号
6	2	2表示总共有2应答包
7	1	1表示当前是第1个应答包
8	0013953759, 0013953758, 0013953757，，，，，	表示该组存储的授权号列表

消息体内容（DataBody）实例2：

(700160818000,1,001,ELOCK,15,1,3,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	1	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	增加3个授权卡号成功

5	200	当前总共有200个卡

消息体内容（DataBody）实例3：

(700160818000,1,001,ELOCK,15,2,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	2	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	表示删除3个授权卡号成功。(每次最多一次性删除20)

消息体内容（DataBody）实例4：

(700160818000,1,001, ELOCK,15,3,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	3	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	0	表示删除全部授权卡号成功，目前剩余0个授权卡

3.35.现场刷卡授权模式配置指令

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,16,2,0013953759,0013953751)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	2	授权的卡个数
4	0013953759,0013953751	授权的卡号

消息体内容（DataBody）实例2：

(700160818000,1,001, ELOCK,16,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	0	1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。

3.36.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.

4	20120102	当前设备的Firmware版本号.

3.37.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.38.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.39.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.40.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.41.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT709A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）
3	JT709A_V103R001	固件版本号

附件1

事件ID（16进制）	事件ID	事件描述
0x01	1	静态密码远程开锁
0x02	2	动态密码远程开锁
0x03	3	动态密码现场(蓝牙或WIFI)APP开锁
0x05	5	表示静态密码现场(蓝牙或WIFI)APP开锁
0x06	6	错误的静态密码远程开锁
0x07	7	错误的动态密码远程开锁
0x08	8	错误的动态密码现场(蓝牙或WIFI)APP开锁
0x0B	11	长时间开锁事件
0x0C	12	锁绳剪断事件
0x0D	13	关锁事件(包括自动关锁和远程关锁)
0x10	16	远程执行开锁异常，没有定位不执行开锁
0x11	17	远程执行开锁异常，有定位在围栏外不执行开锁
0x12	18	电机异常
0x18	24	蓝牙执行开锁异常，没有定位不执行开锁
0x19	25	蓝牙执行开锁异常，有定位在围栏外不执行开锁
0x1C	28	开锁并拔出锁绳
0x1E	30	短信静态密码远程开锁

0x1F	31	短信动态密码远程开锁
0x20	32	表示错误的短信动态密码
0x22	34	刷授权卡开锁事件
0x23	35	刷非法卡开锁事件
0x28	40	错误的静态密码现场(蓝牙或WIFI)APP开锁
0x29	41	错误的短信静态密码开锁
0x2A	42	RFID执行开锁异常，没有定位不执行开锁
0x2B	43	RFID执行开锁异常，有定位在围栏外不执行开锁

Java 版本

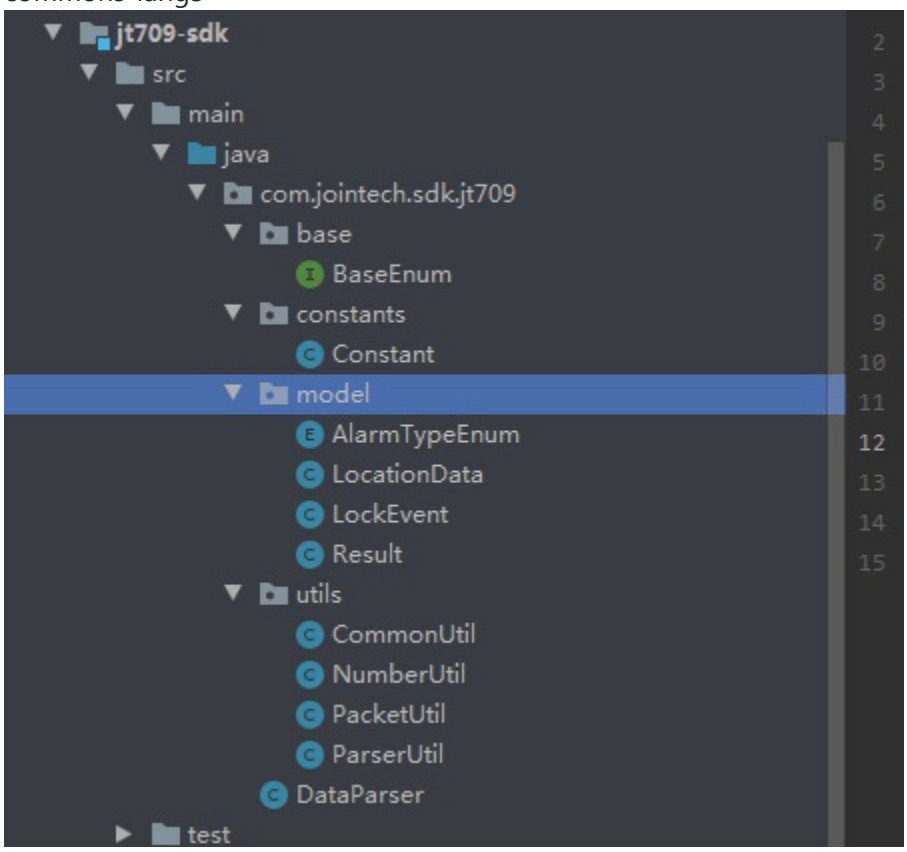
1.Jar包下载

jt709-sdk-1.0.0.jar 下载地址
如果需要Jar包开发源码，请与商务申请

2.集成开发说明

2.1.集成开发语言及框架说明

jt709-sdk-1.0.0.jar是基于Java语言，SpringBoot2.x框架，使用到了netty，fastjson，lombok，commons-lang3



BaseEnum：基础枚举
Constant：自定义常量
AlarmTypeEnum:终端报警类型枚举
LocationData：定位实体类
LockEvent：锁事件实体类
Result：结果实体类
CommonUtil：公共方法类
NumberUtil：数字操作工具类

PacketUtil：用来处理预处理数据以及分包方法封装类
ParserUtil：解析方法工具类
DataParser：解析主方法

2.2.集成说明

将jt709-sdk-1.0.0.jar引入到自己的网关程序中，引入方法如下：
在pom.xml引入jar包

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

调用jt709-sdk-1.0.0.jar，DataParser类中receiveData()方法
receiveData()方法是重载方法

```
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: 解析方法的主体</p>
 *
 * @author lenny
 * @version 1.0.1
 */
public class DataParser {
    /**
     * 解析Hex字符串原始数据
     * @param strData 16进制字符串
     * @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);
```

```
    }

    /**
     * 解析byte[]原始数据
     * @param bytes
     * @return
     */
    private static Object receiveData(byte[] bytes)
    {
        ByteBuf msgBodyBuf =Unpooled.buffer(bytes.length);
        msgBodyBuf.writeBytes(bytes);
        return receiveData(msgBodyBuf);
    }

    /**
     * 解析ByteBuf原始数据
     * @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.核心代码

ParserUtil：解析方法工具类

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

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: 解析方法工具类</p>
 * @author HyoJung
 */
public class ParserUtil {
    /**
     * 定位数据解析0x0200
     * @param in
     * @return
     */
    public static Result decodeBinaryMessage(ByteBuf in) {
        //对数据进行反转移处理
        ByteBuf msg = (ByteBuf) PacketUtil.decodePacket(in);
        //消息长度
        int msgLen = msg.readableBytes();
        //包头
        msg.readByte();
        //消息ID
        int msgId = msg.readUnsignedShort();
        //消息体属性
        int msgBodyAttr = msg.readUnsignedShort();
        //消息体长度
        int msgBodyLen = msgBodyAttr & 0b00000011_11111111;
        //是否分包
        boolean multiPacket = (msgBodyAttr & 0b00100000_00000000) > 0;
        //去除消息体的基础长度
        int baseLen = Constant.BINARY_MSG_BASE_LENGTH;

        //根据消息体长度和是否分包得出后面的包长
        int ensureLen = multiPacket ? baseLen + msgBodyLen + 4 : baseLen +
msgBodyLen;
        if (msgLen != ensureLen) {
            return null;
        }
```

```
            //终端号数组
            byte[] terminalNumArr = new byte[6];
            msg.readBytes(terminalNumArr);
            //终端号(去除前面的0)
            String terminalNumber = StringUtils.stripStart(ByteBufUtil.hexDump(
terminalNumArr), "0");
            //消息流水号
            int msgFlowId = msg.readUnsignedShort();
            //消息总包数
            int packetTotalCount = 0;
            //包序号
            int packetOrder = 0;
            //分包
            if (multiPacket) {
                packetTotalCount = msg.readShort();
                packetOrder = msg.readShort();
            }
            //消息体
            byte[] msgBodyArr = new byte[msgBodyLen];
            msg.readBytes(msgBodyArr);
            if(msgId==0x0200) {
                //解析消息体
                LocationData locationData=parseLocationBody(Unpooled.wrappedBuf
fer(msgBodyArr));
                locationData.setIndex(msgFlowId);
                locationData.setDataLength(msgBodyLen);
                //校验码
                int checkCode = msg.readUnsignedByte();
                //包尾
                msg.readByte();
                //获取消息回复内容
                String replyMsg= PacketUtil.replyBinaryMessage(terminalNumArr,m
sgFlowId);
                //定义定位数据实体类
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("Location");
                model.setDataBody(locationData);
                model.setReplyMsg(replyMsg);
                return model;
            }else {
                //定义定位数据实体类
                Result model = new Result();
                model.setDeviceID(terminalNumber);
                model.setMsgType("heartbeat");
                model.setDataBody(null);
```

```
            return model;
        }
    }

    /**
     * 解析指令数据
     * @param in 原始数据
     * @return
     */
    public static Result decodeTextMessage(ByteBuf in) {

        //读指针设置到消息头
        in.markReaderIndex();
        //查找消息尾，如果未找到则继续等待下一包
        int tailIndex = in.bytesBefore(Constant.TEXT_MSG_TAIL);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }
        //定义定位数据实体类
        Result model = new Result();
        //包头(
        in.readByte();
        //字段列表
        List<String> itemList = new ArrayList<String>();
        while (in.readableBytes() > 0) {
            //查询逗号的下标截取数据
            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;
    }

    /**
     * 解析定位消息体
     * @param msgBodyBuf
     * @return
     */
    private static LocationData parseLocationBody(ByteBuf msgBodyBuf){
        //报警标志
        long alarmFlag = msgBodyBuf.readUnsignedInt();
        //状态
        long status = msgBodyBuf.readUnsignedInt();
        //纬度
        double lat = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //经度
        double lon = NumberUtil.multiply(msgBodyBuf.readUnsignedInt(), Numb
erUtil.COORDINATE_PRECISION);
        //海拔高度,单位为米
        int altitude = msgBodyBuf.readShort();
        //速度
        double speed = NumberUtil.multiply(msgBodyBuf.readUnsignedShort(),
NumberUtil.ONE_PRECISION);
        //方向
        int direction = msgBodyBuf.readShort();
        //定位时间
        byte[] timeArr = new byte[6];
        msgBodyBuf.readBytes(timeArr);
        String bcdTimeStr = ByteBufUtil.hexDump(timeArr);
        ZonedDateTime gpsZonedDateTime = CommonUtil.parseBcdTime(bcdTimeStr
);
        //根据状态位的值判断是否南纬和西经
        if (NumberUtil.getBitValue(status, 2) == 1) {
            lat = -lat;
        }
```

```
        if (NumberUtil.getBitValue(status, 3) == 1) {
            lon = -lon;
        }
        //定位状态
        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;
        }
        //锁绳状态
        int lockRope=NumberUtil.getBitValue(status, 20);
        //锁电机状态
        int lockMotor=NumberUtil.getBitValue(status, 21);
        //锁状态(由锁绳/按钮+电机来组合判断，当锁绳/按钮为开（1）则锁状态必然为开（1）；或者电机状态为开（1）则锁状态也为开（1）；其他的则为关（0））
        int lockStatus=0;
        if(lockRope==1||lockMotor==1) {
            lockStatus=1;
        }
        int backCover=NumberUtil.getBitValue(status, 7);
        //唤醒源
        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);
        //处理附加信息
        if (msgBodyBuf.readableBytes() > 0) {
            parseExtraInfo(msgBodyBuf, locationData);
        }
        return locationData;
    }
```

```
    /**
     * 解析附加信息
     *
     * @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) {
                //锁事件
                case 0x0B:
                    LockEvent event=new LockEvent();
                    //锁事件
                    int type=extraInfoBuf.readUnsignedByte();
                    event.setType(type);
                    if(type==0x01||type==0x02||type==0x03||type==0x05||type
==0x1E||type==0x1F){
                        //凭密开锁
                        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){
                        //凭密开锁
                        byte[] passwordArr = new byte[6];
                        extraInfoBuf.readBytes(passwordArr);
                        String password=new String(passwordArr);
```

```
                        event.setPassword(password);
                        event.setUnLockStatus(0);
                    }else if(type==0x22){
                        //卡号
                        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){
                        //卡号
                        long cardId = extraInfoBuf.readUnsignedInt();
                        if(cardId!=0) {
                            event.setCardNo(String.format("%010d", cardId))
;
                        }
                    }
                    location.setLockEvent(event);
                    break;
                //无线通信网络信号强度
                case 0x30:
                    int fCellSignal=extraInfoBuf.readByte();
                    location.setGSMSignal(fCellSignal);
                    break;
                //卫星数
                case 0x31:
                    int fGPSSignal=extraInfoBuf.readByte();
                    location.setGpsSignal(fGPSSignal);
                    break;
                //电池电量百分比
                case 0xD4:
```

```
                    int fBattery=extraInfoBuf.readUnsignedByte();
                    location.setBattery(fBattery);
                    break;
                //电池电压
                case 0xD5:
                    int fVoltage=extraInfoBuf.readUnsignedShort();
                    location.setVoltage(fVoltage*0.01);
                    break;
                case 0xF9:
                    //版本号
                    int version=extraInfoBuf.readUnsignedShort();
                    location.setProtocolVersion(version);
                    break;
                case 0xFD:
                    //小区码信息
                    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;
            }
        }
    }

    /**
     * 报警解析
     * @param alarmFlag
     * @return
     */
    private static int parseAlarm(long alarmFlag) {
        int alarm=-1;
```

```
            //单次触发报警
            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：用来处理预处理数据以及恢复方法封装类

```
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;

/**
 * 解析包预处理（进行反转义）
 * @author HyoJung
 */
public class PacketUtil {
    /**
     * 解析消息包
     *
```

```
     * @param in
     * @return
     */
    public static Object decodePacket(ByteBuf in) {
        //可读长度不能小于基本长度
        if (in.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            return null;
        }

        //防止非法码流攻击，数据太大为异常数据
        if (in.readableBytes() > Constant.BINARY_MSG_MAX_LENGTH) {
            in.skipBytes(in.readableBytes());
            return null;
        }

        //查找消息尾，如果未找到则继续等待下一包
        in.readByte();
        int tailIndex = in.bytesBefore(Constant.BINARY_MSG_HEADER);
        if (tailIndex < 0) {
            in.resetReaderIndex();
            return null;
        }

        int bodyLen = tailIndex;
        //创建ByteBuf存放反转义后的数据
        ByteBuf frame = ByteBufAllocator.DEFAULT.heapBuffer(bodyLen + 2);
        frame.writeByte(Constant.BINARY_MSG_HEADER);
        //消息头尾之间的数据进行反转义
        unescape(in, frame, bodyLen);
        in.readByte();
        frame.writeByte(Constant.BINARY_MSG_HEADER);

        //反转义后的长度不能小于基本长度
        if (frame.readableBytes() < Constant.BINARY_MSG_BASE_LENGTH) {
            ReferenceCountUtil.release(frame);
            return null;
        }
        return frame;
    }

    /**
     * 消息头、消息体、校验码中0x7D 0x02反转义为0x7E，0x7D 0x01反转义为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 {
                    //异常数据
                    frame.writeByte(b);
                    frame.writeByte(nextByte);
                }
                i += 2;
            } else {
                frame.writeByte(b);
                i++;
            }
        }
    }

    /**
     * 消息头、消息体、校验码中0x7E转义为0x7D 0x02，0x7D转义为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);
            }
        }
    }

    /**
     * 回复内容
     * @param terminalNumArr
```

```
     * @param msgFlowId
     * @return
     */
    public static String replyBinaryMessage(byte[] terminalNumArr,int msgFl
owId) {
        //去除包头包尾的长度
        int contentLen = Constant.BINARY_MSG_BASE_LENGTH + 4;
        ByteBuf bodyBuf = ByteBufAllocator.DEFAULT.heapBuffer(contentLen-2)
;
        ByteBuf replyBuf = ByteBufAllocator.DEFAULT.heapBuffer(25);
        try {
            //消息ID
            bodyBuf.writeShort(0x8001);
            //数据长度
            bodyBuf.writeShort(0x0005);
            //终端ID
            bodyBuf.writeBytes(terminalNumArr);
            Random random = new Random();
            //生成1-65534内的随机数
            int index = random.nextInt() * (65534 - 1 + 1) + 1;
            //当前消息流水号
            bodyBuf.writeShort(index);
            //回复消息流水号
            bodyBuf.writeShort(msgFlowId);
            //应答的消息ID
            bodyBuf.writeShort(0x0200);
            //应答结果
            bodyBuf.writeByte(0x00);
            //校验码
            int checkCode = CommonUtil.xor(bodyBuf);
            bodyBuf.writeByte(checkCode);
            //包头
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            //读指针重置到起始位置
            bodyBuf.readerIndex(0);
            //转义
            PacketUtil.escape(replyBuf, bodyBuf);
            //包尾
            replyBuf.writeByte(Constant.BINARY_MSG_HEADER);
            return ByteBufUtil.hexDump(replyBuf);
        } catch (Exception e) {
            ReferenceCountUtil.release(replyBuf);
            return "";
        }
    }
```

```
    /**
     * 授时指令回复
     * @param itemList
     */
    public static String replyBASE2Message(List<String> itemList) {
        try {
            //设置日期格式
            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：数字操作工具类

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

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

/**
 * 数字工具类
 * @author HyoJung
 * @date 20210526
 */
public class NumberUtil {
    /**
     * 坐标精度
     */
    public static final BigDecimal COORDINATE_PRECISION = new BigDecimal("0
.000001");

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

```java
    /**
     * 小数点一位精度
     */
    public static final BigDecimal ONE_PRECISION = new BigDecimal("0.1");

    private NumberUtil() {
    }

    /**
     * 格式化消息ID(转成0xXXXX)
     *
     * @param msgId 消息ID
     * @return 格式化字符串
     */
    public static String formatMessageId(int msgId) {
        return String.format("0x%04x", msgId);
    }

    /**
     * 格式化短数字
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String formatShortNum(int num) {
        return String.format("0x%02x", num);
    }

    /**
     * 转4位的十六进制字符串
     *
     * @param num 数字
     * @return 格式化字符串
     */
    public static String hexStr(int num) {
        return String.format("%04x", num).toUpperCase();
    }

    /**
     * 解析short类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 解析int类型的值,获取值为1的位数
     *
     * @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;
    }

    /**
     * 获取二进制第index位的值
     *
     * @param number
     * @param index
     * @return
     */
    public static int getBitValue(long number, int index) {
        return (number & (1 << index)) > 0 ? 1 : 0;
    }

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

    /**
     * BigDecimal乘法
```

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

CommonUtil：公共方法类

```
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: 用来存储一些解析中遇到的公共方法</p>
 *
 * @author lenny
 * @version 1.0.1
 * @date 20210328
 */
public class CommonUtil {
    /**
     * 去掉字符串最后一个字符
     * @param inStr 输入的字符串
     * @param suffix 需要去掉的字符
     * @return
     */
    public static String trimEnd(String inStr, String suffix) {
        while(inStr.endsWith(suffix)){
            inStr = inStr.substring(0,inStr.length()-suffix.length());
        }
        return inStr;
    }

    /**
```

```
     * 16进制转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();
    }

    /**
     * 转换GPS时间
     *
     * @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;
    }

    /**
     * 每个字节进行异或求值
     *
     * @param buf
     * @return
     */
    public static int xor(ByteBuf buf) {
        int checksum = 0;
        while (buf.readableBytes() > 0) {
            checksum ^= buf.readUnsignedByte();
        }
        return checksum;
    }
}
```

BaseEnum：基础枚举

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

import java.io.Serializable;

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

Constant：自定义常量

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

/**
 * 常量定义
 * @author HyoJung
 * @date 20210526
 */
public class Constant {
    private Constant(){}
    /**
     * 二进制消息包头
     */
    public static final byte BINARY_MSG_HEADER = 0x7E;
    /**
     * 不包含消息体的基本长度
     */
    public static final int BINARY_MSG_BASE_LENGTH = 15;

    /**
     * 消息最大长度
     */
    public static final int BINARY_MSG_MAX_LENGTH = 102400;

    /**
     * 文本消息包头
     */
    public static final byte TEXT_MSG_HEADER = '(';

    /**
```

```
     * 文本消息包尾
     */
    public static final byte TEXT_MSG_TAIL = ')';

    /**
     * 文本消息分隔符
     */
    public static final byte TEXT_MSG_SPLITER = ',';
}
```

AlarmTypeEnum：终端报警类型枚举

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

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

/**
 * 报警枚举
 * @author HyoJung
 */

public enum AlarmTypeEnum implements BaseEnum<String> {

    ALARM_1("超速报警", "1"),
    ALARM_2("低电报警", "2"),
    ALARM_3("主机开盖报警", "3"),
    ALARM_4("进围栏报警", "4"),
    ALARM_5("出围栏报警", "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：定位实体类

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

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

import java.io.Serializable;

/**
 * <p>Description: 定位实体类</p>
 *
 * @author lenny
 * @version 1.0.1
 */
@Data
public class LocationData implements Serializable {
    /**
     * 消息体
     */
    @JSONField(name = "DataLength")
    public int DataLength;
    /**
     * 定位时间
     */
    @JSONField(name = "GpsTime")
    public String GpsTime;
    /**
     * 纬度
     */
    @JSONField(name = "Latitude")
    public double Latitude;
    /**
     * 经度
```

```
     */
    @JSONField(name = "Longitude")
    public double Longitude;
    /**
     * 定位方式
     */
    @JSONField(name = "LocationType")
    public int LocationType;
    /**
     * 速度
     */
    @JSONField(name = "Speed")
    public int Speed;
    /**
     * 方向
     */
    @JSONField(name = "Direction")
    public int Direction;
    /**
     * 里程
     */
    @JSONField(name = "Mileage")
    public long Mileage;
    /**
     * 海拔高度
     */
    @JSONField(name = "Altitude")
    public int Altitude;
    /**
     * GPS信号值
     */
    @JSONField(name = "GpsSignal")
    public int GpsSignal;
    /**
     * GSM信号质量
     */
    @JSONField(name = "GSMSignal")
    public int GSMSignal;
    /**
     * 电量值
     */
    @JSONField(name = "Battery")
    public int Battery;
    /**
     * 电压
     */
```

```
    @JSONField(name = "Voltage")
    public double Voltage;
    /**
     * 锁状态
     */
    @JSONField(name = "LockStatus")
    public int LockStatus;
    /**
     * 锁绳状态
     */
    @JSONField(name = "LockRope")
    public int LockRope;
    /**
     * 后盖状态
     */
    @JSONField(name = "BackCover")
    public int BackCover;
    /**
     * 协议版本号
     */
    @JSONField(name = "ProtocolVersion")
    public int ProtocolVersion;
    /**
     * 围栏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;
    /**
     * 锁事件
     */
    @JSONField(name = "LockEvent")
    public LockEvent LockEvent;
    /**
     * 流水号
     */
    @JSONField(name = "Index")
    public int Index;
}
```

LockEvent：锁事件实体类

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

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

/**
 * 锁事件
 * @author HyoJung
 */
@Data
public class LockEvent {
    /**
     * 事件类型
     */
    @JSONField(name = "Type")
    public int Type;
    /**
     * 刷卡卡号
     */
    @JSONField(name = "CardNo")
```

```
    public String CardNo;
    /**
     * 开锁密码
     */
    @JSONField(name = "Password")
    public String Password;
    /**
     * 开锁状态(1:成功；0：失败)
     */
    @JSONField(name = "UnLockStatus")
    public int UnLockStatus=0;
    /**
     * 与开锁有关的围栏ID
     */
    @JSONField(name = "FenceId")
    public int FenceId=-1;
}
```

Result：结果实体类

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

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

import java.io.Serializable;

/**
 * 结果实体类
 * @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.返回消息及说明

（1）心跳数据
原始数据：

```
7E000200007501804283100001267E
```

返回消息：

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

返回消息描述

```
{"DeviceID":设备ID,"MsgType":消息类型（heartbeat：心跳）}
```

（2）定位数据
原始数据（不带锁事件）：

```
7E0200003A790011000094180C00000000234000201E807C80713B76A00780000000021080
4150651D4014AD502018C30011A310108FE0400000BCCFD0901CC000000308D51D0F27E
```

返回消息：


```
{
    "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"
    }
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间），
        "Latitude": 纬度（WGS84），
        "Longitude": 经度（WGS84），
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关），
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%,255表示电池充电中），
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开），
        "LockRope": 锁绳状态（0：插入；1：拔出），
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏报警；5:出围栏报警），
        "DataLength": 数据长度（字节数），
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位），
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值
    },
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
```

```
    "MsgType": 数据类型（Location：定位数据）
}
```

原始数据（带锁事件）：

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

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": "790011000094",
    "DataBody": {
        "GpsTime": 定位时间（UTC时间）,
        "Latitude": 纬度（WGS84）,
        "Longitude": 经度（WGS84）,
        "MCC": 小区码信息MCC,
        "MNC": 小区码信息MNC,
        "LAC": 小区码信息LAC,
        "CELLID": 小区码信息CI,
        "FenceId": 电子围栏Id,
        "BackCover": 后盖状态（1：开；0：关）,
        "Index": 流水号,
        "Awaken": 唤醒源 （1：RTC闹钟唤醒，2：Gsens震动唤醒 3：开盖唤醒 4：剪绳唤醒 5：充电唤醒 6：刷卡唤醒7：Lora唤醒8：VIP号码唤醒9：非VIP唤醒10：蓝牙唤醒）
        "ProtocolVersion":协议版本号,
        "Direction": 正北为0，顺时针0-360,
        "Battery": 电量值（0~100%,255 表示电池充电中）,
        "GpsSignal": GPS当前接收到的卫星颗数,
        "Voltage": 电压值，单位V,
        "Speed": 速度，单位km/h,
        "LockStatus": 锁状态（0：关；1：开）,
        "LockRope": 锁绳状态（0：插入；1：拔出）,
        "Mileage": 里程值，单位km,
        "Alarm": 报警类型（1：超速报警；2：低电报警；3：主机开盖报警；4：进围栏报警；5:出围栏报警）,
        "LockEvent": {
            "Type": 事件类型（参见附件1）,
            "FenceId": 事件关联的围栏ID（-1：与围栏无关）,
            "UnLockStatus": 开锁状态（1：开锁成功；0：开锁失败）,
            "Password": 开锁密码
            "CardNo":如果事件类型为刷卡开锁，这里表示的是卡号
        },
        "DataLength": 数据长度（字节数）,
        "LocationType": 定位方式（0：不定位；1：GPS定位；2：基站定位）,
        "Altitude": 海拔高度km,
        "GSMSignal": GSM信号值
    },
```

```
    "ReplyMsg": 需要回复终端的内容（为空则表示不需要回复），
    "MsgType": 数据类型（Location：定位数据）
}
```

（3）指令数据解析
原始数据：

```
28373931303035303030303032c312c3030312c424153452c312c4a54373039415f3230323
0303932325f48572d56312e305f4e6f524649445f53494d434f4d373630305f56325f312c30
2c4c45313142303353494d373630304d31315f412c383938363034313231303138433037333
83535342c383637353834303334363131313131362c3436302c302c313830353535343736342c31
3033343229
```

返回消息：

```
{
    "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"
}
```

返回消息描述

```
{
    "DeviceID": 设备ID,
    "DataBody": 消息体内容,
    "ReplyMsg": 回复设备的消息（为空则表示不需要回复），
    "MsgType": 指令类型
}
```

3.指令消息

3.1.查询终端基本信息

消息体内容（DataBody）：

(791005000003,1,001,BASE,1,JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1,0,LE11B03SIM7600M11_A,898604121018C0738554,867584034611116,460,0,180554764,10342)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,1	指令类型（BASE1）
3	JT709A_20200922_HW-V1.0_NoRFID_SIMCOM7600_V2_1	当前设备的版本号。
4	0	前面的0表示英文，1表示其它语言的Unicode码
5	LE11B03SIM7600M11_A	GSM模块版本。
6	898604121018C0738554	SIM卡的ICCID。
7	867584034611116	GSM模块的IMEI号。
8	460,00,4243,6877	网络信息：460 移动国家代码，即MCC信息，此处表示中国；00电信运营商网络号码，MNC信息（中国移动为00，中国联通为01）；4243 基站编号CELL ID信息；6877 位置区域码LAC信息。CELL ID与LAC为十六进制，即4243转为十进制为16963。

3.2.授时(同步GMT时间)

消息体内容（DataBody）：

(700160818000,1,001,BASE,2,20111018123820)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,2	指令类型（BASE2）
3	20111018123820	设置的时间(yyyyMMddHHmmss)

3.3.远程重启终端

消息体内容（DataBody）：

(700160818000,1,001,BASE,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,3	指令类型（BASE3）

3.4.恢复出厂设置

消息体内容（DataBody）：

(700160818000,1,001,BASE,4)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,4	指令类型（BASE4）

3.5.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,5, 1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,5	指令类型（BASE5）
3	1,1,1,1,1,1,1,1,1,1,1	从左至右依次为(1)锁挂绳剪断报警、(2)刷非法卡报警、(3)开锁状态保持一段时间报警、(4)指令开锁密码连续输错5次报警、(5)震动报警、(6)进区域报警、(7)出区域报警的开关、(8)低电报警（电量低于满电的30%）、(9)开后盖报警、(10)卡锁报警、(11)

		超速报警;(每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)

3.6.查询/设置上传间隔和休眠定时唤醒间隔

消息体内容（DataBody）：

(700160818000,1,001,BASE,6,60,30)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,6	指令类型（BASE6）
3	60	上传间隔，以秒为单位，默认60。最长1小时，最短5秒
4	30	设备正常工作间隔，单位分钟，默认60分钟. 当该值为0时，启用持续追踪模式，设备不休眠。最长24小时，最短5分钟

3.7.查询/设置终端休眠模式

消息体内容（DataBody）：

(700160818000,1,001,BASE,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,7	指令类型（BASE7）
3	1	0正常休眠模式任一中断、RTC可以唤醒，1表示在0模式上增加短信电话可以唤醒

3.8.查询/设置终端本地时差

消息体内容（DataBody）：

(2310915002,1,001,BASE,8, 480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,8	指令类型（BASE8）
3	480	时差值，此处单位为分钟

3.9.查询/设置终端本地时差

消息体内容（DataBody）：

(700160818000,1,001,BASE,9,8613998765432,0,0,0,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,9	指令类型（BASE9）
3	8613998765432,0,0,0,0	5个监控手机号码,0表示没有设置

3.10.查询/设置主从IP地址与端口号、APN及用户名与密码等参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,10,211.154.112.98,1088,211.154.112.98,1088,CMNET,abc,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,10	指令类型（BASE10）
3	211.154.112.98	主IP（域名）

4	1088	主端口
5	211.154.112.98	从IP（域名）
6	1088	从端口
7	CMNET	APN名称
8	abc	用户名
9	123456	密码

3.11.查询/设置/删除设备工作闹钟

消息体内容（DataBody）：

(700160818000,1,001,BASE,32,1,480)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,32	指令类型（BASE32）
3	1	则表明当前有效闹钟个数；返回0表示当前没有有效闹钟
4	480	设置一个设备闹钟的时间，以分钟为单位（范围为：0~1439）

3.12.查询/设置仅支持VIP号码

消息体内容（DataBody）：

(700160818000,1,001,BASE,40,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,40	指令类型（BASE40）
		1表示仅支持VIP号码进行设备配置和查询，0表示非vip号码也可以

		配置和查询。默认0

3.13.查询/设置里程统计相关参数

消息体内容（DataBody）：

(700160818000,1,001,BASE,43,10,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,43	指令类型（BASE43）
3	10	表示里程统计的速度限制值操作(即速度小于改值不统计里程,默认10km/h, 参数0-100)
4	10	表示设置里程同步值为10公里

3.14.查询/设置静态飘移处理功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,44,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,44	指令类型（BASE44）
3	1	1打开这个静态漂移处理功能，0表示关闭静态漂移处理功能

3.15.查询/设置GPS追踪定位模式功能

消息体内容（DataBody）：

(700160818000,1,001,BASE,45,1)

消息体内容描述：

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,45	指令类型（BASE45）
3	1	1表示节能定时追踪，0 表示节能定位追踪，2表示全速追踪

3.16.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,46,5,25,6)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,46	指令类型（BASE46）
3	5	HDOP门限值
4	25	单颗卫星信噪比门限值
5	6	联合定位卫星个数门限值

3.17.查询/设置GPS定位判定门限参数(防GPS定位漂移)

消息体内容（DataBody）：

(700160818000,1,001,BASE,47,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1	1: 启用监控IP服务器，0关闭监控IP服务器

3.18.设置报警开关指令

消息体内容（DataBody）：

(700160818000,1,001,BASE,48, 1,1,1,1,1,1,1,1,1,1,1,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	BASE,47	指令类型（BASE47）
3	1,1,1,1, 1,1,1,1, 1,1,1,1,1	从左至右依次为(1) 锁绳剪断报警、(2)刷非法卡报警、(3)长时间开锁报警、(4)开锁密码错误、(5)震动报警、(6)进区域报警、(7)出区域报警、(8)低电报警、(9)开后盖报警、(10)卡锁报警、(11)GSM信号低报警、(12)超速报警 (每个报警开关参数可以取值为0,1,2,3,并且可以任意组合,0表示GPRS和SMS报警都关闭,1表示只开启GPRS报警,2表示只开启SMS报警,3表示GPRS和SMS报警都开启.)，(13)倾斜报警，其中红色的没有相应报警

3.19.查询/设置震动检测阈值

消息体内容（DataBody）：

(700160818000,1,001,GSENS,1, 500)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GSENS,1	指令类型（GSENS1）
3	500	500：单位是mg（gsensor芯片63mg为阶梯配置，实际芯片配置值为500/63取整=7），最小63，最大值8000，默认500。当震动达到这个值，会检测到一次震动。如果值等于0，关闭该功能

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,7)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,7	指令类型（DEBUG7）

3.20.电池产品入库前进入休眠

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,15, 10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,15	指令类型（DEBUG15）
3	10	0表示马上进入睡眠，设置大于0的数表示相应时间后进入休眠

3.21.查询物联卡信息

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,27, ICCID, IMSI)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,27	指令类型（DEBUG27）
3	ICCID	ICCID号码
4	IMSI	IMSI号码

3.22.查询/清除历史(盲区)数据

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,29, 0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,29	指令类型（DEBUG29）
3	0	返回0则表明当前历史数据条数

3.23.查询/设置超速报警速度

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,30,120,10)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,30	指令类型（DEBUG30）
3	120	报警的速度临界值(默认值120)，单位为:公里
4	10	超速此时间后报警(默认值10),单位：秒

3.24.查询/设置低电报警数值

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,31,5)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,31	指令类型（DEBUG31）

3	5	报警的电量临界值；5表明当前电量百分比

3.25.查询/设置关闭看门狗作用

消息体内容（DataBody）：

(700160818000,1,001,DEBUG,38,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	DEBUG,38	指令类型（DEBUG38）
3	0	返回0表明已接收该指令，并生效。

3.26.设置/查询区域详细节点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,7,8,15,1,10,11323.1234...)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,7	指令类型（GFCE7）
3	8	8表示第八个区域
4	15	15表示总点数
5	1	1表示当前页
6	10	10表示当前页的点数
7	11323.1234...	各个点的经度与纬度

3.27.清除设置的全部区域点信息

消息体内容（DataBody）：

(700160818000,1,001,GFCE,8,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	GFCE,8	指令类型（GFCE8）
3	1	1表示删除围栏的ID

3.28.查询/设置没有关锁报警提醒时间间隔

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,2, 1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,2	指令类型（ELOCK2）
3	1	提醒时间，单位(分钟)。当开锁后没有关锁，如果此值不为0(为0无效)时间到就上报开锁提醒消息，最小值为1分钟，最大值为10分钟。默认为1分钟

3.29.终端上报动态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,3,123456)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID

2	ELOCK,3	指令类型（ELOCK3）
3	123456	上报的动态密码

3.30.设置/修改远程开锁静态密码

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,4,666666)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,4	指令类型（ELOCK4）
3	666666	666666: 表示修改成功的密码。

3.31.凭静态或动态密码开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,5,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,5	指令类型（ELOCK5）
3	1	0: 表示密码正确，大于0表示密码错误开锁失败

3.32.查询/设置围栏外禁止开锁

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1,2,3,4)

消息体内容描述：

序	示例	说明

号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示设置了栏外禁止开锁，必须在定位状下围栏内才能开锁。2表示设置了栏外禁止开锁，有定位必须在围栏内开锁，没有定位可以通过远程静态密码或动态密码开锁，如果设置了围栏外禁止开锁没有定位不能通过现场按键或蓝牙APP开锁。0表示没有开启围栏外禁止开锁功能，默认没有开启。
4	2	围栏ID个数：1~10有效，只支持区域围栏。其它类型的围栏不支持
5	3，4	围栏ID： 1~10有效，3,4表示只能在区域围栏ID为3,4的围栏可以开锁，1～10的ID存在对应的位置

3.33.动态密码权限处理(不对外公开)

消息体内容（DataBody）：

(700160818000,1,001,ELOCK,7,1)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,7	指令类型（ELOCK7）
3	1	1表示使能产生动态密码；0表示关闭产生动态密码

3.34.查询/增删/开锁授权号

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,15,0,2,20,2,1,0013953759, 0013953758, 0013953757，，，，，，，，，，，，，)

消息体内容描述：

序号	示例	说明

1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	0	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	2	2为查询第2组数据，一共分3组，分别为1~3
5	20	20表示有20个ID号
6	2	2表示总共有2应答包
7	1	1表示当前是第1个应答包
8	0013953759, 0013953758, 0013953757，，，，，	表示该组存储的授权号列表

消息体内容（DataBody）实例2：

(700160818000,1,001,ELOCK,15,1,3,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	1	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	增加3个授权卡号成功
5	200	当前总共有200个卡

消息体内容（DataBody）实例3：

(700160818000,1,001,ELOCK,15,2,3)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	2	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	3	表示删除3个授权卡号成功。(每次最多一次性删除20)

消息体内容（DataBody）实例4：

(700160818000,1,001, ELOCK,15,3,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,15	指令类型（ELOCK15）
3	3	操作模式， 0表示查询终端已存所有授权号，1表示要增加开锁授权号操作，2表示删除授权号，3表示删除所有的授权号。
4	0	表示删除全部授权卡号成功，目前剩余0个授权卡

3.35.现场刷卡授权模式配置指令

消息体内容（DataBody）实例1：

(700160818000,1,001, ELOCK,16,2,0013953759,0013953751)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	2	授权的卡个数

4	0013953759,0013953751	授权的卡号

消息体内容（DataBody）实例2：

(700160818000,1,001, ELOCK,16,0)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	ELOCK,16	指令类型（ELOCK16）
3	0	1表示打开批量增加终端开锁授权号功能， 0表示关闭批量增加终端开锁授权号功能。

3.36.通知设备MCU升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,1,1,20120102)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,1	指令类型（OTA1）
3	1	1：表示同意升级，只有当设备的当前版本号低于要升级的版本号的时候才返回1.若设备返回0表示拒绝升级，即设备当前的Firmware版本和升级的Firmware版本相同或者更高。服务器只有在收到同意升级以后才发送第一个Firmware数据包.
4	20120102	当前设备的Firmware版本号.

3.37.发送Firmware数据包

消息体内容（DataBody）：

(700160818000,1,001,OTA,2,0,200,100)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,2	指令类型（OTA2）
3	0	1表示保存成功，0表示保存失败,如果失败重发.
4	200	接收到的Firmware数据包序号.
5	100	下一条需要发送的数据包序号.

3.38.取消Firmware升级

消息体内容（DataBody）：

(700160818000,1,001,OTA,3,1,20120222)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,3	指令类型（OTA3）
3	1	1表示取消成功，0表示取消失败。如果该指令的版本号和正在升级的固件版本号不符的话就可能取消失败.
4	20120222	设备正在升级的版本号

3.39.完成Firmware传输

消息体内容（DataBody）：

(700160818000,1,001,OTA,4,1)

消息体内容描述：

序	示例	说明

号	示例	说明
1	700160818000	设备ID
2	OTA,4	指令类型（OTA4）
3	1	该参数可以为1或者为0，1表示接受该指令，并根据指令执行。0表示：接受的数据不完全，拒绝升级.

3.40.查询发送下条Firmware数据包序号

消息体内容（DataBody）：

(700160818000,1,001,OTA,5,1,200)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,5	指令类型（OTA5）
3	1	1：代表目前正在接收的固件和查询的相同，0表示目前接收的固件和查询的不同.
4	200	如果查询的固件版本号比设备的版本号新的话，该参数表示下一条需要发送的序号。如果查询的版本号和正在升级的版本号不同的话，则该参数为1.即代表需要重新下载现在查询的固件.

3.41.查询当前stm32固件的版本号

消息体内容（DataBody）：

(700160818000,1,001,OTA,6,JT709A_V103R001)

消息体内容描述：

序号	示例	说明
1	700160818000	设备ID
2	OTA,6	指令类型（OTA6）

3	JT709A_V103R001	固件版本号

附件1

事件ID（16进制）	事件ID	事件描述
0x01	1	静态密码远程开锁
0x02	2	动态密码远程开锁
0x03	3	动态密码现场(蓝牙或WIFI)APP开锁
0x05	5	表示静态密码现场(蓝牙或WIFI)APP开锁
0x06	6	错误的静态密码远程开锁
0x07	7	错误的动态密码远程开锁
0x08	8	错误的动态密码现场(蓝牙或WIFI)APP开锁
0x0B	11	长时间开锁事件
0x0C	12	锁绳剪断事件
0x0D	13	关锁事件(包括自动关锁和远程关锁)
0x10	16	远程执行开锁异常，没有定位不执行开锁
0x11	17	远程执行开锁异常，有定位在围栏外不执行开锁
0x12	18	电机异常
0x18	24	蓝牙执行开锁异常，没有定位不执行开锁
0x19	25	蓝牙执行开锁异常，有定位在围栏外不执行开锁
0x1C	28	开锁并拔出锁绳
0x1E	30	短信静态密码远程开锁
0x1F	31	短信动态密码远程开锁
0x20	32	表示错误的短信动态密码
0x22	34	刷授权卡开锁事件
0x23	35	刷非法卡开锁事件

0x28	40	错误的静态密码现场(蓝牙或WIFI)APP开锁
0x29	41	错误的短信静态密码开锁
0x2A	42	RFID执行开锁异常，没有定位不执行开锁
0x2B	43	RFID执行开锁异常，有定位在围栏外不执行开锁