远望谷

助推物联 享受生活

# RFID读写器通用API

# 技术参考手册

深圳市远望谷信息技术股份有限公司
Invengo Information Technology Co.,Ltd.

# 目录

# 一、引言

## 1、目的

本文档概要的介绍了远望谷读写器基本运行原理，详细的描述了远望谷读写器与上位机（控制端）的数据通信应用程序编程接口（Application Programming Interface，以下简称API），上位机（控制端）可通过API控制读写器与RFID电子标签之间的数据通信。

## 2、范围

适用的远望谷读写器型号：

- **固定式：** XC-RF807、XC-RF850、XC-RF861、XC-RM825、XC-RM829、XC-RF812、XC-RF811(V3.0)。
- **手持式：** XC2900、XC2903

固定式读写器API适用范围如下：

- **C# API：** 基于Microsoft .NET Framework v2.0框架编写，支持.NET家族语言调用，进行二次开发。
- **JAVA API：** 基于JDK1.5编写，仅支持1.5以上版本的JDK环境进行调用及二次开发
- **C++ API：** 基于Microsoft vc 6.0编写,支持c++,dephi,vb等语言在windows环境下调用及二次开发。

手持式读写器API适用范围如下：

- **C# API：** 基于Microsoft .NET Compact Framework v2.0框架编写。

## 3、定义

- **IRP**：Invengo Reader Protocol，远望谷读写器数据传输控制协议。该文档只讲述IRP1.0版本。
- **API**：应用程序接口。可供二次开发使用。是对读写器通讯协议的封装。
- **上位机**：指用于与读写器进行数据交互的个人电脑[PC]或其他控制终端。
- **下位机**：指读写器。
- **COM（cluster communication port）**：串行通讯端口。
- **TCP（Transfer Control Protocol）**：传输控制协议。
- **USB（Universal Serial BUS）**：通用串行总线。
- **Q值**：系指读写器用于调整标签响应概率的一个参数。在一个盘存周期中，读写器指定标签装入一个Q 位的随机（伪随机）数进入其时隙计数器，读写器也可以命令标签对它们时隙计数器中的值进行减1操作。标签在其时隙计数器的值为零时响应读写器的命令。Q是一个（0，15）范围内的正整数，相应的标签响应概率范围从$2^0$=1到$2^{-15}$ =0.000031。
- **RSSI（Received Signal Strength Indication）**：标签回波强度。
- **UTC(UTC, Universal Time Coordinated)**：通用协调时。默认以1970-1-1 00:00:00为起点。
- **BCD码（Binary-Coded Decimal）**：用4位二进制数来表示1位十进制数中的0~9这10个数码。
- **字**：两个字节为一字。

## 4、参考资料

- EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz－960 MHz Version 1.2.0
- IMPINJ Monza4 Datasheet
- NXP SL3ICS1002 G2XM Datasheet
- 远望谷读写器数据传输协议
- 示例代码

## 5、缩略语

- **6C**：ISO 18000-6C。
- GB：GB/T 29768-2013协议，下文简称为国标。
- **6B**：ISO 18000-6B。
- **807**：XC-RF807型读写器
- **850**：XC-RF850型读写器
- **861**：XC-RF861型读写器
- **825**：XC-RM825型读写器模块
- **829**：XC-RM829型读写器模块
- **812**：XC-RF812型发卡器
- **811**：XC-RF811型发卡器
- **503B**：XC-RF503-B型发卡器
- **2900**：XC2900型便携式读写器
- **2903**：XC2903-F6C型便携式读写器
- **上位机消息**：上位机主动发送的消息。该类消息必须实现IMessage接口，详见4.3节。
- **读写器消息**：读写器主动发送的消息。该类消息必须实现IMessageNotification接口，详见4.2节。

## 6、阅读说明

本API采用三种语言编写，为方便说明通用部分采用UML语法。

如果没特殊说明，则通用于固定式和手持式读写器。

# 二、概述

## 1、读写器基本硬件框架

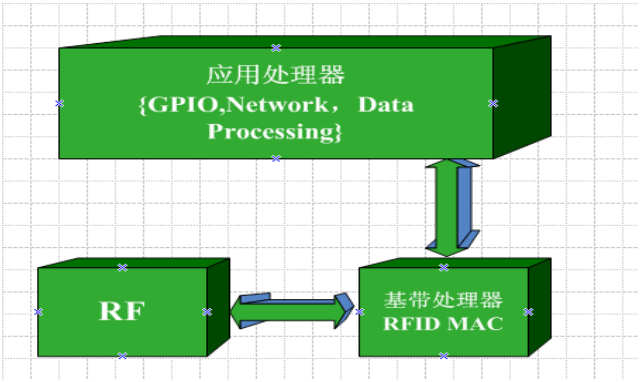



图2-1

读写器基本硬件由应用处理单元、RFID基带处理单元、RF射频硬件电路单元组成。

应用处理单元：主要负责读写器与上位机的网络通信、数据处理、GPIO等与应用相关的处理。

RFID基带处理单元：主要负责读写器与标签之间的数据交换和协议流程控制。

RF电路：负责读写器与标签之间的物理信号传递。

XC-RM825、XC-RM829只提供基本的通信接口（RS232）和RFID功能，在硬件上应用处理单元与基带处理单元是合并的，采用的是单CPU方案。

XC-RF807、XC-RF850、XC-RF861具备完备的通信接口（RS232）和网络通信能力，同时具有较强的数据处理力，为支撑较大规模的组网应用和复杂应用业务逻辑处理，其应用处理单元与RFID基带处理单元在硬件上采用了双CPU架构。

## 2、读写器基本运行机制

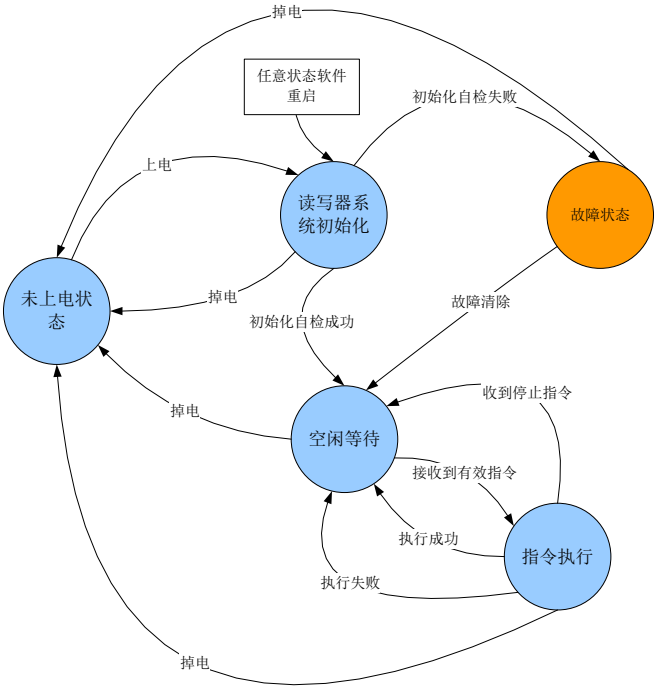



图2-2

读写器系统初始化：操作系统启动，各功能模块硬件状态自检，读写器系统参数初始化，此状态下读写器无法接收和执行任何上位机指令。

空闲等待：读写器完成初始化等待上位机下发指令的状态，此状态下读写器可接收任何指令并立即执行。

指令执行：读写器收到完整合法的上位机指令后会立即切换到指令执行状态，读写器在执行循环操作的读写卡指令时只会响应停止操作、GPIO输入输出操作和参数查询操作。

故障状态：在系统上电初始化和自检过程出现故障时，系统会进入故障状态，此状态主要实现故障告警和系统调试接口，方便研发调试和生产过程中快速定位故障原因。

## 3、协议基本框架

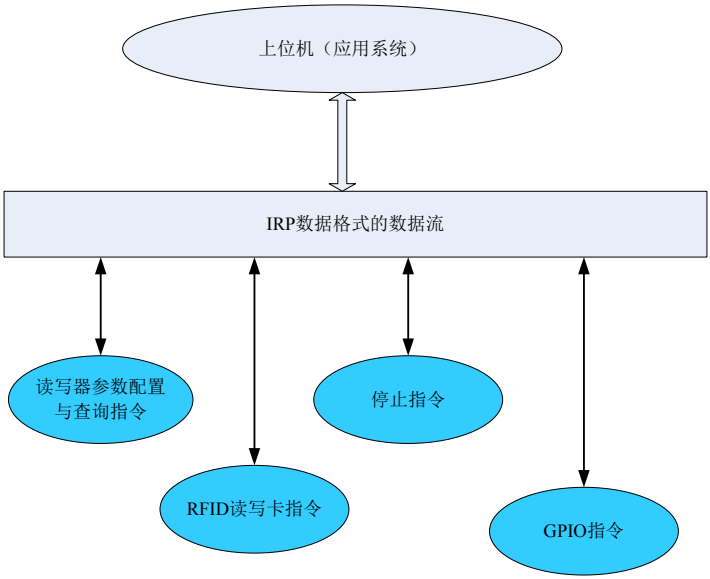



图2-3

远望谷读写器协议基本框架见上图，读写器指令主要包括以下几大类：读写器参数与配置、RFID读写卡、停止指令、GPIO指令。

## 4、API组成

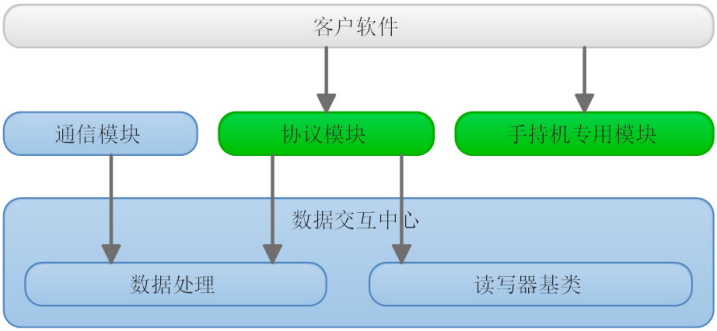



图2-4 API层次结构图

如图，除了客户软件外，其余部分为API。上图中绿色模块为二次开发客户使用模块，蓝色部分客户无需关心。

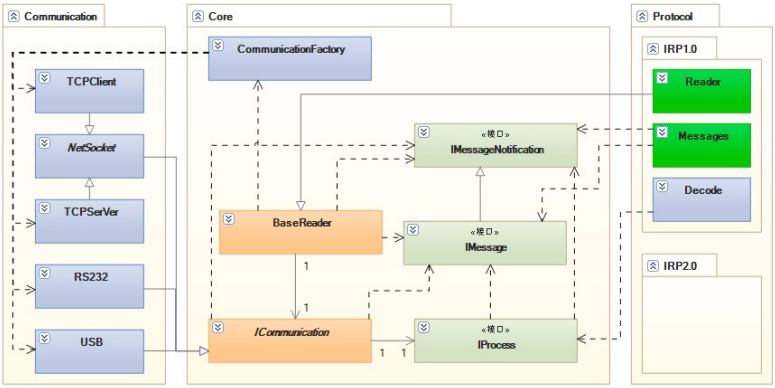



图2-5 API类图（除手持式读写器专用模块外）

API总共分为四大模块：

**① 通讯模块**

对应层关系图中的通讯模块，上图中的“Communication”包即为通讯模块，图中列出的是PC机的四种通讯模式。

**② 协议模块**

对应层关系图中的指令模块，上图中的“Protocol”包即为协议模块，目前只支持IRP1.0协议。

**③ 核心模块**

对应层关系图中的数据交互中心，上图中的“Core”包即为核心模块。该模块使用工厂模式与通讯模块结合，使用了适配器模式与通讯模块交换信息。

**④ 手持式读写器专用模块**

远望谷手持式读写器专用模块，控制远望谷手持式读写器RFID模块、GPRS、Wi-Fi等硬件设备。详见第9章。

固定式读写器提供三套编程语言API可供选择，分别为：C#,JAVA和C++。手持式读写器提供C#语言API。

固定式读写器API组成如下：

- C# API包括：
  - √ RFIDInterface.dll：数据交互中心
  - √ Communication.dll：通信模块
  - √ IRP1.dll、IRP1Message.xml：远望谷协议指令模块
  - √ language文件夹：包含多个语言包的XML文件，ErrCode(****).xml
- JAVA API包括：
  - √ IRP1.jar：远望谷协议模块
- C++ API包括：
  - √ XCRFAPI.DLL　读写器交付api
  - √ XCRFHEAD　dll所应用的头文件

手持式读写器API的主要组成如下：

- C# API包括：
  - √ RFIDInterface.dll：数据交互中心
  - √ Communication.dll：通信模块
  - √ IRP1.dll、IRP1Message.xml：远望谷协议指令模块
  - √ language文件夹：包含多个语言包的XML文件，ErrCode(****).xml
  - √ Invengo.dll： 提供常用的操作
  - √ Invengo.Devices.ModuleControl.dll： 模块电源控制
  - √ Invengo.Barcode.dll：霍尼韦尔或迅宝条码识读

# 三、读写器通信方式

## 1、串口通信

所有读写器都支持RS232接口。通过串口与读写器进行通信须设置好读写器串口通信参数，如波特率。读写器出厂默认的串口（RS232）通讯参数为：波特率115200，8位数据位，1位停止位，无校验。

## 2、以太网通信

读写器提供的以太网通信（基于TCP/IP协议，仅支持IPV4）包括读写器作为服务器模式（也称被动模式）和读写器作为客户端模式（也称主动模式）两种，任一时刻，读写器只能处于这两种模式中的一种。读写器的服务器模式是指外部设备（如PC端）主动向读写器发起的网络连接。读写器的客户端模式是指读写器主动向外部设备发起的网络连接。读写器出厂时，默认配置为服务器模式，读写器的出厂默认IP为192.168.0.210，使用网络连接时，请确保读写器的IP和PC端的IP处于同一网段。

读写器作为客户端，工作时按照设置的参数主动连接远端服务器的一种工作模式。主动模式和备选服务器配合使用，在主服务器故障时，可自动连接到设置的备选服务器上。

主动模式由主动模式开关控制。当主动模式开关开启时，表示启用主动模式，此时被动模式不可用。当主动模式开关关闭时，表示启用被动模式，主动模式功能自动关闭。

# 四、API总述

API主要作为客户软件和读写器通讯之间的接口，主要包括读写器类和一系列的消息类，通过这些类实现与读写器建立、断开连接以及发送、接收消息等功能。

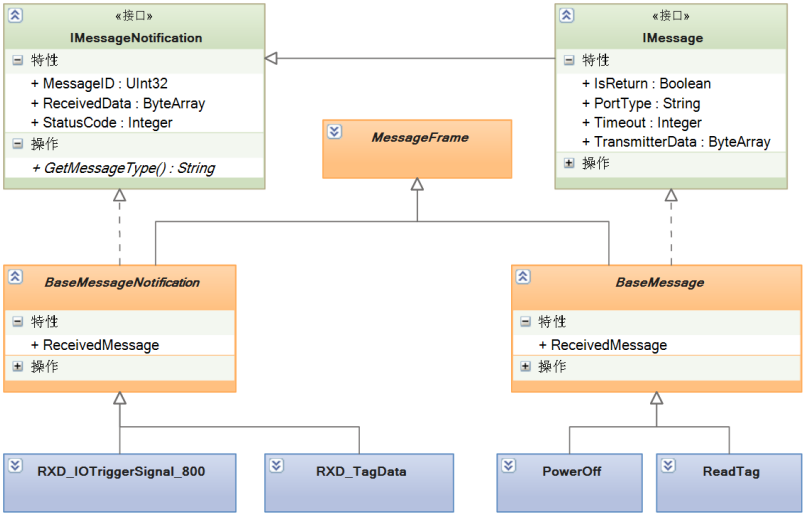



图4-1 消息类图

消息类封装了远望谷协议的指令信息，如上图中的读写器消息RXD_IOTriggerSingnal_800类、RXD_TagData类和上位机消息PowerOff类、ReadTag类等。

读写器类如下图，封装了读写器对消息类的操作和读写器自身状态属性等。

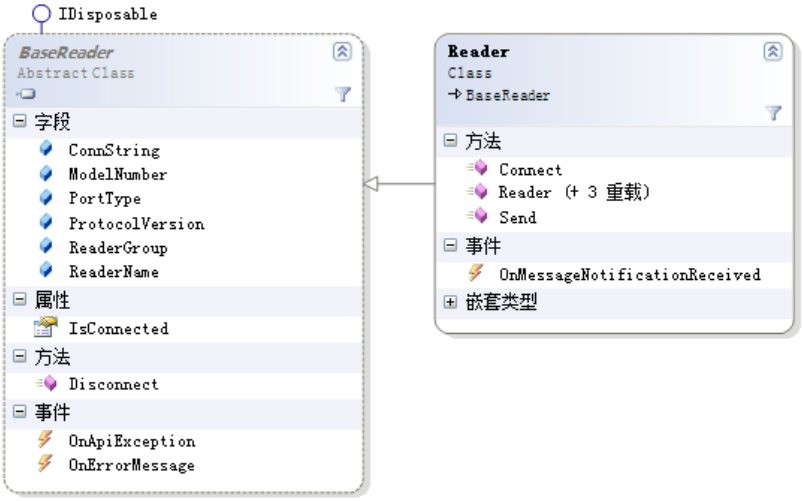



图4-2　读写器类图

这些类存在于以下文件中。注意：实际开发过程中应将“API组成”章节里提到的文件全部包括在内。

C#：

命名空间：Invengo.NetAPI.Protocol.IRP1

程序集： IRP1（在 IRP1.dll 中）

JAVA：

包名：invengo.javaapi.protocol.IRP1（在IRP1.jar中）

C++：

头文件在IRP1APIHEAD//IRP1_H目录下

# 1、读写器类（Reader类）

该类针对一个通讯通道进行操作。封装了读写器的连接、断开、发送数据、接收数据四大核心操作。

## （1）构造函数

**用途：**

实例化Reader子类。

**语法：**

```
+ Reader(readerName: String)
+ Reader(readerName: String,
          portType: String,
          connStr: String)
+ Reader(server : Socket)
```

**参数：**

readerName：读写器名称，字符串。
portType：通信端口类型，字符串。
connStr：连接字符串。
server：读写器连接Socket，用于主动模式，见3.2节。

**参数说明：**

读写器名称自定义，但不能重复。

通信端口类型定义如下：

| portType | 说明 |
|---|---|
| RS232 | RS-232串口 |
| TCPIP_Client | TCP/IP协议客户端 |

连接字符串规则如下：

串口连接字符串例子：“COM1,115200”。其中“COM1”为串口名称，“115200”为波特率，中间以半角的逗号隔开。常见波特率有“115200”、“19200”、“9600”。

TCP/IP协议客户端连接字符串例子：“192.168.0.210:7086”。其中“192.168.0.210”为IP，“7086”为端口号，中间以半角的冒号隔开。

**构造函数说明：**
构造函数一：该构造函数必须与配置文件(见4.5节)配合使用。
构造函数二：该构造函数不与配置文件配合使用。
构造函数三：当上位机为服务端时，将已连接的Socket传入以备使用。

### （2）建立连接（Connect方法）

**用途：**
使用指定的端口与读写器建立通信连接。

**语法：**
+ Connect() : Boolean

**返回值：**
是否连接成功，布尔型。

**备注：**
客户软件作为服务端时，虽然传入的是已连接的Socket，但还是需要调用该方法才能正常使用Reader类。

### （3）断开连接（Disconnect方法）

**用途：**
对已经建立连接的端口断开连接。

**语法：**
+ Disconnect()

### （4）发送消息（Send方法）

**用途：**
发送消息。

**语法：**
+ Send(msg : IMessage) : Boolean
+ Send(msg : IMessage, timeout : Integer) : Boolean

**参数：**
msg：待发送消息具体类，IMessage类型，注意IMessage为消息接口，其所有实现该接口的类都能作为参数。

timeout：超时时间，整型，单位为毫秒。

**返回值：**
消息是否执行成功，布尔型。

**注意：**

- 需要响应的消息（见IMessage接口IsReturn成员），消息发送成功并且返回成功执行的消息才算成功，否则视为失败。
- 不需要响应的消息（见IMessage接口IsReturn成员），发送的字节数等于要发送的字节数则返回成功。
- 没有超时参数的方法，默认超时为1000毫秒。

### （5）API异常通知（OnApiException事件）

**用途：**
API运行时产生的异常，统一到该事件来进行通知。

**语法：**
```
+ OnApiException : ApiExceptionHandle
```

**备注：**
该事件通过实现ApiExceptionHandle委托，回调异常消息。绝大多数API异常都经此进行通知。


图4-3 ApiExceptionHandle委托

参数e为ErrInfo类型。ErrInfo类型包含抛出错误读写器的名称（ReaderName）和错误信息内容（ErrMsg）。

### （6）消息通知（OnMessageNotificationReceived事件）

**用途：**
接收读写器消息。

**语法：**

+ OnMessageNotificationReceived

:MessageNotificationReceivedHandle

**备注：**

该事件通过实现MessageNotificationReceivedHandle委托，回调读写器消息。

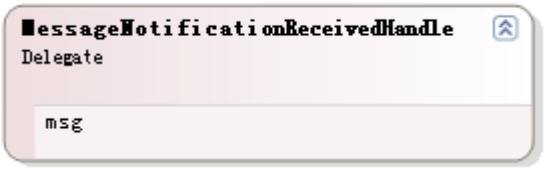



图4-4 MessageNotificationReceivedHandle委托

参数msg为IMessageNotification类型，详见4.2节说明。

### （7）是否连接标志属性（IsConnected成员）

**用途：**

获取是否连接标志。

**语法：**

+ IsConnected : Boolean

**备注：**

该成员为布尔型。在使用时注意，读写器异常断开不一定能被API及时发现。

### （8）读写器型号属性（ModelNumber成员）

**用途：**

获取读写器型号。

**语法：**

+ ModelNumber : String

**备注：**

字符串类型。初始值为“unkown”，只有在读写器成功连接后该成员才会有具体值。

### （9）读写器名称属性（ReaderName成员）

**用途：**
获取读写器名称。

**语法：**
+ ReaderName ： String

**备注：**
字符串类型。同构造函数中readerName参数。

### （10）读写器组别属性（ReaderGroup成员）

**用途：**
获取读写器组别名称。

**语法：**
+ ReaderGroup ： String

**备注：**
字符串类型。自定义字符串。为方便用户将读写器归类，使用配置文件中的分组信息。

### （11）通信协议属性（ProtocolVersion成员）

**用途：**
获取通信协议及版本信息。

**语法：**
+ ProtocolVersion ： String

**备注：**
字符串类型。“IRP1”表示远望谷协议IRP1.0版本。

### （12）通信端口类型属性（PortType成员）

**用途：**
获取通信端口名称。

**语法：**
+ PortType ： String

**备注：**
字符串类型。同构造函数中portType参数。

## 2、读写器消息

读写器消息：读写器主动发送的消息。该类消息必须实现IMessageNotification接口。

消息包括：

- 读写器非应答而主动发送的消息，如读写器I/O口输入信号改变，读写器主动发送消息通知上位机；
- 读写器应答，但不是一对一的应答，如返回标签扫描信息。

读写器消息用法流程如下图所示：（假设读写器消息msg已经通过OnMessageNotificationReceived事件回调）

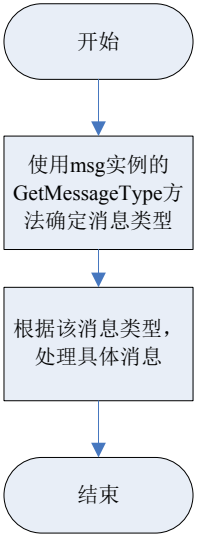



图4-5 读写器消息用法流程

注意：

- 可通过msg实例的StatusCode确定消息状态，然后进一步处理；StatusCode非0则调用msg的ErrInfo显示错误信息。
- 回调消息和发送消息处于不同线程，请注意消息的同步处理。

读写器消息成员和方法如下：

**① 获取当前类的类型名称（GetMessageType方法）**

用途：获取当前类的类型名称。注意，如果当前类是通过接口引用，则获取的是派生类的名称。

语法：+ GetMessageType() : String

返回值：类型名称字符串。

备注：实际应用中，可在OnMessageNotificationReceived事件或回调中，使用GetMessageType方法获得事件或回调参数的类型名称，以此判断返回的读写器消息类型。

**② 获取消息返回状态码（StatusCode成员）**

用途：定义接口，该接口用于实现获取消息返回状态码，整型。

语法：+ StatusCode : Integer

备注：在IRP1.0协议中该状态码为0表示成功，非0表示对应的错误码。错误码详见附录。

**③ 获取消息错误信息（ErrInfo成员）**

用途：定义接口，该接口用于实现获取消息错误信息，字符串型。

语法：+ ErrInfo : String

## 3、上位机消息

上位机消息：上位机主动发送的消息。该类消息必须实现IMessage接口（IMessage继承IMessageNotification）。

消息包括：

- 上位机发送的需要读写器应答的消息；
- 上位机发送的不需要读写器应答的消息；

- 上位机应答读写器的消息。

上位机消息用法流程如下图所示：（假设已经实例化读写器类，并成功建立连接）

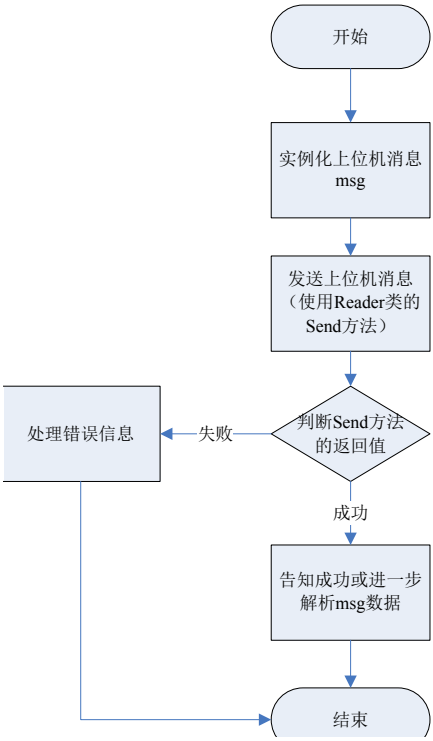



图4-6 上位机消息用法流程

注意：

- 消息发送失败，可通过msg实例的ErrInfo取出错误信息；
- 消息发送成功，可通过msg实例的ReceivedData取出接收到的数据桢，视具体情况决定是否需要进一步解析。部分消息没有可进一步解析的消息（返回的数据桢中不包含实际数据内容），此时只关心Send方法返回值即可。
- 不建议实例化一次消息使用多次，因为消息响应过一次后该类的状态会有所改变。

上位机消息方法和成员如下：

① 获取或设置消息是否需要等待响应（IsReturn成员）

用途：定义接口，该接口用于实现获取或设置消息是否需要响应，布尔型。

语法：+ IsReturn : Boolean

备注：不需要返回的消息可视为读写器消息。

② 设置消息超时时间（Timeout成员）

用途：定义接口，该接口用于实现设置消息超时时间，整型，单位为毫秒。

语法：+ Timeout : Integer

备注：只有在IsReturn为真时超时时间才起作用。当超时时间起作用时，发送消息后会在该时间内等待消息响应，如果没有响应则视为超时。

## 4、国标标签数据区枚举（GBMemoryBank）

指定国标标签各数据区（C#）。

| 成员名称 | 枚举值 | 说明 |
|---|---|---|
| InformationMemory | 0x00 | 保留区 |
| CodeMemory | 0x10 | 编码区 |
| SafeMemory | 0x20 | 安全区 |
| UserMemory1 | 0x30 | 用户子区1 |
| UserMemory2 | 0x31 | 用户子区2 |
| …… | …… | …… |
| UserMemory16 | 0x3f | 用户子区16 |

指定国标标签各数据区（JAVA）。

| 成员名称 | 枚举值 | 说明 |
|---|---|---|
| GBTidMemory | 0x00 | 保留区 |
| GBEPCMemory | 0x10 | 编码区 |
| GBReservedMemory | 0x20 | 安全区 |
| GBUser1Memory | 0x30 | 用户子区1 |
| GBUser2Memory | 0x31 | 用户子区2 |
| …… | …… | …… |
| GBUser16Memory | 0x3f | 用户子区16 |

## 5、6C标签数据区枚举（MemoryBank）

指定6C标签各数据区。

| 成员名称 | 枚举值 | 说明 |
|---|---|---|
| ReservedMemory | 0 | 保留区 |
| EPCMemory | 1 | EPC数据区 |
| TIDMemory | 2 | TID数据区 |
| UserMemory | 3 | 用户数据区 |

## 6、配置文件（Sysit.xml）

配置文件为可选使用项，但多读写器管理必须使用，文件名：“Sysit.cfg”，xml文件，与API同一目录。

配置文件以读写器节点形式存在，基本内容包括：

- 读写器名称、组别。（读写器名称不能重复）
- 启用/禁用项。启用表示该读写器准备用于连接工作。
- 连接端口类型。
- 连接协议。指定读写器使用协议。
- 连接字符串。

配置文件格式如下图所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<Readers>
  <Reader Name="Reader1" Group="Group1" Enable="true">
    <Port Type="RS232" Protocol="IRP2">COM1,115200</Port>
  </Reader>
</Readers>
```

图4-7 配置文件格式图

其中“Readers”为根节点。

“Reader”节点为读写器配置节点。“Name”属性为读写器名称，“Group”属性为读写器组别，名称和组别为自定义的字符串，名称应保持唯一；“Enable”表示启用/禁用读写器，取值为“true/false”，设置为禁用时，API不会为该项建立连接。

“Port”节点为通讯端口节点，该节点内容为连接字符串。“Type”属性表示

端口名称， “Protocol”属性表示通讯协议，目前只支持“IRP1”。

## 7、天线端口定义

### （1）标签读写操作

标签在写操作时，也使用的是实际天线端口号（即：01H、02H、03H、04H）。

### （2）标签扫描操作

扫描操作指的是标签识别，对应于下文的ReadTag类操作。

读写器在返回上位机的应答中会指出完成该操作时实际使用的天线端口（即：01H、02H、03H、04H）。但在扫描过程中，天线号使用有所差异，如下：

**① 使用以下区域**

ID_6B、

ID_UserData_6B、

EPC_6C_ID_6B、

TID_6C_ID_6B、

EPC_TID_UserData_6C_ID_UserData_6B扫描时

扫描前须先设置天线端口（见SysConfig_500类）。

当使用ReadTag类扫描时，指定读写器工作时使用的天线端口，定义如下：

00H：多天线（即经过设置后的所有天线）

01H：天线1

02H：天线2

03H：天线3

04H：天线4

**② 使用其他区域扫描时**

指定读写器工作时使用的天线端口，位定义如下：

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 定义 | 1 | 0 | 0 | 0 | 天线4 | 天线3 | 天线2 | 天线1 |

Bit3：天线4的使能标志，1：允许；0：禁止
Bit2：天线3的使能标志，1：允许；0：禁止
Bit1：天线2的使能标志，1：允许；0：禁止
Bit0：天线1的使能标志，1：允许；0：禁止

# 五、读写器参数查询与配置

## 1、读写器系统参数查询（SysQuery_800类）

用途：查询读写器系统参数。

### ① 构造函数

C#：public SysQuery_800(Byte parameter)
public SysQuery_800(Byte parameter, Byte data)

JAVA：public SysQuery_800(byte parameter)
public SysQuery_800(byte parameter,byte data)

C++：SysQuery_800(unsigned char parameter)
SysQuery_800(unsigned char parameter, unsigned char data)

参数：
parameter：查询参数。
data：目前只有在查询I/O触发参数操作时使用，表示I/O端口号（01H~04H）。

### ② 指令应答（ReceivedMessage）

查询数据

用途：获取查询数据。

C#：public Byte[] QueryData { get; }

JAVA：public byte[] getQueryData()

C++：bool GetQueryData(unsigned char *data,unsigned char &dataLen)

### ③ 指令说明

详见5.3节。

## 2、读写器系统参数配置（SysConfig_800类）

用途：配置读写器系统参数。

### ① 构造函数

C#：public SysConfig_800(Byte parameter, Byte[] pData)

JAVA：public SysConfig_800(byte parameter, byte[] pData)

C++：SysConfig_800(

unsigned char parameter,

unsigned char *pData,

unsigned char pDataLen)

参数：

parameter：配置参数，详见5.3节。

pData：配置数据，详见5.3节。

pData Len：*pData参数长度。

### ② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### ③ 指令说明

详见5.3节。

## 3、读写器系统参数说明

查询配置功能对照表如下：

| 代码 | 参数类型 | 备注 |
|---|---|---|
| 01H | 工作模式 | 支持查询、配置 |
| 05H | MAC地址 | 支持查询 |
| 06H | 以太网配置 | 支持查询、配置 |
| 07H | SOCKET端口号 | 支持查询、配置 |
| 0DH | 串口波特率 | 支持查询、配置 |
| 12H | 读卡间隔时间 | 支持查询、配置 |
| 1EH | 心跳时间间隔+等待次数 | 支持查询、配置 |
| 16H | LBT模式开关(先听后讲) | 支持查询、配置 |
| 17H | DRM模式开关 | 支持查询、配置 |

| | | |
|---|---|---|
| 21H | 读写器产品型号 | 支持查询 |
| 23H | 读写器处理器软件版本号 | 支持查询 |
| 29H | 应用处理器固件版本 | 支持查询 |
| 50H | 读写器客户端模式开关 | 支持查询、配置 |
| 51H | 服务器地址 | 支持查询、配置 |
| 52H | 最大连接失败次数 | 支持查询、配置 |
| 65H | 天线端口功率 | 支持查询、配置 |
| 68H | 天线端口功率档位 | 支持查询 |
| 6DH | 标签读取模式 | 支持查询、配置 |
| 6EH | 标签过滤次数（按时间） | 支持查询、配置 |
| E2H | 查询I/O触发参数 | 支持查询、配置 |

### （1）读写器版本信息

读写器版本信息包含三个方面内容：读写器型号、基带处理器固件版本、应用处理器固件版本。

#### ① 读写器型号

参数：21H，此参数仅供查询不能设置。

查询数据：应转换为采用ASCII编码的字符串，基本格式为“型号数字代号+地区代码”，字符串的最后一个字母为地区代码，表示读写器所符合的地区法规代码，“A” 代表符合FCC相关规定；“C” 代表符合中国无线电委员会相关规定；“E”代表符合ETSI相关规定。如适合中国大陆地区销售使用的XC-RF807读写器型号代码为“807C”（注：本文涉及的字符串均不含结束符\0，下同）。

#### ② 基带处理器固件版本

参数：23H，此参数仅供查询不能设置。

查询数据：应转换为采用ASCII编码的字符串，基本格式为“*.**”，版本V1.00表示为“1.00”。

#### ③ 应用处理器固件版本

参数：29H，此参数仅供查询不能设置。

查询数据：应转换为采用ASCII编码的字符串，基本格式为“读写器型号SVN版本号-RELEASE”。如XC-RF807的应用处理器固件当前版本号为“XC-RF807s-vn236-RELEASE”。

## （2）通信参数配置

### ① 串口通信参数

参数：0DH，波特率查询或配置

数据：0，9600bps；1，19200bps；2，115200bps。默认波特率为115200bps。

### ② 以太网通信参数

以太网通信参数包括MAC地址、IP地址、读写器TCP连接模式配置及相关参数。

读写器提供的以太网通信（基于TCP/IP协议，仅支持IPV4）包括读写器作为服务器模式（也称被动模式）和读写器作为客户端模式（也称主动模式）两种，任一时刻，读写器只能处于这两种模式中的一种。读写器的服务器模式是指外部设备（如PC端）主动向读写器发起的TCP连接。读写器的客户端模式是指读写器主动向外部设备发起的TCP连接。读写器出厂时，默认配置为服务器模式，读写器的出厂默认IP为192.168.0.210，使用网络连接时，请确保读写器的IP和PC端的IP处于同一网段。

读写器作为客户端，工作时按照设置的参数主动连接远端服务器的一种工作模式。客户端模式和备选服务器配合使用，在主服务器故障时，可自动连接到设置的备选服务器上。

- MAC地址

参数：05H，本参数为仅供查询不能设置。

数据：长度6字节，格式为十六进制MAC地址，高位优先，如 {0x00, 0x1D, 0x78, 0x12, 0x34, 0x56}代表MAC地址为00-1D-78-12-34-46。

- IP地址

参数：06H

数据：长度12字节，格式为：4字节IP+4字节子网掩码+4字节网关，高位优先。读写器默认为IP：192.168.0.210，子网掩码： 255.255.255.0，网关：192.168.0.1。如要对读写器IP做以上配置，则配置IP指令数据为：C0 A8 00 D2 FF FF FF 00 C0 A8 00 01。

- 读写器TCP连接模式配置

参数：50H

数据：长度1字节，0，表示读写器做为TCP连接服务器；1，表示读写器做为TCP连接客服端。读写器默认连接方式为读写器做TCP连接服务器。

- 读写器TCP服务端口号

参数：07H

数据：长度2字节，高位优先，用来配置读写器做为TCP服务器时使用的服务器端口号，此参数默认为7086。此参数只在读写器配置为服务器模式下才有意义。

- 备选服务器配置

参数：51H

数据：长度18字节，高位优先，备选服务器允许设置3个，参数中的第一个备选服务器表示首选的工作服务器，后面两个为备用服务器，备选服务器参数中第一个服务器的IP和端口不允许设置为0。备选服务器参数中的第二个服务器（或第三个服务器）参数设置为0，则表示不启用该服务器作为备选服务器。此参数只在读写器配置为客户端模式下才有意义。数据格式：服务器IP1(4字节)+Port1（2字节）+服务器IP2(4字节)+Port2（2字节）+服务器IP3(4字节)+Port3（2字节）

- 连接重试次数

参数：52H

数据：长度2字节，高位优先，最大失败连接次数，表示读写器连接远端服务器时，所允许的最大连接失败的次数。失败次数从零开始，连接失败一次，失败次数加1，成功连接后，失败次数归零。当失败次数累加到最大连接失败次数时，自动连接下一个服务器。当远端服务器主动正常断开连接时，不计失败次数。每次连接失败后，间隔2秒，开始下一次连接。当最后一个服务器连接失败次数超过最大连接失败次数时，读写器重新连接首选服务器（即连接顺序为备选服务器1——备选服务器2——备选服务器3——备选服务器1…）。最大失败连接次数设置为0时，读写器将不连接后续的备选服务器。此参数默认为0，只在读写器配置为客户端模式下才有意义。

### （3）RFID参数配置

#### ① 读写器天线端口功率配置

参数：65H

数据（设置）：长度2字节，天线端口号（0～3或FFH）+功率等级。第一字节表示要配置的天线端口号，天线号范围0～3，分别表示天线1、2、3、4，当天线端口号为FFH时，则表示将所有天线端口配置成相同的功率等级参数；第二字节表示要配置功率的等级参数，取值范围0～36，分别表示对应的发射功率0～36dBm。

数据（查询）：上位机查询天线端口功率配置时，读写器会将所有天线端口功率一起返回。长度4字节，分别代表1-4天线功率等级。

**② 标签读取模式**

参数：6DH

数据：长度1字节，00H表示多标签模式，02H表示单标签模式

### （4）GPI触发参数配置

参数：E2H

数据：用于GPI事件触发机制相关配置。远望谷读写器的GPIO在基本的输入输出功能基础上实现了GPI事件触发机制，采用GPI事件触发机制可以很方便的与其他设备实现联动。GPI事件触发机制的基本原理是：将GPI输入状态与相关读写器的相关操作指令进行绑定，GPI不同的输入状态可触发响应的读写器操作。每个GPI口只能绑定一对读写器指令，每个GPI口可单独绑定。此功能一般用于GPI触发相应的RFID读卡操作，满足停止条件后，读写器自动停止读卡。

IO触发数据结构如下：

| GPI端口号 | 触发条件 | 停止条件 | 延时时间 | 触发条件绑定的指令 | 停止条件绑定的指令 |
|---|---|---|---|---|---|
| 1～4H | 0~4H | 0或1 | (2字节) | (N字节) | (N字节) |

GPI端口号： 指定需要配置的GPI端口号。

触发条件：指定触发事件的GPI输入状态。0，表示关闭当前端口的GPI触发功能；1，采用GPI上升沿触发；2，采用GPI下降沿触发；3，GPI高电平触发；4，GPI低电平触发。电平触发与边沿触发的区别：电平触发在掉电重新上电后，会根据当前IO状态是否符合触发条件来自动执行触发动作。而边沿触发必

须有电平跳变才会执行触发。

停止条件：0，采用GPI状态反转做为停止条件（与触发条件相反， 上升沿对应下降沿，高电平对应低电平）；1，采用延时做为停止条件。

延时时间：16bits无符号整数，高位优先，用于指定延时做为停止条件时的延时时间。

触发条件和停止条件绑定的指令：指定相应的读写器操作指令，完整的IRP1指令，长度不超过128字节。空指令用{0x0, 0x0}表示。注意：当绑定关功放指令时应绑定PowerOff_800类，而不是PowerOff类。

IRP1指令数据获得步骤：

- 实例化一个IRP指令类（如ReadTag、PowerOff_800等）
- 将该类的PortType成员属性设置为“”，空字符串
- 使用该类的TransmitterData获取指令数据

查询GPI触发参数时需在保留域内填充指定的GPI端口号。

## 4、查询读写器系统参数（SysQuery_500类）

用途：查询读写器系统参数。

### ① 构造函数

C#：public SysQuery_500(Byte infoType, Byte infoLength)

JAVA：public SysQuery_500(byte infoType, byte infoLength)

C++：SysQuery_500(
        unsigned char  infoType,
        unsigned char infoLength)

参数：

infoType：信息类型。
infoLength：返回参数长度，单位为字节。

### ② 指令应答（ReceivedMessage）

查询数据
用途：获取查询数据。

C#：public Byte[] QueryData { get; }

JAVA：public byte[] getQueryData()

C++：bool GetQueryData(unsigned char *data,unsigned char &dataLen)

**③ 指令说明**

信息类型：

| 代码 | 参数类型 | 信息长度 |
|---|---|---|
| 02H | 天线端口 | 1字节 |

天线端口定义如下：

| BIT7 | BIT6 | BIT5 | BIT4 | BIT3 | BIT2 | BIT1 | BIT0 |
|---|---|---|---|---|---|---|---|
| 保留 | 保留 | 保留 | 保留 | 端口4 | 端口3 | 端口2 | 端口1 |

Bit3：天线4的使能标志，1：允许；0：禁止

Bit2：天线3的使能标志，1：允许；0：禁止

Bit1：天线2的使能标志，1：允许；0：禁止

Bit0：天线1的使能标志，1：允许；0：禁止

注意：只有当扫描区域（见6.2.1）为ID_6B、ID_UserData_6B、EPC_6C_ID_6B、EPC_TID_UserData_6C_ID_UserData_6B、TID_6C_ID_6B时，读标签前需要设置，设置后信息保存到下一次更改或读写器掉电。

## 5、配置读写器系统参数（SysConfig_500类）

支用途：配置读写器系统参数。

**① 构造函数**

```
C#：public SysConfig_500(
            Byte infoType,
            Byte infoLength,
            Byte[] pData)
JAVA：public SysConfig_500(
            byte infoType,
            byte infoLength,
```

byte[] pData)

C++: SysConfig_500(
unsigned char infoType,
unsigned char infoLength,
unsigned char *pData,
unsigned char pDataLen)

参数:

infoType: 信息类型。
infoLength: 信息长度，单位为字节。
pData: 配置数据。
pDataLen: *pData参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

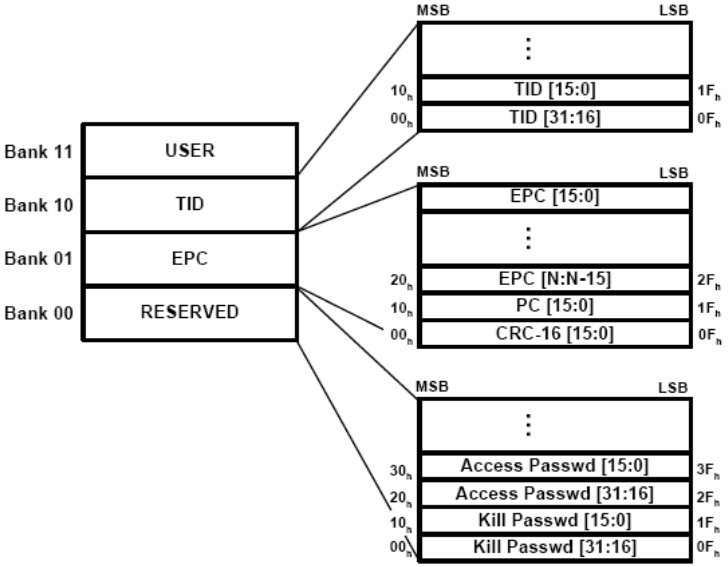
③ 指令说明

信息类型、信息长度同上节，配置数据同上节查询数据。

# 六、电子标签识别与访问

## 1、ISO18000-6C(EPC C1G2)电子标签的存储结构

ISO18000-6C(EPC C1G2)电子标签（以下简称6C标签）从逻辑上，标签的存储区被划分为四个不同的存储区，每一个存储区由一个以上的存储字（word，16bits，下同）组成，6C标签存储读写访问的最小单位为位宽为16bits的字，逻辑存储映射表如下图所示。



这些存储区具体定义为：

① RESERVED存储区

存储区代码为00’b。用于存储kill 和access 功能所需的密码。其中，存储地址00h 到1Fh 存储kill 密码，20h 到3Fh 存储access 密码。

② EPC存储区

从地址00h 到0Fh 存储CRC-16,地址10h 到1Fh 存储协议控制位 (PC)，20h 及其大于20h的地址存储用于识别标签附着对象的代码(像EPC 码，在后面的介绍中都称这个码为EPC 码)。对PC 进行子划分，在地址10h 到14h 存储EPC 码长度，15h 到17h 存储RFU，18h 到1Fh 存储编号系统标识符 (NSI)。CRC-16,PC 和EPC 以高位在先的方式进行存储(EPC 码的高位存储在20h 地址)。

③ TID存储区

在地址00h 到07h 存储一个8 位的ISO/IEC 15693 分配的类识别码(EPCglobal 的代码为11100010’b)。TID 存储库在地址07h 以上的区域应含有充足的识别信息，使读写器能唯一识别标签所支持的定制命令和(或)可选择性命令。对于ISO/IEC 15693 分配类号为11100010’b 的标签，这些识别信息将构成一个12 位字长的标签掩膜设计者标识符 (EPCglobal 组织分配的厂商编号)和一个12 位字长的标签模型码（由标签芯片厂商定义），标签掩膜设计者标识符存储在08h 到13h，模型号码存储在地址14h 到1Fh。标签可以在TID 区域地址大于1Fh 的区域存储标签和提供商的特有数据(一般为标签的序列号)。

④ USER存储区

允许用户特有数据的存储，该存储区的存储组织结构是由用户定义的。

相关的详细信息请参考资料【1】。

## 2、标签数据读取

### （1）标签扫描（ReadTag类）

用途：构造标签扫描指令。

#### ① 构造函数

```
C#:  public ReadTag(ReadMemoryBank rmb)
     public ReadTag(
            ReadMemoryBank rmb,
            Int32 readTime,
            Int32 stopTime)
     public ReadTag(
            ReadMemoryBank rmb,
            Boolean isGetOneTag)
JAVA: public ReadTag(ReadMemoryBank rmb)
     public ReadTag(
            ReadMemoryBank rmb,
            int readTime,
            Int stopTime)
     public ReadTag(
            ReadMemoryBank rmb,
            boolean isGetOneTag)
C++: ReadTag(ReadMemoryBank rmb)
     ReadTag(ReadMemoryBank rmb,
            int readTime,
            int stopTime)
     ReadTag(ReadMemoryBank rmb,
            BOOL isGetOneTag)
```

参数：

rmb：扫描区域，详见本节第3点。

readTime：扫描时间，整型，单位为毫秒。

stopTime：关功放时间，整型，单位为毫秒。

isGetOneTag：是否只取一张标签就返回，布尔型。真，表示扫描到一张标签即返回；假，表示在超时时间内，返回所有扫描到的标签数据。

说明：

- 第二个构造函数的扫描时间表示开功放时间，当扫描指定的时间（readTime）后，自动调用关功放指令，停止指定的时间（stopTime）后再扫描，依此类推进行循环操作，直到发送相应的停止操作（参见PowerOff）。这个处理的机制专门应对那些需要长期开功放的情况。
- 第三个构造函数强制返回标签信息或者等待超时，极大的简化了某些场景的操作流程。

**② 成员/属性**

成员/属性用途为更改标签扫描各项配置。

- 天线端口

  用途：设置天线端口。默认值：01H。

  C#：public Byte Antenna { set; }
  JAVA：public void setAntenna(byte antenna)
  C++：void Antenna(unsigned char ucAntenna)

  支持扫描区域为：ReadMemoryBank所有项。

  注意：天线端口使用详见4.6

- Q值

  用途：设置Q值。取值范围：0-15，默认值：00H。

  C#：public Byte Q { set; }
  JAVA：public void setQ(byte q)
  C++：void SetQ (unsigned char ucQ)

  支持扫描区域为：除ID_6B和ID_UserData_6B外ReadMemoryBank所有项。

- 读取方式

  用途：设置是否循环读取。真为循环读取，假为单次读取。默认值：true。

  C#：public Boolean IsLoop { set; }
  JAVA：public void setLoop(boolean isLoop)
  C++：void SetIsLoop (unsigned char uc IsLoop)

  支持扫描区域为：ReadMemoryBank所有项。

- TID长度

  用途：设置TID长度，单位为字。默认值：04H。

  C#：public Byte TidLen { set; }
  JAVA：public void setTidLen(byte tidLen)
  C++：public void SetTidLen (unsigned char ucTidLen)

  支持扫描区域为：
  · EPC_TID_UserData_6C_2
  · EPC_TID_UserData_Reserved_6C_ID_UserData_6B
  · EPC_TID_UserData_6C_PASSWORD

- 6C标签用户区起始地址

  用途：设置6C标签用户区起始地址，单位为字。默认值：0。

  C#：public UInt32 UserDataPtr_6C { set; }
  JAVA：public void setUserDataPtr_6C(byte userDataPtr)
  C++：void SetUserdataPtr_6C(unsigned char ucUserdataPtr_6C)

  支持扫描区域为：
  · EPC_TID_UserData_6C_2
  · EPC_TID_UserData_Reserved_6C_ID_UserData_6B
  · EPC_TID_UserData_6C_PASSWORD

- 6C标签用户区长度

  用途：设置6C标签用户区长度，单位为字。默认值：0EH。

  C#：public Byte UserDataLen_6C { set; }
  JAVA：public void setUserDataLen_6C(byte userDataLen)
  C++：void SetUserDataLen_6C (unsigned char ucUserDataLen_6C)

  支持扫描区域为：

- EPC_TID_UserData_6C_2
- EPC_TID_UserData_Reserved_6C_ID_UserData_6B
- EPC_TID_UserData_6C_PASSWORD

- 6B标签用户区起始地址

  用途：设置6B标签用户区起始地址，单位为字节。默认值：00H。

  C#：public Byte UserDataPtr_6B { set; }
  JAVA：public void setUserDataPtr_6B(byte userDataPtr)
  C++：void SetUserdataPtr_6B (unsigned char ucUserDataLen_6C)

  支持扫描区域为：
  - EPC_TID_UserData_Reserved_6C_ID_UserData_6B

- 6B标签用户区长度

  用途：设置6B标签用户区长度，单位为字节。默认值：08H。

  C#：public Byte UserDataLen_6B { set; }
  JAVA：public void setUserDataLen_6B(byte userDatLen)
  C++：void SetUserDataLen_6B (unsigned char ucUserDataLen_6B)

  支持扫描区域为：
  - EPC_TID_UserData_Resered_6C_ID_UserData_6B

- 保留区长度

  用途：设置保留区长度，单位为字。默认值：00H。

  C#：public Byte ReservedLen { set; }
  JAVA：public void setReservedLen(byte reservedLen)
  C++：void SetReservedLen (unsigned char ucReservedLen)

  支持扫描区域为：
  - EPC_TID_UserData_Reserved_6C_ID_UserData_6B

- 访问密码

  用途：设置访问密码，4字节。默认值：全0。

  C#：public Byte[] AccessPwd { set; }
  JAVA：public void setAccessPwd(byte[] accessPwd)
  C++：void SetAccessPwd (unsigned char* ucAccessPwd)

  支持扫描区域为：
  - EPC_TID_UserData_Reserved_6C_ID_UserData_6B

· EPC_TID_UserData_6C_PASSWORD

- 6C标签周期读取次数

  用途：设置一轮读取周期中6C标签读取次数。默认值：01H。

  C#：public Byte ReadTimes_6C { set; }
  JAVA：public void setReadTimes_6C(byte readTimes)
  C++：void SetReadTimes_6C(unsigned char ucReadTimes_6C)

  支持扫描区域为：

  · EPC_TID_UserData_Reserved_6C_ID_UserData_6B

- 6B标签周期读取次数

  用途：设置一轮读取周期中6B标签读取次数。默认值：01H。

  C#：public Byte ReadTimes_6B { set; }
  JAVA：public void setReadTimes_6B(byte readTimes)
  C++：public void SetReadTimes_6B(unsigned char ucReadTimes_6B)

  支持扫描区域为：

  · EPC_TID_UserData_Reserved_6C_ID_UserData_6B

- 操作类型

  用途：设置G2iM标签读取操作类型。默认值：01H。

  JAVA：public void setType(byte type)

  支持扫描区域为：

  · EPC_TID_UserData_6C_PASSWORD

**③ 枚举（ReadMemoryBank）**

- EPC_6C

用途：扫描6C标签EPC数据。此功能用于识读标签EPC码，又称盘存，EPC码在实际应用时用于标识被追踪物，其功能类似商品的条码，且EPC码是最容易获取的标签数据。

- TID_6C

用途：扫描6C标签TID数据。此功能用于识读标签TID码，目前主流厂商的标签TID码均不可改写且编码具有唯一性，因此在应用上多以TID码用作唯一标识，

不同型号的标签芯片的TID码长度可能存在不同，本功能可自行适应长度在32～96bits范围内的TID码长度。

- EPC_TID_UserData_6C_2

用途：扫描6C标签EPC、TID和用户数据。此功能可用于将标签的EPC码、TID码、用户数据区数据一次性读取，使用此功能时TID码长度不具备自适应特性。

- ID_6B

用途：扫描6B标签ID数据。

- ID_UserData_6B

用途：扫描6B标签ID和用户数据

- EPC_TID_UserData_Reserved_6C_ID_UserData_6B

用途：扫描6C标签EPC、TID、用户区、保留区数据和6B标签ID、用户数据

- EPC_TID_UserData_6C_PASSWORD

用途：带密码通用读取G2iM标签所有区域

#### ④ 指令应答（ReceivedMessage）

只有第三个构造函数才有指令应答。

- 标签数据

用途：获取标签数据信息。

```
C#: public RXD_TagData[] List_RXD_ TagData { get; }
JAVA: public RXD_TagData [] getList_RXD_TagData ()
C++: RXD_TagData * List_RXD_TagData (unsigned int &len);
```

备注： C++应答len初始值0，循环调用本函数，直到返回List_ RXD_TagData对象指针为NULL

### （2）国标标签读取（GB_ReadTag类）

用途：构造国标标签读取指令。

① 构造函数

```
C#: public GB_ReadTag (
        Byte antenna,
        Byte type,
        Byte[] accessPwd,
        GBMemoryBank readmemoryBank,
        UInt32 ptr,
        Byte length)

        public GB_ReadTag (
        Byte antenna,
        Byte type,
        Byte[] accessPwd,
        GBMemoryBank readmemoryBank,
        UInt32 ptr, Byte length, Byte[] tagID,
        GBMemoryBank tagIDType,
        UInt32 selptr,
        Byte target,
        Byte rule)
```

参数：

antenna：天线端口。

type：读取类型。

accessPwd：访问密码，4字节。

readmemoryBank：读取区域。

ptr：读取的起始地址，单位为字。

length：读取长度，单位为字。

tagID：选择标签的数据。

tagIDType：选择标签的区域。

selptr：选择标签的起始地址。

target：目标。

rule：规则。

说明：

第二个构造函数强制返回指定标签，极大的简化了某些场景的操作流程。

JAVA：public GBAccessReadTag(

GBReadMemoryBank rmb,

GBMemoryBank matchingBank)

参数：

rmb：读取区域-标签信息区、编码区、用户子。

matchingBank：用户子区编号,当rmb为Sub_UserData_GB_Access时有效。

说明：

1.用于单次或多次访问读标签信息区、编码区数据(默认长度为0x20个字)。rmb设置为TID_GB_Access， EPC_GB_Access,可以通过setXXX()方法对天线端口、操作类型、访问密码、访问区域首地址、读取长度等五个属性进行设置；

2.用于多次快速访问指定用户子区数据(读取长度默认为0x20个字)。rmb设置为Sub_UserData_GB_Access,matchingBank设置为用户子区编号(GBUser1Memor

y...GBUser15Memory),可以通过setXXX()方法对天线端口、访问密码、访问区域首地址、读取长度等四个属性进行设置,其余方法禁止设置。

public GBAccessReadTag(

byte antenna,

String password,

GBMemoryBank bank,

int headAddress,

int length)

参数:

antenna:天线端口号。

password:访问密码。

bank:访问区域,值为GBUser1Memory...GBUser16Memory。

headAddress:访问区域首地址。

length:读取长度。

说明:

用于自定义选择单次读用户子区.通过此构造函数实现访问读用户子区数据时,当读取长度大于0x20个字时必须先设置匹配数据和匹配区域参数;当读取长度小于等于0x20个字时必须先发送分类指令。使用此构造函数读取用户子区数据,强烈建议不使用setXXX()更改参数。

② 指令应答(ReceivedMessage)

- 天线端口

用途:获取天线端口。

C#: public Byte Antenna{get;}

JAVA：public byte getAntenna(){}

- 标签编码区

用途：获取标签编码区。

C#：public Byte[] Data{ get; }

JAVA：public byte[] getTagData(){}

### （3）盘存国际标签编码区（GB_InventoryCodeData类）

用途：构造盘存国际标签编码区指令。

① 构造函数

C#：public GB_InventoryCodeData(

Byte antenna,

Byte target,

Byte session,

Byte conditions)

参数：

antenna：设置天线端口。

target：目标。

session：会话。

conditions：条件。

JAVA：public GBInventoryTag(

byte antenna,

int target,

int session,

int condition)

参数：

antenna：天线端口号。

target：目标,0~2。

session：会话,0~3。

condition：条件,0~3。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){}

- 标签编码区

用途：获取标签编码区。

C#：public Byte[] CodeData{ get; }

JAVA：public byte[] getTagData(){}

### （4）接收标签数据信息（RXD_TagData类）

用途：接收标签数据信息。继承自BaseMessageNotification抽象类。

**① 指令应答（ReceivedMessage）**

- 读写器名称

用途：获取读写器名称。

C#：public String ReaderName { get; }

JAVA：public string getReaderName (){ }

C++：CString GetReaderName ()

- 标签类型

用途：获取标签类型，值可为“6C”、“6B”。

C#：public String TagType { get; }

JAVA：public string getTagType (){ }

C++：CString GetTagType ()

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){ }

C++：unsigned char GetAntenna()

- 标签EPC数据

用途：获取标签EPC数据。无数据则为空。

C#：public Byte[] EPC { get; }

JAVA：public byte[] getEPC(){ }

C++：bool GetEPC(unsigned char *Epc,unsigned char &epcLen)

- 标签TID/ID数据

用途：获取6C标签TID数据或者6B标签ID数据。无数据则为空。

C#：public Byte[] TID { get; }

JAVA：public byte[] getTID(){ }

C++：bool GetTID(unsigned char *tid,unsigned char &tidLen)

- 标签用户区数据

用途：获取6C或者6B标签用户区数据。无数据则为空。

C#：public Byte[] UserData { get; }

JAVA：public byte[] getUserData(){ }

C++：bool GetUserData(unsigned char *userdata,unsigned char &user-dataLen)

- 标签保留区数据

用途：获取标签保留区数据。

C#：public Byte[] Reserved { get; }

JAVA：public byte[] getReserved { }

C++：bool GetReserved(unsigned char *pReserved,unsigned char &ReservedLen)

说明：保留区包括访问密码和销毁密码。详见6.1。

- 标签回波速率

用途：获取标签回波速率。无数据则为空。

C#：public Byte[] RSSI { get; }

JAVA：public byte[] getRSSI(){ }

C++：bool GetRSSI(unsigned char *Epc,unsigned char &epcLen)

- 标签读取时间

用途：获取标签读取时间。无数据则为空。

C#：public Byte[] RXDTime{ get; }

JAVA：public byte[] getRXDTime (){ }

C++：bool GetRXDTime (unsigned char *time,unsigned char &timeLen)

备注：读取时间如果是8字节则表示UTC时间，前4字节为秒，后4个字节为微秒。如果读取时间为6字节，则表示BCD码时间，6个字节依次为年、月、日、时、分、秒。

### （5）6C标签读用户数据区（ReadUserData_6C类）

用途：读标签用户数据区。此功能用于获取标签USER区数据，通常与标签选择读取功能组合来读取指定标签的用户数据区内容，此功能仅用于单标签单次操作。

#### ① 构造函数

C#： public ReadUserData_6C(

```
            Byte antenna,
            UInt32 ptr,
            Byte length)
      public ReadUserData_6C(
            Byte antenna,
            UInt32 ptr,
            Byte length,
            Byte[] tagID,
            MemoryBank tagIDType)
JAVA: public ReadUserData_6C(
            byte antenna,
            int ptr,
            byte length)
      public ReadUserData_6C(
            byte antenna,
            int ptr,
            byte length,
            byte[] tagID,
            MemoryBank tagIDType)
C++: ReadUserData_6C(
            unsigned char  antenna,
            unsigned char ptr,
            unsigned char  length)
      ReadUserData_6C(
            unsigned char antenna,
            unsigned char ptr,
            unsigned char length,
            unsigned char *tagID,
            unsigned char tagIDLen,
            unsigned char  tagIDType)
```

参数：antenna：天线端口。

ptr：起始地址，单位为字。
length：读取长度，单位为字。
tagID：指定要操作标签的EPC或TID。
tagIDType：tagID对应的数据区枚举。
tagIDLen：*tagID参数长度。

② 指令应答（ReceivedMessage）

天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }
JAVA：public byte getAntenna(){ }
C++：unsigned char GetAntenna()

③ 标签用户区数据

用途：获取标签用户区数据。

C#：public Byte[] UserData { get; }

JAVA：public byte[] getUserData() { }

C++：bool GetUserData(unsigned char *usderData,unsigned char &userDataLen)

### （6）6B标签读用户数据区（ReadUserData2_6B类）

用途：读标签用户区数据。

① 构造函数

C#：public ReadUserData2_6B(

Byte antenna,

Byte[] tagID,

Byte ptr,

Byte length)

JAVA：public ReadUserData2_6B(

byte antenna,

byte[] tagID,

byte ptr,

byte length)

C++：ReadUserData2_6B(

unsigned char  antenna,

unsigned char *tagID,

unsigned char tagIDLen,

unsigned char ptr,

unsigned char length)

参数：

antenna：天线端口。

tagID：指定要操作标签的ID。

ptr：起始地址（0~215），单位为字节。

length：读取长度，单位为字节。

tagIDLen：* tagID参数长度。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){ }

C++：unsigned char GetAntenna()

- 标签用户区数据

用途：获取标签用户区数据。

C#：public Byte[] UserData { get; }

JAVA：public byte[] getUserData() { }

C++：bool GetUserData(unsigned char *userData,unsigned char &userDataLen)

### （7）6C标签读保留区（ReadReservation_6C类）

用途：读标签读保留区。此功能用于获取标签保留区数据，通常与标签选择读取功能组合来读取指定标签的保留区内容。

① 构造函数

C#：public ReadReservation_6C (

Byte antenna,

Byte[] AccessPwd,

UInt32 ptr,

Byte length)

public ReadUserData_6C(

Byte antenna,

Byte[] AccessPwd,

UInt32 ptr,

Byte length,

Byte[] tagID,

MemoryBank tagIDType)

参数：

antenna：天线端口。

accessPwd：访问密码。

ptr：起始地址，单位为字。

length：读取长度，单位为字。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

tagIDLen：*tagID参数长度。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

- 标签保留区数据

用途：获取标签保留区数据。

C#：public Byte[] ReservationData { get; }

### （8）UcodeDna鉴别读（UcodeDna_ReadUserData类）

用途：读Ucode标签读用户区。此功能用于获取Ucode标签用户区数据。

① 构造函数

```
C#：public UcodeDna_ReadUserData (
        Byte antenna,
        Byte[] tem1,
        Byte[] tem2,
        UInt32 readTime
        UInt32 ptr,
        UInt32 length)
```

参数：

antenna：天线端口。

teml：读写器鉴别标签采用的TAM1方式，总共17字节，第0字节为key值，0x00为key0，0x01为key1值；第1~16字节为key值的具体内容。

teml：读写器鉴别标签采用的TAM2方式，总共17字节，第0字节为key值，0x00为key0，0x01为key1值；第1~16字节为key值的具体内容。

readTime：读取时间。如果为0x0000，表示永久操作，手动关功放停止。

ptr：起始地址，单位为字。

length：读取长度，单位为字。

② 指令应答（ReceivedMessage）

天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

标签用户区数据

用途：获取标签用户区数据。

C#：public Byte[] UserData { get; }

### （9）G2iM标签带密码通用读（ReadTagByAccessPwd_6C类）

① 构造函数

```
C#：public ReadTagByAccessPwd_6C(
    Byte Antenna,
    Byte Qvalue,
    Byte[] AccessPwd,
    Byte tidLen,
```

uint UserDataPtr,

Byte UserDataLen)

参数：

Antenna：天线端口。

Qvalue：Q值，单位为字。

AccessPwd：访问密码。

tidLen：读取TID的长度。

UserDataPtr：读取用户区数据的起始地址。

UserDataLen：读取用户区数据长度。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

- 标签用户区数据

用途：获取标签用户区数据。

C#：public Byte[] UserData { get; }

- 标签TID

用途：获取标签TID。

C#：public Byte[] TID{ get; }

- 标签EPC

用途：获取标签EPC。

C#：public Byte[] EPC{ get; }

### （10）读取国标标签信息区与编码区（GB_ReadInformationAndCode类）

用途：构造读取国际标签信息区与编码区指令。

① 构造函数

```
C#：public GB_ReadInformationAndCode(
        Byte antenna,
        Byte type,
        Byte[] accessPwd,
        Byte length)

        public GB_ReadInformationAndCode(
        Byte antenna, Byte type,
        Byte[] accessPwd,
        Byte length,
        Byte[] tagID,
        GBMemoryBank tagIDType,
        UInt32 selptr,
        Byte target,
        Byte rule)

        public GB_ReadInformationAndCode(
        Byte antenna,
        Byte[] accessPwd,
        Byte length,
        Byte[] tagID,
```

GBMemoryBank tagIDType,

UInt32 selptr)

参数：

antenna：设置天线端口。

Type：读取类型（0x00扫读，0x01单次读）。

accessPwd：访问密码。

length：读取信息区的长度。

tagID：指定要读取的标签数据。

tagIDType：指定要读取的标签数据的对应区域。

Selptr：指定要读取的标签数据的起始位。

target：目标。

rule：规则。

JAVA：public GBCombinationReadTag(

byte antenna,

int operationType,

String tidPassword,

int tidLength)

参数：

antenna：天线端口号。

operationType：操作类型。

tidPassword：tid区域访问密码。

tidLength：读取Tid长度。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){}

- 标签编码区

用途：获取标签编码区数据。

C#：public Byte[] CodeData{ get; }

JAVA：public byte[] getGBEpcData(){}

- 标签信息区

用途：获取标签信息区数据。

C#：public Byte[] InformationData{ get; }

JAVA：public byte[] getGBTidData(){}

### （11）读取国标标签所有区域（GB_ReadAllBank类）

用途：读取国标标签所有区域数据。

① 构造函数

```
C#：public GB_ReadAllBank(
        Byte antenna,
        Byte tidlength,
        Byte[] tidPwd,
        Byte epclength,
        GBMemoryBank userdatabank,
        Byte userdataptr,
        Byte userdatalen,
```

Byte[] userdataPwd)

参数：

antenna：天线端口。

Tidlength：读取信息区的长度。

tidPwd：信息区访问密码。

epclength：读取编码区的长度。

userdatabank：用户子区。

userdataptr：用户子区起始地址。

userdatalen：用户子区读取长度。

JAVA: public GBReadAllBank(

byte antenna,

byte tidlength,

byte[] tidPwd,

byte epclength,

GBMemoryBank userdatabank,

byte userdataptr,

byte userdatalen,

byte[] userdataPwd)

参数说明：

antenna：天线端口。

tidlength：读取信息区的长度。

tidPwd：信息区访问密码。

epclength：读取编码区的长度。

userdatabank：用户子区。

userdataptr：用户子区起始地址。

userdatalen：用户子区读取长度。

② 指令应答（ReceivedMessage）

- 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){}

- 标签编码区数据

用途：获取标签编码区数据。

C#：public Byte[] CodeData { get; }

JAVA：public byte[] getGBEpcData(){}

- 标签信息区数据

用途：获取标签信息区数据。

C#：public Byte[] InformationData { get; }

JAVA：public byte[] getGBTidData(){}

- 标签用户子区数据

用途：获取标签用户子区数据。

C#：public Byte[] UserData { get; }

JAVA：public byte[] getGBUserdata(){}

## 3、标签数据写入

标签写入功能包括：写EPC码、写USER区数据、写标签访问密码、写标签灭活密码、通用写功能，标签写入功能一般与标签选择功能。所有的标签写操作仅用于单标签单次操作。

### （1）6C标签写EPC（WriteEpc类）

用途：写标签EPC数据区。此功能用于修改标签的EPC码数据，由于标签的EPC码的长度由EPC数据区的PC（Protocol Control）为来控制，所以使用此功能时读写器将会根据要写入的EPC码内容长度自动计算PC值并进行改写。

#### ① 构造函数

```
C#: public WriteEpc(
            Byte antenna,
            Byte[] accessPwd,
            Byte[] epcData)
    public WriteEpc(
            Byte antenna,
            Byte[] accessPwd,
            Byte[] epcData,
            Byte[] tagID,
            MemoryBank tagIDType)
JAVA: public WriteEpc(
            byte antenna,
            byte[] accessPwd,
            byte[] epcData)
      public WriteEpc(
            byte antenna,
            byte[] accessPwd,
            byte[] epcData,
            byte[] tagID,
            MemoryBank tagIDType)
C++: WriteEpc(
          unsigned char antenna,
```

```
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char *epcData,
            unsigned char epcDataLen)
      WriteEpc(
            unsigned char  antenna,
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char *epcData,
            unsigned char epcDataLen,
            unsigned char *tagID,
            unsigned char tagIDLen,
            unsigned char tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

epcData：指定要写入的EPC数据。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

accessPwdLen：*accessPwd参数长度。

epcDataLen：*epcData参数长度。

tagIDLen：*tagID参数长度。

#### ② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

#### ③ 指令说明

标签访问密码：32bits无符号整数，高位优先，用于写操作前的密码校验。

### （2）6C标签写用户数据区（WriteUserData_6C类）

用途：写标签用户数据区。

#### ① 构造函数

```
C#: public WriteUserData_6C(
            Byte antenna,
```

```
            Byte[] accessPwd,
            UInt32 ptr,
            Byte[] userData)
      public WriteUserData_6C(
            Byte antenna,
            Byte[] accessPwd,
            UInt32 ptr,
            Byte[] userData,
            Byte[] tagID,
            MemoryBank tagIDType)

JAVA: public WriteUserData_6C(
              byte antenna,
              byte[] accessPwd,
              int ptr,
              byte[] userData)
       public WriteUserData_6C(
              byte antenna,
              byte[] accessPwd,
              int ptr,
              byte[] userData,
              byte[] tagID,
              MemoryBank tagIDType)

C++: WriteUserData_6C(unsigned char antenna,
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char ptr,
            unsigned char *userData,
            unsigned char userDataLen)
      WriteUserData_6C(unsigned char antenna,
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char ptr,
            unsigned char *userData,
```

```
            unsigned char userDataLen,
            unsigned char* tagID,
            unsigned char tagIDLen,
            unsigned char tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

ptr：起始地址，单位为字。

userData：指定要写入的用户数据。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

accessPwdLen：* accessPwd参数长度。

userDataLen：*userData参数长度。

tagIDLen：* tagID参数长度。

#### ② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### （3）6B标签写用户数据区（WriteUserData2_6B类）

用途：写标签用户区数据。

#### ① 构造函数

```
C#: public WriteUserData2_6B(
            Byte antenna,
            Byte[] tagID,
            Byte ptr,
            Byte[] userdata)
JAVA: public WriteUserData2_6B(
            byte antenna,
            byte[] tagID,
            byte ptr,
            byte[] userdata)
```

```
C++: WriteUserData2_6B(
          unsigned char antenna,
          unsigned char *tagID,
          unsigned char tagIDLen,
          unsigned char ptr,
          unsigned char *userData,
          unsigned char userDataLen)
```

参数：

antenna：天线端口。

tagID：指定要操作标签的ID。

ptr：起始地址（0~215），单位为字节。

userdata：要写入的数据。

tagIDLen：* tagID参数长度。

userDataLen：*userData参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### （4）6C标签配置访问密码（AccessPwdConfig_6C类）

用途：本功能用于改写标签的访问密码，6C标签默认访问密码为0。

① 构造函数

```
C#: public AccessPwdConfig_6C(
          Byte antenna,
          Byte[] oldPwd,
          Byte[] newPwd)
    public AccessPwdConfig_6C(
          Byte antenna,
          Byte[] oldPwd,
          Byte[] newPwd,
          Byte[] tagID,
```

```
            MemoryBank tagIDType)
JAVA: public AccessPwdConfig_6C(
            byte antenna,
            byte[] oldPwd,
            byte[] newPwd)
       public AccessPwdConfig_6C(
            byte antenna,
            byte[] oldPwd,
            byte[] newPwd,
            byte[] tagID,
            MemoryBank tagIDType)
C++: AccessPwdConfig_6C(
          unsigned char antenna,
          unsigned char  *oldPwd,
          unsigned char oldPwdLen,
          unsigned char *newPwd,
          unsigned char newPwdLen)
     AccessPwdConfig_6C(
          unsigned char antenna,
          unsigned char  *oldPwd,
          unsigned char oldPwdLen,
          unsigned char *newPwd,
          unsigned char newPwdLen,
          unsigned char* tagID,
          unsigned char tagIDLen,
          unsigned char tagIDType)
```

参数：

antenna：天线端口。
oldPwd：原访问密码，4字节。
newPwd：新访问密码，4字节。
tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。
oldPwdLen：*oldPwd参数长度。
newPwdLen：*newPwd参数长度。
tagIDLen：* tagID参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

标签出厂默认的访问密码为8个零。

如果密码所在区未锁定，全零的密码也可执行写操作。

### （5）6C标签配置销毁密码（KillPwdConfig_6C类）

用途：本功能用于改写标签的灭活密码，6C标签默认密码为0，要对标签进行必须是在标签灭活密码为非0的情况下。

① 构造函数

```
C#：public KillPwdConfig_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte[] killPwd)
    public KillPwdConfig_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte[] killPwd,
            Byte[] tagID,
            MemoryBank tagIDType)

JAVA：public KillPwdConfig_6C(
            byte antenna,
            byte[] accessPwd,
            byte[] killPwd)
      public KillPwdConfig_6C(
```

```
            byte antenna,
            byte[] accessPwd,
            byte[] killPwd,
            byte[] tagID,
            MemoryBank tagIDType)
C++: KillPwdConfig_6C(
          unsigned char antenna,
          unsigned char  *accessPwd,
          unsigned char accessPwdLen,
          unsigned char *killPwd
          unsigned char killPwdLen)
      KillPwdConfig_6C(
          unsigned char antenna,
          unsigned char  *accessPwd,
          unsigned char accessPwdLen,
          unsigned char *killPwd,
          unsigned char killPwdLen,
          unsigned char* tagID,
          unsigned char tagIDLen,
          unsigned char tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

killPwd：销毁密码，4字节。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

accessPwdLen：*accessPwd参数长度。

killPwdLen：*killPwd参数长度。

tagIDLen：* tagID参数长度。

**② 指令应答（ReceivedMessage）**

该消息没有可进一步解析的数据，ReceivedMessage为空。

### ③ 指令说明

EPC协议规定标签在执行销毁操作时其销毁密码必须为非零。

## （6）6C标签通用写操作（WriteTag_6C类）

用途：可写标签保留区、EPC、TID或用户数据区。注意：保留区、EPC和TID不建议用该指令写入，除非您对6C标签非常熟悉。

### ① 构造函数

```
C#: public WriteTag_6C(
            Byte antenna,
            Byte[] accessPwd,
            MemoryBank memoryBank,
            UInt32 ptr,
            Byte length,
            Byte[] data)
    public WriteTag_6C(
            Byte antenna,
            Byte[] accessPwd,
            MemoryBank memoryBank,
            UInt32 ptr,
            Byte length,
            Byte[] data
            Byte[] tagID,
            MemoryBank tagIDType)

JAVA: public WriteTag_6C(
            byte antenna,
            byte[] accessPwd,
            MemoryBank memoryBank,
            int ptr,
            byte length,
            byte[] data)
      public WriteTag_6C(
```

```
        byte antenna,
        byte[] accessPwd,
        MemoryBank memoryBank,
        int ptr,
        byte length,
        byte[] data
        byte[] tagID,
        MemoryBank tagIDType)
C++: WriteTag_6C(unsigned char antenna,
        unsigned char *accessPwd,
        unsigned char accessPwdLen,
        MemoryBank memoryBank,
        unsigned char ptr,
        unsigned char length,
        unsigned char *data,
        unsigned char dataLen)
    WriteTag_6C(
        unsigned char antenna,
        unsigned char *accessPwd,
        unsigned char accessPwdLen,
        MemoryBank memoryBank,
        unsigned char ptr,
        unsigned char length,
        unsigned char *data,
        unsigned char dataLen,
        unsigned char *tagID,
        unsigned char tagIDLen,
        MemoryBank tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

memoryBank：指定要写入的数据区，对应MemoryBank枚举。
ptr：写入起始地址，单位为字。
length：写入数据长度，单位为字。
data：指定要写入的数据。
tagID：指定要操作标签的EPC或TID。
tagIDType：tagID对应的数据区枚举。
accessPwdLen：*accessPwd参数长度。
dataLen：*data参数长度。
tagIDLen：*tagID参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### （7）国标标签通用写操作（GB_WriteData类）

用途：国标标签通用写。

① 构造函数

```
C#: public GB_WriteData(
    Byte antenna,
    Byte[] accessPwd,
    GBMemoryBank writememoryBank,
    UInt32 ptr,
    Byte[] data)
    public GB_WriteData(
    Byte antenna,
    Byte[] accessPwd,
    GBMemoryBank writememoryBank,
    UInt32 ptr,
```

Byte[] data,

Byte[] tagID,

GBMemoryBank tagIDType,

uint selptr,

Byte target,

Byte rule)

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

writememoryBank：指定要写入的数据区，对应GBMemoryBank枚举。

ptr：写入起始地址，单位为字。

data：指定要写入的数据。

tagID：指定选择要操作的标签。

tagIDType：tagID对应的数据区枚举。

selptr：选择标签的起始地址，单位为字。

dataLen：*data参数长度。

target：目标。

rule：规则。

JAVA：public GBWriteTag(

int antenna,

String password,

GBMemoryBank bank,

int headAddress,

byte[] data)

参数：

antenna：天线号。

password：访问密码。

bank：写入区域。

headAddress：写入首地址,单位为字。

data：写入数据。

说明：

指令用于写指定标签指定区域指定首地址指定长度的数据,其中指定长度为数据长度,单位为字。data.length>0x20时需要分段写入,必须先设置匹配数据和匹配区域参数,需要指定写入的数据data。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### （8）国标标签动态写（GB_DynamicWrite类）

用途：连续写入数据，至发关功放指令停止写入。

① 构造函数

C#：public GB_DynamicWrite (

Byte antenna,

Byte[] accessPwd,

GBMemoryBank writememoryBank

UInt32 ptr

Byte[] Data)

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

writememoryBank：写入区域。

UInt32 ptr：起始地址，单位为字。

Data：指定要写入的数据。

JAVA：public GBDynamicWriteTag(

int antenna,

String password,

GBMemoryBank bank,

int headAddress,

byte[] data)

参数：

antenna：天线号。

password：访问密码。

bank：写入区域。

headAddress：写入首地址,单位为字。

data：写入数据。

说明：

该指令用于动态写指定标签指定区域指定首地址指定长度的数据，其中指定长度为数据长度，单位为字。data.length>0x20时需要分段写入，必须先设置匹配数据和匹配区域参数,需要指定写入的数据data。

② 指令应答（ReceivedMessage）

天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

## 4、标签锁操作

### （1）6C标签锁操作（LockMemoryBank_6C类）

用途：功能可以实现对标签数据区的锁定与解锁功能。锁定操作标签数据区不同的锁状态对应着不同的标签数据访问权限限制，标签各个数据区的锁状态可单独控制。6C的标签数据区存在有四种锁状态：解锁、锁定、永久解锁、永久锁定。

#### ① 构造函数

```
C#：public LockMemoryBank_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte lockType,
            Byte bank)
    public LockMemoryBank_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte lockType,
            Byte bank,
            Byte[] tagID,
            MemoryBank tagIDType)

JAVA：public LockMemoryBank_6C(
              byte antenna,
              byte[] accessPwd,
              byte lockType,
              byte bank)
      public LockMemoryBank_6C(
             byte antenna,
             byte[] accessPwd,
```

```
            byte lockType,
            byte bank,
            byte[] tagID,
            MemoryBank tagIDType)
C++: LockMemoryBank_6C(
        unsigned char antenna,
        unsigned char *accessPwd,
        unsigned char accessPwdLen,
        unsigned char lockType,
        unsigned char  memoryBank)
    LockMemoryBank_6C(
        unsigned char antenna,
        unsigned char *accessPwd,
        unsigned char accessPwdLen,
        unsigned char lockType,
        unsigned char  memoryBank,
        unsigned char* tagID,
        unsigned char tagIDLen,
        unsigned char tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

lockType：锁操作类型。00，锁定；01，解锁；02，永久锁定；03，永久解锁。

bank：锁操作区域。00，所有数据区；01，TID数据区（一般情况下，标签芯片厂商会芯片出厂时将TID区永久锁定，不能解锁）；02，EPC数据区；03，USER区；04，访问密码；05，灭活密码。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

accessPwdLen：*accessPwd参数长度。

tagIDLen：* tagID参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

EPC标签的TID区在出厂时已被写入TID码并且设置不可逆的永久锁定，因此执行“所有Bank”的解锁操作时不会执行TID区的解锁。

解锁状态：读取、写入前无需校验密码。

锁定状态：可读（访问密码和灭活密码锁定后需先校验密码才能读取），若标签访问密码非0，写入前需要校验访问密码。

永久解锁：始终可读、可写，且永久解锁后不能改变锁状态。

永久锁定：始终只读（访问密码和灭活密码永久锁定后不可读），且永久锁定后不能改变锁状态。

### （2）国标标签锁操作（GB_LockTag类）

用途：指令可以实现对标签数据区的锁定与解锁功能。执行锁定操作时标签数据区不同的锁状态对应着不同的标签数据访问权限限制，标签各个数据区的锁状态可单独控制。

① 构造函数

```
C#：public GB_LockTag (
        Byte antenna,
        Byte[] accessPwd,
        GBMemoryBank lockmemoryBank,
        Byte configure,
        Byte action)
        public GB_LockTag (
        Byte antenna,
        Byte[] accessPwd,
        GBMemoryBank lockmemoryBank,
        Byte configure,
```

Byte action,
Byte[] tagID,
GBMemoryBank tagIDType,
uint selptr,
Byte target,
Byte rule)

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

lockmemoryBank：锁操作区域。

configure：配置。

action：动作。

tagID：指定选择要操作的标签。

tagIDType：tagID对应的数据区枚举。

selptr：选择标签的起始地址，单位为字。

dataLen：*data参数长度。

target：目标。

rule：规则。

JAVA：public GBConfigTagLockOrSafeMode(

int antenna,

String password,

GBMemoryBank bank,

LockAction action)

参数：

antenna：天线号。

password：标签锁密码。

bank：存储区域。

action：锁动作。

说明：

该构造函数用于配置标签存储区域属性-锁指令。

注意：

1.bank为GBMemoryBank.GBTidMemory时,action仅设置为LockAction.Read_Only_GB,LockAction.No_Read_Write_GB时指令才可能正常设置；

2.bank为GBMemoryBank.GBEPCMemory时,action仅设置为LockAction.Read_Write_GB,LockAction.Read_Only_GB时指令才可能正常设置；

3.bank为GBMemoryBank.GBReservedMemory时,action仅设置为LockAction.Write_Only_GB,LockAction.No_Read_Write_GB时指令才可能正常设置；

4.bank为GBMemoryBank.GBUser1Memory...GBUser16Memory时,action无限制。

public GBConfigTagLockOrSafeMode(

int antenna,

String password,

GBMemoryBank bank,

int mode)

参数：

antenna：天线号。

password：标签锁密码。

bank：存储区域。

mode：安全模式,值为0~3。

说明：

该构造函数用于配置标签存储区域安全模式。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### （3）6B标签锁用户数据（LockUserData_6B类）

用途：标签用户区数据锁定。

#### ① 构造函数

```
C#：public LockUserData_6B(
          Byte antenna,
          Byte[] tagID,
          Byte[] lockInfo)
JAVA：public LockUserData_6B(
          byte antenna,
          byte[] tagID,
          byte[] lockInfo)
C++：LockUserData_6B(
          unsigned char antenna,
          unsigned char *tagID,
          unsigned char tagIDLen,
          unsigned char *lockInfo,
          unsigned char lockInfoLen)
```

参数：

antenna：天线端口。

tagID：指定要操作标签的ID。

lockInfo：锁定信息，长度与标签用户数据区长度一致，每个字节信息表示是否对该字节进行锁定操作，非0为锁定该位置的用户数据。

tagIDLen：* tagID参数长度。

lockInfoLen：*lockInfo参数长度。

#### ② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

#### ③ 指令说明

注意：锁定后不能解锁。

### （4）6B标签锁状态查询（LockStateQuery_6B类）

用途：标签用户区数据锁状态查询。

#### ① 构造函数

```
C#: public LockStateQuery_6B(
          Byte antenna,
          Byte[] tagID,
          Byte[] queryInfo)
JAVA: public LockStateQuery_6B(
          byte antenna,
          byte[] tagID,
          byte[] queryInfo)
C++: LockStateQuery_6B(
          unsigned char antenna,
          unsigned char *tagID,
          unsigned char tagIDLen,
          unsigned char *queryInfo,
          unsigned char queryInfoLen)
```

参数：

antenna：天线端口。

tagID：指定要操作标签的ID。

queryInfo：查询信息，长度与标签用户数据区长度一致，每个字节信息表示对该字节是否执行查询操作，非0为执行操作。

tagIDLen：* tagID参数长度。

queryInfoLen：*queryInfo参数长度。

#### ② 指令应答（ReceivedMessage）

锁状态

用途：获取标签数据区锁状态。对应字节00H表示数据未锁定，01H表示数据已锁定。

C#：public Byte[] LockResult { get; }

JAVA：public byte[] getLockResult() { }

C++：bool GetLockResult(unsigned char *LockResult,unsigned char &LockResultLen)

③ 指令说明

queryInfo和LockResult相对应，并且对应于用户数据区。例如，要查询用户区第10字节的锁状态，则queryInfo第10字节应设置1，其余设成0。而应答LockResult第10字节表示查询结果。

## 5、标签销毁操作

### （1）6C标签销毁（KillTag_6C类）

用途：此功能可用于销毁（kill）标签，使标签不再响应任何读写器指令从而失去使用价值，此过程不可逆。读写器只能对标签销毁密码为非0的标签进行灭销毁操作。

① 构造函数

```
C#：public KillTag_6C(
          Byte antenna,
          Byte[] killPwd,
          Byte[] epcData)
JAVA：public KillTag_6C(
          byte antenna,
          byte[] killPwd,
          byte[] epcData)
C++：KillTag_6C(
          unsigned char antenna,
          unsigned char *killPwd,
          unsigned char killPwdLen,
          unsigned char *epcData,
          unsigned char epcDataLen)
```

参数：

antenna：天线端口。
killPwd：销毁密码，4字节。用于标签销毁操作的非0密码。
epcData：指定要操作标签的EPC码。
killPwdLen：*killPwd参数长度。
epcDataLen：*epcData参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

因标签销毁指令为不可逆转操作，为防止对其他标签做误操作，该指令应附加一个EPC编码域，读写器在EPC码与读取标签的EPC码匹配时才执行销毁操作。

### （2）国标标签销毁（GB_KillTag类）

用途：此功能可用于销毁（kill）标签，使标签不再响应任何读写器指令从而失去使用价值，此过程不可逆。

① 构造函数

```
C#：public GB_KillTag (
        Byte antenna,
        Byte[] killPwd,
        Byte[] tagID,
        GBMemoryBank tagIDType,
        uint selptr,
        Byte target,
        Byte rule)
```

参数：

antenna：天线端口。

killPwd：销毁密码，4字节。用于标签销毁操作的非0密码。

tagID：指定选择要操作的标签。

tagIDType：tagID对应的数据区枚举。

selptr：选择标签的起始地址，单位为字。

target：目标。

rule：规则。

JAVA：public GBInactivateTag(

int antenna,

String password)

参数：

antenna：天线号。

password：灭活密码。

说明：

该指令必须配合国标分类指令使用。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 6、国标标签擦除（GB_Erase类）

用途：此功能可用于擦除（Erase）标签。

① 构造函数

C#：public GB_Erase （

Byte antenna,

Byte[] accessPwd,

GBMemoryBank erasememoryBank,

UInt32 ptr,

Byte eraseSize,

Byte[] tagID,

GBMemoryBank tagIDType,

uint selptr,

Byte target,

Byte rule)

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

erasememoryBank：指定要擦除的数据区。

ptr：擦除起始地址，单位为字。

eraseSize：擦除长度，单位为字。

tagID：指定选择要操作的标签。

tagIDType：tagID对应的数据区枚举。

selptr：选择标签的起始地址，单位为字。

target：目标。

rule：规则。

JAVA：public GBEraseTag(

int antenna,

String password,

GBMemoryBank bank,

int headAddress,

int length)

参数：

antenna：天线号。

password：访问密码。

bank：擦除区域。

headAddress：擦除首地址,单位为字。

length：擦除长度。

说明：

该指令必须配合国标分类指令使用。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 7、特定标签的私有功能

由于ISO18000-6C协议允许标签芯片厂商增加特色定制功能，远望谷读写器对实际应用较多的标签私有功能进行了支持。主要支持了NXP标签的EAS功能和Impinj MONZA标签的QT功能。

### （1）NXP标签EAS功能

NXP标签芯片内置有EAS存储标志位，当EAS标志位置位时，标签若收到读写器发射的EAS告警命令就会向读写器返回一串告警信息表示当前EAS标志已置位，否则标签不会响应该EAS告警命令，此功能一般用于门禁系统。NXP EAS功能存在如下已知的局限性：1.所有标签返回的告警信息内容的，所以告警信息不具备标识功能；2.由于此机制缺乏防冲撞设计，多张标签可能会同时向读写器返回告警信息时，此时读写器将可能信号冲撞而无法准确分辨告警信息从而产生漏报。更多的详细参考信息可参考资料【3】。

使用NXP EAS功能需要用到EAS位配置和EAS命令广播两条指令。

- 6C标签EAS标志配置（EasConfig_6C类）

用途：EAS标志设置或取消。

① 构造函数

```
C#:   public EasConfig_6C(
              Byte antenna,
              Byte[] accessPwd,
              Byte flag)
      public EasConfig_6C(
              Byte antenna,
              Byte[] accessPwd,
              Byte flag,
              Byte[] tagID,
              MemoryBank tagIDType)
JAVA: public EasConfig_6C(
              byte antenna,
              byte[] accessPwd,
              byte flag)
      public EasConfig_6C(
              byte antenna,
              byte[] accessPwd,
              byte flag,
              byte[] tagID,
              MemoryBank tagIDType)
C++: EasConfig_6C(
         unsigned char antenna,
         unsigned char *accessPwd,
         unsigned char accessPwdLen,
         unsigned char flag)
     EasConfig_6C(
         unsigned char antenna,
         unsigned char *accessPwd,
         unsigned char accessPwdLen,
```

```
        unsigned char flag,
        unsigned char *tagID,
        unsigned char tagIDLen,
        MemoryBank tagIDType)
```

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

flag：EAS标志配置。00H，将EAS标志复位，标签将不响应EAS广播命令；01H，将EAS标志置位，标签将响应EAS广播命令。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

accessPwdLen：*accessPwd参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

● 6C标签EAS命令广播（EasAlarm_6C类）

用途：EAS监控功能设置（只应用于采用NXP芯片的标签）。

① 构造函数

C#：public EasAlarm_6C(Byte antenna, Byte easCfg)

JAVA：public EasAlarm_6C(byte antenna, byte easCfg)

C++：public  EasAlarm_6C(unsigned char antenna, unsigned char easCfg)

参数：

antenna：天线端口。

easCfg：EAS监控配置。01H为启动EAS监控。其他值无意义。上位机发生停止指令时读写器才会中止EAS命令广播。

② 指令应答（ReceivedMessage）

◇ 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }
JAVA：public byte getAntenna(){ }
C++：unsigned char GetAntenna()

◇ EAS应答类型

用途：获取EAS应答类型。

C#：public Byte AnswerType { get; }
JAVA：public byte getAnswerType() { }
C++：unsigned char GetAnswerType()

说明：应答类型

00H　EAS监控启动成功
A0H　发现EAS位设置标签

### （2）6C标签QT指令（QT_6C类）

用途：设置或查询标签的QT属性。Impinj MONZA标签QT功能主要为标签提供了两种存储器工作模式以及限制远距离访问标签数据的功能。有关QT功能详细信息请参考资料【2】。

#### ① 构造函数

```
C#：public QT_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte opType,
            Byte persistent,
            Byte[] payload)
    public QT_6C(
            Byte antenna,
            Byte[] accessPwd,
            Byte opType,
            Byte persistent,
            Byte[] payload,
```

```
            Byte[] tagID,
            MemoryBank tagIDType)
JAVA: public QT_6C(
            byte antenna,
            byte[] accessPwd,
            byte opType,
            byte persistent,
            byte[] payload)
       public QT_6C(
            byte antenna,
            byte[] accessPwd,
            byte opType,
            byte persistent,
            byte[] payload,
            byte[] tagID,
            MemoryBank tagIDType)
C++: QT_6C(unsigned char antenna,
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char opType,
            unsigned char persistent,
            unsigned char *payload,
            unsigned char payloadLen)
     QT_6C(unsigned char antenna,
            unsigned char *accessPwd,
            unsigned char accessPwdLen,
            unsigned char opType,
            unsigned char persistent,
            unsigned char *payload,
            unsigned char payloadLen,
            unsigned char tagIDLen,
```

MemoryBank tagIDType)

参数：

antenna：天线端口。

accessPwd：访问密码，4字节。

opType：操作类型（R/W）。00H，查询标签的原有QT属性；01H，设置标签的QT属性

persistent：配置持久性。00H，配置仅在标签本次上电过程中有效；01H，配置可一直保存

payload：配置参数，2字节。

accessPwdLen：*accessPwd参数长度。

payloadLen：* payload参数长度。

tagID：指定要操作标签的EPC或TID。

tagIDType：tagID对应的数据区枚举。

配置参数说明：

第一字节常用位控制模式意义如下表，第二字节保留

| 比特 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 定义 | QT_SR | QT_MEM | 保留 | 保留 | 保留 | 保留 | 保留 | 保留 |

QT_SR：1，标签在开状态和安全态时会减少响应距离（或仅在近距离响应）；0，标签不减少响应距离

QT_MEM：1，标签存储器映射为公共模式；0，标签存储器映射为私密模式

② 指令应答（ReceivedMessage）

◇ 天线端口

用途：获取天线端口。

C#：public Byte Antenna { get; }

JAVA：public byte getAntenna(){ }

C++：unsigned char GetAntenna()

◇ 配置参数

用途：获取配置参数。

C#：public Byte[] Payload { get; }

JAVA：public byte[] getPayload() { }

C++：bool GetPayload(unsigned char *data, unsigned char &dataLen)

## 8、标签选择操作

### （1）国标标签选择（GB_SelectTag类）

用途：选择标签。

① 构造函数

C#：public GB_SelectTag (

GBMemoryBank memoryBank,

Byte target,

Byte rule,

UInt32 ptr,

Byte matchBitLength,

Byte[] tagData)

参数：

memoryBank：指示要匹配的数据区，GBMemoryBank类型。

target：匹配目标，值为:0,1,2,3,4。

rule：匹配规则，值为:0,1,2,3。

ptr：匹配数据起始位，单位为比特。

matchBitLength：匹配数据长度，单位为比特。

tagData：匹配数据。

JAVA：public GBSelectTag(

GBMemoryBank bank,

int target,

int rule,

int headAddress,

byte[] data)

参数：

bank：匹配存储区，0x01:标签信息区,02:编码区,03:安全区,0x30~3F:用户子区0~15。

target：匹配目标，值为:0,1,2,3,4。

rule：匹配规则，值为:0,1,2,3。

headAddress：匹配存储区起始地址，以bit为单位进行匹配。

data：匹配数据，最大长度为32bytes。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

1)选择指令只对下一次操作有效。

2)匹配数据长度为要做匹配的比特数，而匹配数据则是要匹配的数据，因为数据传输无法传送单独比特，因此采用字节传输，尾部不足8位时应补0。

例：匹配的比特数 = 6，则匹配数据应为

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| D | D | D | D | D | D | 0 | 0 |

D：比特数据　　0：补充的比特

3)标签选择指令不能单独使用，需要与标签读写指令组合使用，以实现选择特定标签进行读写的功能。使用此功能时，上位机先向读写器发送标签选择指令

再发响应标签读写指令，标签选择指令仅对标签读写指令生效一次。

### （2）6C标签选择（SelectTag_6C类）

用途：选择标签。

#### ① 构造函数

```
C#: public SelectTag_6C(
            MemoryBank memoryBank,
            UInt32 ptr,
            Byte matchBitLength,
            Byte[] tagData)

JAVA: public SelectTag_6C(
            MemoryBank memoryBank,
            int ptr,
            byte matchBitLength,
            byte[] tagData)

C++: SelectTag_6C(
            MemoryBank memoryBank,
            unsigned char ptr,
            unsigned char matchBitLength,
            unsigned char *tagData,
            unsigned char tagDataLen)
```

参数：

memoryBank：指示要匹配的数据区，MemoryBank类型。注意：不能设置为保留区。
ptr：匹配数据起始地址，单位为比特。
matchBitLength：匹配数据长度，单位为比特。
tagData：匹配数据。
tagDataLen：*tagData参数长度。

#### ② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

### ③ 指令说明

- 选择指令只对下一次操作有效。
- 匹配数据长度为要做匹配的比特数，而匹配数据则是要匹配的数据，因为数据传输无法传送单独比特，因此采用字节传输，尾部不足8位时应补0。

  例：匹配的比特数 = 6，则匹配数据应为

| B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|---|---|---|---|---|---|---|---|
| D | D | D | D | D | D | 0 | 0 |

  D：比特数据　　0：补充的比特

- 标签选择指令不能单独使用，需要与标签读写指令组合使用，以实现选择特定标签进行读写的功能。使用此功能时，上位机先向读写器发送标签选择指令再发响应标签读写指令，标签选择指令仅对标签读写指令生效一次。例如读写器需要先读取标签A的TID码再写标签A的USER区数据，则需要进行以下下流程来实现：上位机发送标签选择指令选择A——>上位机发送读TID码指令——>上位机发送标签选择指令选择A——>上位机发送写标签USER区数据指令。

## 9、标签按时间过滤（FilterByTime类）

用途：指定时间内相同标签的数据不会再次上传。该功能用于过滤指定时间读写器循环读卡时收到的重复标签数据，减少冗余数据上传来减少上位机数据处理的开销。

### ① 构造函数

C#：public FilterByTime(Byte opType, Byte[] time)

JAVA：public FilterByTime(byte opType, byte[] time)

C++：FilterByTime(
```
            unsigned char opType,
            unsigned char *time,
            unsigned char timeLen)
```

参数：

opType：操作类型。00H：设置；01H：取消(后一项忽略)

time：过滤时间，2字节，单位为100毫秒。
timeLen：*time参数长度。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

# 七、停止指令，关功放（PowerOff类）

用途：停止指令用于中止读写器的标签读写操作，并关闭读写器的载波发射，让读写器切换到空闲等待状态。

① 构造函数

C#：public PowerOff ()
JAVA：public PowerOff ()
C++：PowerOff ()

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

注意：该指令将停止当前读写器所有天线工作。

# 八、GPIO功能

## 1、GPI输入状态查询（Gpi_800类）

用途：本功能用于查询读写器GPI端口输入电平状态。

### ① 构造函数

C#：public Gpi_800()
JAVA：public Gpi_800()
C++：Gpi_800(unsigned char  checkMode)

### ② 指令应答（ReceivedMessage）

查询数据

用途：获取查询数据，2字节，分别为I/O状态和端口状态。

C#：public Byte[] QueryData { get; }

JAVA：public byte[] getQueryData()

C++：bool GetQueryData(unsigned char *data,unsigned char &dataLen)

### ③ 指令说明

I/O状态

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 定义 | 保留 | 保留 | 保留 | 保留 | 端口4 | 端口3 | 端口2 | 端口1 |

端口状态：0：低电平；1：高电平

## 2、GPO输出操作（Gpo_800类）

用途：本功能控制指定的读写器GPO端口的输出状态，需要特别说明的是，有些型号的读写器具有继电器输出端口，我们也将其定义为GPO端口，在逻辑上我们将其闭合状态与断开状态分别对应与电平的高和低。

### ① 构造函数

C#：public Gpo_800(Byte ioPort, Byte level)
JAVA：public Gpo_800(byte ioPort, byte level)
C++：Gpo_800(unsigned char ioPort,unsigned char level)

参数：

ioPort：输出端口。用于指定需要控制输出的端口号，与其硬件标识一致。

level：输出状态。00，代表输出低电平或继电器开路；01，代表输出高电平或继电器闭合。

② 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

③ 指令说明

| | | |
|---|---|---|
| 输出端口： | 01H | 端口1 |
| | 02H | 端口2 |
| | 03H | 端口3 |
| | 04H | 端口4 |
| 输出电平： | 00H | 输出低电平 |
| | 01H | 输出高电平 |
| | 03H | 输出正脉冲 |
| | 04H | 输出负脉冲 |

## 3、GPI事件触发信息上传（RXD_IOTriggerSignal_800类）

用途：GPI触发信号。继承自BaseMessageNotification抽象类。由于我们的读写器GPI端口实现了事件触发机制，所以当GPI的状态满足5.3.4中定义的触发或停止条件时，读写器或上上位机主动上传一条通知信息。

① 指令应答（ReceivedMessage）

◇ 输入端口

用途：获取输入端口号。

C#：public Byte GPIPort { get; }

JAVA：public byte getGPIPort()

C++：unsigned char GetGPIPort ()

◇ 触发信号

用途：获取触发开始或停止信号。真为开始，假为停止。

C#：public Boolean IsStart{ get; }

JAVA：public boolean isStart ()

C++：bool GetIsStart ()

◇ 触发时间

用途：获取触发时间，8字节UTC时间。前4字节代表UTC时间的秒，后4字节代表UTC时间的微秒。

C#：public Byte[] UTCTime{ get; }

JAVA：public byte[] getUTCTime()

C++：bool GetUTCTime (unsigned char *pUTCTime, unsigned char UTC-TimeLen)

**② 指令说明**

输入端口

| 位 | Bit7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|---|
| 定义 | 保留 | 保留 | 保留 | 保留 | 保留 | 保留 | 端口2 | 端口1 |

# 九、手持式读写器专用模块

远望谷手持式读写器专用模块，控制远望谷手持式读写器RFID模块、GPRS、Wi-Fi等硬件设备。

## 1、数据包

封装了字节数组（byte）的数据包。

程序集：Invengo.dll

命名空间：Invengo

类名：Datagram

支持的读写器：XC2900、XC2903

### （1）构造

public Datagram()

构造空实例；

public Datagram(int length)

指定长度实例化数据包；

public Datagram(byte[] data)

使用有效的字节数据实例化数据包。

public Datagram(string hexstring)

十六进制字符串实例化数据包，字符串的长度必须是偶数，并且是十六进制的字符串，如：“19A0B1C2D3F4”；

### （2）数据属性

public byte[] Data　　数据包的字节数据；

public int Length　　数据的长度；

### （3）转换成字符串

用途：把数据包的数据转换成指定的字符串；

原形：public string ToString(string formatter) 或

public override string ToString()

参数：

string formatter

　　ASCII、UTF-8、UTF-16、UNICODE，其它默认转成十六进制字符。

返回值：

转换之后的字符串。

## 2、RFID控制

打开RFID模块电源，打开RFID电源之后才可以进行相关的RFID操作。

程序集：Invengo.Devices.ModuleControl.dll

命名空间：Invengo.Devices.ModuleControl

类名：RfidControl

支持的读写器：XC2900、XC2903

### （1）打开RFID

功能：打开RFID模块的电源，使之可以正常工作；

原形：public static int Active()

参数：（无）

返回值：

int　　0 模块上电失败

1 模块上电成功（断电状态到上电的状态）

2 模块已经上电

### （2）关闭RFID

功能：关闭RFID模块的电源；

原型：public static bool Deactive()

参数：（无）

返回值：

true：成功；false：失败；

### （3）RFID电源状态

用途：获取RFID模块当前电源是打开或关闭状态；

原形：public static bool IsActived()

参数：（无）

返回值：

true： 打开； false： 关闭；

## 3、GPRS控制

实现GPRS模块的激活、关闭、拔号连接、断开拔号；激活了GPRS模块，就能够使用操作系统的GPRS拔号程序进行拔号操作；除了提供的拔号连接方法之外，也可以参考Windows CE操作系统提供的RAS API资料，实现拔号功能。

程序集：Invengo.Devices.ModuleControl.dll

命名空间：Invengo.Devices.ModuleControl

类名：GprsControl

支持的读写器：XC2900、XC2903

### （1）打开GPRS

功能：打开GPRS模块电源，使GPRS模块处于工作状态；

原形：public static bool Active()

参数：（无）

返回值：

true： 激活成功； false： 失败；

注意：当第1次打开GPRS模块的时候，GPRS模块需要扫描网络，所以要等待至少20秒的时间才能拔号；

### （2）关闭GPRS

用途：关闭GPRS模块；

原形：public static bool Deactive()

参数：（无）

返回值：

true： 成功； false： 失败；

### （3）GPRS电源状态

用途：获取当前模块电源是打开或是关闭状态。

原形：public static bool IsActived()

参数：（无）

返回值：

true： 打开； false： 关闭；

### （4）拔号连接

用途：打开GPRS模块，并进行拔号；拔号之前，可以使用操作系统自带的GPRS拔号程序更改连接参数；

原形：public static bool Connect()

参数：（无）

返回：

true： 成功； false： 失败；

### （5）断开拔号

用途：断开拔号连接，并关闭GPRS模块。

原形：public static bool Disconnect()

参数：（无）

返回值：

true：成功；false：失败；

## 4、Wi-Fi控制

实现Wi-Fi模块电源的打开、关闭，Wi-Fi模块打开之后可以使用操作系统的Wi-Fi连接功能进行配置Wi-Fi。

程序集：Invengo.Devices.ModuleControl.dll

命名空间：Invengo.Devices.ModuleControl

类名：WifiControl

支持的读写器：XC2900、XC2903

### （1）打开Wi-Fi

用途：打开Wi-Fi模块电源，使得Wi-Fi能够正常工作；Wi-Fi模块激活之后，可以使用操作系统自带的Wi-Fi配置程序进行相关操作；

原形：public static bool Active()

参数：（无）；

返回值：

true：成功；false：失败；

### （2）关闭Wi-Fi

用途：关闭Wi-Fi模块。

原形：public static bool Deactive()

参数：（无）

返回值：

true：成功；false：失败；

### （3）Wi-Fi电源状态

用途：获取当前模块电源是打开或是关闭状态。

原形：public static bool IsActived()

参数：（无）

返回值：

true：打开；false：关闭；

## 5、捕捉系统电源消息

手持式读写器（PDA）切换省电状态（待机），相关的模块电源（例如RFID）有可能自动断开，重新唤醒PDA时得重新给打开模块电源才能继续工作；手持式读写器操作系统提供了捕获这种电源变化的消息，为了方便应用开发，我们

也提供了捕捉电源状态变化的处理方法。

程序集：Invengo.dll

命名空间：Invengo.Platform

类名：PowerStateNotify

支持的读写器：XC2900、XC2903

### （1）电源消息的委托

用途：捕捉电源状态消息的委托，用于接收消息。

原形：public delegate void FnPowerNotifyCallback(uint stateFlags)

参数：

uint stateFlags 电源状态消息值；

### （2）启动捕捉电源消息

用途：启动捕捉电源状态的消息。

原形：public static void PowerNotifyStart(

PowerStateNotify.FnPowerNotifyCallback func)

参数：

FnPowerNotifyCallback func FnPowerNotifyCallback委托例，用于接收消息；

### （3）停止捕捉电源消息

用途：停止捕捉电源消息，程序退出时必须调用。

原形：public static extern void PowerNotifyStop()

### （4）电源状态的消息值

唤醒状态：

public const uint POWER_STATE_ON = 0x10000

休眠状态：

public const uint POWER_STATE_OFF = 0x20000

启动状态：

public const uint POWER_STATE_BOOT = 0x80000

空闲状态：

public const uint POWER_STATE_IDLE = 0x100000

挂起状态：

public const uint POWER_STATE_SUSPEND = 0x200000

重置状态：

public const uint POWER_STATE_RESET = 0x800000

## 6、条码识别/拍照的切换

如果是XC2903手持式读写器，条码识别和拍照功能不能同使用，只选择其中一种功能。

程序集：Invengo.Devices.ModuleControl.dll

命名空间：Invengo.Devices.ModuleControl

类名：CameraBarcodeSwitch

支持的读写器：XC2903

### （1）查询当前模式

用途：查询当前使用的是读取条码或是拍照模式；

原形：public static int BarCodeOrCameraQuery()

参数：（无）

返回值：

int　　0：拍照；1：读取条码；

### （2）选择条码识别

用途：切换到读取条码模式；

原形：public static void SelBarCode()

### （3）选择拍照

用途：切换到拍照模式；

原形：public static void SelCamera()

## 7、条码识别

手持式读写器有可能装配的条码头不同（具体装配了哪种型号的条码头请参考用户手册），不同类型的条码头提供相应的类。

### （1）BarcodeScannerSymbol类

用途：控制迅宝1/2维条码头工作；

程序集：Invengo.Barcode.dll

命名空间：Invengo.Barcode

支持的读写器：XC2900、XC2903

① 实例化

功能：实例化BarcodeScannerSymbol类。

原形：public static BarcodeScannerSymbol CreateInstance()

参数：（无）

返回值：

返回BarcodeScannerSymbol实例。

② 接收条码数据事件

用途：接收条码数据事件；

原形：public event BarcodeRecvEventHandler BarcodeRecvEvent

事件参数：

BarcodeRecvEventArgs e　Code成员就是条码数据；

③ 初始化

用途：初始化读取条码的功能；

原形：public void Initialize()

④ 触发读条码

功能：触发条码头进行扫描条码，适用于讯宝(Symbol)1/2维自适应条码头；

原形：public bool BarcodeRead()

返回值：

true： 成功； false：失败；

⑤ 关闭读取条码

用途：关闭读取条码功能；

原形：public void Close()

⑥ 释放资源

功能：释放读取条码的资源，同时停止条码工作；

原形：public void Dispose()

### （2）BarcodeScannerHoneywell1D类

功能：控制霍尼韦尔1维自适应条码头工作；

程序集：Invengo.Barcode.dll

命名空间：Invengo.Barcode

支持的读写器：XC2900、XC2903

① 实例化

功能：实例化BarcodeScannerHoneywell1D类。

原形：public static BarcodeScannerHoneywell1D CreateInstance()

参数：（无）

返回值：

返回BarcodeScannerHoneywell1D实例。

② 接收条码数据事件

用途：接收条码数据事件；

原形：public event BarcodeRecvEventHandler BarcodeRecvEvent

事件参数：

BarcodeRecvEventArgs e　Code成员就是条码数据；

③ 初始化

用途：初始化读取条码的功能；

原形：public void Initialize()

④ 触发读条码

功能：触发条码头进行扫描条码，适用于讯宝(Symbol)1/2维自适应条码头；

原形：public bool BarcodeRead()

返回值：

true：　成功；　false：失败；

⑤ 关闭读取条码

用途：关闭读取条码功能；

原形：public void Close()

⑥ 释放资源

功能：释放读取条码的资源，同时停止条码工作；

原形：public void Dispose()

### （3）BarcodeScannerHoneywell2D类

功能：控制霍尼韦尔1/2维自适应条码头工作；

程序集：Invengo.Barcode.dll

命名空间：Invengo.Barcode

支持的读写器：XC2900、XC2903

① 实例化

功能：实例化BarcodeScannerHoneywell2D类。

原形：public static BarcodeScannerHoneywell2D CreateInstance()

参数：（无）

返回值：

返回BarcodeScannerHoneywell2D实例。

② 接收条码数据事件

用途：接收条码数据事件；

原形：public event BarcodeRecvEventHandler BarcodeRecvEvent

事件参数：

BarcodeRecvEventArgs e　Code成员就是条码数据；

③ 初始化

用途：初始化读取条码的功能；

原形：public bool BarcodeCE5_SInitBig()

返回值：

true：　成功；　false：失败；

④ 触发读条码

功能：触发条码头进行扫描条码，适用于讯宝(Symbol)1/2维自适应条码头；

原形：public bool BarcodeRead()

返回值：

true：　成功；　false：失败；

⑤ 释放资源

功能：释放读取条码的资源，同时停止条码工作；

原形：public void Dispose()

### （4）BarcodeScannerMoto类

功能：控制摩托罗拉1/2维自适应条码头工作；

程序集：invengo.barcode.dll

命名空间：Invengo.Barcode

类名：BarcodeScannerMoto

支持的读写器：XC2900、XC2903

① 实例化

功能：实例化BarcodeScannerMoto类。

原形：public static BarcodeScannerMoto CreateInstance()

参数：（无）

返回值：

返回BarcodeScannerMoto实例。

② 单次读取条码

用途：单次读取条码；

原形：public void SingleRead()

参数：（无）

③ 扫读条码

用途：扫读条码；

原形：public void StartScan()

参数：（无）

④ 停止扫读条码

用途：停止扫读条码；

原形：public void StopScan()

参数：（无）

⑤ 接收条码解码数据事件

用途：接收条码解码数据事件；

原形：public event EventHandler Decoded

事件参数：

EventArgs e

⑥ 恢复触发模式

用途：恢复触发模式；

原形：public void ResetTrigger()

返回值：（无）

⑦ 把扫描仪设为解码模式

功能：把扫描仪设为解码模式

原形：public void DecodeMode()

返回值：（无）

⑧ 设置触发读条码

功能：触发条码头进行扫描，适用于摩托罗拉1/2维自适应条码头；

原形：public bool Trigger

设置：

true： 触发扫描； false：停止扫描；

⑨ 解码会话

功能：判断是否正在解码

原形：public bool fInSession

返回值：

true： 正在解码； false：解码结束；

⑩ 释放资源

功能：释放读取条码的资源

原形：public void Release()

返回值：（无）

### （5）DecodeData类

功能：摩托罗拉1/2维自适应条码头解码数据

程序集：Invengo.Barcode.dll

命名空间：Invengo.Barcode

类名：DecodeData

支持的读写器：XC2900、XC2903

① 解码数据

功能：解码数据。

原形：public static string text

## 8、功能按键消息

### （1）捕捉消息

当点击功能键时，他们分别会产生不同的按键消息，把窗体属性KeyPreview设置为“true”，实现窗体事件KeyDown和KeyPress事件就可以捕捉这些消息。

① XC2900手持式读写器的功能按键定义：

SCAN键：0XF5

左功能键：0xF2

右功能键：0xF1

② XC2903手持式读写器的功能按键定义：

左功能键：0xF2

右功能键：0xF1

## 9、声音音量

获取或设置声音的音量大小

程序集：Invengo.dll

命名空间：Invengo.Audio

类名：SoundVolume

支持的读写器：XC2900、XC2903

### （1）设置音量

用途：设置声音音量。

原形：public static void SetSoundVolume(ulong dwVolume);

参数：

ulong dwVolume

音量值，范围：0x0000～0xFFFF；也可按百分比的形式计算，如：oriVolume = (ulong)(0xFFFF * 1.0 * 70 / 100.0)。

### （2）获取音量

用途：获取声音音量。

原形：public static void GetSoundVolume(ref ulong dwVolume);

参数：

ref ulong dwVolume

返回当前声音音量值。

## 10、播放声音文件

用于播放声音文件。

程序集：Invengo.dll

命名空间：Invengo.Audio

类名：SoundPlayer

支持的读写器：XC2900、XC2903

### （1）播放WAV

用途：播放WAV格式的声音文件。

原形：public static bool PlayWAV(string strFileName)

参数：

string strFileName

声音文件的完整名称。

返回值：

true：成功； false： 失败

## 11、日期时间

设置/获取手持式读写器的系统日期时间。

程序集：Invengo.dll

命名空间：Invengo.Platform

类名：SysTime

支持的读写器：XC2900、XC2903

### （1）日期时间的结构体

日期时间的结构体定义：

```
public struct SYSTEMTIME
{
   public ushort wYear;          //年
   public ushort wMonth;         //月
   public ushort wDayOfWeek;     //周
public ushort wDay;              //日
   public ushort wHour;          //时
   public ushort wMinute;        //分
   public ushort wSecond;        //秒
   public ushort wMilliseconds; //毫秒
}
```

### （2）获取日期时间

用途：获取当前系统的日期时间；

原形：public static void GetLocalTime(ref SYSTEMTIME lpSystemTime)

参数：

SYSTEMTIME lpSystemTime　　日期时间的结构体；

### （3）设置日期时间

用途：设置当前系统的日期时间；

原形：public static bool SetLocalTime(ref SYSTEMTIME lpSystemTime)

参数：

SYSTEMTIME lpSystemTime　　日期时间的结构体；

返回值：

true：成功；false：失败；

## 12、待机/重启/关机

控制手持式读写器的待机、重启、关机操作。

程序集：Invengo.dll

命名空间：Invengo.Platform

类名：SystemPower

支持的读写器：　XC2903

### （1）待机

用途：控制手持式读写器切换到待机状态；

原形：public static bool Suspend()

参数：（无）

返回值：

true：成功；false：失败；

### （2）重启

用途：重新启动手持式读写器；

原形：public static bool Reboot()

参数：（无）

返回值：

true：成功；false：失败；

### （3）关机

用途：关闭手持式读写器；

原形：public static bool ShutDown()

参数：（无）

返回值：

true：成功；false：失败；

## 13、数据转换

提供数据格式的转换，如字节数据转换成十六进制字符串或者十六进制字符串转换成字节数组；

程序集：Invengo.dll

命名空间：Invengo

类名：FormatConvert

支持的读写器：XC2900、XC2903

### （1）字节数据转换成十六进制字符串

用途：字节数据转换成十六进制字符串(由0-9、A-F组成的十六进制字符串)；

原形：public static string BytesToString(byte[] dataBytes, int size)
或

public static string BytesToString(byte[] dataBytes, int offset, int size)

参数：

byte[] dataBytes

字节数组；

int offset

起始地址；

int size

转换的字节个数；

返回值：

十六进制字符串(由0-9、A-F组成的十六进制字符串)；

### （2）十六进制字符串转换成字节数据

用途：十六进制字符串转换成字节数据

原形：public static byte[] HexStringToBytes(string hexValue)

参数：

string hexValue

由0-9、A-F组成的十六进制字符串，长度必须是偶数，如：“19A0B1C2D3F4”；

返回值：

字节数组；

# 十、附录

## 1、上位机控制读写器读写标签流程示例

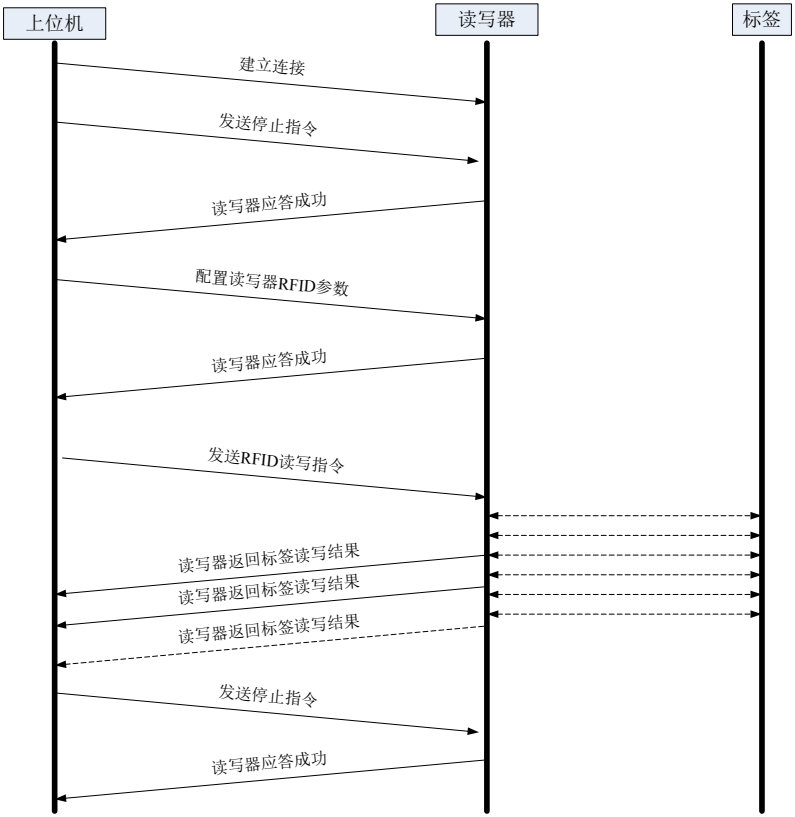



图 10-1

上位机通过指定端口向读写器发起连接，连接成功后，上位机首先向读写器发送停止指令的主要目的是：1读写器切换到空闲状态，可正常响应随后的指令；2. 根据读写器的响应是否成功来确认读写器当前状态是否正常可用。

## 2、读写器错误代码表

错误代码表为读写器执行操作失败时返回到上位机的错误代码。

表1 读写器系统错误代码表

| 错误代码 | 错误代码定义 |
|---|---|
| 01H | 保留 |
| 02H | 保留 |
| 03H | 保留 |
| 04H | 保留 |
| 05H | 保留 |
| 06H | 读写器射频硬件错误 |
| 07H | 保留 |
| 08H | 保留 |
| 09H | 保留 |
| 0AH | 保留 |
| 0BH | 保留 |
| 0CH | 保留 |
| 0DH | 保留 |
| 0EH | 保留 |
| 0FH | 属于该范围但读写器无法判断或未知类型错误 |

表2 读写器操作错误代码表

| 错误代码 | 错误代码定义 |
|---|---|
| 11H | 保留 |
| 12H | 使用了错误的天线端口 |
| 13H | 保留 |
| 14H | 保留 |
| 15H | 当前状态不允许执行此操作 |
| 16H | 保留 |
| 17H | 保留 |
| 18H | 保留 |
| 19H | 读写器内部温度超限 |
| 1AH | 前后向功率差值过低（驻波比过大） |
| 1BH | 参数不可改写 |
| 1CH | 保留 |
| 1DH | 保留 |
| 1EH | 保留 |
| 1FH | 属于该范围但读写器无法判断或未知类型错误 |

表3 指令接收/数据传输错误代码表

| 错误代码 | 错误代码定义 |
|---|---|
| 20H | 接收指令或数据不完整 |
| 21H | 接收指令或数据的CRC校验错误 |
| 22H | 保留 |
| 23H | 保留 |
| 24H | 指令参数有错误 |
| 25H | 指令帧结构错误（缺少域） |
| 26H | 不支持的指令类型 |
| 27H | 读写器接收到太多指令，暂时无法处理 |
| 28H | 保留 |
| 29H | 保留 |
| 2AH | 保留 |
| 2BH | 保留 |
| 2CH | 保留 |
| 2DH | 保留 |
| 2EH | 保留 |
| 2FH | 属于该范围但读写器无法判断或未知类型错误 |

表4 EPC标签操作错误代码表

| 错误代码 | 错误代码定义 |
|---|---|
| 60H | 标签无响应或不存在 |
| 61H | 保留 |
| 62H | 标签返回信息：标签操作地址溢出 |
| 63H | 标签返回信息：操作存储区被锁定 |
| 64H | 标签存取密码错误 |
| 65H | 标签灭活密码错误 |
| 66H | 保留 |
| 67H | 保留 |
| 68H | 保留 |
| 69H | 标签返回信息：未知类型错误 |
| 6AH | 标签返回信息：功率不足 |
| 6BH | 保留 |
| 6CH | 保留 |
| 6DH | 保留 |
| 6EH | 保留 |
| 6FH | 属于该范围但读写器无法判断或未知类型错误 |

## 3、读写器功能表

注:“●”表示标准功能,“○”表示定制产品功能

| 读写器功能对应项 | 807 | 850 | 861 | 825 | 829 | 811 | 812 | 503B | 2900 | 2903 |
|---|---|---|---|---|---|---|---|---|---|---|
| 5.1 SysQuery_800 | | | | | | | | | | |
| 21H 读写器型号 | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 23H 基带处理器固件版本 | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 29H 应用处理器固件版本 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 0DH 串口波特率 | ● | ● | ● | ● | ● | ● | ● | ● | | |
| 05H MAC地址 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 06H IP地址 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 50H 读写器TCP连接模式 | ● | ● | ● | | | ● | ● | ● | | |
| 07H 读写器TCP服务端口号 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 51H 备选服务器 | ● | ● | ● | | | ● | ● | ● | | |
| 52H 连接重试次数 | ● | ● | ● | | | ● | ● | ● | | |
| 03H 读写器天线端口功率 | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| E2H GPI触发参数 | ● | ● | ● | | ● | ● | ● | ● | ● | ● |
| 5.2 SysConfig_800 | | | | | | | | | | |
| 0DH 串口波特率 | ● | ● | ● | ● | ● | ● | ● | ● | | |
| 06H IP地址 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 50H 读写器TCP连接模式 | ● | ● | ● | | | ● | ● | ● | | |
| 07H 读写器TCP服务端口号 | ● | ● | ● | | | ● | ● | ● | ● | ● |
| 51H 备选服务器 | ● | ● | ● | | | ● | ● | ● | | |
| 52H 连接重试次数 | ● | ● | ● | | | ● | ● | ● | | |
| 03H 读写器天线端口功率 | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| E2H GPI触发参数 | ● | ● | ● | | ● | ● | ● | ● | ● | ● |
| 6.2.1 ReadTag | | | | | | | | | | |
| EPC_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| TID_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| EPC_TID_UserData_6C | | | | | | | | | ● | ● |
| EPC_TID_UserData_6C_2 | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| ID_6B | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| ID_UserData_6B | | | | | | | | | ● | ● |
| EPC_6C_ID_6B | | | | | | | | | ● | ● |
| TID_6C_ID_6B | | | | | | | | | | |
| EPC_TID_UserData_6C_ID_UserData_6B | | | | | | | | | | |
| EPC_TID_UserData_Reserved_6C_ID_UserData_6B | ● | ● | ● | ● | ● | ● | ● | ● | | |
| EPC_PC_6C | | | | | | | | | | |
| 6.2.2 RXD_TagData | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.2.3 ReadUserData_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.1 WriteEpc | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.2 WriteUserData_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.3 WriteUserData2_6B | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.4 AccessPwdConfig_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.5 KillPwdConfig_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.3.6 WriteTag_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.4.1 LockMemoryBank_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.4.2 LockUserData_6B | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.4.3 LockStateQuery_6B | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.5 KillTag_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.6.1.1 EasConfig_6C | ● | ● | ● | ● | ● | ● | ● | ● | | |
| 6.6.1.2 EasAlarm_6C | ● | ● | ● | ● | ● | ● | ● | ● | | |
| 6.6.2 QT_6C | ● | ● | ● | ● | ● | ● | ● | ● | | |
| 6.7 SelectTag_6C | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 6.8 FilterByTime | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 7 PowerOff | ● | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| 8.1 Gpi_800 | ● | ● | ● | | ● | ● | ● | ● | | |
| 8.2 Gpo_800 | ● | ● | ● | | ● | ● | ● | ● | | |
| 8.3 RXD_IOTriggerSignal_800 | ● | ● | ● | | ● | ● | ● | ● | | |

版本号：V1.4.1

深圳市远望谷信息技术股份有限公司
Invengo Information Technology Co.,Ltd.
地址：深圳市光明新区甲子塘同观路远望谷射频识别产业园
邮编：518132
电话：0755-26525035
传真：0755-26711693
www.invengo.cn
E-mail: sales@invengo.cn

免费咨询热线：400-888-0058