|  |
| --- |
| 深圳运宝通电子科技有限公司 |
| RFID 读写器API 技术参考手册 |
|  |
|  |
|  |
| **2019-06-25** |

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日 期 | 修订版本 | 描 述 | 作 者 |
| 2019/06/25 | V1.00 | 初稿完成 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1. 概述 4](#_Toc22284160)

[1.1 API组成 4](#_Toc22284161)

[1.2适用范围 4](#_Toc22284162)

[1.3示例代码 4](#_Toc22284163)

[1.4缩略语定义 4](#_Toc22284164)

[1.5阅读说明 4](#_Toc22284165)

[2. 公共调用说明 5](#_Toc22284166)

[2.1 API使用概述 5](#_Toc22284167)

[2.2 读写器基类（Reader类） 9](#_Toc22284168)

[2.3 消息指令抽象类（AbstractHostMessage抽象类） 15](#_Toc22284169)

[2.4 配置文件（Sysit.xml） 17](#_Toc22284170)

[3. Message接口列表 18](#_Toc22284171)

[3.1固件程序升级准备（MsgFirmwareUpgradePreparation类） 18](#_Toc22284172)

[3.2固件程序在线升级（MsgFirmwareOnlineUpgrade类） 18](#_Toc22284173)

[3.3查询模块软硬件版本（MsgReaderVersionQuery类） 19](#_Toc22284174)

[3.4设置/查询使用的天线（MsgAntennaConfig类） 19](#_Toc22284175)

[3.5查询 RFID 工作状态（MsgRfidStatusQuery类） 20](#_Toc22284176)

[3.6设置/查询RS232波特率（MsgRs232BaudRateConfig类） 21](#_Toc22284177)

[3.7设置/查询RS485波特率（MsgRs485BaudRateConfig类） 22](#_Toc22284178)

[3.8设置/查询本机 RS-485 地址（MsgRs485AddressConfig类） 22](#_Toc22284179)

[3.9查询 GPIO 接口定义（MsgGpioDefinitionQuery类） 23](#_Toc22284180)

[3.10设置/查询 GPIO 输入触发的标签操作（MsgGpiTriggerConfig类） 23](#_Toc22284181)

[3.11设置GPIO输出电平（MsgGpoConfig类） 24](#_Toc22284182)

[3.12查询GPIO输入电平（MsgGpiQuery类） 25](#_Toc22284183)

[3.13设置/查询重复标签过滤时间（MsgFilteringTimeConfig类） 26](#_Toc22284184)

[3.14设置/查询间隔读标签时间（MsgIntervalTimeConfig类） 26](#_Toc22284185)

[3.15设置/查询信号强度（RSSI）过滤阀值（MsgRssiThresholdConfig类） 27](#_Toc22284186)

[3.16设置/查询工作频段（MsgUhfBandConfig类） 28](#_Toc22284187)

[3.17设置/查询端口功率值（MsgPowerConfig类） 28](#_Toc22284188)

[3.18设置/查询支持的空口协议（MsgAirProtocolConfig类） 29](#_Toc22284189)

[3.19设置/查询工作频率（MsgFrequencyConfig类） 29](#_Toc22284190)

[3.20恢复出厂设置（MsgResetToFactoryDefault类） 32](#_Toc22284191)

[3.21设置/查询空闲等待时间（MsgIdleTimeConfig类） 32](#_Toc22284192)

[3.22设置/查询IP地址（MsgIpAddressConfig类） 33](#_Toc22284193)

[3.23设置/查询服务端、客户端模式（MsgTcpModeConfig类） 34](#_Toc22284194)

[3.24查询MAC地址（MsgMacConfig类） 35](#_Toc22284195)

[3.25设置/查询读 6C 标签数据帧包含字段（Msg6CTagFieldConfig类） 35](#_Toc22284196)

[3.26读6C标签EPC区（MsgTagInventory类） 36](#_Toc22284197)

[3.27通用读6C标签（MsgTagRead类） 37](#_Toc22284198)

[3.28写6C标签（MsgTagWrite类） 38](#_Toc22284199)

[3.29 6C标签锁操作（MsgTagLock类） 39](#_Toc22284200)

[3.30销毁 6C 标签（MsgTagKill类） 40](#_Toc22284201)

[3.31设置/查询UTC时间（MsgUtcConfig类） 40](#_Toc22284202)

[3.32开功放（内部测试使用）（MsgPowerOn类） 41](#_Toc22284203)

[3.33关功放（MsgPowerOff类） 41](#_Toc22284204)

[3.34应用端固件程序升级准备（MsgAppFirmwareUpgradePreparation类） 42](#_Toc22284205)

[3.35应用端固件在线升级（MsgAppFirmwareOnlineUpgrade类） 42](#_Toc22284206)

[3.36查询应用处理器软件及硬件版本（MsgCpuVersionQuery类） 43](#_Toc22284207)

[3.37选择6C标签指令（MsgTagSelect类） 43](#_Toc22284208)

[3.38查询读写器能力指令（MsgReaderCapabilityQuery类） 43](#_Toc22284209)

[4. 常见应用 44](#_Toc22284210)

[4.1读写器连接 44](#_Toc22284211)

[4.2标签扫描 49](#_Toc22284212)

[4.3标签操作 52](#_Toc22284213)

[附录A：上位机控制读写器读写标签流程示例 56](#_Toc22284214)

[附录B：错误代码描述 56](#_Toc22284215)

# 概述

## 1.1 API组成

提供三套编程语言API可供选择

C#：包含API.dll、Core.dll文件，基于Microsoft .NET Framework v4.0框架编写。

Java: VbotoUhfAPI.jar文件，基于jdk-1.8编写。

C++:待完善

## 1.2适用范围

本文档适用的读写器型号有：

## 1.3示例代码

提供三套编程语言API可供选择

C#：VbotoUhfTest工程文件，基于Visual Studio 2015编写。

Java: VbotoUhfTest工程文件，基于Eclipse IDE for Eclipse Committers Version: 4.11.0编写。

C++:待完善

## 1.4缩略语定义

VRP：Vboto Reader Protocol，运宝通读写器数据传输控制协议。

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。

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。

字：两个字节为一字。

## 1.5阅读说明

本参考手册，以C#语言为主导，如无特殊说明，其他语言参考C#语言说明。

# 公共调用说明

## 2.1 API使用概述

本API采用面向对象设计，支持多读写器调用，每个读写器实例化一个Reader类控制，每个Reader示例之间互不干扰。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。

本API简单分为读写器操作类，和通讯消息类。

读写器操作类（Reader类）简单说明如下

1. 指定读写器连接，构造对应读写器操作，public Reader(String readerName, IPort commPort)
2. 建立读写器连接，public ConnectResponse Connect()
3. 发送接收指令消息，public bool Send(IHostMessage msg)
4. 接收盘存消息，C#：public event InventoryReceivedHandle OnInventoryReceived

JAVA：public EventHandler OnInventoryReceived = new EventHandler();

1. 断开读写器连接，public void Disconnect()

通讯消息类，简单说明如下

1. 读写器主动上传消息类，IReaserMessage接口
2. 上位机消息指令接口，IHostMessage，该接口继承自IReaserMessage接口
3. 第3章所有接口均继承自IHostMessage，作为Reader类的Send方法的参数，完成指令交互
4. 建议一条消息新建一个消息类，如需重复使用，IReaserMessage接口提供Clone方法初始化消息。

本API约定，如未特殊说明，则遵循如下规则

1. Reader类C#：位于NetAPI.Protocol.VRP命名空间中

JAVA：位于JavaAPI.Protocol.VRP包中

1. 所有IHostMessage的子类，位于NetAPI.Protocol.VRP命名空间中

JAVA：位于JavaAPI.Protocol.VRP包中

1. 所有参数、枚举类，位于NetAPI.Entities命名空间中

JAVA：位于JavaAPI.Entities包中

1. **API使用总体流程**

如图：



上图中，读写器建立连接后，发送指令方式有两种：

1. 一种是发送消息后，同步等待，等待时间可指定，默认1秒
2. 一种是发送消息后，不等待读写器执行，直接将控制权交还系统。

另外对于读写器主动上传的消息，也可通过事件回调机制，异步接收读写器信息；此类信息如标签扫描数据、IO触发数据等。

1. **API中同步消息流程**

如图：



上图中，发送指令流程为：

1. 实例化消息类，该类消息继承自IHostMessage接口，利用具体类的构造函数参数，传递消息参数；
2. 使用Reader类中Send方法发送指令给读写器；
3. 等待消息返回（此步骤API自动完成，无需客户干预）
4. 消息返回于第1）步骤实例化的消息类中，利用此类的ReceivedMessage属性（JAVA：getReceivedMessage方法）获取返回消息内容；
5. 如果消息执行失败或超时，Send方法返回False，并通过IHostMessage接口的ErrorInfo属性（JAVA：getErrorInfo方法）返回错误信息
6. **API中发送不需等待返回消息流程**

如图：



上图中，发送指令流程为：

1. 实例化消息类，该类消息继承自IHostMessage接口，IsReturn变量值为false（JAVA：setIsReturn方法设置false），利用具体类的构造函数参数，传递消息参数；
2. 使用Reader类中Send方法发送指令给读写器；
3. 发送指令后，方法直接返回指令是否发送成功，不会等待读写器响应
4. 如果成功，Send方法返回True，则程序继续往下执行
5. 如果失败，Send方法返回False，并通过IHostMessage接口的ErrorInfo属性（JAVA：getErrorInfo方法）返回错误信息；程序继续往下执行
6. **API中接收读写器主动上传信息流程**

如图：



上图中，接收读写器主动上传消息流程为：

1. 读写器建立连接成功后，添加监听事件（OnInventoryReceived）；
2. 当读写器有标签信息上传时，会触发该事件；
3. 处理该事件回调；
4. 在读写器断开连接前，删除监听事件。

注意：读写器触发事件为一单独线程，该线程不是程序主线程。

## 2.2 读写器基类（Reader类）

C#：命名空间： NetAPI.Protocol.VRP

JAVA：包位置： JavaAPI.Protocol.VRP

该类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（Reader方法）**

用途：实例化Reader类。

语法：public Reader(String readerName)

参数：

readerName：读写器名称，字符串。

备注：

该构造函数必须与配置文件(见2.4节)配合使用。readerName为自定义字符串，在配置文件中保持唯一。构造函数完成连接参数的赋值。

1. **构造函数（Reader方法）**

用途：实例化Reader类。

语法：public Reader(String readerName, IPort commPort)

参数：

readerName：读写器名称，字符串。

commPort：通讯接口定义。

备注：

该构造函数不用与配置文件配合使用。readerName定义同上。构造函数完成连接参数的赋值。

通信接口IPort为抽象类，Rs232Port、Rs485Port和TcpClientPort继承自Iport类，完成几个接口的定义，这几个类位于NetAPI命名空间中（JAVA：位于JavaAPI包中）

构造函数如下：

public Rs232Port(String portName, BaudRate baudRate)

public Rs485Port(String portName, BaudRate baudRate, byte[] addresses)

public TcpClientPort(String IP, Int32 port)

参数：

portName：串口名，字符串。例如：COM1

baudRate：波特率，BaudRate枚举。定义如下：

public enum BaudRate

{

R9600,//9600bps

R19200, //19200bps

R38400, //38400bps

R115200, //115200bps(出厂默认)

R230400//230400bps

}

Addresses：RS485地址列表。RS485地址1-255

IP：读写器IP。例如：192.168.1.100

port：读写器网络端口。例如：9090。

1. **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，指定端口在构造函数中已经指定。

语法：public ConnectResponse Connect()

返回值：

连接回应，ConnectResponse定义如下

public class ConnectResponse

{

public bool IsSucessed = false;//是否连接成功

public ErrInfo ErrorInfo;

}

参数：

ErrorInfo：错误消息类，ErrInfo类；如果连接错误时，该类返回错误信息。

1. **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：public void Disconnect()

1. **发送消息（Send方法）**

用途：发送消息。

语法：public bool Send(IHostMessage msg)

public bool Send(IHostMessage msg, int timeout)

参数：

msg：待发送消息具体类，IHostMessage类型。

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：

* 1. 需要响应的消息，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。
  2. 不需要响应的消息，发送的字节数等于要发送的字节数则返回成功。
  3. 没有超时参数的方法，默认超时为1000毫秒。

备注：如何设置消息是否需要响应，请参见第3章节。

1. **发送消息数据（Send方法）**

用途：发送字节数组消息数据。

语法：public bool Send(byte[] msgBuff)

参数：

msgBuff：待发送的数据，字节数组。

返回值：

消息是否成功发送，布尔型。

备注：

该方法发送数据后不会等待数据响应。注意：该方法不会进行任何的数据封装和校验，只是简单的将msgBuff内容发送出去，可用于测试或自定义消息发送。

1. **盘存消息通知（OnInventoryReceived事件）**

用途：接收读写器消息。在本API中，MsgTagInventory类和MsgTagRead类的循环读取会引发此事件。

语法：

C#：public event InventoryReceivedHandle OnInventoryReceived;

C#：public delegate void InventoryReceivedHandle(string readerName, RxdTagData tagData);

JAVA：public EventHandler OnInventoryReceived;

参数：

readerName：读写器名称，在构造函数中指定的名称。

RxdTagData：标签数据，定义如下

public class RxdTagData

{

private byte Antenna;

private byte[] EPC;

private byte[] TID;

private byte[] User;

private byte[] Reserved;

private byte RSSI;

}

Antenna：天线号，0表示没返回此字段

EPC：标签EPC数据，空表示没返回此数据

TID：标签TID数据，空表示没返回此数据

User：标签User数据，空表示没返回此数据

Reserved：标签保留区数据，空表示没返回此数据

RSSI：回波强度，0表示没返回此字段

备注：

C#通过实现InventoryReceivedHandle委托，回调读写器盘存消息。

JAVA中的OnInventoryReceived回调两个信息，一个是Reader类，一个是RxdTagData类；JAVA调用步骤：

1）Utils.Event msgNotify = new Utils.Event(this, "Reader\_OnInventoryReceived");//定义回调方法，this指回调方法所在的对象，视具体实现而定；Reader\_OnInventoryReceived为回调方法名称；Event在Utils包中。

2）reader.OnInventoryReceived.addEvent(msgNotify);//加载事件回调

3）void Reader\_OnInventoryReceived(Reader sender, RxdTagData msg){}//事件回调的实现

1. **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：

C#：public static event ApiExceptionHandle OnApiException;

C#：public delegate void ApiExceptionHandle(String senderName, ErrInfo e);

JAVA：public static EventHandler OnApiException;

参数：

senderName：读写器名称，在构造函数中指定的名称。

ErrInfo：错误消息类，ErrInfo类。

备注：

C#通过实现ApiExceptionHandle委托，回调异常消息。

JAVA中的OnApiException回调两个信息，一个是senderName字符串，一个是ErrInfo类；JAVA调用步骤：

1）Utils.Event apiErrNotify = new Utils.Event(this, "Reader\_OnApiException");//定义回调方法，this指回调方法所在的对象，视具体实现而定；Reader\_OnApiException为回调方法名称；Event在Utils包中。

2）Reader.OnApiException.addEvent(apiErrNotify);//加载事件回调

3）void Reader\_OnApiException (String senderName, ErrInfo e){}//事件回调的实现

绝大多数API异常都经此进行通知。

1. **网络断开通知（OnBrokenNetwork事件）**

用途：接收读写器断网通知。此事件只支持TCP口通讯情况。

语法：

C#：public event BrokenNetworkHandle OnBrokenNetwork;

C#：public delegate void BrokenNetworkHandle(string senderName, ErrInfo e);

JAVA：public static EventHandler OnBrokenNetwork;

参数：

senderName：读写器名称，在构造函数中指定的名称。

ErrInfo：错误消息类，ErrInfo类。

备注：

C#通过实现BrokenNetworkHandle委托，回调读写器消息。

JAVA中的OnBrokenNetwork回调两个信息，一个是senderName字符串，一个是ErrInfo类；JAVA调用步骤：

1）Utils.Event brokenNetworkNotify = new Utils.Event(this, "Reader\_OnBrokenNetwork");//定义回调方法，this指回调方法所在的对象，视具体实现而定；Reader\_OnBrokenNetwork为回调方法名称；Event在Utils包中。

2）Reader.OnBrokenNetwork.addEvent(brokenNetworkNotify);//加载事件回调

3）void Reader\_OnBrokenNetwork (String senderName, ErrInfo e){}//事件回调的实现

1. **读写器名称属性（ReaderName成员）**

用途：获取读写器名称。

语法：public String ReaderName = "Device1";

备注：

字符串类型。同构造函数中readerName参数。默认为"Device1"，如有多读写器连接，请保证此名称不要重复，在构造函数中指定后，此处自动更改；强烈建议读写器名称以构造函数的定义为准。

1. **是否连接标志属性（IsConnected成员）**

用途：获取是否连接标志。

语法：

C#：public bool IsConnected{get;}

JAVA：public Boolean getIsConnected();

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。

1. **读写器型号属性（ModelNumber成员）**

用途：获取读写器型号。

语法：

C#：public string ModelNumber { get; }

JAVA：public String getModelNumber();

备注：

字符串类型。初始值为空，只有在读写器成功连接后该成员才会有具体值。

1. **标签数据是否返回天线信息属性（IsEnableAntenna成员）**

用途：获取标签数据是否返回天线信息标志。

语法：

C#：public bool IsEnableAntenna { get; }

JAVA：public boolean getIsEnableAntenna();

备注：

详见3.25节。

1. **标签数据是否返回RSSI信息属性（IsEnableRSSI成员）**

用途：获取标签数据是否返回回波强度信息标志。

语法：

C#：public bool IsEnableRSSI { get; }。

JAVA：public boolean getIsEnableRSSI();

备注：

详见3.25节。

1. **通信端口类型属性（CommPort成员）**

用途：获取通信端口类型。

语法：

C#：public IPort CommPort { get; set; }

JAVA：public IPort getCommPort();

JAVA：public void setCommPort(IPort value);

备注：

强烈建议在构造函数或配置文件中指定，此处仅用于查询。

1. **RS485读写器属性（RS485Items成员）**

用途：RS485通讯接口，该成员为**RS485Item**类数组，表示每个地址的接口。

语法：public RS485Item[] RS485Items;

说明：RS485Item类实现以下接口，用法类似Reader类相应成员或方法。

C#：

public byte Address；// RS485地址

public FrequencyArea UhfBand = FrequencyArea.Unknown;

public string ModelNumber;

public bool IsEnableRSSI;

public bool IsEnableAntenna;

public bool Send(IHostMessage msg);

public bool Send(IHostMessage msg, Int32 timeout);

public bool Send(byte[] msgBuff);

JAVA：

public byte getAddress()；// 获取RS485地址

public byte setAddress()；// 设置RS485地址

public FrequencyArea UhfBand = FrequencyArea.Unknown;

public String getModelNumber();

public boolean getIsEnableRSSI();

public boolean getIsEnableAntenna();

public boolean Send(IHostMessage msg);

public boolean Send(IHostMessage msg, int timeout);

public boolean Send(byte[] msgBuff);

备注：

该成员在Reader的构造函数中实例化后自动生成，如不是RS485通讯，则该成员为空。该接口的标签盘存数据接收与Reader类相同，见Reader类的OnInventoryReceived事件。

## 2.3 消息指令抽象类（AbstractHostMessage抽象类）

C#：命名空间： NetAPI.Protocol.VRP

JAVA：包位置： JavaAPI.Protocol.VRP

上位机主动发送的消息基类。所有上位机发送的消息类均继承自该类。上位机消息为上位机与读写器通讯的基础，所有读写器的动作都是通过上位机消息指令控制的。

上位机消息包括：

1）上位机发送的需要读写器应答的消息；

2）上位机发送消息后，不需要等待读写器应答的消息；

3）上位机应答读写器的消息。

上位机消息类关系图如下图所示。



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：

C#：bool IsReturn { get; set; }

JAVA：boolean getIsReturn();

JAVA：void setIsReturn(boolean value);

备注：

本API中所有消息都已经默认设置好了该成员，除非特殊需要，一般不要更改该成员值。

1. **获取消息返回状态（Status成员）**

用途：定义接口，该接口用于实现获取消息返回状态，MsgStatus枚举类型。

语法：

C#：MsgStatus Status { get; set; }

JAVA：MsgStatus getStatus();

JAVA：void setStatus(MsgStatus value);

返回值：

MsgStatus枚举类型状态，包括成功、失败和超时状态，定义如下

public enum MsgStatus

{

Timeout = -1,//超时

Success = 0,//成功

Failed = 1//失败

}

备注：

1. 读写器在指定的超时时间内没有返回，则返回时该状态被设置为超时；
2. 当IsReturn设置为true时，读写器返回成功的消息，则返回时该状态被设置为成功；读写器返回失败的消息，则返回时该状态被设置为失败；
3. 当IsReturn设置为false时，上位机发送消息成功，则返回时该状态被设置为成功；上位机发送消息失败，则返回时该状态被设置为失败。
4. **返回消息内容（ReceivedMessage成员）**

用途：定义接口，该接口用于实现获取消息返回内容，消息内容根据子类信息而定，该内容将在子类中说明。

语法：

C#：public ReceivedInfo ReceivedMessage { get; }

JAVA：public ReceivedInfo getReceivedMessage();

备注：

由于各子类消息实现的功能不同，所以此处返回的ReceivedInfo内容无法确定，默认为null。

1. **返回消息错误内容（ErrorInfo成员）**

用途：定义接口，该接口用于实现获取消息返回的错误信息，当消息返回状态Status为失败时，可以通过该成员获取错误信息。

语法：

C#：ErrInfo ErrorInfo { get; }

JAVA：public ErrInfo getErrorInfo();

返回值：

ErrInfo类型，包括消息内容和错误代码，定义如下

public class ErrInfo

{

public String ErrMsg;// 消息内容

public String ErrCode;// 错误代码

}

备注：

使用前可先判断一下，是否null

## 2.4 配置文件（Sysit.xml）

配置文件为可选使用项，文件名：“DeviceCfg”，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：

1.读写器名称。（读写器名称不能重复）

2.连接端口类型。

3.连接字符串。

配置文件格式如下所示：

<?xml version="1.0" encoding="utf-8"?>

<Devices>

<LogLevel>INFO</LogLevel>

<Device Name="Device1">

<Port Type="TcpClient">192.168.1.100:9090</Port>

</Device>

<Device Name="Device2">

<Port Type="RS232">COM1,115200</Port>

</Device>

<Device Name="Device3">

<Port Type="RS485">COM2,115200</Port>

<Address>1</Address>

<Address>2</Address>

</Device>

</Devices>

其中“Devices”为根节点。

“Device”节点为读写器配置节点。“Name”属性为读写器名称，名称为自定义的字符串，名称应保持唯一。

“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。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。

TCP/IP协议客户端连接字符串例子：“192.168.1.100:5840”。其中“192.168.1.100”为IP，“9090”为端口号，中间以半角的冒号隔开。

“Address”节点为RS485地址，取值范围1~255。

“LogLevel”选项为设置日志级别选项，日志级别包含“Debug”、“Info” 、“War”、“Error”和“Fatal”，日志级别依次增高，默认为“Info”表示Debug级别的日志不做记录，Info以上的记入日志。

# Message接口列表

消息接口分为两大类，一类是读写器消息接口类（IReaserMessage），另外一类是上位机消息接口类（IHostMessage）。本章节以下所提到的消息，除非特殊说明，否则全部继承自IHostMessage接口，有关消息接口说明，参见2.3节说明。

## 3.1固件程序升级准备（MsgFirmwareUpgradePreparation类）

用途：该指令的目是上位机告诉读写器要升级了，让读写器准备好，当上位机接收到读写器返回的成功信息后，上位机再通过固件程序升级指令（见3.2节）发送升级数据给读写器。

1. 构造函数

C#：public MsgFirmwareUpgradePreparation()

JAVA：public MsgFirmwareUpgradePreparation()

参数：无。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 3.2固件程序在线升级（MsgFirmwareOnlineUpgrade类）

用途：固件程序在线升级。

1. 构造函数

C#：public MsgFirmwareOnlineUpgrade(UInt32 fAddr,byte[] data)

JAVA：public MsgFirmwareOnlineUpgrade(int fAddr,byte[] data)

参数：

fAddr：上位机发送升级数据帧地址从 1开始，即为 0x00000001。

data：帧数据内容，不大于200字节。

1. 指令应答（ReceivedMessage）

1）帧地址

用途：获取帧地址，如为0xFFFFFFFF则表示升级成功完成。

C#：public UInt32 FrameAddress { get; }

JAVA：public int getFrameAddress();

1. 指令说明

升级流程为：

1）上位机按 16 进制顺序读取“\*.bin ”的内容，并依照不大于200 字节为一帧拆分并下发给读写器，最后一帧可以小于200 个字节；

2）帧地址为 0时，帧数据内容表示总的帧数；

3）上位机发送升级数据帧地址从1开始，即为 0x00000001；

4）上位机发送帧地址为 0xffffffff （数据为任意四个字节）的确认帧，代表所有数据下发完成；

5）读写器必须有回应，2秒无回应需超时重发。

## 3.3查询模块软硬件版本（MsgReaderVersionQuery类）

用途：查询模块产品型号及软硬件版本。

1. 构造函数

C#：public MsgReaderVersionQuery()

JAVA：public MsgReaderVersionQuery()

参数：无。

1. 指令应答（ReceivedMessage）

1）产品型号

用途：获取产品型号。

C#：public string ModelNumber { get; }

JAVA：public String getModelNumber();

2）硬件版本

用途：获取硬件版本。

C#：public string HardwareVersion{ get; }

JAVA：public String getHardwareVersion();

3）软件版本

用途：获取软件版本。

C#：public string SoftwareVersion { get; }

JAVA：public String getSoftwareVersion();

## 3.4设置/查询使用的天线（MsgAntennaConfig类）

用途：设置/查询使用的天线。

1. 构造函数

C#：public MsgAntennaConfig()//查询

public MsgAntennaConfig(AntennaStatus[] antennaStatusAry)//设置

JAVA：public MsgAntennaConfig()//查询

public MsgAntennaConfig(AntennaStatus[] antennaStatusAry) //设置

参数：

antennaStatusAry：天线使用列表。

1. 指令应答（ReceivedMessage）

1）使用的天线

用途：获取使用的天线。

C#：public AntennaStatus[] AntennaStatusAry { get; }

JAVA：public AntennaStatus[] getAntennaStatusAry();

返回值：天线列表，AntennaStatus类型数组。

备注：

AntennaStatus类定义：

public class AntennaStatus

{

public byte AntennaNO;//天线号，取值范围1-64

public bool IsEnable;//是否启用

}

说明：

请根据实际情况启用已连接的天线，禁止启用未连接的天线，以免造成读写器射频模块损坏。

## 3.5查询 RFID 工作状态（MsgRfidStatusQuery类）

用途：查询RFID工作状态。

1. 构造函数

C#：public MsgRfidStatusQuery ()//查询

JAVA：public MsgRfidStatusQuery ()//查询

参数：无。

1. 指令应答（ReceivedMessage）

1）状态

用途：获取读写器状态。

C#：public RfidStatus Status { get; }

JAVA：public RfidStatus getStatus();

返回值：状态枚举，RfidStatus定义如下

public enum RfidStatus

{

Idle,// 00 表示读写器空闲

Busy // 01 表示读写器正在对标签操作

}。

2）空口协议

用途：获取空口协议。注意：仅当状态为读卡时，此部分才有意义。

C#：public AirProtocol Protocol { get; }

JAVA：public AirProtocol getProtocol();

返回值：空口协议枚举，AirProtocol定义如下

public enum AirProtocol

{

ISO18000\_6C,

ISO18000\_6B,

ISO18000\_6B\_6C

}

3）工作频段

用途：获取工作频段。

C#：public FrequencyArea UhfBand { get; }

JAVA：public FrequencyArea getUhfBand();

返回值：工作频段枚举，FrequencyArea定义如下

public enum FrequencyArea

{

CN,

FCC,

EU,

Unknown = -1

}

备注：

CN (920MHz~925MHz)

FCC(902MHz~928MHz)

EU(865MHz~868MHz)

4）天线启用及功率状态

用途：获取天线启用及功率状态。

C#：public AntennaPowerStatus[] Antennas { get; }

JAVA：public AntennaPowerStatus[] getAntennas();

返回值：天线启用及功率状态，AntennaPowerStatus定义如下

public class AntennaPowerStatus

{

public byte AntennaNO;//天线号

public bool IsEnable;//是否启用

public byte PowerValue; //功率值

}

备注：

功率值单位为dBm

## 3.6设置/查询RS232波特率（MsgRs232BaudRateConfig类）

用途：设置/查询应用端的 RS232 通信接口波特率。

1. 构造函数

C#：public MsgRs232BaudRateConfig()//查询

public MsgRs232BaudRateConfig(BaudRate baudRate) //设置

JAVA：public MsgRs232BaudRateConfig()//查询

public MsgRs232BaudRateConfig(BaudRate baudRate) //设置

参数：

baudRate：波特率，BaudRate定义见（2.2节）

1. 指令应答（ReceivedMessage）

1）波特率

用途：获取波特率。

C#：public BaudRate RS232BaudRate { get; }

JAVA：public BaudRate getRS232BaudRate();

返回值：波特率，定义同上。

备注：

如果当前连接为串口连接，那么更改波特率后，需重新以更改后的波特率进行连接。

## 3.7设置/查询RS485波特率（MsgRs485BaudRateConfig类）

用途：设置/查询应用端的 RS485 通信接口波特率。

1. 构造函数

C#：public MsgRs485BaudRateConfig()//查询

public MsgRs485BaudRateConfig(BaudRate baudRate) //设置

JAVA：public MsgRs485BaudRateConfig()//查询

public MsgRs485BaudRateConfig(BaudRate baudRate) //设置

参数：

baudRate：波特率，BaudRate定义见（2.2节）

1. 指令应答（ReceivedMessage）

1）波特率

用途：获取波特率。

C#：public BaudRate RS485BaudRate { get; }

JAVA: public BaudRate getRS485BaudRate();

返回值：波特率，定义同上。

备注：

如果当前连接为串口连接，那么更改波特率后，需重新以更改后的波特率进行连接。

## 3.8设置/查询本机 RS-485 地址（MsgRs485AddressConfig类）

用途：设置/查询应用端的 RS485 通信接口波特率。

1. 构造函数

C#：public MsgRs485AddressConfig ()//查询

public MsgRs485AddressConfig (byte address) //设置

JAVA：public MsgRs485AddressConfig ()//查询

public MsgRs485AddressConfig (byte address) //设置

参数：

address：地址，1-255。

1. 指令应答（ReceivedMessage）

1）波特率

用途：获取地址。

C#：public byte Address{ get; }

JAVA：public byte getAddress{ get; }

返回值：地址，1-255。

## 3.9查询 GPIO 接口定义（MsgGpioDefinitionQuery类）

用途：设置/查询应用端的 RS232 通信接口波特率。

1. 构造函数

C#：public MsgGpioDefinitionQuery()//查询

JAVA：public MsgGpioDefinitionQuery()//查询

参数：无。

1. 指令应答（ReceivedMessage）

1）GPIO接口定义

用途：获取GPIO接口定义。

C#：public GpioInfo[] GpioInformation { get; }

JAVA：public GpioInfo[] getGpioInformation();

返回值：GPIO接口定义信息数组，其中Gpio类定义如下。

public class GpioInfo

{

public byte PortNO;

public GpioDefinition Definition;

}

public enum GpioDefinition:byte

{

GPI,//IO输入

GPO,//IO输出

Disabled = 0xff//禁用

}

## 3.10设置/查询 GPIO 输入触发的标签操作（MsgGpiTriggerConfig类）

用途：设置/查询 GPIO 输入触发的标签操作。

1. 构造函数

C#：public MsgGpiTriggerConfig(byte portNO) //查询

public MsgGpiTriggerConfig(GpiTriggerParameter param)//设置

JAVA：public MsgGpiTriggerConfig(byte portNO) //查询

public MsgGpiTriggerConfig(GpiTriggerParameter param)//设置

参数：

portNO：Gpi端口号。

Param：触发参数。定义如下

public class GpiTriggerParameter

{

public byte PortNO;// 指定需要配置的端口号，1表示 GPIO1，2表示GPIO2

public GpiTriggerCondition TriggerCondition;//触发条件

public GpiStopCondition StopCondition;//停止条件

public ushort DelayTime;//延时时间，指定延时作为停止条件时的延时时间

public IHostMessage TriggerMsg;//触发消息，空指令用null

}

public enum GpiTriggerCondition

{

Disabled,// 表示关闭当前端口的GPIO触发功能；

HighLevel,// 表示 GPIO高电平触发；

LowLevel//表示 GPIO低电平触发；

}

public enum GpiStopCondition

{

StateInversion,//采用GPIO状态反转作为停止条件（与触发条件相反，高电平对应低电平）；

Delayed//采用延时作为停止条件；

}

1. 指令应答（ReceivedMessage）

1）GPIO触发参数

用途：获取GPIO触发参数。

C#：public GpiTriggerParameter TriggerParameter{ get; }

JAVA：public GpiTriggerParameter TriggerParameter();

返回值：GPIO触发参数，定义同上。

1. 指令说明

读写器的 GPIO在基本的输入输出功能基础上实现了 GPIO事件触发机制，采用GPIO事件触发机制可以很方便的与其他设备实现联动。

GPIO事件触发机制的基本原理是：将GPIO输入状态与相关读写器的相关操作指令进行绑定， GPIO不同的输入状态可触发响应的读写器操作。每个GPIO口只能绑定一对读写器指令，每个 GPIO口可单独绑定。

此功能一般用于GPIO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。

## 3.11设置GPIO输出电平（MsgGpoConfig类）

用途：设置/查询 GPIO 输出电平。

1. 构造函数

C#：public MsgGpoConfig(GpioLevelParameter param) //设置

JAVA：public MsgGpoConfig(GpioLevelParameter param) //设置

参数：

portNO：GPO端口号。读写器定义了2个对外输出口(GPIO3, GPIO4)，指定需要配置的GPIO端口号，3表示 GPIO3，4表示 GPIO4。

Param：GPO电平参数。定义如下

public class GpioLevelParameter

{

public byte PortNO;// GPO端口号

public GpioLevel Level;//电平

}

public enum GpioLevel

{

Low,//低电平

High//高电平

}

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 3.12查询GPIO输入电平（MsgGpiQuery类）

用途：设置/查询 GPIO 输出电平。读写器定义了2个对外输出口(GPIO1, GPIO2)，指定需要配置的GPIO端口号，1表示 GPIO1，2表示 GPIO2。

1. 构造函数

C#：public MsgGpiQuery(byte portNO) //查询

JAVA：public MsgGpiQuery(byte portNO) //查询

参数：

portNO：GPO端口号。

Param：GPO电平参数。定义如下

public class GpioLevelParameter

{

public byte PortNO; // GPO端口号

public GpioLevel Level;//电平

}

public enum GpioLevel

{

Low, //低电平

High//高电平

}

1. 指令应答（ReceivedMessage）

1）GPI电平参数

用途：获取GPI电平参数。

C#：public GpioLevelParameter GpiLevel { get; }

JAVA：public GpioLevelParameter getGpiLevel();

返回值：GPI电平参数，定义同上。

## 3.13设置/查询重复标签过滤时间（MsgFilteringTimeConfig类）

用途：设置/查询重复标签过滤时间。标签过滤时间只有在设置了数据上传方式为时间过滤时，才生效。

1. 构造函数

C#：public MsgFilteringTimeConfig() //查询

public MsgFilteringTimeConfig(ushort time) //设置

JAVA：public MsgFilteringTimeConfig()//查询

public MsgFilteringTimeConfig(int time) //设置

参数：

time：过滤时间,单位100ms。取值范围：0-65535；用于指定重复数据过滤的时间间隔 ，为0表示不过滤。

1. 指令应答（ReceivedMessage）

1）过滤时间

用途：获取过滤时间。

C#：public ushort Time { get; }

JAVA：public int getTime();

返回值：过滤时间，定义同上。

1. 指令说明

重复标签过滤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。

## 3.14设置/查询间隔读标签时间（MsgIntervalTimeConfig类）

用途：设置/查询间隔读标签时间。

1. 构造函数

C#：public MsgIntervalTimeConfig() //查询

public MsgIntervalTimeConfig(ushort readTime, ushort stopTime)//设置

JAVA：public MsgIntervalTimeConfig()

public MsgIntervalTimeConfig(int readTime, int stopTime)

参数：

readTime：读标签时间,单位ms，最大为65535ms

stopTime：停止读时间,单位ms，最大为65535ms

1. 指令应答（ReceivedMessage）

1）读标签时间

用途：读标签时间,单位ms。

C#：public ushort ReadTime { get; }

JAVA：public int getReadTime();

返回值：读标签时间,单位ms。

2）停止读时间

用途：获取停止读时间,单位ms。

C#：public ushort StopTime { get; }

public int getStopTime();

JAVA:

返回值：停止读时间,单位ms。

1. 指令说明

设置 /查询间隔读标签功能用于读写器间歇读标签，具体为读标签一段时间（读标签时间长度），然后停止一段时间（停止读标签时间长度）不读标签，然后再读标签一段时间，然后停止读标签一段时间，周而复始。

## 3.15设置/查询信号强度（RSSI）过滤阀值（MsgRssiThresholdConfig类）

用途：设置/查询标签信号强度（RSSI）过滤阈值。

1. 构造函数

C#：public MsgRssiThresholdConfig() //查询

public MsgRssiThresholdConfig(byte rssi) //设置

JAVA：public MsgRssiThresholdConfig() //查询

public MsgRssiThresholdConfig(byte rssi) //设置

参数：

rssi：标签信号强度（RSSI）过滤阈值

1. 指令应答（ReceivedMessage）

1）标签信号强度（RSSI）过滤阈值

用途：获取标签信号强度（RSSI）过滤阈值。

C#：public byte RSSI { get; }

JAVA：public byte getRSSI();

返回值：标签信号强度（RSSI）过滤阈值。

1. 指令说明

RSSI过滤阈值最大为65535；设置为 0时，表示不启用RSSI过滤功能。当值大于0时，读到的标签RSSI值小于设置值的情况下，读写器将不上传该标签。透传时，RSSI值无效。

## 3.16设置/查询工作频段（MsgUhfBandConfig类）

用途：设置/查询工作频段。

1. 构造函数

C#：public MsgUhfBandConfig() //查询

public MsgUhfBandConfig(FrequencyArea band)//设置

JAVA：public MsgUhfBandConfig() //查询

public MsgUhfBandConfig(FrequencyArea band)//设置

参数：

band：工作频段，FrequencyArea定义如下

public enum FrequencyArea

{

CN,// (920MHz~925MHz)

FCC,// (902MHz~928MHz)

EU,// (865MHz~868MHz)

Unknown = -1

}

1. 指令应答（ReceivedMessage）

1）工作频段

用途：获取工作频段。

C#：public FrequencyArea UhfBand{ get; }

JAVA：public FrequencyArea getUhfBand();

返回值：工作频段，定义同上。

## 3.17设置/查询端口功率值（MsgPowerConfig类）

用途：设置/查询端口功率值。

1. 构造函数

C#：public MsgPowerConfig() //查询

public MsgPowerConfig(byte[] powers) //设置

JAVA：public MsgPowerConfig() //查询

public MsgPowerConfig(byte[] powers) //设置

参数：

powers：端口功率值，按天线顺序，不能跳过；功率范围0-36dBm；最大支持64个天线

1. 指令应答（ReceivedMessage）

1）端口功率值

用途：获取端口功率值。

C#：public AntennaPower[] Powers { get; }

JAVA：public AntennaPower[] getPowers();

返回值：端口功率值，定义如下。

public class AntennaPower

{

public byte AntennaNO;

public byte PowerValue;

}

1. 指令说明

功率值与读卡距离有关，功率值越大，读卡距离越远，读卡距离远也容易产生误读；反之则反。

## 3.18设置/查询支持的空口协议（MsgAirProtocolConfig类）

用途：设置/查询支持的空口协议。

1. 构造函数

C#：public MsgAirProtocolConfig() //查询

public MsgAirProtocolConfig(AirProtocol param)//设置

JAVA：public MsgAirProtocolConfig() //查询

public MsgAirProtocolConfig(AirProtocol param)//设置

参数：

param：空口协议，定义如下

public enum AirProtocol

{

ISO18000\_6C,// 表示支持 6C 协议标签，为默认设置参数

ISO18000\_6B,// 表示支持 6B 协议标签

ISO18000\_6B\_6C// 表示支持 6B+6C 协议标签

}

1. 指令应答（ReceivedMessage）

1）空口协议

用途：获取支持的空口协议。

C#：public AirProtocol Protocol { get; }

JAVA：public AirProtocol getProtocol();

返回值：空口协议，定义同上。

## 3.19设置/查询工作频率（MsgFrequencyConfig类）

用途：设置/查询工作频率。

1. 构造函数

C#：public MsgFrequencyConfig() //查询

public MsgFrequencyConfig(FrequencyTable param) //设置

JAVA：public MsgFrequencyConfig() //查询

public MsgFrequencyConfig(FrequencyTable param) //设置

参数：

param：工作频率参数，定义如下

public class FrequencyTable

{

public bool IsAutoSet;//频率自动设置

public byte[] FreqTable;//频率列表

}

1. 指令应答（ReceivedMessage）

1）工作频率

用途：获取工作频率。

C#：public FrequencyTable FreqInfo { get; }

JAVA：public FrequencyTable getFreqInfo();

返回值：工作频率，定义同上。

备注：

频率列表和频率点存在对应关系，频率列表定义如下

public class UhfBandTable

{

public String[] List;// 频点列表

public enum Name

{

CN\_920\_925,

FCC\_902\_928,

EU\_865\_868

}

public UhfBandTable(Name ftName)

{

}

}

当查询到工作频率列表FreqInfo时，其类型为FrequencyTable，该类中的FreqTable值为UhfBandTable.List的索引值，例如FreqTable数组中某个元素的值为0x03，当前频段为CN，那么0x03对应的频点为UhfBandTable.List[3]，其频点为921.375MHz。

1. 指令说明
2. 频率自动设置，为true时，读写器在RF频段内自动选择频点；为false时，读写器不自动选择频点，只能根据指定的频率列表使用频点。
3. 频率列表，用于指定非自动频率选择模式下的读写器工作频点，频率列表为当前工作频段下的信道号列表，如在国标920~925MHz频段下需要指定920.625、922.375、924.375MHz三个频点，则频率列表应为{0,7,15}。频点数最少为1，最大为50。

国标频段(CN)频率列表和频率点对应关系

0x00 -- 920.625M

0x01 -- 920.875M

0x02 -- 921.125M

0x03 -- 921.375M

0x04 -- 921.625M

0x05 -- 921.875M

0x06 -- 922.125M

0x07 -- 922.375M

0x08 -- 922.625M

0x09 -- 922.875M

0x0A -- 923.125M

0x0B -- 923.375M

0x0C -- 923.625M

0x0D -- 923.875M

0x0E -- 924.125M

0x0F -- 924.375M

北美频段(FCC)频率列表和频率点对应关系

0x00 -- 902.75M

0x01 -- 903.25M

0x02 -- 903.75M

0x03 -- 904.25M

0x04 -- 904.75M

0x05 -- 905.25M

0x06 -- 905.75M

0x07 -- 906.25M

0x08 -- 906.75M

0x09 -- 907.25M

0x0a -- 907.75M

0x0b -- 908.25M

0x0c -- 908.75M

0x0d -- 909.25M

0x0e -- 909.75M

0x0f -- 910.25M

0x10 -- 910.75M

0x11 -- 911.25M

0x12 -- 911.75M

0x13 -- 912.25M

0x14 -- 912.75M

0x15 -- 913.25M

0x16 -- 913.75M

0x17 -- 914.25M

0x18 -- 914.75M

0x19 -- 915.25M

0x1a -- 915.75M

0x1b -- 916.25M

0x1c -- 916.75M

0x1d -- 917.25M

0x1e -- 917.75M

0x1f -- 918.25M

0x20 -- 918.75M

0x21 -- 919.25M

0x22 -- 919.75M

0x23 -- 920.25M

0x24 -- 920.75M

0x25 -- 921.25M

0x26 -- 921.75M

0x27 -- 922.25M

0x28 -- 922.75M

0x29 -- 923.25M

0x2a -- 923.75M

0x2b -- 924.25M

0x2c -- 924.75M

0x2d -- 925.25M

0x2e -- 925.75M

0x2f -- 926.25M

0x30 -- 926.75M

0x31 -- 927.25M

## 3.20恢复出厂设置（MsgResetToFactoryDefault类）

用途：恢复出厂默认设置。

1. 构造函数

C#：public MsgResetToFactoryDefault()

JAVA：public MsgResetToFactoryDefault()

参数：无。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 3.21设置/查询空闲等待时间（MsgIdleTimeConfig类）

用途：设置/查询空闲等待时间。

1. 构造函数

C#：public MsgIdleTimeConfig() //查询

public MsgIdleTimeConfig(ushort time) //设置

JAVA：public MsgIdleTimeConfig() //查询

public MsgIdleTimeConfig(int time) //设置

参数：

time：空闲等待时间，单位ms。

1. 指令应答（ReceivedMessage）

1）空闲等待时间

用途：获取空闲等待时间。

C#：public ushort Time { get; }

JAVA：public ushort getTime();

返回值：空闲等待时间，单位ms。

## 3.22设置/查询IP地址（MsgIpAddressConfig类）

用途：设置/查询应用端网口的IP地址。

1. 构造函数

C#：public MsgIpAddressConfig() //查询

public MsgIpAddressConfig(string ip,string subnet,string gate) //设置

JAVA：public MsgIpAddressConfig()//查询

public MsgIpAddressConfig(String ip,String subnet,String gate) //设置

参数：

ip：IP地址，如192.168.1.100。

subnet：子网掩码，如255.255.255.0。

gate：网关，如192.168.1.1。

1. 指令应答（ReceivedMessage）

1）IP地址

用途：获取IP地址。

C#：public string IP { get; }

JAVA：public String getIP();

返回值：IP地址，如192.168.1.100；出厂默认为192.168.1.100。

2）子网掩码

用途：获取子网掩码。

C#：public string Subnet { get; }

JAVA：public String getSubnet();

返回值：子网掩码，如255.255.255.0。

3）网关

用途：获取网关。

C#：public string Gateway { get; }

JAVA：public String getGateway();

返回值：网关，如192.168.1.1。

## 3.23设置/查询服务端、客户端模式（MsgTcpModeConfig类）

用途：设置/查询应用端的网口服务端/客户端模式。

1. 构造函数

C#：public MsgTcpModeConfig()//查询

public MsgTcpModeConfig(ReaderTcpMode mode,ushort serverPortNO) //设置

JAVA：public MsgTcpModeConfig()//查询

public MsgTcpModeConfig(ReaderTcpMode mode,int serverPortNO) //设置

参数：

mode：读写器端TCP连接模式，定义如下。

public enum ReaderTcpMode

{

server,//服务器模式（出厂默认配置为服务器模式）

client//客户端模式

}

serverPortNO：读写器服务端口，建议在1024~65535之间。默认为9090，此参数只在读写器配置为服务器模式下才有意义。

1. 指令应答（ReceivedMessage）

1）读写器端TCP连接模式

用途：获取读写器端TCP连接模式。

C#：public ReaderTcpMode Mode { get; }

JAVA：public ReaderTcpMode getMode();

返回值：读写器端TCP连接模式，定义同上。

2）读写器服务端口

用途：获取读写器服务端口。

C#：public ushort ServerPortNO { get; }

JAVA：public int getServerPortNO();

返回值：读写器服务端口，建议在1024~65535之间。默认为9090，此参数只在读写器配置为服务器模式下才有意义。

1. 指令说明

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。

## 3.24查询MAC地址（MsgMacConfig类）

用途：设置/查询应用端网口的MAC地址。

1. 构造函数

C#：public MsgMacConfig()//查询

JAVA：public MsgMacConfig()//查询

1. 指令应答（ReceivedMessage）

1）读写器端MAC地址

用途：获取读写器端MAC地址。

C#：public byte[] MAC { get; }//应用端网口的MAC地址，6字节。

public string getStringMAC();

JAVA：public byte[] getMAC();//应用端网口的MAC地址，6字节。

public String getStringMAC();

返回值：读写器端MAC地址，定义同上。

## 3.25设置/查询读 6C 标签数据帧包含字段（Msg6CTagFieldConfig类）

用途：设置/查询读6C标签数据包含字段。

1. 构造函数

C#：public Msg6CTagFieldConfig()//查询

public Msg6CTagFieldConfig(bool isEnableAntenna, bool isEnableRSSI) //设置

JAVA：public string getStringMAC()//查询

public Msg6CTagFieldConfig(boolean isEnableAntenna, boolean isEnableRSSI) //设置

参数：

isEnableAntenna：是否包含天线字段。

isEnableRSSI：是否包含RSSI字段。

1. 指令应答（ReceivedMessage）

1）是否包含天线字段

用途：获取是否包含天线字段。

C#：public bool IsEnableAntenna

JAVA：public boolean getIsEnableAntenna();

返回值：是否包含天线字段。false：表示数据帧不含接收标签的天线端口；true：表示数据帧含接收标签的天线端口；

注意：天线端口字段只适用于读标签数据

2）是否包含RSSI字段

用途：获取是否包含RSSI字段。

C#：public bool IsEnableRSSI

JAVA：public boolean getIsEnableRSSI();

返回值：是否包含RSSI字段。false：表示数据帧不含接收标签的RSSI值；true：表示数据帧含接收标签的RSSI值。

## 3.26读6C标签EPC区（MsgTagInventory类）

用途：读6C标签EPC，标签盘存。

1. 构造函数

C#：public MsgTagInventory()//扫描标签EPC，不限时，不限次

public MsgTagInventory(InventoryTagParameter param) //扫描标签EPC

JAVA：public MsgTagInventory()//扫描标签EPC，不限时，不限次

public MsgTagInventory(InventoryTagParameter param) //扫描标签EPC

参数：

param：扫描标签参数，定义如下

public class InventoryTagParameter

{

public ushort ReadCount;//读取次数

public ushort TotalReadTime;//读取总时间

public ushort TagFilteringTime;//重复标签过滤时间

public ushort ReadTime;// 间歇读取时间，与StopTime搭配使用

public ushort StopTime;// 间歇停止时间，与ReadTime搭配使用

public ushort IdleTime;// 空闲等待时间

public TagParameter SelectTagParam = null;// 标签选择参数

}

public class TagParameter

{

public MemoryBank MemoryBank;//标签数据区域

public UInt32 Ptr;//标签起始地址

public byte[] TagData;//标签数据

}

MemoryBank：标签数据区域。注意：用于选择标签时不要指定保留区，用于写标签时，不要指定TID区。

Ptr：标签起始地址，因不同厂家标签的各个数据区大小均可能不同，因此起始地址的范围也不同 ,最大不能超过被操作标签的匹配数据区的最大值。具体数值参考相应标签的数据手册。如果不能确定标签数据区最大值，在操作前可以尝试读取标签的操作Bank ，来确认匹配数据区的最大值。注意：做为选择标签参数起始地址以bit为单位；做为读、写标签参数起始地址以字为单位。

public enum MemoryBank

{

ReservedMemory,// 保留区

EPCMemory,// EPC数据区

TIDMemory,// TID数据区

UserMemory// 用户数据区

}

ReadCount：设置需要读取的次数，达到设定次数后停止读卡。该值为 0表示不限次数。

TotalReadTime：设置需要读取的时间，单位 10ms，达到设定时间后停止读卡。该值为0表示不限时间。

1. 指令应答（ReceivedMessage）

1）标签数据

用途：获取标签数据，标签数据只包含EPC，另外可能会有天线号和RSSI数据，取决于Msg6CTagFieldConfig配置。

C#：public RxdTagData TagData {get;}

JAVA：public RxdTagData getTagData();

返回值：标签数据。RxdTagData说明见OnInventoryReceived节。

备注：

仅当读取次数等于1时，标签数据直接返回于本消息的应答中。循环读卡只有读到标签数据才返回，不返回失败信息，并且标签数据返回到Reader类的OnInventoryReceived事件中。

1. 指令说明

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。

## 3.27通用读6C标签（MsgTagRead类）

用途：通用读6C标签。

1. 构造函数

C#：public MsgTagRead() //扫描标签TID，不限时，不限次

public MsgTagRead(ReadTagParameter param) //扫描标签

JAVA：public MsgTagRead() //扫描标签TID，不限时，不限次

public MsgTagRead(ReadTagParameter param) //扫描标签

参数：

param：读标签参数，继承自InventoryTagParameter，注意，与InventoryTagParameter具有相同的参数，参见InventoryTagParameter说明。ReadTagParameter定义如下

public class ReadTagParameter : InventoryTagParameter

{

public bool IsLoop = false;

public byte[] AccessPassword = new byte[4];

public bool IsReturnEPC = false;

public bool IsReturnTID = false;

public UInt32 UserPtr;

public byte UserLen;

public bool IsReturnReserved = false;

new public ushort ReadCount = 1;// 注意：此处默认读取1次

new public ushort ReadTime = 100;

}

IsLoop：false,单次读取模式，读写器尽在各个使能的天线上进行一轮读卡操作便结束读卡操作并自动进入空闲状态。true,表示连续读取模式，读写器一直进行读卡操作直到读写器收到停止指令后结束读卡。

AccessPassword：标签访问密码，4字节，默认全0。

IsReturnEPC：指定是否返回EPC数据。false表示不需要返回，true表示需要返回。

IsReturnTID：指定是否返回TID数据。false表示不需要返回，true表示需要返回。说明：读标签TID码时，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 ～112bits范围内的TID码长度。

UserPtr：用户区首地址。

UserLen：用户区长度，单位为字，0~127，0代表不返回User数据。

IsReturnReserved：指定是否返回保留区数据，保留区数据内容：销毁密码 2个字 +访问密码 2个字

ReadCount：同上。

ReadTime：同上。

1. 指令应答（ReceivedMessage）

1）标签数据

用途：获取标签数据，可能会有天线号和RSSI数据，取决于Msg6CTagFieldConfig配置。

C#：public RxdTagData TagData {get;}

JAVA：public RxdTagData getTagData();

返回值：标签数据。RxdTagData说明见OnInventoryReceived节。

备注：

仅当读取次数等于1时，标签数据直接返回于本消息的应答中。循环读卡标签数据返回到Reader类的OnInventoryReceived事件中。

## 3.28写6C标签（MsgTagWrite类）

用途：写6C标签。

1. 构造函数

C#：public MsgTagWrite(WriteTagParameter param) //

JAVA：public MsgTagWrite(WriteTagParameter param)

参数：

param：写标签参数，定义如下

public class WriteTagParameter

{

public bool IsLoop = false;

public byte[] AccessPassword = new byte[4];

public TagParameter SelectTagParam = null;

public TagParameter[] WriteDataAry = null;

}

IsLoop：false，表示单次写入模式，只针对指定的标签进行写操作，写操作完成后结束写操作。true，表示连续写入模式，针对符合匹配规则的多个标签连续进行写操作，直到收到关功放指令才停止写操作。

AccessPassword：标签访问密码，4字节，默认全0。

SelectTagParam：选择标签参数，指定要写入的标签。定义同上。

WriteDataAry：指定写入标签内容，可同时指定EPC，User和保留区，但不要重复指定，例如不能同时指定两个EPC区。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

1. 指令说明

EPC 长度：0~31 个字，以为单位；以字为单位；

USER 长度：0~127个字 ，以字为单位；

保留区长度： 0~4个字 ，以字为单位；

保留区数据内容：销毁密码 2个字 + 访问密码 2个字

## 3.29 6C标签锁操作（MsgTagLock类）

用途：标签锁操作。

1. 构造函数

C#：public MsgTagLock(LockTagParameter param) //

JAVA：public MsgTagLock(LockTagParameter param) //

参数：

param：锁标签参数，定义如下

public class LockTagParameter

{

public LockBank LockBank;

public LockType LockType;

public TagParameter SelectTagParam = null;

public byte[] AccessPassword = new byte[4];

}

public enum LockBank

{

All, //0，所有数据区

TID, //1，TID区

EPC, //2，EPC区

User, //3，用户数据区

AccessPassword,//4，访问密码区

KillPassword //5，销毁密码区

}

public enum LockType

{

Lock, //0，锁定

UnLock, //1，解锁

PermanentLock, //2，永久锁定（保留）

PermanentUnLock//3，永久解锁（保留）

}

AccessPassword：标签访问密码，4字节。

SelectTagParam：选择标签参数，指定要操作的标签。定义同上。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

1. 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“所有数据区”的解锁操作时不会执行TID区的解锁。

## 3.30销毁 6C 标签（MsgTagKill类）

用途：标签销毁操作。

1. 构造函数

C#：public MsgTagKill(KillTagParameter param)//

JAVA：public MsgTagKill(KillTagParameter param)//

参数：

param：销毁标签参数，定义如下

public class KillTagParameter

{

public byte[] KillPassword = new byte[4];// 销毁密码，4字节

public TagParameter SelectTagParam = null;

}

KillPassword：标签销毁密码，4字节。

SelectTagParam：选择标签参数，指定要操作的标签。定义同上。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

1. 指令说明

销毁标签为不可逆操作，请注意数据安全。

## 3.31设置/查询UTC时间（MsgUtcConfig类）

用途：设置/查询UTC时间。

1. 构造函数

C#：public MsgUtcConfig() //查询

public MsgUtcConfig(DateTime time)//设置

JAVA：public MsgUtcConfig() //查询

public MsgUtcConfig(GregorianCalendar time)//设置

参数：

time：UTC时间

1. 指令应答（ReceivedMessage）

1）UTC时间

用途：获取读写器UTC时间。

C#：public DateTime UTC { get; }

JAVA：public GregorianCalendar getUTC();

返回值：UTC时间。

1. 指令说明

协调世界时，又称世界统一时间、世界标准时间、国际协调时间。由于英文（CUT）和法文（TUC）的缩写不同，作为妥协，简称UTC。

协调世界时是以原子时秒长为基础，在时刻上尽量接近于世界时的一种时间计量系统。中国大陆采用ISO 8601-1988的《数据元和交换格式信息交换日期和时间表示法》（GB/T 7408-1994）称之为国际协调时间，代替原来的GB/T 7408-1994；中国台湾采用CNS 7648的《资料元及交换格式–资讯交换–日期及时间的表示法》，称之为世界统一时间。

中国大陆、中国香港、中国澳门、中国台湾、蒙古国、新加坡、马来西亚、菲律宾、西澳大利亚州的时间与UTC的时差均为+8，也就是UTC+8。

本系统UTC时间以1970-1-1为起始时间计算。

## 3.32开功放（内部测试使用）（MsgPowerOn类）

用途：开功放。

1. 构造函数

C#：public MsgPowerOn(AntennaStatus[] antennaStatusAry)

JAVA：public MsgPowerOn(AntennaStatus[] antennaStatusAry)

参数：

antennaStatusAry：天线开启状态，定义见“设置/查询使用的天线”。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空

## 3.33关功放（MsgPowerOff类）

用途：关功放。停止读卡。

1. 构造函数

C#：public MsgPowerOff()//关功放，默认不等待返回

JAVA：public MsgPowerOff()//关功放，默认不等待返回

参数：无。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空

## 3.34应用端固件程序升级准备（MsgAppFirmwareUpgradePreparation类）

用途：该指令的目是上位机告诉读写器应用端要升级了，让读写器准备好，当上位机接收到读写器返回的成功信息后，上位机再通过应用端固件程序升级指令（见3.35）发送升级数据给读写器。

1. 构造函数

C#：public MsgAppFirmwareUpgradePreparation()

JAVA：public MsgAppFirmwareUpgradePreparation()

参数：无。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 3.35应用端固件在线升级（MsgAppFirmwareOnlineUpgrade类）

用途：应用端固件在线升级。

1. 构造函数

C#：public MsgAppFirmwareOnlineUpgrade(UInt32 fAddr, byte[] data)

JAVA：public MsgAppFirmwareOnlineUpgrade(int fAddr, byte[] data)

参数：

fAddr：上位机发送升级数据帧地址从 1开始，即为 0x00000001。

data：帧数据内容，不大于200字节。

1. 指令应答（ReceivedMessage）

1）帧地址

用途：获取帧地址，如为0xFFFFFFFF则表示升级成功完成。

C#：public UInt32 FrameAddress { get; }

JAVA：public int getFrameAddress();

1. 指令说明

升级流程同固件程序在线升级（见3.2）。

## 3.36查询应用处理器软件及硬件版本（MsgCpuVersionQuery类）

用途：查询模块产品型号及软硬件版本。

1. 构造函数

C#：public MsgCpuVersionQuery()

JAVA：public MsgCpuVersionQuery()

参数：无。

1. 指令应答（ReceivedMessage）

1）产品型号

用途：获取产品型号。

C#：public string ModelNumber { get; }

JAVA：public string getModelNumber();

2）硬件版本

用途：获取硬件版本。

C#：public string HardwareVersion{ get; }

JAVA：public string getHardwareVersion();

3）软件版本

用途：获取软件版本。

C#：public string SoftwareVersion { get; }

JAVA：public string getSoftwareVersion();

## 3.37选择6C标签指令（MsgTagSelect类）

用途：选择操作的6C标签。

1. 构造函数

C#：public MsgTagSelect(TagParameter param)

JAVA：public MsgTagSelect(TagParameter param)

参数：

param：选择标签参数，定义同上。注意：起始地址以bit为单位；匹配区域不能选保留区。

1. 指令应答（ReceivedMessage）

该消息没有可进一步解析的数据，ReceivedMessage为空。

## 3.38查询读写器能力指令（MsgReaderCapabilityQuery类）

用途：查询读写器能力。

1. 构造函数

C#：public MsgReaderCapabilityQuery()

JAVA：public MsgReaderCapabilityQuery()

参数：

无。

1. 指令应答（ReceivedMessage）

1）产品型号

用途：获取产品型号。

C#：public string ModelNumber { get; }

JAVA：public string getModelNumber();

2）读写器支持的天线数量

用途：获取读写器支持的天线数量。

C#：public byte AntennaCount { get; }

JAVA：public byte getAntennaCount();

3）读写器支持的最小功率

用途：获取读写器支持的最小功率。

C#：public byte MinPowerValue { get; }

JAVA：public byte getMinPowerValue();

4）读写器支持的最大功率

用途：获取读写器支持的最大功率。

C#：public byte MaxPowerValue { get; }

JAVA：public byte getMaxPowerValue();

# 常见应用

本章节对RFID读写器在日常使用中常见应用功能做进一步详细说明。

## 4.1读写器连接

读写器连接有两种方式，一种是直接指定连接参数，一种是通过配置文件获取连接参数进行连接。如要多读写器连接，则建立多个Reader类实例即可。注意：读写器名称不能重复。



读写器连接步骤：

1. 构造读写器类Reader类，指定参数，实例化读写器类
2. 调用读写器连接方法（Connect方法）连接读写器
3. 连接成功后加载监听事件，监听盘存消息、错误消息、断网消息
4. 断开连接之前卸载监听事件
5. 调用读写器断开连接方法（Disconnect方法）断开读写器连接。

**C#代码：**

using NetAPI;

using NetAPI.Entities;

using NetAPI.Protocol.VRP;

namespace RFIDTest

{

public class ReaderConnTest

{

static void Main()

{

ReaderConnTest test = new ReaderConnTest();

test.Conn();

// ...

test.DisConn();

}

Reader reader;

public ReaderConnTest()

{

Reader.OnApiException += Reader\_OnApiException;

}

private void Reader\_OnApiException(string senderName, ErrInfo e)

{

}

private void Reader\_OnBrokenNetwork(string senderName, ErrInfo e)

{

}

private void Reader\_OnInventoryReceived(string readerName, RxdTagData tagData)

{

}

public void Conn()

{

//reader = new Reader("Reader1");//通过配置文件指定

//reader = new Reader("Reader1", new Rs232Port("COM1,115200"));//串口连接

reader = new Reader("Reader1", new TcpClientPort("192.168.1.100:9090"));//网口连接

ConnectResponse resp = reader.Connect();

if (resp.IsSucessed)

{

reader.OnInventoryReceived += Reader\_OnInventoryReceived;//加载盘存监听事件

reader.OnBrokenNetwork += Reader\_OnBrokenNetwork;//加载网络断开事件，仅支持网口

}

}

public void DisConn()

{

if(reader != null && reader.IsConnected)

{

//卸载事件

reader.OnInventoryReceived -= Reader\_OnInventoryReceived;

reader.OnBrokenNetwork -= Reader\_OnBrokenNetwork;

//断开连接

reader.Disconnect();

}

reader = null;

}

~ReaderConnTest()

{

Reader.OnApiException -= Reader\_OnApiException;

}

}

}

**JAVA代码：**

import JavaAPI.Rs232Port;

import JavaAPI.TcpClientPort;

import JavaAPI.Core.ErrInfo;

import JavaAPI.Core.Util;

import JavaAPI.Entities.\*;

import JavaAPI.Protocol.VRP.\*;

public class ReaderTest

{

public static void main(String[] args)

{

ReaderTest test = new ReaderTest();

if (test.Conn())

{

//...

test.DisConn();

}

}

Reader reader;

byte[] lastEpc;

private Utils.Event inventoryReceivedEvent = new Utils.Event(this, "Reader\_OnInventoryReceived");

private Utils.Event brokenNetworkEvent = new Utils.Event(this, "Reader\_OnBrokenNetwork");

private Utils.Event apiExceptionEvent = new Utils.Event(this, "Reader\_OnApiException");

public ReaderTest()

{

Reader.OnApiException.addEvent(apiExceptionEvent);

Reader.OnBrokenNetwork.addEvent(brokenNetworkEvent);// 加载网络断开事件，仅支持网口

}

private void Reader\_OnApiException(String senderName, ErrInfo e)

{

}

private void Reader\_OnBrokenNetwork(String senderName, ErrInfo e)

{

}

private void Reader\_OnInventoryReceived(Reader reader, RxdTagData tagData)

{

}

public boolean Conn()

{

// reader = new Reader("Reader1");//通过配置文件指定

// reader = new Reader("Reader1", new Rs232Port("COM1,115200"));// 串口连接

reader = new Reader("Reader1", new TcpClientPort("192.168.1.100:9090"));//

// 网口连接

ConnectResponse resp = reader.Connect();

if (resp.IsSucessed)

{

reader.OnInventoryReceived.addEvent(inventoryReceivedEvent);// 加载盘存监听事件

System.out.println("读写器连接成功");

}

else

{

System.out.println("读写器连接失败：" + resp.ErrorInfo.getErrMsg());

}

return reader.getIsConnected();

}

public void DisConn()

{

if (reader != null && reader.getIsConnected())

{

// 卸载事件

reader.OnInventoryReceived.removeEvent(inventoryReceivedEvent);

// 断开连接

reader.Disconnect();

}

reader = null;

System.out.println("读写器断开连接");

}

protected void finalize() throws java.lang.Throwable

{

super.finalize();

if (Reader.OnApiException != null)

Reader.OnApiException.removeEvent(apiExceptionEvent);

if (Reader.OnBrokenNetwork != null)

Reader.OnBrokenNetwork.removeEvent(brokenNetworkEvent);

}

}

## 4.2标签扫描

标签扫描即为快速连续识别标签EPC、TID、User等数据区，指令详见3.27章节。在这里我们主要介绍常见的扫描EPC和扫描TID。MsgTagInventory类专门用来扫描标签EPC的，也叫盘存，MsgTagRead类用做通用读标签的功能，该类的无参数的构造函数，提供了扫描标签TID的快捷方式。

标签数据从OnInventoryReceived监听事件中返回，该事件处于主线程外的另一个线程，在处理是请注意。标签扫描首先需在3.1中建立连接成功，并加载监听事件后，方可进行。步骤如下：

1. 发送扫描指令
2. 读写器将触发消息返回事件
3. 处理接收到的标签信息
4. 重复执行第2）3）步骤
5. 发送停止扫描指令

**C#代码（在4.1基础上做以下修改）：**

static void Main()

{

ReaderConnTest test = new ReaderConnTest();

test.Conn();

test.ScanEPC();

test.Stop();

test.ScanTID();

test.Stop();

test.DisConn();

}

public void ScanEPC()

{

MsgTagInventory msg = new MsgTagInventory();

if(reader.Send(msg))

{

//成功

}

else

{

//失败

}

}

public void ScanTID()

{

MsgTagRead msg = new MsgTagRead();//循环扫描TID快捷指令

if (reader.Send(msg))

{

//成功

}

else

{

//失败

}

}

public void Stop()

{

reader.Send(new MsgPowerOff());

}

**JAVA代码（在4.1基础上做以下修改）：**

public static void main(String[] args)

{

ReaderTest test = new ReaderTest();

if (test.Conn())

{

test.ScanEPC();

try

{

Thread.sleep(1000 \* 3); // 休眠3秒

}

catch (Exception e)

{

System.out.println("Got an exception!");

}

test.Stop();

test.ScanTID();

try

{

Thread.sleep(1000 \* 3); // 休眠3秒

}

catch (Exception e)

{

System.out.println("Got an exception!");

}

test.Stop();

test.DisConn();

}

}

private void Reader\_OnInventoryReceived(Reader reader, RxdTagData tagData)

{

if (tagData.getEPC() != null && tagData.getEPC().length > 0)

{

lastEpc = tagData.getEPC();

System.out.println(Util.ConvertByteArrayToHexWordString(lastEpc));

}

if (tagData.getTID() != null && tagData.getTID().length > 0) System.out.println(Util.ConvertByteArrayToHexWordString(tagData.getTID()));

}

public void ScanEPC()

{

MsgTagInventory msg = new MsgTagInventory();

if (reader.Send(msg))

{

System.out.println("扫描EPC");

}

else

{

System.out.println("扫描失败：" + msg.getErrorInfo().getErrMsg());

}

}

public void ScanTID()

{

MsgTagRead msg = new MsgTagRead();// 循环扫描TID快捷指令

if (reader.Send(msg))

{

System.out.println("扫描TID");

}

else

{

System.out.println("扫描失败：" + msg.getErrorInfo().getErrMsg());

}

}

public void Stop()

{

reader.Send(new MsgPowerOff());

System.out.println("扫描停止");

}

## 4.3标签操作

标签操作即为标签的读、写、锁、销毁。本章节主要讲述写标签操作，其他操作请自行参考本手册或示例代码。

标签在写操作时，不能同时进行标签扫描，也就是说，如果读写器正在扫描标签，请先停止扫描，在进行写操作。标签写操作步骤如下：

1. 指定写操作标签参数（标签选择）
2. 指定写标签的区域和写入的数据
3. 指定标签访问密码，默认为全0
4. 发送指令进行写操作
5. 根据写操作返回的状态，可进行下一步操作

**C#代码（在4.1基础上做以下修改）：**

private void WriteTag()

{

byte[] targetEPC = new byte[] { 0x12, 0x34, 0x56, 0x78 };//标签EPC，根据实际情况而定

TagParameter selectParam = new TagParameter();

selectParam.MemoryBank = MemoryBank.EPCMemory;

selectParam.Ptr = 0x00; //选择标签参数，起始地址单位为bit

selectParam.TagData = targetEPC;

WriteTagParameter wp = new WriteTagParameter();

wp.SelectTagParam = selectParam;

wp.AccessPassword = new byte[4];//密码

wp.WriteDataAry = new TagParameter[2];//可同时写几个区，

int p = 0;

// 写标签EPC区

{

TagParameter tp = new TagParameter();

tp.MemoryBank = MemoryBank.EPCMemory;

//写标签参数，起始地址单位为字

tp.Ptr = 0x00;

tp.TagData = new byte[] { 0x11, 0x22, 0x33, 0x44 };

wp.WriteDataAry[p] = tp;

p++;

}

// 写标签User区

{

TagParameter tp = new TagParameter();

tp.MemoryBank = MemoryBank.UserMemory;

tp.Ptr = 0x00;

tp.TagData = new byte[] { 0xaa, 0xbb, 0xcc, 0xdd };

wp.WriteDataAry[p] = tp;

}

MsgTagWrite msg = new MsgTagWrite(wp);

if (reader.Send(msg))

{

string str = "写入成功";

}

else

{

string str = "写入失败：" + msg.ErrorInfo.ErrMsg;

}

}

**JAVA代码（在4.1基础上做以下修改）：**

public static void main(String[] args)

{

ReaderTest test = new ReaderTest();

if (test.Conn())

{

test.ScanEPC();

try

{

Thread.sleep(1000 \* 3); // 休眠3秒

}

catch (Exception e)

{

System.out.println("Got an exception!");

}

test.Stop();

test.ScanTID();

try

{

Thread.sleep(1000 \* 3); // 休眠3秒

}

catch (Exception e)

{

System.out.println("Got an exception!");

}

test.Stop();

try

{

Thread.sleep(1000 \* 1); // 休眠1秒

}

catch (Exception e)

{

System.out.println("Got an exception!");

}

test.WriteTag();

test.DisConn();

}

}

public void WriteTag()

{

byte[] targetEPC = lastEpc;// 标签EPC，根据实际情况而定

TagParameter selectParam = new TagParameter();

selectParam.MemoryBank = MemoryBank.EPCMemory;

// 选择标签参数，起始地址单位为bit

selectParam.Ptr = 0x00;

selectParam.TagData = targetEPC;

WriteTagParameter wp = new WriteTagParameter();

wp.SelectTagParam = selectParam;

wp.AccessPassword = new byte[4];// 密码

wp.WriteDataAry = new TagParameter[2];// 可同时写几个区，

int p = 0;

// 写标签EPC区

{

TagParameter tp = new TagParameter();

tp.MemoryBank = MemoryBank.EPCMemory;

// 写标签参数，起始地址单位为字

tp.Ptr = 0x00;

tp.TagData = new byte[] { 0x11, 0x22, 0x33, 0x44 };

wp.WriteDataAry[p] = tp;

p++;

}

// 写标签User区

{

TagParameter tp = new TagParameter();

tp.MemoryBank = MemoryBank.UserMemory;

tp.Ptr = 0x00;

tp.TagData = new byte[] { 0x44, 0x33, 0x22, 0x11 };

wp.WriteDataAry[p] = tp;

}

MsgTagWrite msg = new MsgTagWrite(wp);

if (reader.Send(msg))

{

String str = "写入成功";

System.out.println(str);

}

else

{

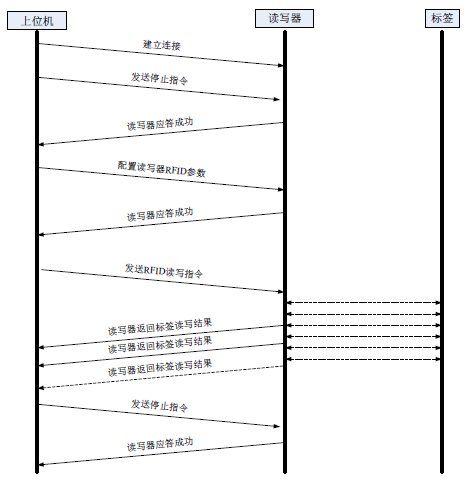
String str = "写入失败：" + msg.getErrorInfo().getErrMsg();

System.out.println(str);

}

}

# 附录A：上位机控制读写器读写标签流程示例



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。

# 附录B：错误代码描述

错误代码描述：

F0 表示指令字错误

F1 表示指令执行失败

F2 表示CRC错误

F3 表示数据帧格式、帧参数错误

F4 表示存储区超限

F5 表示存储区被锁

F6 表示功率不够

F7 表示操作标签不支持

F8 表示未检测到标签或标签无回应

F9 表示操作标签失败

0x12 天线错误

0x14 写flash错误