# 控制器接口文档说明(汉字标签)Java

## 一、简介

系统软件接口采用SDK 包的方式提供给客户进行二次业务层的开发，目前提供支持C# SDK DLL接口(csharp\_tag\_cn.dll)和调用DEMO，以及Java的jar包，供用户二次开发的终端软件在PC/平板终端上运行。

本文档为Java版本。

## 使用环境

### 1、编译环境

Java版本：Java11

### 2、控制器支持类型

TCP/IP 接口的网络通讯控制器.

每个控制器建立三个连接：

4001 端口用于通道一的标签通信；

4002 端口用于通道二的标签通信；

1. 端口用于控制控制器屏幕，声音和灯带。

### 3、调用方式

Java工程中添加jar包为库依赖，即可调用jar包中的函数

## 接口函数列表

### 1、开启监听\_客户端

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | new Api(string host, int port); | | | | |
| **功能说明** | **启动监听服务**，需控制器配置为客户端模式，可监听多台控制器 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| host | | string | 是 | 服务器IP地址 |
| port | | int | 是 | 监听端口 |
| **C#调用** | | | | | |
| **示例** | **调用请求** | Api api = new Api("192.168.1.68", 4001); | | | |
| **返回** |  | | | |

### 2、标签亮灭灯（betKeyLedDisplay）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | bool setKeyLedDisplay(string ipAddress, int port, int tagID, LightMode extLed, int timeOut, int retryTimes) | | | |
| **功能说明** | 设置标签灯光颜色, 将指定的标签按键LED设置为指定的颜色 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 通道端口 |
| tagID | int | 是 | 当前地址 |
| extLed | LightMode | 是 | 标签显示的颜色(见标签和塔灯显示颜色列表对应LightMode枚举值) |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //IP地址为192.168.1.108的控制器，端口为4001的1号标签亮绿灯  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bResult = api.setKeyLedDisplay(ipAddress, port, 1，LightMode.Green,100,3); | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | |

### 3、分配地址（batchSetIdEnter）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | bool batchSetIdEnter(string ipAddress, int port, int start\_id) | | | | |
| **功能说明** | 分配标签地址  **【警告：慎用！** **此方法会让所选通道的所有标签进入地址分配模式，之前的地址可能会被覆盖，且无法恢复】**   * **所有标签都会亮红灯**，需要**依次手动拍灯**完成地址分配； * **拍灯后的标签屏幕上会显示所分配的地址，并亮绿灯；** * **完成后需调用 BatchSetIdExit 退出分配模式；** * **建议：在调用前确认当前通道是否需要重新分配标签地址！** | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 待分配标签所在通道端口 |
| start\_id | | int | 是 | 标签起始地址 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //给IP地址为192.168.1.108的控制器的通道一的标签，从1开始分配地址  byte bNupStart = 1;//起始地址  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bRes = api.batchSetIdEnter(ipAddress, port, bNupStart); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 4、退出分配地址（batchSetIdExit）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | bool batchSetIdExit(string ipAddress, int port) | | | | |
| **功能说明** | 标签地址分配完成后，需调用此接口来退出分配地址状态 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 待退出分配标签所在通道端口 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //给IP地址为192.168.1.108的控制器的一通道的标签，退出分配地址状态  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bResult = api.batchSetIdExit(ipAddress, port); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 5、标签自检（UpdateCheck）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | J1939\_MESSAGE UpdateCheck(string ipAddress, int port, int tagID, int timeOut, int retryTimes) | | | |
| **功能说明** | 标签自检 检查目标id的升级状态，在指定端口上，向目标ID发送更新检查请求，并等待响应。 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 自检的通道端口 |
| tagID | int | 是 | 目标地址 |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签自检，等待时间10ms  J1939\_MESSAGE outMsg = api.UpdateCheck("192.168.1.108", 4001, 1, 10, 3); System.out.println("UpdateCheck: " + outMsg); | | |
| **返回** | if (bResult)  {  AppendLog($"标签id[{i}]在线");  //打印硬件版本号  AppendLog($"HV: {outMsg.Mxe.Data[0]}.{outMsg.Mxe.Data[1]}.{outMsg.Mxe.Data[2]}");  //打印软件版本号  AppendLog($"SV: {outMsg.Mxe.Data[3]}.{outMsg.Mxe.Data[4]}.{outMsg.Mxe.Data[5]}");  //检查程序运行状态  if (outMsg.Mxe.Data[6] == 0xaa && outMsg.Mxe.Data[7] == 0xaa)  {  AppendLog("标签自检 程序在app中运行");  }  if (outMsg.Mxe.Data[6] == 0x55 && outMsg.Mxe.Data[7] == 0x55)  {  AppendLog("标签自检 程序在boot中运行");  }  }  else  {  AppendLog($"标签自检 标签id[{i}]异常");  } | | |

### 6、修改标签地址（setCanId）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | bool setCanId(string ipAddress, int port, int can\_id, int set\_can\_id, int timeOut, int retryTimes) | | | | |
| **功能说明** | 修改标签地址 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 自检的通道端口 |
| can\_id | | int | 是 | 当前地址 |
| set\_can\_id | | int | 是 | 需修改为的目标地址 |
| timeOut | | int | 是 | 等待时间(毫秒) |
| retryTimes | | int | 是 | 重试次数 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签地址改为8  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bResult = api.setCanId(ipAddress, port, 1, 8, 100, 3); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 7、更新库存（setStockCN）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | bool setStockCN(string ipAddress, int port, int target\_id, int stock\_num, int timeOut, int retryTimes) | | | | |
| **功能说明** | 更新库存数量,通过指定的端口和目标ID，更新库存数量，并等待目标设备响应 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 自检的通道端口 |
| target\_id | | int | 是 | 当前地址 |
| stock\_num | | int | 是 | 要设置的新库存数量 |
| timeOut | | int | 是 | 等待时间(毫秒) |
| retryTimes | | int | 是 | 重试次数 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签库存设置为80  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bResult = api.setStockCN(ipAddress, port, 1, 80, 100, 3); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 8、获取库存（getStockCN）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | int getStockCN(string ipAddress, int port, int target\_id, int timeOut, int retryTimes) | | | | |
| **功能说明** | 获取库存数量, 通过指定的端口和目标ID，获取库存数量 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 自检的通道端口 |
| target\_id | | int | 是 | 当前地址 |
| timeOut | | int | 是 | 等待时间(毫秒) |
| retryTimes | | int | 是 | 重试次数 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //获取控制器为192.168.1.108，一通道的1号标签的库存数量  string ipAddress = “192.168.1.108”;  int port = 4001;  int nStockNum = api.getStockCN(ipAddress, port, 1, 100, 3); | | | |
| **返回** | nStockNum | | | |

### 9、出库（setOutStock）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | bool setOutStock(string ipAddress, int port, int target\_id, int stock\_num, LightMode lightMode, int timeOut, int retryTimes) | | | | |
| **功能说明** | 设置出库数量, 根据指定的端口、目标ID、库存数量和灯光颜色，设置出库数量。 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 自检的通道端口 |
| target\_id | | int | 是 | 当前地址 |
| stock\_num | | int | 是 | 出库数量 |
| lightMode | | LightMode | 是 | 标签显示的颜色(见标签和塔灯显示颜色列表) |
| timeOut | | int | 是 | 等待时间(毫秒) |
| retryTimes | | int | 是 | 重试次数 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签出库10，并亮绿灯  string ipAddress = “192.168.1.108”;  int port = 4001;  int nStockNum = 10;  bool bResult = api.setOutStock(ipAddress, port, 1, nStockNum，LightMode.Green, 100, 3); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 10、设置显示信息（setDisplay）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | bool setDisplay(string ipAddress, int port, int target\_id, DisplayInfo info, byte[] binaryBytes, int timeOut, int retryTimes) | | | |
| **功能说明** | 设置显示信息 局部更新背景, 通过指定的端口和目标ID，设置显示信息，并发送包含图像字符串的J1939消息 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 自检的通道端口 |
| target\_id | int | 是 | 当前地址 |
| info | DisplayInfo | 是 | 显示信息结构体 |
| binaryBytes | byte[] | 是 | 图片字符串 |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签设置信息显示  string ipAddress = “192.168.1.108”;  int port = 4001;  string filePath = "C:/Users/lskj/TcpListenerTest/data2.txt";  byte[] binaryBytes = api.readHexFromFile(filePath);  DisplayInfo info = DisplayInfo.builder().build();  info.setStartX(32);  info.setStartY(32);  info.setPictureHeight(24);  info.setPictureWidth(100);  byte[] binaryBytes = ImageToBinaryBytes(image);  bool bResult = api.setDisplay(ipAddress, port, 1, info，binaryBytes, 1000, 3); | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | |

### 11、设置背景图片（setBackground）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | bool setBackground(string ipAddress, int port, int target\_id, byte[] binaryBytes, int timeOut, int retryTimes) | | | |
| **功能说明** | 设置背景图片, 将指定的图片字符串通过给定的端口和目标 ID 发送到设备，以设置背景图片。 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 自检的通道端口 |
| target\_id | int | 是 | 当前地址 |
| binaryBytes | byte[] | 是 | 图片字符串，图片像素296\*128 |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //给控制器为192.168.1.108，一通道的1号标签设置模板  string ipAddress = “192.168.1.108”;  int port = 4001;  string filePath = "C:/Users/TcpListenerTest/data.txt"; byte[] binaryBytes = api.readHexFromFile(filePath);  bool bResult = api.setBackground(ipAddress, port, 1, binaryBytes, 10000, 3); | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | |

### 12、显示背景（displayBackground）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | bool displayBackground(string ipAddress, int port, int target\_id, int timeOut, int retryTimes) | | | |
| **功能说明** | 显示背景, 通过指定的端口和目标ID，发送显示背景指令，并等待响应。 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 自检的通道端口 |
| target\_id | int | 是 | 当前地址 |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //让控制器为192.168.1.108，一通道的1号标签显示模板  string ipAddress = “192.168.1.108”;  int port = 4001;  bool bResult = api.displayBackground(ipAddress, port, 1, 100, 3); | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | |

### 13、获取灯色（getLedColor）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **接口名** | int getLedColor(string ipAddress, int port, int target\_id, int timeOut, int retryTimes) | | | | |
| **功能说明** | 获取标签灯的显示状态 | | | | |
| **参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 控制器IP地址 |
| port | | int | 是 | 自检的通道端口 |
| target\_id | | int | 是 | 当前地址 |
| timeOut | | int | 是 | 等待时间(毫秒) |
| retryTimes | | int | 是 | 重试次数 |
| **返回结果** | bResult | | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | | |
| **示例** | **调用请求** | //获取控制器为192.168.1.108，一通道的1号地址的灯色  string ipAddress = “192.168.1.108”;  int port = 4001;  int ledColor = api.getLedColor(ipAddress, port, 1, 100, 3); | | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | | |

### 14、设置数字显示（setNumDisplayInfo）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **接口名** | bool setNumDisplayInfo(string ipAddress, int port, int target\_id, NumDisplayInfo outStockInfo,NumDisplayInfo stockNumInfo, int timeOut, int retryTimes) | | | |
| **功能说明** | 该函数用于设置指定库存和出库的数字显示信息，包括位置及大小。 | | | |
| **参数** | **参数名** | **数据类型** | **是否必填** | **备注** |
| ipAddress | string | 是 | 控制器IP地址 |
| port | int | 是 | 自检的通道端口 |
| target\_id | int | 是 | 当前地址 |
| outStockInfo | NumDisplayInfo | 是 | 库存数量显示位置的X坐标 0-298像素 |
| stockNumInfo | NumDisplayInfo | 是 | 库存数量显示位置的Y坐标 0-128像素 |
| timeOut | int | 是 | 等待时间(毫秒) |
| retryTimes | int | 是 | 重试次数 |
| **返回结果** | bResult | bool | 是 | 结果（true-成功，false-失败） |
| **C#调用** | | | | |
| **示例** | **调用请求** | //为控制器为192.168.1.108，一通道的1号标签设置库存和出库的位置及显示大小  string ipAddress = “192.168.1.108”;  int port = 4001;  NumDisplayInfo stockNumInfo = NumDisplayInfo.builder().x((char)70).y((byte)65).size((byte)2).build(); NumDisplayInfo outStockInfo = NumDisplayInfo.builder().x((char)115).y((byte)85).size((byte)4).build();  bool bResult = api.setNumDisplayInfo(ipAddress, port, 1, outStockInfo, stockNumInfo, 100, 3); | | |
| **返回** | if(bResult)  { //成功 }  else  { //失败 } | | |

## 四、反馈事件

**本系统采用观察者模式（发布-订阅机制），通过注册ApiListener监听器来接收服务器端事件通知（连接、断开、短按、长按、出库等）。客户端只需实现所需的方法即可响应对应事件**

**添加引用接口**：import com.example.ApiListener;

**添加监听器代码参考：**

NettyServerHandler.addListener(new ApiListener() {  
 // 请参考下方各方法的具体说明  
});

**ApiListener接口定义：**

public interface ApiListener {

    default void onConnect(int id, String ipAddress, int port, int type, int num, int seq){}

   default void onDisconnect(int clientId, String ipAddress, int port) {}

    default void onPushButton(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum){}

    default void onSetOutStock(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum){}

    default void onPushLongButton(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum){}

}

**各事件说明：**

### 1、控制器连接（onConnect）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **回调事件** | onConnect | | | | |
| **功能说明** | 连接成功时触发 | | | | |
| **返回参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 连接上的控制器IP |
| port | | int | 是 | 连接端口 |
| **C#调用** | | | | | |
| **示例** | **调用** | @Override public void onConnect(int id, String ipAddress, int port, int type, int num, int seq) {  System.*out*.println("onConnect: ip=" + ipAddress + "; port=" + port); } | | | |

### 2、释放连接事件（onDisconnect）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **回调事件** | onDisconnect | | | | |
| **功能说明** | 控制器连接断开时触发 | | | | |
| **返回参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| clientId | | int | 是 | 离线的控制器连接ID |
| ipAddress | | string | 是 | 离线的控制器IP |
| port | | int | 是 | 离线的控制器端口 |
| **C#调用** | | | | | |
| **示例** | **调用** | @Override public void onDisconnect(int clientId, String ipAddress, int port) {  System.*out*.println("onDisconnect: ip=" + ipAddress + "; port=" + port); } | | | |

### 3、标签拍灯事件（onPushButton）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **回调事件** | onPushButton | | | | |
| **功能说明** | 拍灯事件，手动短拍标签灯，触发的回调函数 | | | | |
| **返回参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 连接上的控制器IP |
| port | | int | 是 | 连接端口 |
| sourceAddress | | int | 是 | 所拍标签地址 |
| lightColor | | int | 是 | 所拍灯色(见标签和塔灯显示颜色列表) |
| stockNum | | int | 是 | 库存数量 |
| outNum | | int | 是 | 出库数量 |
| **C#调用** | | | | | |
| **示例** | **调用** | @Override public void onPushButton(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum) {  System.*out*.printf(  "onPushButton IP/Port:%s/%d tagID:%d LightColor:%d StockNum:%d OutNum:%d%n",  ipAddress, port, sourceAddress, lightColor, stockNum, outNum); } | | | |

### 4、出库拍灯事件（onSetOutStock）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **回调事件** | onSetOutStock | | | | |
| **功能说明** | 出库拍灯事件，手动短拍标签灯，触发的回调函数 | | | | |
| **返回参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 连接上的控制器IP |
| port | | int | 是 | 连接端口 |
| sourceAddress | | int | 是 | 所拍标签地址 |
| lightColor | | int | 是 | 所拍灯色(见标签和塔灯显示颜色列表) |
| stockNum | | int | 是 | 库存数量 |
| outNum | | int | 是 | 出库数量 |
| **C#调用** | | | | | |
| **示例** | **调用** | @Override public void onSetOutStock(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum) {  System.*out*.printf(  "onSetOutStock IP/Port:%s/%d tagID:%d LightColor:%d StockNum:%d OutNum:%d%n",  ipAddress, port, sourceAddress, lightColor, stockNum, outNum); } | | | |

### 5、标签长按事件（onPushLongButton）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **回调事件** | onPushLongButton | | | | |
| **功能说明** | 标签灯长按事件，手动长按标签灯3秒，触发的回调函数 | | | | |
| **返回参数** | **参数名** | | **数据类型** | **是否必填** | **备注** |
| ipAddress | | string | 是 | 连接上的控制器IP |
| port | | int | 是 | 连接端口 |
| sourceAddress | | int | 是 | 所拍标签地址 |
| lightColor | | int | 是 | 所拍灯色(见标签和塔灯显示颜色列表) |
| stockNum | | int | 是 | 库存数量 |
| outNum | | int | 是 | 出库数量 |
| **C#调用** | | | | | |
| **示例** | **调用** | @Override public void onPushLongButton(String ipAddress, int port, int sourceAddress, int lightColor, int stockNum, int outNum) {  System.*out*.printf(  "onPushLongButton IP/Port:%s/%d tagID:%d LightColor:%d StockNum:%d OutNum:%d%n",  ipAddress, port, sourceAddress, lightColor, stockNum, outNum); } | | | |

## 标签和塔灯显示颜色列表

color 参数使用 LightMode 枚举值

|  |  |  |
| --- | --- | --- |
| **内容** | **值** | **枚举定义** |
| 标签亮灯颜色 | 循环色：0x00  红灯：0x01 红灯闪烁：0x11  绿灯：0x02 绿灯闪烁：0x12  蓝灯：0x03 蓝灯闪烁：0x13  黄灯：0x04 黄灯闪烁：0x14  紫灯：0x05 紫灯闪烁：0x15  淡蓝色灯：0x06 淡蓝色灯闪烁：0x16  白灯：0x07 白灯闪烁：0x17  灭灯：0x08  其中数量：0(不显示) | public enum LightMode  {  MultiColor = 0x00,  Red = 0x01, RedFlicker = 0x11,  Green = 0x02, GreenFlicker = 0x12,  Blue = 0x03, BlueFlicker = 0x13,  Yellow = 0x04, YellowFlicker = 0x14,  Purple = 0x05, PurpleFlicker = 0x15,  LightBlue = 0x06, LightBlueFlicker = 0x16,  White = 0x07, WhiteFlicker = 0x17,  TurnOff = 0x08  } |
| 塔灯亮灯颜色 | 红灯：0x01 红灯闪烁：0x11  绿灯：0x02 绿灯闪烁：0x12  黄灯：0x04 黄灯闪烁：0x14  灭灯：0x08 | 塔灯显示和标签联动同步显示 |