自动服务主机 M3

接口文档

概述

本文档描述了自动售货机系统系列产品接口的使用和设置方法。调用接口的时候需要熟悉 android 调用.so 的相关方法。

目录

1. 串口服务接口

1.1. 开发包及函数说明

1) 文件夹 SerialLibs 是开发 app 所需要的库文件；
2) HogoSerialDemo1.2 是编译好的 demo；
3) HogoSerialDemo.zip 是 demo 的源码，供参考。

系统中有一个虚拟串口服务（/system/bin/hjserial）,服务会在系统启动后自动运行。此部分接口是与虚拟串口服务通信，实现数据的发送与接收。

libMySerial.so 是通过 ndk 编译出来的动态库，该库实现与虚拟串口服务通信。

app 开发人员不用关注如何操作底层设备，只需使用该库以及该库提供出来的接口即可。

1.2. 接口介绍

1.2.1.系统初始化

功能	初始化接口内部的数据，APP 启动后先调用此接口且整个 app 只调用一次。
原型	public native int init();
参数	无
返回	0:成功 -1:失败

1.2.2.打开串口

功能	打开串口
原型	public native int open(int port);
参数	Port：要打开的串口编号 0～3（值为 1 时是调试口，不推荐使用） 打开串口时使用默认配置：指定串口号、波特率：57600、校验位：NONE、数据位 8、停止位：1
返回	0：成功 负值(-1~-6)：失败

1.2.3.关闭串口

功能	将打开的串口关闭
原型	public native void close(int port);
参数	Port：要打开的串口编号 0～3
返回	无

1.2.4.配置串口

功能	打开串口时，使用的默认配置，可以通过此函数更改配置。注意参数的类型。
原型	public native int setinfo(int port, int baud, String parity, int databit, String stopbit);
参数	Port：要打开的串口编号 0～3 Baud：波特率（2400, 4800, 9600, 19200, 57600, 115200） Parity:校验位（“NONE”，“ODD”，“ EVEN”） Databit：数据位（8） Stopbit：停止位（“1”，“1.5”，“2”） 注意：配置信息只能是以上所列的配置项
返回	0：成功　负值(-1~-6)：失败

1.2.5.获取串口信息

功能	获取串口信息
原型	public native String getinfo(int port);
参数	Port：要打开的串口编号 0～3
返回	成功：{"comx":"COM0", "baud":115200, "parity":"NONE", "databit":8 "stopbit": "1"} 失败：{}

1.2.6.发送数据

功能	向串口发送数据
原型	public native int write(int port, byte[] buf, int len);
参数	Port：要打开的串口编号 0～3 Buf: 发送数据缓冲区 Len：发送数据长度

返回	成功：写入数据的长度 len 失败：负值

1.2.7.读取数据

功能	从串口读取数据
原型	public native int read(int port, byte[] buf, int len, int waittime);
参数	Port：要打开的串口编号 0～3 Buf: 接收数据缓冲区 Len：接收数据缓冲区大小 Waittime：超时等待时间
返回	成功：读取到数据的长度 失败：负值

2. 扫码摄像头接口

2.1. 开发包及函数说明

1) SC60_CAMERA_Libs 是开发 app 所需要的库文件；
2) HogoCameraDemo1.2_M3 是编译好的 demo。
3) HogoScanCameraSetting.zip 是 demo 的源码，供参考。

该 demo 采用 android jni 开发实现 uvc 摄像头预览图像。

需要用到的动态库：libImageProc.so。

libImageProc.so 是通过 ndk 编译出来的动态库，该库实现对外界 uvc 摄像头的打开，关闭，图像预览等操作。

app 开发人员不用关注如何操作底层设备，只需使用该库以及该库提供出来的接口即可。

2.2. 设备权限

```
//执行修改设备权限命令
Process sh = null;
try {
    sh = Runtime.getRuntime().exec("su", null,null);
    } catch (IOException e) {
    e.printStackTrace();
}
OutputStream os = sh.getOutputStream();
try {
```

```
            os.write(("chmod 666 /dev/devname").getBytes("ASCII"));
            } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            os.flush();
            } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            os.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

2.3. 接口介绍

2.3.1.获得当前指定 vid、pid 的摄像头设备节点名

功能	获得当前指定 vid、pid 的摄像头设备节点名
原型	public native String getcameraname(void);
参数	无
返回	返回设备节点名称 如:video1

2.3.2.指定打开某个摄像头

功能	指定打开某个摄像头
原型	public native int prepareCamera(int videoid);
参数	videoid:摄像头编号 id.
返回	0:成功 -1:失败

2.3.3.设置初始化摄像头信息

功能	设置初始化摄像头信息
原型	public native int setCameraInitInfo(int pixelformat, int width, int height, int denominator);

参数	Pixelformat:视频流格式 0：mjepg 1:yuyv Width:分辨率 Height:分辨率 Denominator:帧率
返回	-1:失败 0:成功

2.3.4.读取每帧数据(视频流为 YUYV)

功能	保持相机图像（像素数据存储在 JNI 中的数组中）
原型	public native void processCameraYUYV(void);
参数	无
返回	无

2.3.5.读取每帧数据(视频流为 MJPEG)

功能	保持相机图像（像素数据存储在 JNI 中的数组中）
原型	public native int processCameraMJPEG(void);
参数	无
返回	返回获取每帧数据的大小

2.3.6.获取设置后的摄像头信息（必须在打开成功后调用）

功能	获取设置后的摄像头信息
原型	public native String getCameraInitInfo(void);
参数	无
返回	返回读取到的当前设定的摄像头参数

2.3.7.将相机图像转到 bmp（当视频流格式为 yuyv 时，该接口必须调用）

功能	获得一个相机图像(像素数据存储在 JNI 中的一个数组中)
原型	public native void pixeltobmp(Bitmap bitmap);

参数	bitmap:用来存储 bitmap 类型参数，如用来解析的二维码
返回	无

2.3.8.停止打开的摄像头

功能	关闭打开的摄像头
原型	public native void stopCamera();
参数	无
返回	无

2.3.9.设置存储 bytebuffer 数据的内存给 jni 使用（当视频流格式为 mjpeg 时，该接口必须调用）

功能	设置存储 bytebuffer 数据的内存给 jni 使用
原型	public native int setDirectBuffer(ByteBuffer buf,int size);
参数	Buf:ByteBuffer 类型数据，用于存储每帧的数据 Size:每帧的最大字节数
返回	0：成功 -1：失败