

# SmartPos SDK 用户指南

版本 2.6

2024-08-05

版本	发布日期	版本描述
1.0	2015-06-28	第一次发布
2.0	2015-10-17	新增磁卡、智能卡、热敏打印新增字符编码
2.1	2019-02-16	打印新增特性 QR，多线段，速度提高到 70mm/s
2.2	2019-04-10	新增安全访问模块
2.3	2020-03-21	修订
2.4	2020-08-17	智能卡 API 更新支持多种卡模式
2.5	2024-02-02	新增 LED API
2.6	2024-08-05	更新 SAM 卡 API，不再支持旧 SAM 卡模块

未经本公司许可，任何单位和个人不得擅自摘抄、复制本文当内容的部分或全部，并不得以任何形式传播。

## 目录

1.概述.....	4
2.目录结构.....	4
3.集成方法.....	5
4.热敏打印.....	6
4.1 概述.....	6
4.2 功能特点.....	6
4.3 工作流程.....	7
4.4 函数接口.....	8
4.5 FAQ.....	24
5.磁卡读卡器.....	26
5.1 概述.....	26
5.2 工作流程.....	26
5.3 函数接口.....	27
5.4 FAQ.....	29
6.智能卡读卡器.....	30
6.1 概述.....	30
6.2 工作流程.....	30
6.3 函数接口.....	31
7.NFC 读卡器.....	36
7.1 概述.....	36
7.2 FAQ.....	36
8.摄像头.....	37
8.1 概述.....	37
8.2 FAQ.....	37
9.指纹模块.....	38
9.1 概述.....	38
9.2 工作流程.....	38
9.3 函数接口.....	39

10.客户显示屏.....	41
10.1 概述.....	41
10.2 FAQ.....	41
11.串口设备.....	42
11.1 概述.....	42
11.2 工作流程.....	42
11.3 函数接口.....	43
12.安全访问模块（SAM/PSAM） .....	45
12.1 概述.....	45
12.2 工作流程.....	45
12.3 函数接口.....	46
12.4 FAQ.....	50
13.钱箱.....	51
13.1 概述.....	51
13.2 函数接口.....	51
13.3 FAQ.....	51
14.扫码器.....	52
14.1 概述.....	52
14.2 FAQ.....	52
15.LED.....	53
15.1 概述.....	53
15.2 函数接口.....	53
16.附录.....	54
16.1 打印机固件恢复.....	54
16.2 全屏模式.....	54

# 1.概述

SmartPos SDK 提供了易用的 API，帮助 Android 应用开发工程师快速开发客户化应用，包括热敏打印、磁卡读卡器、智能卡读卡器、NFC、指纹识别、条形码识别、二维码识别等功能。

# 2.目录结构

SDK 包含 2 部分，开发者只需要导入这些文件到项目里：

## 1.paydevice-smartpos-sdk.jar

该 JAR 包集成了热敏打印、磁卡读卡器、智能卡读卡器、指纹识别等外设模块的 API，方便 java 层调用。

## 2.libpaydevice-smartpos.so libAlUSB.so

这些文件为动态链接库，它实现了操作相关硬件设备的 jni 层接口。

## Demo

该工程演示了大部分常用 API 的调用方法。

## zbar.jar libiconv.so libzbarjni.so

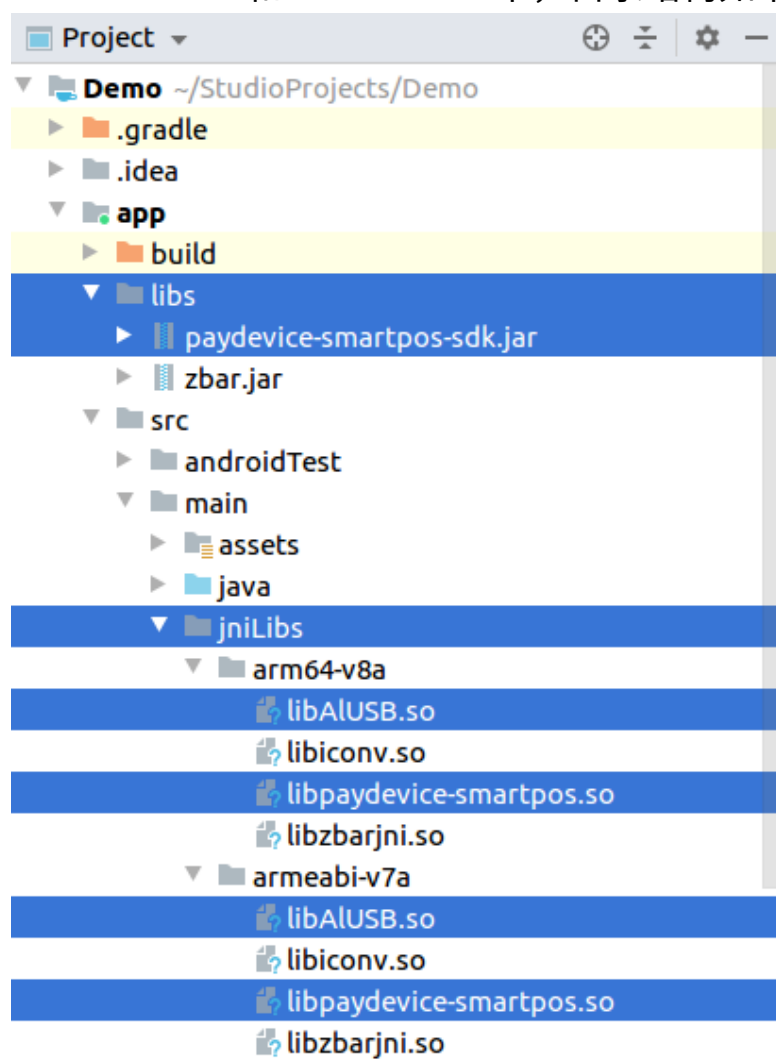
**注：**示例基于 ZBar 实现条码和二维码的扫描，如不需要摄像头扫码可以忽略。

<https://github.com/ZBar/ZBar>

## 3.集成方法

Android 应用开发环境以 Android Studio 为主要开发工具，这里以 Android Studio 开发工具为例介绍集成方法。用户只需集 2 步集成 SDK 到项目中：

1. 将 SDK 包中的 **paydevice-smartpos-sdk.jar** 拷贝到工程的 **libs** 目录下
2. 将 SDK 中 **libpaydevice-smartpos.so** **libAlUSB.so** 拷贝到工程目录的 **armeabi-v7a** 和 **arm64-v8a** 下，目录结构如下图所示：



## 4.热敏打印

### 4.1 概述

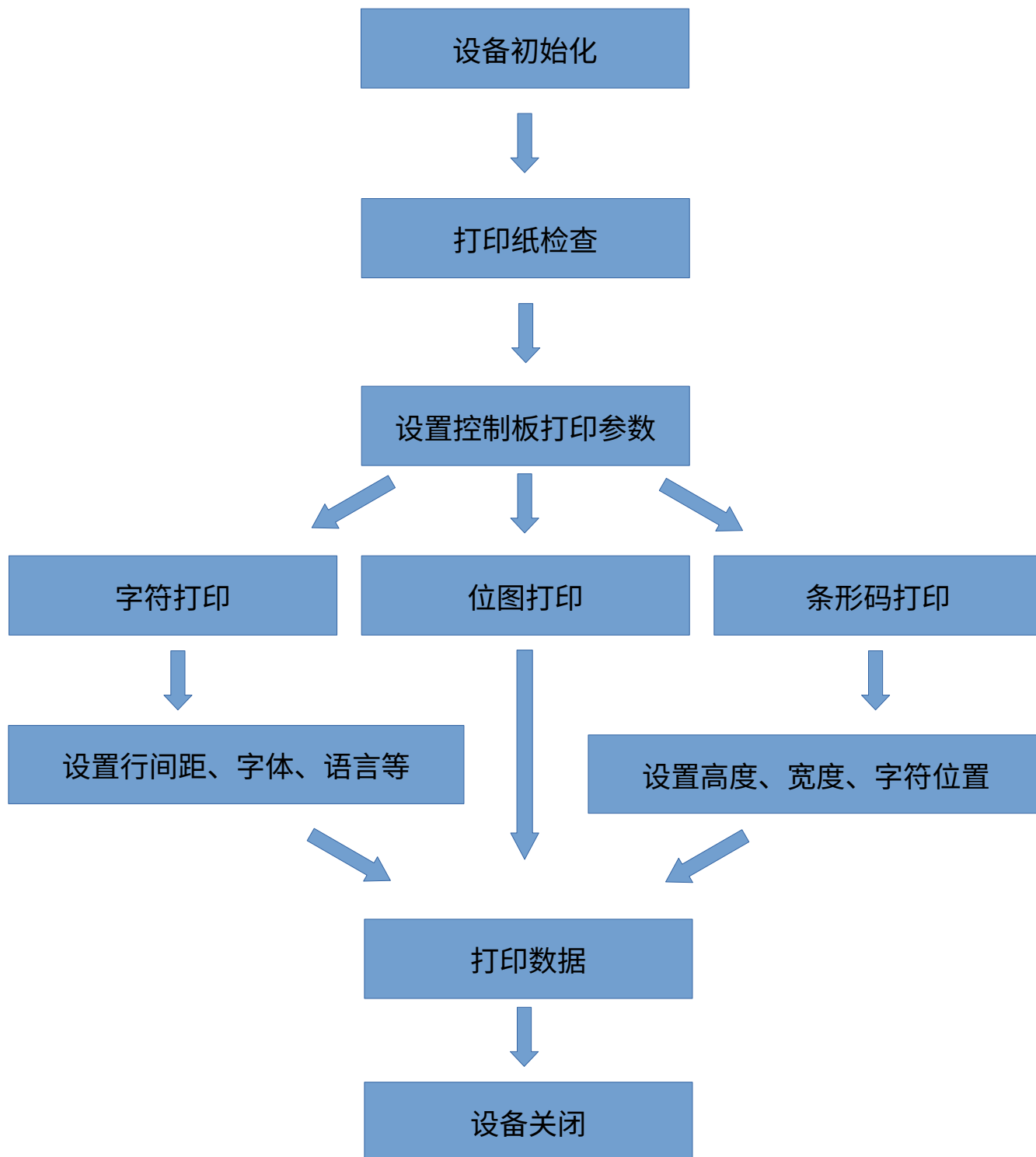
热敏打印模块提供丰富的 API 来实现打印功能

### 4.2 功能特点

热敏打印模块支持以下功能：

- \* 支持多国语言、打印参数、字符格式等设置
- \* 支持字符串、QR 码、条形码、位图打印

## 4.3 工作流程



## 4.4 函数接口

### 名词解释

点数：打印点的数量，点即打印的最小单位。1 点=0.125 毫米

最大打印宽度：打印机有效打印宽度。

58mm 打印机：最大打印宽度 48mm，最大点数 384 点。

80mm 打印机：最大打印宽度 72mm，最大点数 576 点。

### 函数异常抛出错误码：

<code>public static final int PRINTER_ERR_OPEN</code>	打印打开错误
<code>public static final int PRINTER_ERR_CLOSE</code>	打印机关闭错误
<code>public static final int PRINTER_ERR_READ</code>	打印机读错误
<code>public static final int PRINTER_ERR_WRITE</code>	打印机写错误
<code>public static final int PRINTER_ERR_PARAM</code>	函数参数错误
<code>public static final int PRINTER_ERR_NO_PAPER</code>	无打印纸
<code>public static final int PRINTER_ERR_UPDATE</code>	固件更新错误

### 打印机型号：

<code>public static final int PRINTER_MODEL_UNKNOWN</code>	//未知型号
<code>public static final int PRINTER_MODEL_PRN2103</code>	//型号 PRN2103



打印机类型:

```
public static final int PRINTER_TYPE_USB //USB 打印机
```

```
public static final int PRINTER_TYPE_SERIAL //串口打印机
```

打印纸宽度:

```
public static final int TYPE_PAPER_WIDTH_58MM //58mm
```

```
public static final int TYPE_PAPER_WIDTH_80MM //80mm
```

切纸方式:

```
public static final int FULL_CUT //全切
```

```
public static final int HALF_CUT //半切
```

打印对齐方式:

```
public static final int ALIGN_LEFT      左对齐
```

```
public static final int ALIGN_MIDDLE    居中
```

```
public static final int ALIGN_RIGHT    右对齐
```

字体格式:

```
public static final int FONT_DEFAULT          =0      默认格式
```

```
public static final int FONT_SMALL            =1<<0    9x17 字体
```

```
public static final int FONT_INVERSE          =1<<1    字体反白
```

<code>public static final int FONT_UPSIDE_DOWN</code>	<code>=1&lt;&lt;2</code>	上下颠倒
<code>public static final int FONT_EMPHASIZED</code>	<code>=1&lt;&lt;3</code>	字体加粗
<code>public static final int FONT_DOUBLE_HEIGHT</code>	<code>=1&lt;&lt;4</code>	双倍高度
<code>public static final int FONT_DOUBLE_WIDTH</code>	<code>=1&lt;&lt;5</code>	双倍宽度
<code>public static final int FONT_UNDERLINE</code>	<code>=1&lt;&lt;7</code>	有下划线

下划线格式:

<code>public static final int UNDERLINE_ZERO</code>	无下划线
<code>public static final int UNDERLINE_HIGH_1</code>	下划线高度 1
<code>public static final int UNDERLINE_HIGH_2</code>	下划线高度 2

条形码字符位置:

<code>public static final int CODEBAR_STRING_MODE_NONE</code>	无字符串
<code>public static final int CODEBAR_STRING_MODE_ABOVE</code>	位于条形码上方
<code>public static final int CODEBAR_STRING_MODE_BELOW</code>	位于条形码下方
<code>public static final int CODEBAR_STRING_MODE_BOTH</code>	上方和下方都有

条形码编码格式:

<code>public static final int UPC_A</code>	
<code>public static final int UPC_E</code>	
<code>public static final int EAN13</code>	

```
public static final int EAN8
public static final int CODE39
public static final int I25
public static final int CODEBAR
public static final int CODE93
public static final int CODE128
public static final int CODE11
public static final int MSI
```

字符编码（45 种）：

字符表参考 [https://en.wikipedia.org/wiki/Code\\_page](https://en.wikipedia.org/wiki/Code_page)

```
public static final int CODE_PAGE_CP437    //English
public static final int CODE_PAGE_CP720    //Arabic
public static final int CODE_PAGE_CP737    //Greek
public static final int CODE_PAGE_CP755    //East Europe, Latvian2
public static final int CODE_PAGE_CP775    //Estonian, Lithuanian,
Latvian
public static final int CODE_PAGE_CP850    //Western Europe
public static final int CODE_PAGE_CP852    //Latin2
public static final int CODE_PAGE_CP855    //Cyrillic script
```

```
public static final int CODE_PAGE_CP856    //Hebrew
public static final int CODE_PAGE_CP857    //Turkish
public static final int CODE_PAGE_CP858    //West Europe
public static final int CODE_PAGE_CP860    //Portuguese
public static final int CODE_PAGE_CP862    //Hebrew
public static final int CODE_PAGE_CP863    //Canadian-French
public static final int CODE_PAGE_CP864    //Arabic
public static final int CODE_PAGE_CP865    //Nordic
public static final int CODE_PAGE_CP866    //Cyrillic2
public static final int CODE_PAGE_CP874    //Thai
public static final int CODE_PAGE_CP1250   //Central Europe
public static final int CODE_PAGE_CP1251   //Cyrillic
public static final int CODE_PAGE_CP1252   //Latin1
public static final int CODE_PAGE_CP1253   //Greek
public static final int CODE_PAGE_CP1254   //Turkish
public static final int CODE_PAGE_CP1255   //Hebrew
public static final int CODE_PAGE_CP1256   //Arabic
public static final int CODE_PAGE_CP1257   //Baltic
public static final int CODE_PAGE_CP1258   //Vietnamese
public static final int CODE_PAGE_ISO_8859_1 //West Europe
```

```
public static final int CODE_PAGE_ISO_8859_2 //Latin2
public static final int CODE_PAGE_ISO_8859_3 //Latin3
public static final int CODE_PAGE_ISO_8859_4 //Baltic
public static final int CODE_PAGE_ISO_8859_5 //Cyrillic
public static final int CODE_PAGE_ISO_8859_6 //Arabic
public static final int CODE_PAGE_ISO_8859_7 //Greek
public static final int CODE_PAGE_ISO_8859_8 //Hebrew
public static final int CODE_PAGE_ISO_8859_9 //Turkish
public static final int CODE_PAGE_ISO_8859_15 //Latin9
public static final int CODE_PAGE_BIG5 //Traditional Chinese
public static final int CODE_PAGE_GB18030 //Simplified Chinese
public static final int CODE_PAGE_LATVIAN //Latvian
public static final int CODE_PAGE_IRAN //Iran System encoding
standard
public static final int CODE_PAGE_IRAN2 //Iran2
public static final int CODE_PAGE_KATAKANA //Japanese
public static final int CODE_PAGE_MIK //Cyrillic,Bulgarian
public static final int CODE_PAGE_THAI //Thai1
public static final int CODE_PAGE_THAI2 //Thai2
```

函数接口列表：

\* 初始化打印机

`void connect()`

**注：** 串口打印机初始化需要约 100ms

\* 打印机关闭

`void disconnect()`

**注：** 串口打印机需要等待打印完成才能关闭

\* 检查打印纸

`void checkPaper()`

无打印纸抛出异常 `PRINTER_ERR_NO_PAPER`

\* 获取打印机类型

`int getPrinterType()`

返回值： `PRINTER_TYPE_USB`, `PRINTER_TYPE_SERIAL`

\* 将打印缓冲区数据发送到打印机并打印，同时清空打印缓冲区

`void sendData(String data)`

参数： `data` 需要打印的字符

`void sendData(String data, String encoding)`

参数： `data` 需要打印的字符

`encoding` 字符编码(CP437, ISO-8859-1 等)

**注：**Java 中 String 是 Unicode/UCS-2 编码，需要转换成对应的字符编码打印机才能正确打印，通常我们只打印一种语言时通过调用 `void setStringEncoding(String encoding)` 设定全局字符编码后直接调用 `void sendData(String data)` 打印即可。如需同时打印不同字符编码的字符请使用函数 `void sendData(String data, String encoding)`

\* 设置全局字符编码

`void setStringEncoding(String encoding)`

参数： `encoding` 字符编码

\* 走纸一行

`void cmdLineFeed()`

\* 走纸 n 行

`void cmdLineFeed(int n)`

参数： `n` 行数

**注：**行间距由函数 `void cmdSetDefaultLineSpacing()` 或 `void cmdSetLineSpacing(int dots)` 决定。

\* 跳转到下个制表位置

`public void cmdJumpTab()`

**注：** 跳转位置由函数 `void cmdSetTable(byte[] offset)` 决定

\* 设置水平制表位置

`void cmdSetTable(byte[] offset)`

parameters: `offset` 水平制表位置数组，数组长度 1-16，单位字符数（12 点）

**注：** 具体用法参考 Demo

\* 取消水平制表位置

`void cmdUnSetTable()`

\* 设置默认行间距（32 点）

`void cmdSetDefaultLineSpacing()`

\* 设置行间距为 n 点（默认 32 点）

`void cmdSetLineSpacing(int dots)`

参数: `dots` 点数

\* 设置打印对其方式（默认左对齐）

`void cmdSetAlignMode(int mode)`



参数: `mode` 对齐方式

`mode=ALIGN_LEFT`      左对齐

`mode=ALIGN_MIDDLE`    居中

`mode=ALIGN_RIGHT`     右对齐

\* 设置打印偏移点数

`void cmdSetPrintOffset(int offset)`

parameters: `offset` 水平偏移的点数

\* 设置打印模式

`void cmdSetPrintMode(int mode)`

参数: `mode` 打印模式

位 0:默认

位 1:字符反白

位 2:字符上下倒置

位 3:字符加粗

位 4:双倍高度

位 5:双倍宽度

位 6:顺时针旋转 90

位 7:下划线

**注：**设置相应的位为 1 即生效, 如果设置了旋转和反白则下划线无效。

\* 设置下划线

`void cmdSetUnderlineHeight(int n)`

参数: `n` 下划线

`n=UNDERLINE_ZERO`      无下划线

`n=UNDERLINE_HIGH_1`    下划线 1 点高度

`n=UNDERLINE_HIGH_2`    下划线 2 点高度

\* 设置字体放大倍数

`void cmdSetFontScaleSize(int scaleWidth, int scaleHeight)`

parameters: `scaleWidth` 宽度倍数 0~7

`scaleHeight` 高度倍数 0~7

\* 设置条形码字符串的位置

`void cmdSetBarCodeStringPosition(int mode)`

参数: `mode` 字符串的位置

`mode=CODEBAR_STRING_MODE_NONE`    不打印字符串

`mode=CODEBAR_STRING_MODE_ABOVE`    在条形码上方

`mode=CODEBAR_STRING_MODE_BELOW`    在条形码下方

`mode=CODEBAR_STRING_MODE_BOTH`    在上下方都有

\* 设置条形码的高度

`void cmdSetBarCodeHeight(int n)`

参数： `n` 条形码高度（单位：点数）

$1 \leq n \leq 255$  (默认 50)

\* 设置条形码基本线条的宽度

`void cmdSetBarCodeWidth(int n)`

参数： `n` 条形码基本线条宽度（单位：点数）

$n = 2$  或  $n = 3$  (默认 3)

\* 设置条形码打印左边距

`void cmdSetBarCodeLeftSpacing(int n)`

参数： `n` 条形码左边距（单位：点数）

\* 打印条形码

`void cmdBarCodePrint(int type, String string)`

参数： `type` 条形码编码方式

`string` 条形码对应的字符串

\* 位图打印(一次打印整个图片，当打印机缓存为空时有效)

`void cmdBitmapPrint(Bitmap bitmap, int left, int top)`

参数: **bitmap** Android Bitmap 实例

**left** 水平方向偏移点数

**top** 垂直方向偏移点数

**注:** 假设 bitmap 的宽和高分别为  $w$ 、 $h$ ，则  $left + w < 384$ ， $top + h < 384$

\* 位图打印(将位图分割成多个高度 24 点的图片逐个打印,当打印机缓存为空时有效)

**void cmdBitmapPrintEx(Bitmap bitmap, int left, int top)**

参数: **bitmap** Android bitmap 实例

**left** 水平方向偏移点数

**top** 垂直方向偏移点数

**Note:** 图片高度必须是 8 的整数倍

\* 设置控制板打印参数

**void cmdSetHeatingParam(int dots, int time, int interval)**

参数: **dots** 最多加热点数，范围 0-255，单位(8dots)。默认值 7

**time** 加热时间，范围 0-255，单位(10us)。默认值 80

**interval** 加热间隔时间，范围 0-255，单位(10us)，默认值 2

**注:** 该函数用于设置打印的最多加热点、加热时间、间隔时间。加热点数多，则控制板的最大耗电电流大，打印速度快。最大加热点数为  $8 \times (n1+1)$ ，加热时间越长，则打印黑度高，打印速度越慢。加热时间过短,则可能出现打印空白。

间隔时间越长,打印越清晰,打印速度越慢。

\* 设置打印浓度

`void cmdSetPrintDensity(int density, int delay)`

参数: `density` 打印浓度, 范围 0-31 对应 50%+5%\*density 的打印浓度

`delay` 打印延时, 范围 0-7 对应 delay\*250us

**注:** 该函数用于设置打印浓度, 不同品牌打印纸需调整不同打印浓度。

\* 设置打印字符集

`void cmdSetPrinterLanguage(int code)`

参数: `code` 打印语言字符集

\* 二维码打印(打印控制板 v1.04 之后支持)

`void cmdQrCodePrint(int version, int ecc, String data)`

parameters: `version` QR 版本 0~16(n 点 x n 点 矩形大小)

`ecc` QR 错误纠正等级 1~4

`data` QR 二维码字符

\* 线段打印(打印控制板 v1.04 之后支持)

`void cmdPrintMultipleLines(int lineCount, int[] lineStartPos, int[]`

## lineEndPos)

parameters: **lineCount** 线段总数

**lineStartPos** 线段开始点横坐标数组

**lineEndPos** 线段结束点横坐标数组

注意：多个线段可以组成实线或曲线

\* 保存位图到 NVRAM（即使打印机断电后打印控制板的 NVRAM 仍可以保存位图）

**void cmdSaveBitmapToNVRAM(Bitmap[] bmpArray)**

参数: **bmpArray** 位图数组

**注意:** NVRAM 最大空间 64KB,位图的宽和高应必须是 8 的整数倍。

**警告:** 请不要频繁擦写 NVRAM，每天最好不要超过 10 次，否则可能导致 NVRAM 损坏。通常位图只保存一次到 NVRAM，每次使用时打印即可。

通常,NVRAM 用于打印大图或者函数 **cmdBitmapPrint()** 或 **cmdBitmapPrintEx()**无法打印的情况，NVRAM 打印效果最好。

\* 依据索引从 NVRAM 中打印位图

**void cmdPrintBitmapFromNVRAM(int index, int zoom)**

参数: **index** 位图索引(1~255)

**zoom** 放大模式

BITMAP\_ZOOM\_NONE: 原始大小

BITMAP\_ZOOM\_WIDTH: 2 倍宽

BITMAP\_ZOOM\_HEIGHT: 2 倍高

BITMAP\_ZOOM\_BOTH: 2 倍宽和倍高

\* 删除 NVRAM 里的所有位图

`void cmdDeleteBitmapFromNVRAM()`

\* 打印测试页

`void cmdPrintTest()`

\* 获取打印机型号（串口打印机有效）

`int cmdGetPrinterModel()`

\*切纸（USB 打印机有效）

`public void cmdCutPaper(int mode)`

参数: mode 切纸方式 FULL\_CUT,HALF\_CUT

## 4.5 FAQ

问题 1:

图片打印错误或空白

解决方法:

1.调整加热参数 `cmdSetHeatingParam` 或者尝试函数 `cmdBitmapPrintEx()` 来分割打印图片。推荐使用 NVRAM 打印所有的位图。图片黑色区域越多加热时间需要减小, 可以使用 `ImageUtils` 类来将彩色图片转换为黑白图。

2.图片宽度不要超过有效打印宽度, 58mm 打印机有效宽度 48mm 对应图片宽度 384px, 80mm 打印机有效宽度 72mm 对应图片宽度 576px。

**注:** 打印浓度越高越清晰但打印越慢, 不要在多个线程里同时调用打印命令。电池供电时, 电压越低打印速度越慢, 电压过低可能导致打印失败。

问题 2:

打印字符错误或是其它语言

解决方法:

一般情况下是打印语言不匹配造成的问题, 打印的字符编码必须和打印字符集 (Code Page) 匹配才行, 也就是说函数 `setStringEncoding()` 和 `cmdSetPrinterLanguage()` 的参数必须匹配。具体参考 Demo 中 `PosSalesSlip.java` 的函数 `languageTest()`。



如果您的语言打印机不支持或者打印不正确，尝试通过位图打印字符。  
参考 Demo 中 [PosSalesSlip.java](#) 的函数 [printUnsupportLanguage\(\)](#)。

**另外注意打印流程，部分用户在打印过程中去发送其它指令或重复调用 `connect()`，会导致打印乱码或者打印机异常，打印机异常通常调用 `fwUpdate()` 修复即可，也不要多线程操作打印机，多任务请用队列实现。正确打印流程是 `connect->check paper->printing->disconnect`**

### 问题 3:

如何调用外部打印机打印

解决方法:

外部串口打印机可以调用 [selectPrinter\(String path, int speed\)](#)，外部 USB 打印机调用 [UsbPrinter.getPrinterList\(\)](#) 获取打印机列表然后调用 [selectPrinter\(UsbDevice device\)](#)。注意，不同品牌打印机的 ESC 指令可能有差异，可以调用 [sendCmd\(byte\[\] cmd\)](#) 来直接发指令。

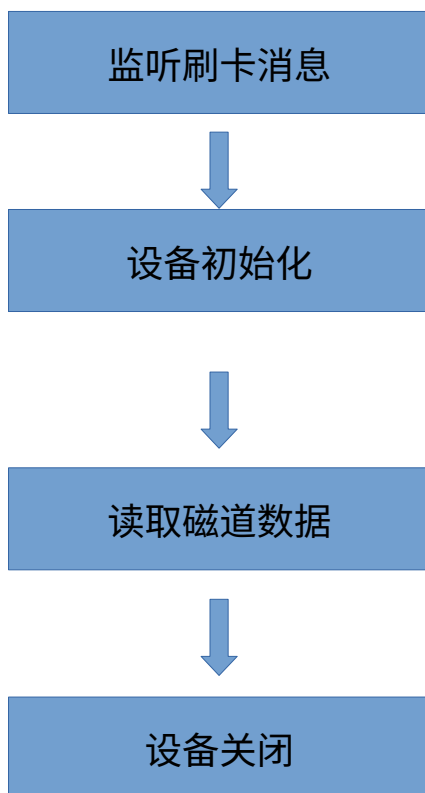
## 5.磁卡读卡器

### 5.1 概述

磁卡（又称为磁条卡），磁卡相关标准：ISO7810、ISO7811(1-6 部分)、ISO7812、ISO7813、ISO15457。

磁卡读卡器用来读取磁条卡信息，例如信用卡、银行卡、会员卡、购物卡等。

### 5.2 工作流程



## 5.3 函数接口

函数异常抛出错误码：

<code>public static final int MCR_ERR_INIT</code>	磁卡读卡器初始化错误
<code>public static final int MCR_ERR_DEINIT</code>	磁卡读卡器关闭错误
<code>public static final int MCR_ERR_NO_INIT</code>	磁卡读卡器未初始化
<code>public static final int MCR_ERR_ALREADY_INIT</code>	磁卡读卡器已经初始化
<code>public static final int MCR_ERR_PARAM</code>	函数参数错误

磁卡读卡器错误码：

```
//Preamble error in card read data
public static final int MCR_ERR_CODE_PREAMBLE    = 0x51;

//Postamble error in card read data
public static final int MCR_ERR_CODE_POSTAMBLE    = 0x52;

//LRC error in card read data
public static final int MCR_ERR_CODE_LRC          = 0x53;

//Parity error in card read data
public static final int MCR_ERR_CODE_PARITY       = 0x54;

//Blank track
public static final int MCR_ERR_CODE_BLANK_TRACK  = 0x55;
```

```
//STX/ETX error in command communication
public static final int MCR_ERR_CODE_STX_ETX          = 0x61;

//Class/Function un-recognizable in command
public static final int MCR_ERR_CODE_CLASS_FUNCTION = 0x62;

//BCC error in command communication
public static final int MCR_ERR_CODE_BBC              = 0x63;

//Length error in command communication
public static final int MCR_ERR_CODE_LENGTH           = 0x64;

//No data available to re-read
public static final int MCR_ERR_CODE_NO_DATA          = 0x65;

//No more space available for OTP write
public static final int MCR_ERR_CODE_OTP_WRITE_FULL   = 0x71;

//OTP write try without data
public static final int MCR_ERR_CODE_OTP_WRITE        = 0x72;

//CRC error in read data from OTP
public static final int MCR_ERR_CODE_OTP_CRC          = 0x73;

//No data stored in OTP
public static final int MCR_ERR_CODE_OTP_EMPTY        = 0x74;
```

## 函数接口列表：

- \* 初始化磁卡读卡器

`void init()`

- \* 关闭磁卡读卡器

`void deinit()`

- \* 查询数据状态

`int query()`

返回值： 0 有磁条数据

-1 无磁条数据

- \* 读取指定磁轨的数据

`byte[] getTrack(int trackId)`

返回值： 磁轨数据（16 进制字节数组）

参数： `trackId` 磁轨编号（1，2，3）

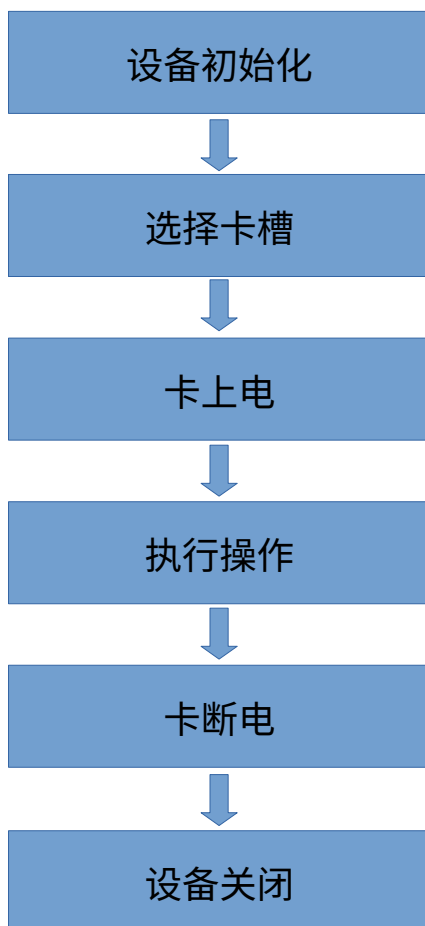
## 5.4 FAQ

## 6. 智能卡读卡器

### 6.1 概述

智能卡（接触式卡）相关标准：ISO7816（1-5 部分）、中国金融集成电路（IC）卡规范（PBOC2.0），其中 ISO7816-4 部分定义了 2 种传输协议：字符传输协议 T0，块传输协议 T1。支持的卡模式：ISO7816/AT24C/AT45D/AT88SC10X/AT88SC160X/SLE4442/SLE4428/SLE6636

### 6.2 工作流程



## 6.3 函数接口

函数异常抛出错误码：

<code>public static final int SCR_ERR_INIT</code>	//读卡器初始化错误
<code>public static final int SCR_ERR_DEINIT</code>	//读卡器关闭错误
<code>public static final int SCR_ERR_NO_INIT</code>	//读卡器未初始化
<code>public static final int SCR_ERR_ALREADY_INIT</code>	//读卡器已经初始化
<code>public static final int SCR_ERR_PARAM</code>	//函数参数错误
<code>public static final int SCR_ERR_CODE_SUCCESSFUL</code>	//无错误码
<code>public static final int SCR_ERR_CODE_TRANSMIT_ERROR</code>	//传输错误
<code>public static final int SCR_ERR_CODE_CMD_FAIL</code>	//发送命令失败
<code>public static final int SCR_ERR_CODE_CMD_BUSY</code>	//设备繁忙
<code>public static final int SCR_ERR_CODE_NOT_SUPPORT</code>	//命令不支持
<code>public static final int SCR_ERR_CODE_NO_CARD</code>	//卡槽无卡
<code>public static final int SCR_ERR_CODE_MEM_ERROR</code>	//内部存储错误
<code>public static final int SCR_ERR_CODE_TIMEOUT</code>	//通信超时
<code>public static final int SCR_ERR_CODE_INVALID_PARAMETER</code>	//非法参数
<code>public static final int SCR_ERR_CODE_NOT_INITIALIZED</code>	//卡未初始化
<code>public static final int SCR_ERR_CODE_PBOC_FAIL</code>	//PBOC 错误
<code>public static final int SCR_ERR_CODE_CARD_INACTIVE</code>	//卡停止

```
public static final int SCR_ERR_CODE_CARD_LOCKED    //卡锁定  
public static final int SCR_ERR_CODE_DEV_ERR
```

卡模式：

```
public static final int SCR_MODE_ISO7816  
public static final int SCR_MODE_AT24C  
public static final int SCR_MODE_SLE4428  
public static final int SCR_MODE_SLE4442  
public static final int SCR_MODE_AT88SC1608  
public static final int SCR_MODE_AT45D041  
public static final int SCR_MODE_SLE6636  
public static final int SCR_MODE_AT88SC102
```

CCID 协议：

```
public static final int PROTOCOL_T0    //字符传输协议  
public static final int PROTOCOL_T1    //块传输协议
```



## 函数接口列表：

- \* 初始化智能卡读卡器

`void init()`

- \* 关闭智能卡读卡器

`void deinit()`

- \* 选择卡槽

`void selectSlot(int slot)`

参数： `slot` 卡槽号

**注：**双卡读卡器有卡槽 0 和 1，单卡读卡器只有卡槽 0

- \* 卡上电

`void powerOn()`

- \* 卡断电

`void powerOff()`

- \* 获取 ATR (Answer To Reset)

`public byte[] getATR()`

返回值：ATR 数据（16 进制字节数组）

\* 获取卡使用的传输协议

`public int getProtocol()`

返回值： PROTOCOL\_T0

PROTOCOL\_T1

\* 获取卡的序列号

`public byte[] getSN()`

返回值：卡序列号（16 进制字节数组）

\* 发送 APDU 命令

`public byte[] sendAPDU(byte[] apdu)`

返回值：APDU 响应（16 进制字节数组）

参数： `apdu` APDU 命令（16 进制字节数组）

\* 发送 APDU 命令，用于长响应字节。

`public void sendAPDU(byte[] apdu, byte[] response, int[] responseLen)`

参数： `apdu` 指令，格式 CLA+INS+P1+P2+Lc+cmd+Le

`response` 应答字节（16 进制字节数组）

`responseLen` 应答字节长度数组

非 ISO7816 模式 API 请参考 doc 和 Demo

## 6.4 FAQ

## 7.NFC 读卡器

### 7.1 概述

Android 标准 API，支持读写模式。兼容 ISO/IEC 14443 A/B，ISO/IEC 15693，MIFARE，FeliCa 多种标准卡，Demo 演示了读取 UID 和标签读写。

### 7.2 FAQ

API 参考：<https://developer.android.google.cn/guide/topics/connectivity/nfc/nfc>

## 8. 摄像头

### 8.1 概述

摄像头参数：5 百万/1300 百万像素、720P@30fps 预览、自动对焦、AWB/AEC/AGC，闪光灯，具体参数根据机型有所不同。

### 8.2 FAQ

摄像头主要用于条形码识别和二维码识别，用户可参考开源项目

Zxing: <https://github.com/zxing/zxing/wiki/Getting-Started-Developing>

Zbar: <http://zbar.sourceforge.net>

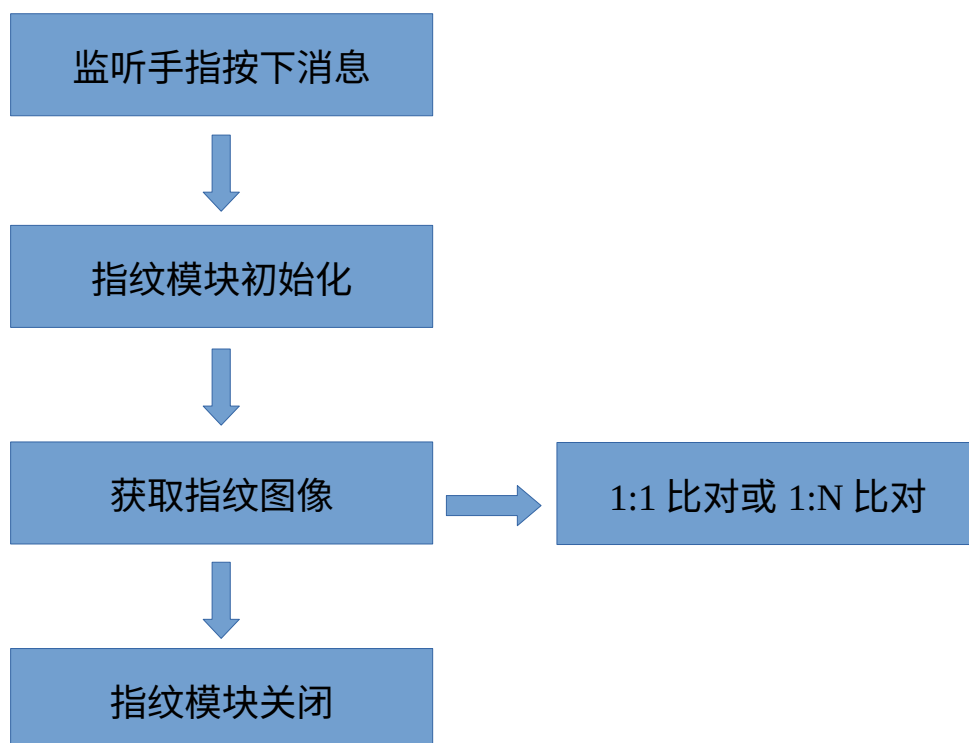
## 9. 指纹模块

### 9.1 概述

指纹模块提供接口：读取指纹图像、指纹特征提取、1：1 比对功能，用户可提取指纹图像后实现自己的特征提取和比对功能。相关参数：

- 图像深度：8 位
- 图像大小：192 x 192
- DPI：508

### 9.2 工作流程



## 9.3 函数接口

监听按键值

```
public static final int FINGERPRINT_FINGER_PRESENT    //手指按键码
```

函数异常抛出错误码：

```
public static final int FINGERPRINT_ERR_NO_INIT //指纹模块未初始化
```

```
public static final int FINGERPRINT_ERR_INIT      //指纹模块初始化错误
```

```
public static final int FINGERPRINT_ERR_DEINIT   //指纹模块关闭错误
```

```
public static final int FINGERPRINT_ERR_ALREADY_INIT //指纹模块已初始化
```

```
public static final int FINGERPRINT_ERR_NO_FINGER //没有检测到指纹
```

```
public static final int FINGERPRINT_ERR_IMAGE_UNCLEAR //图像不清晰
```

```
public static final int FINGERPRINT_ERR_IO        //通讯错误
```

函数接口列表：

\* 初始化指纹模块

```
void init()
```

\* 关闭指纹模块

`void deinit()`

\* 采集指纹图像

`byte[] getBitmapBytes()`

返回值：指纹图像字节数组(包含 1078 字节 BMP 头信息)

\* 获取指纹图像大小

`int getBitmapWidth()`

`int getBitmapHeight()`

\* 获取当前指纹特征模板(自行实现)

`byte[] getMinutiae()`

返回值：指纹特征(格式标准 ISO/IEC 19794-2:2005)

\* 1:1 比对(自行实现)

`int verify(byte[] probe, byte[] candidate)`

## 9.4 FAQ



## 10. 客户显示屏

### 10.1 概述

客户显示屏可用来给客户展示各种内容和输入数据（例如播放广告图片、视频，提供给客户输入帐号和密码）。Demo 中包含了基本功能演示

**注：**单屏设备只有一个 LCD，第二显示屏可以通过 HDMI 来实现。双屏设备包含两个 LCD

目前有两种方法在第二显示屏上显示：

1. Presentation （在 API 级别 17 中添加，Android4.2）
2. 启动 Activity 到第二显示屏 （在 API 级别 26 中添加，Android8.0）

### 10.2 FAQ

Presentation: <https://developer.android.google.cn/reference/android/app/Presentation.html>

启动 Activity 到第二显示屏：

<https://developer.android.google.cn/about/versions/oreo/android-8.0?#mds>

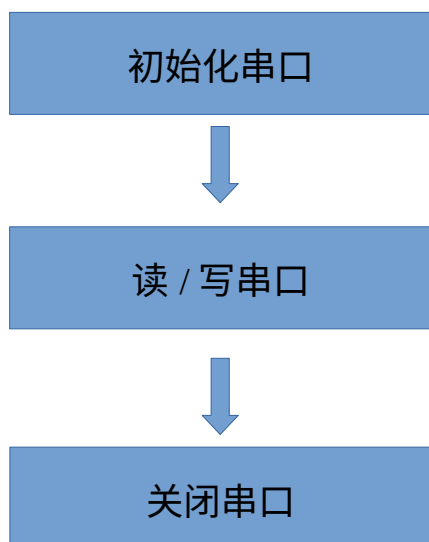
## 11. 串口设备

### 11.1 概述

SmartPos 设备包含一个 DB9 串口（RS232 协议），可以连接电子称、外置打印机等串口设备。

**注：**DB9 端口是/dev/db9。如果连接外置 USB 串口设备，则端口为/dev/ttyUSBx(ttyUSB0, ttyUSB1, ...)。如果设备带 4G 模块，注意不要操作 ttyUSB0-4，这些节点是 4G 模块生成的。部分 USB-CDC 设备会生成 ttyACMx 这样的节点，调试过程中使用 lsusb，logcat 等方法确认正确的节点。

### 11.2 工作流程



## 11.3 函数接口

### \* 串口初始化

SerialPort (File device, int baudrate)

参数: **device** 串口设备

**baudrate** 串口波特率

SerialPort (File device, int baudrate, int dataBits, int stopBits, char parity, char flowCtrl)

parameters: **device** 串口节点

**baudrate** 串口波特率 9600~4000000

**dataBits** 数据位 5,6,7,8

**stopBits** 停止位 1,2

**parity** 校验位 N/n,E/e,O/o (N-none, E-Even, O-Odd)

**flowCtrl** 流控 N/n,S/s,H/h (N-none, S-XON/XOFF, H-RTS/CTS)

### \* 读串口数据

InputStream getInputStream()

### \* 写串口数据

OutputStream getOutputStream()

\* 关闭串口

**void close()**

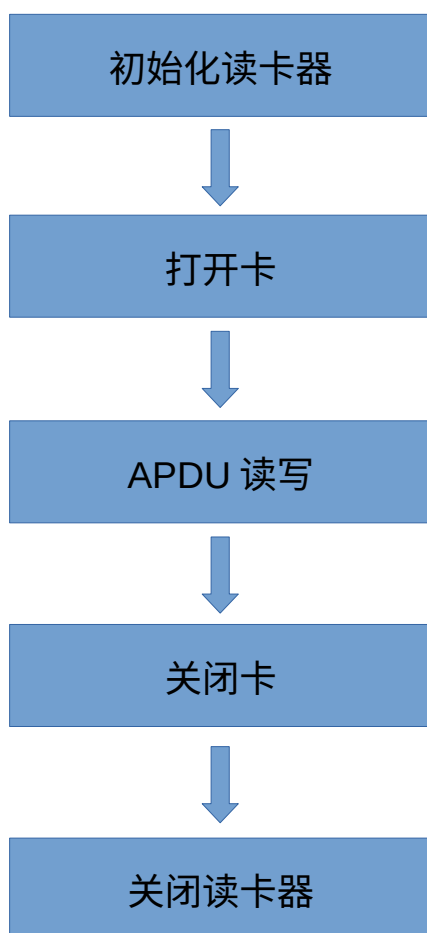
## 11.4 FAQ

## 12.安全访问模块（SAM/PSAM）

### 12.1 概述

SmartPos 最多支持 SAM 卡槽，支持卡的波特率为 9600、19200、38400

### 12.2 工作流程



## 12.3 函数接口

卡协议类型：

```
public static final int CARD_TYPE_T0;
```

```
public static final int CARD_TYPE_T1;
```

卡槽值：

```
public static final int CARD_SLOT_1;
```

```
public static final int CARD_SLOT_2;
```

卡波特率值：

```
public static final int CARD_BPS_9600;
```

```
public static final int CARD_BPS_19200;
```

```
public static final int CARD_BPS_38400;
```

卡参数：

```
public static final int CARD_VOLTAGE_1V8; //A 类卡
```

```
public static final int CARD_VOLTAGE_3V0; //B 类卡
```

```
public static final int CARD_VOLTAGE_5V0; //C 类卡
```

```
public static final int CARD_PPS_NOSUPPORT; //不支持 PPS
```

```
public static final int CARD_PPS_SUPPORT; //支持 PPS
public static final int CARD_PPS_ENFORCE; //强制 PPS, 可能无效
public static final int CARD_BPS_9600; //ATR 速度 9600
public static final int CARD_BPS_19200; //ATR 速度 19200
public static final int CARD_BPS_38400; //ATR 速度 38400
public static final int CARD_EMV_SPEC; //EMV 规格卡
public static final int CARD_ISO_SPEC; //普通 ISO7816 卡
public static final int CARD_NOCHECK_SPEC; //某些特别卡, 例如 SIM 卡
```

#### API 错误码:

<code>public static final int ERR_NODEV;</code>	<code>//读卡器未找到</code>
<code>public static final int ERR_PARAM ;</code>	<code>//参数错误</code>
<code>public static final int ERR_NOINIT;</code>	<code>//读卡器未初始化</code>
<code>public static final int ERR_SLOT;</code>	<code>//卡槽值错误</code>
<code>public static final int ERR_WRITE;</code>	<code>//写错误</code>
<code>public static final int ERR_READ;</code>	<code>//读错误</code>
<code>public static final int ERR_OOM;</code>	<code>//内存溢出</code>
<code>public static final int ERR_RESPONSE;</code>	<code>//读卡器答错误</code>
<code>public static final int ERR_RESPONSE_SUM;</code>	<code>//应答校验码错误</code>
<code>public static final int ERR_CARD_DETECT;</code>	<code>//没有卡</code>

```
public static final int ERR_CARD_OPEN;           //打开卡错误  
public static final int ERR_CARD_EXCHANGE;      //交换数据错误
```

## 函数列表：

### \* 读卡器初始化

`int init ()`

返回值： 0 - 初始化成功，负数为错误码

### \* 关闭读卡器

`void deinit()`



#### \* 打开卡

`int cardOpen(int slot, int config, byte[] atr)`

参数: `slot` 卡槽值

`config` 卡参数:

bit0-1: 卡工作电压参数

bit2-3: pps 参数

bit4-5:卡波特率参数

bit6-7:卡规格参数

`atr` 卡 ATR 值, 格式为 2 字节长度+ATR 字节数组

返回值:

0 - T0 卡, 1 - T1 卡, 负数为错误码

#### \* 关闭卡

`void cardClose(int slot)`

参数: `slot` 卡槽值

#### \* 检测卡

`int cardDetect(int slot)`

参数: `slot` 卡槽值

返回值: 0 - 有卡, 负数为错误码

\* 交换数据

```
int cardExchange(int slot, byte[] command, int cmdLen, byte[] response,  
int respLen, int timeoutMs)
```

参数：`slot` 卡槽值

`command` APDU 指令

`cmdLen` APDU 指令长度

`response` APDU 应答 buffer

`respLen` APDU 应答 buffer 长度

`timeoutMs` APDU 应答读取超时，单位毫秒

返回值：卡应答字节数组实际长度，负数为错误码

## 12.4 FAQ

SDK 1.3.7 版本开始使用新 SAM 卡模块，支持 Extended APDU。如果因为卡响应速度限制导致应答超时问题时，可以尝试增大参数 `timeoutMS` 来解决。

## 13.钱箱

### 13.1 概述

SmartPos 有两种连接钱箱的方法：

- 1.直接连接 RJ11(部分型号支持)
- 2.通过 USB 转 RJ11 连接

### 13.2 函数接口

\* 通过 RJ11 接口打开钱箱

`static void open()`

\* 通过 USB 转 RJ11 接口打开钱箱

`static void openEx()`

### 13.3 FAQ

## 14. 扫码器

### 14.1 概述

扫码器指用来识别条形码或者二维码的专用模块。扫码器有 USB 和串口两种接口，一般 USB 接口是 HID 模式，扫码结果以字符“\n”结尾，串口接口是通过串口输出结果。

### 14.2 FAQ

部分 USB 扫码模块可以通过设置码切换到虚拟串口模块，通过节点/dev/ttyACMx 读取扫码结果。

Demo 里只演示了如何读取 HID 模式的扫码结果，串口模式通过串口读取，具体参考扫码模块文档。

## 15.LED

### 15.1 概述

LED API 可以控制设备上的 LED，目前仅型号 FH101A1-D(CPU 版本 RK3568)支持 LED。

### 15.2 函数接口

函数列表：

\* 设置红色 LED

```
void setRedLed(boolean on)
```

参数：on true 打开，false 关闭

\* 设置绿色 LED

```
void setGreenLed(boolean on)
```

参数：on true 打开，false 关闭

\* 设置蓝色 LED

```
void setBlueLed(boolean on)
```

参数：on true 打开，false 关闭

## 16.附录

### 16.1 打印机固件恢复

参考 Demo 中 `fwUpdate(getAssets(), "fw.bin")`，当打印机无法正常工作时可以尝试调用这个函数来恢复打印机固件。升级耗时约 12 秒，升级过程请勿中断！

**注：**仅内置串口打印机支持

### 16.2 全屏模式

全屏模式下状态栏和导航栏将一直隐藏

```
Intent intent = new Intent();  
  
intent.setAction("com.paydevice.action.CONTROL_STATUSBAR");  
  
intent.putExtra("cmd", "hide");  
  
sendBroadcast(intent);
```

退出全屏模式

```
Intent intent = new Intent();  
  
intent.setAction("com.paydevice.action.CONTROL_STATUSBAR");  
  
intent.putExtra("cmd", "show");  
  
sendBroadcast(intent);
```