

CH934X 系列芯片串口 Android 程序开发说明

一、概述

本文档是针对 CH934X/CH348 系列的 USB 转串口安卓库的开发说明文档。

CH934X 串口提供的 Android 接口需要基于 Android 4.4 及以上版本系统，使用 CH934X 串口 Android 驱动条件：

- 1、基于 Android 4.4 及以上版本系统
- 2、Android 设备具有 USB Host 或 OTG 接口

二、接口说明

2.1. getInstance

```
public static CH934XManager getInstance()
```

用于获取全局唯一实例

返回	返回全局唯一实例
----	----------

2.2. init

```
public void init(android.app.Application application)
```

初始化上下文，注册动态广播监听设备状态变化

参数	application - 全局上下文
----	---------------------

2.3. enumDevice

```
public ArrayList<UsbDevice> enumDevice() throws UartLibException
```

枚举当前的 CH934X 设备

返回	设备列表
抛出	UartLibException

2.4. getChipType

```
public ChipType getChipType(@NonNull UsbDevice usbDevice)
```

根据 UsbDevice 获取芯片类型

参数	usbDevice - USB 设备
返回	如果为 null 则表示参数 USB 设备并不是 CH934X

2.5. openDevice

```
public boolean openDevice(@NonNull UsbDevice usbDevice)
```

```
throws . UartLibException, NoPermissionException, ChipException
```

打开设备

参数	usbDevice - USB 设备
返回	true 成功; false 失败

抛出	UartLibException NoPermissionException ChipException
----	--

2.6. requestPermission

```
public void requestPermission(@NonNullContext context,
                             @NonNullUsbDevice usbDevice)
    throws UartLibException
```

请求 USB 设备权限

参数	context - 上下文 usbDevice - USB 设备
抛出	UartLibException

2.7. setUsbStateListener

```
public void setUsbStateListener(@NonNullUsbStateChange usbStateListener)
    监听设备的状态变化
```

参数	usbStateListener - 设备状态监听回调
----	-----------------------------

2.8. getSerialCount

```
public int getSerialCount(@NonNull UsbDevice usbDevice)
    获取设备的串口数目
```

参数	usbDevice - USB 设备
返回	返回串口数目;如果为负,说明获取失败

2.9. setSerialParameter

```
public boolean setSerialParameter(@NonNull
    UsbDevice usbDevice,
    int serialNumber,
    int baud,
    int dataBit,
    int stopBit,
    int parityBit,
    boolean flow)
    throws .UartLibException, ChipException
```

设置串口参数

参数	usbDevice - USB 设备 serialNumber - 串口号 baud - 波特率 dataBit - 数据位 stopBit - 停止位 parityBit - 校验位 flow - 流控
----	--

返回	true 设置成功;false 设置失败
抛出	UartLibException ChipException

2.10. writeData

```
public int writeData(@NonNull
                    UsbDevice usbDevice,
                    int serialNumber,
                    byte[] data,
                    int length,
                    int timeout)
    throws UartLibException, ChipException
```

发送数据

参数	usbDevice - USB 设备 serialNumber - 串口号 data - 待发送的数据 length - 待发送的数据的长度 timeout - 超时时间
返回	发送成功的数据的长度
抛出	UartLibException ChipException

2.11. readData

```
public byte[] readData(@NonNull UsbDevice usbDevice, int serialNumber)
    throws ChipException
```

读取数据

参数	usbDevice - USB 设备 serialNumber - 串口号
返回	读取到的数据
抛出	ChipException

2.12. registerDataCallback

```
public void registerDataCallback(@NonNull UsbDevice usbDevice,
                                IDataCallback dataCallback) throws ChipException
```

注册串口数据回调, 解除注册使用 registerDataCallback(device, null) 方法, 或者 removeDataCallback(device) 方法

参数	usbDevice - USB 设备 dataCallback - 回调
返回	读取到的数据
抛出	ChipException

2. 13. removeDataCallback

```
public void removeDataCallback(@NonNull UsbDevice usbDevice)
```

解除注册串口数据回调

参数	usbDevice - USB 设备
----	--------------------

2. 14. setBreak

```
public boolean setBreak(@NonNull UsbDevice usbDevice, int serialNumber,
                        boolean valid) throws Exception
```

设置 Break 信号

参数	usbDevice - USB 设备 serialNumber - 串口号 valid - 是否有效（低电平标识有效）
返回	true 设置成功;false 设置失败
抛出	Exception

2. 15. setDTR

```
public boolean setDTR(UsbDevice usbDevice, int serialNumber, boolean dtr)
                        throws UartLibException, ChipException
```

设置 DTR 信号

参数	usbDevice - USB 设备 serialNumber - 串口号 dtr - true 有效;false 无效
返回	true 设置成功;false 设置失败
抛出	ChipException UartLibException

2. 16. setRTS

```
public boolean setRTS(UsbDevice usbDevice, int serialNumber, boolean rts) throws
                        UartLibException, ChipException
```

设置 RTS 信号

参数	usbDevice - USB 设备 serialNumber - 串口号 rts - true 有效;false 无效
返回	true 设置成功;false 设置失败
抛出	ChipException UartLibException

2. 17. registerModemStatusCallback

```
public void registerModemStatusCallback(@NonNull UsbDevice usbDevice,
                                         IModemStatus modemStatus) throws Exception
```

注册 Modem 输入信号状态的回调

参数	usbDevice - USB 设备 modemStatus - 状态回调
----	--

抛出	Exception
----	-----------

2.18. querySerialErrorCount

```
public int querySerialErrorCount(@NonNull UsbDevice usbDevice,
                                int serialNumber, @NonNull SerialErrorType errorType)
                                throws Exception
```

查询串口错误状态

参数	usbDevice - USB 设备 serialNumber - 串口号 errorType - 错误类型
返回	出现错误的次数
抛出	Exception

2.19. getCurrentMode

```
public Mode getCurrentMode(UsbDevice usbDevice, int serialNumber)
```

获取当前串口的模式。初始默认状态为普通模式（仅针对 CH934X 型号设备，CH348 无效）

参数	usbDevice - USB 设备
	serialNumber-串口序号
返回	Mode.NORMAL 普通模式；Mode.HARDFLOW 硬件流控模式； Mode.GPIO GPIO 模式；
抛出	UartLibException

2.20. getSpecificType

```
public SpecificChipType getSpecificType(UsbDevice usbDevice)
                                throws UartLibException
```

获取芯片具体型号

参数	usbDevice - USB 设备
返回	芯片具体型号
抛出	UartLibException

2.21. getGPIOCount

```
public int getGPIOCount(UsbDevice usbDevice) throws UartLibException
```

获取 GPIO 数量

参数	usbDevice - USB 设备
返回	GPIO 具体数量（从 0 开始使用）
抛出	UartLibException

2.22. getGPIOGroup

```
public int getGPIOGroup(UsbDevice usbDevice) throws UartLibException
```

获取 GPIO 组

参数	usbDevice - USB 设备
返回	GPIO 组数量（从 0 开始使用）
抛出	UartLibException

2.23. enableGPIO

```
public boolean enableGPIO(@NonNull UsbDevice usbDevice, ChipType chipType,
                          int gpioGroup, int enable) throws UartLibException
```

使能串口的 GPIO 功能，如果当前模式为硬件流控模式，需要先退出。

参数	usbDevice - USB 设备 chipType - 芯片类型 gpioGroup - GPIO 组号 enable - 使能状态 CH9344: 1 使能组内所有 GPIO;0 失能组内所有 GPIO CH348: bits0-7 对应 GPIO[0*N-7*N], 1 使能;0 失能
返回	true 操作成功;false 操作失败
抛出	UartLibException

2.24. setGPIDir

```
public boolean setGPIDir(@NonNull UsbDevice usbDevice, int gpioGroup, int
                        gpioNumber, @NonNull GPIO_DIR dir) throws UartLibException
```

设置 GPIO 口的方向，成功后会同步至缓存

参数	usbDevice - USB 设备 gpioGroup - GPIO 组号 gpioNumber - GPIO 序号 dir - 方向
返回	true 操作成功;false 操作失败
抛出	UartLibException

2.25. queryGPIDirFromCache

```
public GPIO_DIR queryGPIDirFromCache(@NonNull UsbDevice usbDevice, int
                                      gpioGroup, int gpioNumber) throws UartLibException
```

从缓存中获取某个 GPIO 的方向。初始每个 GPIO 初始默认方向是 GPIO_DIR.IN

参数	usbDevice - USB 设备 gpioGroup - GPIO 组号 gpioNumber - GPIO 序号
返回	GPIO_DIR.IN IN 方向;GPIO_DIR.OUT OUT 方向
抛出	UartLibException

2. 26. setGPIOValue

```
public boolean setGPIOValue(@NonNull UsbDevice usbDevice, int gpioGroup, int
    gpioNumber, @NonNull GPIO_VALUE value) throws UartLibException
```

设置 GPIO 口的值, 成功后会同步至缓存

参数	usbDevice - USB 设备 gpioGroup - GPIO 组号 gpioNumber -GPIO 序号 value - GPIO 值
返回	true 操作成功;false 操作失败
抛出	UartLibException

2. 27. getGPIOValue

```
public boolean getGPIOValue(@NonNull UsbDevice usbDevice) throws
    UartLibException
```

从硬件获取 GPIO 值, 成功后会刷新缓存

参数	usbDevice - USB 设备
返回	true 操作成功;false 操作失败
抛出	UartLibException

2. 28. queryGPIOValueFromCache

```
public GPIO_VALUE queryGPIOValueFromCache(@NonNull UsbDevice usbDevice, int
    gpioGroup, int gpioNumber) throws UartLibException
```

从缓存中获取某个 GPIO 值。使用此方法前需要先成功使用 getGPIOValue() 方法, 刷新缓存。初始每个 GPIO 初始默认值是 GPIO_VALUE. LOW

参数	usbDevice - USB 设备 gpioGroup - GPIO 组号 gpioNumber -GPIO 序号
返回	GPIO_VALUE. HIGH 高电平;GPIO_VALUE. LOW 低电平
抛出	UartLibException

2. 29. isConnected

```
public boolean isConnected(@NonNull UsbDevice usbDevice)
```

设备是否已经被打开

参数	usbDevice - USB 设备
返回	true 已经被打开;false 没有被打开

2. 30. getConnectedDevices

```
public ArrayList<UsbDevice> getConnectedDevices()
```

获取当前已经被打开的设备列表

返回	已经被打开的设备列表
----	------------

2.31. disconnect

```
public void disconnect(@NonNull UsbDevice usbDevice)
```

断开连接

参数	usbDevice - USB 设备
----	--------------------

2.32. close

```
public void close(@NonNull Context context)
```

释放资源。断开所有连接设备

参数	context -上下文
----	--------------