# JoyLink-Bluetooth
# 设备端 SDK 开发文档 V1.63
# 京东智能协议组

目录

修订记录：

| 版本号 | 修订人 | 修订日期 | 修订描述 |
| --- | --- | --- | --- |
| V 1.6.3 | | 2017.4.19 | 整理协议文档 |
| | | | |
| | | | |
| | | | |

# 1 简介

这份 SDK 文档是需要厂商重点阅读的。协议文档只是参考，是为了配合这个 SDK 文档而写，与此 SDK 相关处就仔细看，无关处可不必深究。

模块 SDK（即 JoyLink BLE SDK）主要用来方便设备端厂家开发自己的应用程序，快速接入微联。 模块端的 SDK 是根据 JoyLink-Bluetooth 协议实现设备与微联 APP 的交互，部分交互可能会有云端的参与。

设备端软件架构是：


图 1.1

SDK 对上层要提供接口函数给第三方 App，同时 SDK 也需要使用第三方 HAL 的接口函数。SDK 源代码由京东实现，第三方 HAL 和第三方 App 由设备厂商实现。

## 2 主要文件说明

SDK 主要内容包括：

名称：joylink_ble_dev_sdk_v1.0

目录结构：

./joylink_ble_dev_sdk_v1.0

├── joylink_hal.h　　HAL 层的函数以及数据结构定义

├── joylink_sdk.h　　SDK 层的函数以及数据结构定义

├── joylink_syshdr.h　　系统头文件定义，可以根据编译环境不同，配置是否使用某些基础库

└── target_XXX　　编译后生成文件

　　├── libjoylink.a　　库

target_XXX 中的 XXX 是设备厂商名称。

源代码是同一份，但是最终编译后生成文件要根据设备厂商提供的编译器不同而不同。

# 3 编译

SDK 采用 C 语言编写。 对于首次接入京东微联的厂商，需要提供编译器给京东，由京东使用厂商的编译器生成 libjoylink.a 给厂商。然后建议设备厂商和京东使用 libjoylink.a 进行一对一联调，这时需要设备厂商提供开发板和下载器等工具。

# 4 SDK 接口

SDK 接口指的是 SDK 提供给 App 层的接口函数。

SDK 接口函数由 JoyLink BLE SDK 实现，提供给设备 App 使用。调用方向单一，只能由设备 App 调用 SDK,而不能反向。这组 SDK 接口函数在功能上要具备高度的概括性，既要涵盖所有功能，又要划分科学不能重叠。另外不需要 SDK 处理的操作建议由 App 层直接调用底层硬件驱动完成，这样设计遵循的一个重要原则就是 SDK 极简原则：只有跟手机云端交互相关的操作，需要使用 SDK，本地操作不需要，由第三方 APP 直接使用第三方 Driver 进行本地的操作。


图 4.1

## (1) SDK 初始化

第三方 App 要想使用 JoyLink BLE SDK,第一件要做的事就是先申请一块合适自己的内存，然后把这个内存的地址传给 JoyLink BLE SDK。SDK 在运行期间是不能自己申请大内存的。

**函数名**：int jl_init(uint8_t* sendBuffer, uint16_t sendSize, uint8_t* rcvBuffer, uint16_t rcvSize)

**参数**：uint8_t* sendBuffer //第三方 App 层申请一块内存给 JoyLink BLE SDK 用于设备发送数据，
//注意：SDK 不能自己动态申请内存
uint16_t sendSize // sendBuffer 的大小，单位为字节
uint8_t* rcvBuffer //第三方 App 层申请一块内存给 JoyLink BLE SDK 用于设备接收数据，
//注意：SDK 不能自己动态申请内存
uint16_t rcvSize // sendBuffer 的大小，单位为字节

**返回值**：0: 执行成功
<0: 错误码

## (2) 接收帧

这个函数的输入是帧，输出是包。

手机和设备 App 层之间以包的形式通信。

JoyLink BLE SDK 支持全双工模式，即接收帧和发送包能同时进行。

SDK 接收到的是带 count 域和 num 域的原始帧，返回给 App 的是包。包放在第三方 App 层申请的一块内存 zone 中，从第 0 字节开始放置。

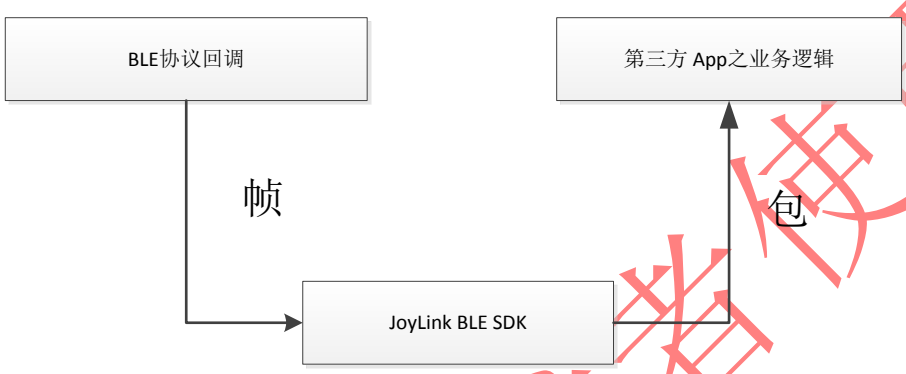


图 4.2

**函数名：** int jl_receive(uint8_t* frame)

**参数：** uint8_t* frame //第三方 App 接收到的帧

**返回值：** 如下表所示

| 宏 | 编号 | 含义 |
|---|---|---|
| RECEIVE_WAIT | 0x00 | SDK 期待下一个帧 |
| RECEIVE_END | 0x01 | SDK 已成功接收一包 |
| ERR_NUM_UNORDER | -201 | 帧顺序不对 |
| ERR_NUM_OVERFLOW | -202 | 帧序号超出最大值 |

## (3) 发送包（安全级别 0）

设备有什么数据想发送了，它不必担心数据长短、安全等，只需要通过本 API 丢给 SDK，由 SDK 负责发送等操作。

发送包时由 SDK 负责添加为凑够 20 字节而添加的 padding。

发送时常最多两秒，如果 2 秒钟之内发送仍旧没有成功完成，则 sdk 会向 app 层返回失败码，并重置 sdk。

如果当前不许发送，则立即返回错误码-2。

包一律是 seq+Operate + Length + Content+crc 的形式。

| seq | operate | length | content | crc |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Byte | n Byte | 2Byte |

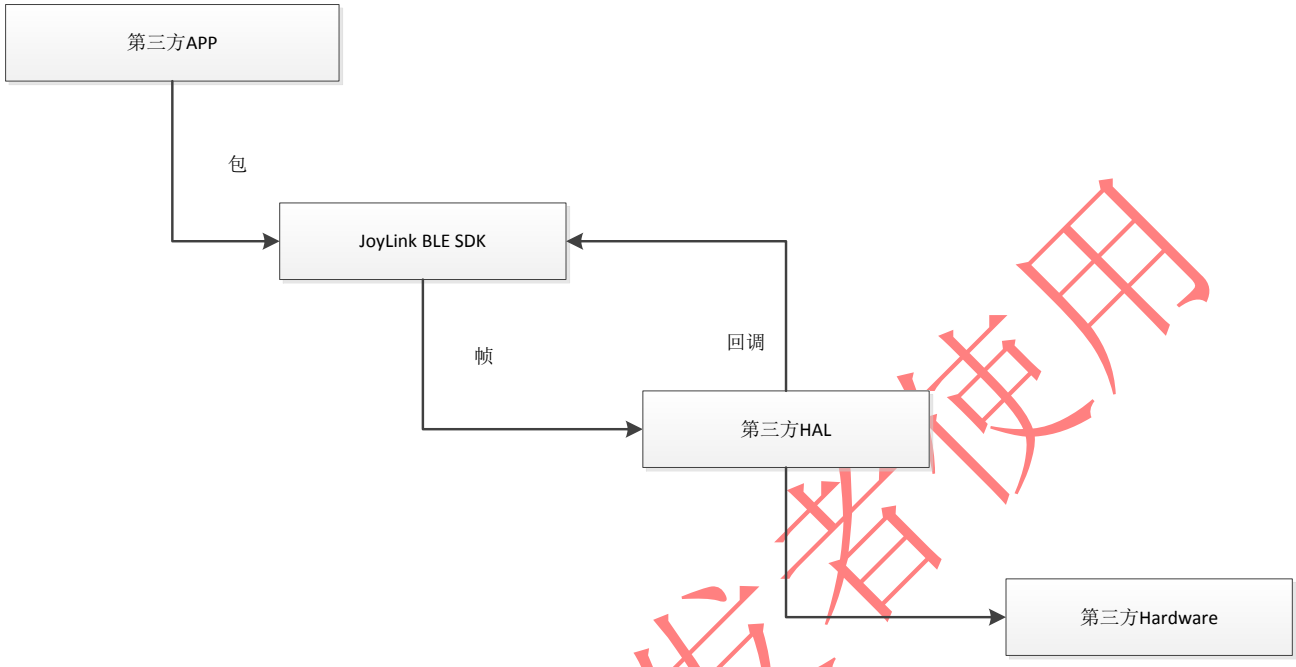


图 4.3

**函数名：** int jl_send_seclevel_0(uint16_t lenth, uint8_t* privData)

**参数：** uint16_t length //第三方 App 想发送包的长度，包的具体内容放置在 privData 内存中。

uint8_t* privData //要发送的 data

**返回值：** 0: 此包第一帧发送成功

<0: 错误码（-1：发送超时, -2:当前不许发送）。

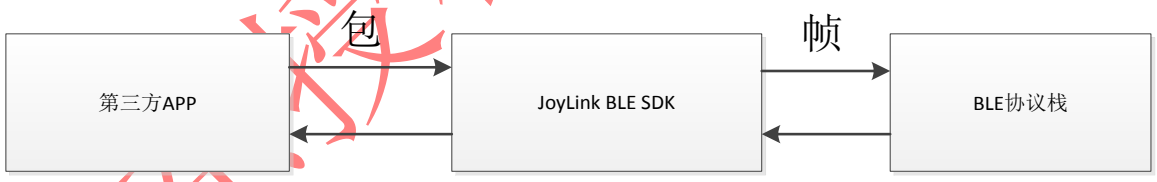


图 4.4

其中，在 SDK 发帧时 BLE 协议栈是 HAL,在收帧时则是一个普通的协议栈 API。



## (4) 发送包（安全级别 1）

设备有什么数据想发送了，它不必担心数据长短、安全等，只需要通过本 API 丢给 SDK，由 SDK 负责发送等操作。

发送包时由 SDK 负责添加为凑够 20 字节而添加的 padding。

发送时常最多两秒，如果 2 秒钟之内发送仍旧没有成功完成，则 sdk 会向 app 层返回失败码，并重置 sdk。

如果当前不许发送，则立即返回错误码-2。

包一律是 seq+Operate + Length + Content+crc 的形式。

| seq | operate | length | content | crc |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Byte | n Byte | 2Byte |

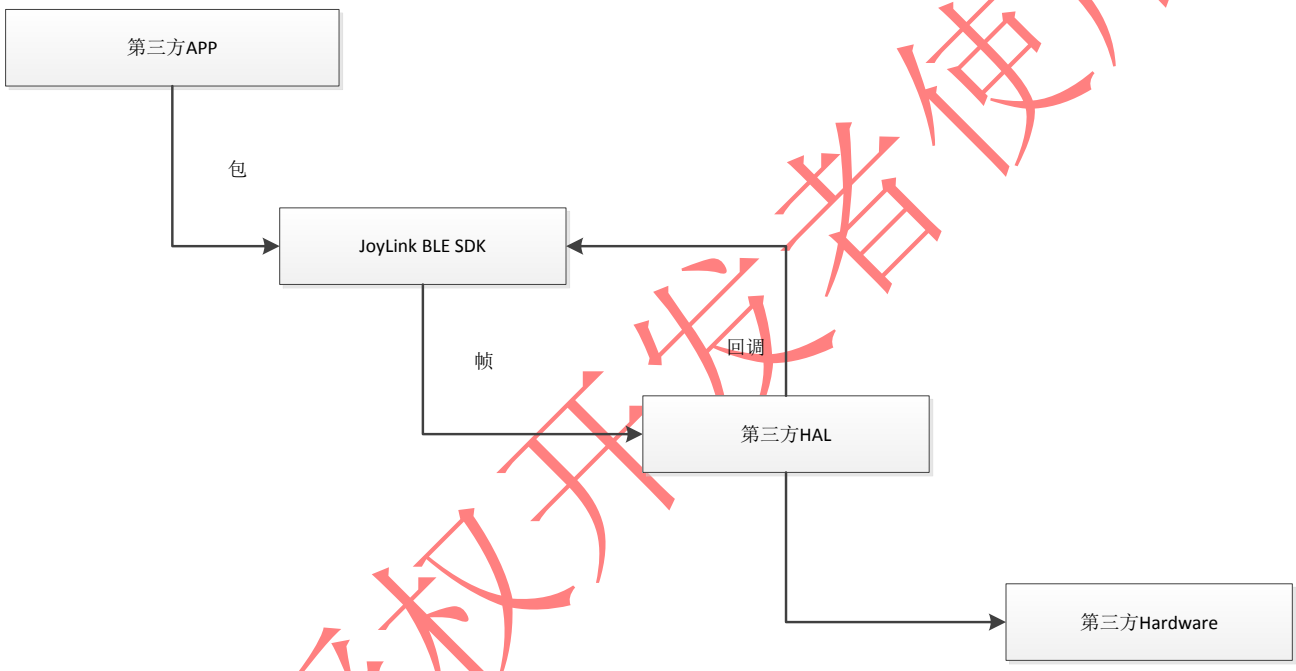


图 4.3

**函数名：** int jl_send_seclevel_1(uint16_t lenth, uint8_t* privData)

**参数：** uint16_t length　//第三方 App 想发送包的长度，包的具体内容放置在 zone 内存中。

uint8_t* privData　//要发送的 data

**返回值：** 0:　此包第一帧发送成功

<0: 错误码（-1：发送超时, -2:当前不许发送）。


图 4.4

其中，在 SDK 发帧时 BLE 协议栈是 HAL,在收帧时则是一个普通的协议栈 API。

## (5) 发送包（安全级别 2）

设备有什么数据想发送了，它不必担心数据长短、安全等，只需要通过本 API 丢给 SDK，由 SDK 负责发送等操作。

发送包时由 SDK 负责添加为凑够 20 字节而添加的 padding。

发送时常最多两秒，如果 2 秒钟之内发送仍旧没有成功完成，则 sdk 会向 app 层返回失败码，并重置 sdk。

如果当前不许发送，则立即返回错误码-2。

包一律是 seq+Operate + Length + Content+crc 的形式。

| seq | operate | length | content | crc |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Byte | n Byte | 2Byte |

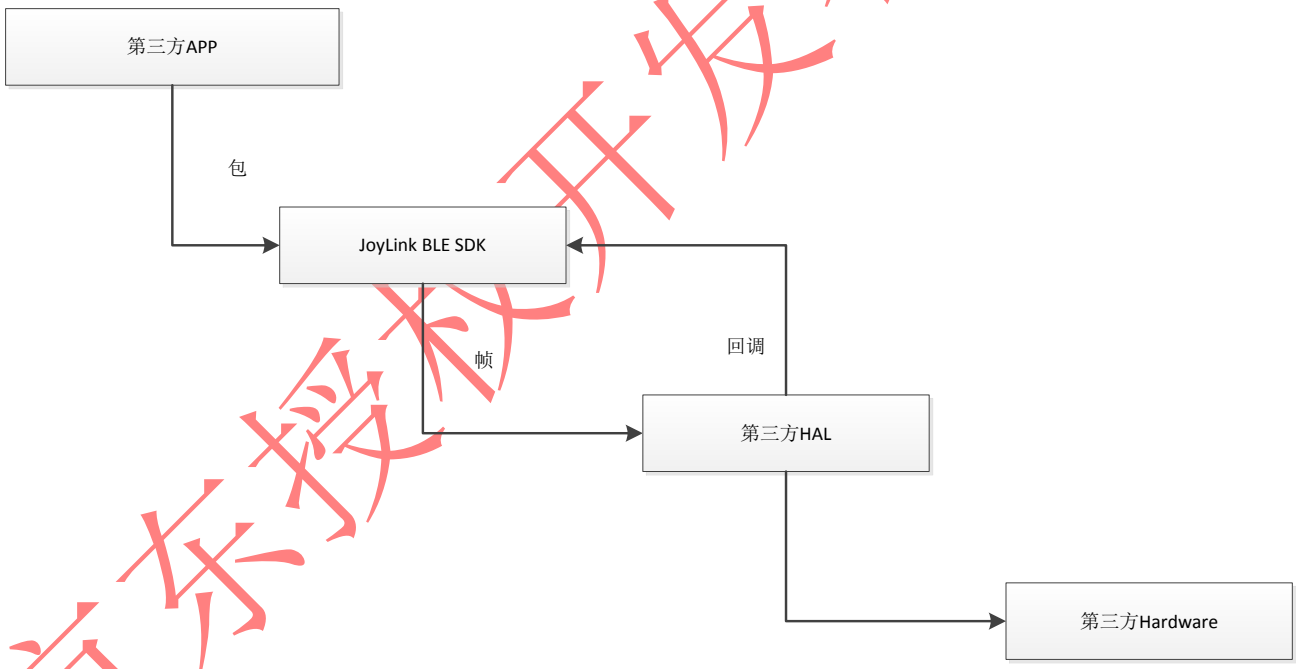


图 4.5

**函数名：** int jl_send_seclevel_2(uint16_t lenth, uint8_t* privData)

**参数：** uint16_t length　//第三方 App 想发送包的长度，包的具体内容放置在 zone 内存中。

uint8_t* privData　//要发送的 data

**返回值：** 0:　此包第一帧发送成功

<0: 错误码（-1：发送超时, -2:当前不许发送）。


图 4.6

其中，在 SDK 发帧时 BLE 协议栈是 HAL,在收帧时则是一个普通的协议栈 API。

## (6) 发送包（安全级别 3）

设备有什么数据想发送了，它不必担心数据长短、安全等，只需要通过本 API 丢给 SDK，由 SDK 负责发送等操作。

发送包时由 SDK 负责添加为凑够 20 字节而添加的 padding。

发送时常最多两秒，如果 2 秒钟之内发送仍旧没有成功完成，则 sdk 会向 app 层返回失败码，并重置 sdk。

如果当前不许发送，则立即返回错误码-2。

包一律是 seq+Operate + Length + Content+crc 的形式。

| seq | operate | length | content | crc |
|---|---|---|---|---|
| 1 Byte | 1 Byte | 2 Byte | n Byte | 2Byte |

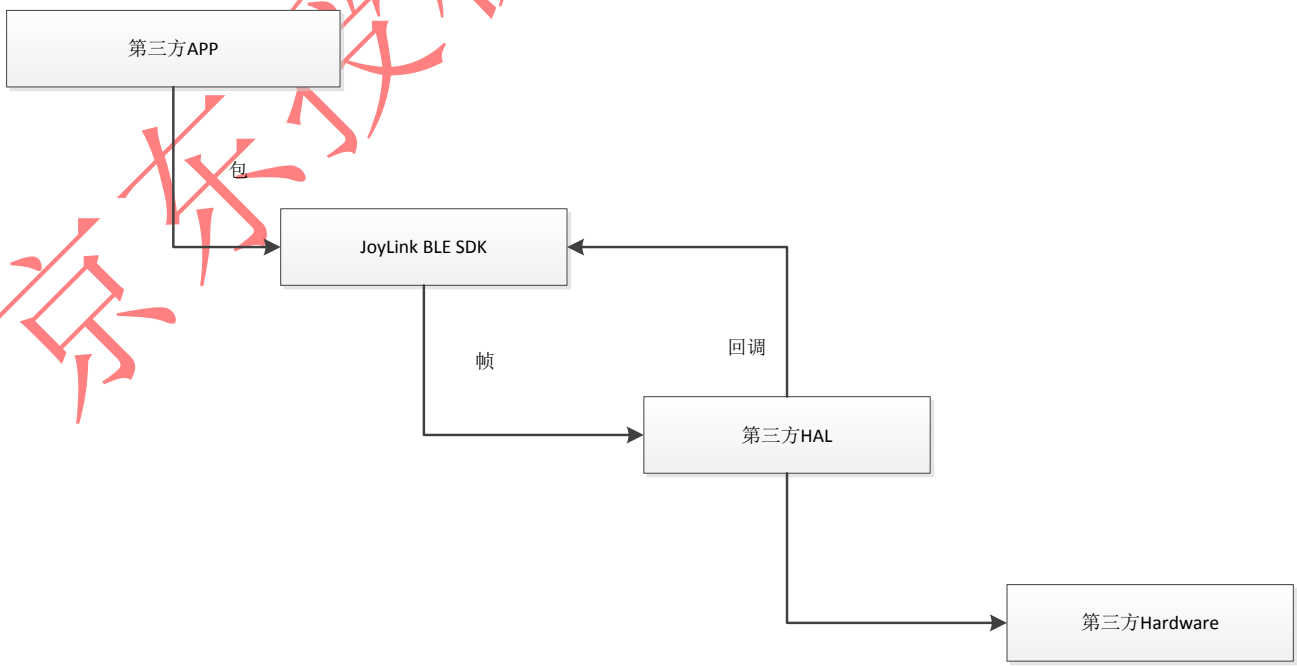


图 4.7

**函数名：** int jl_send_seclevel_3(uint16_t lenth, uint8_t* privData)

**参数：** uint16_t length　//第三方 App 想发送包的长度，包的具体内容放置在 zone 内存中。

uint8_t* privData　//要发送的 data

**返回值：** 0:　此包第一帧发送成功

<0: 错误码（-1：发送超时, -2:当前不许发送）。


图 4.8

其中，在 SDK 发帧时 BLE 协议栈是 HAL,在收帧时则是一个普通的协议栈 API。

## (7)　复位 Receive Buffer

App 强制复位 Receive Buffer。

当 SDK 在成功收到一包数据并回传给 App，App 使用完这包数据，这包数据不再有保存价值之后，App 应当调用此函数。

**函数名：** int jl_rcv_reset(void)

**参数：** 无

**返回值：** 0:　reset 成功，且没有不良后果。

<0:　reset 成功，但是会将以前收到的帧全部丢弃。

## (8)　保存 GUID

建议（非强制要求）GUID 和 localley 保存在 flash 的 sector 0 中,布局如下表：

| offset | 0 |
|---|---|
| data | GUID |
| length | 32 |

当设备激活时，手机客户端要将云端为这个设备生成的 feed id 写死到设备的 flash 中，此后一成不变，永不改变。

**函数名：** int jl_save_feedid(unsigned char *guid)

**参数：** unsigned char *feedid //32 字节的 GUID
**返回值：** 0: 执行成功
<0: 错误码

## (9) 获取 GUID

App 层获取存储在 flash 中的 feedid。
**函数名：int jl_get_guid (unsigned char *guid)**
**参数：** unsigned char *guid 存储 guid 的内存空间地址
**返回值：** 1 成功
0 失败

## (10) 保存 local key

当设备第一次绑定到一个账户下或者解绑后再次绑定时，会生成一份该设备的 accesskey 给手机客户端，手机客户端据此生成该设备绑定到该账户的 localkey。
**函数名：** int jl_save_locallkey (unsigned char*localkey)
**参数：** unsigned char *localkey //16 字节的 local key 存放的地址空间
**返回值：** 0: 执行成功
<0: 错误码

## (11) Indication 回调函数

HAL 发送帧是通过 characteristic 0xFE72 这个 indication characteristic 发送的。当 HAL 发送一帧成功，设备会收到手机端的 confirm。设备对 confirm 的处理中必须调用一次这个函数。
**函数名：int jl_indication_confirm_cb(void)**
**参数：** 无
**返回值：** 0: 正常
-1：数据包在发送过程中出现失败

## (12) ECDH 安全准备函数

当使用安全级别 3 时，如果给手机发送了设备公钥，并且从手机侧获得了手机公钥，这时需要调用一下本函数。
**函数名：int jl_secure_prepare(void)**
**参数：** 无
**返回值：** 0：准备失败
1：准备成功

### (13) 两个数据结构

在 app 层要声明这两个变量：

```
extern uint8_t compressed_dev_pub_key[21];
extern uint8_t compressed_app_pub_key[21];
```

compressed_app_pub_key 用于将手机公钥从 app 层传到 sdk 层；compressed_dev_pub_key 用于将设备公钥从 sdk 层传到 app 层

## 5 HAL 接口

HAL 接口指的是 HAL 层提供给 SDK 层的接口函数。

HAL 接口函数由第三方设备厂商 HAL 层实现，提供给 JoyLink BLE SDK 使用。调用方向单一，只能由设备 SDK 调用 HAL,而不能反向。这组 HAL 接口函数在功能上要具备高度的概括性涵盖所有功能，又要划分科学不能重叠。而且要有极强的适应性，一组 HAL 接口要适应任何一种 BLE 设备。

### (1) 发送帧

设备通过 BLE 信道将这些数据发送出去。

**函数名**：int jh_send(uint8_t* frame)
**参数**： uint8_t* frame //SDK 想发送的帧，长度固定， 20 Byte。
**返回值**：0: 执行成功
<0: 错误码

### (2) 获取 MAC 地址

**函数名**：int jh_load_mac(uint8_t * zone);
**参数**： zone, 用于放置 mac 地址的内存的指针
**返回值**：0: 执行成功
<0: 错误码

### (3) 获取 PUID

**函数名：** int jh_load_puid(uint8_t *puid);
**参数：** puid，用于放置 PUID 的内存的指针
**返回值：** 0: 执行成功
<0: 错误码

## (4) 打日志

SDK 打印日志。
**函数名：** void jh_logf(const char* fmt, ...);
**参数：** fmt，日志内容字符串
**返回值：** 无

## (5) 从 flash 读取 GUID

SDK 需要从 flash 中读取 GUID 时可以调用这个函数。
**函数名：** int jh_get_guid(unsigned char *guid);
**参数：** unsigned char *guid //32 字节的 guid 存放的地址空间
**返回值：** 0: 执行成功
<0: 错误码

## (6) 向 flash 写入 GUID

当设备激活时，手机客户端要将云端为这个设备生成的 guid 写死到设备的 flash 中。

**函数名：** int jh_save_guid(unsigned char *guid)
**参数：** unsigned char *guid //32 字节的 guid 存放的地址空间
**返回值：** 0: 执行成功
<0: 错误码

## (7) 从 flash 读出 local key

**函数名：** int jh_get_locallkey (unsigned char*localkey)
**参数：** unsigned char *localkey //16 字节的 local key 存放的地址空间
**返回值：** 0: 执行成功
<0: 错误码

## (8) 向 flash 写入 local key

**函数名：** int jh_save_locallkey (unsigned char*localkey)
**参数：** unsigned char *localkey //16 字节的 local key 存放的地址空间
**返回值：** 0: 执行成功
<0: 错误码

## (9) 检查 indication confirm 是否收到

**函数名：** int jh_check_ready_indication(void);
**参数：** 无
**返回值：** 0: indication 还没有发送成功，并且没收到手机侧的 confirm，需要继续等待
1: indication 发送成功，并且收到了手机侧的 confirm，可以继续发送下一帧

# 6 数据结构及定义

基本数据结构是由数据格式决定的。

## (1) Property 相关数据结构

```
//属性
typedef struct{
    //属性标签
    Uint16_t tag;

    //属性长度
    Uint8_t len;

    //属性值，长度自定
    Uint8_t value[];
}jl_property_t;
```

## (2) Packet 相关数据结构

```
//包
typedef PACKED struct
{
    uint8_t seq;            //包序号
    uint8_t operation;      //操作

    uint16_t len;           //长度
    uint8_t content[];      //内容，长度自定
    uint16_t crc;           //循环校验码
}JD_APP_PACKAGE_T;
```

## (3) 宏定义

```
#define ERR_NUM_UNORDER    (-201)       帧顺序不对
#define ERR_NUM_OVERFLOW (-202)         帧序号超出最大值
#define ERR_UNSUPPORT_KEY (-203)        key 的编码不对

#define RECEIVE_WAIT     0     继续接收帧
#define RECEIVE_END      1     一包数据接收完毕
```