

JoyLink v2.0.4

模块端 SDK 使用说明

京东智能系统研发组

本文档可能包含公司技术机密以及其他需要保密的信息，本文档所包含的所有信息均为北京京东智能集团公司版权所有。未经本公司书面许可，不得向授权许可方以外的任何第三方泄露本文档内容，不得以任何形式擅自复制或传播本文档。若使用者违反本版权保护的约定，本公司有权追究使用者由此产生的法律责任。

目录

1. 简介.....	1
1.1. 资源占用.....	1
1.2. 主要文件.....	1
1.3. 主要信息.....	2
1.4. 开发者配置注意事项.....	2
1.5. 设备与云互认注意事项.....	3
2. 编译.....	6
3. 开发者要实现的接口	7
4. 激活绑定	12
5. 调试.....	14
6. 一键配置相关接口	15

1. 简介

模端的 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 JoyLink2.0 协议实现设备与微联 APP 通过局域网、远程云的交互。包括设备发现、feedid、accesskey、localkey 写入、获取设备快照，控制设备等功能。文档中以一个灯为案列，展示 JoyLink 协议在设备端的数据交互过程。在开发过程中建议，首先在 PC 端开发调试，（如在 ubuntu 下），将云端，局域网内的协议调试通过。其次将调试通过的代码移植到设备中去。此 SDK 是在 ubuntu-14-10 的环境下调试通过的。

1.1. 资源占用

Joylink2.0_dev_sdk 在 ubuntu 运行时资源占用情况:

包括子设备部分:

ram: 4-6K

rom: 184K

其中的代码可以根据设备特性裁剪，如没有子设备，则将 xxx_sub_dev.c 踢出编译，减小代码空间。

1.2. 主要文件

SDK 主要内容包括:

名称: joylink_dev_sdk_v2.0.x

目录结构:

./joylink_dev_sdk_v2.0

—— auth	加解密头文件，实现以库的形式提供。
—— example	设备相关的逻辑处理参考案例
—— joylink	协议相关的逻辑处理
—— json	cJSON 相关的解析打包
—— lua	lua 脚本的参考源码
—— target	编译后生成文件
—— bin	编译后生成的可执行文件
—— lib	编译后生成的静态、动态库

设备厂商和模块只关注 **example** 下的文件，其余的都是协议相关的流程。

1.3. 主要信息

设备端必须要实现如下信息的存储和获取的接口：

关键词	描述
feedid	设备识别码,由云端分配，APP 绑定设备时写入设备
accesskey	设备远程通讯认证时加密的 key，APP 绑定设备时写入设备
localkey	局域网通讯 key，APP 绑定设备时写入设备
uuid	设备品类的识别码
version	设备的固件版本
serverinfo	智能云的域名和端口如 live.smart.jd.com:2002

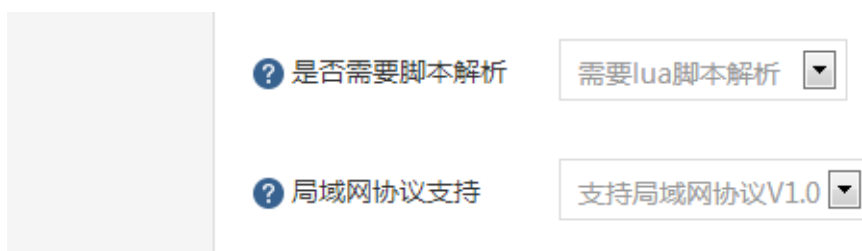
1.4. 开发者配置注意事项

注意：在开发者中心我们建议采用支持 **lua** 的方式控制设备，**sdk** 中默认的 **joylink_dev_sdk.c** 代码头部

`.jlp.trantype = 1` 默认支持 **lua**，所以参照 **sdk/lua/** 下的脚本编辑设备自己的脚本，并上传到开发者中心，不然会出现云端链接不了的情况。如果设备自身也支持 **json** 格式的控制，建议上传 **sdk/lua/only_trans.lua**,这个 **lua** 是个空实现，仅为匹配流程，便于以后扩展。

开发者中心配置如图：

选择支持 **lua** 脚本



上传 **lua** 脚本



上传 HTML5

sdk/example/index.zip 是 html5 的一个开发包，这是灯的一个例子，可以上传后先体验再修改。体验灯的例子时候，开发者中心注册的 streamid 名字是”power”，类型是 int。

1.5. 设备与云互认注意事项

为加强智能云平台的安全，引入设备与云端的互认流程，流程在 SDK 中已经实现，厂商需要做的是如下事项：

获取云端公钥

在开发者中心 <https://smartdev.jd.com>，成功注册产品后，在“基本信息”栏，如下图，有”安全认证>公钥信息”这一栏，这是云端颁发给设备的云端公钥信息，以字符串类型存放。将此公钥信息妥善保管，如有泄露，注册的设备有被伪造的风险，所有后果需要注册者承担。



SDK 中存放公钥

从开发者中心得到 SDK 后，在 SDK/example/joylink_extern.c 中实现 joylink_dev_get_idt
() 这个方法，
SDK 中已经有灯设备的例子，按照例子来实现即可。

eg:

```
LightManage_t _g_lightMgr = {
    .conn_st = -1,
    .jlp.mac= "70:55:44:33:22:11",
    .jlp.devtype = E_JLDEV_TYPE_NORMAL,
    .jlp.version = 1,
    .jlp.uuid = "5HVVWPT",
    .jlp.lancon = E_LAN_CTRL_ENABLE,
    .jlp.cmd_tran_type = E_CMD_TYPE_LUA_SCRIPT,
    .idt.cloud_pub_key=
"0350C987CFDD70B5A21EC16617D74EC0A5AF139B8510124FAA4072E99DAF5AF121",
    .lightCtrl.cmd = LIGHT_CMD_NONE,
    .lightCtrl.para_value = LIGHT_CTRL_OFF
};
```

```
LightManage_t *_g_pLightMgr = &_amp;_g_lightMgr;
```

```
/**
 * brief:
 * @Param: jlp
 *
 * @Returns:
 */
```

```
E_JLRetCode_t
joylink_dev_get_idt(jl2_d_idt_t *pidt)
{
    if(NULL == pidt){
        return E_RET_ERROR;
```

```
    }  
    strcpy(pidt->pub_key, _g_pLightMgr->idt.pub_key);  
    strcpy(pidt->sig, _g_pLightMgr->idt.sig);  
    strcpy(pidt->rand, _g_pLightMgr->idt.rand);  
    strcpy(pidt->f_sig, _g_pLightMgr->idt.f_sig);  
    strcpy(pidt->f_pub_key, _g_pLightMgr->idt.f_pub_key);  
    strcpy(pidt->cloud_pub_key, _g_pLightMgr->idt.cloud_pub_key);  
    return E_RET_OK;  
}
```

2. 编译

SDK 的编译采用 `make`，管理编译文档，可以在各个路径下独立编译测试，详细请参考 `Makefile`。`Makefile.rule` 是配置基本的编译规则。

编译步骤：

1 修改配置文件 `Makefile.rule`

修改 SDK 在 PC 端的路径。

```
PROJECT_ROOT_PATH:=/home/steven/joylink_dev_sdk_v2.0
```

2 编译

```
make
```

3 运行

生成可执行文件

```
sudo target/bin/jt
```


3. 开发者要实现的接口

设备只关注 example 下的 joylink_extern.c joylink_extern_sub_dev.c joylink_extern.h 文件。实现标注“todo”的空接口。以下文档主要说明接口作用。

E_JLRetCode_t

joylink_dev_is_net_ok()

功能描述：检查设备是否能连接外网。

E_JLRetCode_t

joylink_dev_set_connect_st(int st);

功能描述：SDK 通知设备与云端连接的状态。

E_JLRetCode_t

joylink_dev_set_attr_jlp(JLPInfo_t *jlp);

功能描述：存储协议相关的 feedid, accesskey, localkey 等信息。

JLPInfo_t 所有字段都要存储。

typedef struct {

int isUsed;

short version;

char ip[JL_MAX_IP_LEN];

int port;

char mac[JL_MAX_MAC_LEN];

char uuid[JL_MAX_UUID_LEN];

int lancon; // 1 suport Lan

int cmd_tran_type; // 0:bin, 1:Lua, 2:Js

int devtype;

int protocol; // 0:WIFI 1:zigbee 2:bluetooth 3:433

char feedid[JL_MAX_FEEDID_LEN];

char accesskey[33];

char localkey[33];

char devdbg[JL_MAX_DBG_LEN];

char servropt[JL_MAX_OPT_LEN];

```

char joylink_server[JL_MAX_SERVER_NAME_LEN];
int server_port;

char CID[10];
char firmwareVersion[10];
char modelCode[66];
char is_activated;

}JLPInfo_t;

```

注意事项：存储的这些信息，重新上电后能够正确获得。

注意事项：JLPInfo_t 中有 is_activated 字段，SDK 通过此字段判断设备是否进入到待激活状态，只有当进入到待激活状态设备才能接受激活指令。激活成功后 SDK 将此字段设置为 1。设备刚出厂后或者用户与设备进行物理操作，按照设备定义的方式(如组合按键)进入待激活状态，设备将此字段设置为 0，以待设备被激活。

E_JLRetCode_t

```
joylink_dev_get_jlp_info(JLPInfo_t *jlp);
```

功能描述：获取协议相关的 feedid, accesskey, localkey 等信息。

E_JLRetCode_t

```
joylink_dev_set_attr(XXX_t *wi);
```

功能描述：存储设备相关的属性等信息，例如：冰箱的温度，湿度等。

注意事项：与设备相关，数据结构需要依据设备而定。

int

```
joylink_dev_get_snap_shot(char *out_snap, int32_t out_max);
```

功能描述：获取设备快照。

注意事项：要注意判断 out_max, 不能内存越界。

int

```
joylink_dev_get_json_snap_shot(char *out_snap, int32_t out_max, int code, char *feedid);
```

功能描述：获取 json 格式的设备快照

注意事项：json 格式要正确，请在 json.net 网站验证。

E_JLRetCode_t

joylink_dev_lan_json_ctrl(const char *json_cmd);

功能描述： 局域网 json 格式的控制指令控制

E_JLRetCode_t

joylink_dev_script_ctrl(const char *cmd, JLContrl_t *ctr, int from_server);

功能描述： 局域网脚本转换后的控制指令控制,cmd 是设备上传的 lua 转化后的二进制。

E_JLRetCode_t

joylink_dev_sub_add(JLDevInfo_t *dev, int num);

功能描述： 子设备添加

注意事项： 子设备信息重新上电后能够获得

E_JLRetCode_t

joylink_sub_dev_del(JLDevInfo_t *dev, int num);

功能描述： 子设备删除

注意事项：

E_JLRetCode_t

joylink_dev_sub_get_by_feedid(char *feedid, JLDevInfo_t *dev);

功能描述： 通过 feedid 获得子设备的信息。

E_JLRetCode_t

joylink_sub_dev_get_by_uuid_mac(char *uuid, char *mac, JLDevInfo_t *dev);

功能描述： 通过 uuid 和 mac 获得子设备信息。

注意事项：

E_JLRetCode_t

joylink_dev_sub_update_keys_by_uuid_mac(char *uuid, char *mac, JLDevInfo_t *dev);

功能描述： 通过 uuid 和 mac 来更新 accesskey, localkey, feedid 等信息，设备绑定的时候用。

注意事项： 子设备信息重新上电后能够正确获得。

JLDevInfo_t *

joylink_dev_sub_devs_get(int *count, int scan_type);

功能描述： 通过 scan_type 获得设备信息。

注意事项： 获取三类设备信息：所有设备，等待配置，已经配置。

E_JLRetCode_t

joylink_dev_sub_ctrl(const char* cmd, int cmd_len, char* feedid);

功能描述： 控制子设备

注意事项：

char *

joylink_dev_sub_get_snap_shot(char *feedid, int *out_len);

功能描述： 获取子设备快照

注意事项： 返回的是 malloc 的 char*， 要返回长度 *out_len = 长度。

E_JLRetCode_t

joylink_dev_sub_unbind(const char *feedid);

功能描述： 子设备解绑

注意事项： 将子设备 feedid 清空

E_JLRetCode_t

joylink_dev_ota(JLOtaOrder_t *otaOrder);

功能描述： 设备升级

注意事项： 如果设备类型为网关，则要考虑是否本地是否由于相应固件。如果有，则不需要重新下载。产品（模块）端收到升级指令从固件资源端下载固件采用 http 协议。

void

joylink_dev_ota_status_upload();

功能描述： 设备升级状态上报

注意事项： 设备升级状态有变化时进行升级状态主动上报。下载完固件后，须计算固件的 crc32 的值是否正确，如果正确说明固件下载成功，否则升级失败，可以在 status_desc 字段中详细描述状态的相关描述信息。厂商必须保证无论下载失败还是安装失败，都不能影响设备正常使用。

int

joylink_dev_register_attr_cb(
 const char *name,

```
E_JL_DEV_ATTR_TYPE type,  
E_JL_DEV_ATTR_GET_CB attr_get_cb,  
E_JL_DEV_ATTR_SET_CB attr_set_cb);
```

功能描述：通过注册回调的方式，管理设备属性，参考 example 中的案例。

4. 激活绑定

激活绑定是用户拿到设备后通过 APP 给设备授权，并将设备与账号建立映射的过程。具体过程如下：

- (1) 设备上电。
- (2) 用户扫描设备二维码（或者从 APP 三级类目中找到二维码进行扫描）。
- (3) 用户根据 APP 页面提示的操作，操作设备，例如组合按键，使设备进入到配网模式和待激活模式。
- (4) APP 开始激活绑定设备。
- (5) 绑定成功后出现设备列表。
- (6) 进入详情页后可操控设备。

在此过程中，用户必须要物理操作设备，让设备进入待激活状态，只有在这个状态下设备才接受 APP 的激活指令。测试过程会专项测试这个流程。如果设备需要配网，在用户进行上述操作后设备也必须进入配网状态，设备接受 APP 下发的网络信息。如果设备自身设计直接可联网则不必进入入网状态。

设备绑定后，如果用户还需要再次激活绑定设备，必须按照上述流程，特别是需要有(3)中的物理操作流程。

其中涉及到设备 SDK 中需要厂商修改和完善以下两个接口。

`E_JLRetCode_t`

`joylink_dev_set_attr_jlp(JLPInfo_t *jlp);`

这个接口在设备激活的时候调用，目的是存储协议相关的 `is_activated`, `feedid`, `accesskey`, `localkey` 等信息到 flash 中。`JLPInfo_t` 所有字段都要存储，具体字段见上文中的描述。激活绑定成功后 `JLPInfo_t` 的 `is_activated` 字段，被设置为 1，并存储在内存中。通过调用此接口将所有 `JLPInfo_t` 中相关的值保存到 flash 中。

`E_JLRetCode_t`

`joylink_dev_get_jlp_info(JLPInfo_t *jlp);`

功能描述：从 flash 获取协议相关的 `is_activated`, `feedid`, `accesskey`, `localkey` 等信息。

设备刚出厂后或者用户操作设备让设备进入待激活状态，设备要修改 flash 中 `is_activated` 的值为 0，以待设备被激活。

注意：设备进入待激活状态是要将全局变量 `_g_pdev->jlp.is_activated` 也设置为 0。如果设备是通过让设备回复出厂设置的方式进入待激活状态，这个全局变量自动设置为 0。如果不是上述方式，则需要厂商特殊设置为 0。

5. 调试

建议调试的时候现在 pc 端调试整个协议，可以造假数据，协议流程调试通过后再移植到设备。下图是调试时候抓包的图，可以分析其中 type 字段看到数据在手机，云端，设备端的交互。

0000	88 01 30 00 c8 3a 35 18 12 13 a8 8e 24 12 9c 33	..0...:5.\$.3
0010	c8 3a 35 11 12 13 80 d5 00 00 aa aa 03 00 00 00	.:5.....
0020	08 00 45 00 00 3c 1e 03 00 00 40 11 da f6 c0 a8	..E...<.. ..@.....
0030	00 66 c0 a8 00 01 fc ee 00 50 00 28 64 5f bb 55	.f..... .P.(d..U
0040	34 12 00 00 10 00 01 04 00 00 03 00 43 06 1a 59	4..... ..C..Y
0050	dd 48 78 e8 79 58 de 1a 00 b6 7a bc 8f 61 00 00	.Hx.yX.. ..Z..a..
0060	00 00	..

Wireshark 抓包分析 payload 对应的从绿色 bb 开始就是 JLPacket_t 对应的数据，第 10 个字节就是 04 对应的包体内容就是 PT_SCRIPTCONTROL，说明此包是控制报文。在调试协议的时候用此方式可分析报文的交互。

6. 一键配置相关接口

一键配置 joylink 这边只提供相应的库，具体使用的时候联系产品，厂商提供交叉编译环境，我们交叉编译后提供库。

int

joylink_cfg_init();

功能描述： 一键配置初始化

注意事项：

joylink_cfg_status_t

joylink_cfg_recv (

unsigned char *da,

unsigned char *sa,

int len,

void *user_data

);

功能描述： 处理空中抓取到的数据包

注意事项：

int

joylink_cfg_result(joylink_cfg_result_t *result);

功能描述： 获取一键配置结果

注意事项：

char*

joylink_cfg_dinit (void);

功能描述： 获取当前库的版本信息

注意事项：