JeeSDK 开发者参考手册

MCU 平台

中科物栖（北京）科技有限责任公司

软件中心

文档修订记录

版本	修订日期	修订原因	修订人	审核人
V1.0	2021/07/01	初始撰写	白东洋	
V1.1	2021/08/06	根据正式发布的SDK 修改代码示例	白东洋	
V1.2	2022/03/17	添加上报，BLE 配网，API 索引表	白东洋	
V1.3	2022/04/6	添加加密工具使用说明	山晓妍	

目录

JeeSDK 开发者参考手册 1
文档修订记录 2
目录 3
撰写说明 5
设备接入 5
设备配网 7
    AP 配网 7
    BLE 配网 14
接收命令 17
    原理 17
    实现 18
    示例 19
上报状态 22
    概述 22
    实现 23
    示例 23
OTA 升级 25
    平台配置 25
    OTA 流程 26
    基于 SDK 的开发示例 27

设备烧写序列号及密钥 30
示例 33
API 索引表 33

撰写说明

本文档示例代码均基于 BK7231 开发板。

设备接入

JeeSDK 是物栖 OS 提供的设备端软件开发工具包，可简化开发过程，实现设备快速接入物栖 OS。厂商获取 SDK 后，根据需要选择相应功能进行开发，即可快速集成 JeeSDK，实现设备的接入。

厂商需要通过以下几个步骤，将设备接入到平台；

一、 产品功能定义

登录到**中科物栖设备接入开放平台**（简称：设备开放平台），选择一个品类创建产品，从品类标准功能库中选择需要的功能，也可以自定义产品支持的个性化功能；

二、 移动端交互设计

中科物栖移动应用客户端（即：冒泡物联 App）可用于远程控制智能设备。厂商通过面板定义和命令定义，可实现在冒泡物联 App 上的远程控制功能设计。

1） 面板的定义

厂商可以选择使用已定义的标准交互界面面板，也可以基于面板的开发规则自定义交互界面并开发；

（登录设备开放平台，通过产品开发模块，选择或定义面板）

2） 命令的定义

厂商可以选择使用已定义的标准控制命令集，也可以基于规则自定义个性化命令；

（登录设备开放平台，通过产品开发模块，选择或定义命令）

三、 硬件开发调试

厂商自主选择合适的硬件模组，完成产品硬件的设计，并选择设备嵌入式系统开发方式，集成对应的JeeSDK进行固件开发。完成固件开发后，厂商可在设备开放平台申请调试激活码，并写入待调试的设备，进行功能调试。

四、 云端配置

厂商无需在设备端实现复杂的业务逻辑，物栖 OS 提供多种云端配置功能，包括：自定义配网信息、多语言管理、场景联动等；

五、 完成测试并发布

中科物栖为各种品类提供完整的测试方案，帮助开发者顺利完成软硬件测试验证，使设备顺利接入到平台。

设备接入的原理如图 1 所示。

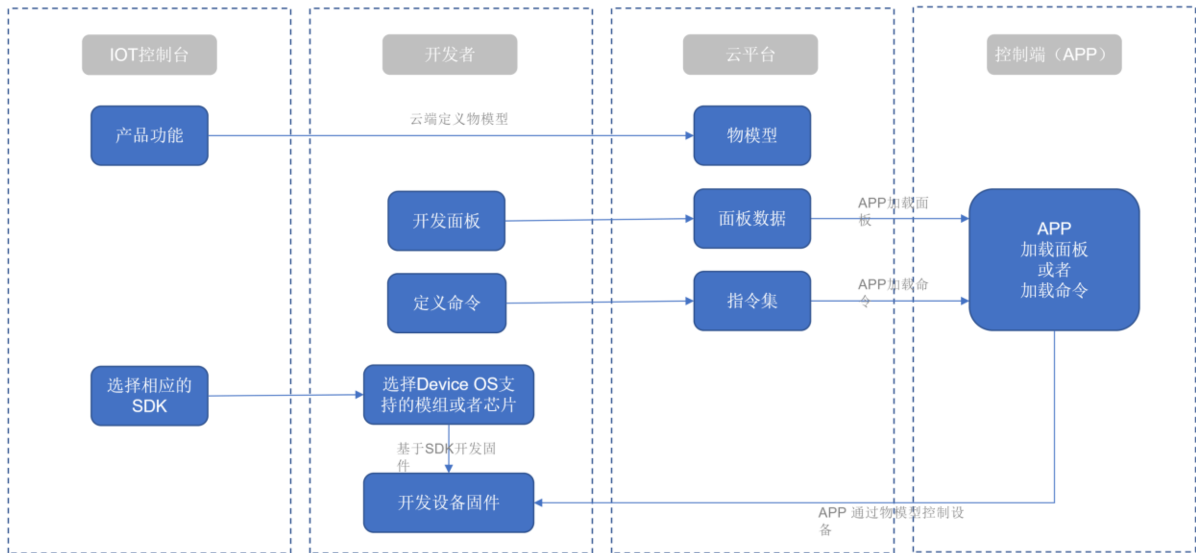


图 1 设备接入原理图

设备配网

用户首次使用设备时，需要通过冒泡物联 App 为设备配置 Wi-Fi，联网成功后，设备自动接入物栖 OS。然后，用户即可通过冒泡物联 App 对设备进行远程控制。

AP 配网

原理

设备首次开机后，需开启热点，并启动 UDP Server 等待来自网络的配网广播。

用户使用冒泡物联 App 登录，并执行添加设备操作。此时，冒泡物联 App 会要求用户输入 Wi-Fi 的 SSID 和密码，并尝试使手机连接到设

备热点。一旦手机成功连接上设备热点，就会将 SSID 和密码通过 UDP 广播的方式发送出去。

当设备端接收到来自广播的数据后将 SSID 和密码解析出来，然后尝试连接 Wi-Fi。当 Wi-Fi 连接成功后，设备将自动登录到物栖 OS，并使设备处于接收命令的状态。

原理如图 2-1 所示。

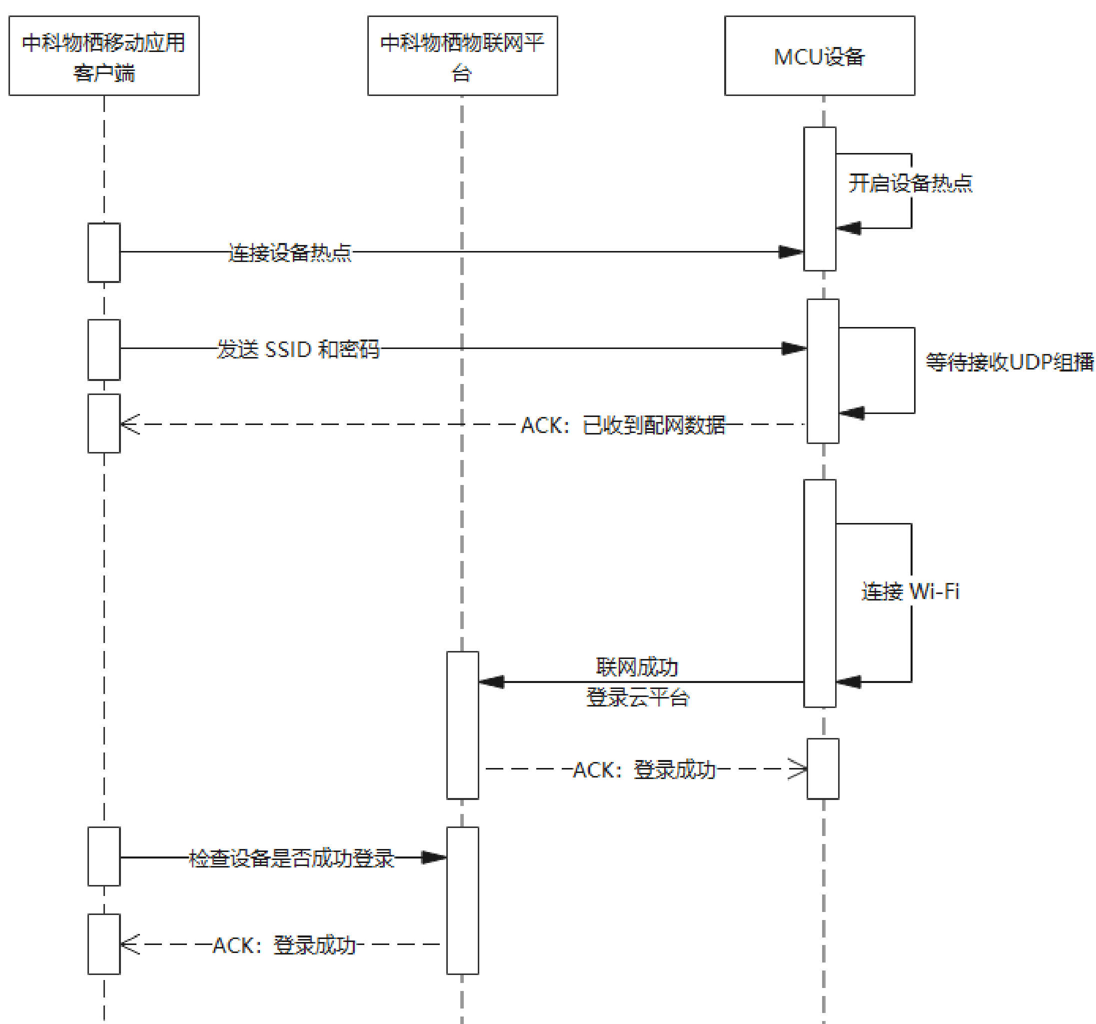


图 2-1 设备 AP 配网

实现

按照以下步骤开发，即可实现设备配网：

1. 开启设备热点（AP）；
2. 建立 UDP Socket 广播监听；
3. 接收 UDP 数据；
4. 调用 SDK 配网函数，并注册回调函数，等待接收配网参数；
5. 在回调函数中使用配网参数连接 Wi-Fi，并在 Wi-Fi 连接成功后启动核心服务；
6. 释放 UDP Socket 资源。

示例

配置 socket 头文件

由于不同的 MCU 设备 socket 头文件路径不尽相同，因此需要将当前设备的 socket 相关头文件路径配置在 sdk_jee.h 中。

开启 UDP 广播 Server

```
1. void (*callback_network)(const char *ssid, const char *psk, void *data);
2.
3. void conn_wifi_callback(const char *ssid, const char *psk, void *data)
4. {
5.         /* 5. Connect to Wi-Fi. */
6.         conn_wifi(ssid, psk);
7.         if (wifi_conn_state == success)
8.         {
```

```
9.                  // Connect to Wi-Fi successful.
10.         // Can be start the core service on SDK at here.
11.         }
12.
13.         /*  Save the  ‘data’  parameter, which needs to be passed in every time you
    start the SDK core service.  */
14.         save_data(data);
15. }
16.
17. int  open_ap(void)
18. {
19.         //  do  sth...
20.         if  (success)
21.         {
22.                 return  0;
23.         }
24.         return  errno;
25. }
26.
27. void  close_ap(void)
28. {
29.         //  do  sth...
30. }
31.
32. int  start_udp_server(const  char  *listen_addr,  u16  port)
33. {
34.         int  fd  =  -1;
35.         int  ret  =  0;
36.         struct  sockaddr_in  addr;
37.
38.         addr.sin_family  =  AF_INET;
39.         addr.sin_port  =  htons(port);
40.         addr.sin_addr.s_addr  =  inet_addr(listen_addr);
41.
```

```
42.         fd = socket(AF_INET, DGRAM, 0);
43.         if (fd < 0)
44.         {
45.                 os_printf("Error: open socket failed!\n");
46.                 goto e_socket;
47.         }
48.
49.         if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &enable, sizeof(int)) < 0)

50.         {
51.                 os_printf("setsockopt(SO_REUSEADDR) failed");
52.                 goto e_close;
53.         }
54.
55.         fcntl(fd, F_SETFL, fcntl(fd, F_GETFL, 0) | O_NONBLOCK);
56.
57.         ret = setsockopt(fd, SOL_SOCKET, SO_BROADCAST, (const void *)&opt, size
    of(int));
58.         if (ret < 0)
59.         {
60.                 os_printf("Setsockopt udp broadcast, err!\n");
61.                 goto e_close;
62.         }
63.
64.         ret = bind(fd, (struct sockaddr *)&addr, sizeof(addr));
65.         if (ret < 0)
66.         {
67.                 os_printf("Udp bind erron (%d), %s\n", ret, strerror(errno));
68.                 goto e_close;
69.         }
70.         os_printf("udp fd success\r\n");
71.
72.         return fd;
73.
```

```
74. e_close:
75.         close(fd);
76. e_socket:
77.         return  ret;
78. }
79.
80. void  stop_udp_server(int  fd)
81. {
82.         close(fd);
83. }
84.
85. int  udp_recv(int  fd,  u8  *buf,  u32  length)
86. {
87.         int  ret,  len;
88.         fd_set  rfds;
89.
90.         FD_ZERO(&rfds);
91.         FD_SET(fd,  &rfds);
92.         ret  =  select(fd  +  1,  &rfds,  NULL,  NULL,  NULL);
93.         if  (ret  <  0)
94.         {
95.                 return  ret;
96.         }
97.         if  (!FD_ISSET(fd,  &rfds))
98.         {
99.                 return  -1;
100.        }
101.
102.        len  =  recvfrom(fd,  buf,  length,  MSG_DONTWAIT,  NULL,  NULL);
103.
104.        return  len;
105.}
106.
107.int  main(void)
```

```
108. {
109.         char recBuffer[256] = {0};
110.         int len = -1;
111.         int ret = 0;
112.
113.         /* 1. Open device AP. */
114.         if ((ret = open_ap()))
115.         {
116.                 os_printf("Error: Open AP failed: %d\n", ret);
117.                 goto e_open_ap;
118.         }
119.
120.         /* 2. Listen UDP broadcast. */
121.     // LOCAL_UDP_PORT Defined by sdk_jee.h
122.         int fd = start_udp_server("0.0.0.0", LOCAL_UDP_PORT);
123.         if (fd < 0)
124.         {
125.                 os_printf("Error: open socket failed!\n");
126.                 goto e_close_ap;
127.         }
128.
129.         /* 3. Receive data from broadcast. */
130.         do {
131.                 if ((len = udp_recv(fd, recBuffer, sizeof(recBuffer))) < 0)
132.                 {
133.                         ret = errno;
134.                         if ((errno != EINTR) && (errno != EAGAIN))
135.                         {
136.                                 os_printf("Udp recvfrom errno: %d, %s\n", errno, s
    trerror(errno));
137.                                 goto e_close_server;
138.                         }
139.                 }
140.
```

```
141.                /* 4. Call setup network function in SDK. */
142.                ret = setup_network(recBuffer, len, conn_wifi_callback);
143.                if (ret)
144.                {
145.                        os_printf("Setup network failed!\n");
146.                        continue;
147.                }
148.                else
149.                {
150.                        os_printf("Setup network successful!");
151.                        break;
152.                }
153.        } while (1);
154.
155.        /* 6. Release resources. */
156.e_close_server:
157.        stop_udp_server();
158.e_close_ap:
159.        close_ap();
160.e_open_ap:
161.        return ret;
162.}
```

BLE 配网

设备进入蓝牙配网后启动 BLE 服务。

用户使用冒泡物联 App 登录，并执行添加设备操作。此时冒泡物联 App 会要求用户输入 Wi-Fi 的 SSID 和密码，并尝试使手机搜索 BLE 设备。一旦手机成功连接上对应的设备，就会将 SSID 和密码通过的 BLE 的连接发送出去。

当设备端接收到来自 BLE 的数据后将 SSID 和密码解析出来，然后尝试连接 Wi-Fi。当 Wi-Fi 连接成功后，设备将自动登录到物栖 OS，并使设备处于接收命令的状态。

原理如图 2-2 所示。

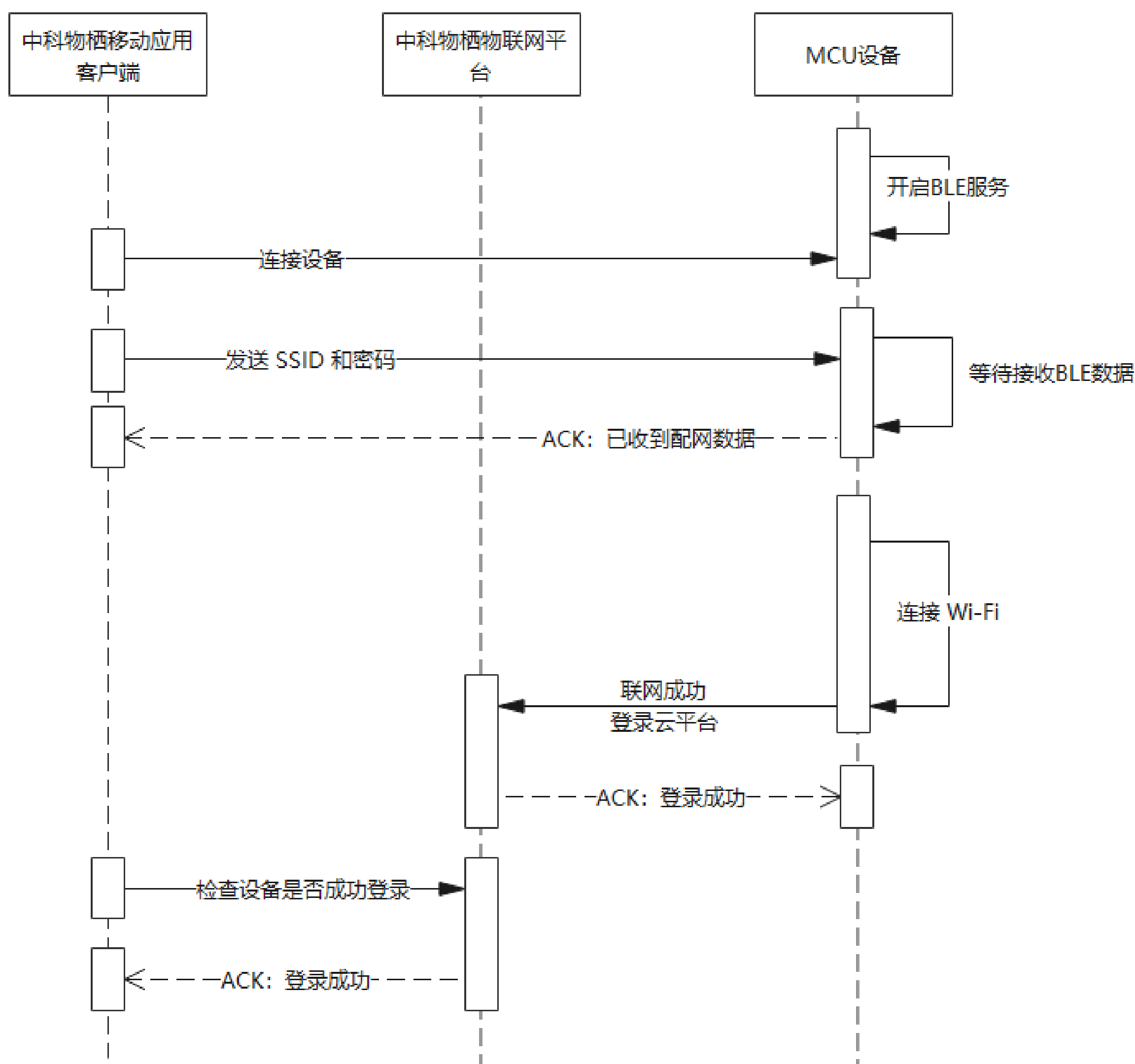


图 2-2 设备 BLE 配网

实现

按照以下步骤开发，即可实现设备配网：

1. 开启 BLE 服务；
2. 启动 BLE 广播；
3. 接收 BLE 数据，解析数据，获取 wifi 数据；
4. 回复手机 APP，断开 BLE 连接
5. 配网参数连接 Wi-Fi，并在 Wi-Fi 连接成功后启动核心服务

BLE 配网代码流程

```
1.  /*初始化  ble
2.                建立特征：
3.                0xFF01（write）
4.                0xFF02  （indicate）*/
5.  device_ble_init();
6.
7.  /*创建 ble  广播*/
8.    ble_create_advertising();
9.  /*配置广播数据
10.       配置广播名称：Jeejio-XXXXX
11.       启动广播
12. */
13. device_adv_start（）；
14.
15. /*接收 ble  数据，并解析
16.   此函数放置 ble  的接收  0xFF01 的特征的数据的位置
17.   ret  最后返回 SMT_OK，ble_ap_result 为传出参数会带出配网信息
18. */
19.   ret  =  ble_smt_data_parser(data，length,&ble_ap_result);
```

```
20.     if  (ret  ==  SMT_OK)
21.     {
22.             bk_printf("ble_smt_data_parser    ok      +++++++\r\n");
23.             ble_data_completeFlag  =  1;
24.       }
25.
26. /*回复数据，表示设备接收完毕
27. */
28.
29. smt_response_data(&sendBuffer,  &sendLength);
30. ble_send(sendBuffer, sendLength))
31. smt_release_reponse_data(sendBuffer);
```

接收命令

设备配网成功后，启动 SDK 核心服务，此时设备会自动执行登录操作，并在登录成功后进入等待接收命令的状态。

原理

设备端系统向 SDK 注册命令处理函数，等待接收用户命令。

用户使用冒泡物联 App 向设备端发送控制命令，经过物栖 OS 路由到设备后，设备端核心服务回调设备系统注册的命令处理函数，并将处理结果返回给移动应用客户端，向用户进行展示。

流程见图 3 所示。

设备接收命令流程图

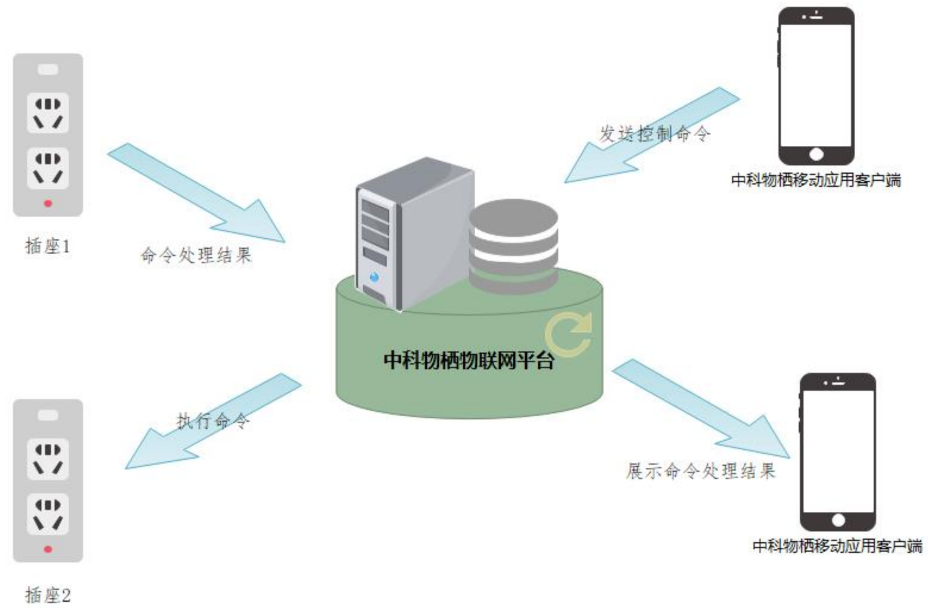


图 3 设备接收命令流程图

实现

按照以下步骤开发，即可实现命令处理：

1. 在设备开放平台注册设备命令，并配置命令的参数；
2. 设备配网成功后，启动 SDK 核心服务，并注册命令处理回调函数；
3. 在命令处理回调函数中处理接收到的命令，并返回处理结果。

示例

首先初始化 SDK 并启动核心服务，初始化的时候需要提供一些参数，这些参数有的来自配网时由冒泡物联 App 提供，有些需要开发者自己从开放平台获取。

在启动核心服务之前，需要注册相关的命令处理函数。

```
1.  void message_thread(beken_thread_arg_t arg)
2.  {
3.          struct login_info_st login_info;
4.
5.          // 在开放平台申请的设备号，共 40 字节数据
6.          login_info.secret_data = sys_config_manage(SYS_SECRET_KEY, SYS_READ, 0, 0);
7.          // 配网时由手机客户端传给设备的 user ID 参数
8.          uint64 *puser_id = sys_config_manage(SYS_USER_ID, SYS_READ, 0, 0);
9.          login_info.userId = *puser_id;
10.         // 配网时由手机客户端传给设备的 Server Key 参数
11.         login_info.serveKey = (uint8 *)sys_config_manage(SYS_SERVER_KEY, SYS_READ,
    0, 0);
12.         // 固件的版本号
13.         login_info.version = versionStringToint();
14.
15.         // SDK 初始化，每次上电只能调用一次。
16.         __jeesdk = jeesdk_init(&login_info);
17.
18.         // 注册回调函数：与服务器断开连接时回调
19.         jeesdk_register_on_disconnected(__jeesdk, on_disconnected);
20.         // 注册回调函数：收到来自服务器的命令时回调
21.         jeesdk_register_on_recv_cmd(__jeesdk, execCommand);
22.         // 注册回调函数：收到来自服务器的 OTA 升级命令时回调
```

```
23.         jeesdk_register_ota(__jeesdk,  on_ota_begin,  on_ota_download,  on_ota_finish);
24.
25.         while  (1)
26.         {
27.                 //  检查设备是否配网成功
28.                 if  (mhdr_get_station_status()  ==  RW_EVT_STA_GOT_IP  &&  reConnectFla
    g  ==  0)
29.                 {
30.                         reConnectFlag  =  1;
31.
32.                         struct  sockaddr_in  s_addr;
33.                         //  DNS  解析
34.                         message_update_hostInfo(&s_addr);
35.                         //  启动  SDK  核心函数，接收来自服务器的命令
36.                         jeesdk_main(__jeesdk,  &s_addr);
37.                 }
38.
39.                 rtos_delay_milliseconds((TickType_t)2000);  //mS
40.         }
41. }
```

断网回调函数的实现示例：

```
1.  static  void  on_disconnected(JEESDK  jeesdk)
2.  {
3.          //  检查是否可以正常访问网络
4.          if  (mhdr_get_station_status()  ==  RW_EVT_STA_GOT_IP)
5.          {
6.                  struct  sockaddr_in  s_addr;
7.                  //  DNS  解析
8.                  message_update_hostInfo(&s_addr);
9.                  //  连接云平台并等待接收命令
10.                 jeesdk_main(jeesdk,  &s_addr);
```

```
11.                 return;
12.         }
13.         reConnectFlag = 0;
14. }
```

收到命令回调函数的示例：

```
1. // 插座开关命令，此命令需要开发者在创建产品时在开放平台自行创建
2. #define TRON (0x54524F4E)
3. // TRON 命令的第一个参数
4. #define ARG_TRON_SW (0x01)
5. // TRON 命令第一个参数的值
6. #define TRON_SW_OFF (0x00) // 打开开关
7. #define TRON_SW_ON (0x01)  // 关闭开关
8.
9. static int execCommand(int event, ArgsObj arr[])
10. {
11.         switch (event)
12.         {
13.         case TRON:
14.         {
15.                 uint32 swithcValue = 0;
16.                 // 获取此命令的参数
17.                 ArgsObj sw = arr[ARG_TRON_SW];
18.                 // 将参数解析为 int 值
19.                 swithcValue = __conbineTo32(sw.data[0], sw.data[1], sw.data[2], sw.
    data[3]);
20.                 if ((TRON_SW_ON == swithcValue) && (smt_get_connect_flag_value()
    == 0))
21.                 {
22.                         //turn on the switch
23.                         printf("trun  off\r\n");
24.                         led_status_set(LED_STATE_OFF);
25.                         k1_output_status = K1_STATUS_OFF;
```

```
26.                     k1_set(k1_output_status);
27.                 }
28.                 else if ((TRON_SW_OFF == swithcValue) && (smt_get_connect_flag_va
    lue() == 0))
29.                 {
30.                     printf("trun    on\r\n");
31.                     led_status_set(LED_STATE_CONNECTED);
32.                     k1_output_status = K1_STATUS_ON;
33.                     k1_set(k1_output_status);
34.                 }
35.                 return EXECUT_OK;
36.         }
37.         break;
38.         default:
39.                 break;
40.         }
41.
42.         return NO_MEANING;
43. }
```

上报状态

概述

根据业务需要，设备端可以主动上报状态到手机客户端的 H5 页面和小应用后台，SDK 提供了 jeesdk_send_to_app()和 jeesdk_send_to_server()函数来实现此功能，详细请参见文末的 API 索引。

实现

按照以下步骤开发，即可实现上报状态：

1. 根据业务需要，在开放平台申请创建小应用，取得小应用 ID（后文称为 CallID）。
2. 开发小应用，例如：移动端小应用、设备端小应用或云应用（node.js）；
3. 小应用开发者约定通信的数据格式（通常为 json）；
4. 当设备接收到状态变化时（如温度传感器中断等），通过调用 SDK API 将数据发送给移动端小应用的 H5 页面展示或发送给云应用进行逻辑处理。

示例

1. 发送到 H5 端，与 H5 直接交互

```
2.  /**
3.    *  设备上报数据到 H5
4.    *  @jeesdk   SDK   句柄，初始化 SDK 时取得
5.    *  @appid   接收数据的小应用 ID，从开放平台取得
6.    *  @json   上报给小应用的数据，为 JSON 格式，设备开发者与前端开发者自行约定
7.    *  @length   json 参数的长度
8.    *  @return   返回零-成功、非零-失败
9.    */
10. extern  int  jeesdk_send_to_app(JEESDK  jeesdk,  uint64  appid,  uint32  event,  uint8  *json,  uint16  length);
11.
```

```
12. ret_send = jeesdk_send_to_app(__jeesdk, -3000282574505115648, 0x4241434b, "{\"status\": \"hello\"}", strlen("{\"status\": \"hello\"}"));
```

2. 发送到小应用后台

```
1.  /**
2.   * 设备上报数据到小应用后台
3.   * @jeesdk  SDK 句柄，初始化 SDK 时取得
4.   * @appid  接收数据的小应用 ID，从开放平台取得
5.   * @json  上报给小应用的数据，为 JSON 格式，设备开发者与小应用开发者自行约定
6.   * @length  json 参数的长度
7.   * @callback  回调函数，有三个参数：
8.   *              @status  零-成功、非零-失败
9.   *              @buf  小应用后台返回的数据，status 为非零时该参数为 NULL
10.  *              @len  buf 的数据长度
11.  * @return  返回零-成功、非零-失败
12.  */
13.
14. extern int jeesdk_send_to_server(JEESDK jeesdk, uint64 appid, uint32 event, uint8 *json, uint16 length, void(callback)(int status, const uint8 *buf, uint16 len));
15.
16. ret_send = jeesdk_send_to_server(__jeesdk, -3000000000001000014, 0x74696d65, "{\"status\": \"hello\"}", strlen("{\"status\": \"hello\"}"), callback);
```

OTA 升级

您可以通过设备开放平台，先将需要更新的固件文件上传至服务器，然后设备通过 OTA 协议对文件进行下载更新，最终实现固件的升级。通过集成 JeeSDK 的方式，固件侧仅需要实现少量代码即可完成升级。

平台配置

前置条件：厂商已在产品开发>硬件开发>新增自定义固件中，新增了基础版本的固件。

OTA 升级的配置步骤如下：

1. 登录设备开放平台进入产品-固件升级。
2. 选择一款已开发的产品，在固件版本管理页面选择固件；
3. 点击页面新增固件版本，创建新固件版本。

固件上传：固件升级包为.bin 格式。

固件版本：版本号格式为 xx.xx.xx，例如 1.0.6。

升级方式：

1）强制升级：冒泡物联 App 中不出现升级弹窗，OTA 平台直接推送固件版本的命令并升级。

2）提醒升级：冒泡物联 App 中出现升级推送消息，可选择升级或不升级。

3）检测升级：冒泡物联 App 中不出现升级推送消息，点击相关固件版本检测，并主动更新。

4. 固件推送并验证

1） 在固件版本管理页面，点击固件升级栏的验证。

- 在验证页面，支持以添加设备 ID 或从白名单管理中选择设备 ID 的方式推送 OTA 升级命令。
- 点击验证是否完成升级进行测试设备验证。

2） 固件升级计划发布

- 在固件版本管理页面，点击固件升级栏的发布。
- 创建固件升级发布计划，支持以产品生产批次创建发布升级计划，或者根据设备 ID 发布定向升级计划。
- 通过设备最新版本反馈查看当前发布计划中设备的升级完成情况。

OTA 流程

OTA 流程如图 4 所示。

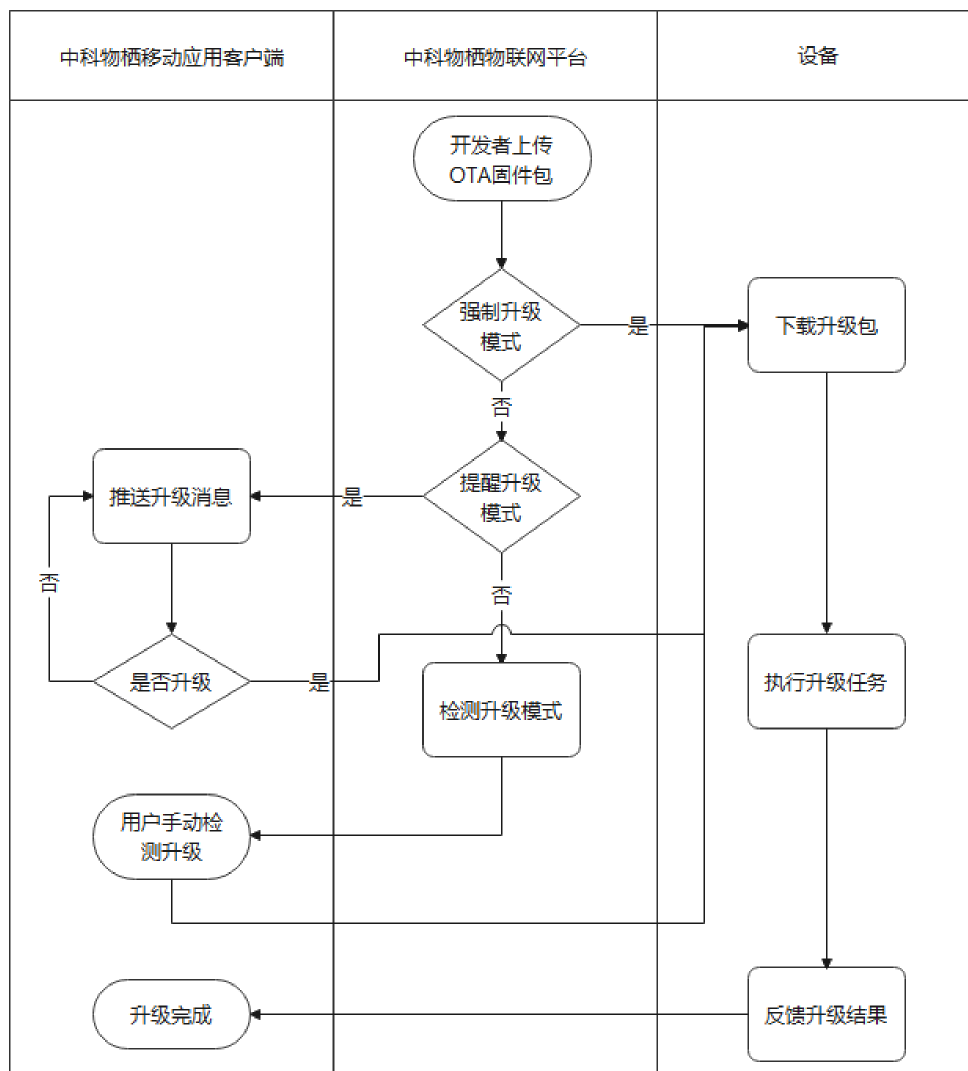


图 4 OTA 流程图

基于 SDK 的开发示例

在接收命令一节的示例代码中，我们注册了 OTA 的事件回调函数，其中 ：

on_ota_begin()函数在收到 OTA 命令时回调，应在此函数中初始化 flash，为下载数据做准备。

on_ota_download()函数在下载升级包数据时回调，此函数可能会被调用多次，按顺序依次下载固件数据。应在此函数中将下载的数据写入 flash。

on_ota_finish()函数在升级包下载完成后回调，其参数指示了下载是否成功。如果下载成功，应当在此函数中校验下载的数据，校验通过后开始进入 OTA 流程；如果下载失败，则应当擦除已下载的数据，并报错。

```
1. void on_ota_begin(void)
2. {
3.         dataWriteAddr = 0;
4.         dlPart = NULL;
5.
6.         // 使 flash 升级分区可写
7.         bk_flash_enable_security(FLASH_UNPROTECT_LAST_BLOCK); // last or custom
8.         if ((dlPart = fal_partition_find(RT_BK_DL_PART_NAME)) == NULL)
9.         {
10.                printf("Firmware download failed! Partition (%s) find error!", RT
   _BK_DL_PART_NAME);
11.        }
12. }
13.
14. void on_ota_download(uint8 *data, uint32 length)
15. {
16.        int res;
17.
18.        // 擦除 flash 升级分区
19.        res = fal_partition_erase(dlPart, dataWriteAddr, length);
20.        if (res < 0)
21.        {
```

```
22.                 printf("Firmware download failed! Partition (%s) write data error
    !", dlPart->name);
23.         }
24.         // 将下载的固件数据写入 flash 升级分区
25.         res = fal_partition_write(dlPart, dataWriteAddr, data, length);
26.         if (res < 0)
27.         {
28.                 printf("Firmware download failed! Partition (%s) write data error
    !", dlPart->name);
29.         }
30.
31.         // 累加已写入数据的总长度
32.         dataWriteAddr += length;
33. }
34.
35. void on_ota_finish(uint32 successful)
36. {
37.         // 固件下载成功
38.         if (successful)
39.         {
40.                 // 对 OTA 分区进行校验
41.                 if (rt_ota_part_fw_verify(dlPart) < 0)
42.                 {
43.                         printf("Firmware download failed! Partition (%s) header an
    d body verify failed!", RT_BK_DL_PART_NAME);
44.                         return;
45.                 }
46.                 bk_flash_enable_security(FLASH_UNPROTECT_LAST_BLOCK); // last or cus
    tom
47.                 printf("reboot \r\n");
48.                 // 重启设备，开始升级
49.                 bk_reboot();
50.         }
51.         else // 固件下载失败
```

```
52.         {
53.                 printf("Firmware download failed! erase data length %x\r\n", dataWriteAddr);
54.                 // 擦除 OTA 分区
55.                 if (fal_partition_erase(dlPart, 0, dataWriteAddr) < 0)
56.                 {
57.                         printf("Firmware download failed! Partition (%s) write data error!", dlPart->name);
58.                 }
59.                 message_upgrade_timer_stop();
60.         }
61. }
```

设备烧写序列号及密钥

设备与物栖 OS 建立连接采用一机一密的方式，因此每台设备都需要烧写序列号及密钥，SDK 内部在与物栖 OS 建立连接时需要读取此数据。

设备创建及证书生成步骤如下：

1. 登录设备开放平台进入产品-设备管理。

选择一款已开发的产品，在设备管理页面批量创建设备；

2. 点击页面批量创建，创建设备。

产品 ID：选择一款已开发的产品。

设备数量：填写当前批次要生成的设备数量。

提交完成后，系统会为厂商生成指定数量的设备号；

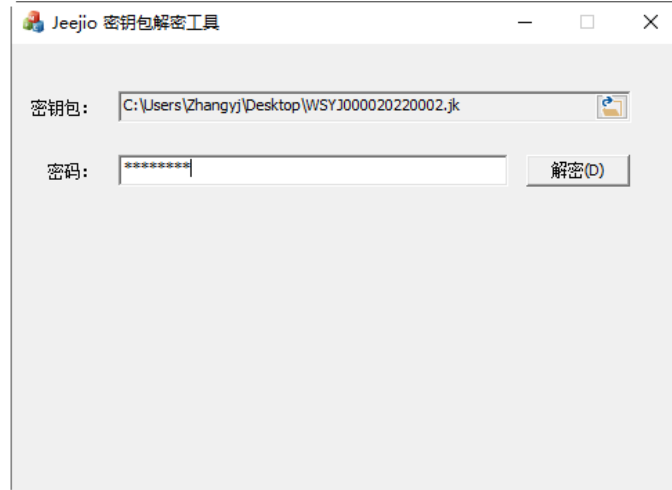
3. 下载设备证书

在批量管理页面，点击下载证书。

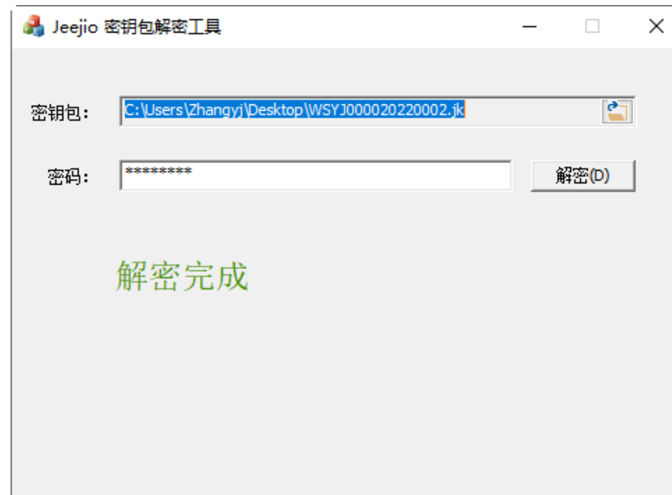
每个设备烧录其唯一的设备证书 ProductKey（产品 ID）、DeviceKey（设备 ID） 和 DeviceSecret（设备密钥）。当设备与物栖 OS 建立连接时，物栖 OS 对其携带的设备证书信息进行认证。

4. 点击下载证书，直接将此批次包含的设备证书文件下载到本地。
5. 将证书烧写到设备。
    i. 下载解密工具压缩文件：
    http://devdoc.jeejio.com/download/JKeyDecrypt.rar
    ii. 解压后获得解密工具：**JKeyDecrypt**
    iii. 运行解密工具

    iv. 导入证书文件包，并填写证书创建时设置的密码。



v. 点击【解密】。



vi. 解密完成，可获得证书压缩文件。

vii. 解压缩文件，获得证书文件。

名称	修改日期	类型	大小
116900000002	2022/4/6 14:38	文本文档	1 KB
WSYJ000000020906.bin	2022/4/6 14:38	BIN 文件	1 KB
WSYJ000000023675.bin	2022/4/6 14:38	BIN 文件	1 KB
WSYJ000000027190.bin	2022/4/6 14:38	BIN 文件	1 KB

viii. 使用对应的烧录工具，将这些证书烧录到设备上。

示例

与配网时的例子相同，调用 jeesdk_init ()函数即可启动 SDK 核心服务，并将相关密钥数据传入即可。

```
1. void conn_wifi_callback(const char *ssid, const char *psk, void *data)
2. {
3.         /* 5. Connect to Wi-Fi. */
4.         conn_wifi(ssid, psk);
5.         if (wifi_conn_state == success)
6.         {
7.                 // 配网成功，可以在这里启动 SDK 核心服务。
8.                 // login_info 参数为 struct login_info_st 结构体，已经在 SDK 中定义
9.                 jeesdk_init(&login_info);
10.        }
11.
12.        /* 将 Data 参数保存起来，每次启动 SDK 核心服务时都需要传入 */
13.        save_data(data);
14. }
```

API 索引表

```
1. /**
2.  * 配网时收到的 UDP 数据处理函数
3.  *
4.  * data：udp 数据 buffer 指针
5.  * len ：udp 数据数据长度
6.  * pSoftap_res：传出参数，需要外部创建实例，待 udp 数据处理完成后，将有效数据填充到
   pSoftap_res
```

```
7.   * return :
8.   * ERROR (-1)   表示解析失败，需要重新接收 udp 数据进行解析
9.   * OK (1)       表示解析成功，pSoftap_res 内数据有效，可正常使用
10.  *
11.  */
12. extern int smt_parse_udp_data(char *data, uint16_t len, SoftAP_Result_t *pSoftap_res);
13.
14. /**
15.  * SDK 初始化，每次上电只能调用一次。
16.  * @param login_info 初始化 SDK 所需的数据，数据来源详见 login_info_st 结构体描述。
17.  */
18. extern JEESDK jeesdk_init(struct login_info_st *login_info);
19.
20. /**
21.  * SDK 的入口，当准备工作完成后，可以开始与云平台建立连接时调用此函数。
22.  * @param jeesdk 代表 SDK 实例，通过 jeesdk_init() 函数获得。
23.  * @param paddr 云平台服务地址，该地址在配网时由手机客户端传入。
24.  */
25. extern void jeesdk_main(JEESDK jeesdk, struct sockaddr_in *paddr, uint64_t userId);
26.
27. extern void jeesdk_close(JEESDK jeesdk);
28.
29. typedef void (*jeesdk_cb_on_disconnected)(JEESDK jeesdk);
30. /**
31.  * 注册接收与云平台断开连接通知的回调函数。
32.  * 应当在此函数中重新调用 jeesdk_main() 函数重新与云平台建立连接。
33.  */
34. extern void jeesdk_register_on_disconnected(JEESDK jeesdk, jeesdk_cb_on_disconnected
    callback);
35.
36. /**
37.  * 收到控制命令的回调函数。
38.  * 发送命令的控制端一般是手机客户端的 H5 面板，面板的功能由开发者自行在开放平台编辑。
39.  *
```

```
40.  * @param event 命令类型，由开发者自行在开放平台配置。
41.  * @param arr 命令的参数，由开发者自行在开放平台配置。
42.  *
43.  * @return 返回给控制端的值。
44.  */
45. typedef int (*jeesdk_cb_on_recv_cmd)(int event, ArgsObj arr[], uint8_t **ppoutData, int
    *plength);
46.
47. /**
48.  * 注册收到云平台发送来的命令时回调函数。
49.  */
50. extern void jeesdk_register_on_recv_cmd(JEESDK jeesdk, jeesdk_cb_on_recv_cmd callback);
51.
52. /**
53.  * 收到 OTA 升级命令时的回调函数，建议在此函数中初始化 flash。
54.  */
55. typedef void (*jeesdk_cb_on_ota_begin)(void);
56.
57. /**
58.  * 收到下载数据时的回调函数。
59.  * 由于数据可能会被分段下载，此函数可能会被回调多次，
60.  * 每次返回一段下载的数据，依次将本函数返回的数据写入 flash 即可。
61.  *
62.  * @param data 已下载的数据段。
63.  * @param length data 参数的长度。
64.  */
65. typedef void (*jeesdk_cb_on_ota_download)(uint8_t *data, uint32_t length);
66.
67. /**
68.  * OTA 数据下载完成时的回调函数。
69.  *
70.  * @param successful 为 1 表示下载成功；为 0 表示下载失败。
71.  */
72. typedef void (*jeesdk_cb_on_ota_finish)(uint32_t successful);
```

```
73.
74. /**
75.  * 注册 OTA 事件监听。
76.  * @param on_begin 收到 OTA 升级命令时的回调函数。
77.  * @param on_download 收到下载数据时的回调函数。
78.  * @param on_finish OTA 数据下载完成时的回调函数。
79.  */
80. extern void jeesdk_register_ota(JEESDK jeesdk,
81.                                 jeesdk_cb_on_ota_begin on_begin,
82.                                 jeesdk_cb_on_ota_download on_download,
83.                                 jeesdk_cb_on_ota_finish on_finish);
84.
85. /**
86.  * 设备上报数据到 H5
87.  * @jeesdk SDK 句柄，初始化 SDK 时取得
88.  * @appid 接收数据的小应用 ID，从开放平台取得
89.  * @json 上报给小应用的数据，为 JSON 格式，设备开发者与前端开发者自行约定
90.  * @length json 参数的长度
91.  * @return 返回零-成功、非零-失败
92.  */
93. extern int jeesdk_send_to_app(JEESDK jeesdk, uint64_t appid, uint32_t event, uint8_t
    *json, uint16_t length);
94.
95. /**
96.  * 设备上报数据到小应用后台
97.  * @jeesdk SDK 句柄，初始化 SDK 时取得
98.  * @appid 接收数据的小应用 ID，从开放平台取得
99.  * @json 上报给小应用的数据，为 JSON 格式，设备开发者与小应用开发者自行约定
100. * @length json 参数的长度
101. * @callback 回调函数，有三个参数：
102. *       @status 零-成功、非零-失败
103. *       @buf 小应用后台返回的数据，status 为非零时该参数为 NULL
104. *       @len buf 的数据长度
105. * @return 返回零-成功、非零-失败
```

```
106. */
107. extern int jeesdk_send_to_server(JEESDK jeesdk, uint64_t appid, uint32_t event, uint8_t
     *json, uint16_t length, void(callback)(int status, const uint8_t *buf, uint16_t
     len));
108. }
```