ALIBABA CLOUD

阿里云

生活物联网平台（飞燕平台）
设备端开发指南

文档版本：20220705

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK介绍

1.1. SDK新增功能介绍

生活物联网平台发布了多个设备端SDK版本，介绍各版本新增的功能以及开启相应功能的方法。

SDK V1.6.6新增功能

版本	新增功能说明

1.6.6	• 新的蓝牙辅助配网方案，支持天猫精灵App V4.13.0以上版本与云智能App V3.5.5以上版本。
	• 支持多模块OTA。
	• 支持子设备reset，与账号不解除绑定关系。
	• 增加TG7100C芯片支持
	  ○ 支持蓝牙辅助配网、一键配网、零配、设备热点配网。
	  ○ 优化网络协议栈，减少内存消耗。
	• 其他BSP更新
	  ○ BK7231：增加ADC CLOCK支持。
	  ○ RTL8710：增加ADC支持。
	  ○ ASR5502：增加MATH LIB。
	• 功能优化
	  ○ C-SDK透传示例代码优化。
	  ○ 品类配网支持64位长密码。
	  ○ 本地律动优化。
	• 问题修复
	  ○ 修复timer_service特殊条件下出现Null point异常。
	  ○ 修复子设备上线过程中region切换的问题。
	  ○ 修复RTL8710模组示例固件异常。
	  ○ 修复本地定时数据处理问题。

1.6.6-4	• 幻彩灯带标品应用，需单独联系业务申请，联系方式，请参见获取SDK。
	• TG7100C芯片BSP更新
	  ○ 增加TG7100C支持设备热点配网。
	  ○ 内存优化：网络协议栈内存占用减少与AliOS的多heap使能。
	  ○ reboot关中断与GPIO中断优化。
	• 问题修复
	  ○ 修复ASR5502本地定时问题。
	  ○ 修复部分平台下编译报错问题。
	  ○ 修复子设备解除订阅问题。
	  ○ 修复设备影子离线Reset问题。

	• 增加基于DeviceTimer属性的本地定时功能，该属性目前可以在天猫精灵生态项目中使用。注意

1.6.6-5

示例应用固件中已经默认打开DeviceTimer，在开发自有品牌项目时，注意仍使用LocalTimer等属性并且需要在mk文件中调整宏定义，按如下配置。

- 天猫精灵生态项目：使用DeviceTimer属性，宏配置如下：

```
GLOBAL_CFLAGS += -DAIOT_DEVICE_TIMER_ENABLE     //新版设备端DeviceTimer支持的宏开关，默认为打开状态
# GLOBAL_CFLAGS += -DAOS_TIMER_SERVICE          //老版本定时服务的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_COUNTDOWN_LIST      //老版本本地倒计时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_LOCALTIMER          //老版本本地定时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_PERIOD_TIMER        //老版本周期定时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_RANDOM_TIMER        //老版本随机定时的宏，默认为关闭状态
```

- 自有品牌项目：关闭DeviceTimer属性，使用LocalTimer等属性，宏配置如下：

```
# GLOBAL_CFLAGS += -DAIOT_DEVICE_TIMER_ENABLE  //DeviceTimer支持的宏开关，自有品牌项目关闭
GLOBAL_CFLAGS += -DAOS_TIMER_SERVICE          //老版本定时服务的宏，自有品牌项目打开
GLOBAL_CFLAGS += -DENABLE_COUNTDOWN_LIST      //老版本本地倒计时的宏，自有品牌项目打开
GLOBAL_CFLAGS += -DENABLE_LOCALTIMER          //老版本本地定时的宏，自有品牌项目打开
GLOBAL_CFLAGS += -DENABLE_PERIOD_TIMER        //老版本周期定时的宏，自有品牌项目打开
GLOBAL_CFLAGS += -DENABLE_RANDOM_TIMER        //老版本随机定时的宏，自有品牌项目打开
```

详细说明请参见本地定时功能

- 网关子设备上线优化
  - 子设备单次上线数量提升到50个。注意当网关下子设备数量超过30时，建议配置FEATURE_ALCS_ENABLED为**n**，关闭本地通信功能。
  - 可根据内存情况与子设备数量加大SDK缓存队列长度，通过配置CONFIG_MSGCACHE_QUEUE_MAXLEN宏实现。如网关系统内存充足（如Linux系统），缓存队列长度可以配置到160。
  - 可根据单次发包最大数据量加大MQTT收发缓存，通过配置CONFIG_MQTT_RX_MAXLEN与CONFIG_MQTT_TX_MAXLEN宏实现。如网关系统内存充足（如Linux系统），可配置到16384或更大。
- TG7100C芯片BSP更新
  - 支持CE认证的自适应测试。
  - 优化ADC采集噪声。
  - 修复PWM channel 4不可用问题。
  - 修复部分场景GPIO中断清除问题。
- 其他芯片BSP更新
  - 清理不再维护的芯片、board代码。
  - ASR5501改名ASR5502。

<table>
<tr><td></td><td></td></tr>
<tr><td>1.6.6-6</td><td>

KV 存储功能优化。
修复SSID包含特殊字符导致配网失败问题。
TG7100C芯片BSP更新

支持WiFi低功耗，使用方法如下。设备连云成功后，调用 <code>wifi_mgmr_sta_powersaving(2)</code> 启用低功耗；设备断网时，调用 <code>wifi_mgmr_sta_powersaving(0)</code> 退出低功耗。
<pre>
#if (defined (TG7100CEVB))
extern int wifi_mgmr_sta_powersaving(int ps);
#endif
static int user_connected_event_handler(void)
{
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    LOG_TRACE("Cloud Connected");
#if (defined (TG7100CEVB))
    wifi_mgmr_sta_powersaving(2);   /* 启用低功耗 */
#endif
    user_example_ctx->cloud_connected = 1;
    ......
}
static int user_disconnected_event_handler(void)
{
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    LOG_TRACE("Cloud Disconnected");
#if (defined (TG7100CEVB))
    wifi_mgmr_sta_powersaving(0);   /* 退出低功耗 */
#endif
    set_net_state(CONNECT_CLOUD_FAILED);
    user_example_ctx->cloud_connected = 0;
    return 0;
}
</pre>

修复弱网环境OTA成功率低问题。
修复断网重连存在的问题。
Wi-Fi安全增强。
其他BSP优化。

</td></tr>
<tr><td>1.6.6-8</td><td>
本地定时优化。

定时配置字符串长度加大到1024（原长度限制为768）。
本地定时配置，改为所有数据kv保存。
APP get property直接回复kv保存的数据。
本地定时初始化时序调整。
本地定时NTP数据获取优化。

</td></tr>
</table>

1.6.6-9	• SDK部分更新
	  ○ NTP 同步优化。
	  ○ 解决LV应用线程释放问题。
	  ○ 解决开启WiFi低功耗，设备OTA过程中AP断电，设备重连AP问题。
	• TG7100C芯片BSP更新
	  ○ 设备支持WPA3安全加密配置。
	  ○ 解决个别路由器兼容性问题。

1.6.6-10	• SDK部分更新
	  ○ SDK同时支持设备热点、蓝牙辅助配网。
	  ○ 新增蓝牙辅助配网设备交互埋点。
	  ○ 蓝牙辅助配网优化。
	• TG7100C芯片（WiFi低功耗）BSP更新
	  ○ 蓝牙连接稳定性提升。
	  ○ Flash读写接口优化。
	  ○ Boot更新。
	  ○ 烧录工具更新到1.6.5（OTA生成工具，libc库要求更新到2.25或以上版本）
	• 删除部分不再维护的Board。

1.6.6-12	• SDK部分更新
	  ○ 蓝牙配网失败后，设备重新启动配网广播。
	  ○ 设备连ap超时时长从30秒增加到40秒。
	  ○ 修复pid字串长度大约7，导致蓝牙辅助配网异常问题。
	• TG7100C芯片（WiFi低功耗）BSP更新
	  ○ 烧录工具更新到1.6.7。
	  ○ 支持片内新版flash。
	  ○ 支持us级定时精度。
	  ○ 路由器兼容性优化。

说明 请通过git pull更新rel_1.6.6分支到最新版本（1.6.6-12）。

SDK V1.6.2新增功能

- 网关支持子设备三元组批量申请、异步下发。
- 网关支持子设备批量上线。
- 无AliOS Things SDK支持适配FreeRTOS。

  请参见无AliOS Things的SDK适配指南。
- 灯品类支持设备本地拾音、律动，授权开放。

SDK V1.6.0新增功能

- 支持全球统一激活中心

  编译固件时参数default_region配置为SINGAPORE 或者MAINLAND，设备可全球使用。

- 设备端重连策略优化

  此功能优化了MQTT断连之后重新连云的策略，默认开启。

- 蓝牙辅助配网时长优化

  此功能缩短了蓝牙辅助配网与绑定的总时长，默认开启。

- 蓝牙辅助配网的离线配网和控制

  此功能使得在外网不可靠的条件下仍能完成配网，并且App仍能通过蓝牙控制设备。

- 设备本地时钟同步服务

  此功能提供了一种设备间时间同步机制，如可用于多个灯组控时的场景同步。

SDK V1.5.0新增功能

- 本地组控优化

  此功能提升了本地组控时的性能，可以优化灯品类组控、组律动的同步效果，默认开启。

- 设备证书分发工具

  详情请参见设备证书分发工具使用指导。

- 离线恢复出厂设置

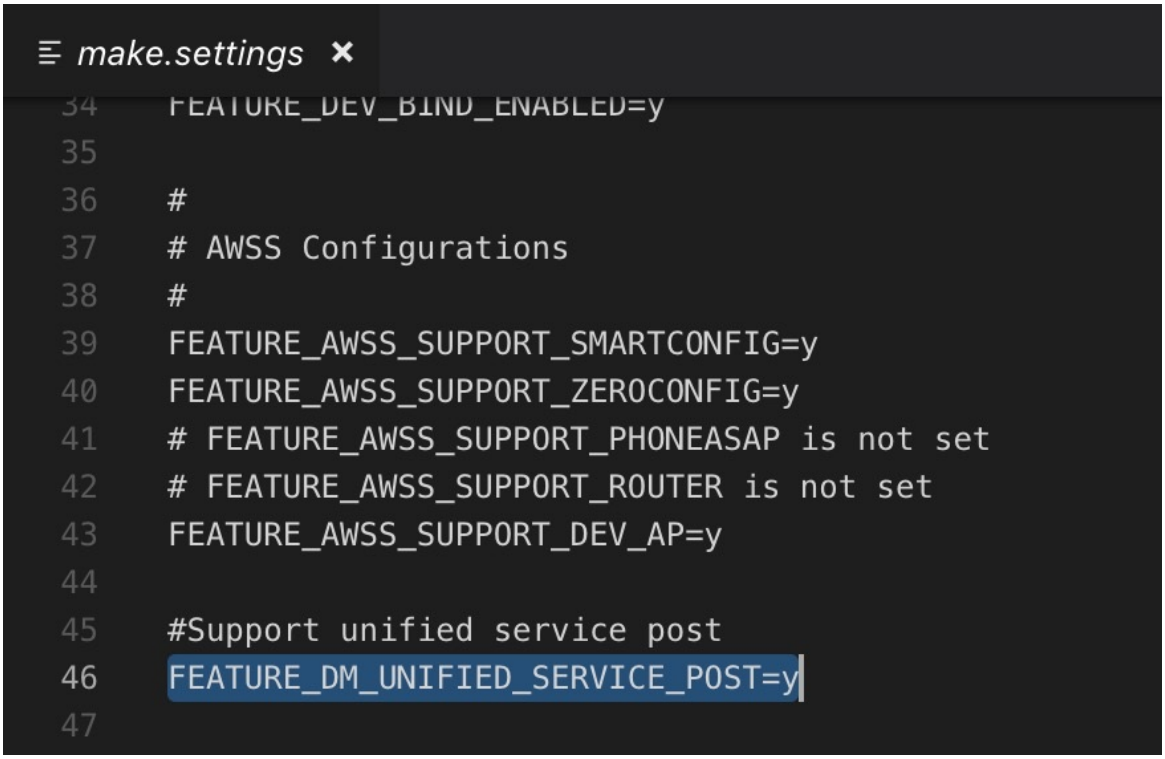
  此功能支持对离线的设备进行恢复出厂设置操作，设备重新上线后能获得离线时的恢复出厂设置操作。

SDK V1.4.0新增功能

- 网关与子设备支持统一激活中心

  V1.4.0版本对网关参考应用做了重构，支持中国内地之外地区的子设备可以切换数据中心，请参考如下配置使能该功能。

- 如您基于AliOS版本SDK开发，需要将*Products/example/linkkit_gateway/make.settings*文件中FEATURE_DM_UNIFIED_SERVICE_POST设置为y。

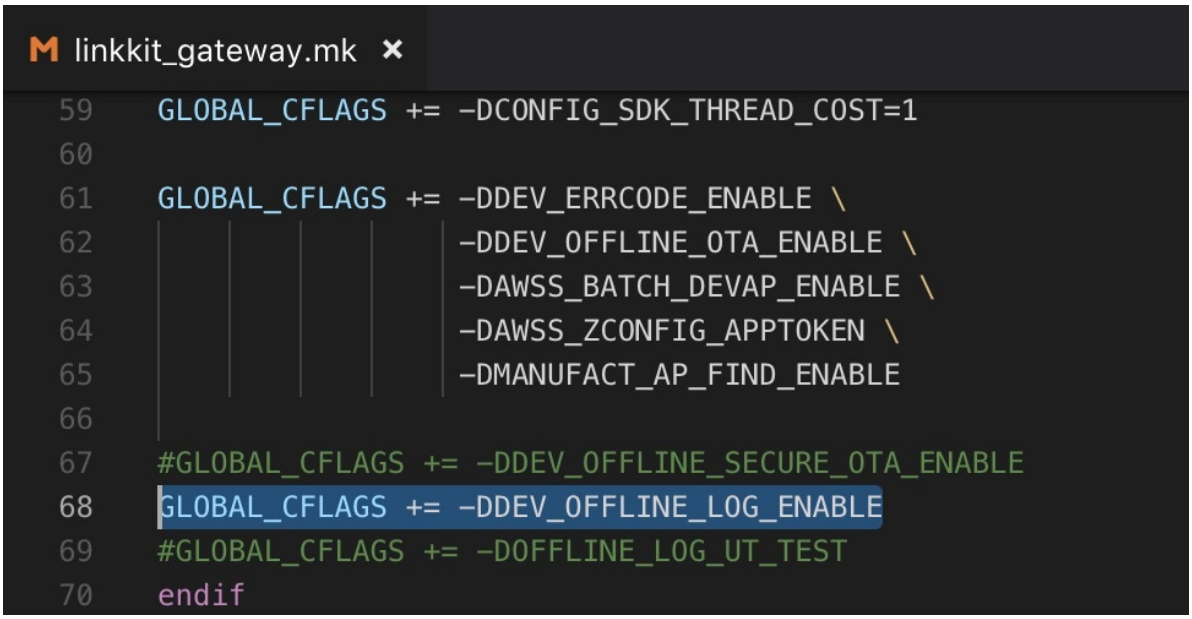


- 如您基于无AliOS版本SDK开发，可以编辑根目录下面的*make.settings*文件，增加`FEATURE_DM_UNIFIED_SERVICE_POST=y`。更多详情请参见网关开发。

- 设备离线日志功能

  对于具备设备热点配网的能力的设备，如果在设备配网绑定的过程中出现失败，App会引导用户进行诊断。进入诊断之后设备会将一些关键日志（例如错误码、上报云端的token、获取IP的时长等）发给App，通过App将这些日志上传到云端，然后可以在飞燕控制台的设备管理页面中进行查看，便于设备的远程运维。

您如需开启此功能，在应用mk文件中增加配置项 `GLOBAL_CFLAGS += -DDEV_OFFLINE_LOG_ENABLE` （如您基于linkkit_gateway开发，在*linkkit_gateway.mk*文件中已包含该配置项，如下图所示）。



> **说明** ：该功能会占用约4KB的代码空间和将近3 KB的RAM空间，其中占用的RAM空间诊断结束之后会立即释放，不诊断的情况会在上电五分钟之后释放，您可以根据需要决定是否打开该功能。

- 设备热点配网优化

  此项优化减小了设备通过设备热点配网连接中国内地之外地区站点的时长，默认开启。

- 蓝牙辅助配网优化

  此项优化减少了设备通过蓝牙辅助配网连接中国内地之外的国家和地区（包括港澳台地区）站点的时长，并在配网进行过程中增加了设备异常自检信息的获取和显示，并能够在移动端App界面看到详细的设备异常信息。蓝牙辅助配网的以上功能在应用示例comboapp中已开启，即*comboapp.mk*中已开启以下配置项。

  ```
  GLOBAL_CFLAGS += -DAWSS_REGION_ENABLE
  GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE
  ```

  > **说明** ：使用蓝牙辅助配网功能的设备，请参考应用示例comboapp开发。

- 设备连路由器失败诊断

  此功能默认开启，支持在设备Wi-Fi配网连接路由器失败时，启动自检并分析具体的路由器连接失败原因。失败详情错误码可以通过手机App页面显示。

- Wi-Fi产线检测工具模块
  - 功能说明：此功能作为独立检测工具模块默认开启，提供给应用层开发时调用，用于搜索指定路由器的详情。
    - 默认搜索时间最长为3秒。
    - 可检测路由器是否能被搜索到。
    - 可检测路由器信号强度是否过弱。
    - 可在搜索到指定路由器时返回结果给应用层。

- 使用限制
  - 不能在设备热点模式时使用。
  - 不能在设备和路由器处于连接状态时使用。
  - 不能在配网模式下使用。
- 示例代码：

  关于该工具模块的使用，您可以参考应用示例living_platform中*app_entry.c*的示例代码的实现。

```
#define TEST_LINE_AP                        "ali_product_line_test"
#define TEST_RSSI_THRESHOLD                 (-60)
static void handle_apscan_cmd(char *pwbuf, int blen, int argc, char **argv)
{
    int ret = 0;
    ap_scan_info_t scan_result;
    int ap_scan_result = -1;
    if (argc == 0) {
        // start ap scanning for default 3 seconds
        memset(&scan_result, 0, sizeof(ap_scan_info_t));
        ap_scan_result = awss_apscan_process(NULL, TEST_LINE_AP, &scan_result);
        if ( (ap_scan_result == 0) && (scan_result.found) ) {
            aos_cli_printf("AP Info: auth(%d) chan(%d) mac(%02X:%02X:%02X:%02X:%02X:%02
X) rssi(%d)\r\n", scan_result.auth,
                           scan_result.channel, scan_result.mac[0], scan_result.mac[1]
, scan_result.mac[2],
                           scan_result.mac[3], scan_result.mac[4], scan_result.mac[5],
scan_result.rssi);
            if (scan_result.rssi < TEST_RSSI_THRESHOLD) {
                aos_cli_printf("AP_SCAN AP rssi too low\r\n");
            } else {
                aos_cli_printf("AP_SCAN AP found\r\n");
            }
        } else {
            aos_cli_printf("AP_SCAN AP not found\r\n");
        }
    }
}
```

  运行living_platform应用的设备上电之后，输入命令**apscan**即可触发一次指定路由器的检测。
  - 如果检测到AP热点的RSSI过低，设备端会有日志打印提示：`AP_SCAN AP rssi too low`。
  - 如果检测到AP热点，且RSSI较高，设备端会有日志打印提示：`AP_SCAN AP found`。
  - 如果未检测到AP热点，设备端会有日志打印提示：`AP_SCAN AP not found`。

SDK V1.3.0新增功能

- 测试批量配网

  如果您是设备厂商，您可以在产线上将设备连到指定的产测路由器，开启此功能时设备在上电3s内自动连接路由器。该功能默认关闭，您如需开启此功能，请按以下步骤操作。
  - 在应用mk文件中增加配置项`GLOBAL_CFLAGS += -DMANUFACT_AP_FIND_ENABLE`。

- 在*app_entry.c*文件中更改产测路由器SSID和密码。
  - 默认SSID为 `ali_mprov_TEST_AP` ，其中 `ali_mprov_` 是模块自动加入的SSID前缀， `TEST_AP` 是开发者可自定义设定的字段
  - 默认密码为 `TEST_PASSWORD`

```
app_entry.c
795     ota_init();
796 #endif
797     netmgr_init();
798 #ifdef MANUFACT_AP_FIND_ENABLE
799     {
800         uint8_t m_manu_flag = 0;
801         int m_manu_flag_len = sizeof(m_manu_flag);
802         aos_kv_get(MANUAP_CONTROL_KEY, &m_manu_flag, &m_manu_flag_len);
803         if (m_manu_flag) {
804             netmgr_manuap_info_set("TEST_AP", "TEST_PASSWORD", manufact_ap_find_process);
805         }
806     }
807 #endif
```

> **说明** 如果从应用层设定的部分SSID字段，或路由器密码字段为空，该模块功能不会启用。如果设定了正确的SSID和密码，设备在未配网状态下开机时会搜索该路由器，如果搜索到就会连接，搜索不到则会进入正常的待配网状态。

- 终端用户批量配网

  支持App终端用户对多个同一型号的设备进行快速的批量配网，设备需支持设备热点配网或零配，每次最多可批量配网20个设备。

  您如需开启此功能，需在应用mk文件中增加配置项 `GLOBAL_CFLAGS += -DAWSS_BATCH_DEVAP_ENABLE` （如您基于living_platform开发，在*living_platform.mk*文件中已包含该配置项，如下图所示）。

```
living_platform.mk
57  #Create thread of cm_yield in mqtt_client.c
58  GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1
59
60  GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
61                   -DDEV_OFFLINE_OTA_ENABLE \
62                   -DAWSS_BATCH_DEVAP_ENABLE \
63                   -DAWSS_ZCONFIG_APPTOKEN
```

- 设备热点配网错误码诊断

  此功能支持设备通过热点配网连路由器或连云过程中产生的异常，通过App端的“错误诊断”功能展示出来。

  您如需开启此功能，需在应用mk文件中增加配置项 `GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE` （如您基于living_platform开发，在*living_platform.mk*文件中已包含该配置项，如下图所示）。

living_platform.mk

```
#Create thread of cm_yield in mqtt_client.c
GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1

GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
                 -DDEV_OFFLINE_OTA_ENABLE \
                 -DAWSS_BATCH_DEVAP_ENABLE \
                 -DAWSS_ZCONFIG_APPTOKEN
```

- 本地组控/组律动

  此功能提供了灯品类组内设备同步律动能力。

  您如需开启此功能，请按以下步骤操作。

  - 确认本地通信功能已经打开，即*make.settings*文件中FEATURE_ALCS_ENABLED设置为y，如下图所示。

    make.settings

    ```
    # Configure Link Kit SDK for IoT Embedded Devices
    #
    FEATURE_SRCPATH="."
    FEATURE_MQTT_COMM_ENABLED=y
    FEATURE_ALCS_ENABLED=y
    #
    # MQTT Configurations
    #
    ```

  - 在应用mk文件中增加配置项。如您基于living_platform开发，在*living_platform.mk*文件中增加以下配置项。

    ```
    GLOBAL_CFLAGS += -DALCS_GROUP_COMM_ENABLE
    GLOBAL_CFLAGS += -DDM_UNIFIED_SERVICE_POST
    ```

- 离线OTA

  此功能提供了设备在设备热点下配网或连云失败后通过固件升级进行固件修复的手段，支持普通和安全离线升级。

  您如需开启此功能，需在应用mk文件中增加配置项 `GLOBAL_CFLAGS += -DDEV_OFFLINE_OTA_ENABLE` （如您基于living_platform开发，在*living_platform.mk*文件中已包含该配置项，如下图所示）。

```
living_platform.mk
57  #Create thread of cm_yield in mqtt_client.c
58  GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1
59
60  GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
61                   -DDEV_OFFLINE_OTA_ENABLE \
62                   -DAWSS_BATCH_DEVAP_ENABLE \
63                   -DAWSS_ZCONFIG_APPTOKEN
```

您还可以按照以下步骤启用离线OTA安全签名校验。

i. 在应用mk文件中增加配置项：`GLOBAL_CFLAGS += -DDEV_OFFLINE_SECURE_OTA_ENABLE` 。

ii. 获取产品的公钥。在控制台的**运营中心 > 设备运维 > 固件升级 > 安全升级**页面中， 打开对应产品的安全升级开关，并单击**复制**。

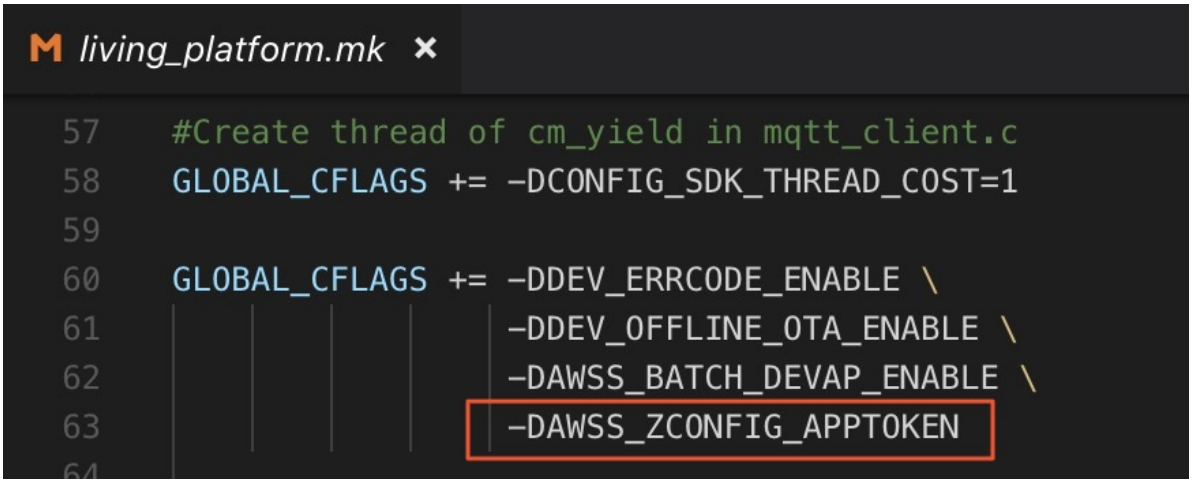
iii. 在*Living_SDK/framework/uOTA/src/verify/ota_public_key_config.h*文件中，将下图的两行数据（红框所示），替换为获取的公钥信息。

```
ota_public_key_config.h
1   #ifndef OTA_PUBLIC_KEY_CONFIG_H_
2   #define OTA_PUBLIC_KEY_CONFIG_H_
3
4   /*below two bufs are about the rsa public key, user needs sign in alibaba cloud to get them*/
5   //static const unsigned char ota_pubn_buf[256] = {0x00};
6   //static const unsigned char ota_pube_buf[3] = {0x00};
7
8
9       static const unsigned char ota_pubn_buf[256] = {0xA9, 0x82, 0x89, 0x44, 0x46, 0xC3, 0x1A, 0xA3, 0x50, 0x8C, 0x46, 0x
10      static const unsigned char ota_pube_buf[3] = {0x01, 0x00, 0x01};
11
12  #endif
```

- Wi-Fi设备零配token优化

  此功能支持在零配过程中，由App端生成绑定用的token并传输给设备，从而解决零配方案在路由器设置AP隔离模式下绑定失败的问题。

  您如需开启此功能，需在应用mk文件中增加配置项 `GLOBAL_CFLAGS += -DAWSS_ZCONFIG_APPTOKEN` （如您基于living_platform开发，在*living_platform.mk*文件中已包含该配置项，如下图所示）。



SDK V1.1.0新增功能

- 除中国内地以外地区的设备支持统一激活

  在中国内地之外的地区具有3个数据中心：新加坡、美国和德国。中国内地以外地区的设备激活联网时，将统一连接到新加坡激活中心。在设备绑定时，平台将根据App用户所在区域，自动将设备切换到相应的数据中心。详情请参见基于V1.1.0 SDK编译（linkkitapp）。

- 定时组件

  增加了对倒计时、本地定时、循环计时、以及随机定时的简单产品物模型解析支持。

- 连云优化

  提升了设备连云的速度。

- 配网与绑定优化
  - 一键配网组播编码，避免AP隔离导致的无法绑定，提升绑定成功率。
  - 增加蓝牙辅助配网功能。
  - 提供密码加密存储。
  - 绑定解绑感知，绑定解绑成功通知设备。

1.2. 获取SDK

生活物联网平台SDK是阿里云IoT针对生活物联网平台所提供的设备端SDK。基于该SDK开发设备固件，可以实现设备与云端通信，以及Wi-Fi配网、本地控制等功能。

需要您发送邮件申请SDK权限。请按以下模板发送邮件至TGmeshSDK@list.alibaba-inc.com联系我们。我们会在收到邮件后的5个工作日内联系您。

> **说明** 仅支持使用生活物联网平台官方芯片的用户申请获取本SDK权限。

- 邮件主题：生活物联网平台SDK和操作说明文档
- 邮件内容：（此SDK只针对于阿里云IoT智能生活领域WiFi&BLE的接入方式，预计审核周期5个工作日）

公司名称：
公司地址：
联系人：
联系电话：
应用场景描述：需要进描述完详细的应用场景，以便于判断平台的SDK能力是否可以满足需求
官方芯片型号（WiFi&BLE）：（用于提供对应的SDK版本）

1.3. 编译SDK

本文介绍如何搭建开发环境和基于各版本的SDK编译固件。

准备开发环境

1. 搭建SDK的开发环境。

   建议您在64位Ubuntu下搭建设备端SDK的开发环境，并使用vim编辑代码。该部分的操作请自行查阅网络相关文档完成。

   > **说明** 暂不支持在Windows系统（含Windows子系统）下编译生活物联网平台SDK。

2. 安装Ubuntu（版本16.04 X64）程序运行时库。

   请您按顺序逐条执行命令。

   ```
   sudo apt-get update
   sudo apt-get -y install libssl-dev:i386
   sudo apt-get -y install libncurses-dev:i386
   sudo apt-get -y install libreadline-dev:i386
   ```

3. 安装Ubuntu（版本16.04 X64）依赖软件包。

   请您按顺序逐条执行命令。

   ```
   sudo apt-get update
   sudo apt-get -y install git wget make flex bison gperf unzip
   sudo apt-get -y install gcc-multilib
   sudo apt-get -y install libssl-dev
   sudo apt-get -y install libncurses-dev
   sudo apt-get -y install libreadline-dev
   sudo apt-get -y install python python-pip
   ```

4. 安装Python依赖包。

   请您按顺序逐条执行命令。

   ```
   python -m pip install setuptools
   python -m pip install wheel
   python -m pip install aos-cube
   python -m pip install esptool
   python -m pip install pyserial
   python -m pip install scons
   ```

   > **说明** 安装完成后，请您使用**aos-cube --version**查看aos-cube的版本号，需确保aos-cube的版本号大于等于0.5.11。

如果在安装过程中遇到网络问题，可使用国内镜像文件。

```
### 安装/升级 pip
python -m pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/p
ypi/simple/ --upgrade pip
### 基于pip依次安装第三方包和aos-cube
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple
/   setuptools
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple
/   wheel
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple
/   aos-cube
```

编译含AliOS Things的SDK（V1.3.0及以上版本）

如果您基于含AliOS Things的SDK（V1.3.0及以上版本）来开发设备固件，请根据以下步骤来编译SDK。

1. 在开发环境中解压下载的SDK压缩包。

   如果您通过Git命令的方式下载SDK，则无需解压。

2. 将开发的业务代码存放到SDK相应的目录下。

   如果您基于V1.3.0以上版本开发设备固件，可以将开发的代码存放到*Products/品类目录/应用名/...*下（例如*Products/example/smart_outlet*为单路智能插座参考实现）。

   SDK V1.3.0及以上版本的详细目录结构如下所示。

```
ali-smartliving-device-alios-things
├── build.sh
├── LICENSE
├── Living_SDK
│   ├── 3rdparty
│   ├── app
│   ├── board
│   ├── build
│   ├── device
│   ├── doc
│   ├── example
│   ├── framework
│   ├── gen_firmware.sh -> example/smart_led_strip/gen_firmware.sh
│   ├── include
│   ├── kernel
│   ├── LICENSE
│   ├── NOTICE
│   ├── platform
│   ├── projects
│   ├── README.md
│   ├── security
│   ├── site_scons
│   ├── test
│   ├── tools
│   ├── ucube.py
│   └── utility
├── Products
│   ├── example
│   ├── Smart_lighting
│   └── Smart_outlet
├── README.md
├── Service
│   └── version
└── tools
    ├── 5981a.sh
    ├── amebaz_dev
    ├── amebaz_dev.sh
    ├── asr5501
    ├── asr5501.sh
    ├── bk7231u
    ├── bk7231udevkitc.sh
    ├── cmd
    ├── mk3060.sh
    ├── mk3080.sh
    ├── moc108
    ├── mx1270.sh
    ├── rda5981
    ├── rtl8710bn
    └── scripts
```

3. 编译SDK。

   i. 在SDK根目录，执行 `vim build.sh` 命令打开*build.sh*文件。

ii. 根据硬件使用的模组型号和要编译的应用，修改文件中的如下参数。

```
default_type="example"       //配置产品类型
default_app="smart_outlet"   //配置编译的应用名称
default_board="uno-91h"      //配置编译的模组型号
default_region=SINGAPORE     //配置设备的连云区域：
//V1.5.0及以下版本：设备在中国内地激活配置为MAINLAND，其余地区激活配置为SINGAPORE
//V1.6.0及以上版本：配置为MAINLAND或SINGAPORE都可以，设备可以全球范围内激活
default_env=ONLINE       //配置连云环境
Debug log：default_debug=0
default_args=""          //配置其他编译参数
//更多介绍请参见README.md
```

iii. 执行./build.sh命令。

以基于uno-91h模组编译smart_outlet应用为例，编译完成后，在*out\smart_outlet@uno-91h*目录下会生成*smart_outlet@uno-91h_all.bin*文件。该文件为需要烧录到真实设备中的固件。

> **说明** ：*build.sh*脚本会自动判断指定模组的toolchain（交叉编译工具链）是否已经安装，如果没有安装，脚本会自动安装。
>
> 编译该版本SDK时，如果出现头文件、静态库缺失等错误，请参见常见问题。

编译含AliOS Things的SDK（V1.1.0及以下版本）

如果您基于含AliOS Things的SDK V1.1.0或者V1.0.0版本（原天猫精灵Wi-Fi SDK）来开发设备固件，请根据以下步骤来编译SDK。

1. 在开发环境中解压下载的SDK压缩包。

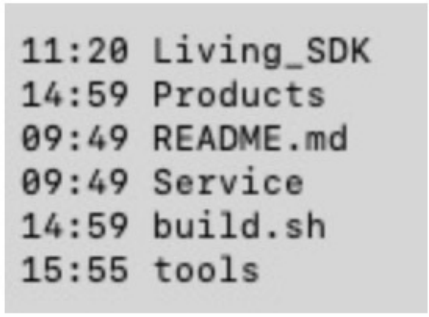
   如果您通过Git命令的方式下载SDK，则无需解压。

2. 将开发的业务代码存放到SDK相应的目录下。

   可以将开发的代码存放到*example/应用名*下，如您基于linkkit app开发，即把新增加的代码合入*example/linkkit app*目录。

   SDK V1.1.0之前版本与SDK V1.3.0及以上版本的目录对比如下。

rel_1.1.0 及之前的版本目录结构



rel_1.3.0 及之后的目录结构

3. 编译SDK。
   i. 以mk3060模组为例，如果是第一次编译，在SDK根目录，执行编译如下命令，会自动安装指定模组的toolchain。

      ```
      aos make helloworld@mk3060
      ```

   ii. 如已经安装toolchain，在SDK根目录，执行编译如下命令。

      ```
      aos make linkkitapp@mk3060
      ```

      编译完成后，在*out\linkkit app@mk3060\binary*目录下会生成*linkkit app@mk3060_crc.bin*文件。该文件为需要烧录到真实设备中的固件。

      如果编译失败且提示头文件或静态库缺失的错误，请参见常见问题。

编译无AliOS Things的SDK

如果您基于无AliOS Things的SDK来开发设备固件，请根据以下步骤来编译SDK（无AliOS Things SDK各版本的编译操作一致）。

1. 在开发环境中解压下载的SDK压缩包。

   如果您通过Git命令的方式下载SDK，则无需解压。

2. 将开发的业务代码存放到SDK相应的目录下。

   不含AliOS Things的SDK目录结构如下所示。

```
ali-smartliving-device-sdk-c
├── build-rules
│   ├── docs
│   ├── funcs.mk
│   ├── hooks
│   ├── misc
│   ├── pre-build.sh
│   ├── _rules-complib.mk
│   ├── _rules-coverage.mk
│   ├── _rules-dist.mk
│   ├── _rules-flat.mk
│   ├── _rules-kmod.mk
│   ├── _rules-libs.mk
│   ├── rules.mk
│   ├── _rules-modinfo.mk
│   ├── _rules-origin.mk
│   ├── _rules-prefix.mk
│   ├── _rules-prog.mk
│   ├── _rules-repo.mk
│   ├── _rules-submods.mk
│   ├── _rules-top.mk
│   ├── scripts
│   └── settings.mk
├── CMakeLists.txt
├── components
│   ├── tick_notify
│   └── timer_service
├── Config.in
├── Config.linkkit
├── docs
│   ├── 智能生活C-SDK适配FreeRTOS帮助文档.pdf
│   ├── 智能生活SDK复位功能介绍.pdf
│   └── 智能生活设备配网绑定问题分析参考.pdf
├── examples
│   ├── cJSON.c
│   ├── cJSON.h
│   ├── iot.mk
│   └── linkkit
├── include
│   ├── exports
│   ├── imports
│   ├── iot_export.h
│   └── iot_import.h
├── lib
│   ├── ANDES_N10
│   ├── ARM968E-S
│   ├── Cortex-M4
│   ├── linux
```

```
│   └── xtensa
├── LICENSE
├── linkkit_kv.bin
├── makefile
├── make.settings
├── output
│   └── release
├── prebuilt
│   ├── extra
│   ├── mdm9206
│   ├── ubuntu
│   └── win7
├── project.mk
├── README.md
├── sdk-c.mk
└── src
    ├── board
    ├── infra
    ├── protocol
    ├── ref-impl
    ├── sdk-impl
    ├── security
    ├── services
    └── tools
```

3. 编译SDK。
   i. 在SDK根目录，执行 `vim make.settings` 命令打开*make.settings*文件。
   ii. 根据硬件使用的模组型号和要编译的应用，修改文件中的如下参数。

   其中详细的参数解释请参见基于非认证的模组开发。

```
#=========Basic Features==========
FEATURE_SRCPATH="."
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_MQTT_AUTO_SUBSCRIBE=y
FEATURE_DEVICE_MODEL_ENABLED=y
FEATURE_OTA_ENABLED=y
FEATURE_DEV_BIND_ENABLED=y
FEATURE_SUPPORT_TLS=y
#Support unified service post
FEATURE_DM_UNIFIED_SERVICE_POST=y
#=========Basic Features==========
#=========User Config Features==========
FEATURE_ALCS_ENABLED=y
# FEATURE_DEVICE_MODEL_GATEWAY=n
#
# AWSS Configurations
#
FEATURE_WIFI_PROVISION_ENABLED=y
FEATURE_AWSS_SUPPORT_SMARTCONFIG=y
FEATURE_AWSS_SUPPORT_ZEROCONFIG=y
FEATURE_AWSS_SUPPORT_DEV_AP=y
```

   iii. 执行**make distclean**命令。

iv. 执行**make reconfig**命令，并在出现以下提示后，输入选项对应的序号。

请根据要编译的目标平台（即board）来选择。如果您编译demo，则选择7。

```
SELECT A CONFIGURATION:
1) config.esp8266.aos       6) config.rhino.make
2) config.freertos.esp8266  7) config.ubuntu.x86
3) config.macos.x86         8) config.win7.mingw32
4) config.mk3060.aos        9) config.xboard.make
5) config.mk3080.aos
```

v. 执行**make**命令。

以编译config.ubuntu.x86为例，编译完成后，在*output/release/bin*目录下会生成*living_platform*文件为demo。如果您编译真实固件请参见无AliOS Things的SDK适配指南。

1.4. 常见问题

在获取和编译生活物联网平台SDK时遇到的常见问题以及解决方法。

如何设置SSH?

SSH key用于在您的电脑和Code服务器之间建立安全的加密连接。当您使用Git命令获取SDK时，需要配置SSH。

1. 进入Linux系统，执行以下命令，判断本地是否存在SSH key。

```
cat ~/.ssh/id_rsa.pub
```

查看返回结果，是否返回以 `ssh-rsa` 或 `ssh-dsa` 开头的长字符串。

- 是：表示本地存在SSH key，请跳至“步骤3”执行。
- 否：表示本地没有SSH key，请按顺序步骤执行。

2. 执行以下命令生成SSH Key。

```
ssh-keygen -t rsa -C "xxxx@xxx.com"  //xxxx@xxx.com为您的邮箱地址
```

当页面出现提示时，您可以按**Enter**键使用默认值。

命令执行完成后，您可以使用步骤1中的命令**cat ~/.ssh/id_rsa.pub**检查SSH key。

3. 进入阿里云管理控制台。
4. 在左侧导航栏中，选择**设置 > SSH公钥**。

如果您在左侧导航栏中找不到以上菜单项，请在首页单击**创建项目**，创建一个项目后再重试。

5. 单击**增加SSH公钥**，并配置相关参数。

在**增加SSH公钥**页面中，公钥配置为**cat ~/.ssh/id_rsa.pub**命令的返回结果。

编译V1.3.0及以上版本SDK时出现的错误

头文件、静态库缺失错误

以编译bk7231u芯片为例，您可以参考以下操作来解决该问题。

1. 进入tools/bk7231udevkitc.sh脚本文件。
2. 在代码 `files_cp` 中加入复制命令，将相应的头文件或静态库，复制到对应的目录下。
   - 头文件（*.h）：复制到*prebuild/include*目录下。
   - 静态库（*.a）：复制到*prebuild/lib*目录下。

   复制命令的示例如下。

```
//静态库的复制命令
cp Living_SDK/platform/mcu/bk7231u/beken/beken.a  prebuild/lib/
//头文件的复制命令
cp Living_SDK/platform/mcu/bk7231u/beken/driver/ble/ble.h  prebuild/include/
```

3. （可选）如果在应用增加静态库文件，您还需要在该应用的*/Products/example/smart_outlet/makefile*文件中，增加静态库文件（*.a），并把新增的静态库文件编译到应用库文件中。

   请参考以下示例来新增静态库。

```
LIBFILE = lib/ARM968E-S/libtest.a
$(target):$(obj_app)
        mkdir -p obj
        $(foreach n, $(LIBFILE), $(AR) x $(n);)
        @cp *.o obj/
        @cp $(obj_app) obj/
        @$(RM) -rf *.o
        @$(RM) -rf $(obj_app)
        @$(AR) -rcs $(PWD)/../../../prebuild/lib/$(target).a obj/*.o 2>/dev/null
        @$(RM) -rf obj
```

2.Wi-Fi设备开发

2.1. Wi-Fi芯片移植

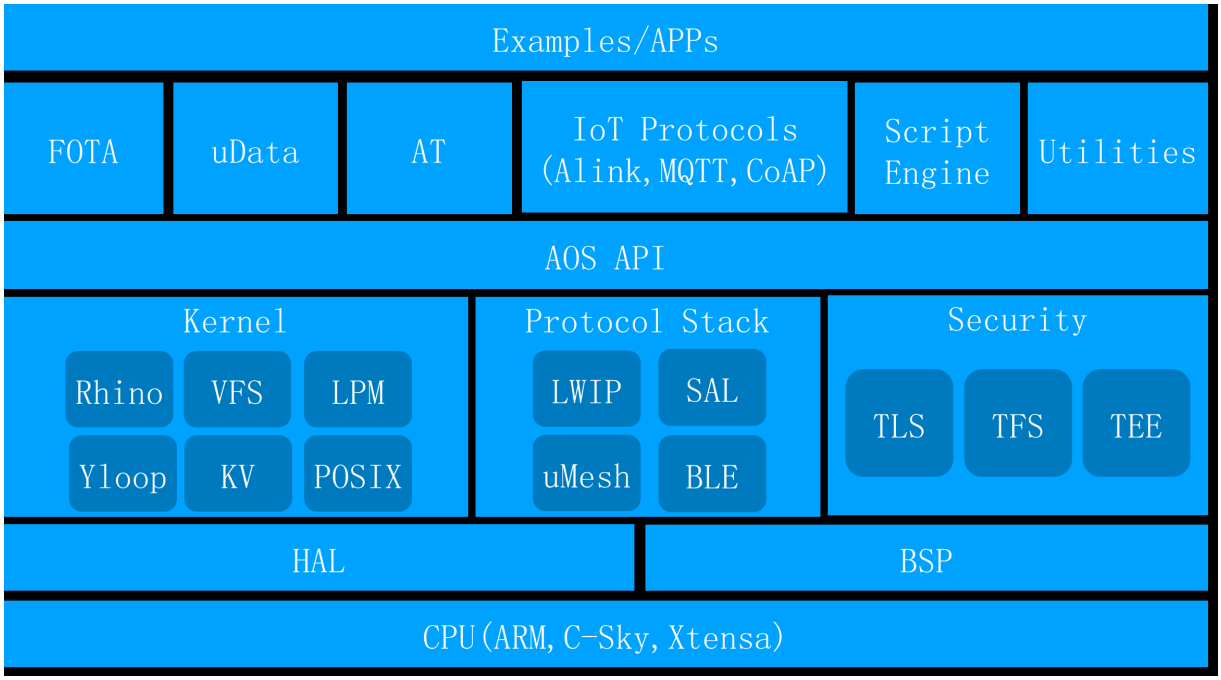
含AliOS Things的生活物联网平台SDK即包含AliOS Things物联网操作系统（基于AliOS Things V1.3.4）和Link Kit V2.3.0。本文档基于含AliOS Things版本SDK，介绍Wi-Fi芯片的移植过程。

前提条件

已下载含AliOS Things的生活物联网平台SDK，请参见获取SDK。

概述

AliOS Things的架构可适用于分层架构和组件化架构，如下图所示。



结构图从底部到顶部的内容如下。

- 板级支持包（BSP）：主要由SoC供应商开发和维护。
- 硬件抽象层（HAL）：例如Wi-Fi和UART。
- 内核：包括Rhino实时操作系统内核、Yloop、VFS、KV 存储。
- 协议栈：包括TCP/IP协议栈（LwIP）、uMesh网络协议栈。
- 安全：安全传输层协议（TLS）、可信服务框架（TFS）、可信运行环境（TEE）。
- AOS API：提供可供应用软件和中间件使用的API。
- 中间件：生活物联网平台提供了常用的增值服务中间件。
- 示例应用：生活物联网平台提供了自主开发的示例代码，以及完备测试通过了的应用程序（例如Link Kit App）。

在Wi-Fi芯片上移植含AliOS Things的SDK主要包括以下工作：

- 内核移植
- HAL移植

- Wi-Fi HAL和配网移植
- LwIP协议栈移植
- OTA移植

内核移植

AliOS Things中使用的内核为Rhino，详细介绍请参见Rhino内核移植。

本文以移植Rhino最小系统到STM32平台为例。

- 目标开发板：STM32L496G-Discovery，Cotex-m4架构。
- 移植目标：基本任务运行，tick时钟实现krhino_task_sleep，基本串口打印。

1. 配置环境和系统初始化。请参见Rhino系统移植。
2. Rhino基于Keil最小内核移植。
3. 验收测试。

   系统移植完成后，验收方法如下。

   - 系统能启动一个任务并调用 `krhino_task_sleep` 函数实现定时打印，例如每秒打印一次日志。
   - 运行Helloworld程序。

HAL移植

硬件抽象层HAL（Hardware Abstraction Layer）普遍存在于各个操作系统之中，用于屏蔽不同芯片平台的差异，使上层的应用软件不会随芯片改变而改变。目前AliOS Things定义了全面的HAL抽象层，您只需对接相应的HAL接口，就能控制芯片的控制器，从而达到控制硬件外设的目的。

AliOS Things中定义的硬件HAL包括以下内容。AliOS Things硬件HAL移植请参见硬件抽象层移植。

- ADC
- GPIO
- I2C总线
- Flash（NAND、NOR）
- PWM
- RNG
- RTC
- SD
- SPI
- Timer
- UART
- Watchdog
- DAC

AliOS Things中设计了一套简单使用的永久存储机制KV（Key-Value），该机制依赖Flash HAL。因此平台移植时，需要对Flash HAL进行移植，以实现KV功能。详细请参见Flash和KV移植。

以RDA5981平台为例，硬件抽象相关的移植代码请参见RDA5981 HAL参考实现。

Wi-Fi和配网移植

1. 移植Wi-Fi HAL。

对于Wi-Fi芯片，AliOS Things的Wi-Fi配网功能需要Wi-Fi HAL的支持。因此在平台对接过程中，需要对这些Wi-Fi HAL接口进行移植。如果是同时支持Wi-Fi和BLE的Combo芯片，您还需额外对接蓝牙能力，请参见蓝牙对接。

2. 移植与Wi-Fi HAL相关联的结构体。

   在AliOS Things中，与Wi-Fi HAL相关联的结构体为 `hal_wifi_module_t` 。Wi-Fi相关的操作和接口都封装在 `hal_wifi_module_t` 结构体中，Wi-Fi HAL相关的数据结构和接口定义请参见wifi.h。

   您可以根据以下代码示例调用对应的接口函数来实现Wi-Fi模块结构体。详细的接口函数介绍如下表所示。

```
struct hal_wifi_module_s {
    hal_module_base_t    base;
    const hal_wifi_event_cb_t *ev_cb;
    int  (*init)(hal_wifi_module_t *m);
    void (*get_mac_addr)(hal_wifi_module_t *m, uint8_t *mac);
    void (*set_mac_addr)(hal_wifi_module_t *m, const uint8_t *mac);
    int  (*start)(hal_wifi_module_t *m, hal_wifi_init_type_t *init_para);
    int  (*start_adv)(hal_wifi_module_t *m, hal_wifi_init_type_adv_t *init_para_adv);
    int  (*get_ip_stat)(hal_wifi_module_t *m, hal_wifi_ip_stat_t *out_net_para, hal_wif
i_type_t wifi_type);
    int  (*get_link_stat)(hal_wifi_module_t *m, hal_wifi_link_stat_t *out_stat);
    void (*start_scan)(hal_wifi_module_t *m);
    void (*start_scan_adv)(hal_wifi_module_t *m);
    int  (*power_off)(hal_wifi_module_t *m);
    int  (*power_on)(hal_wifi_module_t *m);
    int  (*suspend)(hal_wifi_module_t *m);
    int  (*suspend_station)(hal_wifi_module_t *m);
    int  (*suspend_soft_ap)(hal_wifi_module_t *m);
    int  (*set_channel)(hal_wifi_module_t *m, int ch);
    void (*start_monitor)(hal_wifi_module_t *m);
    void (*stop_monitor)(hal_wifi_module_t *m);
    void (*register_monitor_cb)(hal_wifi_module_t *m, monitor_data_cb_t fn);
    void (*register_wlan_mgnt_monitor_cb)(hal_wifi_module_t *m, monitor_data_cb_t fn);
    int  (*wlan_send_80211_raw_frame)(hal_wifi_module_t *m, uint8_t *buf, int len);
};
```

Wi-Fi接口函数说明

接口名称	接口说明
init	通过该接口可以对Wi-Fi进行初始化，使Wi-Fi达到可以准备进行连接工作的状态，如分配Wi-Fi资源、初始化硬件模块等操作。
get_mac_addr	通过该接口可以获取到Wi-Fi的物理硬件地址。 **说明** 回传的MAC地址格式为6个字节（不含英文逗号）二进制值，例如 `uint8_t mac[6] = {0xd8,0x96,0xe0,0x03,0x04,0x01};` 。

接口名称	接口说明
set_mac_addr	通过该接口可以设置Wi-Fi的物理硬件地址。
start	通过该接口可以启动Wi-Fi，根据启动参数wifi_mode来区分启动模式（0表示AP模式，1表示station模式）。 ○ station模式：去连接AP。 在station模式下，该函数触发AP连接操作后即返回。后续底层处理过程中，拿到IP信息后，需要调用 `ip_got` 回调函数来通知上层获取IP事件。 ○ AP模式：自身为AP，只需根据参数启动AP功能即可。 **注意** ○ station模式下启动Wi-Fi连接时，传入的SSID长度不超过32位。 ○ 在连接AP后，Wi-Fi底层需要维护自动重连操作。
start_adv	该接口类似 `start` ，但启动的参数更丰富，为可选接口。
get_ip_stat	通过该接口可以获取Wi-Fi工作状态下的IP信息，如IP、网关、子网掩码、MAC地址等信息。
get_link_stat	通过该接口可以获取Wi-Fi工作状态下的链路层信息，如连接信号强度、信道、SSID等信息。
start_scan	通过该接口启动station模式下的信道扫描。扫描结束后，调用 `scan_compeleted` 回调函数，将各个信道上扫描到的AP信息通知给上层。需要得到的扫描信息在 `hal_wifi_scan_result_t` 中定义。 **说明** 扫描结果存储所需要的内存在底层实现中分配，回调函数返回后再将该内存释放。
start_scan_adv	该接口与 `hal_wifi_start_scan` 类似，但扫描的信息更多，例如bssid、channel信息等。在 `hal_wifi_scan_result_adv_t` 中定义通过扫描想得到的信息。扫描结束后，通过调用 `scan_adv_compeleted` 回调函数通知上层。 **说明** 扫描结果存储所需要的内存在底层实现中分配，回调函数返回后再将该内存释放。
power_off	通过该接口对Wi-Fi硬件进行断电操作。

接口名称	接口说明
power_on	通过该接口对Wi-Fi硬件进行上电操作。
suspend	通过该接口断开Wi-Fi的所有连接，同时支持station模式和soft AP模式。
suspend_station	通过该接口断开Wi-Fi的所有连接，仅支持station模式。
suspend_soft_ap	该接口断开Wi-Fi的所有连接，仅支持soft AP模式。
set_channel	通过该接口可以设置信道。
wifi_monitor	通过该接口启动监听模式，并且在收到任何数据帧（包括beacon、probe request等）时，调用 `monitor_cb` 回调函数进行处理。 **说明** 回调函数是上层通过 `register_monitor_cb` 注册的。监听模式下，上层 `cb` 函数期望处理的数据包不带帧校验序列FCS（Frame Check Sequence），所以如果底层的数据包带FCS，应当先剥离再往上层传递。
stop_wifi_monitor	通过该接口关闭侦听模式。
register_monitor_cb	通过该接口注册侦听模式回调函数，回调函数在底层接收到任何数据帧时被调用。
register_wlan_mgnt_monitor_cb	通过该接口注册管理帧回调函数（非监听模式下），该回调函数在底层接收到管理帧时被调用。
start_debug_mode	通过该接口进入调试模式，可选（需模块支持）。
stop_debug_mode	通过该接口退出调试模式，可选。
wlan_send_80211_raw_frame	该接口可以用于发送802.11格式的数据帧。 您可以参考RDA5981平台实现：*platform/mcu/rda5981x/hal/wifi_port.c*。

3. 移植Wi-Fi事件回调。

在配网过程中，netmgr模块（请参见*network/netmgr/*目录）负责定义和注册Wi-Fi事件回调函数。而在Wi-Fi启动和运行的过程中，通过调用回调函数来通知上层应用，以执行相应的动作。这些Wi-Fi事件的回调函数，应该在Wi-Fi网络驱动（通常是HAL层实现或更底层的代码）的任务上下文中被触发，示例如下。

- 在Wi-Fi底层拿到IP后，应执行 `ip_got` 回调，并将IP信息传递给上。
- 在Wi-Fi完成信道扫描后，应调用 `scan_compeleted` 或 `scan_adv_compeleted` 回调，将扫描结果传递给上层。
- 在Wi-Fi状态改变时，应调用 `stat_chg` 回调。

> **说明** 这些事件回调函数由netmgr配网模块定义并注册，在Wi-Fi底层（如HAL）里面触发调用。

下面是AliOS Things中定义的Wi-Fi事件回调函数和接口，相关定义请参见wifi.h。

回调函数的定义，请参考netmgr模块中的 `g_wifi_hal_event` 函数。

```
typedef struct {
    void (*connect_fail)(hal_wifi_module_t *m, int err, void *arg);
    void (*ip_got)(hal_wifi_module_t *m, hal_wifi_ip_stat_t *pnet, void *arg);
    void (*stat_chg)(hal_wifi_module_t *m, hal_wifi_event_t stat, void *arg);
    void (*scan_compeleted)(hal_wifi_module_t *m, hal_wifi_scan_result_t *result,
                            void *arg);
    void (*scan_adv_compeleted)(hal_wifi_module_t *m,
                                hal_wifi_scan_result_adv_t *result, void *arg);
    void (*para_chg)(hal_wifi_module_t *m, hal_wifi_ap_info_adv_t *ap_info,
                     char *key, int key_len, void *arg);
    void (*fatal_err)(hal_wifi_module_t *m, void *arg);
} hal_wifi_event_cb_t;
```

4. 注册和初始化Wi-Fi模块。

   在完成Wi-Fi接口和事件回调的实现后，定义一个 `hal_wifi_module_t` 的结构体，将各个接口和回调的实现地址赋值给结构体中对应的域，示例如下。

```
hal_wifi_module_t sim_aos_wifi_vendor = {
    .base.name           = "alios_wifi_vender_name",
    .init                =  wifi_init,
    .get_mac_addr        =  wifi_get_mac_addr,
    .start               =  wifi_start,
    .start_adv           =  wifi_start_adv,
    .get_ip_stat         =  get_ip_stat,
    .get_link_stat       =  get_link_stat,
    .start_scan          =  start_scan,
    .start_scan_adv      =  start_scan_adv,
    .power_off           =  power_off,
    .power_on            =  power_on,
    .suspend             =  suspend,
    .suspend_station     =  suspend_station,
    .suspend_soft_ap     =  suspend_soft_ap,
    .set_channel         =  set_channel,
    .start_monitor       =  start_monitor,
    .stop_monitor        =  stop_monitor,
    .register_monitor_cb =  register_monitor_cb,
    .register_wlan_mgnt_monitor_cb = register_wlan_mgnt_monitor_cb,
    .wlan_send_80211_raw_frame = wlan_send_80211_raw_frame,
};
```

一般在板级初始化的过程中，先通过 `hal_wifi_register_module` 接口对Wi-Fi模块进行注册，再调用 `hal_wifi_init` 接口对Wi-Fi硬件模块进行初始化。Wi-Fi HAL模块完成初始化后才可以使用。参考实现：*platform/mcu/rda5981x/bsp/entry.c*。

```
void hal_wifi_register_module(hal_wifi_module_t *m);
int hal_wifi_init(void);
```

5. 参考实现。

   以RDA5981平台为例：

   - Wi-Fi HAL接口实现代码位于*platform/mcu/rda5981x/hal/wifi_port.c*。
   - Wi-Fi芯片注册和初始化代码位于*platform/mcu/rda5981x/bsp/entry.c*。

6. 验收测试。

   在完成W-iFi的移植后，可以通过Wi-Fi APP来验证移植工作的有效性。Wi-Fi App位于*app/example/wifihalapp/*目录下，验证流程如下。

   i. 编译和运行Wi-Fi HAL App。

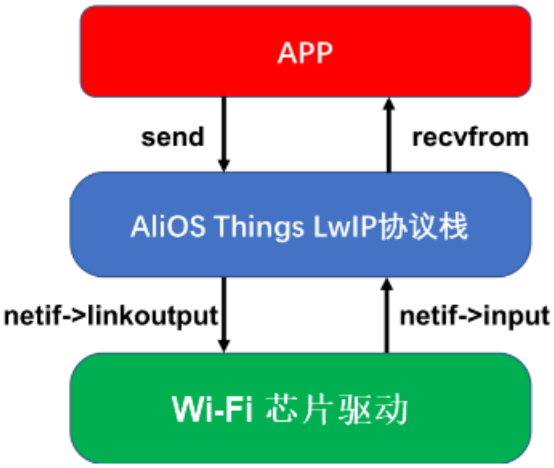
   ii. 在CLI输入 `testwifihal all <AP SSID> <AP password>` 命令。

   正常情况下所有测试子项都通过。若测试不通过，可在Log中搜索 `failed` 关键字，查看具体失败的测试子项，并根据日志排查。

LwIP移植

1. 对接网卡与驱动。

   对接网卡与驱动需要完成底层初始化、输出对接、输入对接。Wi-Fi芯片驱动的原理图如下。



i. 进入*network/lwip/netif/ethernet.c*文件。

ii. 修改以下示例代码。

```
static void low_level_init(struct netif *netif);
static err_t low_level_output(struct netif *netif, struct pbuf *p);
static struct pbuf *low_level_input(struct netif *netif);
```

其中详细的函数说明如下。

- low_level_init

  主要作用：底层初始化。主要工作是填充网卡（netif）信息，例如设置MAC地址、MTU大小、flag标识等。

```
static void low_level_init(struct netif *netif)
{
    u8_t wireless_mac[NETIF_MAX_HWADDR_LEN];
    wifi_get_mac_address((char *)wireless_mac);
    /* set MAC hardware address length */
    netif->hwaddr_len = ETHARP_HWADDR_LEN;
    os_memcpy(netif->hwaddr, wireless_mac, ETHARP_HWADDR_LEN);
    /* maximum transfer unit */
    netif->mtu = 1500;
    /* device capabilities */
    /* don't set NETIF_FLAG_ETHARP if this device is not an ethernet one */
    netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP | NETIF_FLAG_LINK_UP
| NETIF_FLAG_IGMP;
    ETH_INTF_PRT("leave low level!\r\n");
}
```

- `low_level_output` 调用驱动层发送函数。

  主要作用：输出对接。调用驱动层发送函数，该函数指针将赋给 `netif->output` 。

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
    int ret;
    ret = bmsg_tx_sender(p);
    if (ret == 0)
        return ERR_OK;
    else
        return  ERR_TIMEOUT;
}
```

- low_level_input

  主要作用：输入对接。该函数调用Wi-Fi芯片驱动的函数，用于向上交付数据包。

  未实现low_level_input时，即直接在驱动函数里调用 `netif->input` 。

```
void ethernetif_input(int iface, struct pbuf *p)
{
    …
netif = (struct netif *)net_get_netif_handle();
    …
switch (htons(ethhdr->type))
    {
    …
    case ETHTYPE_IP:
        /* full packet send to tcpip_thread to process */
        if (netif->input(p, netif) != ERR_OK)    // ethernet_input
        {
            LWIP_DEBUGF(NETIF_DEBUG, ("ethernetif_input: IP input error\r\n"));
            pbuf_free(p);
            p = NULL;
        }
     break;
    …
}
```

iii. 存放源代码。

修改完成后，源代码需要存放在对应的平台目录*platform/mcu/xxxx*下。以某一设备为例，若其相关修改项在*platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/port/ethernetif.c*文件中。

2. 移植平台相关内容。

平台相关的移植工作，主要定义包括类型定义，大小端设置，内存对齐等（如下表所示）。如果参考实现与开发者实现一致，可以直接拷贝存放在对应的平台 `platform` 下面。

平台相关定义	示例
类型定义	`typedef unsigned char u8_t;`
大小端设置	`#define BYTE_ORDER LITTLE_ENDIAN`

平台相关定义	示例
内存对齐	`#define PACKSTRUCTSTRUCT_attribute((packed))`

以某一设备为例，若其相关修改项在*platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/port/cc.h*目录中。

```
/*
 *   Typedefs for the types used by lwip -
 *   u8_t, s8_t, u16_t, s16_t, u32_t, s32_t, mem_ptr_t
 */
typedef unsigned   char    u8_t;      /* Unsigned 8 bit quantity         */
typedef signed     char    s8_t;      /* Signed    8 bit quantity        */
typedef unsigned   short   u16_t;     /* Unsigned 16 bit quantity        */
typedef signed     short   s16_t;     /* Signed   16 bit quantity        */
typedef unsigned   long    u32_t;     /* Unsigned 32 bit quantity        */
typedef signed     long    s32_t;     /* Signed   32 bit quantity        */
...
```

3. 定制协议栈配置。

   LwIP配置通常在*lwipopts.h*文件中，该头文件放置在平台*platform/mcu/xxxx*目录下。以某设备为例，其相关修改项在*platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/lwipopts.h*目录中，具体配置项如下所示。

```
#define IP_DEBUG                        LWIP_DBG_OFF
#define ETHARP_DEBUG                    LWIP_DBG_OFF
#define NETIF_DEBUG                     LWIP_DBG_OFF
#define PBUF_DEBUG                      LWIP_DBG_OFF
#define MEMP_DEBUG                      LWIP_DBG_OFF
#define API_LIB_DEBUG                   LWIP_DBG_OFF
#define API_MSG_DEBUG                   LWIP_DBG_OFF
#define ICMP_DEBUG                      LWIP_DBG_OFF
#define IGMP_DEBUG                      LWIP_DBG_OFF
#define INET_DEBUG                      LWIP_DBG_OFF
```

4. 编译配置。

   LwIP与OS的对接AliOS Things已经默认完成，您无需关注，只需编译配置即可。

   以某一设备为例，若其相关修改项在*platform/mcu/moc108/aos.mk*目录中，编译配置示例如下。

```
GLOBAL_INCLUDES += mx108/mx378/func/mxchip/lwip-2.0.2/port
$(NAME)_SOURCES += mx108/mx378/func/mxchip/lwip-2.0.2/port/ethernetif.c  \
                   mx108/mx378/func/mxchip/lwip-2.0.2/port/net.c
```

OTA移植

远程空中OTA（Over The Air）升级作为物联网设备的一项基础功能，可以快速修复软件漏洞、更新系统，对于快速迭代的物联网产品是刚性需求。生活物联网平台为您提供一套云端一体化的OTA升级方案，以满足产品功能快速迭代，同时降低产品开发和部署的成本。您只需按照此文档实现移植层接口后，就可以轻松使用生活物联网平台的OTA升级服务。OTA移植的更多介绍请参见OTA移植文档。

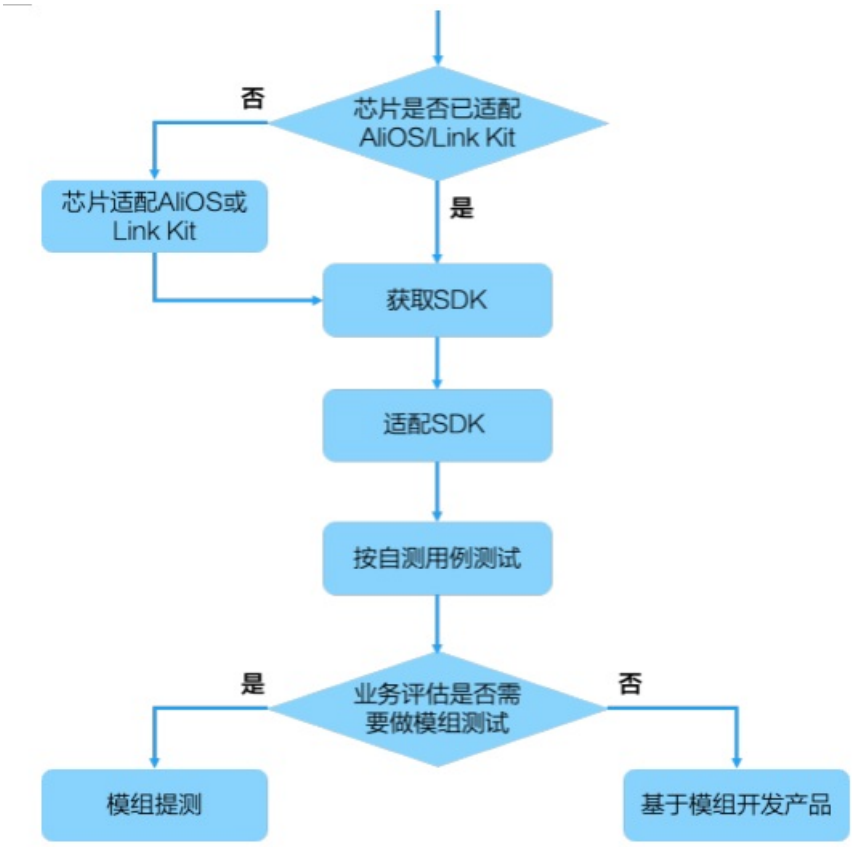
以RTL8710平台为例，OTA相关的平台移植实现代码位于*platform/mcu/rtl8710bn/hal/ota_port.c*目录下。详细移植验证请参见OTA移植demo。

2.2. Wi-Fi模组移植

含AliOS Things的生活物联网平台SDK即包含AliOS Things物联网操作系统（基于AliOS Things V1.3.4）和Link Kit V2.3.0。本文档基于含AliOS Things版本SDK，介绍Wi-Fi模组的移植过程。

概述

基本流程如下图所示。



芯片适配

在适配生活物联网平台SDK前，请确认模组使用的Wi-Fi芯片是否已经支持AliOS或者Link Kit。如果模组使用的Wi-Fi芯片尚未支持AliOS或者Link Kit，可以推动芯片原厂移植生活物联网平台SDK。

生活物联网平台SDK已支持的Wi-Fi芯片如下表所示。

芯片厂商	芯片型号
天猫精灵	TG7100C
翱捷科技	ASR5501
	ASR5502
展锐	RDA5981X
	BK7231

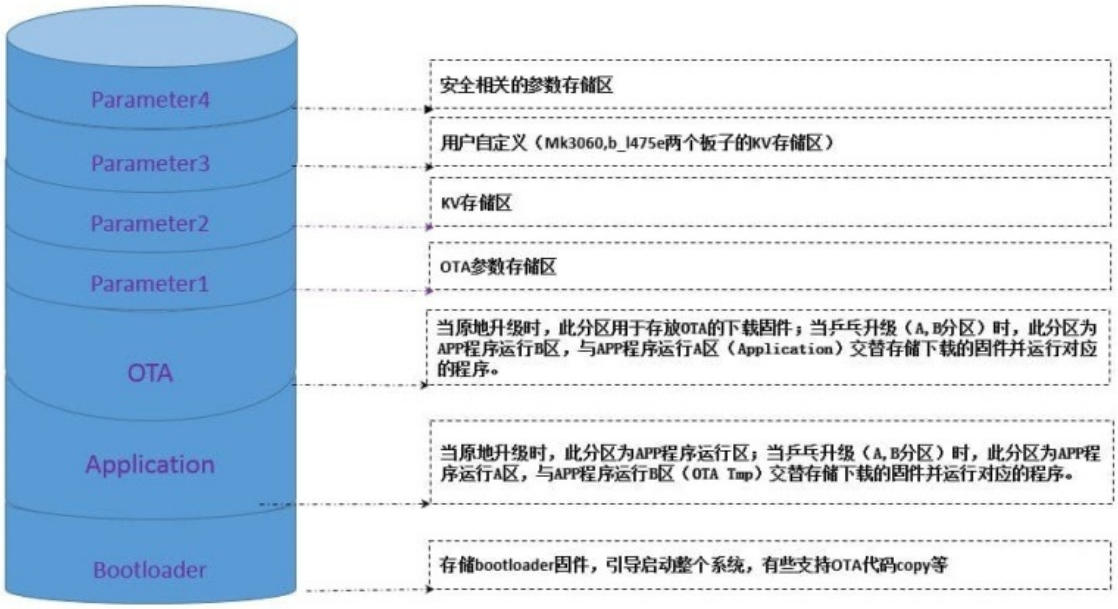
芯片厂商	芯片型号
博通集成	BK7231S
	BK7231U
瑞昱	RTL8710BN
庆科	MOC108
	MX1101

模组移植

1. 获取含AliOS Things的SDK。请参见获取SDK。
2. 业务代码存放到SDK相应的目录下。
3. 划分Bootloader、Flash分区。

   Bootloader和Flash分区是board目录下对应模组文件夹的重要部分。

   AliOS Things由于功能需要维护了一张Flash分区表，这张表包括bootloader区、Application区、OTA TMP区以及parameters区，如下图所示。



   模组可以结合自身Flash大小、Bootloader实现等调整Flash分区。Flash分区大小划分原则如下。

   - 获取芯片平台的Flash大小。
   - 获取bootloader信息，包括：bootloader支持的升级类型（原地或乒乓）、bootloader跳转地址（如果是乒乓会有两个跳转地址）。
   - 跟据bootloader获取的信息，划分整个Flash。

   根据以上划分原则，以HF-LPT230模组为例说明其Flash分区。其芯片RDA5981A，Flash size为1Mbytes。分区表数组 `hal_logic_partition_t hal_partitions[]` 定义在*board/hf-lpt230/board.c*文件中。

分区	地址范围	大小	备注
Bootloader	0x18001000 ~ 0x18004000	12 Kbytes	存储bootloader固件
Application	0x18004000 ~ 0x18095000	580 Kbytes	应用代码分区
OTA Storage	0x18095000 ~ 0x180F7000	372 Kbytes	OTA代码分区
PARAMETER1	0x180F7000 ~ 0x180F8000	4 Kbytes	OTA参数存储区
PARAMETER2	0x180F8000 ~ 0x180FA000	8 Kbytes	KV存储区
PARAMETER3	0x180FA000 ~ 0x180FB000	4 Kbytes	用户自定义
PARAMETER4	0x180FB000 ~ 0x180FC000	4 Kbytes	安全相关的参数存储区（如ID²密钥）
SYS RF Data	0x180FC000 ~ 0x180FD000	4 Kbytes	Wi-Fi模组RF参数
HFILOP	0x180FD000 ~ 0x180FF000	8 Kbytes	模组商用于存放设备证书和MAC地址

模组自测

模组完成SDK移植后，请参见生活物联网平台模组厂家自测用例集 文档规定的自测用例完成模组自测。

自测主要从以下几个方面验证模组功能和稳定性。

- 配网
- 设备控制（云端控制和本地控制）
- 通道稳定性
- 固件升级

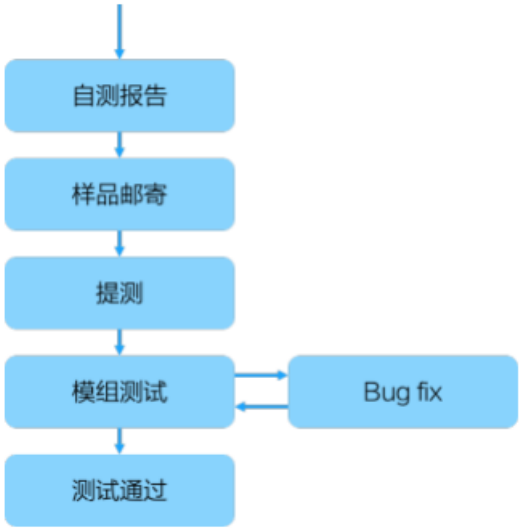
模组提测

模组测试并非强制，完成模组移植生活物联网平台SDK后，即可基于模组开发产品。是否需要做模组测试，请根据实际业务需要，先与阿里云IoT智能生活业务团队或者阿里天猫精灵业务团队确认。

如需做模组测试（提测窗口为：feiyan_certification@alibabacloud.com），您需要提供以下内容。

- 代码库位置（提供board文件夹代码位置）
- 自测报告
- 模组开发板 8块 + 配套供电线 + 串口线（用于获取设备端日志）+ 天线等配件

模组测试流程如下。



常见问题

Q：生活物联网平台SDK与AliOS和Link Kit是什么关系？

A：生活物联网平台SDK是基于AliOS v1.3.4和Link Kit v2.3.0的适合量产的稳定版本，优化了海外连接，并针对生活物联网平台业务进行了定制。

Q：我的模组已经适配过AliOS或者Link Kit别的版本，是否还需要切换生活物联网平台SDK?

A：生活物联网平台SDK针对配网成功率、稳定性、海外连接、生活物联网平台业务定制等方面进行了优化，对于量产产品，特别是要在海外大量出货的产品，需要切换到生活物联网平台SDK，否则可能会出现配网、连云、稳定性、兼容性等问题。

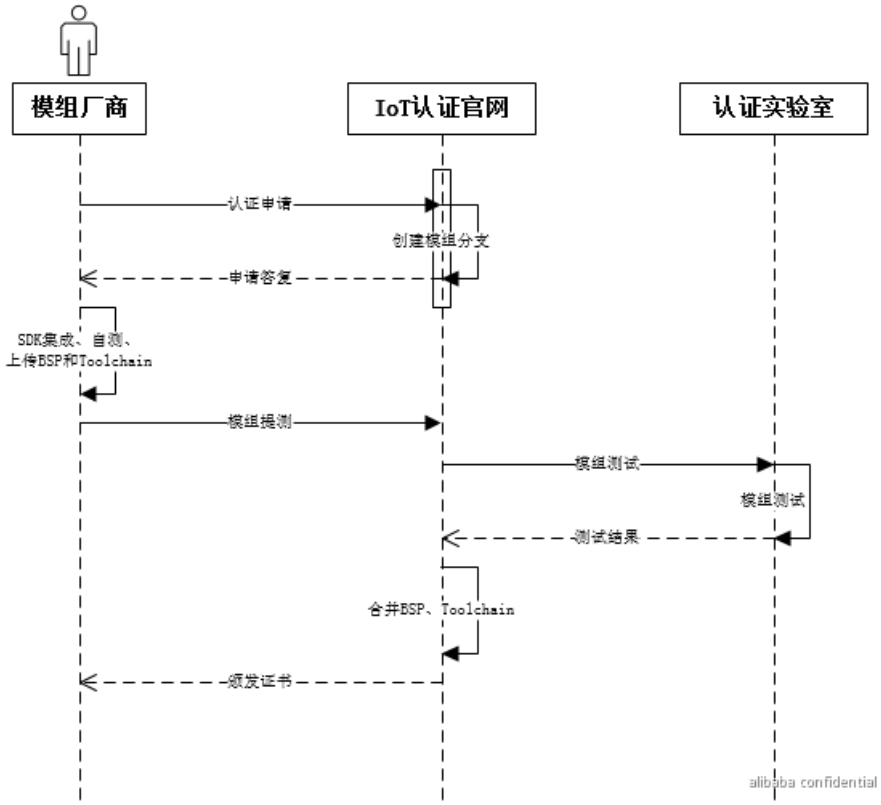
Q：模组如何开始配网绑定？

A：可以编译linkkit app（编译命令 `aos make clean; aos make linkkitapp@yourboardname` ），启动后依次执行**awss**命令和**active_awss**命令，设备会进入配网绑定流程。

2.3. Wi-Fi模组认证指导

模组认证非强制，目前主要由阿里云IoT智能生活业务团队或者阿里天猫精灵业务团队，根据项目需要邀请进行。一般情况下，模组移植生活物联网平台SDK完成后，即可基于模组开发产品。如您收到明确的模组认证的需求，可以根据本文档完成Wi-Fi模组认证。

模组认证的流程图如下。



- 模组厂商：申请进行Wi-Fi模组认证，希望模组具备接入生活物联网平台能力的厂商。
- IoT认证官网：接受模组厂商认证申请的阿里云IoT认证服务。
- 认证实验室：实际执行模组认证的第三方认证机构。

下面从三种角色的角度，分别描述各自在模组认证流程中的操作。

模组厂商

当模组厂商希望其模组成为生活物联网平台的认证模组时，向阿里云IoT认证服务官网提出模组认证申请。

1. 在阿里云IoT认证服务官网，提交模组认证申请。

   提交申请时需要包含以下内容。

申请包含内容	描述
厂商名称	厂商的完整公司名称
厂商地址	厂商公司所在的地址
厂商简介	厂商公司的简介，如公司主营业务、规模以及在行业内的位置等信息
联系人	认证时与阿里云IoT沟通的联系人
联系方式	包含联系人的电话、邮件地址
模组型号	模组的正式型号
模组说明	模组用途、功能说明
Wi-Fi芯片型号	Wi-Fi芯片的生产厂商及芯片

申请包含内容	描述

模组图片	模组的外观图片
售卖链接	模组的客户可以购买该模组的链接
认证实验室列表	与阿里云IoT合作的认证实验室的列表、说明以及联系方式
Username的参数值	访问 https://code.aliyun.com/profile查看Username的参数值，主要用于登录代码仓库，提交BSP代码等

2. 等待认证申请回复。

   认证申请的回复中包含以下内容：

   - 模组认证流程说明
   - 模组厂商用于上传BSP、Toolchain的代码仓库的地址，以及模组厂商用于上传代码的身份信息
   - 模组移植开发指南（含自测说明）
   - 认证实验室列表说明，含认证实验室的介绍及联系方式
3. 按照“认证申请回复”中的模组移植开发指南，完成模组集成生活物联网SDK移植工作。

   如果在模组移植中碰到问题，可以按照“认证申请回复”中提供的技术支持方式进行问题定位和处理。
4. 按照“认证申请回复”中的模组自测内容，测试模组。
5. 模组自测成功后，将模组的BSP、Toolchain上传到“认证申请回复”中指定的位置。
6. 在“认证申请回复”中，选择任一家阿里云IoT提供的认证实验室，并提交模组测试。

   模组提交测试时需包含以下内容：

   - 模组硬件
   - BSP、Toolchain所在位置
   - 模组固件编译、烧写方法
   - 模组串口连接、控制按钮、LED指示说明
7. （可选）在阿里云IoT认证服务官网查询认证进展。
8. 待认证试验室测试通过后，在阿里云IoT认证服务官网获取认证证书。

IoT认证官网

阿里云IoT认证服务是阿里云IoT提供的面向Wi-Fi模组商的一套流程管理服务，用于向模组厂商展示相关的流程、文档、并提供操作页面等。IoT认证服务主要提供以下服务项。

- 处理模组厂商发送的认证申请
  - 记录模组厂商提供的信息。
  - 为厂商的模组创建一个单独的代码上传分支，供模组厂商上传BSP、Toolchain。
  - 回复模组厂商递交的认证申请。
- 处理模组厂商发送的模组提测
  - 验证厂商是否已经按照要求上传BSP、Toolchain、自测结果说明。
  - 模组厂商是否已经选定某个认证实验室进行测试。

- 向指定认证实验室转交认证申请、客户提供的各种信息和硬件。

- 处理认证实验室发送的模组测试结果
  - 留存模组厂商的BSP、Toolchain
  - 为模组厂商生成认证证书

认证实验室

认证实验室是执行模组认证的第三方认证机构，当收到阿里云IoT的模组测试通知后，按以下流程执行。

1. 根据模组厂商的BSP、Toolchain，编译模组固件。
2. 依据阿里云IoT提供的测试案例，测试模组。
3. （可选）测试过程中发现Bug时，向模组厂商反馈。当模组厂商将修复Bug的代码提交到代码仓库后，重新编译固件并回归测试模组。
4. 将测试结果反馈给阿里云IoT认证服务，包含实际测试中的实际测试结果和数据。

2.4. 基于已认证的模组开发

本文讲解设备开发者如何通过阿里云IoT已认证的Wi-Fi模组上开发产品功能，并将设备连接到生活物联网平台。

前提条件

已完成开发环境的安装，请参见编译SDK。

获取SDK代码及编译

1. 获取SDK代码，请参见获取SDK。
2. 解压下载的zip包。
3. 验证aos是否可以正常编译。

   以在某一模组上编译一个living_platform程序为例。

   ```
   ./build.sh example living_platform mk3080
   ```

   以模组名为mk3080为例，完成编译后，会在*out\living_platform@mk3080\binary*目录下生成*living_platform@mk3080.all.bin*文件，该文件即为需要烧写的文件，用户可以将其烧写到模组上查看程序是否可以正常运行。

   > **说明** 可以在board目录，查看已支持的模组。

烧写固件到模组

在编译得到固件后，即可以将固件烧写到模组中执行。不同的模组烧写过程不一样，请联系模组厂商获取烧写工具以及烧写说明。

在生活物联网平台定义产品

1. 创建产品。

   参见创建产品，并按如下要求设置参数。

   - 选择节点类型为设备

- 选择是否接入网关为否
- 选择连网方式为WiFi
- 选择数据格式为ICA标准数据格式（Alink JSON）

2. 产品功能定义。

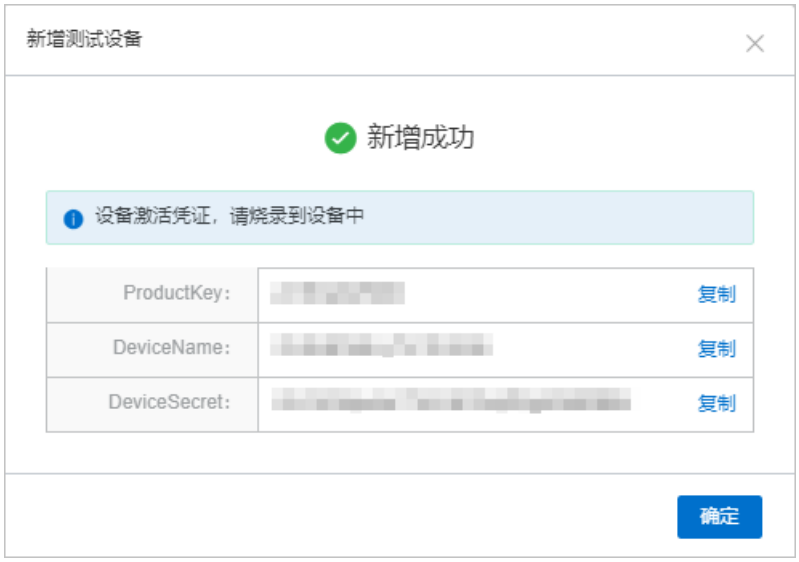
   参见功能概述来定义产品的功能。

3. 调试设备。

   i. 选择已认证的模组。

ii. 新增测试设备，并获取设备激活凭证。

详细操作参见新增测试设备。



4. 在产品的**人机交互**页面，打开使用公版App控制产品的开关，使用生活物联网平台提供的公版App对设备进行控制。App参数的配置请参见配置人机交互。
5. 在**批量投产**页面确认设备信息。

> **说明** 设备开发结束之后才会投产，开发阶段请不要单击**完成开发**。

产品功能开发

1. 开发产品模型。

   对于Wi-Fi设备来说，首先需要通过Wi-Fi配网来获得Wi-Fi热点的SSID/密码。而Wi-Fi配网的移植和调试比较耗费时间，提供以下两种方式，可以实现在产品开发阶段，同时进行产品功能开发和Wi-Fi配网功能移植调试。

   - 让设备直接连接一个指定的热点，然后连接到生活物联网平台，从而可以开始开发与调试产品物模型功能。如下示例代码中注释掉配网的代码。

     > **说明** 被修改的 `start_netmgr()` 函数位于*AliOS-Things/example/linkkit_app/app_entry.c*文件中。

```
int application_start(int argc, char **argv)
{
  ...
#if 0
#ifdef SUPPORT_DEV_AP
     aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
#else
    aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
#endif
#endif
    //下面的代码让设备直接去连接一个指定SSID
    netmgr_ap_config_t config;
    strncpy(config.ssid, "your_ssid", sizeof(config.ssid) - 1);
    strncpy(config.pwd, "your_ssid_password", sizeof(config.pwd) - 1);
    netmgr_set_ap_config(&config);
    netmgr_start(false);
...
  }
```

- 在调试阶段可以用**cli**命令进行配网。当设备连接到Wi-Fi热点，获得一个IP地址之后会自动去连接生活物联网平台。

```
netmgr connect ssid password
```

2. 配置设备身份信息。

   在*linkkit_example_solo.c*的代码中设置了设备的身份信息，设备开发者需要将测试设备的信息对其进行替换。

```
// for demo only
#define PRODUCT_KEY     "a15****PqM"
#define PRODUCT_SECRET  "4uZsr*****zhjPM"
#define DEVICE_NAME     "IFn6******OaI2cJy"
#define DEVICE_SECRET   "qwvShyphC*******NZFjc8S"
```

   设备开发者将设备身份信息设置到程序之后，可以将代码进行编译并遵循模组商提供的烧写工具将固件写入模组，确保设备可以连接到阿里云物联网平台。如果模组提供串口打印输出，当模组连接到阿里云物联网平台后将会输出类似如下的提示信息。

```
[inf] iotx_mc_init(2183): MQTT init success!

[023265]<I> Loading the CA root certificate ...

[023274]<I>  ok (0 skipped)

[023279]<I> Connecting to /a15trrE4PqM.iot-as-mqtt.cn-shanghai.aliyuncs.com/1883...

[023412]<I>  ok

[023414]<I>   . Setting up the SSL/TLS structure...

[023419]<I>  ok

[023423]<I> Performing the SSL/TLS handshake...

[023761]<I>  ok

[023764]<I>   . Verifying peer X.509 certificate..


[023769]<I> certificate verification result: 0x00

[inf] iotx_mc_connect(2533): mqtt connect success!
```

如果模组可以正常连接阿里云物联网平台，在生活物联网平台的商家后台可以看到该设备已激活，以及设备连接到物联网平台的时间信息。

3. 上报产品属性。

   产品的属性发生变化时，需要将变化后的数值上报到物联网平台。属性变化的检测以及上报是由设备开发者定义和实现的。

   示例代码中对应的产品具有一个LightSwitch的属性，类型为bool，还具有一个RGBColor的属性，类型为复合型。在 `linkkit_example()` 函数中调用了 `user_post_property()` 函数，用于上报相关属性，用户可以参照该代码上报产品的属性变化。

```
void user_post_property(void)
{
    static int example_index = 0;
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *property_payload = "NULL";
    if (example_index == 0) {
        /* Normal Example */
        property_payload = "{\"LightSwitch\":1}";
        example_index++;
    } else if (example_index == 1) {
        /* Wrong Property ID */
        property_payload = "{\"LightSwitchxxxx\":1}";
        example_index++;
    } else if (example_index == 2) {
        /* Wrong Value Format */
        property_payload = "{\"LightSwitch\":\"test\"}";
        example_index++;
    } else if (example_index == 3) {
        /* Wrong Value Range */
        property_payload = "{\"LightSwitch\":10}";
        example_index++;
    } else if (example_index == 4) {
        /* Missing Property Item */
        property_payload = "{\"RGBColor\":{\"Red\":45,\"Green\":30}}";
        example_index++;
    } else if (example_index == 5) {
        /* Wrong Params Format */
        property_payload = "\"hello world\"";
        example_index++;
    } else if (example_index == 6) {
        /* Wrong Json Format */
        property_payload = "hello world";
        example_index = 0;
    }
    res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                             (unsigned char *)property_payload, strlen(property_payload
));
    EXAMPLE_TRACE("Post Property Message ID: %d", res);
}
```

4. 上报产品事件。

   如果产品定义了事件，当事件发生时也需要向云端发送事件。事件的检测以及上报由设备开发者实现。

   例如产品定义了一个标识符为Error的事件，该事件还有一个标识符为ErrorCode的输出参数。如下示例代码描述了如何向物联网平台发送一个事件。

```
void user_post_event(void)
{
    static int example_index = 0;
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *event_id = "Error";
    char *event_payload = "NULL";
    if (example_index == 0) {
        /* Normal Example */
        event_payload = "{\"ErrorCode\":0}";
        example_index++;
    } else if (example_index == 1) {
        /* Wrong Property ID */
        event_payload = "{\"ErrorCodexxx\":0}";
        example_index++;
    } else if (example_index == 2) {
        /* Wrong Value Format */
        event_payload = "{\"ErrorCode\":\"test\"}";
        example_index++;
    } else if (example_index == 3) {
        /* Wrong Value Range */
        event_payload = "{\"ErrorCode\":10}";
        example_index++;
    } else if (example_index == 4) {
        /* Wrong Value Range */
        event_payload = "\"hello world\"";
        example_index++;
    } else if (example_index == 5) {
        /* Wrong Json Format */
        event_payload = "hello world";
        example_index = 0;
    }
    res = IOT_Linkkit_TriggerEvent(user_example_ctx->master_devid, event_id, strlen(eve
nt_id),
                                   event_payload, strlen(event_payload));
    EXAMPLE_TRACE("Post Event Message ID: %d", res);
}
```

`linkkit_example()` 函数中的linkkit_ops定义了系统的各种事件处理函数，处理产品回调函数。示例代码如下所示。

```
int linkkit_example()
{
...
      /* Register Callback */
    IOT_RegisterCallback(ITE_CONNECT_SUCC, user_connected_event_handler);
    IOT_RegisterCallback(ITE_DISCONNECTED, user_disconnected_event_handler);
    IOT_RegisterCallback(ITE_RAWDATA_ARRIVED, user_down_raw_data_arrived_event_handler)
;
    IOT_RegisterCallback(ITE_SERVICE_REQUST, user_service_request_event_handler);
    IOT_RegisterCallback(ITE_PROPERTY_SET, user_property_set_event_handler);
    IOT_RegisterCallback(ITE_PROPERTY_GET, user_property_get_event_handler);
    IOT_RegisterCallback(ITE_REPORT_REPLY, user_report_reply_event_handler);
    IOT_RegisterCallback(ITE_TRIGGER_EVENT_REPLY, user_trigger_event_reply_event_handle
r);
    IOT_RegisterCallback(ITE_TIMESTAMP_REPLY, user_timestamp_reply_event_handler);
    IOT_RegisterCallback(ITE_INITIALIZE_COMPLETED, user_initialized);
    IOT_RegisterCallback(ITE_FOTA, user_fota_event_handler);
    IOT_RegisterCallback(ITE_COTA, user_cota_event_handler);
    };
```

关于回调函数的说明如下。

事件	回调函数原型	事件触发条件说明
ITE_CONNECT_SUCC	`int callback(void);`	与云端连接成功时
ITE_DISCONNECTED	`int callback(void);`	与云端连接断开时
ITE_RAWDATA_ARRIVED	`int callback(const int devid, const unsigned char *payload, const int payload_len);`	Linkkit收到raw data数据时
ITE_SERVICE_REQUEST	`int callback(const int devid, const char _serviceid, const int serviceid_len, const char request, const int request_len, char _response, int *response_len);`	Linkkit收到服务（同步/异步）调用请求时
ITE_PROPERTY_SET	`int callback(const int devid, const char *request, const int request_len);`	Linkkit收到属性设置请求时

事件	回调函数原型	事件触发条件说明
ITE_PROPERTY_GET	`int callback(const int devid, const char _request, const int request_len, char _response, int response_len);`	Linkkit收到属性获取的请求时
ITE_REPORT_REPLY	`int callback(const int devid, const int msgid, const int code, const char *reply, const int reply_len);`	Linkkit收到上报消息的应答时
ITE_TRIGGER_EVENT_REPLY	`int callback(const int devid, const int msgid, const int code, const char eventid, const int eventid_len, const char message, const int message_len);`	Linkkit收到事件上报消息的应答时
ITE_TIMESTAMP_REPLY	`int callback(const char *timestamp);`	当Linkkit收到查询时间戳请求的应答时
ITE_TOPOLIST_REPLY	`int callback(const int devid, const int msgid, const int code, const char * payload, const int payload_len);`	Linkkit收到查询拓扑关系请求的应答时
ITE_PERMIT_JOIN	`int callback(const char * product_key, const int time);`	Linkkit收到允许子设备入网的请求时
ITE_INITIALIZE_COMPLETED	`int callback(const int devid);`	设备初始化完成时
ITE_FOTA	`int callback(int type, const char *version);`	Linkkit收到可用固件的通知时

事件	回调函数原型	事件触发条件说明
ITE_COTA	`int callback(int type, const char config_id, int config_size, const char get_type, const char sign, const char sign_method, const char *url);`	Linkkit收到可用远程配置文件的通知时

5. 主循环处理。

   在 `linkkit_example()` 中存在一个循环，其中 `IOT_Linkkit_Yield` 必须周期调用，用于linkkit业务处理。代码如下所示。

```
time_begin_sec = user_update_sec();
    while (1) {
        IOT_Linkkit_Yield(USER_EXAMPLE_YIELD_TIMEOUT_MS);
        time_now_sec = user_update_sec();
        if (time_prev_sec == time_now_sec) {
            continue;
        }
        if (max_running_seconds && (time_now_sec - time_begin_sec > max_running_seconds
)) {
            EXAMPLE_TRACE("Example Run for Over %d Seconds, Break Loop!\n", max_running
_seconds);
            break;
        }
        /* Post Property Example */
        if (time_now_sec % 11 == 0 && user_master_dev_available()) {
            user_post_property();
        }
        /* Post Event Example */
        if (time_now_sec % 17 == 0 && user_master_dev_available()) {
            user_post_event();
        }
        /* Device Info Update Example */
        if (time_now_sec % 23 == 0 && user_master_dev_available()) {
            user_deviceinfo_update();
        }
        /* Device Info Delete Example */
        if (time_now_sec % 29 == 0 && user_master_dev_available()) {
            user_deviceinfo_delete();
        }
        /* Post Raw Example */
        if (time_now_sec % 37 == 0 && user_master_dev_available()) {
            user_post_raw_data();
        }
        time_prev_sec = time_now_sec;
    }
```

`linkkit_example()` 中还包含了如下所示一段演示代码，用于上报所有的属性、事件等，在实际产品开发时需要将其修改为设备商自己的逻辑。

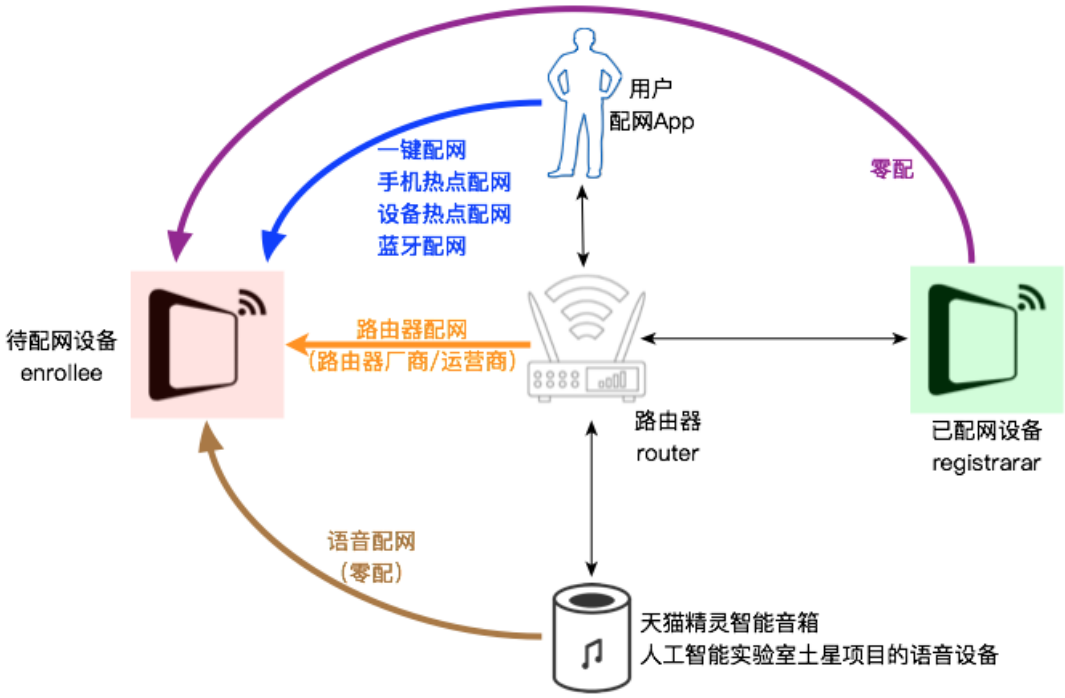
```
while (1) {
        IOT_Linkkit_Yield(USER_EXAMPLE_YIELD_TIMEOUT_MS);
        /* Post Property Example */
        if (user_master_dev_available()) {
            user_post_property();
        }
        /* Post Event Example */
        if (user_master_dev_available()) {
            user_post_event();
        }
    }
```

设备商完成自己物模型功能的代码编写之后，可以将固件编译出来并根据相关模组的烧写方法把固件烧写到模组上进行功能验证和调试。

Wi-Fi配网

配网支持如下所示。

- 一键配网（Smartconfig）：App直接给设备配网
- 手机热点配网（phone-config）：App直接给设备配网
- 路由器热点配网（router-config）：输出到路由器厂商/运营商
- 零配（zero-config）：已配网设备为待配网设备配网
- 设备热点配网（dev-ap）：设备开热点，手机连接设备热点完成为设备配网
- 蓝牙配网（ble-config）：借助BT/BLE为设备配网



注意 在开发Wi-Fi配网时，请确保去除前面章节开发产品物模型时对 `start_netmgr()` 的修改。

```
static void start_netmgr(void *p)
{
    /*
     * register event callback to detect event of AWSS
     */
    iotx_event_regist_cb(linkkit_event_monitor);
    netmgr_start(true);
    aos_task_exit(0);
}
```

详细的开发过程如下步骤所示。

1. 配网对外披露的API列表。

```
/*
 * Copyright (C) 2015-2018 Alibaba Group Holding Limited
 */
#ifndef __IOT_EXPORT_AWSS_H__
#define __IOT_EXPORT_AWSS_H__
#if defined(__cplusplus)  /* If this is a C++ compiler, use C linkage */
extern "C" {
#endif
/**
 * @brief   start Wi-Fi setup service
 *
 * @retval  -1 : Wi-Fi setup fail
 * @retval  0 : sucess
 * @note: awss_config_press must been called to enable Wi-Fi setup service
 */
int awss_start();
/**
 * @brief   stop wifi setup service
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      if awss_stop is called before exit of awss_start, awss and notify will stop.
 *      it may cause failutre of awss and device bind.
 */
int awss_stop();
/**
 * @brief   make sure user touches device belong to themselves
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note: AWSS dosen't parse awss packet until user touch device using this api.
 */
int awss_config_press();
/**
 * @brief   start Wi-Fi setup service with device ap
 *
```

```
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      1. if awss_stop or awss_dev_ap_stop is called before exit of awss_dev_ap_start
 *         awss with device ap and notify will stop, it may cause failutre of device ap
 *         and device bind.
 *      2. awss_dev_ap_start doesn't need to call awss_config_press to been enabled.
 */
int awss_dev_ap_start();
/**
 * @brief   stop Wi-Fi setup service with device ap
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      if awss_dev_ap_stop is called before exit of awss_dev_ap_start
 *      awss with device ap and notify will stop, it may cause failutre of device ap
 */
int awss_dev_ap_stop();
/**
 * @brief   report token to cloud after Wi-Fi setup success
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 */
int awss_report_cloud();
/**
 * @brief   report reset to cloud.
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      device will save reset flag if device dosen't connect cloud, device will fails
to send reset to cloud.
 *      when connection between device and cloud is ready, device will retry to report
reset to cloud.
 */
int awss_report_reset();
enum awss_event_t {
    AWSS_START = 0x1000,       // AWSS start without enbale, just supports device disco
ver
    AWSS_ENABLE,               // AWSS enable
    AWSS_LOCK_CHAN,            // AWSS lock channel(Got AWSS sync packet)
    AWSS_CS_ERR,               // AWSS AWSS checksum is error
    AWSS_PASSWD_ERR,           // AWSS decrypt passwd error
    AWSS_GOT_SSID_PASSWD,      // AWSS parse ssid and passwd successfully
    AWSS_CONNECT_ADHA,         // AWSS try to connnect adha (device discover, router so
lution)
    AWSS_CONNECT_ADHA_FAIL,    // AWSS fails to connect adha
    AWSS_CONNECT_AHA,          // AWSS try to connect aha (AP solution)
    AWSS_CONNECT_AHA_FAIL,     // AWSS fails to connect aha
    AWSS_SETUP_NOTIFY,         // AWSS sends out device setup information (AP and route
r solution)
    AWSS_CONNECT_ROUTER,       // AWSS try to connect destination router
```

```
    AWSS_CONNECT_ROUTER_FAIL,  // AWSS fails to connect destination router.
    AWSS_GOT_IP,               // AWSS connects destination successfully and got ip add
ress
    AWSS_SUC_NOTIFY,           // AWSS sends out success notify (AWSS sucess)
    AWSS_BIND_NOTIFY,          // AWSS sends out bind notify information to support bin
d between user and device
    AWSS_ENABLE_TIMEOUT,       // AWSS enable timeout(user needs to call awss_config_pr
ess again to enable awss)
    AWSS_RESET = 0x3000,       // Linkkit reset success (just got reset response from c
loud without any other operation)
};
#if defined(__cplusplus)  /* If this is a C++ compiler, use C linkage */
}
#endif
#endif
```

2. App中调用配网。

```
/*
 * application_start is application entrance based on sdk.
 */
int application_start(int argc, char **argv)
{
    ......
    /*
     * set device triple ID information before AWSS, otherwise AWSS will fail.
     * HAL_SetProductKey(product_key);
     * HAL_SetProductSecret(product_secret);
     * HAL_SetDeviceName(dev_name);
     * HAL_SetDeviceSecret(dev_secret);
     */
    set_iotx_info();
    /*
     * set log level to print debug information about AWSS
     */
    LITE_set_loglevel(5);  // 5 for debug level
    /*
     * Start netmgr task for AWSS
     * default stack size of netmgr task is 4096B,
     * if the module or die takes more stack size,
     * please set larger stack size (maybe, 6KB)
     */
#ifdef SUPPORT_DEV_AP
    aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
#else
    aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
#endif
    aos_loop_run();
    return 0;
}
#ifdef SUPPORT_DEV_AP
void awss_open_dev_ap(void *p)
{
    iotx_event_regist_cb(linkkit_event_monitor);
    LOG("%s\n", __func__);
    if (netmgr_start(false) != 0) {
        aos_msleep(2000);
        awss_dev_ap_start();
    }
    aos_task_exit(0);
}
#endif
static void start_netmgr(void *p)
{
    /*
     * register event callback to detect event of AWSS
     */
    iotx_event_regist_cb(linkkit_event_monitor);
    netmgr_start(true);
    aos_task_exit(0);
}
```

> 说明
>
> - `set_iotx_info` 一定要在awss_start或netmgr_start之前调用设备证书信息，否则设备无法解析路由器的PASSWORD，PASSWORD采用加密措施保证安全性，而密钥需要借助于设备证书信息计算。
> - `LIET_set_loglevel` 函数如果需要调试打开log，该函数需要在awss_start前调用。
> - awss_start只是开始AWSS服务（发现周围的AP列表），并未使能AWSS服务。如果需要设备解析配网包，考虑安全问题，还需要调用 `awss_press_config` 来使能AWSS（设备热点配网除外）。
> - 关于调用 `awss_press_config` 的时机和策略，虽然Demo中采用按键触发，但产品厂商可以根据自己的产品的特点来自行设计。

目前awss_start已经被AliOS封装在netmgr模块中（netmgr_start），具体可以阅读netmgr_start的代码。

```
int netmgr_start(bool autoconfig)
{
    ......
    /*
     * if the last AP information exists
     * try to connect the last AP.
     */
    if (has_valid_ap() == 1) {
        aos_post_event(EV_WIFI, CODE_WIFI_CMD_RECONNECT, 0);
        return 0;
    }
    ......
     /*
     * if the last AP information doesn't exist
     * start AWSS (Alibaba Wireless Setup Service)
     */
    if (autoconfig) {
        netmgr_wifi_config_start();  // call awss_start()
        return 0;
    }
    ......
    return -1;
  }
```

3. 特定Demo说明。
   - 从设备热点配网切换到其他配网模式，调用do_awss。

```
void do_awss()
{
    aos_task_new("dap_close", awss_close_dev_ap, NULL, 2048);
    aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
}
static void awss_close_dev_ap(void *p)
{
    awss_dev_ap_stop();
    LOG("%s exit\n", __func__);
    aos_task_exit(0);
}
static void start_netmgr(void *p)
{
    iotx_event_regist_cb(linkkit_event_monitor);
    LOG("%s\n", __func__);
    aos_msleep(2000);
    netmgr_start(true);
    aos_task_exit(0);
}
```

- 从其他配网模式切换到设备热点配网，调用do_awss_dev_ap。

```
void do_awss_dev_ap()
{
    aos_task_new("netmgr_stop", stop_netmgr, NULL, 4096);
    aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
}
static void stop_netmgr(void *p)
{
    awss_stop();
    LOG("%s\n", __func__);
    aos_task_exit(0);
}
static void awss_open_dev_ap(void *p)
{
    iotx_event_regist_cb(linkkit_event_monitor);
    LOG("%s\n", __func__);
    if (netmgr_start(false) != 0) {
        aos_msleep(2000);
        awss_dev_ap_start();
    }
    aos_task_exit(0);
}
```

更多的Wi-Fi配网的描述可参见配网开发文档。

云端解绑与恢复出厂默认设置通知

设备被解绑后，云端会下发一个解绑事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。

如果通过App将设备恢复出厂默认设置，云端会下发一个Reset事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Reset"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。设备开发者可以结合具体产品类型，决定收到解绑和恢复出厂默认设置通知后做哪些清空操作。

可以参考示例代码*example/smart_outlet/smart_outlet_main.c*中 `notify_msg_handle` 函数，做如下改动。

```
static int notify_msg_handle(const char *request, const int request_len)
{
    ....
    if (!strcmp(item->valuestring, "awss.BindNotify")) {
        cJSON *value = cJSON_GetObjectItem(request_root, "value");
        if (item == NULL || !cJSON_IsObject(value)) {
            cJSON_Delete(request_root);
            return -1;
        }
        cJSON *op = cJSON_GetObjectItem(value, "Operation");
        if (op != NULL && cJSON_IsString(op)) {
            if (!strcmp(op->valuestring, "Bind")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }
    ....
}
```

设备重置

对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用 `awss_report_reset()` 函数告知云端清除设备与用户的绑定关系。

因此，设备商需要在处理reset按键的逻辑中增加对 `awss_report_reset()` 的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

开发OTA

若使能了OTA功能，请参见OTA编程。

2.5. 基于非认证的模组开发

如果用户不使用已认证的模组，本文讲解用户如何直接通过生活物联网SDK（不含AliOS Things）开发Wi-Fi单品设备，用以连接生活物联网平台。

在生活物联网平台定义产品

1. 创建项目。

   参见创建项目。

2. 创建产品。

   参见创建产品，连网方式选择为WiFi。生活物联网的公版App可以通过本地发现方式来添加设备。

   > **说明**
   >
   > 如果手机App使用生活物联网的公版App对设备进行管理，那么添加设备时如果发现设备是Wi-Fi设备，那么就会进入阿里的Wi-Fi配网过程对设备进行配网，因此设备需要集成阿里的Wi-Fi配网功能。
   >
   > 如果设备并没有集成阿里的Wi-Fi配网功能，而是通过触摸屏输入Wi-Fi热点的SSID/密码或者其它配网方式，联网方式可以选择以太网，以避免生活物联网的公版App使用阿里配网流程来添加设备，但是同时可以使用本地发现方式来添加设备。

3. 产品功能定义。

   参见功能概述来定义产品的功能。

4. 添加测试设备。

   参见设备调试概述来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见获取SDK下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行 `make menuconfig` 来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台
FEATURE_MQTT_DIRECT	○ y：连国内服务器 ○ n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	y：使能本地控制功能
FEATURE_ALCS_SERVER_ENABLED	y：使能本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	y：使能用户绑定相关功能

功能	说明
FEATURE_SUPPORT_TLS	y：使能TLS加密
FEATURE_OTA_ENABLED	y：使能OTA
FEATURE_WIFI_PROVISION_ENABLED	y：使能Wi-Fi配网
FEATURE_AWSS_SUPPORT_DEV_AP	y/n：根据需要打开或关闭设备热点配网
FEATURE_AWSS_SUPPORT_SMARTCONFIG	y/n：根据需要打开或关闭一键配网
FEATURE_AWSS_SUPPORT_PHONEASAP	y/n：根据需要打开或关闭手机热点配网
FEATURE_AWSS_SUPPORT_ROUTER	y/n：根据需要打开或关闭路由器配网
FEATURE_AWSS_SUPPORT_ZEROCONFIG	y/n：根据需要打开或关闭零配配网

设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配
  - 线程相关HAL的适配
  - Wi-Fi配网HAL的适配注：若未选择集成阿里提供的Wi-Fi配网功能，可以不用适配这些HAL
  - OTA HAL的适配
  - 本地通信HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- Wi-Fi配网开发

  请访问生活物联网SDK的Wi-Fi配网开发指南了解如何调用配网相关的API实现Wi-Fi配网。

- 云端解绑与恢复出厂默认设置

  设备被解绑后，云端会下发一个解绑事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。

  如果通过App将设备恢复出厂默认设置，云端会下发一个Reset事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Reset"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。设备开发者可以结合具体产品类型，决定收到解绑和恢复出厂默认设置通知后做哪些清空操作。

  可以参考示例代码*example/smart_outlet/smart_outlet_main.c*中 `notify_msg_handle` 函数，做如下改动。

```
static int notify_msg_handle(const char *request, const int request_len)
{
    ....
    if (!strcmp(item->valuestring, "awss.BindNotify")) {
        cJSON *value = cJSON_GetObjectItem(request_root, "value");
        if (item == NULL || !cJSON_IsObject(value)) {
            cJSON_Delete(request_root);
            return -1;
        }
        cJSON *op = cJSON_GetObjectItem(value, "Operation");
        if (op != NULL && cJSON_IsString(op)) {
            if (!strcmp(op->valuestring, "Bind")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }
    ....
}
```

- 设备重置

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用 `awss_report_reset()` 函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对 `awss_report_reset()` 的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

- OTA开发

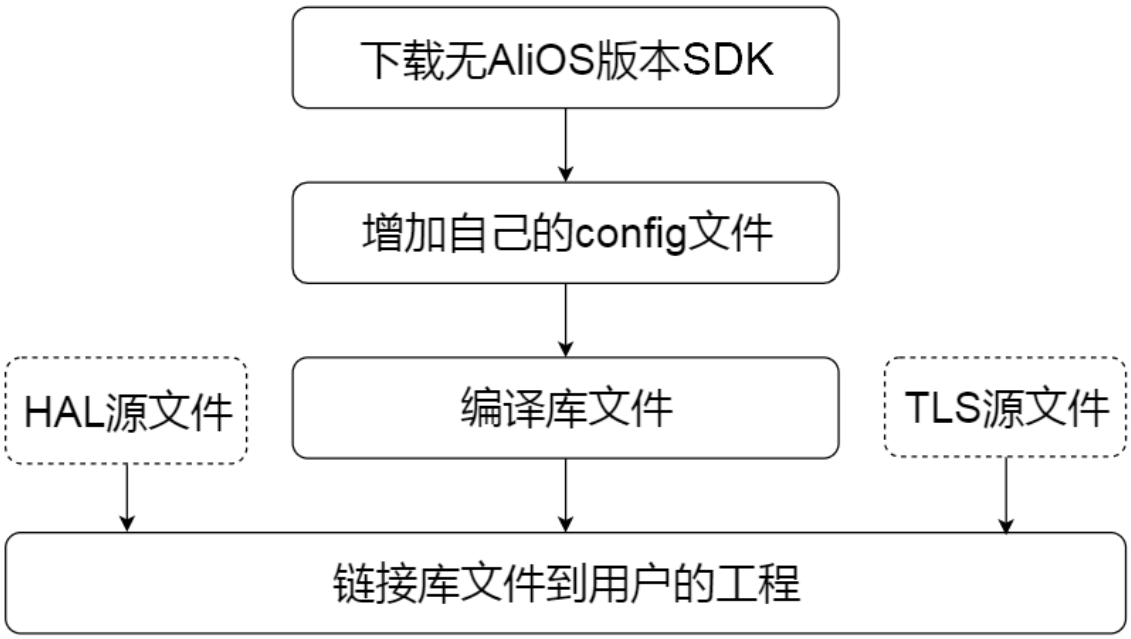
  若使能了OTA功能，请参见OTA编程。

2.6. 无AliOS Things的SDK适配指南

本文介绍无AliOS Things的SDK适配过程。

适配流程

整体流程如下图所示，其中HAL和TLS建议以源文件的形式在用户工程进行编译，以避免在工程中编译产生头文件的依赖问题。



移植步骤

以无AliOS Things的SDK的V1.6.0版本为例。

1. 下载无AliOS Things的SDK的V1.6.0版本。请参见获取SDK。
2. 编译无AliOS Things的SDK。
   i. 配置交叉编译器路径。

   文件*build-rules/settings.mk*中修改 `TOOLCHAIN_DLDIR := /home/mytoolchain` 配置编译器的文件夹所在的路径，然后修改*build-rules/funcs.mk*里面的函数 `Relative_TcPath` 增加编译器的相对路径，如以下代码所示。

```
define
( \
    case $(1) in \
        xtensa-lx106-elf-gcc ) \
            echo "gcc-xtensa-lx106-linux/main/bin" ;; \
        arm-none-eabi-gcc ) \
            echo "gcc-arm-none-eabi-linux/main/bin" ;; \
        nds32le-elf-gcc ) \
            echo "bin" ;; \
    esac \
)
endef
```

ii. 增加config文件。

参考*src/board/*目录下增加一个新的config配置文件，以ESP8266芯片上适配FreeRTOS系统为例，可以增加一个*config.freertos.esp8266*文件，示例代码如下。

```
#添加芯片平台相关的配置
CONFIG_ENV_CFLAGS += \
    -DBOARD_ESP8266 -u call_user_start \
    -fno-inline-functions \
    -ffunction-sections \
    -fdata-sections \
    -mlongcalls \
    -Wl,-static
#配置编译器选项
CONFIG_ENV_CFLAGS += -Os
#配置不需要编译的子模块
CONFIG_src/ref-impl/tls         :=
CONFIG_src/ref-impl/hal         :=
CONFIG_examples                 :=
CONFIG_tests                    :=
CONFIG_src/tools/linkkit_tsl_convert :=
#交叉编译器的前缀，这里不要带路径
CROSS_PREFIX := xtensa-lx106-elf-
```

iii. 编译库文件。

a. 配置SDK的功能。

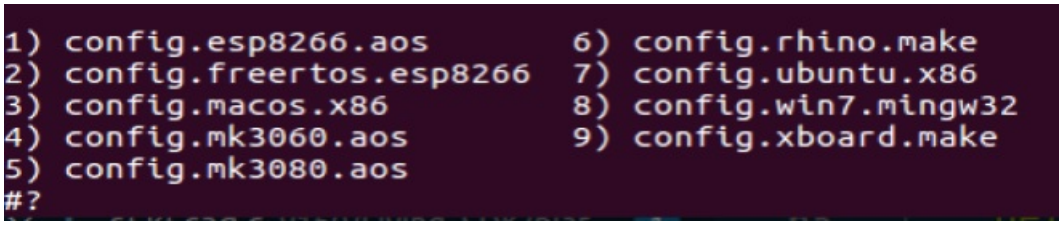
您可以选择以下任一方式：

- 直接编辑根目录下面的*make.settings*文件。
- 执行**make menuconfig**命令配置。

b. 执行**make clean**命令。

c. 执行**make reconfig**选择配置。

如下图输入数字2就是选择config.freertos.esp8266配置。



d. 执行**make**命令。

如果没有编译错误，生成的库文件*libiot_sdk.a*在*output/release/lib*目录下面。

> **说明** 建议您直接在自己的工程中编译HAL和TLS的代码。如果一定要在SDK工程中编译hal，需要先注释掉*config.freertos.esp8266*中的 `CONFIG_src/ref-impl/hal :=` ，同时将hal源文件放在文件夹*src/ref-impl/hal/os/freertos*文件夹中，依赖的头文件放到*prebuild/freertos/include*文件夹中。无AliOS Things的SDK V1.6.0之前的版本头文件依赖有个问题，需要修改根目录下面的*project.mk*文件中的 `IMPORT_DIR` 的值，修改为 `IMPORT_DIR := $(TOP_DIR)/prebuilt` 。

3. 链接库文件到用户工程。

i. 将库文件*output/release/lib/libiot_sdk.a*链接到自己的工程中。

ii. 将目录*output/release/include*下面的头文件放置到自己的工程中，并配置头文件路径。

iii. 在用户的应用程序中添加头文件 `include "iot_import.h"` 。

该头文件会包含SDK的其他头文件。

iv. （可选）如果直接在您自己的工程中编译HAL和TLS的代码，还需在*config.freertos.esp8266*文件中增加以下代码，并执行**make reconfig**生效。

```
CONFIG_src/ref-impl/tls         :=
CONFIG_src/ref-impl/hal         :=
//表示不生成TSL和HAL的库
```

> **说明** 每次修改了*config.freertos.esp8266*文件，都需要执行make reconfig才能生效。

v. 将*example/linkkit/living_platform*下面的文件放到用户的工程中编译，然后启动一个任务执行living_platform_main就可将整个SDK运行起来。

> **说明** SDK运行起来后，即可以调试适配的运行错误，例如coap出错可检查HAL_UDP_xxx.c是否适配好。

make相关命令说明

命令	解释
make distclean	清除一切构建过程产生的中间文件，使当前目录仿佛和刚刚clone下来一样。
make	使用默认的或已选中的平台配置文件开始编译。
make env	显示当前编译配置，非常有用，例如可显示交叉编译链，编译CFLAGS等。
make reconfig	弹出多平台选择菜单，用户可按数字键选择，然后根据相应的硬件平台配置开始编译。
make config	显示当前被选择的平台配置文件。
make menuconfig	以图形化的方式编辑和生成功能配置文件make.settings，直接编辑make.settings文件也是有效的。
make help	打印帮助文本。

用户适配HAL的说明

- 如果是Linux系统，可以直接参考*src/ref-impl/hal/os/ubuntu*目录下面的C文件，大部分文件可以直接使用。
- 如果不是Linux的系统，例如FreeRTOS系统，可以参考*src/ref-impl/hal/os/ubuntu*目录下面的文件实现各个HAL函数的功能。
- 配网相关的HAL函数比较多，详细介绍请参考文档Wi-Fi设备配网适配开发。

3.网关及子设备开发

3.1. 网关开发

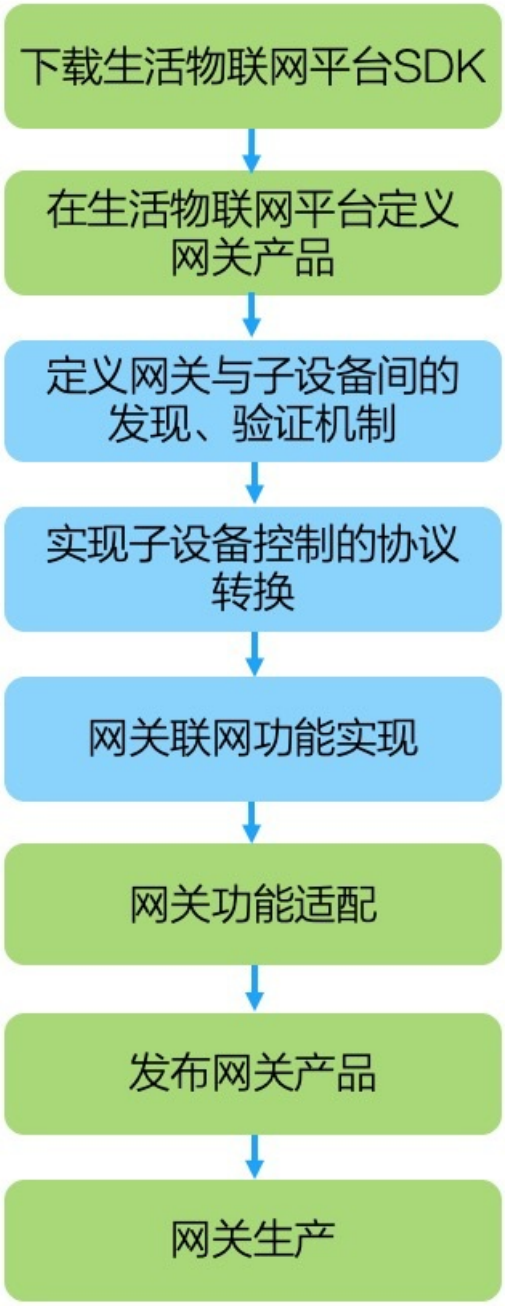
本文介绍如何基于生活物联网SDK开发接入生活物联网平台的网关。

术语解释

- 网关设备：也叫主设备，是指可以挂载子设备的直连设备，网关具有子设备管理模块，维持子设备的拓扑关系，并且可以将拓扑关系同步到云端。
- 子设备：本质上也是设备，但子设备不能直接连接到云端，只能通过挂接到网关上，通过网关间接连云，例如使用zigbee协议联网的设备。
- 设备ID：设备句柄，在网关SDK中用于标识一个具体的设备。
- 设备证书：指设备的ProductKey、DeviceName和DeviceSecret，用于唯一标识设备。
- 拓扑关系：子设备和网关的关联关系为拓扑关系，子设备与网关建立拓扑关系后，便可以复用网关的物理通道进行数据通信。
- 子设备动态注册：子设备无需烧录一机一密（设备证书），只需要烧录ProductKey、ProductSecret，然后基于子设备的唯一标识作为设备证书中的DeviceName（例如SN），再使用网关SDK到云端动态注册获取DeviceSecret，从而得到完整的设备证书。子设备就可以使用该设备证书到云端进行设备的身份认证，并进而被云端统一管理。
- TSL：Thing Specification Language，基于JSON格式，用于描述设备所具备的功能和能力，详细说明参见物模型介绍。

网关开发流程

网关产品的开发流程如下图所示。



- 底色为蓝色的步骤表示该步骤中的功能完全由网关厂商自己定义与实现。
- 网关如何去发现以及连接子设备是由网关厂商与子设备厂商去协商和定义的，平台并不参与网关与子设备之间的通信过程和数据格式定义。但是当网关添加一个子设备后，需要使用平台提供的子设备验证机制来验证子设备的合法性。
- 当手机App对子设备进行控制时，命令将会通过手机发送到云端、再由云端发送到网关设备后转发给子设备。网关与子设备之间的数据格式由网关厂商和子设备厂商定义，因此网关需要进行数据格式从平台格式到厂商数据格式的转换。
- 平台提供的网关SDK并不包含网关如何获取IP地址的功能。网关可能使用以太网、WiFi、或者蜂窝网技术（GRPS、3G、4G等）方式连网，网关上如何通过DHCP获取IP地址、域名服务器地址、下一跳路由器的IP地址等功能由网关厂商实现。
- 如果网关使用WiFi接入网络，并且网关具有串口、或者web server这样的功能可以让用户输入WiFi热点的

SSID/密码，那么网关厂商可以不用适配与集成SDK中的WiFi配网功能。

下载SDK与网关编程说明

- 请参见获取SDK下载SDK。如您的产品需要海外出货，建议下载V1.6.0版本（支持海外子设备的Region切换）。
- SDK中的网关示例代码请参见网关应用示例说明

在生活物联网平台定义网关产品

1. 创建项目。

   参见创建项目。

2. 创建产品。

   参见创建产品，并按照以下要求设置参数。

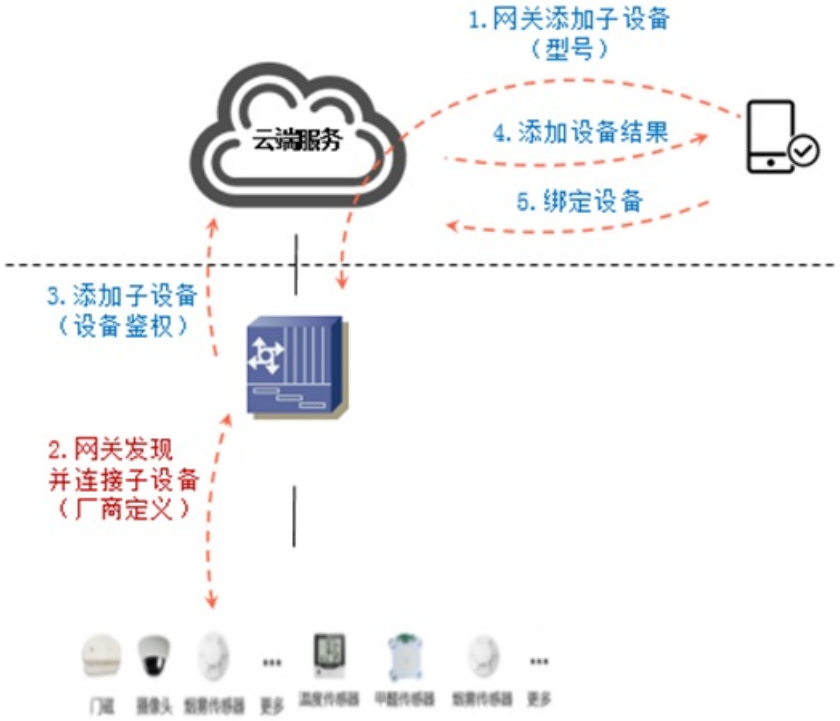
   - 选择所属分类为网络设备中的网关
   - 选择节点类型为网关
   - 根据网关实际连网方式选择连网方式

3. 添加测试设备。

   参见设备调试概述来添加测试设备，并设置设备的绑定策略等。

终端用户添加子设备的交互示意图

本节介绍用户添加子设备的整个交互示意，以让网关厂商了解子设备添加到云端的整体过程。设备用户购买了网关和子设备之后，需要先添加网关，再将子设备添加到网关。完成后才能对子设备进行控制。下面是已添加网关后，添加子设备的多端交互示意图。



当用户发起子设备添加时，手机App将会向云端发送PermitJoin命令，之后云端会将该命令转发给网关，此时网关需要去发现与连接子设备，网关如何去发现与连接子设备是由网关厂商与子设备厂商来定义和实现的。当网关收到PermitJoin后，将发现的子设备上报到云端。

> **说明** 如果网关没有收到来自云端的permitJoin消息，即使子设备已连接到网关，也不要向云端添加子设备；可以等到收到permitJoin之后再将已连接的子设备通知云端。

云端与网关之间数据格式由阿里云IoT定义，网关与子设备之间的数据格式是网关厂商与子设备厂商定义，因此网关需要实现阿里云IoT数据格式与子设备之间数据格式的转换工作。

当云端添加子设备到网关之后，将会发送消息到手机端，从而让用户可以看见添加的子设备，继而对子设备进行控制。

SDK应用于网关时的推荐配置

解压获取到的SDK包后，在主目录下的*make.settings*文件，定义功能模块的需求。对网关而言，除了不同联网方式（WiFi，以太网、蜂窝网）的不同配置之外，还需要配置以下参数的使能。

```
FEATURE_DEVICE_MODEL_GATEWAY=y
```

3.2. 网关应用示例说明

本文介绍生活物联网平台SDK V1.6.0版本中的网关产品示例代码。

请您根据需求下载生活物联网平台SDK。详细下载地址请参见获取SDK。

- 包含AliOS Things的SDK，网关参考应用程序位于*Products/example/linkkit_gateway/*，代码文件如下。

```
gateway_cmds.c
gateway_cmds.h
gateway_entry.c
gateway_entry.h
gateway_main.c
gateway_main.h
gateway_ut.c
gateway_ut.h
gateway_api.c
gateway_api.h
```

- 无AliOS Things的SDK，网关参考应用程序位于*examples/linkkit/gateway/*，代码文件如下。

```
gateway_entry.c
gateway_entry.h
gateway_main.c
gateway_main.h
gateway_ut.c
gateway_ut.h
gateway_api.c
gateway_api.h
```

> **说明** :此处相比于包含AliOS Things版本的SDK，源代码少了*gateway_cmds.c*与*gateway_cmds.h*文件，其他代码一样。

下面详细介绍网关参考应用的文件。

gateway_entry.c

*gateway_entry.c*为应用程序主入口文件。

```
int application_start(int argc, char **argv)
{
... ... ... ... ... ... ... ... ...
  设置默认的日志级别
#ifdef DEFAULT_LOG_LEVEL_DEBUG
    IOT_SetLogLevel(IOT_LOG_DEBUG);
#else
#ifdef CSP_LINUXHOST
    IOT_SetLogLevel(IOT_LOG_DEBUG);
#else
    IOT_SetLogLevel(IOT_LOG_INFO);
#endif
#endif
    加载四元组信息，用户需要实现该函数
    load_device_meta_info();
... ... ... ... ... ... ... ... ...
    注册连云成功回调函数，执行ota初始化
    IOT_RegisterCallback(ITE_MQTT_CONNECT_SUCC, mqtt_connected_event_handler);
    注册串口命令，代码在gateway_cmds.c中
#ifdef CONFIG_AOS_CLI
    gateway_register_cmds();
#endif
... ... ... ... ... ... ... ... ...
    main函数的loop循环
    aos_loop_run();
    return 0;
}
```

gateway_main.c

由于子设备的某些入网操作是同步阻塞式的，该文件里已创建了如下两个线程。

- user_dispatch_yield线程：执行Link Kit SDK代码及注册的用户回调函数。
- gateway_main主线程：调用Link Kit SDK接口，以及处理与子设备间的通信。

gateway_cmds.c

此文件主要包括命令行相关的功能代码，方便测试和调试操作。

gateway_ut.c

此文件为用户测试代码，例如kv模拟子设备信息、get topo list之后上线子设备、permit join的时候模拟添加子设备等。

gateway_api.c

此文件是对子设备操作的SDK接口函数的封装，支持的功能有添加子设备、删除子设备、复位子设备、查询子设备ID等操作，您可以直接使用该接口函数。

- 添加一个子设备

  添加一个子设备需要对SDK进行三次接口调用。

  - 调用 `IOT_Linkkit_Open` ：将子设备添加到网关本地的数据结构。
  - 调用 `IOT_Linkkit_Connect` ：将子设备注册并添加拓扑关系到云端。

- 调用 `IOT_Linkkit_Report` ：告诉云端子设备上线了。

```
GATEWAY_API int gateway_add_subdev(iotx_linkkit_dev_meta_info_t *subdev_mate)
{
... ... ... ... ... ... ... ... ...
    devid = IOT_Linkkit_Open(IOTX_LINKKIT_DEV_TYPE_SLAVE, subdev_mate);
... ... ... ... ... ... ... ... ...
    res = IOT_Linkkit_Connect(devid);
... ... ... ... ... ... ... ... ...
    res = IOT_Linkkit_Report(devid, ITM_MSG_LOGIN, NULL, 0);
... ... ... ... ... ... ... ... ...
}
```

- 批量添加子设备
  - 通过 `IOT_Linkkit_Report( ... ITM_MSG_CONNECT_SUBDEV ...)` 支持最多一次批量添加五个子设备。
  - 如果批量添加成功就调用 `IOT_Linkkit_Report(subdev_id, ITM_MSG_LOGIN, NULL, 0)` 上线对应的子设备。

```
GATEWAY_API int gateway_add_multi_subdev(int master_devid, iotx_linkkit_dev_meta_info_t *
subdev_list, int subdev_num)
{
... ... ... ... ... ... ... ... ...
    connect_times = subdev_num / GATEWAY_SUBDEV_ONE_TIME_CONNECT_MAX_NUM + (((subdev_num
% GATEWAY_SUBDEV_ONE_TIME_CONNECT_MAX_NUM) == 0) ? 0 : 1);
    for (index = 0; index < connect_times; index++)
    {
        ... ... ... ... ... ... ... ... ...
        res = IOT_Linkkit_Report(master_devid, ITM_MSG_CONNECT_SUBDEV, (unsigned char *)p
_cur_subdev, sizeof(iotx_linkkit_dev_meta_info_t) * cur_subdev_num);
        if (res == SUCCESS_RETURN)
        {
            批量上线子设备
            res = IOT_Linkkit_Report(subdev_id, ITM_MSG_BATCH_LOGIN, (unsigned char *)p_c
ur_subdev, sizeof(iotx_linkkit_dev_meta_info_t) * cur_subdev_num);
            ... ... ... ... ... ... ... ... ...
        }
    }
    return res;
}
```

- 删除一个子设备

  调用函数 `IOT_Linkkit_Report(subdev_id, ITM_MSG_DELETE_TOPO, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t))` 删除一个子设备和网关的拓扑关系，同时会释放网关中子设备的资源。

  > **说明** 删除子设备时，优先根据子设备的ProductKey和DeviceName查找子设备，找到就直接删除；如果找不到子设备，则根据第一个参数subdev_id继续查找。

```
GATEWAY_API int gateway_del_subdev(iotx_linkkit_dev_meta_info_t *subdev_mate)
{
... ... ... ... ... ... ... ... ...
    //Here pk and dn of subdev is priorior than subdev id
    ret = IOT_Linkkit_Report(subdev_id, ITM_MSG_DELETE_TOPO, (unsigned char *)subdev_mate
, sizeof(iotx_linkkit_dev_meta_info_t));
... ... ... ... ... ... ... ... ...
}
```

- 复位一个子设备

  调用函数 `IOT_Linkkit_Report(subdev_id, ITM_MSG_SUBDEV_RESET, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t))` 复位一个子设备，复位子设备除了会删除和网关的拓扑关系之外，还会删除子设备和手机用户账号的绑定关系，同时会释放网关中子设备的资源。

  > **说明** 复位子设备时，优先根据子设备的ProductKey和DeviceName查找子设备，找到就直接复位；如果找不到子设备，则根据第一个参数subdev_id继续查找。

```
GATEWAY_API int gateway_reset_subdev(iotx_linkkit_dev_meta_info_t *subdev_mate)
{
... ... ... ... ... ... ... ... ...
    //Here pk and dn of subdev is priorior than subdev id
    ret = IOT_Linkkit_Report(subdev_id, ITM_MSG_SUBDEV_RESET, (unsigned char *)subdev_mat
e, sizeof(iotx_linkkit_dev_meta_info_t));
... ... ... ... ... ... ... ... ...
}
```

- 查询一个子设备的DeviceID

  调用函数 `IOT_Linkkit_Query(master_devid, ITM_MSG_QUERY_SUBDEV_ID, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t));` 根据子设备的ProductKey和DeviceName信息查询一个子设备的DeviceID（数据结构索引，为大于0的整数）。

```
GATEWAY_API int gateway_query_subdev_id(int master_devid, iotx_linkkit_dev_meta_info_t *s
ubdev_mate)
{
... ... ... ... ... ... ... ... ...
    subdev_id = IOT_Linkkit_Query(master_devid, ITM_MSG_QUERY_SUBDEV_ID, (unsigned char *
)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t));
... ... ... ... ... ... ... ... ...
}
```

- 批量上线子设备接口

  该接口函数会将输入的子设备数组拆分成五个一组进行批量上线。

```
GATEWAY_API int gateway_batch_login(int master_devid, iotx_linkkit_dev_meta_info_t *subde
v_list, int subdev_num)
{
... ... ... ... ... ... ... ... ...
    for (index = 0; index < connect_times; index++)
    {
... ... ... ... ... ... ... ... ...
        res = IOT_Linkkit_Report(subdev_id, ITM_MSG_BATCH_LOGIN, (unsigned char *)p_cur_s
ubdev, sizeof(iotx_linkkit_dev_meta_info_t) * cur_subdev_num);
... ... ... ... ... ... ... ... ...
    return res;
}
```

- 批量下线子设备接口

  该接口函数会将输入的子设备数组拆分成五个一组进行批量下线。

```
GATEWAY_API int gateway_batch_logout(int master_devid, iotx_linkkit_dev_meta_info_t *subd
ev_list, int subdev_num)
{
... ... ... ... ... ... ... ... ...
    for (index = 0; index < connect_times; index++)
    {
... ... ... ... ... ... ... ... ...
        res = IOT_Linkkit_Report(subdev_id, ITM_MSG_BATCH_LOGOUT, (unsigned char *)p_cur_
subdev, sizeof(iotx_linkkit_dev_meta_info_t) * cur_subdev_num);
... ... ... ... ... ... ... ... ...
    }
    return res;
}
```

3.3. 子设备开发

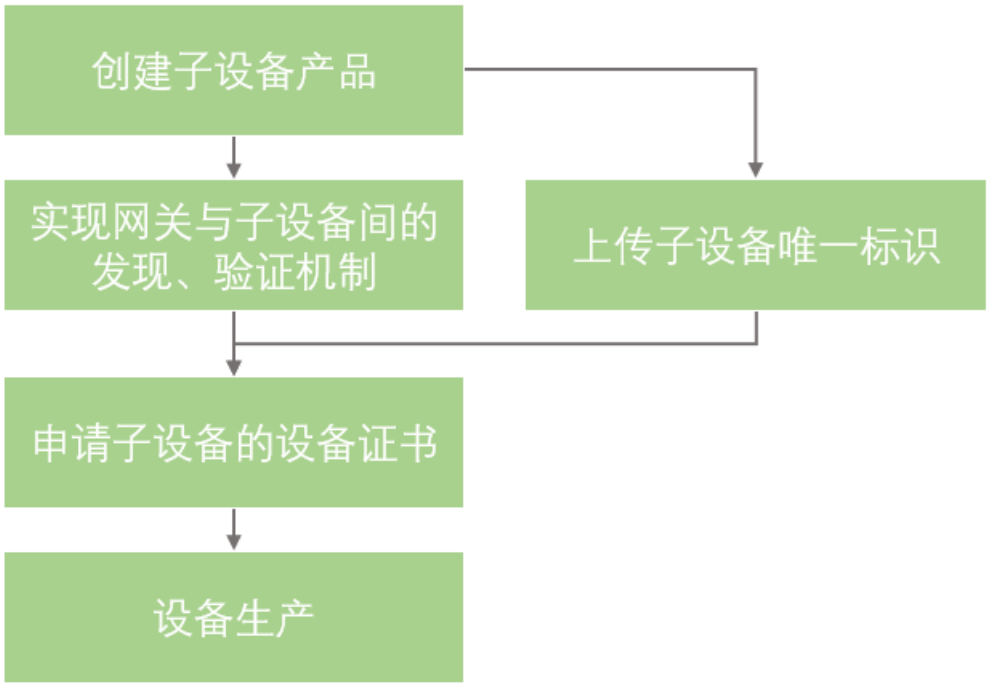
本文介绍通过生活物联网SDK开发接入生活物联网平台的子设备。

术语解释

- 网关设备：也叫主设备，是指可以挂载子设备的直连设备，网关具有子设备管理模块，维持子设备的拓扑关系，并且可以将拓扑关系同步到云端。
- 子设备：本质上也是设备，但子设备不能直接连接到云端，只能通过挂接到网关上，通过网关间接连云，例如使用zigbee协议联网的设备。
- 设备ID：设备句柄，在网关SDK中用于标识一个具体的设备。
- 设备证书：指设备的ProductKey、DeviceName和DeviceSecret，用于唯一标识设备。
- 拓扑关系：子设备和网关的关联关系为拓扑关系，子设备与网关建立拓扑关系后，便可以复用网关的物理通道进行数据通信。
- 子设备动态注册：子设备无需烧录一机一密（设备证书），只需要烧录ProductKey、ProductSecret，然后基于子设备的唯一标识作为设备证书中的DeviceName（例如SN），再使用网关SDK到云端动态注册获取DeviceSecret，从而得到完整的设备证书。子设备就可以使用该设备证书到云端进行设备的身份认证，并进而被云端统一管理。
- TSL：Thing Specification Language，基于JSON格式，用于描述设备所具备的功能和能力，详细说明参见物模型介绍。

子设备产品开发

子设备产品开发流程如下图所示。



在生活物联网平台为子设备创建产品，并按照以下要求设置参数。

- 选择节点类型为设备
- 选择是否接入网关为是
- 根据子设备实际连网方式选择连网方式

说明 每个子设备也需要在智能生活平台上创建产品，并定义该产品支持的属性、事件、服务。子设备产品的注册方式为“动态注册”，每个子设备都需要一个有效且唯一的阿里IoT激活码。目前该激活码不需要烧写到子设备上，但是子设备厂商需要为每个产品申请激活码。子设备的激活码的DeviceName需要由子设备厂商指定，并上传到生活物联网平台，请参见量产设备了解步骤。

4.蜂窝网设备端开发

本文介绍用户如何通过生活物联网SDK开发接入生活物联网的蜂窝网（2/3/4/5G移动通信协议、NB-IoT）设备。

在生活物联网平台定义产品

1. 创建项目。

   参见创建项目。

2. 创建产品。

   参见创建产品，连网方式选择蜂窝（2/3/4G）。

3. 添加测试设备。

   参见设备调试概述来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见获取SDK下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”，了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行 `make menuconfig` 来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台
FEATURE_MQTT_DIRECT	y：连国内服务器 n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	n：关闭本地控制功能
FEATURE_ALCS_SERVER_ENABLED	n：关闭本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	n：关闭用户绑定相关功能
FEATURE_SUPPORT_TLS	y：使能TLS加密
FEATURE_OTA_ENABLED	y：使能OTA

  设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配

- OTA HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- OTA开发

  若使能了OTA功能，请参见OTA编程。

- 设备重置开发

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用 `awss_report_reset()` 函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对 `awss_report_reset()` 的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

5.以太网设备端开发

本文介绍用户如何通过生活物联网SDK开发接入生活物联网的以太网设备。

在生活物联网平台定义产品

1. 创建项目。

   参见创建项目。

2. 创建产品。

   参见创建产品，连网方式选择为以太网。生活物联网的公版App可以通过本地发现方式来添加设备。

3. 产品功能定义。

   参见功能概述来定义产品的功能。

4. 添加测试设备。

   参见设备调试概述来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见获取SDK下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行 `make menuconfig` 来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台
FEATURE_MQTT_DIRECT	○ y：连国内服务器 ○ n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	y：使能本地控制功能
FEATURE_ALCS_SERVER_ENABLED	y：使能本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	y：使能用户绑定相关功能
FEATURE_SUPPORT_TLS	y：使能TLS加密
FEATURE_OTA_ENABLED	y：使能OTA

  设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配

  - 线程相关HAL的适配
  - OTA HAL的适配
  - 本地通信HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- OTA开发

  若使能了OTA功能，请参见OTA编程。

- 云端解绑与恢复出厂默认设置

  设备被解绑后，云端会下发一个解绑事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。

  如果通过App将设备恢复出厂默认设置，云端会下发一个Reset事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Reset"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。设备开发者可以结合具体产品类型，决定收到解绑和恢复出厂默认设置通知后做哪些清空操作。

  可以参考示例代码*example/smart_outlet/smart_outlet_main.c*中 `notify_msg_handle` 函数，做如下改动。

```
static int notify_msg_handle(const char *request, const int request_len)
{
    ....
    if (!strcmp(item->valuestring, "awss.BindNotify")) {
        cJSON *value = cJSON_GetObjectItem(request_root, "value");
        if (item == NULL || !cJSON_IsObject(value)) {
            cJSON_Delete(request_root);
            return -1;
        }
        cJSON *op = cJSON_GetObjectItem(value, "Operation");
        if (op != NULL && cJSON_IsString(op)) {
            if (!strcmp(op->valuestring, "Bind")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }
    ....
}
```

- 设备重置开发

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用 `awss_report_reset()` 函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对 `awss_report_reset()` 的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

6.蓝牙设备开发

6.1. 蓝牙设备端开发

阿里云IoT生活物联网平台对低功耗蓝牙通信类型（BLE）的设备，提供了Breeze协议接入方案，可以便捷实现App-设备-云的完整链路。

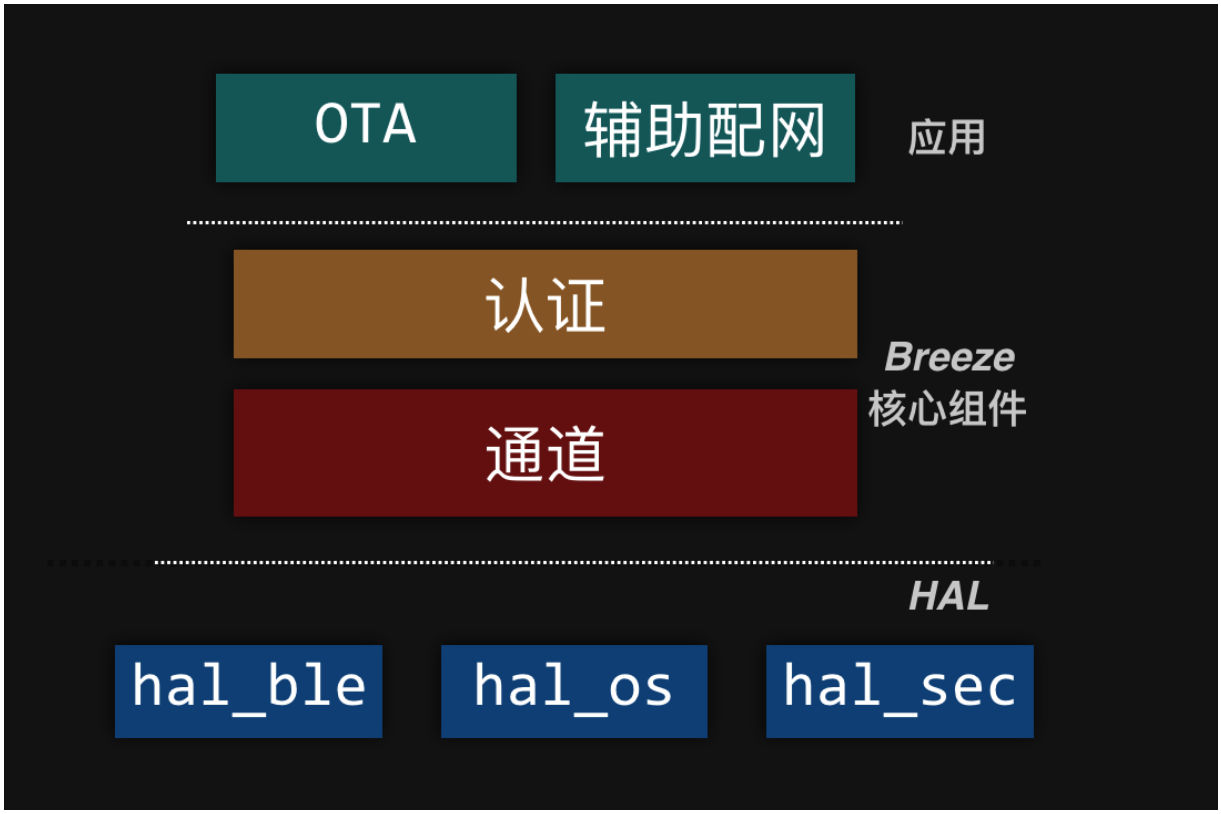
背景介绍

本文主要介绍在具备低功耗蓝牙（BLE）能力的设备端上，如何集成Breeze协议，并与手机端App建立连接、对BLE设备的身份进行认证、建立BLE设备与手机之间的安全通信通道，以及将BLE设备端数据通过手机App通道推送至云端等。手机端客户端和云端开发请参见蓝牙连接开发指南。

Breeze协议类似于BLE协议栈中的Profile（配置文件），需要适配并运行于BLE协议栈之上。

设备端Breeze SDK

Breeze协议规定了接入阿里云IoT生活物联网平台的BLE设备与移动端App之间的通信协议，为了方便用户实现更快速的接入，生活物联网设备端SDK提供了Breeze协议的代码实现：Breeze SDK，其结构框图如下所示。



- HAL层：Breeze SDK需要适配BLE Stack，HAL层就是用来适配BLE Stack之用，不同的BLE Stack需要适当修改进行移植和适配。
- 核心组件层：实现了设备端-移动端-云端的认证、通道等逻辑的源码，用户一般不用做修改即可使用。
- 应用层：Breeze提供了设备OTA通道、辅助Wi-Fi配网通道等，用户需要根据实际场景进行使用。

设备端Breeze SDK代码结构

设备端Breeze SDK随生活物联网设备端SDK对外开源，最新版本请获取最近发布的获取SDK含AliOS Things版本。

代码位于 `$(SDK Src)/framework/bluetooth/breeze` ，目录如下。

```
.
├── Config.in  //配合meunconfig组件配置
├── breeze.mk  //makefile文档
├── README.md  //readme文档
├── api    //目录，展示给应用的接口为breeze_export.*是Breeze SDK提供的接口；breeze_awss_export.*是蓝牙辅助配网的接口。依照应用场景选择其一使用
├── core  //目录，源码核心实现，包含安全，通道以及扩展指令等的实现
├── hal  //参考移植实现，使用AliOS Things OS，BLE协议栈，以及内部mebdtls和安全部分参考
├── include   //Breeze SDK的头文件
```

使用开发指南

开发者可以通过SDK提供的HAL接口进行对接，在自己的运行了BLE协议栈的设备上支持Breeze协议。

移植SDK到第三方平台

`breeze_hal_sec` 对接安全部分，包含AES128 CBC加解密算法和SHA256接口。其余部分代码和应用维持不变。参见源代码目录下的hal，将第三方的实现替换此文件夹下的HAL实现：OS、BLE协议栈、Security，定义在*$(SDK Src)/framework/bluetooth/breeze/hal*下，其余部分代码和应用维持不变。

- breeze_hal_ble：对接BLE蓝牙协议栈部分接口，包含蓝牙广播，注册Breeze蓝牙服务等
- breeze_hal_sec：对接安全部分，包含AES128 CBC加解密算法
- breeze_hal_os：对接不同OS的接口，包含OS timer的资源，KV（存储和读取接口）操作等

API接口列表

详细定义和说明参见蓝牙设备端SDK用户编程接口。

HAL接口列表

详细定义和说明参见蓝牙设备端SDK移植接口。

在生活物联网平台定义产品

由于设备最终要通过生活物联网平台认证，所以需要在生活物联网平台申请对应的产品认证信息。

1. 创建项目。

   参见创建项目。

2. 新建产品。

   参见创建产品，并按照以下要求配置。

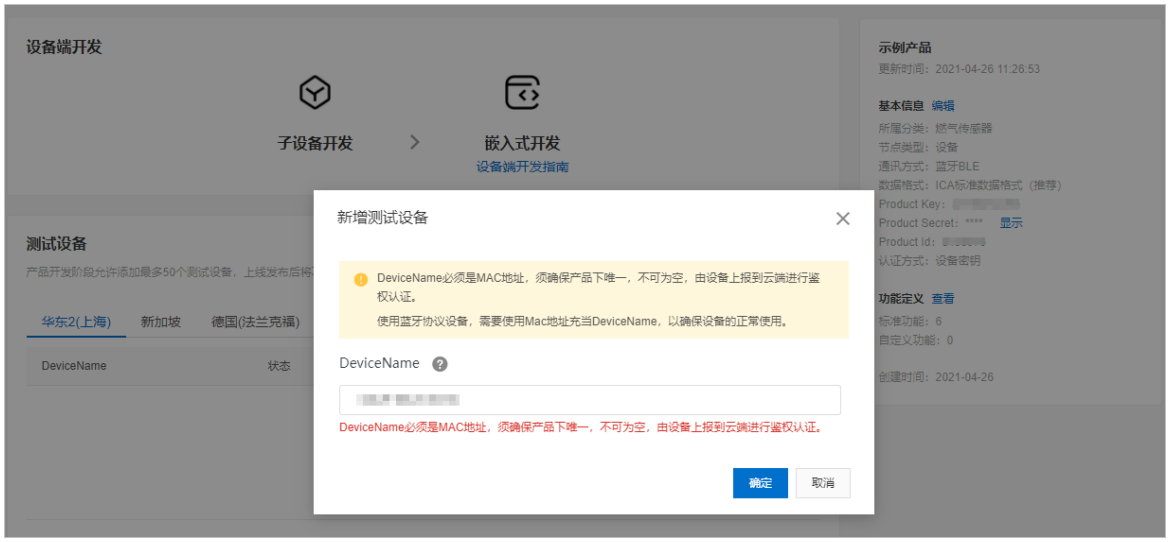
   - 选择节点类型为设备
   - 选择是否接入网关为是
   - 选择接入网关协议为BLE

3. 定义产品功能。

   参见功能概述来定义产品的功能。

4. （可选）在**产品-设备调试**界面，单击**新增测试设备**。

   在产品定义界面右边可以看到产品的三个信息：Product Key、Product Secret和Product ID。这对于一型一密设备来说已经足够，但一机一密还需要在**产品-设备调试**中生成对应同一类型产品的多个设备，根据DeviceName进行区分Device Secret。此时，一机一密需要的五个元素信息都已经获得。



设备端高级应用场景开发

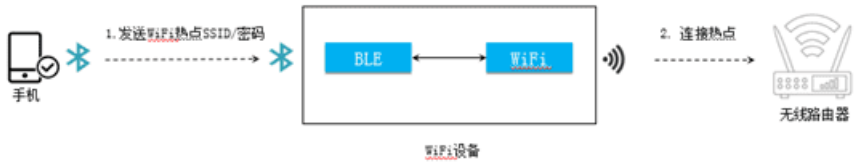
在SDK基础建立安全通道基础上，主要有以下两应用场景：低功耗蓝牙辅助Wi-Fi配网和低功耗蓝牙设备OTA。

蓝牙辅助配网

- 使用场景

  Wi-Fi设备需要连接Wi-Fi热点（Wi-Fi AP）之后才能与其它设备进行IP通信。对于没有键盘、没有触摸屏、没有Web Server的Wi-Fi IoT设备而言，如何获取Wi-Fi热点的SSID、密码是实现设备远程管理的第一个关键步骤。

  我们将Wi-Fi设备获取到Wi-Fi热点的SSID、密码的步骤称为Wi-Fi配网，蓝牙辅助Wi-Fi配网方案适用于同时支持蓝牙BLE以及Wi-Fi的设备，该方案通过蓝牙BLE通道将Wi-Fi热点的SSID/密码发送给Wi-Fi设备，从而让Wi-Fi设备可以连接到Wi-Fi AP（通常为Wi-Fi无线路由器）。蓝牙辅助Wi-Fi配网的工作原理示意图如下。

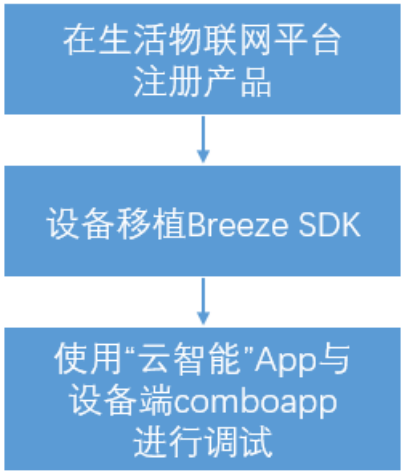


  可以很容易得出蓝牙辅助Wi-Fi配网构成。

  - 设备端SDK：设备端硬件支持Combo芯片（同时支持Wi-Fi和BLE），并运行基于Breeze SDK的蓝牙配网例程。
  - 配网手机端SDK：手机集成配套的蓝牙配网SDK或者“云智能”App。

- 开发流程

  如果您是一个Wi-Fi的模组商或者设备商，您的模组/设备支持蓝牙BLE以及Wi-Fi并且希望集成阿里云IoT生活物联网平台提供的蓝牙辅助Wi-Fi配网功能，可以按照下面的过程进行模组/设备开发。



  - 产品注册，参见设备开发流程，并在产品的**产品-人机交互**的配网引导中增加蓝牙辅助配网的配网方式。
  - 使用“云智能”App与设备端的comboapp开发。如果集成移动端SDK开发自有App的方式，可参见蓝牙连接开发指南。

- 设备端comboapp开发

  在完成设备移植Breeze SDK步骤后，可参考 `example/comboapp` 实现低功耗蓝牙辅助Wi-Fi配网的功能，并将“产品注册”阶段的设备信息设置到comboapp中。

Breeze设备OTA

- OTA编译配置

  可以参考breezeapp示例，默认已经使能OTA功能。

- OTA对接移植

  OTA模块代码实现在 `framework/uOTA/src/device/ble` 下，包含以下目录。

```
.
├── README.md
├── ble.mk
├── inc   //包含了对外的接口和对接API
│   ├── ais_ota.h
│   ├── ota_breeze.h
│   ├── ota_breeze_export.h
│   ├── ota_breeze_plat.h
│   └── ota_breeze_transport.h
└── src   // OTA内部逻辑的实现
    ├── ota_breeze.c
    ├── ota_breeze_plat.c
    ├── ota_breeze_service.c
    └── ota_breeze_transport.c
```

  具体对接函数请参见蓝牙设备端SDK OTA接口。

- OTA调试和使用

OTA的具体用法请参见蓝牙连接开发指南中“运营中OTA配置”章节。

6.2. 蓝牙设备端SDK用户编程接口

Breeze协议提供基于蓝牙链路连接阿里云IoT的安全通道和服务，并且提供蓝牙辅助配网功能（通过BLE链路获取AP的SSID和password）。基本SDK API在*breeze_export.h*中，蓝牙配网的接口API在*breeze_awss_export.h*中。

breeze_start

启动breeze SDK服务。用户使用此接口初始化和启动breeze服务。

参数

名称	类型	描述
dev_conf	device_config	初始化Breeze SDK的信息，包含设备信息，回调函数等。

返回值

0-成功；-1-失败

breeze_end

停止breeze服务，用户调用此接口停止breeze服务。

参数

无

返回值

0-成功；-1-失败

breeze_post

推送设备端状态数据至移动端，使用BLE indicate方式。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

0-成功；其他错误值-失败

breeze_post_fast

和breeze_post类似，推送设备端状态数据至移动端，区别在于使用BLE notify方式。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

0-成功；其他错误值-失败

breeze_post_ext

设备端上报带有cmd字段的数据至移动端。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。
cmd	uint8_t	推送给移动端的cmd类型。

返回值

0-成功；其他错误值-失败

breeze_append_adv_data

广播内容增加用户自定义数据。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

无

breeze_restart_advertising

SDK重启蓝牙广播。

参数

无

返回值

无

breeze_awss_init

该接口对蓝牙配网SDK进行初始化。在用户业务逻辑初始化阶段调用。

参数

名称	类型	描述
cb	apinfo_ready_cb	为设备完成WiFi信息（SSID、密码）获取后的回调函数，由用户定义/提供，并由SDK完成调用。
info	breeze_dev_info_t	为设备信息，包括ProductID，Product Key，Product Secret，Device Name，Device Secret等字段，由用户提供。

返回值

无

breeze_awss_end

停止蓝牙配网服务。

参数

无

返回值

无

6.3. 蓝牙设备端SDK移植接口

Breeze SDK是阿里巴巴IoT提供的基于低功耗蓝牙协议栈上层提供的一套安全接入的解决方案，并可通过蓝牙通道支持WiFi辅助配网功能，本文对Breeze SDK的HAL接口进行说明，厂商对接时可以参考本文档。

HAL接口说明

Breeze SDK的HAL接口分为协议栈、OS、安全三个部分，这些接口的定义在 `$(SDK Src)/framework/bluetooth/breeze/hal` 目录下。

- breeze_hal_ble.h

  定义了蓝牙配网SDK对接到不同厂商蓝牙协议栈的接口，参考实现见 `breeze_hal_ble.c` 文件。

- breeze_hal_os.h

  定义了蓝牙配网SDK对接到不同OS系统的接口，基于AliOS Things的参考实现见 `breeze_hal_os.c` ，用户如果使用AliOS Things时，该部分接口不需要对接。

- breeze_hal_sec.h

  定义了蓝牙配网SDK对接到不同安全算法实现的接口，基于mbedtls的参考实现见 `breeze_hal_security.c` 文件，用户使用AliOS Things时，该部分接口不需要对接。

蓝牙协议栈HAL列表

- ble_stack_init

  该接口完成对BLE协议栈的初始化。

  参数

名称	类型	描述
ais_init	ais_bt_init_t	指定了蓝牙service和characteristics，以及其权限属性等相关设置，包括以下内容。 ○ uuid_svc：AIS服务的uuid（type、value）信息 ○ rc/wc/ic/nc/ wwnrc：AIS服务的characteristic的属性信息 ○ on_connected：蓝牙连接时间的回调函数 ○ on_disconnected：蓝牙断开连接事件的回调函数
init_done	stack_init_done_t	协议栈初始化完成回调函数，需要在蓝牙协议栈完成之后主动调用传入成功参数 `AIS_ERR_SUCCESS(0)`

返回值

该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_stack_deinit

  该接口负责协议栈停止和资源销毁等工作。

  参数

  无

  返回值

  该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_send_notification

  该接口通过notify方式发送数据。

  参数

名称	类型	描述
p_data	uint8_t*	发送数据的buffer地址
length	uint16_t	数据长度

  返回值

  该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_send_indication

  该接口通过indicate方式发送数据。

  参数

名称	类型	描述
p_data	uint8_t*	发送数据的buffer地址
length	uint16_t	数据长度

名称	类型	描述
txdone	callback	回调处理函数，由HAL实现代码在发送indication数据完毕后调用

返回值

该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_disconnect

  该接口断开已有的蓝牙连接。与 `ble_stack_deinit` 的区别是，后续还可以进行连接。

  参数

名称	类型	描述
reason	uint8_t	SDK断开蓝牙连接原因，注意非蓝牙spec规定的断开原因，如Remote User Terminated Connect（0x13）、Connection Accept Timeout Exceeded（0x10）等，需要在实现中对接做一次映射。例如SDK传入 `AIS_BT_REASON_REMOTE_USER_TERM_CONN(0x00)` ，在实现的时候映射成 `BT_HCI_ERR_REMOTE_USER_TERM_CONN(0x13)`

  返回值

  无

- ble_advertising_start

  该接口开启蓝牙广播。

  参数

名称	类型	描述
adv	ais_adv_init_t*	输入参数adv指定了所需的广播信息，包括flag、设备名、厂商数据段内容

  返回值

  该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_advertising_stop

  该接口停止蓝牙广播。

  参数

  无

  返回值

  该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时在*breeze_hal_ble.h*中返回对应的错误编码。

- ble_get_mac

  该接口用于获取蓝牙设备的MAC地址。

  参数

名称	类型	描述
mac	uint8_t*	用于存储获取到的蓝牙MAC地址，6字节二进制格式：0xAA，0xBB，0xCC，0xDD，0xEE，0xFF（对应MAC地址“AA:BB:CC:DD:EE:FF”）

返回值

该函数成功时返回 `AIS_ERR_SUCCESS` ，错误发生时返回Breeze错误编码。

OS HAL列表

OS HAL接口提供以下功能：

- 定时器支持
- 系统重启接口
- 系统时间戳获取和睡眠接口
- Key-Value键值读取接口

OS HAL接口如下。

- os_timer_new

  该接口用于创建定时器。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针
cb	os_timer_cb_t	定时器超时回调处理函数
arg	void**	回调函数参数
ms	int*	超时时间

  返回值

  成功返回0，否则返回非0值。

- os_timer_start

  该接口用于启动定时器。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针

  返回值

  成功返回0，否则返回非0值

- os_timer_stop

  该接口用于停止定时器。

参数

名称	类型	描述
timer	os_timer_t*	定时器指针。

返回值

成功返回0，否则返回非0值

- os_timer_free

  该接口用于销毁定时器资源。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针

  返回值

  成功返回0，否则返回非0值

- os_msleep

  该接口触发系统/进程进行睡眠。

  参数

名称	类型	描述
ms	int	休眠时长，单位为ms。

  返回值

  无

- os_reboot

  该接口用于重启OS。

  参数

  无

  返回值

  无

- os_now_ms

  该接口用于获取系统当前时间戳（从系统启动开始计数）。

  参数

  无

  返回值

  系统时间戳，单位为ms。

- os_kv_get

该接口用于读取永久存储Key-Value键值。

参数

名称	类型	描述
key	const char*	键名（字符串）
buffer	void*	键值（任意值）
buffer_len	int*	键值长度指针

返回值

成功返回0，否则返回非0值

- os_kv_del

  该接口用于删除永久存储Key-Value键值。

  参数

名称	类型	描述
key	const char*	键名（字符串）。

  返回值

  成功返回0，否则返回非0值

- os_rand

  该接口用于产生一个整型的随机值。

  参数

  无

  返回值

  随机有符号整型值。

安全相关的HAL

- ais_aes128_init

  该接口对AES128加解密算法context进行初始化。

  参数

名称	类型	描述
key	const uint8_t*	AES128算法要求的key输入。
iv	const uint8_t*	AES128 CBC算法要求输入的iv值。

  返回值

  `void` 指针，指向初始化后的context，后续加解密流程中使用该返回值

- ais_aes128_destroy

销毁AES128加解密算法context，释放相关资源。

参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。

返回值

成功返回0，否则返回-1

- ais_aes128_cbc_encrypt

  调用该接口进行AES128 CBC加密计算。

  参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。
src	const void*	指针指向需要加密的数据。
block_num	size_t	需要加密的数据块数（16字节为一块，不足16字节按一块计算）。
dst	void*	解密后数据的存储地址。

  返回值

  成功返回0，否则返回-1

- ais_aes128_cbc_decrypt

  调用该接口进行AES128 CBC解密计算。

  参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。
src	const void*	指针指向需要加密的数据。
block_num	size_t	需要加密的数据块数（16字节为一块，不足16字节按一块计算）。
dst	void*	解密后数据的存储地址。

  返回值

  成功返回0，否则返回-1

6.4. 蓝牙设备端SDK OTA接口

Breeze协议提供基于蓝牙链路连接阿里云的安全通道和服务，Breeze OTA是在其基础上实现的BLE固件升级功能。用户移植需要对接的接口和数据结构有：上层应用接口，底层驱动对接接口和bootloader对接数据结构。

上层应用对接API

ota_breeze_service_init

```
int ota_breeze_service_init(ota_breeze_service_manage_t* ota_manage);
```

参数

名称	类型	描述
ota_manage	ota_breeze_service_manage_t*	初始化ota信息，包括版本号，消息接收回调函数等

类型介绍

```
typedef struct _ota_ble_service_dat{
    unsigned char is_ota_enable; /*0:disable ota; 1:enable ota*/
    ota_breeze_version_t verison;/*os version:used to version verify*/
    ota_breeze_get_data_t get_dat_cb;/* breeze to ota send message callback function*/
}ota_breeze_service_manage_t;
```

返回值

成功：0；失败：-1

详细内容查看*ota_breeze_export.h*。

底层驱动对接API

- hal_flash_erase_sector_size

  ```
  uint32_t hal_flash_erase_sector_size(void);
  ```

  参数

  无

  返回值

  返回芯片flash最小擦写单元大小

- hal_flash_erase

  ```
  int32_t hal_flash_erase(hal_partition_t pno, uint32_t off_set, uint32_t size);
  ```

  参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
off_set	uint32_t	相对分区起始地址偏移量

名称	类型	描述
size	uint32_t	擦写大小

返回值

成功：0；失败：-1

- hal_flash_read

```
int32_t hal_flash_read(hal_partition_t pno, uint32_t* poff, void* buf, uint32_t buf_size)
;
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
poff	uint32_t*	相对分区起始地址偏移量
buf	void*	读取的数据数组
buf_size	uint32_t	读取的数据长度

返回值

成功：0；失败：-1

- hal_flash_write

```
int32_t hal_flash_write(hal_partition_t pno, uint32_t* poff, const void* buf, uint32_t bu
f_size);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
poff	uint32_t*	相对分区起始地址偏移量
buf	const void*	写入数据数组
buf_size	uint32_t	写入数据长度

返回值

成功：0；失败：-1

- hal_flash_get_info

```
hal_logic_partition_t *hal_flash_get_info(hal_partition_t pno);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID

返回值

成功：flash分区信息：分区起始地址，长度等；失败：NULL

详细内容查看*flash.h*。

Bootloader对接数据结构

升级完成后，系统会将固件信息等保存flash参数区，AliOS Things 保存在参数分区1；Bootloader 需要读取保存的参数，完成固件的copy和替代，实现升级。

```
typedef struct
{
    unsigned int settings_crc32;   /*settings crc32 value*/
    unsigned int ota_flag;         /*flag ota status*/
    unsigned int src_addr;         /*image storage address*/
    unsigned int dest_addr;        /*iamge run address*/
    unsigned int image_size;       /*image size*/
    unsigned int image_crc32;      /*image crc32*/
    unsigned int image_has_copy_len;   /*remember the copy len for bootloader off line*/
    unsigned char reserved_buf1[RESERVED_SIZE1]; /*reserved*/
    /*backup image include store addr, recovery bank addr, image size and image crc32*/
    unsigned int  backup_store_addr;
    unsigned int  backup_dest_addr;
    unsigned int  backup_image_size;
    unsigned int  backup_image_crc32;
    unsigned int  backup_has_copy_len;           /*remember the copy len for off line*/
    unsigned char reserved_buf2[RESERVED_SIZE2]; /*reserved*/
    /*down_loader image parameters bootloader don't need to care*/
    unsigned int break_point_offset;                  /*breakpoint save when ota processin
g.*/
    unsigned char version_store_buf[VERSION_BUF_SIZE]; /*version save*/
    unsigned short image_info_crc16;                  /*image info crc value*/
    unsigned char bin_type;                           /*Image type*/
    unsigned char reserved_buf3[RESERVED_SIZE3];
} ota_settings_t;
```

6.5. 蓝牙Mesh设备开发指南

6.5.1. 蓝牙Mesh模组软件规范

本文在蓝牙技术联盟发布的Mesh Profile的基础上，做了一些协议上的定制，细化了部分协议规则，作为蓝牙Mesh模组软件规范，指导芯片或者模组厂家软件设计，开发**智能单品**并对接生活物联网平台，包括自有品牌项目与天猫精灵生态项目均遵循本规范。

名词解释

名词	描述
蓝牙Mesh	蓝牙网状网络连接技术。
BLE	Bluetooth Low Energy 蓝牙低功耗技术。
CID	Company Identifier，公司标识符。阿里巴巴的标识符为0x01A8。
Relay	中继功能：中继节点能在蓝牙Mesh网络中接收和转发其他设备的消息，通过消息在节点之间的中继，实现更远距离的传输和更大规模的网络。承担这一角色的节点一般需要固定的电源和一定的计算资源。
Proxy	代理功能：代理节点能够实现GATT和蓝牙Mesh节点之间的Mesh消息发送与接收。通过Proxy代理使得BLE蓝牙设备能通过GATT接入Mesh网络。承担这一角色的节点一般需要固定的电源和一定的计算资源。
Unprovisioned Device	未配网设备，还未加入蓝牙Mesh网络的设备。
Node	节点，加入蓝牙Mesh网络的设备。
Provisioning	配网过程，设备加入蓝牙Mesh网络的过程。
Low Power	低功耗功能：功率受限的节点可能会利用低功耗特性来减少无线电接通时间并节省功耗。同时低功耗节点（LPN）可以与friend节点协同工作。
LPN	Low Power Node，低功耗节点。
Friend	功率不受限的节点很适合作为Friend节点。Friend节点能够存储发往低功耗节点（LPN）的消息和安全更新；当低功耗节点需要时再将存储的信息传输至低功耗节点。
GLP	Genie Low Power，精灵低功耗方案。
NetKey	蓝牙Mesh网络中用于区分不同网络的关键密钥。
AppKey	蓝牙Mesh网络中用于区分同一网络中不同应用的关键密钥。
PB-ADV	一种通过蓝牙Mesh beacon给Mesh设备进行配网的交互方式。
PB-GATT	一种通过蓝牙GATT给Mesh设备进行配网的交互方式。
Mesh网关	• 在天猫精灵生态项目中，Mesh网关包括天猫精灵音箱与天猫精灵App。 • 在自有品牌项目中，Mesh网关包括自有品牌项目Mesh网关产品与云智能App。

基本要求

- 蓝牙Mesh设备必须支持PB-ADV、PB-GATT。
- 蓝牙Mesh设备必须支持Mesh协议里定义的Proxy功能。
- 非低功耗蓝牙Mesh设备必须支持Mesh协议里定义的Relay功能，并且可以被正确打开和关闭。
- LPN和Friend功能可选。
- 低功耗控制类设备（例如支持开关属性，支持下行控制）采用精灵低功耗（GLP）方案。
- 蓝牙Mesh设备要求至少支持1个NetKey、2个AppKey。

- 特殊产品如果仅需支持部分功能将在产品规范中明确说明具体需要支持的功能。

UUID格式

UUID应用于Unprovisioned Device Beacon广播和PB-GATT广播中。广播包中的Device UUID是识别设备的关键信息，UUID中各字段采用小端模式进行存储，对Device UUID的定义如下表所示。

字段	size（字节数）	描述
CID	2	Company Identifier，公司标识符。阿里巴巴的标识符为0x01A8。

PID	1	• bit3～0：蓝牙广播包版本号，目前是0x01
		• bit4为1：一机一密
		• bit5为1：支持OTA
		• bit7～6：蓝牙协议版本
		  ◦ 00：BLE4.0
		  ◦ 01：BLE4.2
		  ◦ 10：BLE5.0
		  ◦ 11：BLE5.0以上

ProductID	4	生活物联网平台颁发，一型一号。
MAC地址	6	生活物联网平台颁发，一机一号，蓝牙Mesh设备的MAC地址与Device Name一致。

FeatureFlag	1	• bit0：广播状态
		  ◦ 0：处于未配网广播状态
		• bit7～1：UUID版本号，目前版本为1
		  ◦ 0000000：该版本配网流程遵循蓝牙Mesh标准规范；已不推荐使用该版本。
		  ◦ 0000001：该版本优化了配网流程并需要支持Vendor Model，参考蓝牙Mesh设备扩展协议。
		  ◦ 0000010：该版本支持版本1的所有约定，并仅限Genie Mesh SDK使用。

FeatureFlag1	1	• bit0：极速配网标志位，不支持极速配网必须置0 • bit1：扩展AuthValue标志位 • bit2：BLE广播配网标志位，不支持该方案必须置0 • bit3：Combo Mesh标志位，不支持Combo Mesh必须置0，仅在Genie Mesh SDK中使用 • bit7～4：Genie BT Mesh Stack标志，未使用Genie Mesh SDK的必须置0
FeatureFlag2	1	• UUID版本号0～1：0x00 Reserved，必须置0 • UUID版本号大于2：FeatureFlag2字节在Genie Mesh SDK中使用

> 说明
>
> - 上表中的RFU保留字段和要求置0的字段必须全部置为0。在产品生命周期中，当产品的ProductID、MAC地址未变化时，CID、PID不允许发生变化。
> - Genie Mesh SDK获取与对接的联系方式请参见获取SDK。

蓝牙Mesh广播包格式

- 未配网广播 Unprovisioned Device Beacon

  蓝牙Mesh设备上电后如处于未配网状态，需要广播Unprovisioned Device Beacon，每次广播时长不小于120ms，广播间隔500ms，广播持续时间一般为10分钟，具体产品的广播启动方式和广播持续时间以产品需求为准。如果超时后仍未被配网，设备关闭PB_GATT广播。

  未配网广播数据示例如下：

0x14	0x2B	0x00	A8 01 71 81 02 00 00 55 43 CB 07 DA 78 02 00 00	0x00	0x00
	Mesh Beacon	Unprovisioned Device beacon	Device UUID	OOB Information	
Length	Type	Beacon Type	Beacon Data		
Advertising data header		Advertising data payload			

- PB-GATT广播

  不支持PB-ADV的设备，如手机，需要先和节点建立GATT连接，通过Mesh Provisioning Service通信。因此建立GATT连接之前，设备需要广播PB-GATT的广播包。GATT广播包中的UUID也同样参照UUID格式。PB-GATT广播数据格式参考Mesh Profile Specification。

  PB-GATT广播发送规则参照BLE广播发送规则，每次广播间隔建议100ms～160ms。

- AIS广播

  AIS广播请参考蓝牙BLE基础规范。

  AIS广播发送规则参照BLE广播发送规则，每次广播间隔建议100ms～160ms。

- Proxy广播

  Proxy广播数据格式参考Mesh Profile Specification。

  Proxy广播发送规则参照BLE广播发送规则，每次广播间隔建议100ms～160ms。

广播包发送规则

- 设备状态与切换规则

  设备状态包括未配网广播状态、配网状态、已配网状态。

  - 当未配网设备上电后，设备需要进入未配网广播状态；广播持续时间默认10分钟，具体产品的广播启动方式和广播持续时间以产品需求为准。

- 如果设备通过App解除配网，或者设备通过硬件复位方式清除配网信息（包括Mesh网络信息、心跳配置、组播配置）后，设备需要重新进入未配网广播状态。
- 当设备响应连接建立请求（回复link ack或者通过PB-GATT建立连接）后，设备进入配网状态。
- 当设备完成配网流程（以Provision完成配置为标志事件）则进入已配网状态；否则恢复至未配网广播状态。

- 未配网广播状态

  当设备处于未配网广播状态时，设备需要同时广播Unprovisioned Device Beacon、PB-GATT广播、AIS广播。

- 配网状态

  设备同一时间只允许建立一个配网连接（Mesh或者GATT）。当设备处于配网状态，停止广播Unprovisioned Device Beacon及其他GATT广播包。

- 已配网状态

  当设备处于已配网状态，要求设备不再响应配网连接建立请求，并同时发送proxy广播和AIS广播。

配网流程

蓝牙Mesh设备配网流程遵循蓝牙Mesh标准的Provisioning流程，其中几个使用自定义数据的步骤描述如下。

- Provisioning Capabilities阶段。

  蓝牙Mesh设备在Provisioning Capabilities阶段提供OOB（Out of Band）方式，要求唯一支持Static OOB方式，其中的AuthValue计算过程如下：

扩展AuthValue标志位取值	AuthValue计算方法
0	`AuthValue = SHA256(Product ID,MAC,Secret)` 前16字节
1	`AuthValueProvisioner = SHA256(ProductID,MAC,Secret,RandomProvisioner)` 前16字节，`AuthValueDevice = SHA256(ProductID,MAC,Secret,RandomDevice)` 前16字节

  即将设备证书的ProductID，MAC，Secret，配网过程中的随机数通过字符串用英文逗号连接，然后进行SHA256摘要计算，取前16字节。注意这里用于计算SHA256的英文字母全部为小写。**注意自有品牌项目扩展AuthValue标志位必须为1。**

  扩展AuthValue标志位为0时的SHA256计算示例如下：

数据字段	数据格式与示例	计算使用的输入字符串
Product ID	十进制数值：168930，对应十六进制数值：0x293e2	"000293e2"
MAC地址	"AB:CD:F0:F1:F2:F3"（扫描到的蓝牙设备MAC地址）	"abcdf0f1f2f3"
Device Secret	"53daed805bc534a4a93c825ed20a7063"	"53daed805bc534a4a93c825ed20a7063"
连接后字符串Device	"000293e2,abcdf0f1f2f3,53daed805bc534a4a93c825ed20a7063"	

数据字段	数据格式与示例	计算使用的输入字符串
SHA256结果输出（HEX）	c1c76741553236fb7da0a586e62298c231dac2885e735feba6b8b4417c 7d9e72	
AuthValue（HEX）	c1c76741553236fb7da0a586e62298c2	

扩展AuthValue标志位为1时的SHA256计算示例如下：

数据字段	数据格式与示例	计算使用的输入字符串
Product ID	十进制数值：168930，对应十六进制数值：0x293e2	"000293e2"
Mac Address	"AB:CD:F0:F1:F2:F3"（扫描到的蓝牙设备MAC地址）	"abcdf0f1f2f3"
Secret	"53daed805bc534a4a93c825ed20a7063"	"53daed805bc534a4a93c825ed20a7063"
RandomDevice	"7889b0af417b967bdcd7b814d2bbffaf"	"7889b0af417b967bdcd7b814d2bbffaf"
RandomProvisioner	"aa484b099ae4c7762fcb1b71968ba7df"	"aa484b099ae4c7762fcb1b71968ba7df"
连接后字符串Device	"000293e2,abcdf0f1f2f3,53daed805bc534a4a93c825ed20a7063,7889b0af417b967bdcd7b814d2bbffaf"	
SHA256结果输出Device(HEX)	0f66e504a5d3609c06db70162345a0375d9b9d71d4cbe346a1c55d5c25738329	
AuthValueDevice(HEX)	0f66e504a5d3609c06db70162345a037	
连接后字符串Provisioner	"000293e2,abcdf0f1f2f3,53daed805bc534a4a93c825ed20a7063,aa484b099ae4c7762fcb1b71968ba7df"	
SHA256结果输出Device(HEX)	f45f15d1e38339b79fda5b63c3970ff01ba40285fe20b303ff332f54f8de729c	
AuthValueDevice(HEX)	f45f15d1e38339b79fda5b63c3970ff0	

- Provisioning Confirmation阶段。

  这个阶段，生活物联网平台和Mesh设备会使用Static OOB方式来做认证，如果生活物联网平台和Mesh设备两边计算得到的Confirmation值不相同，则认证失败，结束流程。 Mesh协议在此阶段中有一个步骤是设备端生成一个随机数并发送给Mesh网关，Mesh网关会把这个随机数发送给云端鉴权，生活物联网平台会保存设备端每次发送的随机数，如果设备端发送的随机数是之前使用过的，则平台将会拒绝该设备配网，所以务必保证每次生成的随机数都不重复。

- Provisioning Data阶段。

  对于多Element设备，生活物联网平台在该阶段只会下发Primary Element的Unicast Address。其余Element的Unicast Address则根据上一Element的地址自行增加1。

- Provision完成配置阶段。

  生活物联网平台在Provisioning Complete之后，会下发AppKey。其余配置根据UUID版本有以下区别。

UUID版本	配置方式
0	生活物联网平台会下发一次主element的Config_Model_App_bind，设备需要返回成功的status，否则会配网失败；剩余model需要自行绑定AppKey。生活物联网平台会下发一次主element的Config_Model_Subscription_Add，设备需要返回成功的status，否则会配网失败；剩余model需要自行绑定Subscription Address。
1，2	生活物联网平台不会下发Config_model_app_bind和Config_Model_Subscrption_Add消息。IoT设备需要自行给所有Element的所有model绑定下发的AppKey，并根据产品类型为各个model订阅相应的组播地址（具体品类组播地址请参阅各产品软件规范）。蓝牙Mesh设备完成配网后需要进行消息上报，上报消息包括该设备所有支持的可上报的属性。

> **说明** 当前三方接入主要使用UUID版本1，版本0已不推荐使用。版本2仅限Genie Mesh SDK使用。

Mesh数据发送和接收

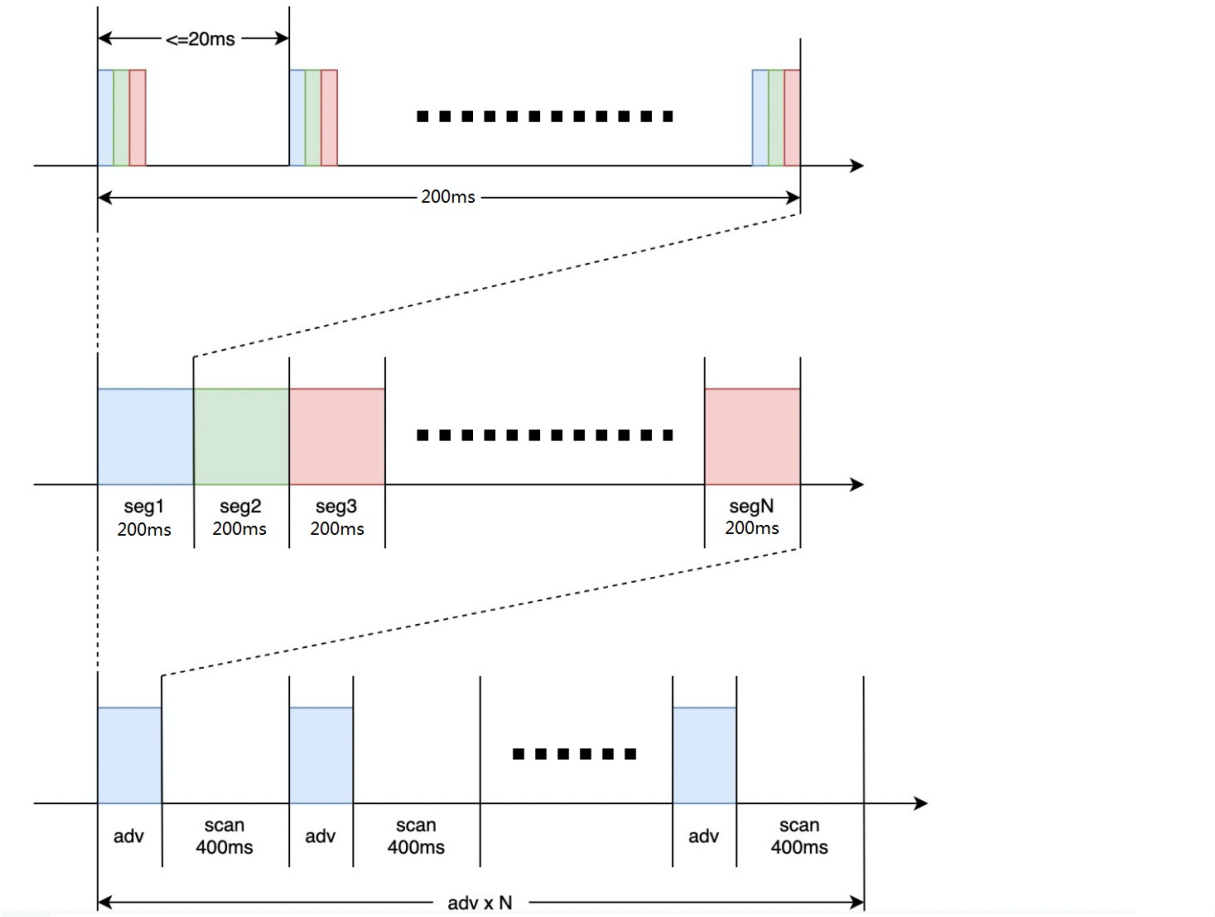
- 数据接收

  通常情况下，我们建议设备端的scan window、interval、period分别为10ms、10ms、100ms。

- 数据发送

  Mesh数据发包每次持续时间200ms左右，每次发包间隔20ms，每次打包向37、38、39三个广播频段同时发送数据。当发送长包时，每个分包数据按上述时序发送完毕后，即发送下一个分包数据。

  当发送长包时，每个分包数据按上述时序发送200ms，当第一个分包数据发送完后马上发送第二个分包数据，按次序发送分包数据后，根据需要可等待若干毫秒后进行重传。重传间隔不能小于400ms，建议按照重传间隔=（200ms+100ms*TTL+200ms*分包数）来计算。发送数据TTL建议不大于5。

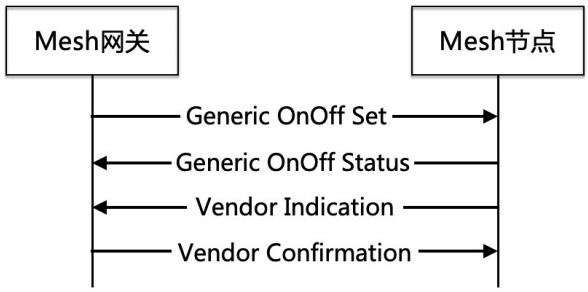


设备状态上报

- 设备在属性发生变化时，需要向Mesh网关的订阅地址（0xF000）发送含状态变化后相应属性值的Vendor Message Attribute Indication消息或者Vendor Message Attribute Status消息。
  - 当待上报的属性值很重要（如安防报警消息），需要收到Mesh网关的回复才停止发送，可以发送Vendor Message Attribute Indication消息。
  - 当设备需要周期上报属性值时，一般发送Vendor Message Attribute Status消息，而非Vendor Message Attribute Indication消息。
  - 如部分上报消息对业务逻辑影响不大（如设备固件版本号等），建议发送Vendor Message Attribute Status消息。
- Vendor Message Attribute Indication消息发包规则

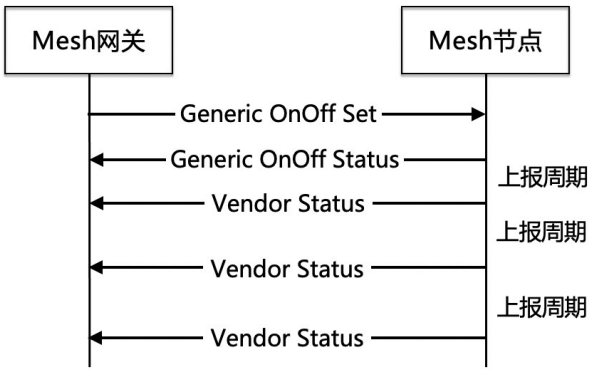
  发送Indication消息，之后打开scan，如果在重传间隔内（参考Mesh数据发送和接收中关于重传间隔的约定）收到Mesh网关回复的Vendor Message Attribute Confirmation消息，则停止发送，否则可以重发若干次直到收到Confirmation消息。
- Vendor Message Attribute Status消息发包规则
  - 注意减少不必要的消息发送，对业务逻辑影响不大的消息，控制重发次数。
  - 如果设备需要周期上报属性值，注意周期需要在几十秒以上的量级，最好是发送周期可以动态调整。
- 无周期属性上报的设备的属性同步

  当设备接收到下行的控制指令时，首先按照协议规定回复对应的状态消息；如果此时设备状态发生了变化，则通过Indication再次上报设备状态。具体交互流程可参照下图。



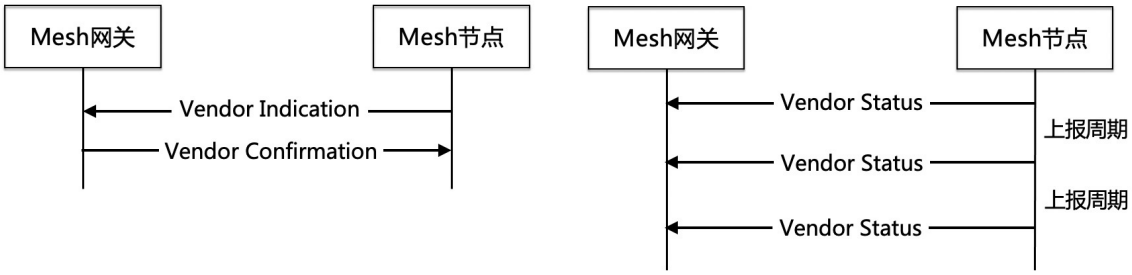
- 有周期属性上报的设备的属性同步

  当设备接收到下行的控制指令时，首先按照协议规定回复对应的状态消息；如果此时设备状态发生了变化，则等待下一个上报周期到时再上报设备状态。具体交互流程可参照下图。



- 设备主动上报场景

  当设备状态因为其他原因（用户直接操作设备、设备达到一定条件等其他因素后）发生改变时，设备需要通过发送Vendor Message Attribute Indication消息或者Vendor Message Attribute Status消息，主动上报设备当前的状态（仅上报改变的具体某一状态，不需要上报设备所有属性）。当某些设备需要周期性持续上报状态时（比如温湿度传感器，缺水状态上报，材料短缺上报等），此类消息上报的周期需要大于30秒（上报内容根据业务场景由各设备自行决定，原则上上报的数据越少越好）；如上报频率过高，可能会导致该产品上行链路被限制。



设备seq规则

- 蓝牙Mesh设备需要区分不同单播地址的seq，建议支持至少10个unicast address的seq存储，各地址的seq在设备上电期间要求保持连续更新。
- 多element设备重新配网，建议保存配网前各地址的seq，并在重新配网后恢复使用。

TID规范

当Mesh消息中包含TID字段（如Generic OnOff Set指令、Vendor Message Attr Set指令等）时，需要遵守以下规则。

- 所有有关联的业务使用相同的TID，比如设备接收到设置或查询指令时，设备回复到状态消息的TID（如果有）必须与下行到设置或查询指令中包含到TID一致。
- 设备主动上行使用的TID范围为128~191（0x80~0xBF）。
- 当Mesh设备收到一条发送给本设备（或本设备属于目标组）时，记录该TID；如果设备在在6秒内收到相同TID的消息，忽略后收到的消息（需要回复对应的状态消息，但业务层面不处理该消息）；被记录的TID在6秒后丢弃。
- 当消息当目标设备不是本设备（或本设备不属于目标组）时，忽略该消息，并且不记录TID。

组播地址

- 蓝牙Mesh设备每个element要求至少支持8个组播地址。当蓝牙Mesh设备收到增加组播地址订阅配置消息（Config Model Subscription Add Message）时发现该element存储的组播地址已满时，需按照Mesh Profile Specification丢弃新增加的组播地址，并在组播地址订阅配置状态消息（Config Model Subscription Status Message）中Status Code字段返回错误码0x05（Insufficient Resources）。
- 设备默认品类组播地址参考设备组播地址中的定义，设备需要在Provision完成配置阶段将设备上所有element的所有model与相应的组播地址进行绑定。
- 用户在配置设备位置、别名时，生活物联网平台会修改设备的组播地址订阅；在该过程中，生活物联网平台会对设备上某个element的其中一个model进行增加或删除组播地址订阅配置（Config Model Subscription Add/Config Model Subscription Delete），设备需要将接收到的组播地址的配置（增加或删除）同步至该element上所有的model。

心跳功能

- 蓝牙Mesh设备要求支持heartbeat功能，具体配置消息参考Mesh Profile Specification。
- 对于支持心跳功能的设备，将会通过给设备主elemet配置心跳设置参数进行设置。要求设备能严格按照所配置的参数发出对应的心跳包，发包规则参考Mesh数据发送和接收。
- 当心跳参数中对应的心跳包发送次数为无限次时，设备需要保存该心跳参数，在设备断电或重启后，重新加载该心跳参数。

通用开关服务模型

支持开关功能（powerstate属性）的设备都必须支持通用开关服务模型（Generic On Off Server Model）。

- 通用开关服务模型详细的消息格式定义请参考Mesh Model Specification。
- 设备需要实现对获取开关状态指令（Generic OnOff Get，Opcode 0x8201）的处理与响应，回复开关状态消息（Generic OnOff Status，Opcode 0x8204）。在回复开关状态消息时，只填充必选字段当前开关状态，不填充可选字段目标开关状态与剩余时间（作为生活物联网平台Mesh规范预留字段）。

字段	字段长度	备注
Opcode	2	0x82 0x04
当前开关状态（必选）	1	设备当前开关状态
目标开关状态（不填充）	1	设备目标开关状态
剩余时间（不填充）	1	设备切换到目标状态还剩余的时间

IV Index Update

为保证生活物联网平台Mesh设备能够长时间稳定运行，对于的IV index update过程中做了如下要求。

- 模组在seq number达到0xB0E500的时候开始进行IV Index Update流程。
- 由于很多低功耗设备会一直处于睡眠或断电状态，无法进行计时，所以我们要求忽略协议里规定的入网96小时之后才能进行IV index update和Recovery的时间限制。
- 要求如果设备在返回网络后收到第一个secure network beacon的IV index等于当前设备的IV index+1且IV Update Flag为0，应选择直接进行IV Index update，而不应忽略Secure Network Beacon。
- 对于需要长时间休眠的低功耗设备，为了防止无法接收到IV更新信息，要求设备至少每48小时唤醒一次，且进行至少15秒及以上的连续扫描。

设备上电与重启规范

- 蓝牙Mesh设备上电后需要延迟随机0~10秒后发送Vendor Message Attribute Indication消息或者Vendor Message Attribute Status消息，上报消息包括该设备所有支持的可上报的属性（必选）和上电事件（可选），注意需要把设备属性和上电消息分开上报，先上报设备属性，再上报上电事件。设备上电事件示例如下。

0xD4	0xA8	0x01	0x80	0x09	0xF0	0x03
Opcode			TID	Attr Name		Event
0xD401A8			0x80	事件上报		设备上电

- 蓝牙Mesh设备重启规范参照设备上电规范，将设备重启视作设备上电。

设备复位规范

在设备复位后，再次进行配网时，要求设备在Provision完成配置阶段按照各产品规范中定义的默认配置重新配置组播。当设备进行复位时，设备根据各产品规范决定是否删除设备应用配置信息（比如风扇的开关状态，冰箱设置的各区域温度等应用层面的配置）。当设备进行复位时，如果存在GATT链接，要求断开所有GATT连接。

- 软件复位（解除配网）

  蓝牙Mesh设备需要支持通过Config Node Reset的软复位的方法来移除节点，当设备进行软件复位的时候，移除设备上保存的配网信息，设备进入未配网广播状态。同时根据各产品特性决定是否删除设备配置信息。如果在软件复位过程中需要重启设备，则要求此时尽可能不影响设备正常工作。

- 硬件复位（恢复出厂设置）

  蓝牙Mesh设备通过硬复位恢复模块出厂设置时，需要移除设备上保存的配网信息，设备进入未配网广播状态。蓝牙Mesh设备进行硬件复位时，首先需要通过Vendor Message Attribute Indication消息上报硬件复位事件，当完成上报后（如上报消息未收到对应的回复，要求上报时间至少持续3秒）再进行硬件复位动作（删除配网信息、进入未配网广播状态等动作）。

0xD4	0xA8	0x01	0x80	0x09	0xF0	0x23
Opcode			TID	Attr Name		Event
0xD401A8			0x80	事件上报		硬件复位

OTA升级

模组需要根据自身产品形态支持通过BLE OTA升级。具体请参看蓝牙BLE OTA规范。

精灵低功耗（GLP）

如果模组要应用于下行数据接收的低延时低功耗设备（例如单火开关，电池供电的窗帘电机）时，可以采用精灵低功耗（Genie Low Power）方案，在开放平台上创建产品时在属性里选择精灵低功耗，这样精灵在给这个设备发送数据的时候，会在1.2s的时间内持续不断地发送数据。所以设备只需要每1.2s醒来60ms，可以在低功耗的情况下能及时接收到精灵下发的数据。

相比LPN方案需要低功耗节点与Friend节点需要建立连接，精灵低功耗的方案无需建立连接，并且休眠和激活的占空比是固定的，可以精确的计算出设备的待机功耗。

6.5.2. 蓝牙Mesh设备扩展协议

本文为介绍为智能家居设备制定的蓝牙Mesh扩展消息定义，便于更多的智能家居设备通过蓝牙Mesh技术来接入生活物联网平台。自有品牌项目与天猫精灵生态项目均遵循本协议。

背景信息

蓝牙技术联盟（Bluetooth SIG）定义的SIG Mesh的模型目前尚未覆盖所有的智能家居设备，我们采用厂商自定义模型（Vendor Model）来实现智能家居设备的控制和状态上报。在天猫精灵生态项目中，Mesh网关包括天猫精灵音箱与天猫精灵App。在自有品牌项目中，Mesh网关包括自有品牌项目Mesh网关产品与云智能App。注意以下部分消息仅在天猫精灵生态项目中支持。

Vendor Model

SIG定义Vendor Model格式为4字节（其中2字节的Company ID和2字节的Vendor-assigned Model ID），其中Alibaba的Company ID为0x01A8，如下表所示。

字段	字节数	说明
16-bit Company Identifier	2	0x01A8
16-bit vendor-assigned model Identifier	2	-

下表为两个VendorModel ID，用于消息扩展用。

Model Name	SIG Model ID
Vendor Model Server	0x01A80000
Vendor Model Client	0x01A80001

> **说明** Mesh设备作为Vendor Model Server，Mesh网关作为Vendor Model Client。

扩展消息操作码

在蓝牙Mesh协议中规定，操作码分为3种：1字节Opcode、2字节Opcode、3字节Opcode。

- 1字节Opcode的最高位为0，所以1字节Opcode的取值范围从0x00-0x7e（其中0x7f用于将来扩展用）。
- 2字节Opcode第一字节的最高位为1，第2位为0，所以2字节OpCode的取值范围从0x8000到0xbfff，都为SIG分配。

- 3字节Opcode第一字节的最高两位均为1，后两字节z是由SIG分配的Company ID，Company ID使用小端优先的方式传输，阿里巴巴使用的Company ID为0x01a8。6 bit的x可由厂商定义64个Opcodes。

Opcode Format	说明
0b0xxxxxxx（排除0b01111111）	1字节Opcode。
0b01111111	保留，用于将来扩展。
0b10xxxxxx xxxxxxxx	2字节Opcode。
0b11xxxxxx zzzzzzzz zzzzzzzz	3字节Opcode。

操作码定义

智能生活平台蓝牙Mesh扩展消息Opcode定义如下表所示。

Vendor Message Name	Opcode
Vendor Message Attribute Get	0xD001A8
Vendor Message Attribute Set	0xD101A8
Vendor Message Attribute Set Unacknowledged	0xD201A8
Vendor Message Attribute Status	0xD301A8
Vendor Message Attribute Indication	0xD401A8
Vendor Message Attribute Confirmation	0xD501A8
Vendor Message Attribute Indication To Tmall Genie	0xDE01A8
Vendor Message Attribute Confirmation From Tmall Genie	0xDF01A8
Vendor Message Transparent msg	0xCF01A8
Vendor Message Transparent Indication	0xCE01A8
Vendor Message Transparent ACK	0xCD01A8

Vendor Model属性消息结构

Vendor message里的数据都使用小端优先方式传输。

- Vendor Message Attribute Get

  该消息用于Vendor Model Client获取Vendor Model Server的一个或多个属性值，消息格式如下。

字段	字节数	说明
Opcode	3	0xD001A8
TID	1	Transaction Identifier，每条新消息递增

字段	字节数	说明
Attribute Type	2	读取的Attribute类型

Attribute Type最多可有15个。当Vendor Model Server收到Attribute Get消息后，必须向Vendor Model Client回复Attribute Status。如Vendor Model Client在下发该命令之后未收到Vendor Model Server返回的Attribute Status，可以再次下发该指令。

- Vendor Message Attribute Set

  该消息用于Vendor Model Client设置Vendor Model Server的一个或多个属性值，消息格式如下。

字段	字节数	说明
Opcode	3	0xD101A8
TID	1	Transaction Identifier，每条新消息递增
Attribute Type	2	设置的Attribute类型
Attribute Parameter	N	设置的Attribute参数

  Attribute Type和Attribute Parameter最多可有15个。当Vendor Model Server收到Attribute Set消息后，必须向Vendor Model Client回复Attribute Status。如Vendor Model Client在下发该命令之后未收到Vendor Model Server返回的Attribute Status，可以再次下发该指令。

- Vendor Message Attribute Set Unacknowledged

  该消息用于Vendor model Client设置Vendor Model Server的一个或多个属性值，消息格式如下。

字段	字节数	说明
Opcode	3	0xD201A8
TID	1	Transaction Identifier，每条新消息递增
Attribute Type	2	设置的Attribute类型
Attribute Parameter	N	设置的Attribute参数

  Attribute Type和Attribute Parameter最多可有15个。当Vendor Model Server收到Attribute Set Unacknowledged消息后，不需要向Vendor Model Client发送Attribute Status消息。

- Vendor Message Attribute Status

  该消息用于Vendor Model Server回复Attribute Get和Attribute Set命令或上报设备状态信息给Vendor Model Client，消息格式如下。

字段	字节数	说明
Opcode	3	0xD301A8
TID	1	Transaction Identifier，每条新消息递增
Attribute Type	2	上报的Attribute类型

字段	字节数	说明
Attribute Parameter	N	上报的Attribute参数

Vendor Model Client收到Attribute Status后，不需要回复消息给Vendor Model Server。

- Vendor Message Attribute Indication

  该消息用于Vendor Model Server发送属性给Vendor Model Client，消息格式如下。

字段	字节数	说明
Opcode	3	0xD401A8
TID	1	Transaction Identifier，每条新消息递增，回复控制命令的TID为下发消息的TID；设备状态主动改变上报的TID从128到191循环。
Attribute Type	2	上报的Attribute类型
Attribute Parameter	N	上报的Attribute参数

Attribute Type和Attribute Parameter最多可有15个。当Vendor Model Client收到Attribute Indication消息后，必须向Vendor Model Server回复Attribute Confirmation。如Vendor Model Server在发出该命令之后未收到Vendor Model Client回复的confirmation，可以再次发送该指令。

- Vendor Message Attribute Confirmation

  该消息用于Vendor Model Client回复给Vendor Model Server，用于表示已收到Vendor Model Server发出的Indication，消息格式如下。

字段	字节数	说明
Opcode	3	0xD501A8
TID	1	Transaction Identifier，收到的Indication消息的TID

Vendor Model Server收到Attribute Confirmation后，不需要回复消息给Vendor Model Client。

- Vendor Message Attribute Indication To TmallGenie

  该消息用于天猫精灵生态项目Vendor Model Server发送属性给Vendor Model Client，该消息由天猫精灵音箱直接处理，消息格式如下。

字段	字节数	说明
Opcode	3	0xDE01A8
TID	1	Transaction Identifier，每条新消息递增
Attribute Type	2	上报的Attribute类型
Attribute Parameter	N	上报的Attribute参数

Attribute Type和Attribute Parameter最多可有15个。当Vendor Model Client收到Attribute Indication消息后，必须向Vendor Model Server回复Attribute Confirmation。如Vendor Model Server在发出该命令之后未收到Vendor Model Client回复的Confirmation，可以再次发送该指令。

- Vendor Message Attribute Confirmation From TmallGenie

  该消息用于天猫精灵生态项目Vendor Model Client回复给Vendor Model Server，用于表示天猫精灵音箱已收到Vendor Model Server发出的Indication，消息格式如下。

字段	字节数	说明
Opcode	3	0xDF01A8
TID	1	Transaction Identifier，每条新消息递增
Attribute Type	2	设置的Attribute类型
Attribute Parameter	N	设置的Attribute参数

Attribute Type和Attribute Parameter的个数从0到15。Vendor Model Server收到Attribute Confirmation后，不需要回复消息给Vendor Model Client。

> **说明** TID：对于TID相同的消息，需按照协议回复对应的消息，但应用层是否处理则由产品自身特性决定。

Vendor Model透传消息结构

Vendor message里的数据都使用小端优先方式传输。

- Vendor Message Transparent Message

  该消息用于Mesh设备与天猫精灵音箱与天猫精灵App之间透传数据，Payload数据格式由各厂家自己实现。

字段	字节数	说明
Opcode	3	0xCF01A8
TID	1	Transaction Identifier，每条新消息递增，回复控制命令的TID为下发消息的TID；设备状态主动改变上报的TID从0到255循环。
Payload	N	由厂商自定义

- Vendor Message Transparent Indication

  该消息用于天猫精灵生态项目中Vendor Model Server发送属性给Vendor Model Client，消息格式如下。

字段	字节数	说明
Opcode	3	0xCE01A8
TID	1	Transaction Identifier，每条新消息递增，回复控制命令的TID为下发消息的TID；设备状态主动改变上报的TID从128到191循环。

字段	字节数	说明
Payload	N	由厂商自定义

- Vendor Message Transparent ACK

  该消息用于天猫精灵生态项目中Vendor Model Client回复给Vendor Model Server，用于表示已收到Vendor Model Server发出的Transparent Indication，消息格式如下。

字段	字节数	说明
Opcode	3	0xCD01A8
TID	1	Transaction Identifier，收到的Indication消息的TID。

> **说明** TID：对于TID相同的消息，需按照协议回复对应的消息，但应用层是否处理则由产品自身特性决定。

Vendor Message 示例数据

- Vendor Model Client设置目标温度

  本示例为Vendor Model Client设置目标温度为22摄氏度。

  - Vendor Model Client下发设置目标温度的Attribute Set命令。

0xD1	0xA8	0x01	0x01	0x0C	0x01	0x4B	0x73
Opcode			TID	Attribute Type		Attribute Value	
0xD101A8			01	目标温度（0x010C）		目标温度值：22摄氏度（0x734B = 295.15K）	

  - Vendor Model Server设置目标温度成功时返回的Attribute Status如下。

0xD3	0xA8	0x01	0x01	0x0C	0x01	0x4B	0x73
Opcode			TID	Attribute Type		Attribute Value	
0xD301A8			01	目标温度（0x010C）		目标温度值：22摄氏度（0x734B = 295.15K）	

  - Vendor Model Server设置目标温度失败时返回的Attribute Status如下。

0xD3	0xA8	0x01	0x01	0x00	0x00	0x0C	0x01	0x80
Opcode			TID	Error Code Type		Attribute Type		Error Code
0xD301A8			01	0x0000		目标温度（0x010C）		状态码：设备未准备好（0x80）

- Vendor Model Client读取数据

本示例为Vendor Model Client读取前后位置，当前温度，当前湿度。

- Vendor Model Client发送Attribute Get读取前后位置、当前温度、当前湿度属性的值如下。

0xD0	0xA8	0x01	0x01	0x10	0x01	0x0D	0x01	0x0F	0x01
Opcode			TID	Attribute Type		Attribute Type		Attribute Type	
0xD001A8			01	前后位置（0x0110）		当前温度（0x010D）		当前湿度（0x010F）	

- Vendor Model Server读取三个属性成功时返回的Attribute Status如下。

D3	A8	01	01	10	01	32	0D	01	4B	73	0F	01	2D	00
Opcode			TID	Attribute Type		Attribute Value	Attribute Type		Attribute Value		Attribute Type		Attribute Value	
0xD301A8			01	前后位置（0x0110）		前后位置：50（0x32）	当前温度（0x010D）		当前温度：22摄氏度（0x734B=295.15K）		当前湿度（0x010F）		当前湿度：45%（0x002D）	

- Vendor Model Server读取温度失败时返回的Attribute Status如下。

D3	A8	01	01	10	01	32	00	00	0D	01	81	0F	01	2D	00
Opcode			TID	Attribute Type		Attr. Value	Error Code Type		Attribute Type		Error Code	Attribute Type		Attribute Value	
0xD301A8			01	前后位置		前后位置：50（0x32）	0x0000		当前温度（0x010D）		错误码：不支持的属性（0x81）	当前湿度		当前湿度值：45%（0x002D）	

- Vendor Model Server上报温度

本示例为Vendor Model Server发送温度给Vendor Model Client。

- Vendor Model Server发送Attribute Indication上报温度如下。

0xD4	0xA8	0x01	0x80	0x0D	0x01	0x4B	0x73
Opcode			TID	Attribute Type		Attribute Value	
0xD401A8			80	当前温度（0x010D）		当前温度值：22摄氏度（0x734B = 295.15K）	

- Vendor Model Client收到Attribute Indication后发送Confirmation如下。

0xD5	0xA8	0x01	0x80
Opcode			TID
0xD501A8			80

- Vendor Model Server上报漏水故障

  本示例为Vendor Model Server上报漏水故障给Vendor Model Client。

  - Vendor Model Server发送Attribute Indication上报漏水故障如下。

0xD4	0xA8	0x01	0x80	0x09	0xF0	0x00	0x00	0x00	0xAA
Opcode			TID	Attribute Type		Attribute Value	Error Code Type		Error Code Value
0xD401A8			80	事件（0xF009）		故障事件（0x00）	错误码属性（0x0000）		错误码：漏水故障（0xAA）

  - Vendor Model Client收到Attribute Indication后发送Confirmation如下。

0xD5	0xA8	0x01	0x80
Opcode			TID
0xD501A8			80

6.5.3. 蓝牙mesh设备开发FAQ

本文介绍蓝牙mesh开发中的部分常见问题及排查方法。

问题：天猫精灵播报找不到设备

需要排查以下几点：

- 检查mesh设备是否有广播Unprovisioned device beacon。
- 检查Unprovisioned device beacon中的UUID是否符合UUID格式。
- 检查Unprovisioned device beacon的发送频率是否符合Mesh数据发送和接收。

问题：天猫精灵播报发现的设备类型与实际产品不符合

检查Unprovisioned device beacon中UUID里的Product ID是否正确，控制台页面与下载的Product ID是10进制，实际使用的时候需要转为16进制。详细格式请参见UUID格式。

问题：天猫精灵播报配网失败

需要排查以下几点：

- 检查设备证书，即Product ID、MAC（即Device Name）、Device Secret 是否烧录正确。
- 确认天猫精灵音箱的WiFi连网链路是否通畅。

- 参考配网流程确认整个过程是否全部正确：
  - 注意Provisioning Capabilities的计算都符合规范。
  - 检查随机数是否出现重复。
  - 在Provision完成配置阶段，设备是否回复了成功的状态消息给天猫精灵。
- 检查设备是否能收到天猫精灵音箱发出来的广播报文。

问题：天猫精灵App配网时找不到设备

需要排查以下几点：

- 检查设备证书，即Product ID、MAC（即Device Name）、Device Secret 是否烧录正确。
- 该产品是否在生活物联网平台控制台-人机交互中，完成了配网引导页面配置。
- 确认设备固件里是否打开了手机配网与Proxy连接的功能。如果基于天猫精灵Mesh SDK开发，检查宏CONFIG_BT_MESH_PB_GATT与CONFIG_BT_MESH_GATT_PROXY是否打开。

问题：配网成功率低

模组或者产品测试时如果遇到配网成功率不达标，或者部分手机型号App配网时经常配不上的问题，需要排查以下几点：

- 确认对模组做过频偏校正或者设置过正确的频偏参数。此问题可以通过Ellisys软件抓指定设备的空口包后查看RF Channel信息中的Initial Center Frequency参数来确认，检查实际频偏是否控制在-40KHz ~ +40KHz之内，频偏较大可能在部分手机型号上导致大概率失败。
- 检查是否有连续一段时间一直失败，并确认音箱的WiFi连网链路是否是通畅。
- 检查出现配网失败时，随机数是否出现重复。

问题：有天猫精灵音箱时设备本地硬件重置，没有提示设备解绑/删除

如果设备本地硬件重置后音箱没有语音提示设备解绑，同时App上设备列表里面没有删除设备，需要排查以下几点：

- 检查在该产品的功能定义里面是否有定义硬件复位事件（hardwareReset，0x0023），没有定义的话要添加。
- 检查固件里有上报硬件复位事件的逻辑，并且有若干次重传。请参考设备复位规范
- 查看音箱的WiFi网络连接没有问题。

问题：无天猫精灵音箱在线时设备本地硬件重置未成功上报

无天猫精灵音箱在线环境下，需要天猫精灵App将设备上报硬件重置事件转发给服务端。要确认设备可以与天猫精灵App通信，并且测试手机上行网络是通的，否则硬件重置事件无法上报。

问题：天猫精灵配网成功后，语音控制反馈无法控制或不支持的功能。

需要排查以下几点：

- 在生活物联网平台上该产品的功能定义页面检查是否添加了正确的属性。
- 尝试使用生活物联网平台设备调试页面-在线调试功能进行指令下发，不使用语音控制。

问题：天猫精灵配网成功后，语音控制之后反馈控制成功但设备无反应

需要排查以下几点：

- 在Provision完成配置阶段，设备是否根据自己的UUID版本做了对应配置。
- 如有接设备端日志，查看配网绑定是否都成功。

- 检查设备与天猫精灵之间的距离，移除遮挡物。
- 通过生活物联网平台设备调试页面-在线调试功能进行二进制指令下发，检查设备是否可以收到下发的指令。
- 确认账号下是否绑定了多个同类型设备，并且无法受控的设备是否配置了对应品类的组播地址。品类组播地址请参见设备组播地址。

问题：天猫精灵下发控制指令后，设备也正常按照指令执行，但向天猫精灵查询设备状态出错

需要排查以下几点：

- 检查设备端是否上报了正确的设备状态给天猫精灵。
- 检查设备端在状态改变后是否用Vendor Model的Indication上报状态给天猫精灵，且天猫精灵回复了Confirmation消息给设备端。

问题：在生活物联网平台设备调试页面-在线调试功能下发指令无效

需要排查以下几点：

- 检查调试选择的天猫精灵是否为当前测试使用的天猫精灵。
- 检查调试选择的设备是否为当前正在使用的设备。
- 检查下发指令是否有格式错误。

问题：天猫精灵配网成功后，语音控制反馈设备不在线

主要排查设备是否遵循设备上电与重启规范。

- 检查设备上电后是否将设备属性值通过Indication上报给天猫精灵并收到Confirmation消息。
- 检查设备上电后是否将设备上电事件通过Indication上报给天猫精灵并收到Confirmation消息。

问题：无天猫精灵音箱环境下，设备发送数据有概率无法到达App

如果基于TG7100B或者TG_B_7101模组开发，请注意升级Mesh SDK到V1.2.6版本，请参见获取SDK。

问题：无天猫精灵音箱环境下，App无法控制设备

排查以下几点：

- 注意设备固件里是否打开了使能Proxy功能的宏，如天猫精灵Mesh SDK中的对应宏为CONFIG_BT_MESH_GATT_PROXY。
- 注意创建该产品时有没有选择低功耗。对于低功耗设备，设备会进入睡眠。

问题：OTA界面一直在设备连接中

需要参考蓝牙BLE OTA规范并排查以下几点：

- 检查设备的蓝牙MAC地址与设备证书中的Device Name是否保持一致，开发调试时更换设备证书后，注意把MAC地址也相应更换。
- 请确认设备是否低功耗设备，设备需要先发广播包，才能建立GATT连接，如果设备在睡眠，要确认是否有唤醒机制并能正确唤醒发出广播包。
- OTA时手机App会与设备建立GATT连接，一般设备只允许建立一个GATT连接，需要排查是否设备已建立GATT连接且并未断开。
- 确认对模组做过频偏校正或者设置过正确的频偏参数，频偏过大可能影响部分手机建立GATT连接。

问题：OTA界面提示设备不允许OTA

需要参考蓝牙BLE OTA规范并排查以下几点：

- 检查是否在生活物联网平台推送的版本号比当前设备烧录的版本号大。
- 检查推送OTA的设备MAC地址是否与设备里实际烧录的MAC地址一致。
- 确认上传的bin文件是否正确，设备端会检查文件大小，如果超出flash OTA存储空间大小会拒绝升级。

问题：设备实际升级成功但是App提示失败

需要参考蓝牙BLE OTA规范并排查以下几点：

- 确认控制台推送是填的固件版本号与固件里实际编译的版本号是否一致，尤其要注意版本格式，详细定义请参见版本格式。
- 设备实际升级成功，交互流程中的第24步已经完成，需要重点排查第25～29步。

问题：传感器类产品需要支持GLP吗

需要区分该传感器是否要支持下行的设置指令。

- 如果不需要支持下行指令，传感器仅上报数据，可以不支持GLP，此时创建产品时不必选择“低功耗”而是选择“非低功耗”。
- 如果传感器需要支持下行的设置指令，需要根据产品交互、用户体验、功耗要求来综合评估是否支持GLP。可以按照GLP规范来实现，此时创建产品时选择“低功耗”；也可以通过按键唤醒设备，在一段时间内处于唤醒状态，然后此时下发设置指令，按键或者超时退出唤醒状态重新睡眠。

6.6. 蓝牙BLE设备开发指南

6.6.1. 蓝牙BLE基础规范

本文介绍天猫精灵BLE基础规范，包括蓝牙广播规范和蓝牙接入服务规范等。

名词解释

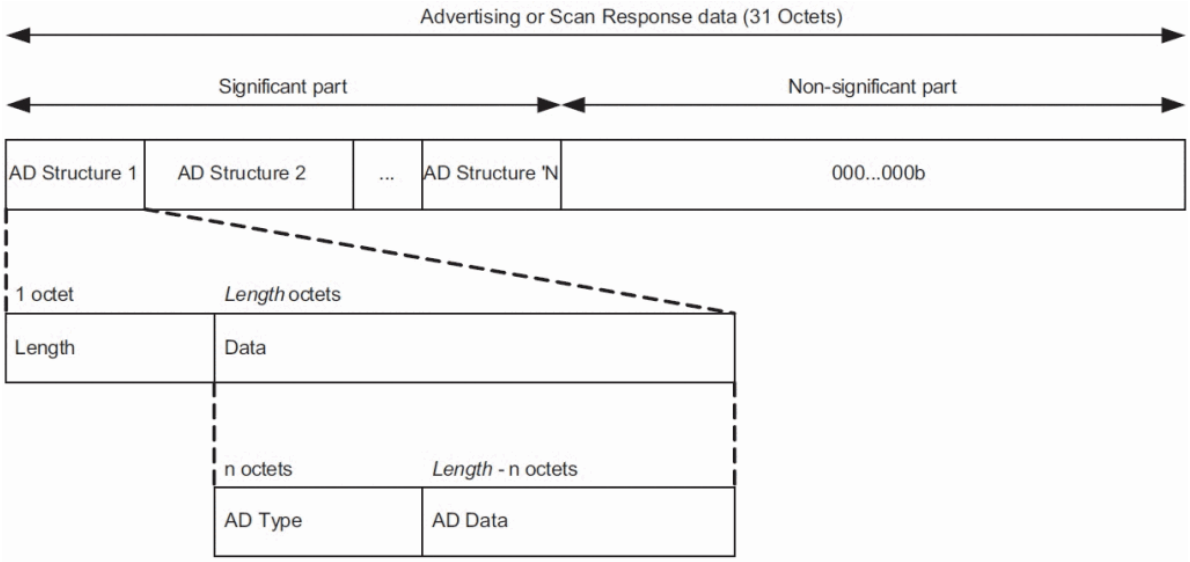
名词	描述
BLE	Bluetooth Low Energy 蓝牙低功耗技术。
CID	Company Identifier，公司标识符。阿里巴巴的标识符为0x01A8。
GATT	Generic Attribute Profile蓝牙通用属性协议。
AD	Advertisement广播。
VID	Protocol Version Code协议版本号。
PID	Product ID
FMSK	Function Mask 功能使能掩码。
MAC	Media AccessControl媒介访问地址。
AIS	Alibaba IoT Service阿里巴巴物联网服务。

名词	描述
GMA	全称Genie Mobile Accessory，中文移动精灵；是天猫精灵低功耗蓝牙、经典蓝牙相关业务能力的统称。

广播规范

- 蓝牙广播包

  蓝牙广播包格式遵循蓝牙4.0规范，由若干AD Structure组成（参见Bluetooth 4.2 Core Specification, Volume 3, Part C, Chapter 11），每一个ADStructure结构由Length、AD Type、AD Data组成。如下图所示。



- 广播数据包格式

  接入生活物联网平台的蓝牙设备的广播包必须包含阿里巴巴制定的厂商自定义格式（Manufacturer Specific Data，AD Type：0xFF）。阿里巴巴自定义广播格式总共由（6+n）字节组成，不同广播类型，对应不同的n值和Content，如下图所示。

字节序	名称	值	说明
0	Length	0x0F	Manufacturer Specific Data Length，本文档使用基础类型，故取值为基础类型的长度

字节序	名称	值	说明
1	Type	0xFF	Manufacturer Specific Data Type
2~3	CID	0x01A8	Company Identifiers，0x01A8为阿里巴巴公司编码
4	VID 和 Subtype	0xb5	◦ bit3～0： 阿里巴巴蓝牙规范版本号，当前版本号为5，值为0b0101 ◦ bit7～4： Subtype类型 ■ 0b1000：蓝牙基础类型，mesh设备AIS广播使用此类型 ■ 0b1001：蓝牙Beacon类型 ■ 0b1010：蓝牙语音类型 ■ 0b1011：蓝牙GATT类型，接入的BLE品类设备使用此类型
5	FMSK	0x03	SDK提供能力Function Mask，比如安全、OTA、蓝牙版本、安全广播等
6~n	Content	xx...xx	对应Subtype的内容数据

说明 Subtype只影响广播中6~n字节的数据格式。本文档采用蓝牙GATT类型进行描述。通常一类设备只需要实现一种广播类型。

FMSK用于表示设备具有的能力，各Bit位定义如下。

Bit序	功能说明
1~0	蓝牙版本，00：BLE4.0； 01：BLE4.2；10：BLE5.0；11：BLE5.0以上
2	0：不支持OTA；1：支持OTA
3	0：不进行安全认证； 1：进行安全认证，详细流程参考安全认证章节
4	0：一型一密；1：一机一密
5	配网标识，0：未配网；1：已配网
7~6	保留将来使用，全部填0

- GATT类型广播格式

GATT类型广播SubType对应0x0b，数据格式如下。

6～15字节定义如下。

字节序	名称	示例值	说明
6~9	PID	0x00ef1000	产品Product ID，4字节，由生活物联网平台颁发。
10~15	MAC	0xb0b448d07882	蓝牙设备MAC地址，6字节，唯一设备地址，由生活物联网平台颁发。

> **说明** 上述广播中的数据必须按照小端（Little-Endian）格式存储。

AIS服务规范

对于接入阿里巴巴平台的蓝牙设备，必须遵守阿里巴巴自定义的蓝牙服务AIS（Alibaba IoT Service）。

- AIS服务声明：

  AIS服务声明为Primary Service，Service UUID为0xFEB3。

- AIS Characteristics说明：

  AIS服务包含以下5个Characteristics。

Characteristics名称	Characteristics UUID	是否必选	属性	许可权限
Read Characteristics	0xFED4	是	Read	Read
Write Characteristics	0xFED5	是	Read或Write	Write
Indicate Characteristics	0xFED6	是	Read或Indicate	None
WriteWithNoRsp Characteristics	0xFED7	是	Read或Write with No Response	Write
Notify Characteristics	0xFED8	是	Read或Notify	None

数据传输规范

由于蓝牙4.0一次只能发送20字节的有效数据，当一包数据超过20字节时，蓝牙无法一次完成数据的传送，所以在发送长数据包时，需要发送时拆包，接收时组包。

- 单包最大数据长度

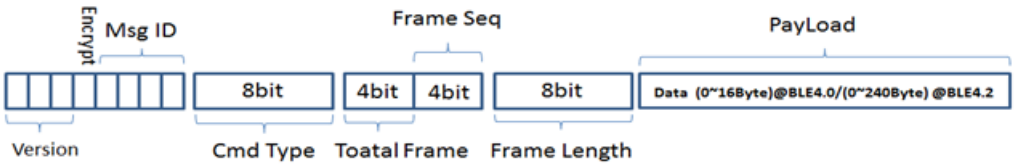
  不同的蓝牙版本下，蓝牙在应用层可以传输长度和本规范应用数据长度如下表所示。下文用N代表规范数据长度。

BLE版本	应用层数据长度	规范数据长度（N）
BLE4.0	20	16
BLE4.2	244	240
BLE5.0	244	240

- 数据格式
  - 每一包数据由Header，Payload组成。
  - Header长度四字节，主要包含信息指示、消息长度、序号、拆包总数等。
  - Payload 长度0~N字节。
  - 消息长度超过N字节时，分成多帧发送，接收到后按照Header描述重新组包。
  - Header占用四字节，Payload数据长度为0~N字节，整包数据长度在4~4+N字节。

Header	Payload
4字节	0~N字节

- 数据格式详细定义



  - Header Byte 0：数据加密标示、消息ID指示等，说明如下。

Bit序	说明	备注
0~3	消息ID，Msg ID	■ 每发送一条消息ID加1 ■ 如果消息有应答时，应答的消息ID需要和请求的消息ID匹配 ■ ID超过15，自动循环到1
4	数据加密指示	置0
5~7	版本信息	置0

- Header Byte 1：指令类型，蓝牙设备应用层和手机App交互的指令定义示例如下。

分类	指令	说明
设备主动上报	0x01	蓝牙设备主动上报的设备状态。
Request-Response模型	0x02	手机App发出请求指令，需要设备回复，与0x03对应。
Request-Response模型	0x03	蓝牙设备回复请求指令，与0x02对应。
Request-Response模型	0x04	蓝牙设备发出请求指令，需要手机App，与0x05对应。
Request-Response模型	0x05	手机App回复请求指令，与0x04对应。
异常上报	0x0F	指令异常通知，用于蓝牙设备通知手机App，设备接收到错误的指令或流程出错。
其他指令	-	为满足业务需求定义的其他指令，分为以下两种： ■ 建立连接的指令集，参见连接建立指令集 ■ 空中升级（OTA）定义了相关的指令集，参见OTA指令集

> **说明** 设备主动上报、Request-Response模型指令为通用指令，Payload格式不做限制，使用方可以自行定义。蓝牙设备接收到错误的指令时，直接抛弃，并通过指令0x0F通知手机App指令错误。

- Header Byte 2：消息帧数及帧序号，说明如下。

Bit序	说明	备注
0~3	帧序号	取值0~15，帧序号从0开始计数
4~7	总拆包帧数	■ 取值0~15，故实际的总拆包帧数等于（Bit4~Bit7）的值加1 ■ Payload为空时，Header中的Byte2、Byte3全为零

- Header Byte 3：帧数据长度。

  蓝牙4.0数据长度0~16，对于蓝牙4.2及以上版本长度是0~240。

- Payload：有效数据。

  Payload也使用小端（Little-Endian）格式传输。

- 消息ID（MsgID）：MsgID用于设备端和App端请求指令和回复指令做一一对应使用。由SDK内部维护。

  设备主动上报状态时，MsgID置0。App下发指令需要设备端应答时，设备端保存MsgID，设备的回复指令填充相同的MsgID。MsgID取值范围1~15。App每次下发需要设备端应答的指令时，MsgID加1，到15时，自动循环到1。空中升级（OTA）场景下，MsgID均置0。

- 数据长度

蓝牙数据拆包后，拆包数量不超过16个，蓝牙4.0每一包最长16字节，故总数据长度不超过256字节。蓝牙4.2以上，总数据长度不超过3840字节。空中升级（OTA）固件长度不受此限制。

- 数据的接收发送规范

  蓝牙设备的数据发送、接收按照串行的方式，在SDK中维护。一条消息（表示需要拆包的完整的数据）发送完成，才能发送下一条消息，或者接收下一条消息。

安全认证

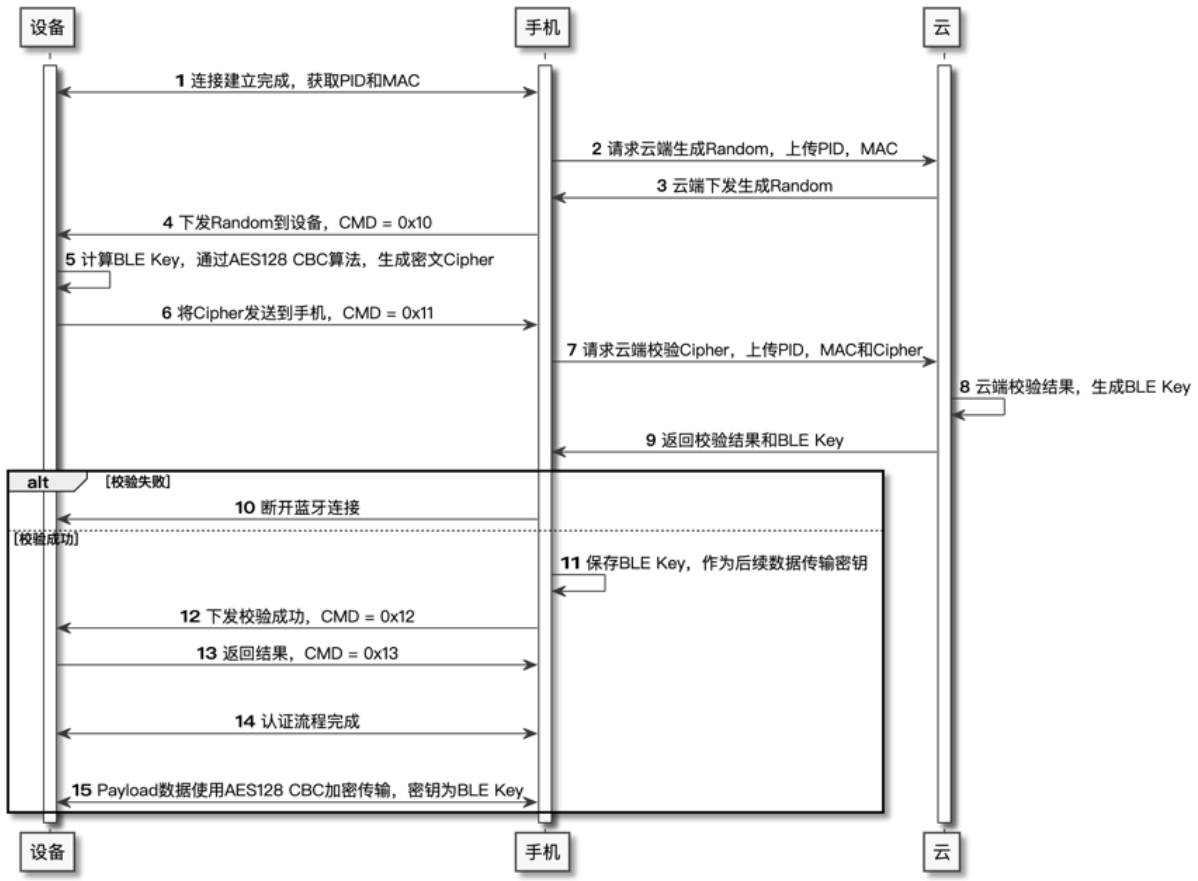
安全认证主要用于设备和手机互相校验身份，避免仿冒情况，应用于安全性要求较高的设备或场景，安全认证需要依赖云端能力。使用安全认证的设备，需要将广播的FMSK第三个Bit位置为0b1。手机在每次连接的时候，会进行安全认证流程。安全认证通过后，手机和设备的数据传输会通过密文传输。安全认证是可选实现，可以根据设备的安全性要求或实际业务场景选择使用。

算法和安全说明

- ProductID，MAC，Secret是云端分配的数据，预先烧录在设备中。为了保证安全，Secret不进行空中传播，也不会传递到手机端。
- 定义Random是16字节随机字符串，每次生成的Random必须不同。
- 定义 `BLE Key = SHA256(Random,PID,MAC,Secret)` 的。即：将Random、PID、MAC、Secret四个值的字符串数据用英文逗号连接，然后进行SHA256摘要计算，取前16字节。
- 定义 $Cipher = AES128_{BLE\ Key}(Random)$ 。即：以BLE Key作为密钥，加密Random生成密文Cipher。其中加密算法使用AES128 CBC，取前16字节。
- 安全认证通过后，数据传输使用密文形式， 加密算法使用AES128 CBC，密钥使用BLE Key。
- 在认证过程中，云端保证设备身份的不会变化，并且是属于有效用户的合法设备。
- 手机每次连接会请求生成新的BLE Key，每次断开连接后，设备和手机需要清除当前使用的BLE Key。
- BLE Key计算过程参数示例如下：

数据字段	数据格式与示例	计算使用的输入字符串
Random	随机字符串："drfiHgbsvomOieog"	"drfiHgbsvomOieog"
ProductID	十进制数值：168930，对应十六进制数值：0x293e2	"000293e2"
Mac Address	"AB:CD:F0:F1:F2:F3"（扫描到的蓝牙设备MAC地址（	"abcdf0f1f2f3"
Secret	字符串："atFY1tGDCxxxxx3PvBI5WXb"	"atFY1tGDCxxxxx3PvBI5WXb"
用英文逗号连接后的字符串	"drfiHgbsvomOieog,000293e2,abcdf0f1f2f3,atFY1tGDCxxxxx3PvBI5WXb"	

连接建立流程



连接建立指令集

- CMD 0x10

  手机发起开始认证流程，下发Random到设备。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x10	0x00	0x10	16字节Random数据

> **说明** Total Frame & Seq字段值为0x00表示共1帧数据且当前数据为第1帧。

- CMD 0x11

  设备认证流程，设备生成密文Cipher并发送到手机。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x11	0x00	0x10	16字节Cipher数据

- CMD 0x12

  手机下发校验结果到设备。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x12	0x00	0x01	0x00：成功，0x01：失败

- CMD 0x13

  设备向手机返回BLE Key处理结果。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x13	0x00	0x01	0x00：成功，0x01：失败

> **说明** 0x13指令Msg ID要求与0x12指令中数据一致。

- CMD 0x14

  手机通知设备配网成功。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x14	0x00	0x01	通知设备配网/解绑结果。0x00：解绑，0x01：配网

- CMD 0x15

  设备回复配网成功结果。

CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload说明（1字节）
0x15	0x00	0x01	配网/解绑成功ACK 0x01

> **说明** 0x15指令Msg ID要求与0x14指令中数据一致。

连接建立指令与Characteristic对应关系

指令类型	说明	传输使用的Characteristic	Characteristics UUID
0x0F	异常通知	Indicate Characteristics	0xFED6
0x10	下发Random	Write Characteristics	0xFED5
0x11	设备上报Cipher	Indicate Characteristics	0xFED6
0x12	下发校验结果	Write Characteristics	0xFED5

0x13	设备返回状态	Indicate Characteristics	0xFED6
0x14	设备解绑/配网	Write Characteristics	0xFED5
0x15	回复解绑/配网结果	Indicate Characteristics	0xFED6

参考资料

- Bluetooth4.2 Core Specification
- Bluetooth CoreSpecification Supplement v5

6.6.2. 蓝牙BLE业务流程与体脂秤示例

本文介绍天猫精灵音箱、App和BLE设备行为和具体协议规范。

背景信息

本规范基于蓝牙BLE基础规范。

基础规范使用

- 广播规范

  将Subtype置为0b1011。

  如上图所示，Manufacturer Specific Data在基本类型的定义上扩展出2个字节（16bit）的Ext字段，用来标识一些额外的信息。Ext字段各bit位定义如下。

Bit序	功能说明
0	是否有数据进行上报，0：没有数据；1：有数据上报
1 ~ 15	保留位数，后续启用，全部配置为0。

- 数据传输规范

  BLE数据传输使用基础规范定义的数据格式和规则。具体介绍，请参见数据传输规范。

- Payload业务规范

- Flag标识业务数据的特征，占用bit 7 ~ 6。

bit位	说明
bit 7	■ 消息为上行消息时表示云端还是天猫精灵音箱或App本地处理，0为云端处理，1为本地处理。 ■ 消息为下行消息时表示消息来自云端还是天猫精灵音箱或App本地，0为来自云端，1为来自本地。
bit 6	当前置0，后续可扩展。

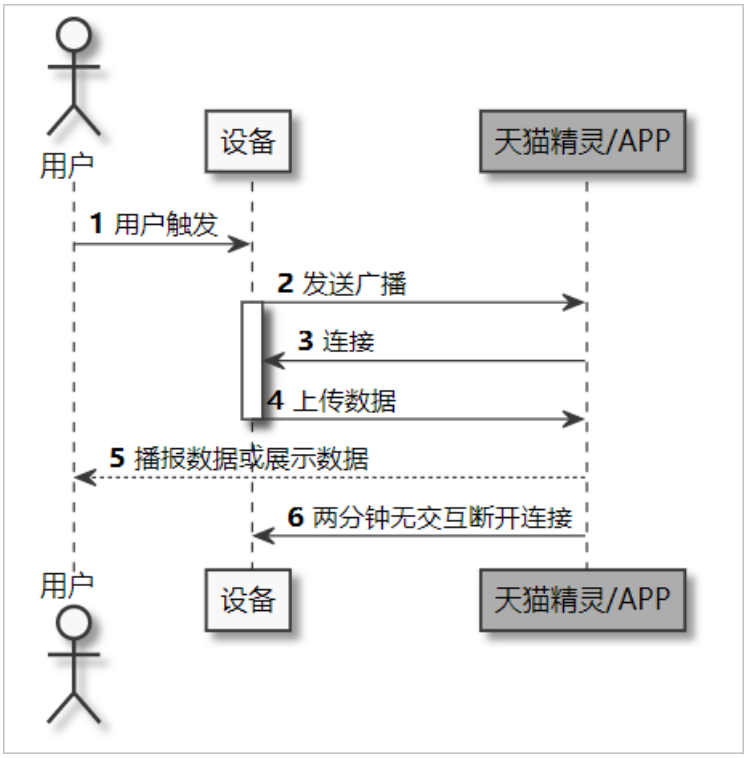
- DataType标识业务数据的类型，占用bit 5 ~ 0：默认取值0x01。适用于当前使用场景，后续可做扩展。

接入设备类型

接入的BLE产品必须符合标准的BLE GATT协议。由于GATT和Mesh协议不同，是基于连接的协议，也就是传输数据之前必须先进行连接，而天猫精灵和App同时连接的BLE 设备是有限的，所以和天猫精灵连接的BLE设备，只有在需要传输数据的时候才建立GATT的连接，数据传输完成之后，GATT连接就需要断开。由于这种差异性，根据产品业务不同，将接入的产品主要分为数据上报型设备和数据下发上报型设备。

数据上报型设备

此类设备以上传数据给天猫精灵或App为主要业务场景，天猫精灵或App侧没有主动下发命令的需求。体脂秤是典型的数据上报型设备，体脂秤都是主动将采集到的体脂信息主动上报到天猫精灵或App，天猫精灵或App在连接还没有建立的情况下，没有主动下发命令的场景。此类设备只在需要上报数据的时候，向外发送BLE 广播包，天猫精灵和App扫描到此广播包后就主动连接此设备，连接建立之后，设备主动上报数据到天猫精灵端或App端，在连接没有断开之前，天猫精灵端或App 端可以下发命令。当设备和天猫精灵在2分钟内没有数据传输的话，精灵需要主动断开和设备的连接。同时也允许设备为了功耗的考虑，在完成数据传输后立刻主动断开和天猫精灵的连接。

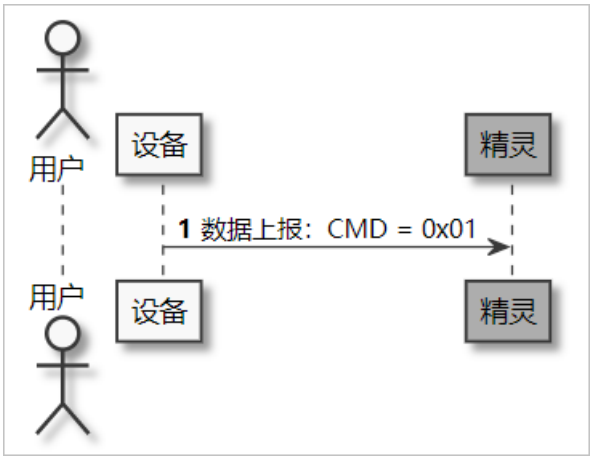


数据交互指令集

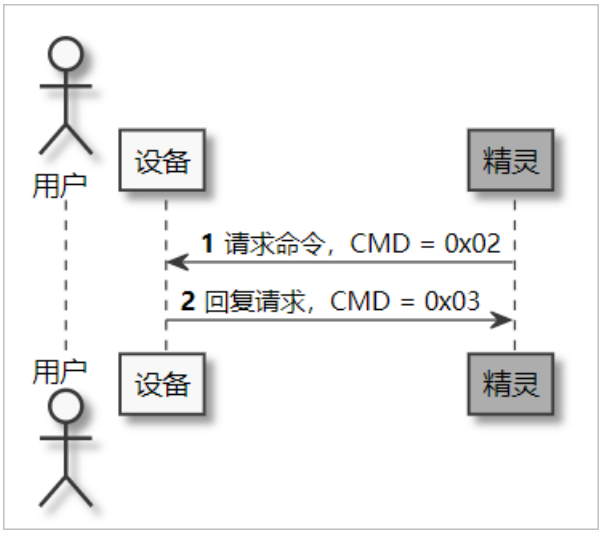
CMD对应数据格式Header中的CmdType字段。

- 设备主动上报

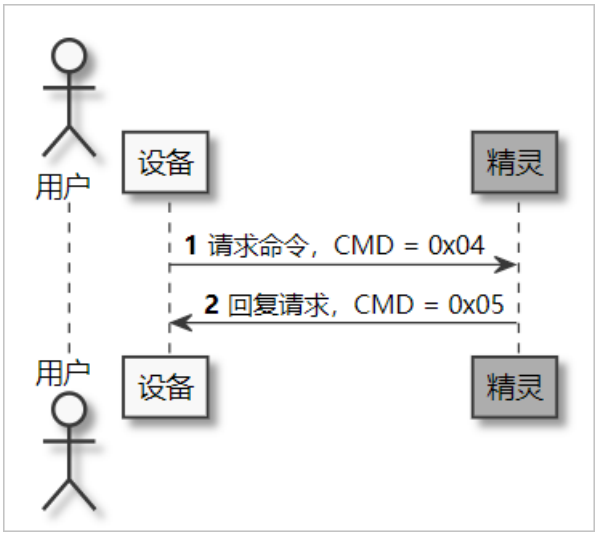
  CMD 0x01：蓝牙设备主动上报的设备状态。



- Request-Response模型（App或音箱发起）
  - CMD 0x02： 手机App或音箱发出请求指令，需要蓝牙设备回复，与CMD 0x03对应。
  - CMD 0x03： 蓝牙设备回复请求指令，与CMD 0x02对应。

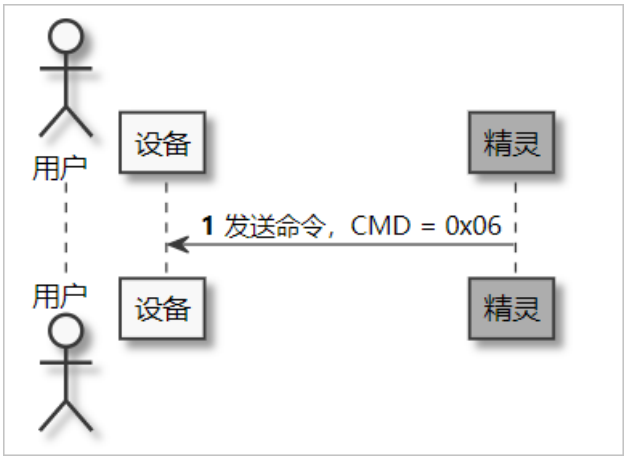


- Request-Response模型（设备发起）
  - CMD 0x04： 蓝牙设备发出请求指令，需要手机App或音箱回复，与CMD 0x05对应。
  - CMD 0x05： 手机App或音箱回复请求指令，与CMD 0x04对应。



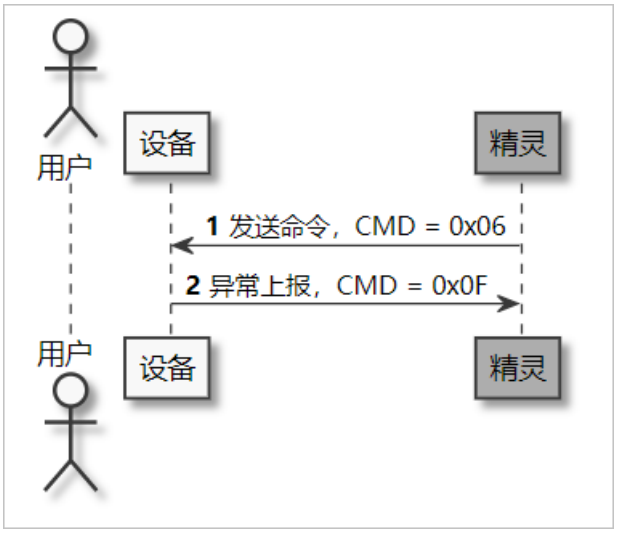
- 命令下发

  CMD 0x06：手机App或音箱发送指令，无需对方回复。



- 异常上报

  CMD 0x0F：指令异常通知，用于蓝牙设备通知手机App，设备接收到错误的指令或流程出错。



数据Data格式选项

根据BLE设备的不同，Data可以有两个选项。

- 厂商自定义格式

  如果厂商选用自定义私有格式，设备端的改动较小，但需到云端配置脚本数据转换。

- 使用阿里定义的格式

  如果选用阿里定义的格式，则需要设备端实现阿里的数据规范，将opcode缩减为1字节。更多介绍，请参见蓝牙Mesh设备扩展协议。缩减后的opcode如下表所示。

Message Name	Opcode
Vendor Message Attr Get	0xD0
Vendor Message Attr Set	0xD1
Vendor Message Attr Set Unacknowledged	0xD2
Vendor Message Attr Status	0xD3
Vendor Message Attr Indication	0xD4
Vendor Message Attr Confirmation	0xD5
Vendor Message Attr Indication To TmallGenie	0xDE
Vendor Message Attr Confirmation From TmallGenie	0xDF
Vendor Message Transparent msg（厂商自定义格式）	0xCF

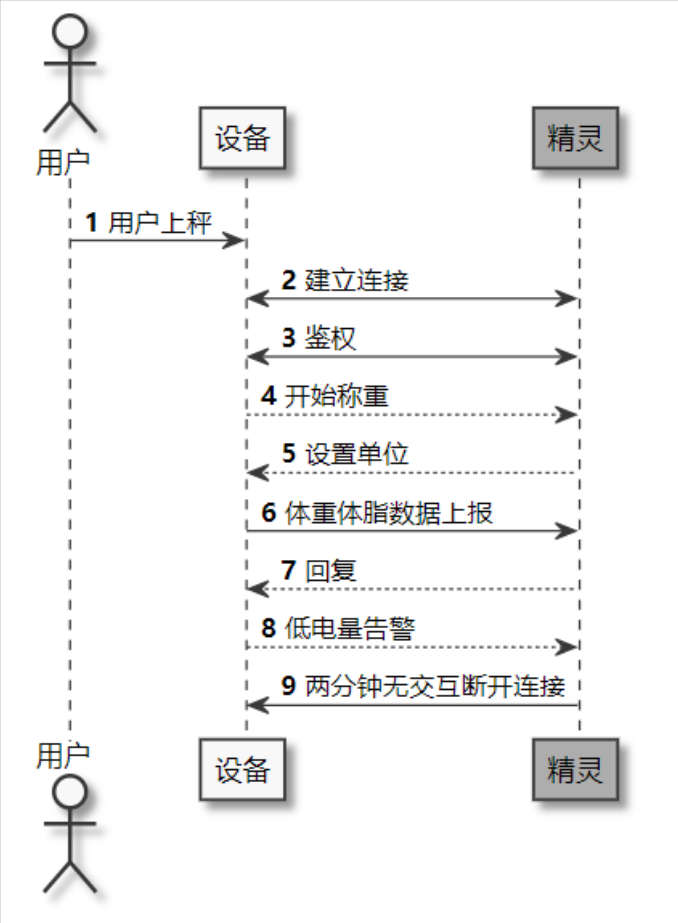
体脂秤属性示例

体脂秤属性定义示例如下。

属性名称	Vendor model Attr	Vendor Model Attr Parameter	备注
体重	0x0200	uint16 weight	精度0.01公斤
体脂	0x0201	uint32 impedance	欧姆（Ω），精度为0.1
称重时间	0x0103	uint8 unix_time[6]	unix时间戳
单位	0x010B	uint8 unit	• 0：公制（公斤，米，摄氏度） • 1：英制（磅，英尺，华氏度） • 2：中国单位（市斤，尺）
开始称重	0xF009	uint8 Event	0x1D： 开始测量
低电告警	0xF009	uint8 Event	0x01
电量	0x0104	uint8 power_percent	电池电量百分比
故障上报	0xF009	uint8 Event	设备发生的事件，详细定义，请参见设备事件表。如事件带参数，则后续跟参数。 例如0x82表示不支持操作；0x84表示设备状态错误；0x91表示传感器故障。

体脂秤场景交互

如下所示，其中开始称重和设置单位是可选的，当设备有低电量情况下，主动上报低电量告警。



体脂秤数据示例

- 开始称重
  - 该消息使用Vendor Message Attr Indication，消息格式如下。

字段	字节数	说明
Opcode	1	0xD4
TID	1	Transaction Identifier，每条新消息递增，回复控制命令的TID为下发消息的TID。
Attr Type	2	0xF009
Attr Parameter	1	0x1D

  - 天猫精灵或App使用Vendor Message Attr Confirmation回复，消息格式如下。

字段	字节数	说明
Opcode	1	0xD5
TID	1	Transaction Identifier，收到的Indication的TID。

- 设置单位

该消息使用Vendor Message Attr Set Unacknowledged，消息格式如下。

字段	字节数	说明
Opcode	1	0xD2
TID	1	Transaction Identifier，每条新消息递增。
Attr Type	2	0x010B
Attr Parameter	1	0x02：中国单位（市斤）

- 上报体重体脂数据（使用Vendor Message Attr Status）

  上报体重体脂数据可以使用Vendor Message Attr Status，此消息天猫精灵或者App无需回复，消息格式如下。

字段	字节数	说明
Opcode	1	0xD3
TID	1	Transaction Identifier，每条新消息递增。
Attr Type	2	0x0200
Attr Parameter	2	体重值
Attr Type	2	0x0201
Attr Parameter	2	体脂值

- 上报体重体脂数据（使用Vendor Message Attr Indication）
  - 上报体重体脂数据可以使用Vendor Message Attr Indication，消息格式如下。

字段	字节数	说明
Opcode	1	0xD4
TID	1	Transaction Identifier，每条新消息递增。
Attr Type	2	0x0200
Attr Parameter	2	体重值
Attr Type	2	0x0201
Attr Parameter	2	体脂值

- 天猫精灵或者App使用Vendor Message Attr Confirmation回复，消息格式如下。

字段	字节数	说明
Opcode	1	0xD5
TID	1	Transaction Identifier，收到的Indication的TID。

- 低电量告警

  该消息使用Vendor Message Attr Status，此消息天猫精灵或者App无需回复，消息格式如下。

字段	字节数	说明
Opcode	1	0xD3
TID	1	Transaction Identifier，每条新消息递增。
Attr Type	2	0xF009
Attr Parameter	1	0x01

6.6.3. 蓝牙BLE OTA规范

蓝牙设备通常需要空中升级（OTA）的能力进行固件更新，本规范定义了空中升级（OTA）的基本流程和指令集。

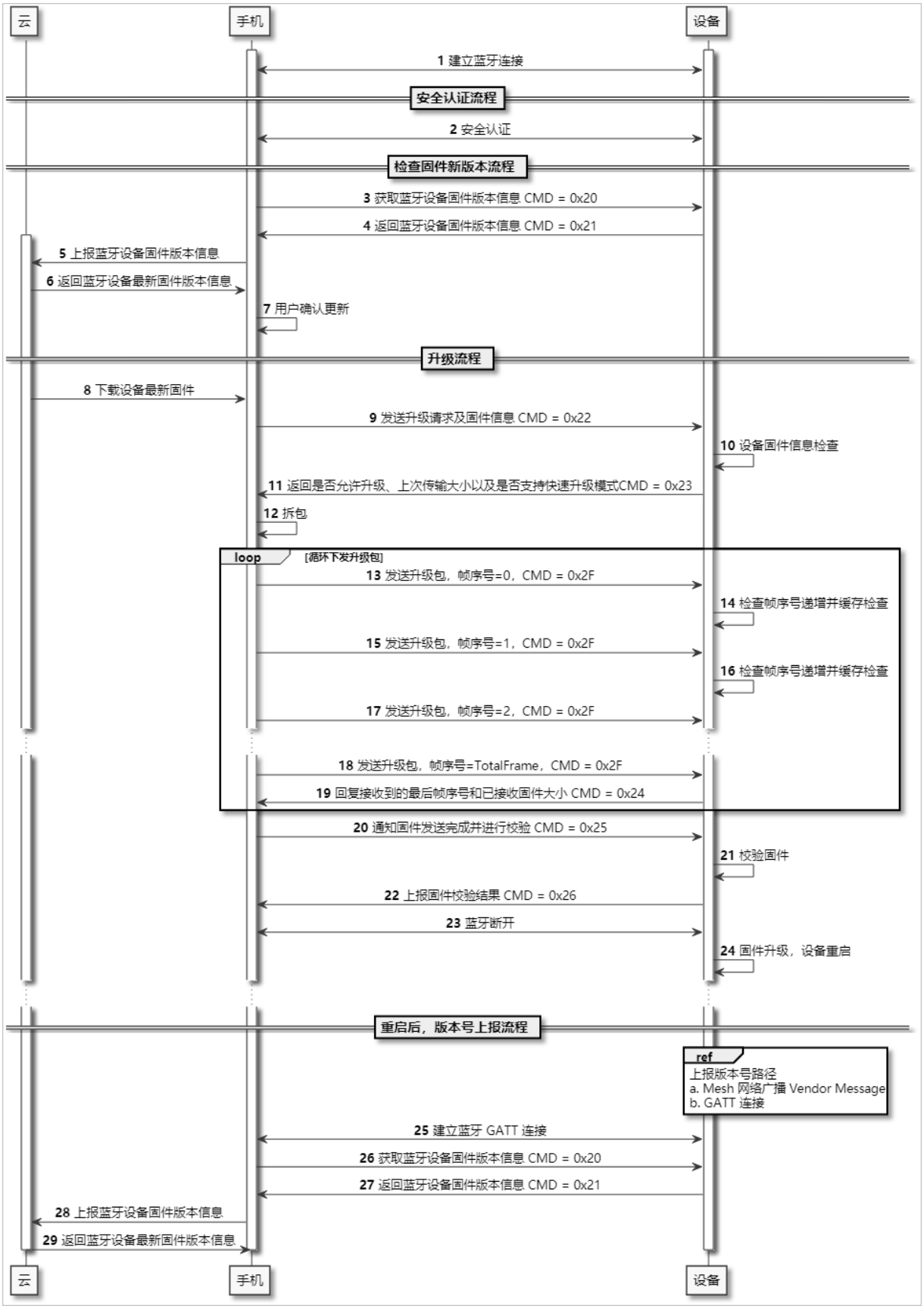
背景信息

本规范基于蓝牙BLE基础规范。

交互流程

为了保证OTA的安全性，在进行设备OTA之前，必须完成安全认证流程。若认证失败则不允许进行OTA。安全认证详情，请参见蓝牙BLE基础规范。

手机与蓝牙设备OTA流程如下。


云
手机
设备
1 建立蓝牙连接
安全认证流程
2 安全认证
检查固件新版本流程
3 获取蓝牙设备固件版本信息 CMD = 0x20
4 返回蓝牙设备固件版本信息 CMD = 0x21
5 上报蓝牙设备固件版本信息
6 返回蓝牙设备最新固件版本信息
7 用户确认更新
升级流程
8 下载设备最新固件
9 发送升级请求及固件信息 CMD = 0x22
10 设备固件信息检查
11 返回是否允许升级、上次传输大小以及是否支持快速升级模式CMD = 0x23
12 拆包
loop
[循环下发升级包]
13 发送升级包，帧序号=0，CMD = 0x2F
14 检查帧序号递增并缓存检查
15 发送升级包，帧序号=1，CMD = 0x2F
16 检查帧序号递增并缓存检查
17 发送升级包，帧序号=2，CMD = 0x2F
18 发送升级包，帧序号=TotalFrame，CMD = 0x2F
19 回复接收到的最后帧序号和已接收固件大小 CMD = 0x24
20 通知固件发送完成并进行校验 CMD = 0x25
21 校验固件
22 上报固件校验结果 CMD = 0x26
23 蓝牙断开
24 固件升级，设备重启
重启后，版本号上报流程
ref
上报版本号路径
a. Mesh 网络广播 Vendor Message
b. GATT 连接
25 建立蓝牙 GATT 连接
26 获取蓝牙设备固件版本信息 CMD = 0x20
27 返回蓝牙设备固件版本信息 CMD = 0x21
28 上报蓝牙设备固件版本信息
29 返回蓝牙设备最新固件版本信息
云
手机
设备

基础规范使用

- 广播规范

  空中升级（OTA）是可选功能，如果蓝牙设备实现了此功能，需要在广播规范的FMSK字段中标示。

- 服务规范

  传输过程使用基础规范已定义的Service和Characteristics，采用指令类型区分。手机App发送固件时，采用WriteWithNoRsp Characteristics(0xFED7)，用于加快传输速度。同时蓝牙设备收到数据包后，采用Notify Characteristics（0xFED8）对App应答。接收固件过程中，发现数据序号错误的时候，蓝牙设备上报最后一次正确的序号。

- 数据传输规范

  数据传输使用基础规范定义的数据格式和规则。

版本格式

固件类型为一个1字节数字，用来区分同一个设备不同类型固件。版本格式为一个4字节数字，如：0x00010302，表示固件版本号为：1.3.2，版本号只允许递增，最大版本号为：99.99.99。版本格式规则如下。

字节序	说明	取值范围	示例
0	修订号	0x00~0x63	1.3.2，修订号为0x02
1	次版本号	0x00~0x63	1.3.2，次版本号为0x03
2	主版本号	0x00~0x63	1.3.2，主版本号为0x01
3	保留	无	无

升级规则

- 待升级的固件推送到服务端，并填写固件版本信息和升级原因。
- 当服务端确认推送的固件版本比设备端上报的固件版本新时，服务端推送升级通知到手机App。
- 手机App转发升级请求及固件版本信息（仅包含应用版本信息）和固件大小，CRC。
- 蓝牙设备检查固件的应用版本信息，当App下发的应用版本信息比设备运行的固件的应用版本新时，蓝牙设备进入升级模式，否则退出升级模式。
- 协议设计支持固件断点续传能力，参考“交互流程”第12步，蓝牙设备可以回复上次传输固件的中断的位置，手机App会从指定位置续传。断点续传非强制要求能力，可根据情况实现。
- 协议设计支持快速传输模式，即手机App会连续传输TotalFrame（0<TotalFrame<=16）个数据包而不需要对每个数据包等待蓝牙设备的回复，TotalFrame个数据包发送完成后，蓝牙设备才给予回复，然后手机App进行下一轮数据的传输。
- 除了0x2F指令（下发固件分包）不需要加密外，其余指令通过鉴权流程后均需要加密。

OTA指令集

- CMD 0x20：手机查询设备固件版本。

  Payload为1字节：固件类型为1字节，默认填0x00。设备端根据固件类型从 0x21 返回对应类型固件的版本号信息。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（1字节）

0x00	0x20	0x00	0x01	0x00

> 说明 Total Frame & Seq字段值为0x00表示共1帧数据且当前数据为第1帧。

- CMD 0x21： 设备上报设备固件版本。

  Payload为5字节：

  - 固件类型（1字节）：默认填0x00，遇到不支持的固件类型，固件类型上报 0xFF。
  - 固件版本号（4字节）：例如0x00000002。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（5字节）
0x00	0x21	0x00	0x05	0x00 0x00000002

> 说明 0x21指令Msg ID要求与0x20指令中数据一致。

- CMD 0x22：手机下发升级请求。

  Payload为12字节：

  - 固件类型（1字节）：默认填0x00
  - 固件版本号（4字节）：例如0x00000001
  - 固件大小（4字节）：例如 0x00123456
  - CRC16（2字节）：采用CRC16_CCITT
  - 升级标示符（1个字节）：0-全量升级，1-增量升级，2-静默升级。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（12字节）
0x00	0x22	0x00	0x0c	0x00 0x00000001 0x00123456 0x7890 0x00

- CMD 0x23：设备应答升级请求。

  Payload（6字节）：

  - 是否允许升级（1字节）： 0-不可以升级，1-允许升级。
  - 上次传输字节数（4字节）
  - 允许一次循环可传输的包个数（1字节）：取值范围0x00~0x0F，表示1～16包。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（6字节）
0x00	0x23	0x00	0x06	0x01 0x00000000 0x0F

> 说明 0x23指令Msg ID要求与0x22指令中数据一致；如设备支持断点续传，则上次传输的字节上报已经接收到的固件长度。

- CMD 0x24： 设备上报当前已接收的帧序号和接收的数据长度。

  Payload（5字节）

  - 接收的帧数（1字节）：totalFrame 4bit + frameSeq 4bit
  - 数据长度（4字节）：例如0x00000200。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（5字节）
0x00	0x24	0x00	0x05	0x20 0x00000200

  当设备发现有丢帧时，设备端上报最后一次收到正确的帧序和数据长度。手机端重新从丢失的帧开始发送，并且发送完本次循环的剩余帧。

- CMD 0x25：手机通知设备固件下发结束，可以做固件检查。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（1字节）
0x00	0x25	0x00	0x01	0x01

- CMD 0x26：设备收完包后，上报固件检查结果。

  固件检查结果（1字节）：0-固件检查失败，1-固件检查成功。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（1字节）
0x00	0x26	0x00	0x01	0x01

> **说明** 0x26指令Msg ID要求与0x25指令中数据一致。

- CMD 0x2F：手机下发固件的分包数据。

Header（1字节）	CmdType（1字节）	Total Frame & Seq（1字节）	Length（1字节）	Payload（N字节）
0x00	0x2F	0xF0	0x10	0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xAA 0xBB 0xCC 0xDD 0xEE 0xFF

  固件按规范的数据长度分包，发送16包作为一次循环，Header中帧序号（Msg ID）按照0~15循环。

> **说明** Total Frame & Seq字段值为0xF0表示共16帧数据且当前数据为第1帧。

OTA数据传输与Characteristic对应关系

指令类型	说明	传输使用的Characteristic	Characteristics UUID
0x20	查询设备固件版本	WriteWithNoRsp Characteristics	0xFED7

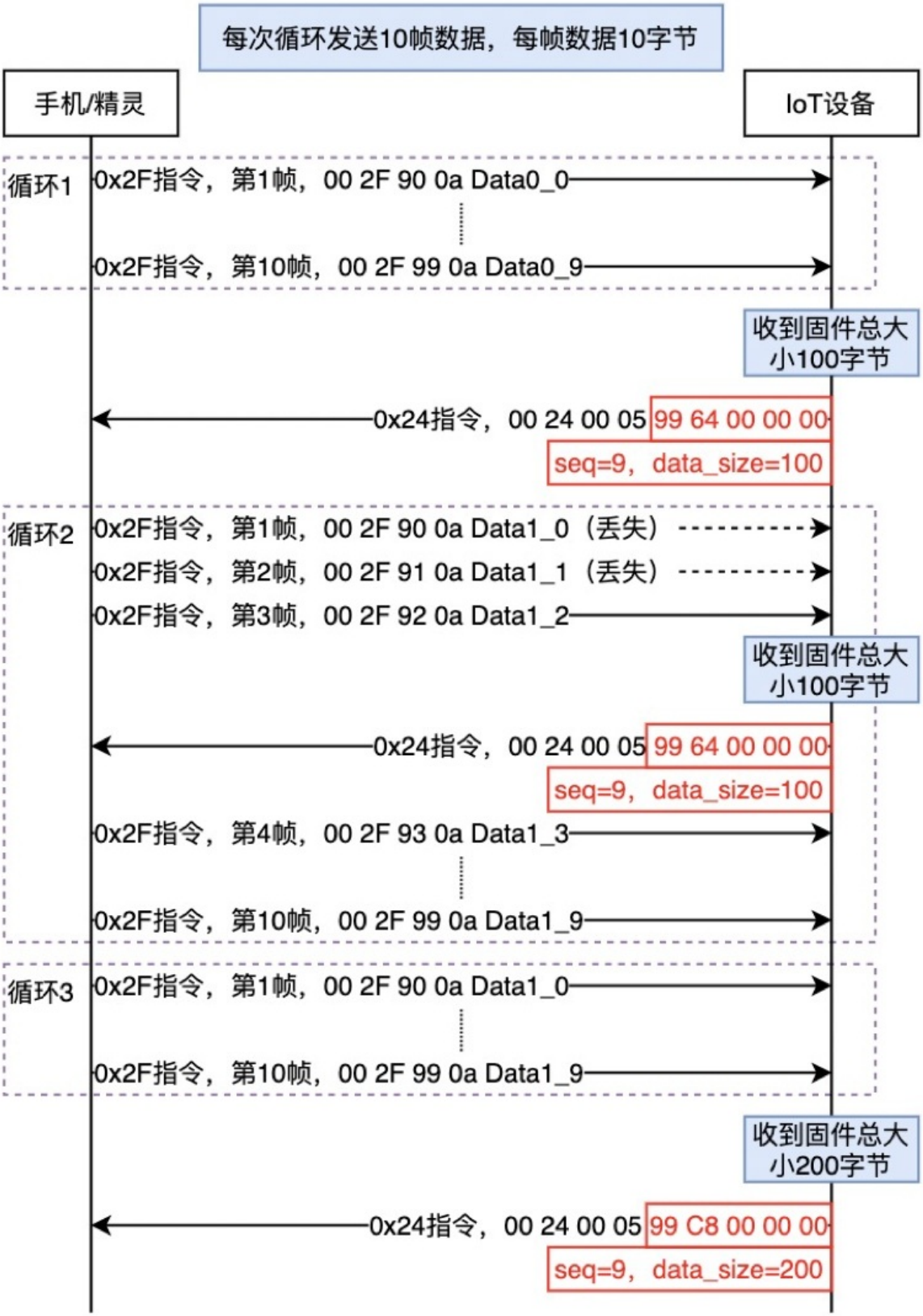
指令类型	说明	传输使用的Characteristic	Characteristics UUID
0x21	上报设备固件版本	Notify Characteristics	0xFED8
0x22	下发升级请求	WriteWithNoRsp Characteristics	0xFED7
0x23	应答升级请求	Notify Characteristics	0xFED8
0x24	上报设备最后接收到的帧序号和实际保存的数据长度	Notify Characteristics	0xFED8
0x25	开始进行固件校验	WriteWithNoRsp Characteristics	0xFED7
0x26	上报固件校验结果	Notify Characteristics	0xFED8
0x2F	固件分包数据	WriteWithNoRsp Characteristics	0xFED7

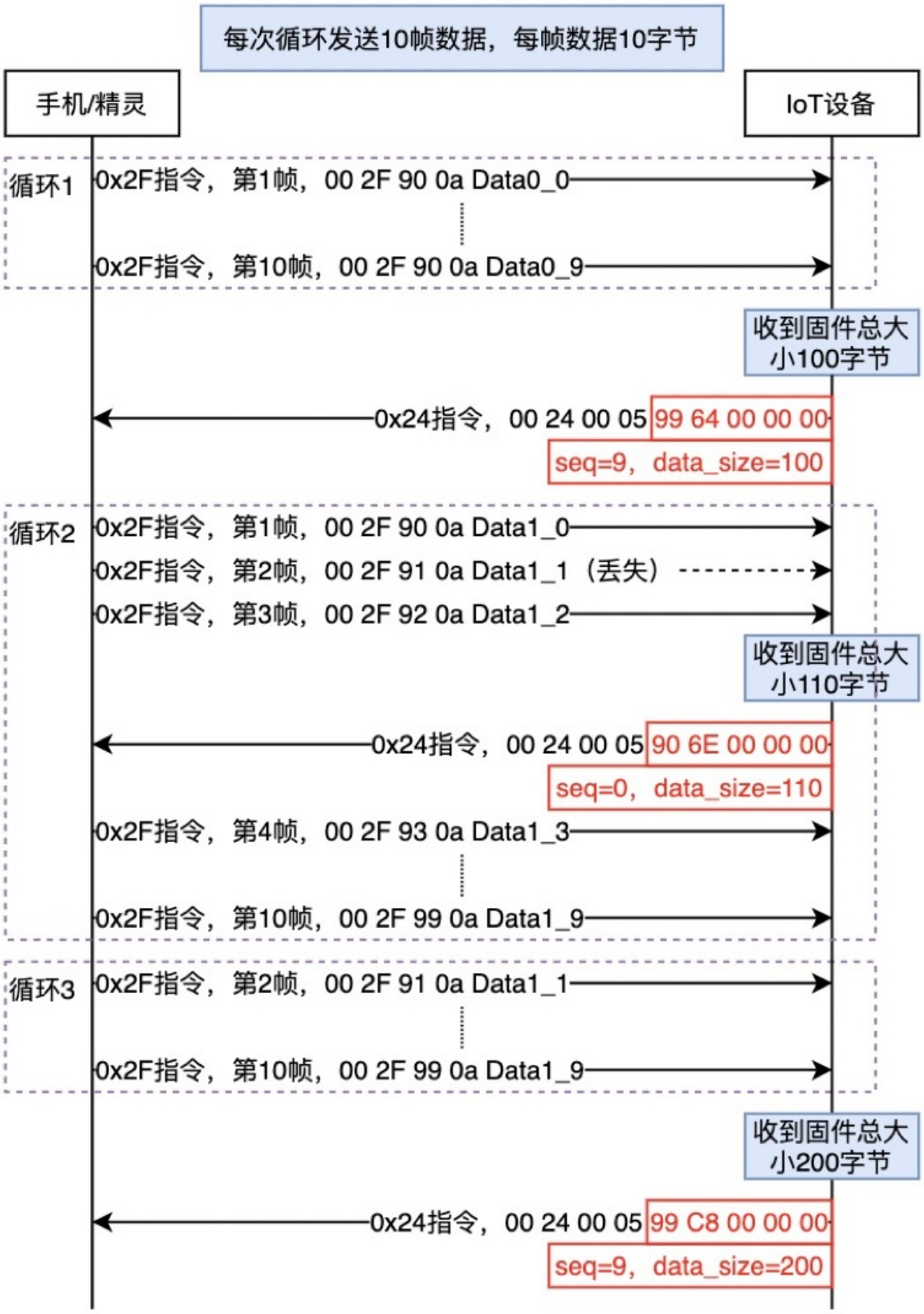
重传周期

在一个重传周期内如果连续出现同一错误，则只需发送一次0x24指令。重传周期的时间计算方式为500ms*TotalFrame；例如每次循环最大允许发送的数据帧数为10，则重传周期为500ms*10=5秒。

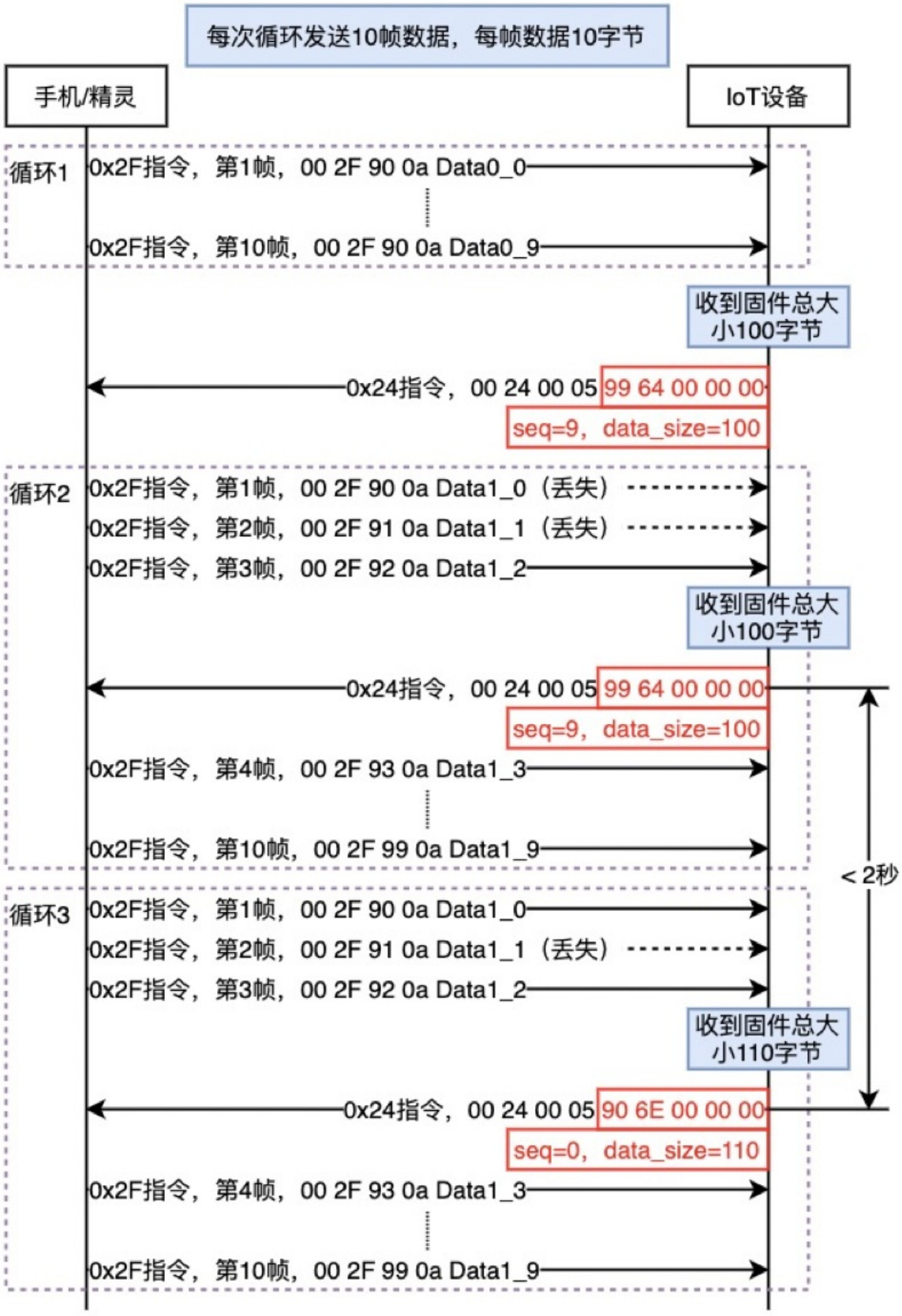
异常处理逻辑

- 当设备发现期望接收的数据包seq与当前收到的数据包seq不相同时，设备回复0x24指令，上报最后一次接收到的连续的数据包的seq和已接收的数据总长度；手机APP或天猫精灵在接收到0x24指令后，使用0x2F指令重新下发丢失的数据包，内容与之前的一致（包括Header和Payload）；设备在接收到剩余的数据包之后需要回复一次0x24指令；然后手机App或天猫精灵继续传输剩余的固件信息。

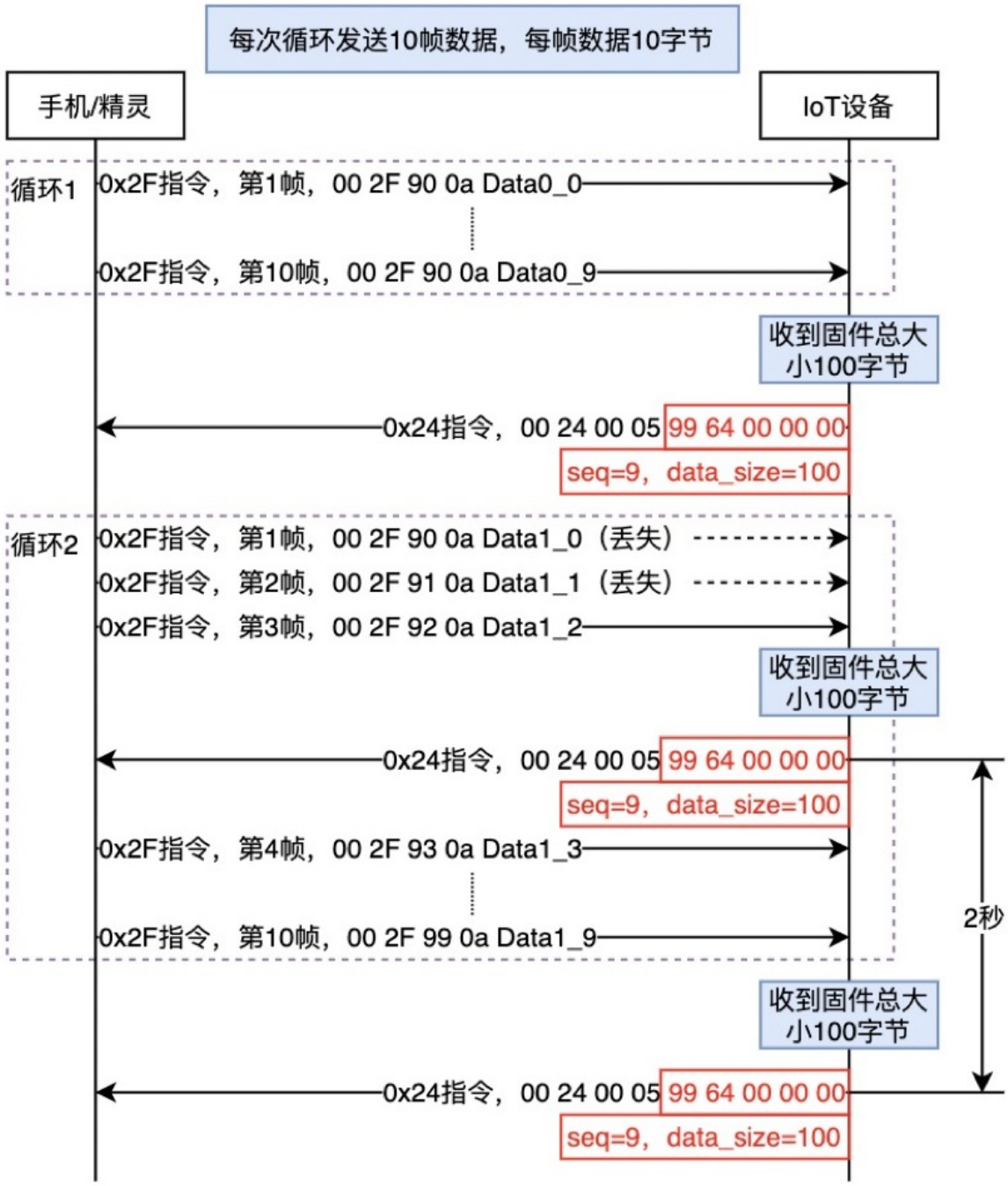

每次循环发送10帧数据，每帧数据10字节
手机/精灵
IoT设备
循环1
0x2F指令，第1帧，00 2F 90 0a Data0_0
0x2F指令，第10帧，00 2F 99 0a Data0_9
收到固件总大小100字节
0x24指令，00 24 00 05 99 64 00 00 00
seq=9，data_size=100
循环2
0x2F指令，第1帧，00 2F 90 0a Data1_0（丢失）
0x2F指令，第2帧，00 2F 91 0a Data1_1（丢失）
0x2F指令，第3帧，00 2F 92 0a Data1_2
收到固件总大小100字节
0x24指令，00 24 00 05 99 64 00 00 00
seq=9，data_size=100
0x2F指令，第4帧，00 2F 93 0a Data1_3
0x2F指令，第10帧，00 2F 99 0a Data1_9
循环3
0x2F指令，第1帧，00 2F 90 0a Data1_0
0x2F指令，第10帧，00 2F 99 0a Data1_9
收到固件总大小200字节
0x24指令，00 24 00 05 99 C8 00 00 00
seq=9，data_size=200



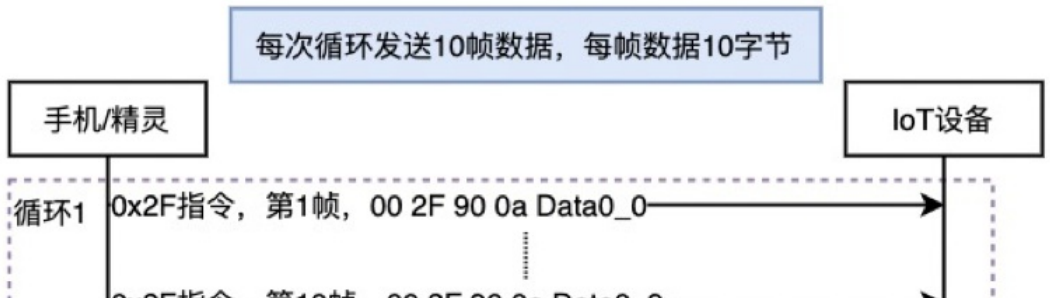
- 设备端如果发现接收数据错误，在重传周期内只允许上报一次0x24指令；如果正确接收到期望的数据则不受此规则限制。

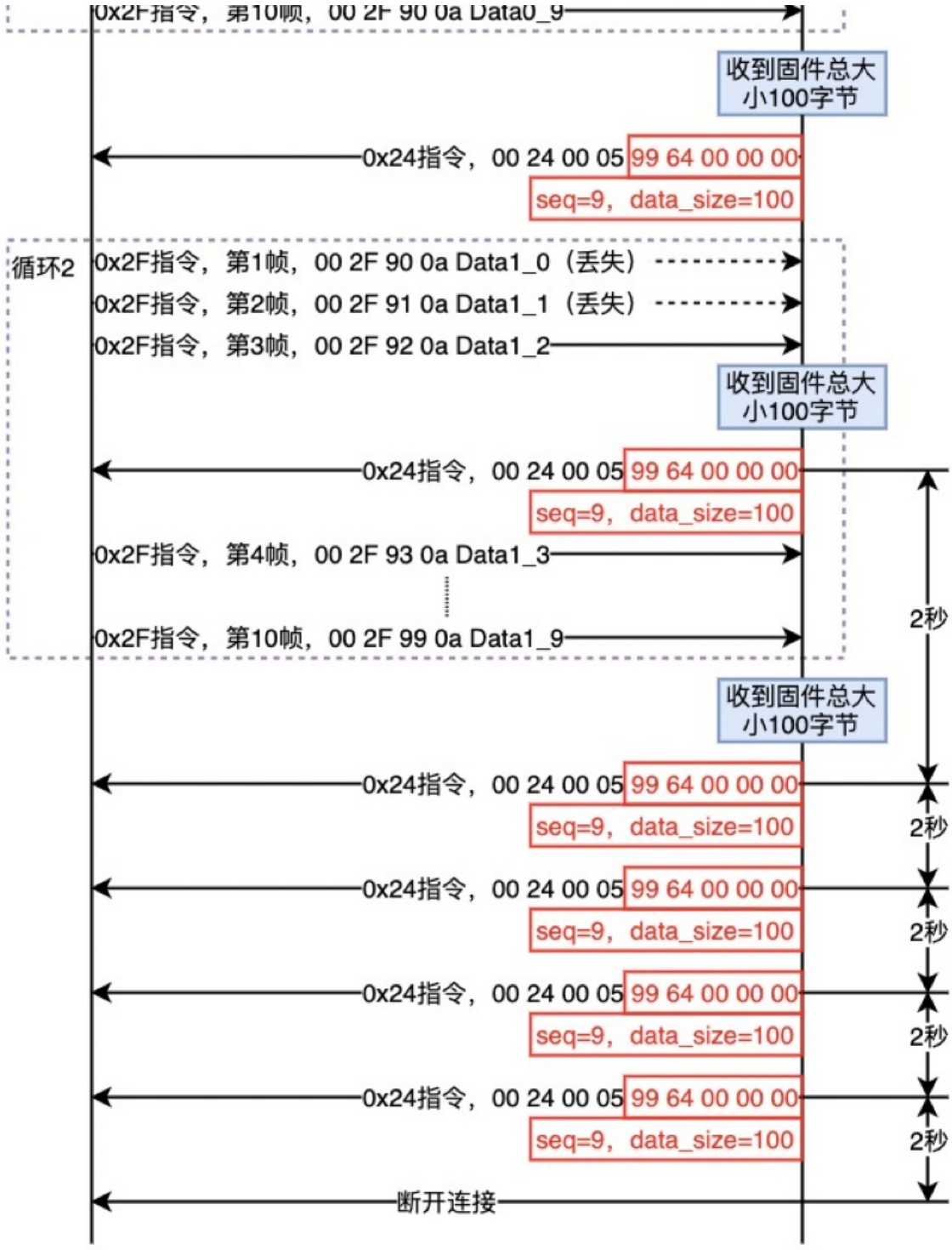


- 当设备回复0x24指令之后，如果在重传周期内没有收到期望的数据包，则再次回复0x24指令。



- 当设备端重复发送同一0x24指令超过5次时，则断开连接。





- 当传输的固件末尾最后剩余的数据小于TotalFrame*MTU时，根据实际的拆包个数确定本轮传输的TotalFrame。当设备端发现已经接收到的数据等于固件总长度时，需要直接回复0x24指令。

异常处理小结

设备端在以下情况下需要回复0x24指令。

- 当设备收齐单次循环发送的所有数据包时，需要回复0x24指令。
- 当设备收到与期望的seq不同的数据包时，需要回复0x24指令，当数据错误时每个重传周期内只允许发送1次0x24指令。

- 当设备收齐重发的单次循环发送的所有数据包时，需要回复0x24指令。
- 当设备在一个重传周期内未接收到期望的seq的数据包，需要回复0x24指令。
- 当设备收到完整固件后，需要回复0x24指令。

参考资料

CRC16算法参数模型。

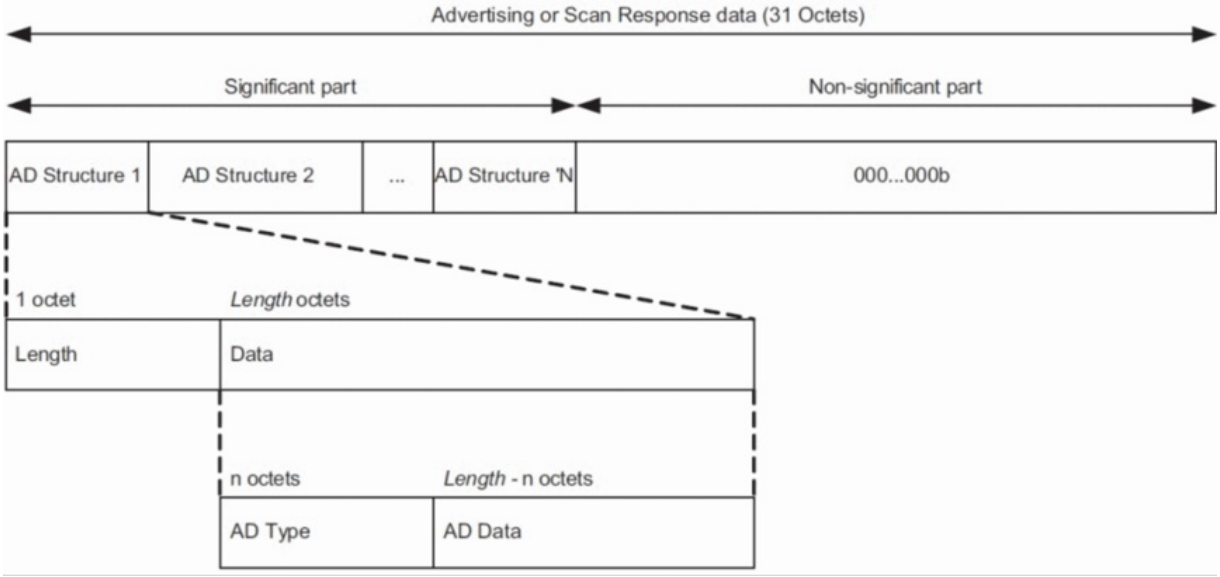
- CRC-16/CCITT-FALSE
- 宽度 16位
- 多项式POLY 0x1021
- 初始值INIT 0xFFFF
- 结果异或值XOROUT 0x0000
- 输入数据反转 REFIN false
- 输出数据反转 REFOUT false

6.6.4. 蓝牙BLE非交互式广播规范

本文对低端蓝牙设备（只能广播数据，不能交互）的BLE广播接入规范进行了详细定义。

广播规范

蓝牙广播包格式遵循蓝牙4.0规范，由若干AD Structure组成（参见Bluetooth 4.2 Core Specification, Volume 3, Part C, Chapter 11），每一个ADStructure结构由Length、AD Type、AD Data组成。如下图所示。



广播数据格式

阿里巴巴的厂商自定义格式（Manufacturer Specific Data）的AD Type须位0xFF。广播中的数据必须按照小端（Little-Endian）格式存储。

类别	字段	字节数	取值	说明
	length	1	0x2	Flags length

类别 Flags	字段	字节数	取值	说明
	AD Type	1	0x1	Flags Type
	Flags	1	0x2	Flags Value
Manufacturer Specific Data	Length	1	根据实际长度	Manufacturer Specific Data Length
	AD Type	1	固定 0xFF	Manufacturer Specific Data Type
	Company ID	2	固定 0x01A8	Company Identifiers，0x01A8为阿里巴巴公司编码
	VID和Subtype	1	0x95	取值为0x95 • bit3~0：阿里巴巴蓝牙规范版本号，当前版本号为5。 • bit7~4：Subtype使用蓝牙Beacon类型，为0b1001。
	Payload	可变	实际报文内容	无

数据传输规范

- 非交互式广播规范采用标准BLE广播的方式，数据传输采用单包形式，不支持分包。
- 数据包的Payload由以下字段组成。

字段	字节数	说明
FMSK	1	FMSK用于表示设备具有的能力
ProductID	2	平台颁发，一型一号
MAC地址	6	平台颁发，一机一号
Random	3	Random数据
Data	0~N	实体数据
CRC	1	CRC签名校验

- FMSK用于表示设备具有的能力，各比特位定义如下。

比特序	说明
0	是否是交互式广播，0：非交互式，1：交互式。非交互式广播填0
2~1	蓝牙广播规范版本，目前为 0b00
7~3	保留将来使用，全部填0

- Random和 Digest 用于表示设备的身份。
  - 定义Random是3字节的随机字节，必须采用真随机算法，可以使用adc采样值来作为随机种子。
  - 三元组（又称设备证书，ProductKey、DeviceName、DeviceSecret）预先烧录在设备中。为了保证安全，Secret不进行空中传播，也不会传递到手机端。
  - 定义 `Data = ali_encrypt(Secret, Random)` ，厂商自定义数据。ali_encrypt是我们针对低端蓝牙设备设计的一种加密算法，由平台提供。

    > **说明** ali_encrypt由平台提供，如有需要请联系商务获取。

  - 定义 `CRC=CRC16(Secret, Data)` ，取低位字节。

数据Payload说明

- 如果需要厂商使用自定义的数据格式，需到云端配置脚本实现数据转换。
- 根据使用场景，厂商自定义数据为空时候，需要用常量来填充。
- 常量（ASCII字符串）"BIND"：用于设备绑定的标识，设备上电后一般不会产生数据，用该常量进行填充。

广播收发规范

- 设备手动上电后，需要连续广播包2分钟，由于自定义数据这个时候为空，需要用常量BIND来填充。
- 绑定设备成功后，App在进入面板后需要实时的扫描目标设备的广播包（天猫精灵音箱则是一直实时扫描）。
- 对设备广播的频率，需要进行如下约定：上报频率较低的设备，体脂秤等，一次有效数据需要持续5秒的时间。对于一次有效数据，5秒之内发送同一个数据包，其Random保持不变。

6.7. 蓝牙设备功能编码

6.7.1. 蓝牙mesh智能家居产品规范

本文介绍蓝牙mesh智能家居产品的属性、事件和场景模式等定义，供开发相关产品时查阅。自有品牌项目与天猫精灵生态项目均遵循相同的规范。

智能家居设备描述

为了适配各种智能家居设备，我们将智能家居设备抽象成属性，事件和场景模式，具体定义如下。

- 属性：包括设备具有的系统属性和物理属性，也包括设备具有的特殊功能，我们通过设置属性来改变设备的工作状态或让设备开启、关闭它的特殊功能。
- 事件：设备在运行过程中出现了需要用户来手动干预的特殊情况，这时候上报给用户的通知；或者是用户主动激活的某些事件，例如按键。如果某个事件引发了设备状态改变，则在事件被解决后，需要上报“事件清除”用于告知云端问题事件已被清除。
- 场景模式：设备为了完成某些特定功能而预设在设备内的一系列属性值，便于人们理解，在某一个时刻，设备只能处于一个场景模式。

> **说明** 如果属性，事件，场景模式无法满足产品开发需求，请及时与我们联系（aligenie.iot@list.alibaba-inc.com）添加新的属性，事件，场景模式类型。

设备属性表

设备属性定义请参见蓝牙设备属性表。

设备事件表

对智能家居设备定义了以下事件。

事件名	事件ID	事件参数	说明
故障上报	0x00	• uint16 Error_Code_Type • uint8 Error_code_Value	
低电量	0x01	• uint16 power_percent_Type • uint8 power_percent_Value	
设备宕机	0x02	NULL	
设备上电	0x03	NULL	
按键单击	0x05	uint8 key code	
按键双击	0x06	uint8 key code	
按键长按	0x07	uint8 key code	
水位过高	0x08	NULL	
水位过低	0x09	NULL	
温度过高（水温过高）	0x0A	NULL	
温度过低（水温过低）	0x0B	NULL	
缺少器件	0x0C	NULL	例如水盒未装，尘盒未装
天然气、煤气报警	0x0D	NULL	
烟雾报警	0x0E	NULL	
火警报警	0x0F	NULL	
闯入报警	0x10	NULL	
定时完成	0x11	uint8 index	
设备联网	0x12	NULL	
开始充电	0x13	NULL	
充电完成	0x14	NULL	

事件名	事件ID	事件参数	说明
余量不足	0x15	NULL	设备耗材余量过低时上报该事件
门未锁报警	0x16	NULL	
劫持报警	0x17	• uint16 userID_Type • uint8 userID_Value • uint16 lockType_Type • uint8 lockType_Value	
防撬报警	0x18		
禁试报警	0x19		
滤芯寿命到期	0x1A	NULL	
传感器开路	0x1B	NULL	
传感器短路	0x1C	NULL	
开始测量	0x1D	NULL	例如体脂秤开始测量体脂体重
添加钥匙	0x1E	• uint16 userID_Type • uint8 userID_Value • uint16 userLimit_Type • uint8 userLimit_Value • uint16 lockType_Type • uint8 lockType_Value • uint16 entryinfoForceTime_Type • uint8 entryinfoForceTime_Value • uint16 activeTimePerDay_Type • uint16 activeTimePerDay_Value • uint16 inactiveTimePerDay_Type • uint16 inactiveTimePerDay_Value • uint16 startTime_Type • uint32 startTime_Value • uint16 ExpiredTime_Type • uint32 ExpiredTime_Value	
删除钥匙	0x1F	• uint16 userID_Type • uint8 userID_Value • uint16 lockType_Type • uint8 lockType_Value	

事件名	事件ID	事件参数	说明
开锁通知	0x20	• uint16 userID_Type • uint8 userID_Value • uint16 userLimit_Type • uint8 userLimit_Value • uint16 lockType_Type • uint8 lockType_value	
布防开门报警	0x21	NULL	
更新钥匙信息	0x22	• uint16 userID_Type • uint8 userID_Value • uint16 userLimit_Type • uint8 userLimit_Value • uint16 lockType_Type • uint8 lockType_Value • uint16 entryinfoForceTime_Type • uint8 entryinfoForceTime_Value • uint16 activeTimePerDay_Type • uint16 activeTimePerDay_Value • uint16 inactiveTimePerDay_Type • uint16 inactiveTimePerDay_Value • uint16 startTime_Type • uint32 startTime_Value • uint16 ExpiredTime_Type • uint32 ExpiredTime_Value	
硬件复位	0x23	NULL	用户手动操作让硬件恢复出厂设置（重新进入待配网状态）
锁重置	0x24	uint8 deletedLockType	
冲洗完成	0x25	NULL	
一氧化碳状态	0x26	NULL	
工作完成	0x27	NULL	
锁定报警	0x28	NULL	
水浸报警	0x29	NULL	
刷头更换提醒	0x2A	NULL	
存放提醒	0x2B		

事件名	事件ID	事件参数	说明
过热提示	0x2C		
云食谱步骤完成	0x2D		
设备端操作事件	0x2F		
洗手提示	0x30		
洗手液补充提示	0x31	NULL	
洗手事件	0x32	NULL	
底座未放置事件	0x33	NULL	
被围困事件	0x34	NULL	
喝水提醒事件	0x35	NULL	
清洗提醒时间	0x36		NULL
服药提醒事件	0x37	unit32 position	
携带提示事件	0x38	NULL	
更换提示事件	0x39	NULL	
睡前提示事件	0x3A	NULL	
云食谱结束事件	0x3B		
豆浆机错误事件	0x3C		
浆杯未正确放置事件	0x3D		
未合盖事件	0x3E		
物料过多事件	0x3F		
电压异常事件	0x40		
不可取消事件	0x41		
杯体放置错误事件	0x42		
设备倾斜事件	0x43		
刷牙记录上报事件	0x44		

设备场景模式表

对智能家居设备定义了以下场景模式：

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0000	0	保留
0x0001	1	默认模式，厂商默认状态
0x0002	2	自动模式
0x0003	3	阅读模式
0x0004	4	影院模式
0x0005	5	温暖模式
0x0006	6	夜灯模式
0x0007	7	助眠模式
0x0008	8	起床模式
0x0009	9	制冷模式
0x000A	10	制热模式
0x000B	11	通风模式
0x000C	12	送风模式
0x000D	13	除湿模式
0x000E	14	睡眠模式
0x000F	15	生活模式
0x0010	16	手动模式
0x0011	17	静音模式
0x0012	18	省电模式
0x0013	19	正常风模式
0x0014	20	自然风模式
0x0015	21	睡眠风模式
0x0016	22	静音风模式
0x0017	23	舒适风模式
0x0018	24	宝宝风模式
0x0019	25	棉织物模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x001A	26	化纤模式
0x001B	27	羊毛模式
0x001C	28	除菌模式
0x001D	29	筒清洁模式
0x001E	30	丝绸模式
0x001F	31	假日模式
0x0020	32	智能模式
0x0021	33	音乐模式
0x0022	34	零重力模式
0x0023	35	止鼾模式
0x0024	36	多人模式
0x0025	37	摇摆模式
0x0026	38	强效模式
0x0027	39	普通模式
0x0028	40	工作模式
0x0029	41	速冷模式
0x002A	42	速冻模式
0x002B	43	微干模式
0x002C	44	全干模式
0x002D	45	超干模式
0x002E	46	夏季模式
0x002F	47	冬季模式
0x0030	48	标准模式
0x0031	49	快洗模式
0x0032	50	婴童洗模式
0x0033	51	单脱水模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0034	52	节能洗模式
0x0035	53	智能光感模式
0x0036	54	学习模式
0x0037	55	睡前模式
0x0038	56	护眼智能感光模式
0x0039	57	护眼模式
0x003A	58	粉碎
0x003B	59	真空
0x003C	60	搅拌
0x003D	61	温热
0x003E	62	解冻
0x003F	63	发酵
0x0040	64	开盖煮
0x0041	65	烘烤、烘焙
0x0042	66	蒸汽烘烤
0x0043	67	炖煮模式
0x0044	68	蒸炖、蒸煮、营养蒸
0x0045	69	无水焗
0x0046	70	煎炸
0x0047	71	火锅
0x0048	72	炒菜
0x0049	73	烧水
0x004A	74	甜品
0x004B	75	除湿风模式
0x004C	76	炖红薯
0x004D	77	肉类

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x004E	78	鸡鸭
0x004F	79	牛羊肉
0x0050	80	鱼类、鱼汤
0x0051	81	大骨
0x0052	82	排骨
0x0053	83	蹄筋、豆
0x0054	84	时蔬
0x0055	85	冷冻食品
0x0056	86	面点
0x0057	87	披萨
0x0058	88	叉烧
0x0059	89	煎带鱼
0x005A	90	宝宝薯丁
0x005B	91	烤面包
0x005C	92	蛋糕
0x005D	93	中药
0x005E	94	凉茶
0x005F	95	滋补品、炖品、燕窝
0x0060	96	养生茶
0x0061	97	花茶、花草茶
0x0062	98	希腊酸奶
0x0063	99	奶酪
0x0064	100	酸奶
0x0065	101	豆浆
0x0066	102	五谷、五谷浆、五谷豆浆
0x0067	103	米糊

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0068	104	果汁、果蔬汁
0x0069	105	浓汤
0x006A	106	沙冰
0x006B	107	冰沙
0x006C	108	干豆
0x006D	109	湿豆
0x006E	110	干湿豆
0x006F	111	果仁露
0x0070	112	果茶、水果茶
0x0071	113	精力汤
0x0072	114	熟料打
0x0073	115	咖啡模式
0x0074	116	酱料模式
0x0075	117	米酒
0x0076	118	宝宝糊
0x0077	119	汤糊
0x0078	120	奶昔
0x0079	121	快煮饭
0x007A	122	标准饭
0x007B	123	精煮饭
0x007C	124	灶烧饭
0x007D	125	粗粮饭
0x007E	126	煲仔饭
0x007F	127	寿司饭
0x0080	128	石锅饭
0x0081	129	柴火饭

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0082	130	稀饭
0x0083	131	煮粥
0x0084	132	八宝粥
0x0085	133	杂粮粥
0x0086	134	热饭
0x0087	135	婴儿粥、宝宝粥
0x0088	136	养生粥
0x0089	137	海鲜粥
0x008A	138	脚底按摩
0x008B	139	小腿按摩
0x008C	140	热风烘干
0x008D	141	加热冲浪
0x008E	142	MED模式
0x008F	143	HARD模式
0x0090	144	放松模式
0x0091	145	舒缓模式
0x0092	146	活络模式
0x0093	147	养神模式
0x0094	148	舒颈模式
0x0095	149	升温模式
0x0096	150	3D摆风
0x0097	151	婴儿风模式
0x0098	152	干衣模式
0x0099	153	标准风模式
0x009A	154	时间倒计模式
0x009B	155	卡路里倒计模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x009C	156	距离倒计模式
0x009D	157	搅拌模式
0x009E	158	加热模式
0x009F	159	加热搅拌模式
0x00A0	160	和面模式
0x00A1	161	清洗模式
0x00A2	162	爆炒模式
0x00A3	163	慢炖模式
0x00A4	164	暴风模式
0x00A5	165	智能风模式
0x00A6	166	循环风模式
0x00A7	167	老人风模式
0x00A8	168	果蔬模式
0x00A9	169	海鲜模式
0x00AA	170	面食模式
0x00C3	195	轻柔模式
0x00C4	196	美白模式
0x00C5	197	护龈模式
0x00C6	198	抛光模式
0x00C7	199	半胆模式
0x00C8	200	整胆模式
0x00C9	201	加湿模式
0x00CA	202	家纺洗模式
0x00CB	203	淋浴模式
0x00CC	204	水温按摩模式
0x00CD	205	高温模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x00CE	206	预约模式
0x00CF	207	定速模式
0x00D0	208	上下管模式
0x00D1	209	发酵模式
0x00D2	210	果干模式
0x00D3	211	热风模式
0x00D4	212	放松模式
0x00D5	213	美颜模式
0x00D6	214	明目模式
0x00D7	215	敲打模式
0x00D8	216	按压模式
0x00D9	217	揉捏模式
0x00DA	218	无敲打模式
0x00DB	219	搓揉模式
0x00DC	220	写作模式
0x00DD	221	呼吸模式
0x00DF	223	闪烁模式
0x00E0	224	舞动模式
0x00E1	225	唤醒模式
0x00E2	226	明亮模式
0x00E3	227	柔和模式
0x00E4	228	健康模式
0x00E5	229	电加热模式
0x00E6	230	节能模式
0x00E7	231	预热模式
0x00E8	232	单人模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x00E9	233	喷射模式
0x00EA	234	强力洗模式
0x00EB	235	恒湿模式
0x00EC	236	减肥模式
0x00ED	237	温控模式
0x00EE	238	充电模式
0x00EF	239	低功率模式
0x00F0	240	速热模式
0x00F1	241	肉类模式
0x00F2	242	老人模式
0x00F3	243	净化模式
0x00F4	244	复合模式
0x00F5	245	厨房模式
0x00F6	246	浴缸模式
0x00F7	247	全日制模式
0x00F8	248	夜间模式
0x00F9	249	蔬菜模式
0x00FA	250	美味蒸模式
0x00FB	251	随心蒸模式
0x00FC	252	即时加热模式
0x00FD	253	自定义模式
0x00FE	254	儿童模式
0x00FF	255	强力模式
0x0100	256	日常消毒
0x0101	257	强力消毒
0x0102	258	紫外消毒

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0103	259	包子模式
0x0104	260	烘干模式
0x0105	261	馒头模式
0x0106	262	暖碟模式
0x0107	263	肉类模式
0x0108	264	全身舒展按摩
0x0109	265	轻松按摩
0x010A	266	酸痛改善
0x010B	267	肩颈重点
0x010C	268	腰椎舒缓
0x010D	269	总裁养身
0x010E	270	女王纤体
0x010F	271	手动按摩模式
0x0110	272	久坐释压
0x0111	273	男士健体
0x0112	274	女王美体
0x0113	275	摇摆助眠
0x0114	276	午间小休
0x0116	278	泰式拉伸
0x0117	279	运动恢复
0x0118	280	区域清扫
0x0119	281	定点清扫
0x011A	282	指定房间清扫
0x011B	283	自动清扫
0x011C	284	地毯清扫
0x011D	285	杀菌清扫

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x011E	286	延边清扫
0x011F	287	标准清扫
0x0120	288	静音清扫
0x0121	289	强力清扫
0x0122	290	除螨模式
0x0123	291	保温模式
0x0124	292	棉麻洗模式
0x0125	293	超净洗模式
0x0126	294	运动服模式
0x0127	295	大件洗模式
0x0128	296	内衣洗模式
0x0129	297	户外服洗模式
0x012A	298	羽绒服洗模式
0x012B	299	护色洗模式
0x012C	300	空气洗模式
0x012D	301	衬衫洗模式
0x012E	302	单烘干模式
0x012F	303	自动煮水模式
0x0130	304	高温烘干
0x0131	305	节能烘干
0x0132	306	浸泡洗模式
0x0133	307	防过敏洗
0x0134	308	快烘模式
0x0135	309	毛毯洗模式
0x0136	310	春秋模式
0x0137	311	夜间洗模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0138	312	夜间烘模式
0x0139	313	搅拌一
0x013A	314	搅拌二
0x013B	315	搅拌三
0x013C	316	搅拌四
0x013D	317	醒面一
0x013E	318	醒面二
0x013F	319	醒面三
0x0140	320	醒面四
0x0141	321	发酵一
0x0142	322	发酵二
0x0143	323	发酵三
0x0144	324	发酵四
0x0145	325	烘烤一
0x0146	326	烘烤二
0x0147	327	烘烤三
0x0148	328	烘烤四
0x0149	329	搅拌烘烤
0x014A	330	搅拌醒面
0x014B	331	发酵烘烤
0x014C	332	待机
0x014D	333	卸妆模式
0x014E	334	深层补水
0x014F	335	肌肤按摩
0x0150	336	晒后修复
0x0151	337	敏感肌肤处理

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0152	338	控油模式
0x0153	339	玻璃洗
0x0154	340	一小时洗
0x0155	341	预冲洗
0x0156	342	自定义洗
0x0157	343	标准洗
0x0158	344	超快洗
0x0159	345	自由模式
0x015A	346	减脂模式
0x015B	347	健康小走模式
0x015C	348	体能特训模式
0x015D	349	马拉松模式
0x015E	350	心率控速模式
0x015F	351	反转模式
0x0160	352	校准模式
0x0161	353	正常模式
0x0162	354	恒功模式
0x0163	355	恒温模式
0x0164	356	吹风模式
0x0165	357	风暖弱
0x0166	358	风暖强
0x0167	359	自动沐浴
0x0168	360	干燥模式
0x0169	361	活力模式
0x016A	362	极速模式
0x016B	363	混合模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x016C	364	增容模式
0x016D	365	自适应节能
0x016E	366	高温消毒
0x016F	367	疲劳恢复按摩
0x0170	368	低温烘干
0x0171	369	E加增容
0x0172	370	智能防霉
0x0173	371	心灵SPA
0x0174	372	催乳模式
0x0175	373	集乳模式
0x0176	374	消毒模式
0x0177	375	自动消毒模式
0x0178	376	煮沸模式
0x0179	377	自动煮沸模式
0x017A	378	休闲模式
0x017B	379	平躺模式
0x017C	380	减压模式
0x017D	381	冷喷模式
0x017E	382	热喷模式
0x017F	383	温喷模式
0x0180	384	清洁模式
0x0181	385	二氧化碳模式
0x0182	386	经济模式
0x0183	387	防霜模式
0x0184	388	舒适模式
0x0185	389	上下擦窗模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0186	390	左右擦窗模式
0x0187	391	左区域模式
0x0188	392	右区域模式
0x0189	393	清洗轮组模式
0x018A	394	除氯模式
0x018B	395	会客模式
0x018C	396	微光模式
0x018D	397	洗漱模式
0x018E	398	爸爸模式
0x018F	399	妈妈模式
0x0190	400	长辈模式
0x0191	401	登山模式
0x0192	402	挑战模式
0x0193	403	温调模式
0x0194	404	热调模式
0x0195	405	世卫模式
0x0196	406	泡茶模式
0x0197	407	温泉模式
0x0198	408	快速模式
0x0199	409	定时休息模式
0x019A	410	舒适安眠模式
0x019B	411	老年养生模式
0x019C	412	背部舒展模式
0x019D	413	加强模式
0x019E	414	冷热喷模式
0x019F	415	棉麻模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x01A0	416	牛仔模式
0x01A1	417	床单被套
0x01A2	418	户外服模式
0x01A3	419	羽绒服模式
0x01A4	420	毛绒玩具模式
0x01A5	421	除潮去味模式
0x01A6	422	冷风清新模式
0x01A7	423	热风烘干模式
0x01A8	424	内衣模式
0x01A9	425	婴儿服模式
0x01AA	426	衬衫模式
0x01AB	427	快烘三十模式
0x01AC	428	毛巾模式
0x01AD	429	智能干衣专家模式
0x01AE	430	蒸汽护理模式
0x01AF	431	大件模式
0x01B0	432	定时烘模式
0x01B1	433	支架烘模式
0x01B2	434	西裤模式
0x01B3	435	智能烘模式
0x01B4	436	四件套模式
0x01B5	437	暖衣模式
0x01B6	438	快烘二十模式
0x01B7	439	蒸汽除菌模式
0x01B8	440	大件六十模式
0x01B9	441	蒸汽熨模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x01BA	442	床上用品模式
0x01BB	443	塑料奶瓶模式
0x01BC	444	硅胶奶瓶模式
0x01BD	445	玻璃奶瓶模式
0x01BE	446	西装模式
0x01BF	447	衣帽配件模式
0x01C0	448	其他衣物模式
0x01C1	449	雨淋烘干模式
0x01C2	450	定时烘干模式
0x01C3	451	标准杀菌模式
0x01C4	452	寝具杀菌模式
0x01C5	453	内衣杀菌模式
0x01C6	454	玩具杀菌模式
0x01C7	455	换气强暖模式
0x01C8	456	换气弱暖模式
0x01C9	457	换气自然风模式
0x01CA	458	强劲风模式
0x01CB	459	煮洗
0x01CC	460	手动消毒模式
0x01CD	461	毛巾除菌洗
0x01CE	462	运动即时洗
0x01CF	463	婴童特渍洗
0x01D0	464	新衣物首洗
0x01D1	465	真丝呵护洗
0x01D2	466	快速筒清洗
0x01D3	467	漂脱模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x01D4	468	快洗烘模式
0x01D5	469	中温模式
0x01D6	470	舒适浴模式
0x01D7	471	随温感模式
0x01D8	472	预约一模式
0x01D9	473	预约二模式
0x01DA	474	换气模式
0x01DB	475	暖奶模式
0x01DC	476	快速暖奶模式
0x01DD	477	辅食模式
0x01DE	478	降温模式
0x01DF	479	果蔬洗模式
0x01E0	480	暖菜模式
0x01E1	481	瓷盘暖菜模式
0x01E2	482	温炖模式
0x01E3	483	暖酒模式
0x01E4	484	暖茶模式
0x01E5	485	解冻模式
0x01E6	486	取暖模式
0x01E7	487	防潮模式
0x01E8	488	温和灸模式
0x01E9	489	雀啄灸模式
0x01EA	490	瘢痕灸模式
0x01EB	491	宝宝粥模式
0x01EC	492	蒸煮模式
0x01ED	493	煮粥模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x01EE	494	米糊模式
0x01EF	495	果泥模式
0x01F0	496	菜泥模式
0x01F1	497	鱼泥模式
0x01F2	498	瓜泥模式
0x01F3	499	肉泥模式
0x01F4	500	蒸蛋模式
0x01F5	501	奶昔模式
0x01F6	502	全自动蒸煮搅拌模式
0x01F7	503	自动搅拌模式
0x01F8	504	自定义搅拌模式
0x01F9	505	无重力模式
0x01FA	506	放平模式
0x01FB	507	热敷模式
0x01FC	508	连续模式
0x0200	512	排水模式
0x0205	517	停止按摩
0x0206	518	餐具模式
0x0207	519	羟基模式
0x0208	520	活氧模式
0x0209	521	排水模式
0x020A	522	意式咖啡
0x020B	523	美式咖啡
0x020C	524	拿铁
0x020D	525	卡布奇诺
0x020E	526	保管模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x020F	527	防冻模式
0x0210	528	循环模式
0x0211	529	日程模式
0x0212	530	强制加热模式
0x0213	531	静泡模式
0x0214	532	强劲模式
0x0215	533	烘干消毒模式
0x0216	534	自清洁模式
0x0217	535	自定义二模式
0x0218	536	自定义三模式
0x0219	537	沐足模式
0x021A	538	洗衣模式
0x021B	539	畅享浴模式
0x021C	540	混水洗模式
0x021D	541	温水浴模式
0x021E	542	宝宝洗模式
0x021F	543	空挡洗模式
0x0220	544	晶柔洗模式
0x0221	545	日常洗模式
0x0222	546	即时洗模式
0x0223	547	智能洗模式
0x0224	548	餐前洗模式
0x0225	549	自清洗模式
0x0226	550	蒸汽模式
0x0227	551	蒸汽热风
0x0228	552	上下烧烤

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0229	553	热风对流
0x022A	554	立体烤
0x022B	555	底部烧烤
0x022C	556	薄层快烤
0x022D	557	双上管
0x022E	558	红外烧烤
0x022F	559	除垢
0x0230	560	蒸汽杀菌
0x0231	561	左出风模式
0x0232	562	右出风模式
0x0233	563	左右出风模式
0x0234	564	常规灸
0x0235	565	舒适灸
0x0236	566	温灸
0x0237	567	隔物灸
0x0238	568	热灸
0x0239	569	暖足模式
0x023A	570	激活模式
0x023B	571	按摩模式
0x023C	572	恢复模式
0x023D	573	懒人模式
0x023E	574	即时消毒模式
0x023F	575	定时消毒模式
0x0240	576	次数倒计模式
0x0241	577	凉干燥模式
0x0242	578	热干燥模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x0243	579	烧水模式
0x0244	580	调温模式
0x0245	581	调奶模式
0x0246	582	漂洗模式
0x0247	583	精洗模式
0x0248	584	亮晶洗模式
0x0249	585	超声波模式
0x024A	586	鲜肉洗模式
0x024B	587	臭氧模式
0x024C	588	干拖模式
0x024D	589	敏感模式
0x024E	590	点喷模式
0x024F	591	滴喷模式
0x0250	592	挤出模式
0x0251	593	婴儿洗模式
0x0252	594	成人洗模式
0x0253	595	老人洗模式
0x0254	596	高温水模式
0x0255	597	洗碗模式
0x0256	598	智温感模式
0x0257	599	热食模式
0x0258	600	左加热模式
0x0259	601	右加热模式
0x025A	602	左右加热模式
0x025B	603	疲劳模式
0x025C	604	推拿模式

Scene Number1（十六进制）	Scene Number（十进制）	模式名称
0x025D	605	叩击模式
0x025E	606	针灸模式
0x025F	607	拔罐模式
0x0260	608	牵引模式
0x0261	609	拉伸模式
0x0262	610	超强洗模式
0x0263	611	轻柔洗模式
0x0264	612	快速洗模式
0x0265	613	正转模式
0x0266	614	定时模式
0x0267	615	杀菌模式
0x0268	616	除湿烘干模式
0x0269	617	海鲜洗模式
0x026A	618	火锅洗模式
0x026B	619	少量洗模式
0x026C	620	静音洗模式
0x026D	621	抑菌储存模式
0x026E	622	祛潮模式

错误码定义

当Vendor Model Server收到一条指令之后无法执行时，需要向Vendor Model Client回复错误码，错误码定义如下。

Error Code（十六进制）	Error Code（十进制）	释义
0x00~0x7F	0~127	sig mesh Current Fault
0x80	128	设备未准备好
0x81	129	不支持的属性
0x82	130	不支持的操作
0x83	131	参数错误

Error Code（十六进制）	Error Code（十进制）	释义
0x84	132	设备状态错误
0x85	133	未找到索引
0x88	136	传感器短路
0x89	137	进水温度过低
0x8A	138	一氧化碳超标
0x8B	139	点火失败
0x8C	140	意外熄火
0x8D	141	残火
0x8E	142	漏电故障
0x8F	143	干烧故障
0x90	144	超温故障
0x91	145	传感器故障
0x92	146	感应异常故障
0x93	147	变频微波故障
0x94	148	炉腔过热
0x95	149	正常
0x96	150	内部感温元件短路
0x97	151	内部感温元件开路
0x98	152	内部温度过高
0x99	153	外部感温元件短路
0x9A	154	外部感温元件开路
0x9B	155	发热丝断开
0x9C	156	无线信号弱、断开
0x9D	157	监测器信号弱、断开
0x9E	158	监测器故障
0x9F	159	进水异常

Error Code（十六进制）	Error Code（十进制）	释义
0xA0	160	加热异常
0xA1	161	溢流报警
0xA2	162	通讯故障下控收不到上控数据
0xA3	163	失速保护
0xA4	164	速度感应失败
0xA5	165	扬升学习失败
0xA6	166	过载保护
0xA7	167	安全开关脱落
0xA8	168	通讯故障上控收不到下控数据
0xA9	169	电机开路
0xAA	170	漏水故障
0xAB	171	制水保护
0xAC	172	缺水、低压报警
0xAD	173	原水箱缺水报警
0xAE	174	加热传感故障
0xAF	175	制冷传感故障
0xB0	176	净水箱高位浮子逻辑故障
0xB1	177	净水箱低位浮子逻辑故障
0xB2	178	净水箱浮子逻辑故障
0xB3	179	板卡通信故障
0xB4	180	上报频次异常
0xB5	181	传感器过热
0xB6	182	工作电压过低
0xB7	183	工作电压过高
0xB8	184	锅具干烧
0xB9	185	主传感器失效

Error Code（十六进制）	Error Code（十进制）	释义
0xBA	186	错误一
0xBB	187	错误二
0xBC	188	错误三
0xBD	189	错误四
0xBE	190	错误五
0xBF	191	错误六
0xC0	192	错误七
0xC1	193	错误八
0xC2	194	错误九
0xC3	195	错误十
0xC4	196	错误十一
0xC5	197	错误十二
0xC6	198	错误十三
0xC7	199	错误十四
0xC8	200	错误十五
0xC9	201	错误十六
0xCA	202	错误十七
0xCB	203	错误十八
0xCC	204	错误十九
0xCD	205	错误二十
0xCE	206	错误二十一
0xCF	207	错误二十二
0xD0	208	错误二十三
0xD1	209	错误二十四
0xD2	210	错误二十五
0xD3	211	错误二十六

Error Code（十六进制）	Error Code（十进制）	释义
0xD4	212	错误二十七
0xD5	213	错误二十八
0xD6	214	错误二十九
0xD7	215	错误三十
0xD8	216	错误三十一
0xD9	217	错误三十二
0xDA	218	错误三十三
0xDB	219	错误三十四
0xDC	220	错误三十五
0xDD	221	错误三十六
0xDE	222	错误三十七
0xDF	223	错误三十八
0xE0	224	错误三十九
0xE1	225	开盖报警
0xE2	226	温度报警
0xE3	227	LED灯损报警
0xE4	228	温度传感器开路
0xE5	229	温度传感器短路
0xE6	230	冰满
0xE7	231	传感器正常
0xE8	232	缺水

本地食谱表

本地食谱属性定义如下。

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0000	0	无
0x0001	1	宝宝糊

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0002	2	五谷浆
0x0003	3	养生粥
0x0004	4	燕窝
0x0005	5	果蔬汁
0x0006	6	酱料
0x0007	7	粉碎
0x0008	8	清洗
0x0009	9	粉碎
0x000A	10	浓汤
0x000B	11	宝宝粥
0x000C	12	保温
0x000D	13	自定义模式
0x000E	14	蒸水蛋
0x000F	15	煮水饺
0x0010	16	煲汤
0x0011	17	煮稀饭
0x0012	18	煮米饭
0x0013	19	烤蛋糕
0x0014	20	烧排骨
0x0015	21	烤红薯
0x0016	22	烤鸡翅
0x0017	23	清蒸鱼
0x0018	24	蒸馒头、包子
0x0019	25	自动翻热
0x001A	26	宝宝牛奶
0x001B	27	酸奶

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x001C	28	营养解冻
0x001D	29	爆米花
0x001E	30	热牛奶
0x001F	31	快速烹调
0x0020	32	低温蒸
0x0021	33	元气蒸
0x0022	34	慢蒸
0x0023	35	健康蒸
0x0024	36	消毒
0x0025	37	健康炸
0x0026	38	有预热烘烤
0x0027	39	无预热烘烤
0x0028	40	发酵
0x0029	41	有预热蒸汽烘烤
0x002A	42	无预热蒸汽烘烤
0x002B	43	炖
0x002C	44	蒸冷冻食品
0x002D	45	蒸时蔬
0x002E	46	蒸鸡
0x002F	47	宝宝薯丁
0x0030	48	披萨
0x0031	49	叉烧
0x0032	50	煎鱼
0x0033	51	炸虾
0x0034	52	法棍面包
0x0035	53	水浴奶酪蛋糕

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0036	54	烤肉
0x0037	55	烤鱼
0x0038	56	土豆
0x0039	57	面包片
0x003A	58	欧式面包
0x003B	59	法式面包
0x003C	60	英式面包
0x003D	61	无筋面包
0x003E	62	快速面包
0x003F	63	西式蛋糕
0x0040	64	米饭面包
0x0041	65	玉米面包
0x0042	66	紫米面包
0x0043	67	玄米面包
0x0044	68	八宝米粥
0x0045	69	年糕
0x0046	70	生面团
0x0047	71	低温发酵
0x0048	72	葡萄酒
0x0049	73	果酱
0x004A	74	自动翻炒
0x004B	75	自定义和面
0x004C	76	烘烤
0x004D	77	欧式甜点
0x004E	78	解冻
0x004F	79	肉松

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0050	80	和面
0x0051	81	蛋糕
0x0052	82	火锅
0x0053	83	温泉蛋
0x0054	84	牛排
0x0055	85	柴火饭
0x0056	86	快速饭
0x0057	87	网红饭
0x0058	88	寿司饭
0x0059	89	石锅饭
0x005A	90	沸腾
0x005B	91	绿茶
0x005C	92	花茶
0x005D	93	奶粉
0x005E	94	红茶
0x005F	95	罗汉果茶
0x0060	96	咖啡
0x0061	97	除氯
0x0062	98	杂粮饭
0x0063	99	米粥
0x0064	100	杂粮粥
0x0065	101	鸡鸭肉
0x0066	102	牛羊肉
0x0067	103	豆蹄筋
0x0068	104	汤类
0x0069	105	开盖煮

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x006A	106	营养蒸
0x006B	107	无水焗
0x006C	108	再加热
0x006D	109	破壁豆浆
0x006E	110	坚果露
0x006F	111	快速水洗
0x0070	112	搅拌一档
0x0071	113	搅拌二档
0x0072	114	搅拌三档
0x0073	115	搅拌四档
0x0074	116	搅拌五档
0x0075	117	高温蒸洗
0x0076	118	果仁露
0x0077	119	轻松洗
0x0078	120	米糊
0x0079	121	熟料打
0x007A	122	果汁
0x007B	123	精力汤
0x007C	124	五谷
0x007D	125	纯豆
0x007E	126	干湿豆
0x007F	127	果茶
0x0080	128	快速豆浆
0x0081	129	杏仁露
0x0082	130	奶茶
0x0083	131	干豆

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0084	132	排浆
0x0085	133	洛神饮
0x0086	134	湿豆
0x0087	135	破壁果汁
0x0088	136	猴菇浆
0x0089	137	薏米露
0x008A	138	姜枣露
0x008B	139	辅食
0x008C	140	藜麦浆
0x008D	141	标准煮
0x008E	142	精华煮
0x008F	143	锅巴饭
0x0090	144	蒸煮
0x0091	145	发芽饭
0x0092	146	煲仔饭
0x0093	147	快煮粥
0x0094	148	糙米饭
0x0095	149	宝宝果泥
0x0096	150	除味
0x0097	151	微波杀菌
0x0098	152	光波杀菌
0x0099	153	蒸汽清洁
0x009A	154	微波
0x009B	155	光波组合
0x009C	156	点动
0x009D	157	奶昔

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x009E	158	玉米汁
0x009F	159	煮茶
0x00A0	160	破壁
0x00A1	161	鱼汤
0x00A2	162	研磨
0x00A3	163	芝麻糊
0x00A4	164	冰沙
0x00A5	165	香浓豆浆
0x00A6	166	元气蒸包
0x00A7	167	苹果汁
0x00A8	168	黄瓜汁
0x00A9	169	猕猴桃汁
0x00AA	170	芒果汁
0x00AB	171	橙汁
0x00AC	172	火龙果汁
0x00AD	173	养生饭
0x00AE	174	温调
0x00AF	175	热调
0x00B0	176	乌龙茶
0x00B1	177	白茶
0x00B2	178	黄茶
0x00B3	179	黑茶
0x00B4	180	泡面
0x00B5	181	柠檬水
0x00B6	182	铁观音
0x00B7	183	煮岩茶

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x00B8	184	玛卡
0x00B9	185	金线莲
0x00BA	186	枸杞
0x00BB	187	石斛
0x00BC	188	绿豆
0x00BD	189	红豆
0x00BE	190	白屋燕盏
0x00BF	191	洞燕盏
0x00C0	192	血燕
0x00C1	193	燕条
0x00C2	194	燕角
0x00C3	195	晨间饮水
0x00C4	196	温开水
0x00C5	197	普洱茶
0x00C6	198	脱脂奶粉
0x00C7	199	全脂奶粉
0x00C8	200	中老年奶粉
0x00C9	201	浅烘焙咖啡
0x00CA	202	中烘焙咖啡
0x00CB	203	深烘焙咖啡
0x00CC	204	速溶咖啡
0x00CD	205	蛋挞
0x00CE	206	戚风蛋糕
0x00CF	207	烤鸡
0x00D0	208	面包
0x00D1	209	蛋糕卷

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x00D2	210	曲奇
0x00D3	211	龙虾
0x00D4	212	松饼
0x00D5	213	章鱼小丸子
0x00D6	214	关东煮
0x00D7	215	吐司
0x00D8	216	早餐包
0x00D9	217	烤串
0x00DA	218	空气炸
0x00DB	219	世卫模式
0x00DC	220	轻养
0x00DD	221	杀菌
0x00DE	222	泡茶
0x00DF	223	蜂蜜
0x00E0	224	绵粥
0x00E1	225	爆炒
0x00E2	226	健康炒
0x00E3	227	煎烙
0x00E4	228	恒温烹饪
0x00E5	229	热菜
0x00E6	230	汤粥
0x00E7	231	纯豆浆
0x00E8	232	热烘杀菌
0x00E9	233	五谷粥
0x00EA	234	花果茶
0x00EB	235	杞菊茶

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x00EC	236	营养炖
0x00ED	237	温奶
0x00EE	238	调奶
0x00EF	239	烧水
0x00F0	240	养生汤
0x00F1	241	煮蛋
0x00F2	242	药膳
0x00F3	243	搅拌
0x00F4	244	营养汤
0x00F5	245	滋补粥
0x00F6	246	清炒
0x00F7	247	全烤
0x00F8	248	高温蒸
0x00F9	249	上烤
0x00FA	250	下烤
0x00FB	251	蒸烤
0x00FC	252	纯蒸
0x00FD	253	嫩烤
0x00FE	254	烘焙
0x00FF	255	预热
0x0100	256	甜品
0x0101	257	滋补糊
0x0102	258	上下烤
0x0103	259	上下烤大温幅
0x0104	260	上烤大温幅
0x0105	261	清蒸鲜鱼

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0106	262	蒸白米饭
0x0107	263	滑嫩蒸蛋
0x0108	264	素蒸时蔬
0x0109	265	炖排骨汤
0x010A	266	炖燕窝羹
0x010B	267	五谷丰登
0x010C	268	粉丝蒸虾
0x010D	269	粉蒸排骨
0x010E	270	嫩烤乳鸽
0x010F	271	蜜汁烤翅
0x0110	272	香烤牛排
0x0111	273	蜜汁烤鸡
0x0112	274	脆烤披萨
0x0113	275	风味蛋挞
0x0114	276	奶酪焗饭
0x0115	277	香烤面包
0x0116	278	快手吐司
0x0117	279	薯条
0x0118	280	木瓜炖牛奶
0x0119	281	蒸玉米
0x011A	282	清蒸大闸蟹
0x011B	283	鸡蛋羹
0x011C	284	满堂彩
0x011D	285	蒸土豆
0x011E	286	海绵蛋糕
0x011F	287	豆豉蒜茸蒸扇贝

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0120	288	吐司面包
0x0121	289	汉堡
0x0122	290	烤椰蓉小面包
0x0123	291	烤肉串
0x0124	292	烤蛋糕卷
0x0125	293	烤马铃薯
0x0126	294	烤披萨
0x0127	295	红枣糕
0x0128	296	玛德琳
0x0129	297	马芬
0x012A	298	剁椒鱼头
0x012B	299	蒸鱼片
0x012C	300	木瓜蒸饭
0x012D	301	清蒸丝瓜虾仁
0x012E	302	香菇蒸鸡
0x012F	303	清蒸黄花鱼
0x0130	304	枸杞蒸甲鱼
0x0131	305	豆豉蒸排骨
0x0132	306	清蒸鲍鱼
0x0133	307	银耳莲子羹
0x0134	308	蒸水饺
0x0135	309	乌鸡白凤汤
0x0136	310	蒸点心
0x0137	311	香菇肉饼
0x0138	312	百合炖雪梨
0x0139	313	粉蒸肉

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x013A	314	香芋扣肉
0x013B	315	清蒸凤爪
0x013C	316	清蒸藕丸
0x013D	317	双皮奶
0x013E	318	蒸烤香肠
0x013F	319	蒸烤五花肉
0x0140	320	蒸烤鸡腿
0x0141	321	蒸烤鸭
0x0142	322	蒸烤虾
0x0143	323	蒸烤猪脚
0x0144	324	蒸烤鸡翅
0x0145	325	蒸烤全鸡
0x0146	326	蒸烤鸽子
0x0147	327	照烧鸡块
0x0148	328	蒸烤生蚝
0x0149	329	烤茄子
0x014A	330	烤三文鱼
0x014B	331	烤玉米
0x014C	332	海鲜焗饭
0x014D	333	蒸烤鱿鱼
0x014E	334	烤香蕉
0x014F	335	烤金针菇
0x0150	336	烤排骨
0x0151	337	烤香菇
0x0152	338	烤燕麦
0x0153	339	烤羊肉串

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0154	340	粉蒸牛肉
0x0155	341	糯米排骨
0x0156	342	腊味合蒸
0x0157	343	烤牛排
0x0158	344	烤猪排
0x0159	345	葱香肉松包
0x015A	346	牛奶面包
0x015B	347	蒜蓉蒸丝瓜
0x015C	348	肉沫蔬菜
0x015D	349	蒜蓉蒸南瓜
0x015E	350	香蒸油豆腐
0x015F	351	蒸馒头
0x0160	352	清蒸排骨
0x0161	353	菊花
0x0162	354	玫瑰
0x0163	355	银耳
0x0164	356	婴儿水
0x0165	357	煮面
0x0166	358	降糖饭
0x0167	359	潮汕粥
0x0168	360	老火汤
0x0169	361	无水焗鸡
0x016A	362	排骨、肉
0x016B	363	常温
0x016C	364	煲粥
0x016D	365	煮酒

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x016E	366	热奶
0x016F	367	凉茶
0x0170	368	水蛋
0x0171	369	萝卜泥
0x0172	370	苹果泥
0x0173	371	香蕉泥
0x0174	372	土豆牛肉泥
0x0175	373	菠萝泥
0x0176	374	牛奶香蕉泥
0x0177	375	海鲜虾泥
0x0178	376	田园土豆泥
0x0179	377	菠菜南瓜泥
0x017A	378	淮山玉米泥
0x017B	379	冬瓜肉泥粥
0x017C	380	猪肝青菜粥
0x017D	381	牛肉燕麦粥
0x017E	382	紫薯吐司糕
0x017F	383	笈瓜鸡蛋饼
0x0180	384	清蒸山药糕
0x0181	385	白菜肉饺子
0x0182	386	果泥
0x0183	387	菜泥
0x0184	388	鱼泥
0x0185	389	瓜泥
0x0186	390	肉泥
0x0187	391	桃胶

Receipt Number（十六进制）	Receipt Number（十进制）	食谱名称
0x0188	392	补汤
0x0189	393	元气浆
0x018A	394	甄果露
0x018B	395	小米粥
0x018C	396	五谷饭
0x018D	397	豆豆饭
0x018E	398	热饭
0x018F	399	养生糊
0x0190	400	方便面
0x0191	401	酸菜鱼
0x0192	402	红烧豆腐
0x0193	403	烤虾
0x0194	404	烤韭菜
0x0195	405	宝宝面条
0x0196	406	南瓜泥
0x0197	407	文火
0x01BD	445	餐具杀菌

设备组播地址

- 特殊组播地址

组播地址分配	组播地址
Mesh网关接收地址	0xF000
OTA组播地址	0xF100
所有产品组播地址	0xCFFF

> **说明** 在天猫精灵生态项目中，Mesh网关包括天猫精灵音箱与天猫精灵App。在自有品牌项目中，Mesh网关包括自有品牌项目Mesh网关产品与云智能App。

- 产品组播地址

具体产品	组播地址
灯泡	0xC000
开关	0xC001
插座	0xC002
窗帘	0xC003
人体秤	0xC004
无线按钮	0xC005
门磁	0xC006
风扇	0xC007
手环	0xC008
万能红外遥控器	0xC00E
取暖器	0xC100
空调扇	0xC101
加湿器	0xC102
电热毯	0xC103
扫地机器人	0xC104
抽湿机、除湿器	0xC105
空气净化器	0xC106
家用新风机	0xC107
洗衣机	0xC110
烘干机	0xC111
冰箱	0xC112
电热水器	0xC113
空调	0xC114
洗碗机	0xC115
电视	0xC116
油烟机	0xC117

具体产品	组播地址
厨宝	0xC118
燃气热水器	0xC119
净水器	0xC120
饮水机	0xC121
电烤箱	0xC122
养生壶	0xC123
电热炉	0xC124
电蒸炉、电蒸锅	0xC125
豆浆机	0xC126
搅拌、料理机、破壁机	0xC127
电热、火锅	0xC128
电饭煲	0xC129
电压力锅	0xC12A
酸奶机	0xC12B
消毒柜	0xC12C
酒柜	0xC12D
冰吧	0xC12E
足浴器	0xC130
按摩枕	0xC131
按摩足疗机	0xC132
按摩椅、沙发	0xC133
家用氧吧	0xC134
按摩仪	0xC135
颈腰靠垫	0xC136
按摩坐垫	0xC137
煎药壶	0xC138

具体产品	组播地址
电水壶	0xC139
电炖锅	0xC13A
电热杯	0xC13B
煮茶器	0xC13C
电茶炉	0xC13D
文火炉	0xC13E
煮粥锅	0xC13F
调光面板	0xC140
空调控制面板	0xC141
地暖控制面板	0xC142
门窗控制器	0xC143
家用场景面板	0xC144
电源控制器	0xC145
人体传感器	0xC146
智能监控摄像机	0xC147
红外探测器	0xC148
智能安防报警器材	0xC14A
家用气体检测报警器、气体检测仪	0xC14B
家用单机烟感探测器	0xC14C
门镜、猫眼	0xC14D
电子门锁	0xC14E
灭蚊器	0xC14F
驱蚊器	0xC151
跑步机	0xC160
动感单车	0xC161
椭圆机	0xC162

具体产品	组播地址
划船机	0xC163
走步机	0xC164
体重秤、健康秤	0xC165
跳绳	0xC166
哑铃	0xC167
移动电源	0xC170
儿童玩具表、智能手表	0xC180
暖奶器、加热器	0xC190
奶瓶消毒器、消毒锅	0xC191
电动奶粉搅拌器	0xC192
BB煲、电粥锅	0xC193
宝宝料理机、食物搅拌器	0xC194
调奶器	0xC195
电动吸奶器	0xC196
辅食机	0xC197
血压计（电子血压计）	0xC1A0
血糖用品	0xC1A1
体温计类	0xC1A2
蒸脸器	0xC1A3
温湿度传感器	0xC1A4
取暖桌	0xC1A5
晾衣架	0xC150
智能垃圾桶	0xC1A7
点读笔	0xC1A8
猫砂盆	0xC1A9
浴霸	0xC1AA

具体产品	组播地址
制冰机	0xC1AB
电炸锅	0xC1AC
烟雾传感器	0xC1AD
水浸传感器	0xC1AE
光线传感器	0xC1AF
燃气传感器	0XC1B0
暖菜板	0xC1B1
艾灸仪	0xC1B2
暖宫带	0xC1B3
智能床	0xC1B4
恒温收纳箱	0xC1B5
恒温口红盒	0xC1B6
电动牙刷	0xC1B7
智能马桶	0XC1B8
炒菜机	0XC1B9
消毒灯	0XC1BA
果蔬消毒机	0XC1BB
电蒸锅	0XC1BC
咖啡机	0XC1BD
香薰机	0xC1BE
吹风机	0xC1C1
毛巾架	0xC1C2
魔法棒	0xC1C3
甩脂机	0xC1C4
智能水杯	0xC1C5
宠物喂食机	0xC1C6

具体产品	组播地址
智能跳绳	0xC1C7
筋膜枪	0xC1C8
牙刷架	0xC1C9
窗	0xC1CA
温控器	0xC1CB
洗手机	0xC1CC
新风浴霸	0xC1CD
拖地机	0xC1CE
踏步机	0xC1CF
椭圆机	0xC1D0
按摩枕	0xC1D1
药盒	0xC1D2
首饰盒	0xC1D3
便当盒	0xC1D4
原汁机	0xC1D5
眼镜盒、隐形眼镜盒	0xC1D6
洗牙器	0xC1D7
直发器	0xC1D8
卷发器	0xC1D9
马桶盖	0xC1DA

6.7.2. 蓝牙设备属性表

本文当前蓝牙智能家居设备的整体属性定义，供设备开发时查阅。

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	错误码	0x0000	uint8 Error_Code	-	-	Error Code：错误码，请参看4.4错误码定义

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
系统属性	版本信息	0xFF01	uint32 version_number	-	-	version_number：与OTA的版本一致
	Device Feature	0xFF02	uint16 device_feature	-	-	与Composition Data Page 0中的feature一致，最高位bit15往下是阿里自定的一些feature，bit0： Relay；bit1： Proxy；bit2： Friend；bit3： Low Power Node；bit14： Low Power and Letency；bit15： Multi-Attr RW support；bit4~12：RFU置1表示该功能打开，置0表示该功能关闭。
	Flash总容量	0xFF03	uint32 total_flash_size	Byte	-	设备Flash的总容量
	Flash已用容量	0xFF04	uint32 used_flash_size	Byte	-	设备Flash的已用容量
	Flash空余容量	0xFF05	uint32 free_flash_size	Byte	-	设备Flash的可用容量
	工程模式	0xFF06	int8 Engineer_Mode	-	-	1：工厂测试模式，各厂商自己定义工厂生产使用模式；2：进入debug模式，设备端log通过其他通道上传到云端；3：重启设备；4：清除设置数据；5：恢复出厂设置；6：用户诊断模式
	ID2ID	0xFF07	uint8 idType uint8 len uint8 id2ID[len]	-	-	IdType： ID² ID是否进行16进制转换；0 - 保持原有格式；1 - 进行16进制转换 len：根据idType取值可为24或12 id2ID[len]：设备ID2 id。
	ID2AuthRand	0xFF08	uint8 authType uint8 len uint8 authRand[len]	-	-	authType： ID²认证方式 0 - 挑战字，默认 1 - 时间戳 2 - TOTP动态密码 len： ID²设备认证码的长度 authCode：ID²设备认证码
	ID2AuthCode	0xFF09	uint8 authType uint8 len uint8 authCode[len]	-	-	authType：ID²认证方式 0 - 挑战字，默认 1 - 时间戳 2 - TOTP动态密码 len：ID²设备认证码的长度 authCode：ID²设备认证码

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	ID2KeyAndPass	0xFF0A	uint8 keyType uint8 dataLen uint8 cipherData[dataLen] uint8 pwdLen uint8 authPwd[pwdLen]	-	-	keyType： 会话密钥的类型 0 - aes_128 1 - aes_192 2 - aes_256 dataLen：ID²加密的会话密钥长度 cipherData：ID²加密的会话密钥数据 pwdLen：服务认证口令长度 authPwd： 服务认证口令数据
	工作状态	0xF001	uint8 workstatus	-	-	0：停止 1：开始/继续 2：暂停 3：待机 4：烹饪 5：预约 6：保温 7：感应中 8：感应暂停中 9：门开 10：完成 11：搅拌1 12：搅拌2 13：醒面1 14：醒面2 15：搅拌3 16：醒面3 17：发酵1 18：发酵2 19：发酵3 20：烘烤+搅拌 21：搅拌+醒面 22：预热 23：报警 24：休眠 25：加热 26：错误 27：wifi设置 28：关机 29：工作 30：热喷中 31：冷喷中 32：温喷中 33：敷面膜开始等待 34：敷面膜中 35：敷面膜时间到 36：上电 37：运行 38：空闲 39：加水 40：出水 41：盒盖不到位 42：制浆结束 43：保温结束 44：吸油烟 45：热洗清洁 46：热洗风干 47：面板清洁 48：制水 49：冲洗 50：缺水 51：满水 52：正常 53：翻面提示 54：保护状态 55：搅拌 56：粉碎 57：混合 58：调和 61：回充 62：充电 63：充电完成 64：电量不足 65：制浆中 66：延时关机中 67：自动热烘中 68：自动清洗中 69：小火慢炖 70：中火熬煮 71：大火延沸 72：准备 73：蒸汽 74：除臭 75：烘干 76：杀菌 77：结束 78：脱水 79：漂洗 80：主洗 81：预洗 82：称重 83：保留 84：满冰 85：制冰 86：脱冰 87：保压 88：猫入厕 89：蒸食中 90：蒸食完成 91：搅拌中 92：搅拌完成 93：消毒中 94：消毒完成 95：暖奶中 96：暖奶完成 97：变频 98：到达上限 99：到达下限 100： 异常

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
通用属性	User ID	0xF002	uint8 userID	-	-	当前用户ID
	设备名	0xF003	uint8 len uint8 device_name[len]	-	-	设备名称
	场景模式	0xF004	uint16 mode_number	-	-	设备的场景模式，与scene model关联
	模式剩余时间	0xF005	uint16 modeRemainingTime	分钟	-	当前模式剩余时间，0xFFFF为无穷大
	开关计划（该命令需要设备具备RTC电路和对时功能）	0xF008	uint8 weekday uint16 time uint8 OnOff			以周为周期来定时开关设备 weekday：bit0~bit6分别代表周日到周六，置1为启用，置0为禁用 time：开关的时间相对于00：00的分钟数 OnOff：开启/关闭
	事件触发	0xF009	uint8 Event uint8 Event para[]	-	-	设备发生的事件，event参见事件表，如事件带参数，则后续跟参数。
	事件清除	0xF019	uint8 Event uint8 Event para[]	-	-	如设备发生的事件引起了设备状态的改变，在设备的状态消失后，设备需要清除之前上报的事件。 例如当设备电池电量过低，设备上报低电量事件，并处于低电量状态；当用户更换电池后，设备退出低电量状态，应当上报清除低电量事件。
	信号强度	0xF00A	int8 RSSI	dbm	-	设备信号强度
	调高/调低某属性	0xF00B	uint16 Attr_Type int8 Attr_para_step	-	-	Attr_para_step为正表示增加，为负表示减少，物理量步长不带小数点
	区域数量	0xF00C	uint8 Element_num	-	-	设备具有几个区域
	属性切换	0xF00D	uint8 Attr_Type uint8 switch	-	-	设备具有的Attr_Type属性在他的取值范围内进行循环切换，Attr_Type为枚举或布尔型

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	受控设备地址	0xF00E	uint16 Remote_Address	-	-	该设备为遥控设备，该属性为这个遥控设备所控制的设备的地址，可为unicast address或者group address
	周边设备信号强度	0xF00F	uint16Device_Address int8 RSSI	dbm	-	周边设备信号强度
	定时设置某属性	0xF010	uint8 index uint32 unix_time uint16 attr_type uint8 attr_para[]	-	-	该功能参数详细说明请参照定时功能文档
	周期定时设置某属性	0xF011	uint8 index uint16 24h_timer uint8 schedule uint16 attr_type uint8 attr_para[]	-	-	该功能参数详细说明请参照定时功能文档
	删除定时	0xF012	uint8 index	-	-	删除索引为index的定时
	请求更新定时	0xF013	uint8 index	-	-	请求更新索引为index的定时
	对时设置	0xF01D	uint16 period_time uint8 retry_delay uint8 retry_times	分钟、分钟、次		每隔period_time进行一次对时请求，如果设备则过retry_delay后再次请求对时，最多重复请求retry_times（包含第一次请求）
	时区	0xF01E	int8 time_zone			取值-12～12
	Unix时间	0xF01F	uint32 unix_timer	秒	-	标准Unix时间
	断电记忆功能	0xF021	bool poweroffMemory	-	-	0：关闭断电记忆 1：打开断电记忆
	开关	0x0100	bool OnOff	-	-	设备开关状态，与generic onoff绑定
	日期	0x0101	uint8 date[4]	-	-	Date[0~1]：年 Date[2]： 月 Date[3]： 日
	时间	0x0102	uint8 time[3]	-	-	Time[0]：时 Time[1]： 分 Time[2]： 秒
	Unix时间戳	0x0103	uint8 unix_time[6]	毫秒	-	unix时间戳
	电量	0x0104	uint8 power_percent	百分比	-	电池电量百分比

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	电压	0x0105	uint16 voltage	伏	0.01	设备电压
	电流	0x0106	uint16 current	安培	0.001	设备电流
	功率	0x0107	uint16 power	瓦特	0.1	设备功率
	功耗	0x0108	uint32 power_consumption	千瓦时	0.01	设备功耗
	音量	0x0109	uint8 volume	-	-	音量
	风速档位	0x010A	uint8 level	-	-	取值1~100 风扇/空调：风力档位，针对某些支持模糊档位的设备，定义如下： 150：静音档 151：低档 152：中低档 153：中档 154：中高档 155：高档 156：超强档 157：凉风档 158：强劲档 240：爆炒档 241：自动巡航 255：自动档
	设备显示计量单位体系	0x010B	uint8 unit	-	-	0：公制（公斤，米，摄氏度） 1：英制（磅，英尺，华氏度） 2：中国单位（市斤，尺）
	目标温度	0x010C	uint16 target_temperature	开尔文	0.01	目标温度。0开尔文=零下273.15摄氏度
	当前温度	0x010D	uint16 present_temperature	开尔文	0.01	当前温度
	目标湿度	0x010E	uint16 target_humidity	百分比	0.01	目标相对湿度
	当前湿度	0x010F	uint16 present_humidity	百分比	1	当前相对湿度
	前后位置	0x0110	uint8 position_FB	-	-	设备可前后移动部位的相对位置，100为最前，0为最后
	左右位置	0x0111	uint8 position_LR	-	-	设备可左右移动部位的相对位置，100为最左，0为最右

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	上下位置	0x0112	uint8 position_UD	-	-	设备可上下移动部位的相对位置，100为最上，0为最下
	前行程	0x0113	uint8 position_Forward_limit	-	-	设备可前后移动部位的前移限制点
	后行程	0x0114	uint8 position_Backward_limit	-	-	设备可前后移动部位的后移限制点
	左行程	0x0115	uint8 position_Left_limit	-	-	设备可左右移动部位的左移限制点
	右行程	0x0116	uint8 position_Right_limit	-	-	设备可左右移动部位的右移限制点
	上行程	0x0117	uint8 position_Up_limit	-	-	设备可上下移动部位的上移限制点
	下行程	0x0118	uint8 position_Down_limit	-	-	设备可上下移动部位的下移限制点
	上下角度	0x0119	uint16 angle_UD	-	-	设备可上下旋转部位角度
	左右角度	0x011A	uint16 angle_LR	-	-	设备可左右旋转部位角度
	上下角度上限	0x011B	uint16 angle_Up_limit	-	-	设备可上下旋转部位向上旋转极限角度
	上下角度下限	0x011C	uint16 angle_Down_limit	-	-	设备可上下旋转部位向下旋转极限角度
	左右角度左限	0x011D	uint16 angle_Left_limit	-	-	设备可左右旋转部位向左旋转极限角度
	左右角度右限	0x011E	uint16 angle_Right_limit	-	-	设备可左右旋转部位向右旋转极限角度
	方位校准	0x011F	uint8 Position_Limit_Area	-	-	0：上行程 1：下行程 2：左行程 3：右行程 4：前行程 5：后行程 6：角度上限左 7：角度上限右 8：角度上限上 9：角度上限下
	水位	0x0120	uint8 water_level	-	-	设备水位

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	亮度	0x0121	uint16 lightness_level	-	-	设备亮度等级，与Lightness Model值一致
	色温	0x0122	uint16 color_temperature	开尔文	-	设备色温值，与Light CTL Model值一致
	颜色	0x0123	uint16 HSL[3]	-	-	设备颜色，与Light HSL Model值一致 Color[0]= Lightness Color[1] = Hue Color[2] = Saturation
	第二开关	0x0124	uint8 Secondary_OnOff	-	-	特殊品类的第二功能开关，电暖气：高温管开关；电热毯：左边区域开关
	第三开关	0x0125	uint8 Third_OnOff			特殊品类的第二功能开关，电暖气：低温管开关；电热毯：右边区域开关
	第二档位	0x0128	uint8 Secondary_Level	-	-	特殊品类的第二功能档位，电暖气：高温管档位；电热毯：左边区域档位
	第三档位	0x0129	uint8 Thrid_Level	-	-	特殊品类的第三档位电暖气：低温管档位；电热毯：右边区域档位
	耗材使用时间	0x012A	uint32 usedTime	秒	-	设备耗材使用时间
	耗材剩余时间	0x012B	uint32 remainingTime	秒	-	设备耗材剩余时间
	耗材剩余寿命	0x012C	uint16 remainingPercentage	百分比	0.01	设备耗材使用剩余百分比
	运行时间	0x012D	uint32 runTime	秒	-	设备本次运行时间
	累计运行时间	0x012E	uint32 totalRunningTime	秒	-	设备累计运行时间
	剩余时间	0x012F	uint32 lefttime	秒	-	
	预计可使用时间	0x0130	uint32 predictTime	秒	-	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	充电剩余时间	0x0131	uint16 chargeRemainingTime	秒	-	
	主板温度	0x0132	uint16 boardTemp	开尔文	0.01	
	电机温度	0x0133	uint16 motorTemp	开尔文	0.01	
	电池温度	0x0134	uint16 batteryTemp	开尔文	0.01	
	启动开关	0x0135	uint8 workSwitch	-	-	0：关闭 1：打开 2：暂停
	语言	0x0136	uint8 language	-	-	0：中文 1：英语
	累计使用时间	0x0137	uint32 totalUsedTime	秒	-	设备累计使用的时间
	质量	0x0138	uint8 mass	克	-	例如食物重量
	保养时间	0x0139	uint16 serviceTime	日	-	
	混水温度	0x013A	uint8 mixTemp	-	-	
	剩余水量	0x013B	uint16 surplusWater	升	-	
	总耗电量	0x013C	uint32 TotalPowerconsumption	度	-	
	出水温度	0x013D	uint8 waterOutTemperature	-	-	
	力度	0x013E	uint8 strength	-	-	
	按摩时间	0x013F	uint8 massagetime	-	-	
	语音开关	0x0140	uint8 voiceswitch	-	-	0：关闭 1：打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
通用物理量	清扫面积	0x0141	uint8 cleanarea	-	-	
	蜂鸣开关	0x0142	uint8 buzzer_switch	-	-	0：关闭 1：打开
	漂洗次数	0x0143	uint8 rinshtimes	-	-	
	烘干时间	0x0144	uint16 drytime	-	-	
	脱水时间	0x0145	uint16 spintime	-	-	
	漂洗时间	0x0146	uint16 rinshtime	-	-	
	洗涤时间	0x0147	uint16 washtime	-	-	
	浸泡时间	0x0148	uint16 soaktime	-	-	
	保温时间	0x0149	uint16 keep_warm_time	分钟	0.1	
	本次用气量	0x014A	uint32 gasonce	-	-	
	甲烷值	0x014B	uint32 methanenumber	-	-	
	预约开关	0x014C	uint8 timingswitch	-	-	0：关闭 1：打开
	优先权	0x014D	uint8 priority	-	-	0：空闲 1：设备 2：手机
	耗水量	0x014E	uint32 waterconsumption	-	-	
	回差温度	0x014F	uint8 differencetemperature	-	-	
	水流量	0x0150	uint8 waterflow	-	-	
	总用气量	0x0151	uint32 totalgas	-	-	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	进水温度	0x0152	uint8 waterintemperature	-	-	
	总用水量	0x0153	uint32 totalwater	升	0.01	
	下管加热时间	0x0154	uint16 downtubebakingtime	分钟	-	
	上管加热时间	0x0155	uint16 uptubebakingtime	分钟	-	
	上管加热温度	0x0156	uint16 uptubebakingtemp	摄氏度	-	
	下管加热温度	0x0157	uint16 downtubebakingtemp	摄氏度	-	
	总洗涤次数	0x0158	uint16 totalwashtimes	-	-	
	保管剩余时间	0x0159	uint16 keep_remaining_time	-	-	
	搅拌速度	0x015A	uint8 stirringspeed	-	-	
	预设保温温度	0x015B	uint16 holding_temp	-	-	
	食物重量	0x015C	uint16 food_weight	-	-	
	录音开关	0x015D	uint8 recording_switch	-	-	0：关闭 1：打开
	扫射开关	0x015E	uint8 gun_switch	-	-	0：关闭 1：打开
	冒烟开关	0x015F	uint8 smoke_switch	-	-	0：关闭 1：打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	发射开关	0x0160	uint8 launcher_switch	-	-	0：关闭 1：打开
	跑步结束时间	0x0161	uint8 end_run_time[3]	-	-	Time[0]：时 Time[1]： 分 Time[2]： 秒
	跑台软硬度	0x0162	uint8 running_plat_form_hardness	-	-	
	目标速度	0x0163	uint16 target_speed	km/H	0.01	
	雾量	0x0164	uint16 fog	-	-	
	目标容量	0x0165	uint8 target_capacity	-	-	1：一百五十毫升 2：二百毫升 3：四百毫升 4：六百毫升 5：八百毫升 6：一千毫升 7：一千二毫升 8：三百毫升 9：五百毫升
	单位类型	0x0166	uint8 unit_type	-	-	0：公制 2：英制 3：中国单位
	净水水量	0x0167	uint16 water_yield	-	-	
	取水水量	0x0168	uint8 get_water_yield	-	-	1：小杯 2：中杯 3：大杯
	取水水温	0x0169	uint8 get_water_temp	-	-	1：冷水 2：常温水 3：热水 4：沸水
	取水开关	0x016A	uint8 get_water_switch	-	-	0：关闭 1：打开
	净水总量	0x016B	uint32 purefity_water	-	-	
	纯水水量	0x016C	uint32 pure_water	-	-	
	原水水质	0x016D	uint16 raw_water_TDS	-	-	
	滤芯寿命五	0x016E	uint8 filter_lefttime_five	百分比	0.01	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	滤芯寿命四	0x016F	uint8 filter_lefttime_four	百分比	0.01	
	滤芯寿命三	0x0170	uint8 filter_lefttime_three	百分比	0.01	
	滤芯寿命二	0x0171	uint8 filter_lefttime_two	百分比	0.01	
	滤芯寿命一	0x0172	uint8 filter_lefttime_one	百分比	0.01	
	空气质量等级	0x0173	uint8 air_quality_grade	-	-	1：优 2：良 3：轻度污染 4：中度污染 5：重度勿扰
	加热档位	0x0174	uint8 heaterPower	档	-	暖气加热档位
	温区	0x0175	uint8 temperatureArea	-	-	
	雾量档位	0x0176	uint8 fogLevel	档	-	
	上管当前温度	0x0177	uint16 UPTubeCurrenttemp	摄氏度	-	
	下管当前温度	0x0178	uint16 DownTubeCurrenttemp	摄氏度	-	
	节能模式温度上限	0x0179	uint8 temperatureUpLimit	摄氏度	-	
	节能模式温度下限	0x017A	uint8 temperatureDownLimit	摄氏度	-	
	开关左	0x017B	uint8 leftPowerstate			
	开关右	0x017C	uint8 rightPowerstate			
	剩余时间左	0x017D	uint32 leftLefttime			

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	剩余时间左	0x017E	uint32 rightLefttime			
	工作模式左	0x017F	uint16 leftMode			
	工作模式右	0x0180	uint16 rightMode			
	目标温度左	0x0181	uint16 leftTargettemperature	摄氏度	0.1	
	目标温度右	0x0182	uint16 rightTargettemperature	摄氏度	0.1	
	温度左	0x0183	uint16 leftTemperature	摄氏度	0.1	
	温度右	0x0184	uint16 rightTemperature	摄氏度	0.1	
	左取暖档位	0x0185	uint8 leftLevel	档	-	左加热档位
	右取暖档位	0x0186	uint8 rightLevel	档	-	右加热档位
	前取暖档位	0x0187	uint8 frontLevel	档	-	前加热档位
	后取暖档位	0x0188	uint8 backLevel	档	-	后加热档位
	保暖档位	0x0189	uint8 warmLevel	档	-	保暖档位
	烹饪档位	0x018A	uint8 cookLevel	档	-	烹饪档位
	烹饪开关	0x018B	uint8 cookOnOff	-	-	0：关闭 1：开启
	档位	0x018C	uint8 gear	-	-	档位
	保温时间左	0x018D	uint16 leftKeepWarmTime	分钟	0.1	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	保温时间右	0x018E	uint16 rightKeepWarmTime	分钟	0.1	
	睡眠时间	0x018F	uint16 sleepingTime	分钟	0.1	
	睡眠时间左	0x0190	uint16 leftSleepingTime	分钟	0.1	
	睡眠时间右	0x0191	uint16 rightSleepingTime	分钟	0.1	
	升温时间	0x0192	uint16 heatUpTime	分钟	0.1	
	升温时间左	0x0193	uint16 leftHeatUpTime	分钟	0.1	
	升温时间右	0x0194	uint16 rightHeatUpTime	分钟	0.1	
	环境温度差	0x0195	uint16 envTemperatureDiff	摄氏度	0.1	
	容量	0x0196	uint16 capacity	毫升	-	
	材质	0x0197	uint8 material	-	-	0：硅胶 1：PPSU 2：玻璃
	光照亮强度	0x0198	uint8 illuminanceState	-	-	0：黑夜 1：白天 2：室内强光
	光照亮值	0x0199	uint16 illuminanceValue	流明	-	
	体重	0x0200	uint16 weight	公斤	0.01	
	阻值	0x0201	uint32 impedance	欧姆	0.001	
	体脂百分比	0x0202	uint16 body_fat_percentage	百分比	0.01	
	BMI	0x0203	uint16 BMI	-	0.01	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	肌肉质量	0x0204	uint16 muscle_mass	公斤	0.01	
	肌肉百分比	0x0205	uint16 muscle_percentage	百分比	0.01	
	去脂体重	0x0206	uint16 fatfree_mass	公斤	0.01	
	身高	0x0207	uint16 height	米	0.001	
	体水质量	0x0208	uint16 body_water_mass	公斤	0.01	
	基础代谢率	0x0209	uin16 basal_metabolism	-	-	
	血氧饱和度	0x020A	uint16 SpO2	百分比	0.01	
	收缩压	0x020B	uint16 systolic_pressure	mmHg	0.1	
	舒张压	0x020C	uint16 diastolic_pressure	mmHg	0.1	
	血糖	0x020D	uint16 bluod_glucose	mmol/L	0.01	
	速度	0x020E	uint16 speed	km/H	0.01	
	配速	0x020F	uint8 pace	min/km	0.1	
	坡度	0x0210	uint16 slope	百分比	0.01	
	阻力档位	0x0211	uint8 motionResistanceLevel	-	-	
	运动次数	0x0212	uint16 sport_count	次数	-	
	运动开始时间	0x0213	uint32 sport_start_time	秒	-	unix时间戳

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
运动健康	运动时间	0x0214	uint16 sporting_time	分钟	-	
	目标运动时间	0x0215	uint16 target_time	分钟	-	
	运动距离	0x0216	uint16 distance	米	-	
	目标运动距离	0x0217	uint16 target_distance	米	-	
	心率	0x0218	uint8 heart_rate	次数/分钟	-	
	转速	0x0219	uint16 tachographs	次数/分钟	-	
	当天累计步数	0x021A	uint32 total_step	次数		
	久坐时长	0x021B	uint32 sitting_time	秒		
	运动时长	0x021C	uint32 sport_time	秒		
	睡眠时长	0x021D	uint32 sleep_time	秒		
	浅睡时长	0x021E	uint32 light_sleep_time	秒		
	深睡时长	0x021F	uint32 deep_sleep_time	秒		
	翻身次数	0x0220	uint16 turn_over	次数		
	累计能量消耗	0x0221	uint16 total_calories	千卡	0.1	
	当前步数	0x0222	uint32 current_step	次数		
	运动结束时间	0x0223	uint32 sportStopTime	秒	-	unix时间戳
	高度	0x0224	uint16 hight	米	-	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	平滑度	0x0225	uint8 smoothness	-	-	
	消耗热量	0x0226	uint16 burnCalories	千卡	0.1	
	当前阻力	0x0227	uint8 currentMotionResistance	-	-	
	身体类型	0x0228	uint8 bodyType	-	-	0：隐胖型 1：偏胖型 2：肌肉型偏胖 3：缺乏锻炼型 4：标准型 5：标准肌肉型 6：偏瘦型 7：偏瘦肌肉型 8：肌肉发达型
	身体得分	0x0229	uint8 bodyScore	-	-	
	身体年龄	0x022A	uint8 bodyAge	-	-	
	桨频	0x022B	uint8 SPM	-	-	
	剩余运动距离	0x022C	uint16 restDistance	米	-	
	剩余运动时间	0x022D	uint16 restSportTime	分钟	-	
	运动打分	0x022E	uint8 sportGrade	-	-	
空气质量数据	PM2.5指数	0x0300	uint16 PM25	ug/m3	-	
	PM10指数	0x0301	uint16 PM10	ug/m3	-	
	二氧化硫	0x0302	uint16 SO2	ug/m3	-	
	二氧化氮	0x0303	uint16 NO2	ug/m3	-	
	一氧化碳	0x0304	uint16 CO	ug/m3	-	
	二氧化碳	0x0305	uint16 CO2	ug/m3	-	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	臭氧	0x0306	uint16 O3	ug/m3	-	
	甲醛含量	0x0307	uint16 HCHO	mg/m3	-	
	TVOC	0x0308	uint16 TVOC	ug/m3	-	
	空气质量指数	0x0309	uint16 AQI	-	-	
水质量数据	PH值	0x030A	uint16 PH	-	0.01	
	溶解氧	0x030B	uint8 dissolved_oxygen	mg/L	-	
	TDS（溶解性总固体）	0x030C	uint16 TDS	mg/L	-	
	门窗状态	0x0400	uint8 Door_state	-	-	0：关闭 1：开启
	人类活动	0x0401	uint8 human_activity	-	-	0：无人 1：有人
	常开状态	0x0402	uint8 alwaysOpenState	-	-	0：未常开 1：常开状态
	反锁状态	0x0403	uint8 innerLockState	-	-	0：未反锁 1：反锁
	水量状态	0x0404	uint8 watervolumeStatus	-	-	
	清洗状态	0x0405	uint8 cleaningStatus	-	-	0：空闲 1：正在清洗 2：清洗完成
	在线状态	0x0406	uint8 onlinestate	-	-	0：离线 1：在线
	充电状态	0x0407	uint8 chargingstate	-	-	0：未充电 1：充电中
	门状态	0x0408	uint8 doorstatus	-	-	0：关闭 1：打开
	水泵状态	0x0409	uint8 pumpstatus	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
设备状态属性	注水状态	0x040A	uint8 waterinjectionstatus	-	-	0：无注水 1：等待注水 2：正在注水 3：注水完成
	持续燃烧状态	0x040B	uint8 hintburningstatus	-	-	0：无 1：有
	水流状态	0x040C	uint8 flowstatus	-	-	0：无 1：有
	火焰状态	0x040D	uint8 flamestatus	-	-	0：关闭 1：打开
	风机状态	0x040E	uint8 fanstatus	-	-	0：关闭 1：打开
	亮碟剂状态	0x040F	uint8 dishbrightstatus	-	-	0：无 1：有
	洗碗盐状态	0x0410	uint8 dishwashsaltstatus	-	-	0：无 1：有
	水箱状态	0x0411	uint8 tankstatus	-	-	0：水箱缺水 1：水箱已满
	杯体状态	0x0412	uint8 cupstatus	-	-	0：检测中 1：冷杯 2：热杯
	设备目标状态	0x0413	uint8 device_target_status	-	-	0：取消 1：启动 2：暂停
	缺水状态	0x0414	uint8 water_shortage	-	-	0：正常 1：缺水
	锅具状态	0x0415	uint8 pot_status	-	-	0：无 1：有
	预热状态	0x0416	uint8 preheating_status	-	-	0：无 1：预热开始 2：预热中 3：预热时限到达 4：预热完保持
	下层门状态	0x0417	uint8 down_door_status	-	-	0：关闭 1：打开
	上层门状态	0x0418	uint8 up_door_status	-	-	0：关闭 1：打开
	滤芯状态五	0x0419	uint8 filter_status_five	-	-	0：关闭 1：需要更换

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	滤芯状态四	0x041A	uint8 filter_status_four	-	-	0：关闭 1：需要更换
	滤芯状态三	0x041B	uint8 filter_status_three	-	-	0：关闭 1：需要更换
	滤芯状态二	0x041C	uint8 filter_status_two	-	-	0：关闭 1：需要更换
	滤芯状态一	0x041D	uint8 filter_status_one	-	-	0：关闭 1：需要更换
	左右旋转/摇头/摆风	0x0500	uint8 Angle_Auto_LR_OnOff	-	-	0：关闭 1：开启
	上下旋转/摇头/摆风	0x0501	uint8 Angle_Auto_UD_OnOff	-	-	0：关闭 1：开启
	左右移动	0x0502	uint8 Position_Auto_LR_OnOff	-	-	0：关闭 1：开启
	前后移动	0x0503	uint8 Position_Auto_FB_OnOff	-	-	0：关闭 1：开启
	上下移动	0x0504	uint8 Position_Auto_UD_OnOff	-	-	0：关闭 1：开启
	负离子功能	0x0505	uint8 Anion_OnOff	-	-	0：关闭 1：开启
	恒湿	0x0506	uint8 Constant_Humidity_OnOff	-	-	0：关闭 1：开启
	睡眠功能	0x0507	uint8 Sleep_OnOff	-	-	0：关闭 1：开启
	节能功能/ECO模式	0x0508	uint8 ECO_OnOff	-	-	0：关闭 1：开启
	强力功能	0x0509	uint8 Strong_OnOff	-	-	0：关闭 1：开启
	自动功能/智能功能	0x050A	uint8 Auto_OnOff	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	童锁功能	0x050C	uint8 Child_Lock_OnOff	-	-	0：关闭 1：开启
	屏显功能	0x050D	uint8 Display_OnOff	-	-	0：关闭 1：开启
	绿净功能	0x050E	uint8 Green_OnOff	-	-	0：关闭 1：开启
	防直吹功能/无风感功能	0x050F	uint8 No_Straight_Blowing_OnOff	-	-	0：关闭 1：开启
	加热功能	0x0510	uint8 heatingfunction	-	-	0：关闭 1：开启
	制冷功能	0x0511	uint8 Cooling_OnOff	-	-	0：关闭 1：开启
	按摩功能/抓捏功能	0x0512	uint8 Massage_OnOff	-	-	0：关闭 1：开启
	气泡功能	0x0513	uint8 Bubble_OnOff	-	-	0：关闭 1：开启
	气压功能	0x0514	uint8 Air_Pressure_OnOff	-	-	0：关闭 1：开启
	震动功能	0x0515	uint8 Vibrating_OnOff	-	-	0：关闭 1：开启
	上管加热	0x0516	uint8 UP_Tube_Baking_OnOff	-	-	0：关闭 1：开启
	下管加热	0x0517	uint8 Bottom_Tube_BakingOnOff	-	-	0：关闭 1：开启
	上下管加热	0x0518	uint8 Both_Tube_Baking_OnOff	-	-	0：关闭 1：开启
	火力	0x0519	uint8 Heat_Power	-	-	0：正常 1：武火 2：文火 3：爆炒 4：高火 5：中高火 6：中火 7：中低火 8：低火 9：风冷
	保温	0x051A	uint8 Preserve_Heat_OnOff	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	口感/味道	0x051B	uint8 Taste	-	-	0：无 1：软糯 2：适中 3：劲道 4：高纤 5：细腻 6：浓郁 7：清淡 8：热饮 9：冷饮 10：标准 11：硬 12：香滑 13：清香
	米类型	0x051C	uint8 Rice_Type	-	-	0：标准米 1：长粒米 2：短粒米 3：糙米 4：香米 5：南方米 6：北方米
	甜度	0x051D	uint8 Sweet_Taste	-	-	0：不甜 1：标准 2：香甜
	微波功能	0x051E	uint8 Micro_Wave	-	-	0：关闭 1：微波 2：变频微波
	光波功能	0x051F	uint8 Light_Wave	-	-	0：关闭 1：光波1 2：光波2 3：变频光波 4：微波 5：组合一 6：组合二 7：变频光波二 8：变频光波三 9：变频光波四
	除味功能	0x0520	uint8 Deodorization_OnOff	-	-	0：关闭 1：开启
	扇叶方向	0x0521	uint8 Fan_Direction	-	-	0：正转 1：反转
	点动功能	0x0522	uint8 Step_Action_OnOff	-	-	0：关闭 1：开启
	人感功能	0x0523	uint8 Human_detect_OnOff	-	-	0：关闭 1：开启
	红外模式	0x0524	uint8 IR_OnOff	-	-	0：关闭 1：开启
	速冷	0x0525	uint8 Quick_Fronzen_OnOff	-	-	0：关闭 1：开启
	除霜/化冻	0x0526	uint8 Defrost_OnOff	-	-	0：关闭 1：开启
	抗菌/杀菌/除螨	0x0527	uint8 Antibiosis_OnOff	-	-	0：关闭 1：开启
	银离子	0x0528	uint8 Silver_Ion_OnOff	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	加强洗功能	0x0529	uint8 Strong_Washing_OnOff	-	-	0：关闭 1：开启
	加速洗功能	0x052A	uint8 Quick_Washing_OnOff	-	-	0：关闭 1：开启
	峰谷电	0x052B	uint8 Off_Peak_OnOff	-	-	0：关闭 1：开启
	低温	0x052C	uint8 Low_Temperature_OnOff	-	-	0：关闭 1：开启
	免熨	0x052D	uint8 Iron_Free_OnOff	-	-	0：关闭 1：开启
	紫外线消毒	0x052E	uint8 Ultraviolet_OnOff	-	-	0：关闭 1：开启
	风干	0x052F	uint8 Air_Dry_OnOff	-	-	0：关闭1：开启
	恒温	0x0530	uint8 Constant_Temperature_OnOff	-	-	0：关闭1：开启
	快炖功能	0x0531	uint8 Quick_Stew_OnOff	-	-	0：关闭1：开启
	减震功能	0x0532	uint8 Shock_Absorbtion_OnOff	-	-	0：关闭 1：沙滩 2：草地 3：塑胶跑道 4：公路 5：开启
	背景灯开关	0x0533	bool backgroundLightOnOff	-	-	0：关闭 1：开启
	主灯开关	0x0534	bool mainLightOnOff	-	-	0：关闭 1：开启
	静音	0x0535	bool muteOnOff	-	-	0：关闭 1：开启
	加湿	0x0536	bool humidificationOnOff	-	-	0：关闭 1：开启
	清洁功能/清洗功能	0x0537	uint8 Clean_OnOff	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	冷风功能	0x0538	uint8 Cool_Wind_OnOff	-	-	0：关闭 1：开启
	热风功能	0x0539	uint8 Hot_Wind_OnOff	-	-	0：关闭 1：开启
	智能风	0x053A	uint8 smartWind	-	-	0：关闭 1：智能风 2：风随人 3：风避人 4：环绕风 5：关怀模式
	8度制热	0x053B	uint8 eightDegreeHeating	-	-	0：关闭 1：开启
	无人节能	0x053C	uint8 nobodySave	-	-	0：关闭 1：开启
	低温除湿	0x053D	uint8 lowtemDry	-	-	0：关闭 1：开启
	防霉	0x053E	uint8 mildewProof	-	-	0：关闭 1：开启
	电加热功能	0x053F	uint8 electricHeating	-	-	0：关闭 1：开启
	当前功率	0x0540	uint16 currentPower	瓦特	1	设备当前功率
	健康功能	0x0541	uint8 healthfunction	-	-	0：关闭 1：开启
	角度	0x0542	uint16 angle	度	1	
				-	-	
	温度档位	0x0544	uint8 tempSwtich	档	-	
	音乐功能	0x0545	uint8 musicSwtich	-	-	0：关闭 1：开启
	气囊按摩力度	0x0546	uint8 massageAirStrength	档	-	
	窗帘控制	0x0547	uint8 curtainConrtol	-	-	0：关闭窗帘 1：打开窗帘 2：停止窗帘
	窗帘打开位置	0x0548	uint8 curtainPosition	百分比	-	取值0~100

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	电池电量状态	0x0549	uint8 batteryStatus	-	-	0：低电量 1：正常
	动作方向	0x054A	uint8 actionDirection			1：前进 2：后退 3：左转 4：右转 5：停止 6：起立 7：坐下 8：上升 9：下降
	安全自检	0x054B	uint8 selfCheck	-	-	0：正常 1：异常
	高温解锁	0x054C	uint8 highTemp_Unlock	-	-	0：关闭 1：打开
	恒温温度设置	0x054D	uint16 constantTemp_Set	开尔文	0.01	设备温度范围
	一键巡航时间	0x054E	uint8 autoTime	-	-	
	预约开始剩余时间	0x054F	uint16 ReservationRemainingTime	-	-	
	预约剩余时间	0x0550	uint16 ReserveLeftTime	-	-	
	计时清零	0x0551	uint8 timerReset	-	-	0：关闭 1：打开
	照明	0x0552	uint8 illumination	-	-	0：关闭 1：打开
	背部局部按摩	0x0553	uint8 backmassage	-	-	0：关闭 1：打开
	头部局部按摩	0x0554	uint8 headmassage	-	-	0：关闭 1：打开
	膝盖局部按摩	0x0555	uint8 kneemassage	-	-	0：关闭 1：打开
	腿部局部按摩	0x0556	uint8 legmassage	-	-	0：关闭 1：打开
	小腿位置	0x0557	uint8 calfposition	-	-	1：小腿升 2：小腿降 3：小腿升降停 4：小腿伸 5：小腿缩 6：小腿伸缩停

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	气囊位置	0x0558	uint8 massageairposition	-	-	1：全身 2：臂肩 3：背腰部 4：坐垫部 5：腿部
	背机芯位置	0x0559	uint8 backrestposition	-	-	1：背机芯上行 2：背机芯上行停 3：背机芯下行 4：背机芯下行停
	3D机芯	0x055A	uint8 3Dmassage	-	-	1：3D1 2：3D2 3：3D3
	膝盖位置	0x055B	uint8 kneeposition	-	-	0：伸 1：缩 2：停
	机芯幅度	0x055C	uint8 massagerange	-	-	1：宽 2：中 3：窄
	按摩位置	0x055D	uint8 massageposition	-	-	1：全身 2：局部 3：定点 4：肩部 5：手部 6：腿脚部 7：颈部 8：腰部 9：四肢
	脚底滚轮	0x055E	uint8 soleroller	-	-	0：关 1：慢速 2：中速 3：快速 4：开
	回充功能	0x055F	uint8 chargeback	-	-	0：关闭 1：打开
	地图轨迹	0x0560	string maptrack	-	-	
	地图	0x0561	string map	-	-	
	脱水转速	0x0562	uint16 targetspinspeed	-	-	
	时间显示类型	0x0563	uint8 timedisplaytype	-	-	1：显示时间 2：跑马灯
	增压	0x0564	uint8 turbomode	-	-	0：关闭 1：开启
	一氧化碳自检	0x0565	uint8 coteststatus	-	-	0：无 1：有
	转叉旋转	0x0566	uint8 turnswitch	-	-	0：关闭 1：开启
	保管功能	0x0567	uint8 keep_function	-	-	0：关闭 1：打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	工作阶段	0x0568	uint8 worksteps	-	-	0：待机 1：预洗 2：主洗 3：漂洗 4：干燥 5：结束 6：保管中
	翻面	0x0569	uint8 over_turn	-	-	0：关闭 1：打开
	解冻类型	0x056A	uint8 thawing_type	-	-	0：默认 1：-1度解冻 2：0度解冻 3：3度解冻
	烘烤中加蒸汽	0x056B	uint16 steam_baking	-	-	
	发酵中加蒸汽	0x056C	uint16 steam_add_fermentation	-	-	
	滤芯复位一	0x056D	uint8 filter_reset_one	-	-	0：关闭 1：打开
	一键冲洗	0x056E	uint8 one_button_wash	-	-	0：关闭 1：打开
	滤芯复位	0x056F	uint8 filter_reset	-	-	0：关闭 1：打开
	冲洗功能	0x0570	uint8 washing_switch	-	-	0：关闭 1：打开
	本地食谱	0x0571	uint16 local_recipe	-	-	参见1.5《本地食谱表》
	夜灯	0x0572	uint8 nightLight	-	-	0：关闭 1：打开
	感光	0x0573	uint8 lightSensor	-	-	0：关闭 1：打开
	自然睡眠	0x0574	uint8 natualSleep	-	-	0：关闭 1：打开
	热敷功能	0x0575	uint8 heating	-	-	0：关闭 1：打开
	烘干功能	0x0576	uint8 drying	-	-	0：关闭 1：打开
	位置控制	0x0577	uint8 motorControl	-	-	0：停止 1：上升 2：下降

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	驱蚊剂状态	0x0578	uint8 mosquitoRepellentStatus	-	-	0：不足 1：正常
	烤色	0x0579	uint8 bakeColor	-	-	0：浅色 1：中色 2：深色
	回水功能	0x057A	uint8 backWaterFunction	-	-	0：关闭 1：打开
	驱蚊功能	0x057B	uint8 mosquitoRepellent	-	-	0：关闭 1：打开
	加热状态	0x057C	uint8 heatStatus	-	-	0：关闭 1：打开
	单吸模式	0x057D	uint8 singleSuckMode	-	-	0：关闭 1：打开
	双吸模式	0x057E	uint8 dualSuckMode	-	-	0：关闭 1：打开
	静谧模式	0x057F	uint8 silentSuckMode	-	-	0：关闭 1：打开
	水控	0x0580	uint8 onOff	-	-	0：关闭 1：打开
	按摩速度	0x0581	uint8 massageSpeed	-	-	
	零重力	0x0582	uint8 zeroGravity	-	-	
	手动按摩	0x0583	uint8 massageManual	-	-	
	准备时间	0x0584	uint16 prepareTime	-	-	用户可在准备时间内取消上次操作
	其他模式温度上限	0x0585	uint16 otherTempUpLimit	摄氏度	0.1	
	是否有回水板	0x0586	uint8 backWaterBoard	-	-	0：无 1：有
	门锁布防	0x0587	uint8 armMode	-	-	0：未布防 1：布防

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	开锁方式	0x0588	uint8 lockType	-	-	1：指纹 2：密码 3：卡 4：机械钥匙 5：一次性密码 6：APP开锁 7：组合开锁
	用户权限	0x0589	uint8 userLimit	-	-	1：普通用户 2：管理员 3：劫持用户
	添加锁用户	0x058A	uint8 userID uint8 lockType	-	-	userID：0表示由锁添加用户ID，其余表示添加userID的用户 lockType：开锁方式
	删除锁用户	0x058B	uint8 userID uint8 lockType	-	-	userID：0表示由锁删除用户ID，其余表示删除userID的用户 lockType：开锁方式
	配置用户权限	0x058C	uint8 userID uint8 lockType uint8 userLimit	-	-	userID：钥匙序号 lockType：开锁方式 userLimit：用户权限
	添加一次性密码	0x058D	uint8 len uint8 pwd[]	-	-	len：密码长度，0表示由锁生成密码 pwd：下发的密码
	钥匙时效	0x058E	uint8 userID uint8 lockType uint8 period uint32 startTime uint32 expiredTime uint16 activeTimePerDay uint16 inactiveTimePerDay	-	-	userID：用户序号 lockType：开锁方式 period：有效日期，bit0~bit6为周日至周六 startTime：钥匙生效时间，unix时间 expiredTime：钥匙失效时间， unix时间 activeTimePerDay：钥匙每日生效时间 inactiveTimePerDay：钥匙每日失效时间
	锁面板操作	0x058F	uint8 onOff	-	-	0：关闭 1：打开
	自动上锁	0x0590	uint8 autoLock	-	-	0：关闭 1：打开
	门锁双重验证	0x0591	uint8 doubleCheck	-	-	0：关闭 1：打开
	门外上锁	0x0592	uint8 outLockState	-	-	0：关闭 1：打开
	门锁睡眠开始时间	0x0593	uint16 sleepStartTime	-	-	门锁睡眠模式开启时间 0x0000 = 00:00
	门锁睡眠结束时间	0x0594	uint16 sleepEndTime	-	-	门锁睡眠模式结束时间 0x0600 = 06:00

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
设备功能属性	未锁报警状态	0x0595	uint8 onOff	-	-	0：关闭 1：开启
	未锁报警延迟时间	0x0596	uint8 delayTime	分钟	-	报警延迟时间，单位分钟
	禁试报警状态	0x0597	uint8 onfOff	-	-	0：关闭 1：开启
	禁试报警触发次数	0x0598	uint8 count	-	-	几次后触发禁试报警
	禁试报警恢复时间	0x0599	uint8 resumeTime	-	-	恢复时间，单位分钟
	验证操作密码	0x059A	uint8 len uint8 pwd[]	-	-	len：密码长度 pwd：下发的密码
	滤芯复位二	0x059B	uint8 onOff	-	-	0：关闭 1：打开
	每日生效时间	0x059C	uint16 activeTimePerDay	-	-	每日生效时间 0x0600 = 06:00
	每日失效时间	0x059D	uint16 inactiveTimePerDay	-	-	每日失效时间 0x0600 = 06:00
	钥匙生效时间	0x059E	uint32 startTime	秒	-	startTime：钥匙生效时间，unix时间
	钥匙失效时间	0x059F	uint32 expiredTime	秒	-	expiredTime：钥匙失效时间， unix时间
	每周生效日期	0x05A0	uint8 entryinfoForceTime	-	-	有效日期，bit0~bit6为周日至周六
	夜灯生效时间	0x05A1	uint16 nightLightStartTime			夜灯生效时间 0x2000 = 20:00
	夜灯结束时间	0x05A2	uint16 nightLightEndTime			夜灯失效时间 0x0600 = 06:00
	干垃圾桶	0x05A3	uint8 residua	-	-	0：关闭 1：开启
	湿垃圾桶	0x05A4	uint8 householdFood	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	可回收垃圾桶	0x05A5	uint8 recyclable	-	-	0：关闭 1：开启
	有害垃圾桶	0x05A6	uint8 hazardous	-	-	0：关闭 1：开启
	睡眠类型	0x05A7	uint8 weekperiod	-	-	0：关闭 1：一般 2：老人 3：青年 4：儿童
	润冷功能	0x05A8	uint8 moistColdOnOff	-	-	0：关闭 1：开启
	室内清洁	0x05A9	uint8 inCleanOnOff	-	-	0：关闭 1：开启
	室外清洁	0x05AA	uint8 outCleanOnOff	-	-	0：关闭 1：开启
	粉碎功能	0x05AB	uint8 brokenOnOff	-	-	0：关闭 1：开启
	一键同步	0x05AC	uint8 oneClickSync	-	-	0：关闭 1：开启
	烘干设置	0x05AD	uint8 drygingSetting	-	-	0x00：无 0x01：晾晒 0x02：即穿 0x03：特干 0x04：三十分钟 0x05：六十分钟 0x06：九十分钟 0x07：一百二十分钟 0x08：一百五十分钟 0x09：一百八十分钟
	当前步骤	0x05AE	uint8 currentStep	-	-	1：结束 2：烘干 3：脱水 4：漂洗 5：主洗 6：预洗 7：称重 8：保留
	壶盖开关	0x05AF	uint8 lidSwitch	-	-	0：关闭 1：开启
	加水开关	0x05B0	uint8 waterSwitch	-	-	0：关闭 1：开启
	水类型	0x05B1	uint8 waterType	-	-	1：自来水 2：纯净水
	加热时间	0x05B2	uint16 heatTime	分钟	0.1	
	浓度	0x05B3	uint8 concentration	-	-	0：高 1：中 2：低
	冰厚度	0x05B4	uint8 IceThickness	-	-	1：薄冰 2：中冰 3：厚冰

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	冷却方式	0x05B5	uint8 coolingMode	-	-	1：水冷 2：风冷
	全天开关	0x05B6	uint8 allDay	-	-	0：关闭 1：开启
	删除钥匙	0x05B7	uint8 deletedLockType			0xFF：全部类型 0x01：指纹 0x02：密码 0x03：卡 0x04：机械钥匙 0x05：一次性密码 0x06：APP开锁 0x07：组合开锁
	保压时间	0x05B8	uint8 keepPressureTime	分钟	-	
	睡眠呼吸质量	0x05B9	uint16 sleepQuality	-	-	
	睡眠呼吸指数	0x05BA	uint16 sleepQualityIndex	-	-	
	入睡时长	0x05BB	uint16 toSleepTime	分钟	-	
	强制模式	0x05BC	uint8 forceMode	-	-	0：关闭 1：开启
	变温空间	0x05BD	uint8 variableTemperatureSpace			0：无 1：臻鲜 2：零度保鲜 3：冰鲜 4：母婴 5：零度养鲜 6：原鲜
	蒸汽时间	0x05BE	uint8 steamingTime	分钟	-	
	搅拌时间	0x05BF	uint8 mixingTime	分钟	-	
	煮水开关	0x05C0	uint8 boilingWater	-	-	0：关闭 1：开启
	消毒开关	0x05C1	uint8 disinfection	-	-	0：关闭 1：开启
	降温开关	0x05C2	uint8 coolSwitch	-	-	0：关闭 1：开启
	炖煮时间	0x05C3	uint16 stewTime	分钟	-	

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	除氯时间	0x05C4	uint8 dechlorinationTime	分钟	-	
	中温模式	0x05C5	uint8 midtempSwitch	-		0：关闭 1：开启
	旋转	0x05C6	uint8 turningSwitch	-	-	0：关闭 1：开启
	错误报警	0x05C7	uint8 wrongAlarm	-	-	0：无报警（正常状态） 1：温度传感器断路 2：温度传感器短路
	ASR控制	0x0600	uint16 asrwords[20]			ASR控制指令
	按键控制	0x0601	uint8 keycode[4]			keycode
	头部控制	0x0602	uint8 headControl	-	-	0： 升起 1：下降 2：停止
	智能记忆	0x0603	bool intelligentMemory	-	-	0：关闭 1：开启
	温度锁定	0x0604	bool temperatureLocking	-	-	0：关闭 1：开启
	节能净泡	0x0605	bool energyConservation	-	-	0：关闭 1：开启
	刷牙次数	0x0606	uint8 brushTime	次	-	
	刷牙得分	0x0607	uint8 brushScore	分	-	
	换区提醒间隔	0x0608	uint8 areaChangeGap	秒	-	
	揉捏开关	0x0609	bool kneadingSwitch	-	-	0：关闭 1：开启
	揉捏速度	0x060A	uint8 kneadingSpeed			
	推拿开关	0x060B	bool massageSwitch	-	-	0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	小腿气囊开关	0x060C	bool calfAirBagSwitch	-	-	0：关闭 1：开启
	手臂气囊开关	0x060D	bool armAirBagSwitch	-	-	0：关闭 1：开启
	臀部气囊开关	0x060E	bool hipAirBagSwitch	-	-	0：关闭 1：开启
	靠背升降	0x060F	uint8 backUpDown	-	-	
	柔风功能	0x0610	bool softWindMode			0：关闭 1：开启
	上下扫风	0x0611	uint8 verticalMode			0：关闭 1：全屋扫风 2：上送风 3：下送风
	上下定格	0x0612	uint8 upDownlAngleSelect			0：关闭 1：当前位置定格 2：上定格 3：偏上定格 4：中定格 5：偏下定格 6：下定格
	左右扫风	0x0613	uint8 horizontalMode			0：关闭 1：全屋扫风 2：左送风 3：中送风 4：右送风 5：广角送风 6：偏左送风 7：偏右送风
	左右定格	0x0614	uint8 leftRightlAngleSelect			0：关闭 1：当前位置定格 2：左定格 3：偏左定格 4：中定格 5：偏右定格 6：右定格 7：左广角定格 8：右广角定格
	吸力	0x0615	uint8 fanLevel			
	滚刷寿命	0x0616	uint8 brushLife			
	重置滚刷	0x0617	bool rollBrushReset	-	-	0：关闭 1：开启
	水量	0x0618	uint8 waterLevvel			
	风随温变	0x0619	bool tempWindSwitch			0：关闭 1：开启

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	定时关分钟十位	0x061A	uint8 timingOffMinute			0：zero/零 1：ten/十分钟 2：twenty/二十分钟 3：thirty/三十分钟 4：forty/四十分钟 5：fifty/五十分钟 6：cleanOne/清除定时关机一 7：cleanTwo/清除定时关机二
	定时关小时	0x061B	uint8 timingOffHour			
	定时关分钟个位	0x061C	uint8 timingOffMinuteEx			
	定时开分钟十位	0x061D	uint8 timingOnMinute			0：zero/零 1：ten/十分钟 2：twenty/二十分钟 3：thirty/三十分钟 4：forty/四十分钟 5：fifty/五十分钟 6：cleanOne/清除定时开机一 9：cleanTwo/清除定时开机二
	定时开小时	0x061E	uint8 timingOnHour			
	定时开分钟个位	0x061F	uint8 timingOnMinuteEx			
	环境温度反馈	0x0620	uint8 temperatureFeedback			
	推拿速度	0x0621	uint8 massageTempo			
	中温温度	0x0622	uint16 mediumTemp	摄氏度	1	
	背部控制	0x0623	uint8 dorsalControl			0：上升 1：下降 2：停止 3：平躺
	腿部控制	0x0624	uint8 legControl			0：上升 1：下降 2：停止 3：平躺
	灭蚊功能	0x0625	uint8 killMosquitoes_OnOff			0：打开 1：关闭
	三百六十度摇头	0x0626	uint8 angleAutoAllOnOff			0：关闭 1：打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	当前水温	0x0627	uint16 currentTemperature	摄氏度		最大：49 最小：1 步长：1
	风干时间	0x0628	uint16 airdryTime			
	消毒时间	0x0629	uint16 ultravioletTime			
	冷水开关	0x062A	uint8 getcoldwaterSwitch			0：关闭 1：打开
	热水开关	0x062B	uint8 gethotwaterSwitch			0：关闭 1：打开
	保温开关	0x062C	uint8 heatpreservation			0：关闭 1：打开
	冷水取水时间	0x062D	uint16 getcoldwaterTime	秒	1	0-60
	热水取水时间	0x062E	uint16 gethotwaterTime	秒	1	0-60
	浸泡功能	0x062F	uint8 immersion			0：关闭 off 1：打开 on
	小夜灯亮度	0x0630	uint16 lampLightness	无	1	0-100
	内部温度	0x0631	uint16 innerTemperature	摄氏度	1	0-100
	外部温度	0x0632	uint16 outerTemperature	摄氏度	1	0-100
	刷头寿命	0x0633	uint16 brushLongevity	天	1	0-90
	刷牙时长	0x0634	uint16 brushLength	分钟	1	0-10
	刷牙起始区	0x0635	uint16 brushstartPosition			待更新

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	下层取暖档位	0x0636	uint8 lowerlayerHeatlevel	档	-	下层加热档位
	冲厕	0x0637	uint8 flushing			0：关闭 off 1：打开 on
	小冲	0x0638	uint8 smallFlushing			0：关闭 off 1：打开 on
	水温档位	0x0639	uint16 waterTemp			0-5
	风温档位	0x063A	uint16 windTemp			0-5
	座温档位	0x063B	uint16 seatTemp			0-5
	水压	0x063C	uint16 waterGage			1-5
	喷嘴位置	0x063D	uint16 nozzleLocation			1 - 后 2 - 偏后 3 - 中 4 - 偏前 5 - 前
	妇洗功能	0x063E	uint8 womanWash			0：关闭 off 1：打开 on
	臀洗功能	0x063F	uint8 hipWash			0：关闭 off 1：打开 on
	烘干功能	0x0640	uint8 Dry			0：关闭 off 1：打开 on
	畅洗功能	0x0641	uint8 freeWash			0：关闭 off 1：打开 on
	坐浴功能	0x0642	uint8 hipBath			0：关闭 off 1：打开 on
	童洗功能	0x0643	uint8 childWash			0：关闭 off 1：打开 on
	一键停止	0x0644	uint8 oneKeystop			0：关闭 off 1：打开 on
	关盖大冲	0x0645	uint8 flushingCloseroof			0：关闭 off 1：打开 on
	关盖小冲	0x0646	uint8 smallFlushcloseroof			0：关闭 off 1：打开 on

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	关盖控制	0x0647	uint8 closeroof			0：关闭 off 1：打开 on
	座圈控制	0x0648	uint8 heatRing			0：关闭 off 1：打开 on
	一键自动	0x0649	uint8 oneKeyauto			0：关闭 off 1：打开 on
	移动清洗	0x064A	uint8 moveClean			0：关闭 off 1：打开 on
	宽幅强洗	0x064B	uint8 wideClean			0：关闭 off 1：打开 on
	银离子杀菌	0x064C	uint8 silverSterilization			0：关闭 off 1：打开 on
	三百六十度杀菌	0x064D	uint8 allSterilization			0：关闭 off 1：打开 on
	紫外线杀菌	0x064E	uint8 uvSterilization			0：关闭 off 1：打开 on
	清洗模式	0x064F	uint8cleanMode			0：自清洗 1：热烘除菌
	开机时间	0x0650	uint16openTime	分钟	1	
	关机时间	0x0651	uint16closeTime	分钟	1	
	速冻	0x0652	uint8 quickFrozenOnOff			0：关闭 off 1：打开 on
	速冻时间	0x0653	uint16 quickFrozenTime	小时	1	取值：1-10
	预约启动剩余时间	0x0654	uint16 AppointmentTime	秒	1	0-1800
	充电状况	0x0655	uint8 Chargingstatus			0（不在充电），1（正在充电），2（已经充满）

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	消毒次数	0x0656	uint16 WorkCount	次		0-65535
	炒菜时间	0x0657	uint8 SetTime	分		0.5-99
	炒菜温度	0x0658	uint8 SetTemp	档		1-6
	炒菜速度	0x0659	uint8 SetSpeed	档		1-6
	蒸汽功能	0x065A	uint8 steam Function			0：关闭 1：打开
	自动清洗	0x065B	uint8 autoClean			0：关闭 1：打开
		0x065C				
	腿部控制角度	0x065D	anglelegControl	度		0-60
	背部控制角度	0x065E	angledorsalControl	度		0-60
	同步升降	0x065F	uint16 synControl			0上升、1下降、2停止
	预约洗澡	0x0660	uint8 appointmentBath			0：关闭1： 晨洗2：夜浴3：晨洗+夜浴4：全天
	晨洗开始时间	0x0661	uint16 morningBathStart	分钟	-	0-1440
	晨洗结束时间	0x0662	uint16 morningBathend	分钟	-	0-1440
	夜浴开始时间	0x0663	uint16 nightBathstart	分钟	-	0-1440
	夜浴结束时间	0x0664	uint16 nightBathend	分钟	-	0-1440
	循环杀菌	0x0665	uint8 cyclic Sterilization	-	-	0：关闭 1：打开
	预约洗漱	0x0666	uint8 appointmentWash			0：关闭1： 晨洗2：夜浴3：晨洗+夜浴4：全天

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	喷嘴清洁	0x0667	uint8 InjectorClean			0：关闭 1：打开
	温感开关	0x0668	uint8 TempSensor			0：关闭 1：打开
	光感开关	0x0669	uint8 LightSense			0：关闭 1：打开
	自动冲洗	0x066A	uint8 AutoFlushing			0：关闭 1：打开
	自动翻盖	0x066B	uint8 AutoFlip			0：关闭 1：打开
	着座状态	0x066C	uint16 SittingStatus			0：未着座1：着座
	座圈加热状态	0x066D	uint16 SeatHeatStatus			0：未加热 1：加热
	人体感应	0x066E	uint8 HumanSensor			0：关闭 1：打开
	除臭状态	0x066F	uint8 Deodorization			0：关闭 1：打开
	智能温控功能	0x0670	uint8 smartTempControl			0：关闭 1：打开
	瞬吸功能	0x0671	uint8 fastInhale			0：关闭 1：打开
	0x0672	uint16 hotworkingTime	分钟	-	0-1440	
	臀洗水压	0x0673	uint8 hipWashing	档	-	0-100
	妇洗水压	0x0674	uint8 WomanWashing	档	-	0-100
	臀洗喷嘴位置	0x0675	uint16 hipwashNozzle			1 - 后 2 - 偏后 3 - 中 4 - 偏前 5 - 前
	妇洗喷嘴位置	0x0676	uint16 womanwashNozzle			1 - 后 2 - 偏后 3 - 中 4 - 偏前 5 - 前

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
即热工作时间	软水次数	0x0677	uint16 softwashTime	次		0-100
	加强干燥	0x0678	uint16 enhancedDrying			0：关闭 1：打开
	加强洗涤	0x0679	uint16 enhancedWashing			0：关闭 1：打开
	新风功能	0x067A	uint16 freshAir			0：关闭 1：打开
	宠物吹风挡位	0x068B	uint8 petWind			小清新：0 小风暴：1
	摸摸泡	0x068C	uint8 momoSoak			0：关闭 1：打开
	三弟灯光	0x068D	uint8 3dLight			0：关闭 1：打开
	烹饪时长	0x068E	uint16 CookingTime	分钟		0-255 步长1
	焖饭时间	0x068F	uint8 braiseTime	分钟		0-255 步长1
	锅内温度	0x0690	uint8 pottemperature	度		0-255 步长1
	加菜倒计时	0x0691	uint8 addfoodcountdown	分钟		0-255 步长1
	即热工作时间	0x0692	uint16 Hotworkingtime	分钟		0-720 步长1
	一键甩脂	0x0693	uint8 fatGoaway			0：关闭 1：打开
	抖动速度	0x0694	uint16 shakeSpeed	档		0-100 步长 1
	音量控制	0x0695	uint16 volumeControl			0：音量加 1：音量减
	内容切换	0x0696	uint16 contentSwitch			0：上一首 1：下一首
	播放控制	0x0697	uint8 playControl			0：播放 1：暂停

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	假日功能	0x0698	uint16 holidayFunction			0：关闭 1：打开
	保鲜功能	0x0699	uint16 freshkeeping Function			0：关闭 1：打开
	果蔬功能	0x069A	uint16 fruitsFuction			0：关闭 1：打开
	微冻功能	0x069B	uint16 freezingFunction			0：关闭 1：打开
	氛围灯	0x069C	uint16 atmosphereLamp			0：关闭 1：打开
	点动功能	0x069C	uint16 pointMovement			0：关闭 1：打开
	注水量	0x069D	uint16 waterInjection			[0-100]
	零冷水温度	0x069E	uint16 coldwaterTemperature	度		[0-60]（默认30）
	预约开始时间	0x069F	uint16 appointStart	分钟		[0-1440]
	预约结束时间	0x06A0	uint16 appointEnd	分钟		[0-1440]
	水流量控制	0x06A1	uint16 waterDischarge			[0-100]
	供暖状态	0x06A2	uint16 heatingStatus			关闭：0打开：1
	防冻状态	0x06A3	uint16 antifreezeStatus			正常：0防冻：1
	温控器状态	0x06A4	uint16 thermostatStatus			断开：0 闭合：1
	火焰大小状态	0x06A5	uint16 flamesizeState			[0-100]

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	蒸烤功能	0x06A6	uint16 steamingBaking			无 empty 0 蒸汽 steam 蒸汽热风 steamHotWind 上下烧烤 upDownGrill 热风对流 airConvection 立体烤 threeDimensionalGrill 底部烧烤 bottomGrill 薄层快烤 laminaGrill 双上管 doubleUpTube 红外烧烤 infraredGrill 除垢 cleaning 蒸汽杀菌 steamdisinfection
	目标卡路里	0x06A7	unit16 targetCalories			[0-65535]
	云食谱状态	0x06A8	unit16 CloudRecipeStatus			结束：0开始：1
筋膜枪	击打次数	0x06A9	uint16 hitCount			0-65535 步长：1
	当前温度	0x06AA	uint16 CuTemperature			0-65535 步长：1
	按摩控制	0x06AB	uint8 massageControl			0： 开始按摩 1： 停止按摩
智能窗	窗操作模式	0x06AC	uint8 WindowOperateMode			0： 关窗 1： 开窗 2： 暂停 3： 翻转
	提示音	0x06AD	uint8 WarningTone			0： 关闭提示音 1： 开启提示音
筋膜枪	按摩枪转速	0x06AE	uint16 MassageGunSpeed			600-3500 步长：10
洗手机	当日洗手次数	0x06AF	uint16 WashHandOften			[0-65535]
豆浆机	开盖状态	0x06B0	uint8 OpenCoverStatus			0： 允许开盖 1： 不允许开盖
	开盖提醒	0x06B1	uint8 OpenCoverReminder			0： 锁盖 1： 合法开盖 2： 非法开盖
	制冷目标温度	0x06B2	uint8 targetRefrigerationtemp	摄氏度		[0-40]

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
饮水机	制热目标温度	0x06B3	uint8 targetheatTemp	摄氏度		[0-100]
	总溶解固体值	0x06B4	uint32 TDS			[0-42949673.0]
	冲洗完成	0x06B5	uint8 washFinished			0： 冲洗完成失败 1： 冲洗完成
	目标水量	0x06B6	uint16 targetWaterYield	毫升		[0-9999]
智能窗	窗控制	0x06B7	uint8 windowControl			0： 继续 2： 暂停
	翻转控制	0x06B8	uint8 flipControl			0： 无 3： 翻转
暖宫带	腹部温度	0x06B9	uint16 waistTemperature	摄氏度		[0-100]
	腰部温度	0x06BA	uint16 abdominalTemperature	摄氏度		[0-100]
	发热区域	0x06BB	uint16 feverArea			1：腹部 2：腰部 3：同时打开
	腹部当前温度	0x06BC	uint16 waistCurrentTemp	摄氏度		[37.0-65.0]
	腰部当前温度	0x06BD	uint16 abdominalCurrentTemp	摄氏度		[37.0-65.0]
智能水杯	当前水量	0x06BE	uint16 waterVolume	毫升		[0.0-6000.0]
	饮水量	0x06BF	uint16 drinkVolume	毫升		[0.0-6000.0]
	绊绳次数	0x06C0	uint16 tripNumber	次		[0-65535]
	当前频率	0x06C1	uint16 currentFrequency	转/分钟		[0-65535]
	平均频率	0x06C2	uint16 averageFrequency	转/分钟		[0-65535]

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	最高频率	0x06C3	uint16 highestFrequency	转/分钟		[0-65535]
	最大连续次数	0x06C4	uint16 maxNumberConsecutive	次		[0-65535]
	累计次数	0x06C5	uint16 cumulativeNumber	次		[0-65535]
热水器	变频速热	0x06C6	uint8 changeRate			0：关闭 1：打开
	自动关机	0x06C7	uint8 autoPower			0：关闭 1：打开
	高温抑菌	0x06C8	uint8 highTemperature			0：关闭 1：打开
	定时时间周	0x06C9	uint8 timingSetWeek			0：星期日 1：星期一 2：星期二 3：星期三 4：星期四 5：星期五 6：星期六
	卫浴温度	0x06CA	uint8 bathTemperature	摄氏度		[0-100]
	采暖温度	0x06CB	uint8 floorTemperature	摄氏度		[0-100]
	采暖模式	0x06CC	uint8 heatingMode			0：全天 1：上班1 2：上班2 3：节约 4：老人 5：夜班 6：儿童
	风机开关	0x06CD	uint8 windMachinePower			0：关闭 1：打开
	水泵开关	0x06CE	uint8 waterMachinePower			0：关闭 1：打开
	防冻功能	0x06CF	uint8 antifreezingSetting			0：关闭 1：打开
	温控器功能	0x06D0	uint8 temperatureControllerSetting			0：断开 1：闭合

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
洗碗机	进水开关	0x06D1	uint8 waterIntakeSetting			0： 关闭 1： 打开
	排水开关	0x06D2	uint8 bleedWaterSetting			0： 关闭 1： 打开
	舱门开关	0x06D3	uint8 hatchDoorSetting			0： 关闭 1： 打开
	碗碟舱	0x06D4	uint8 dishesClassSetting			0： 待机 1： 快洗 2： 标准洗 3： 加强洗
智能跳绳	卡路里	0x06D5	unit16 calories	卡路里		[0-65535]
	电池续航时间	0x06D6	unit16 batteryLife	分		[0-65535]
	出液时间	0x06D7	unit8 transudatesTime	秒		0-10
	剩余次数	0x06D8	unit16 residueDegree	次		0-1500
	洗手次数	0x06D9	unit16 washTimes	次		0-1500
	喝水提醒一	0x06DA	unit16 waterClockOne	分		0-3600
	喝水提醒二	0x06DB	unit16 waterClockTwo			1、6杯水模式 2、7杯水模式 3、8杯水模式 4、9杯水模式 5、10杯水模式 6、无
	食谱编码	0x06DC	unit16 recipeCode			1-65535
	食谱指令长度	0x06DD	unit16 recipeActionSize			0-65535
	食谱指令	0x06DE	int32 recipeAction			
	自动煮水	0x06DF	unit8 autoBoiling			0、关闭 1、打开
	手动煮水	0x06E0	unit8 manualBoiling			0、关闭 1、打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	茶几开关	0x06E1	unit8 teaTableOnOff			0、关闭 1、打开
	暖脚档位	0x06E2	unit8 warmFeetGear			0-10
	智能变升	0x06E3	unit8 intellectChangeSize			0、关闭 1、打开
	智能变频	0x06E4	unit8 intellectfrequency conversion			0、关闭 1、打开
	单次循环	0x06E5	unit8 singleCycle			0、关闭 1、打开
	零冷水	0x06E6	unit8 noneColdWater			0、关闭 1、打开
	安全防护	0x06E7	unit8 safety			0、关闭 1、打开
	智感干洗	0x06E8	unit8 AIdryClean			0、关闭 1、打开
	智能管家	0x06E9	unit8 IntelligentHousekeeper			0、关闭 1、打开
	随烟感	0x06EA	unit8 smokeFollow			0、关闭 1、打开
	操作名	0x06EB	unit16 opreationName			
	脉冲力度	0x06EC	unit8 pulseStrength			0-100
	位置	0x06ED	unit32 position			0-65536
	呼吸灯	0x06EF	unit8 BLNcontrol			0、关闭 1、打开
	运行时间左	0x06F0	unit32 leftRunTime			0-260000
	运行时间右	0x06F1	unit32 rightRunTime			0-260000
	自动消毒	0x06F2	unit8 autoDisinfect			0、关闭 1、打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
设备属性						
	手动消毒	0x06F3	unit8 manualDisinfect			0、关闭 1、打开
	定时功能	0x06F4	unit8 timingFunction			0、关闭 1、打开
	可取消状态	0x06F5	unit8 cancelStatus			0：不可取消 1：可取消
	自清洗状态	0x06F6	unit8 selfCleaningStatus			0、有自清洗 1、无自清洗
	携带提示	0x06F7	unit8 carryTips			0、关闭 1、打开
	更换提示	0x06F8	unit8 replaceTips			0、关闭 1、打开
	睡前提示	0x06F9	unit8 bedtimeTips			0、关闭 1、打开
	保温剩余时间	0x06FA	unit16 keepWarmLeftTime			0-65535
	深度节能	0x06FB	unit8 deepSaveEnergy			0、关闭 1、打开
	自定义温度一	0x06FC	unit16 temperatureOne	0	255	
	自定义温度二	0x06FD	unit16 temperatureTwo	0	255	
	自定义温度三	0x06FE	unit16 temperatureThree	0	255	
	自定义温度四	0x06FF	unit16 temperatureFour	0	255	
	自定义温度五	0x0701	unit16 temperatureFive	0	255	
	红灯开关	0x0702	unit8 redLigntOnOff			0、关闭 1、打开

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	喝水提醒次数	0x0703	unit8 drinkWaterRemindTimes			1-24
	全加档位	0x0704	unit8 fullGearUp			1-5
	全减档位	0x0705	unit8 fullGearDown			1-5
	云食谱完成参数	0x0706	unit8 cloudRecipeDoneParameter			
	坐姿调整	0x0707	unit8 sittingUpDown	1	3	1、上升 2、下降 3、停止
	关盖冲水	0x0708	unit8 closeRoofFlushing			0、关闭 1、打开
	离座冲水	0x0709	unit8 leaveSeatFlushing			0、关闭 1、打开
	香泡泡	0x070A	unit8 fragrantPawpaw			0、关闭 1、打开
	自动香泡泡	0x070B	unit8 autoFragrantBubbles			0、关闭 1、打开
	变升档位	0x070C	unit8 changeLitreGear			1-5
	自动夜灯	0x070D	unit8 autoNightLight			0、关闭 1、打开
	喝水次数	0x070E	unit16 drinkingTimes			0-65535
	刷牙起始区域	0x070F	unit8 brushStartZone			
	换区提醒开关	0x0710	unit8 areaChangeOnOff			0、关闭 1、打开
	延时启动时长	0x0711	unit8 delayStartTime			0-60
	左右手设置	0x0712	unit8 LRhandSetting			0、左手 1、右手

属性分类	属性名称	Attr Type	Data Struct	单位	精度	说明
	当前刷牙面	0x0713	unit8 currentBrushSurface			
	当前刷牙区域	0x0714	unit8currentBrushArea			
	更换复位	0x0715	unit8 replaceReset			0、关闭 1、打开
	更换时间	0x0716	unit16 replaceTime			0-255
	携带提示时间	0x0717	unit16 carryPromptTime			
	睡前提示时间	0x0718	unit8 bedtimePromptTime			
	上室消毒	0x0719	unit8 upDisinfection			
	下室消毒	0x071A	unit8 downDisinfection			
	上室剩余时间	0x071B	unit16 upLefttime			
	下室剩余时间	0x071C	unit16 downLeftime			
	除菌时间	0x071D	unit16 degermingTime			
	热风时间	0x071F	unit16 hotwindTime			
	冷风时间	0x0720	unit16 coolwindTime			
	上层工作状态	0x0721	unit8 upWorkstatus			
	下层工作状态	0x0723	unit8 downWorkstatus			
	消毒提醒	0x0724	unit8 disinfectionReminder			

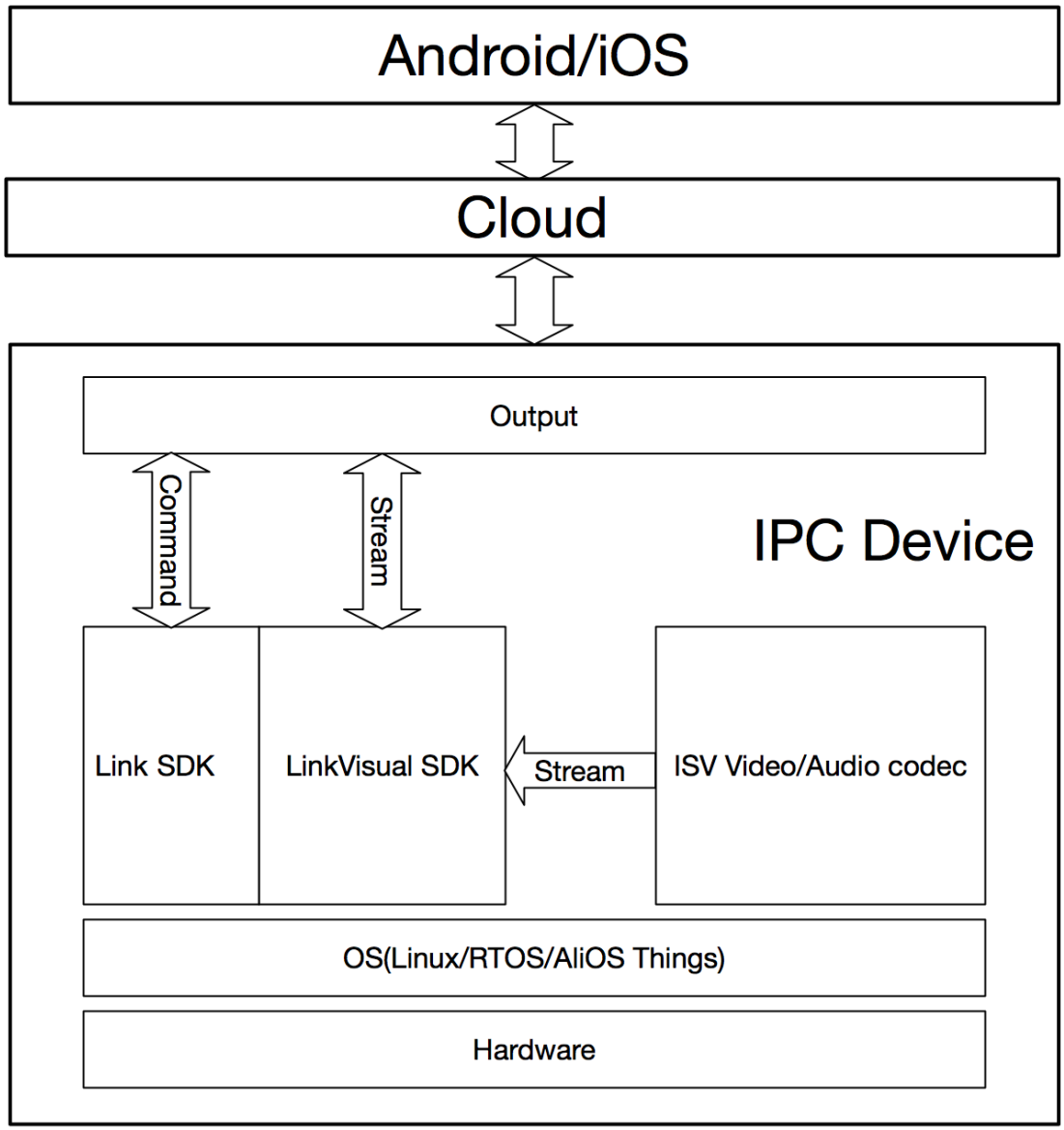
7.Link Visual视频设备开发

7.1. 概述

Link Visual是生活物联网平台提供的视频服务。配合阿里云物联网标准化物模型和Link Visual的设备端SDK，可实现最轻量级的视频设备连云。

产品架构

生活物联网平台SDK提供了设备、云、App的物联通道；Link Visual SDK提供了设备、云、App的音视频传输能力；设备厂商提供音视频流数据及运行环境。产品架构示意图如下图所示。



功能说明

Link Visual提供Android和Linux两种SDK，功能汇总如下（“√”表示已支持，“-”表示无需关注或不涉及）。

功能项	描述	Linux SDK	Android SDK
直播	设备端采集音视频并实时发送至App端。	√	√
点播	将存储在SD卡等外存中的录像文件推到服务端，支持拖动进度条到指定位置播放。	√	√
抓图	抓图分以下两种情况： • App端发起抓图请求，设备抓图后上传至云端。 • 侦测事件触发，设备端自动抓图并上传图片至云端。	√	√
语音对讲	设备端与App端建立双向语音通道，设备端采集录音并实时发送至App端，同时接收到App端发送的语音进行播放。	√	√
云存	将音视频存储在IoT云端，分为事件触发云存和连续云存两种场景。	√	√
生命周期管理	SDK初始化、销毁相关的功能。	√	-
内置功能开关	SDK中内置音视频流、图片数据加密的开关功能，您可以一键开启或关闭该功能。	√	-

版权信息

Link Visual使用了三方librtmp库，相关版权信息声明如下。

```
Copyright (C) 2005-2008 Team XBMC
     http://www.xbmc.org
     Copyright (C) 2008-2009 Andrej Stepanchuk
     Copyright (C) 2009-2010 Howard Chu
 librtmp is free software; you can redistribute it and/or modify
 it under the terms of the GNU Lesser General Public License as
 published by the Free Software Foundation; either version 2.1,
 or (at your option) any later version.
 librtmp is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 GNU General Public License for more details.
 You should have received a copy of the GNU Lesser General Public License
 along with librtmp see the file COPYING.  If not, write to
 the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
 Boston, MA  02110-1301, USA.
 http://www.gnu.org/copyleft/lgpl.html
```

7.2. Link Visual设备端开发-Linux SDK

生活物联网平台提供Linux版本的Link Visual设备端SDK，您可以基于该SDK开发Link Visual视频设备的直播、点播、语音对讲、抓图等功能。

前提条件

- 请您先完成Link Visual Demo体验，以提前熟悉整体流程。体验Link Visual请参见快速体验Link Visual。
- 已完成生活物联网平台SDK的开发。这部分请参见开发指导。

背景信息

Linux版本的Link Visual SDK的资源占用、平台支持和运行依赖如下。

项目	限制条件
资源占用情况	• RAM：1 MB的码流，预计占用500 KB的RAM内存。 • ROM：占用2 MB的ROM存储。
平台支持	目前只能在支持C++11标准（GCC版本4.8.1以上）的Linux平台中使用。
运行依赖	依赖生活物联网平台SDK，您需要同时获取Link Visual SDK和生活物联网平台SDK来完成设备接入。 • 生活物联网平台SDK：主要提供物联网控制通道能力，包括长连接、消息通知、事件上报等。 • Link Visual SDK：主要提供音视频流通道能力，并响应生活物联网平台SDK的控制消息，进行流媒体业务处理。

一、获取Link Visual SDK

Linux版本的Link Visual设备端SDK以静态库的形式提供，支持通过编译SDK接入不同芯片的设备，需要您发送邮件申请。

请按以下模板发送邮件至*fangyu.hfy@alibaba-inc.com*联系我们获取Link Visual设备端SDK。收到邮件后，我们会在收到邮件后的5个工作日内联系您。

- 邮件主题：获取固件升级SDK和操作说明文档
- 邮件内容：

```
公司名称：
公司地址：
联系人：
联系电话：
应用场景描述：
```

> **说明** 我们给您提供Link Visual SDK时会一并提供配套版本的生活物联网平台SDK，您无需单独下载生活物联网平台SDK。

二、了解SDK目录结构及Demo源码

在开发Link Visual SDK前，建议您先了解整个SDK目录结构以及Link Visual Demo的源码，可以帮助您快速熟悉整个开发流程。

- 了解SDK目录结构

  解压获取到的Link Visual SDK压缩包，并查看压缩包的内容。

```
# 解压压缩包，并查看压缩包内容（注：压缩包文件名含有版本等可变信息，执行命令时以实际压缩包名称为准）
$ tar -xf link_visual_ipc_vxxx.tar.gz
$ tree -L 2 link_visual_ipc_vxxx
├── CMakeLists.txt #基于cmake的编译的基础示例
├── linkvisual # lv库和头文件
│   ├── liblink_visual_ipc.a # lv库（注：实际压缩包可能含有动态库）
│   ├── link_visual_def.h # lv头文件
│   └── link_visual_ipc.h # lv头文件
├── sample # 示例代码
│   ├── demo.c # 示例代码的入口文件
│   ├── demo.h # 示例代码的入口头文件
│   ├── ipc_operation # 示例伪造了ipc的功能，相关实现文件
│   └── sdk_api_operation # 示例中lv sdk的相关实现文件
├── third_party # 三方库
│   ├── cJSON-1.7.7.tar.gz # JSON解析库，使用版本为1.7.7
│   ├── libevent-2.1.8-stable.tar.gz # libevent，使用版本为2.1.8
│   └── linkkit-sdk-c.tar.gz # 生活物联网平台SDK，提供生活物联网平台接入能力
└── version.txt  # 版本说明
```

- 了解Demo源码

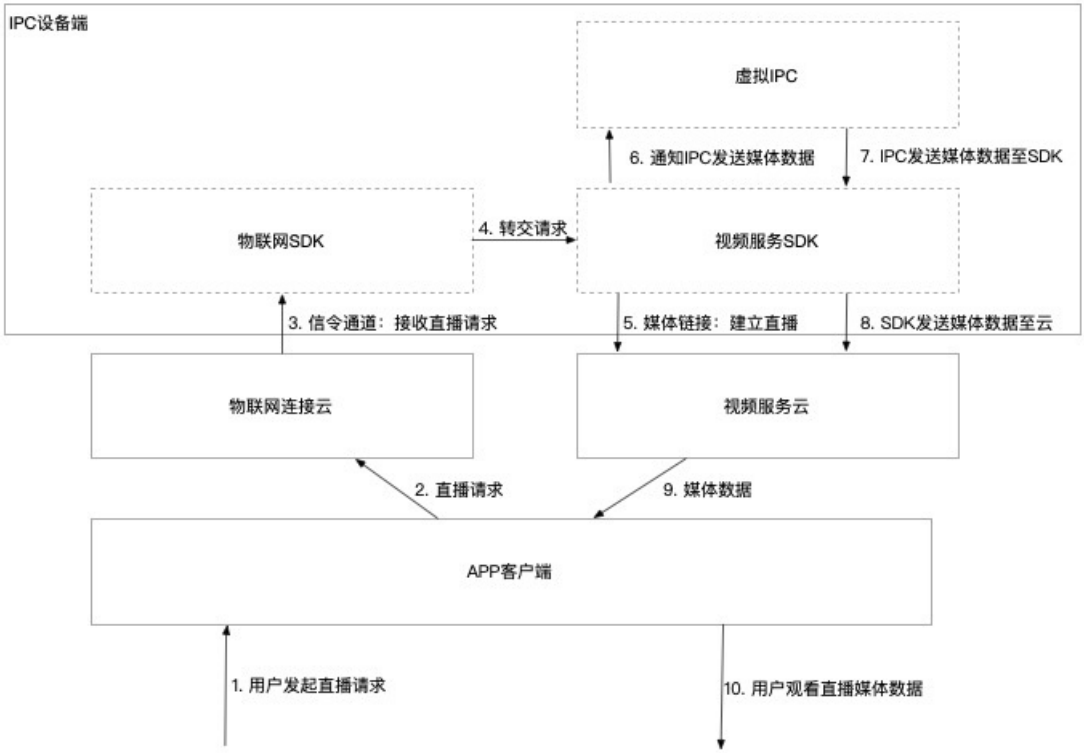
  Link Visual Demo的下载请参见快速体验Link Visual。

```
# 代码取自文件 sample/demo.c
int main(int argc, char* argv[])
{
    char product_key[PRODUCT_KEY_LEN + 1] = {0};
    char device_name[DEVICE_NAME_LEN + 1] = {0};
    char device_secret[DEVICE_SECRET_LEN + 1] = {0};
    char product_secret[PRODUCT_SECRET_LEN + 1] = {0};
    int ret = 0;
#ifdef DUMMY_IPC
    /* 0.初始化虚拟IPC的功能 */
    ret = dummy_ipc_start(&dummy_ipc_config);
#endif // DUMMY_IPC
    /* 1. 初始化LinkVisual的资源 */
    ret = linkvisual_client_init(product_key, device_name, device_secret, (lv_log_level_e
) log_level);
    /* 2. 初始化linkkit的资源，并长连接到服务器 */
    ret = linkkit_client_start(product_key, product_secret, device_name, device_secret);
    /* 3. 运行，等待服务器命令 */
    while(1) {
        usleep(1000 * 500);
    }
    /* 4. linkkit长连接断开，并释放资源 */
    linkkit_client_destroy();
    /* 5. LinkVisual断开音视频连接，并释放资源 */
    linkvisual_client_destroy();
#ifdef DUMMY_IPC
    /* 停止虚拟IPC的功能 */
    dummy_ipc_stop();
#endif // DUMMY_IPC
    return 0;
}
```

如上述代码所示，设备端Link Visual Demo共完成了3个模块的功能。

- 启动了一个虚拟IPC，用于模拟IPC的出流功能。这部分功能仅用于测试，您无需过多关注其实现原理、细节等，实际产品中您需要替换成实际的IPC出流功能。
- 启动了生活物联网平台SDK，建立信令通道，用于接收各类命令，如直播命令等。此外配网、OTA等功能，也属于生活物联网平台SDK功能范畴。
- 启动了视频服务SDK，用于处理视频相关命令，如直播命令等，并建立媒体通道，发送视音频数据。

这3个模块的功能之间的关系如下图所示（以直播数据转发为例）。



三、编译Link Visual SDK

1. 配置环境。

   编译依赖cmake等一些相关软件，这里以Ubuntu 16.04下安装编译环境为例。

   ```
   $ sudo apt-get install -y build-essential make git gcc g++ cmake tree
   ```

2. 编译cJSON。

   i. 进入*third_party*目录，并解压*cJSON-1.7.7*。

   ```
   $ cd third_party
   # 解压代码压缩包
   $ tar -xf cJSON-1.7.7.tar.gz
   $ cd cJSON-1.7.7
   ```

   ii. 在*Makefile*文件中加入工具链的声明，并替换成对应的交叉编译工具链。

   ```
   CC        =   arm-linux-gcc
   LD        =   arm-linux-ld
   AR       =   arm-linux-ar
   ```

iii. 编译cJSON，并确认是否生成了*libcjson.a*，以及头文件*cJSON.h*是否存在。

```
$ make
# 确认libcjosn.a和相关头文件已存在
$ ls lib*.a
$ ls *.h
```

3. 编译libevent。

i. 在*third_party*目录里解压*libevent*。

```
$ cd third_party
# 解压代码压缩包
$ tar -xf libevent-2.1.8-stable.tar.gz
```

ii. 编译libevent。

```
$ cd libevent-2.1.8-stable
# 配置编译条件，注意host和cc需要改为对应的交叉编译工具链信息，编译条件可根据自身需求调整
$ ./configure --host=arm-linux CC=arm-linux-gcc --enable-static --disable-samples --disable-openssl
$ make # 编译
$ ls .libs/libevent.* # 确认是否有libevent.a生成
```

4. 编译生活物联网平台SDK。

i. 在*third_party*目录里解压*ali-smartliving-device-sdk-c*。

```
$ cd third_party
$ tar -xf ali-smartliving-device-sdk-c-1.4.tar.gz;
#解压代码压缩包
$ cd ali-smartliving-device-sdk-c
```

ii. 修改*src/board/config.ubuntu.x86*来准备交叉编译。

```
CROSS_PREFIX :=arm-linux-
#在最后加上CROSS_PREFIX :=交叉编译工具链路径前缀（请替换成对应的交叉编译工具链）
```

iii. 选择Ubuntu编译，并确认生成库*libiot_tls.a/libiot_sdk.a/libiot_hal.a*。

```
# 这里选择ubuntu对应的数字，一般是数字6
$ make reconfig
$ make
# 确认有libiot_tls.a/libiot_sdk.a/libiot_hal.a
$ ls output/lib/*.a
# 确认有如iot_import.h等头文件
$ tree include
```

5. 整理编译。

i. 返回到SDK解压后的文件夹根目录，修改*CMakeLists.txt*里的TOOLCHAINS_PREFIX参数值。

```
set(TOOLCHAINS_PREFIX "arm-linux-" CACHE STRING "set the toolchain")
#第二个参数设置为交叉编译工具链前缀（请替换成对应的交叉编译工具链）
```

ii. 执行编译操作。

```
# 建立一个build文件夹，用于归类编译产物
$ mkdir -p build
# 进入build目录，使用根目录的CMakeLists.txt进行cmake
$ cd build
$ cmake ..
# 编译并安装运行所需相关文件
$ make
$ make install
```

- 编译（compilation）时，您可以添加 `-std=c++11` 来支持C++和C++11。
- 链接（linking）时，您可以在链接选项中添加 `-lstdc++` 来支持C++和C++11。
- 链接（linking）时，库的连接顺序为：*link_visual_ipc*、*iot_sdk cjson*、*iot_hal*、*iot_tls pthread*、*rt*。

6. 运行设备。

```
# 运行方式和Ubuntu Demo或Docker Demo类似，传入设备的激活凭证信息
$ ./link_visual_demo -p your_product_key -n your_product_name -s your_product_secret
```

四、API简述

由于Link Visual SDK依赖与生活物联网平台SDK，开发Link Visual SDK前，请确认已完成生活物联网平台SDK的开发。请参见开发指导。

- 生命周期功能

  生命周期相关的API如下。

功能说明	API名称
SDK初始化	lv_init
SDK停止	lv_destroy

- SDK动态开关等其他功能

功能说明	API名称
生活物联网平台SDK消息注入	lv_linkkit_adapter
SDK功能动态控制	lv_control

- 直播、卡录像点播、云存等功能

功能说明	API名称
通知直播、点播服务已开启	lv_start_push_streaming_cb
通知直播、点播服务已结束	lv_stop_push_streaming_cb
推送视音频配置参数	lv_stream_send_config

功能说明	API名称
推送视频数据	lv_stream_send_video
推送音频数据	lv_stream_send_audio
推流过程中命令控制（暂停等）	lv_on_push_streaming_cmd_cb
点播的文件列表查询	lv_query_storage_record_cb
点播的文件列表按月查询	lv_query_storage_record_by_month_cb
录像文件播放结束	lv_post_file_close
预录数据结束	lv_post_pre_complete

- 抓图功能

功能说明	API名称
图片上传	lv_post_alarm_image
通知上传图片	lv_trigger_pic_capture_cb

- 语音对讲功能

功能说明	API名称
通知开启服务	lv_start_voice_intercom_cb
通知结束服务	lv_stop_voice_intercom_cb
开启服务	lv_voice_intercom_start_service
停止服务	lv_voice_intercom_stop_service
发送音频	lv_voice_intercom_send_audio
接收音频	lv_voice_intercom_receive_data_cb
接收音频参数配置	lv_voice_intercom_receive_metadata_cb

五、API详述-SDK生命周期

SDK生命周期管理相关的API如下。

- lv_init

接口名称	接口详情	描述

接口名称	接口详情	描述
lv_init	int lv_init(const lv_config_s *config)	SDK初始化，该接口的使用说明如下。 ○ lv_init主要完成各个IPC功能的回调，以及一些配置类信息。 ○ lv_init会打印相关版本号信息，便于您追溯问题。 ○ 初始化失败时，一般不会与网络有关，请优先检查入参是否正确或者资源分配是否成功。 ○ lv_init注册了大量的回调函数，这些回调函数共用一个消息队列线程，因此建议您不要在回调中做过于耗时的操作。 ○ lv_init定义了日志类型log_level，建议对接初期将日志类型设置为LV_LOG_DEBUG。 ○ 推荐您在调用lv_init成功后再调用IOT_Linkkit_Connect。

请求参数说明如下。

参数	类型	说明
config	lv_config_s*	配置参数结构体。

示例代码如下。

```
//新建一个配置结构体，并置空
lv_config_s config;
memset(&config, 0, sizeof(lv_config_s));
//以下是设置具体的配置属性
//设备三元组
memcpy(config.product_key, product_key.c_str(), product_key.length());
memcpy(config.device_name, device_name.c_str(), device_name.length());
memcpy(config.device_secret, device_secret.c_str(), device_secret.length());
//SDK的日志配置
config.log_level = LV_LOG_DEBUG;
config.log_dest = LV_LOG_DESTINATION_STDOUT;
//config.log_dest_file;
//音视频推流服务
config.start_push_streaming_cb = startPushStreamingCallback;
config.stop_push_streaming_cb = stopPushStreamingCallback;
config.on_push_streaming_cmd_cb = onPushStreamingCmdCb;
//语音对讲服务
config.start_voice_intercom_cb = startVoiceIntercomCallback;
config.stop_voice_intercom_cb = stopVoiceIntercomCallback;
config.voice_intercom_receive_metadata_cb = voice_intercom_receive_metadata_cb;
config.voice_intercom_receive_data_cb = VoiceIntercomReceiveDataCallback;
//获取存储录像录像列表
config.query_storage_record_cb = queryStorageRecordCallback;
//触发设备抓图
config.trigger_pic_capture_cb = triggerPicCaptureCallback;
//触发设备录像
config.trigger_record_cb = triggerRecordCallback;
//初始化SDK，失败则退出
int ret = lv_init(&config);
if (ret < 0) {
    return -1;
}
```

- lv_destroy

接口名称	接口详情	描述
lv_destroy	int lv_destroy(void);	SDK销毁，该接口的使用说明如下。 ○ 推荐在调用lv_destroy前调用IOT_Linkkit_Close。 ○ 调用lv_destroy时，耗时会高达数秒。在正常的业务逻辑内，不建议您反复调用lv_destroy。

无请求参数。

示例代码如下。

```
//运行结束后
lv_destroy();
```

六、API详述-视频播放功能

视频播放分为直播、卡录像点播、云储存播放，在视音频数据传输的过程，SDK采用了同一套API来实现视频播放。相关API如下。

- lv_start_push_streaming_cb

接口名称	接口详情	描述
lv_start_push_streaming_cb	typedef int (lv_start_push_streaming_cb)(int service_id, lv_stream_type_e type, const lv_stream_param_s *param)	回调函数，通知视频播放链路已经建立成功，并附带一些配置信息。收到此回调后，您应该根据配置信息初始化编码器，并开始推送音视频数据。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
type	lv_stream_type_e	链路的类型，分为直播和卡录像点播。
param	const lv_stream_param_s*	链路的参数，例如卡录像要点播的文件名称。

示例代码如下。

```
//demo，定义了回调函数start_push_streaming_cb作为lv_start_push_streaming_cb的实现
lv_start_push_streaming_cb = start_push_streaming_cb;
//demo,回调函数startPushStreamingCallback
int start_push_streaming_cb(int service_id, lv_stream_type_e cmd_type, const lv_stream_pa
ram_s *param)
{
    if (cmd_type == LV_STREAM_CMD_LIVE) {
        //使用lv_stream_send_video/lv_stream_send_audio推送音视频数据;
        //实际使用中建议新建线程进行数据发送
        ......
        return 0;
    } else if (cmd_type == LV_STREAM_CMD_STORAGE_RECORD_BY_FILE) {
       //使用lv_stream_send_video/lv_stream_send_audio推送音视频数据
       //实际使用中建议新建线程进行数据发送
       ......
       return 0;
    }
    return 0;
}
```

- lv_stop_push_streaming_cb

接口名称	接口详情	描述
lv_stop_push_streaming_cb	typedef int (*lv_stop_push_streaming_cb)(int service_id)	回调函数，通知视频播放链路已经断开。收到此回调后，您应该停止推送音视频数据。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

示例代码如下。

```
//demo，定义了回调函数stop_push_streaming_cb作为lv_start_push_streaming_cb的实现
lv_stop_push_streaming_cb = stop_push_streaming_cb;
//demo,回调函数stop_push_streaming_cb
int stop_push_streaming_cb(int service_id)
{
    if (service_id == g_live_service_id) {
        //您停止推流
    } else if (service_id == g_storage_record_service_id) {
       //您停止推流
    }
    return 0;
}
```

- lv_on_push_streaming_cmd_cb

接口名称	接口详情	描述
lv_on_push_streaming_cmd_cb	typedef int (*lv_on_push_streaming_cmd_cb)(int service_id, lv_on_push_streaming_cmd_type_e cmd, const lv_on_push_streaming_cmd_param_s *param)	回调函数，通知视频播放链路建立期间，可以进行一些命令控制，例如要求发送I帧命令。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
cmd	lv_on_push_streaming_cmd_type_e	控制命令，目前有开始播放、停止播放、暂停播放、恢复播放、定位、强制I帧。
param	lv_on_push_streaming_cmd_param_s *	参数值和lv_on_push_streaming_cmd_type_e有关

示例代码如下。

```
//demo，定义了回调函数on_push_streaming_cmd_cb作为lv_on_push_streaming_cmd_cb的实现
on_push_streaming_cmd_cb = on_push_streaming_cmd_cb;
int on_push_streaming_cmd_cb(int service_id, lv_on_push_streaming_cmd_type_e cmd, const l
v_on_push_streaming_cmd_param_s *param)
{
    if (cmd == LV_STORAGE_RECORD_START) {
        //卡录像开始推流
    } else if (cmd == LV_STORAGE_RECORD_STOP) {
        //卡录像停止推流
    } else if (cmd == LV_STORAGE_RECORD_PAUSE) {
        //卡录像暂停推流
    } else if (cmd == LV_STORAGE_RECORD_UNPAUSE) {
        //卡录像恢复推流
    } else if (cmd == LV_STORAGE_RECORD_SEEK) {
        //卡录像定位到某一段
    } else if (cmd == LV_LIVE_REQUEST_I_FRAME) {
        //对于直播，需要强制生成一个I帧
    }
    return 0;
}
```

- lv_stream_send_config

接口名称	接口详情	描述
lv_stream_send_config	int lv_stream_send_config(int service_id, unsigned int bitrate_kbps, double duration, const lv_video_param_s *video_param, const lv_audio_param_s *audio_param)	配置发送音视频的参数。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
bitrate_kbps	unsigned int	链路的码流，设置的码流越大，内部音视频数据缓冲区越大。
duration	double	文件时长，卡录像点播有效。
video_param	const lv_video_param_s*	视频的参数集。
audio_param	const lv_audio_param_s*	音频的参数集。

示例代码如下。

```
lv_video_param_s video_param;
memset(&video_param, 0, sizeof(lv_video_param_s));
video_param.format = LV_VIDEO_FORMAT_H264;
video_param.fps = 25;
lv_audio_param_s audio_param;
memset(&audio_param, 0, sizeof(lv_audio_param_s));
audio_param.format = LV_AUDIO_FORMAT_G711A;
audio_param.channel = LV_AUDIO_CHANNEL_MONO;
audio_param.sample_bits = LV_AUDIO_SAMPLE_BITS_16BIT;
audio_param.sample_rate = LV_AUDIO_SAMPLE_RATE_8000;
lv_stream_send_config(service_id, 1000, 0, &video_param, &audio_param);
```

- lv_stream_send_video

接口名称	接口详情	描述
lv_stream_send_video	int lv_stream_send_video(int service_id, lv_video_format_e format,unsigned char* buffer, unsigned int buffer_len, bool key_frame, unsigned int timestamp_ms)	发送视频数据。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
format	lv_video_format_e	视频编码类型
buffer	unsigned char*	视频的数据。
buffer_len	unsigned int	视频的数据长度。
key_frame	bool	是否为关键帧。I帧为关键帧，P帧为非关键帧，不接受B帧。
timestamp_ms	unsigned int	视频的时间戳。

示例代码如下。

```
//demo，这个函数回调输入视频帧数据
void linkvisual_client_video_handler(int service_id, lv_video_format_e format, unsigned c
har *buffer, unsigned int buffer_size,
                                     unsigned int present_time, int nal_type) {
    lv_stream_send_video(service_id, format, buffer, buffer_size, nal_type, present_time)
;
}
```

- lv_stream_send_audio

接口名称	接口详情	描述
lv_stream_send_audio	int lv_stream_send_audio(int service_id, lv_audio_format_e format,unsigned char* buffer, unsigned int buffer_len, unsigned int timestamp_ms)	发送音频数据。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
format	lv_audio_format_e	音频编码类型
buffer	unsigned char*	音频的数据。
buffer_len	unsigned int	音频的数据长度。
timestamp_ms	unsigned int	音频的时间戳。

示例代码如下。

```
//demo，这个函数回调输入音频帧数据
void linkvisual_client_audio_handler(int service_id,
                                     lv_audio_format_e format,
                                     unsigned char *buffer,
                                     unsigned int buffer_size,
                                     unsigned int present_time) {
    lv_stream_send_audio(service_id, format, buffer, buffer_size, present_time);
}
```

- lv_query_storage_record_cb

接口名称	接口详情	描述
lv_query_storage_record_cb	typedef void (lv_query_storage_record_cb)(unsigned int start_time, unsigned int stop_time, int num, const char *id, int (on_complete)(int num, const char *id, const lv_query_storage_record_response_s *response))	回调函数，查询卡录像的文件信息。

请求参数说明如下。

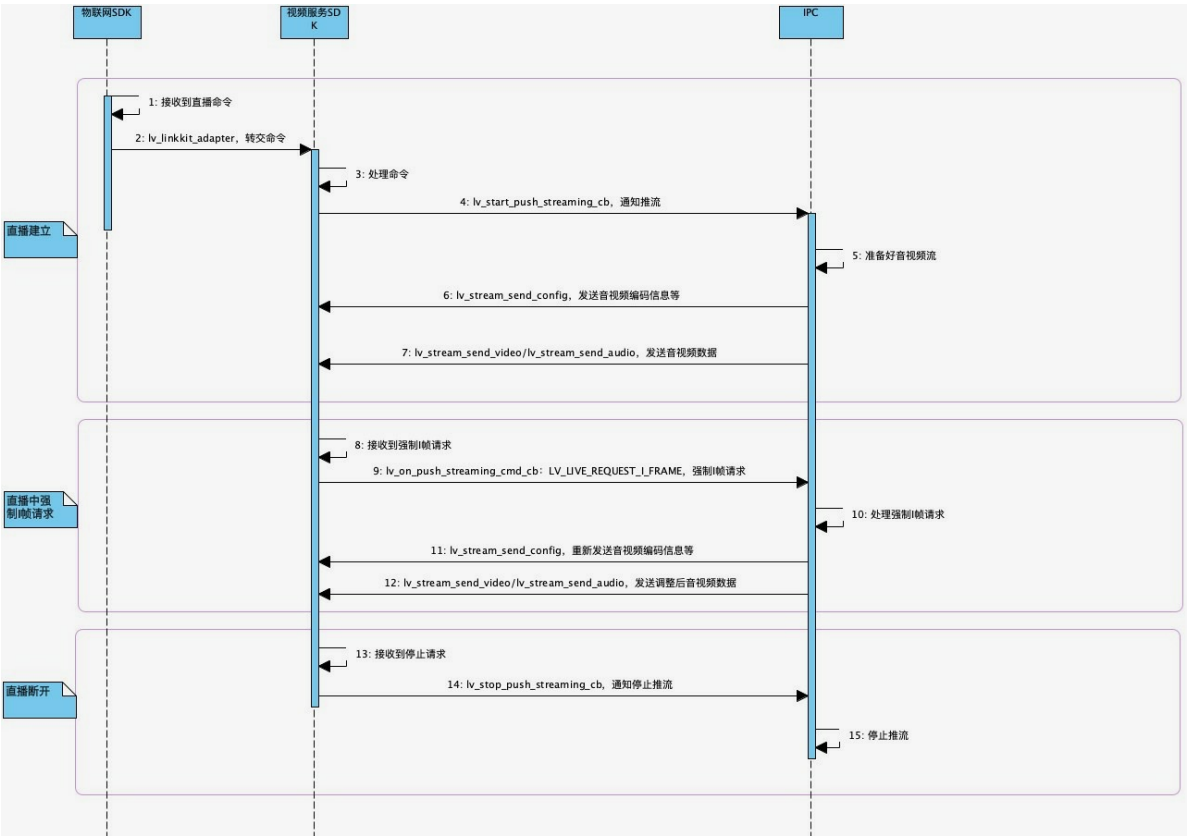
参数	类型	说明
start_time	unsigned int	查询录像的开始时间。
stop_time	unsigned int	查询录像的结束时间。

参数	类型	说明
num	int	录像数量小于等于0的时候表示已查询全部的录像。
id	const char *	查询的会话ID，需要回传。
on_complete	函数指针	将查询结果通过该函数同步返回。

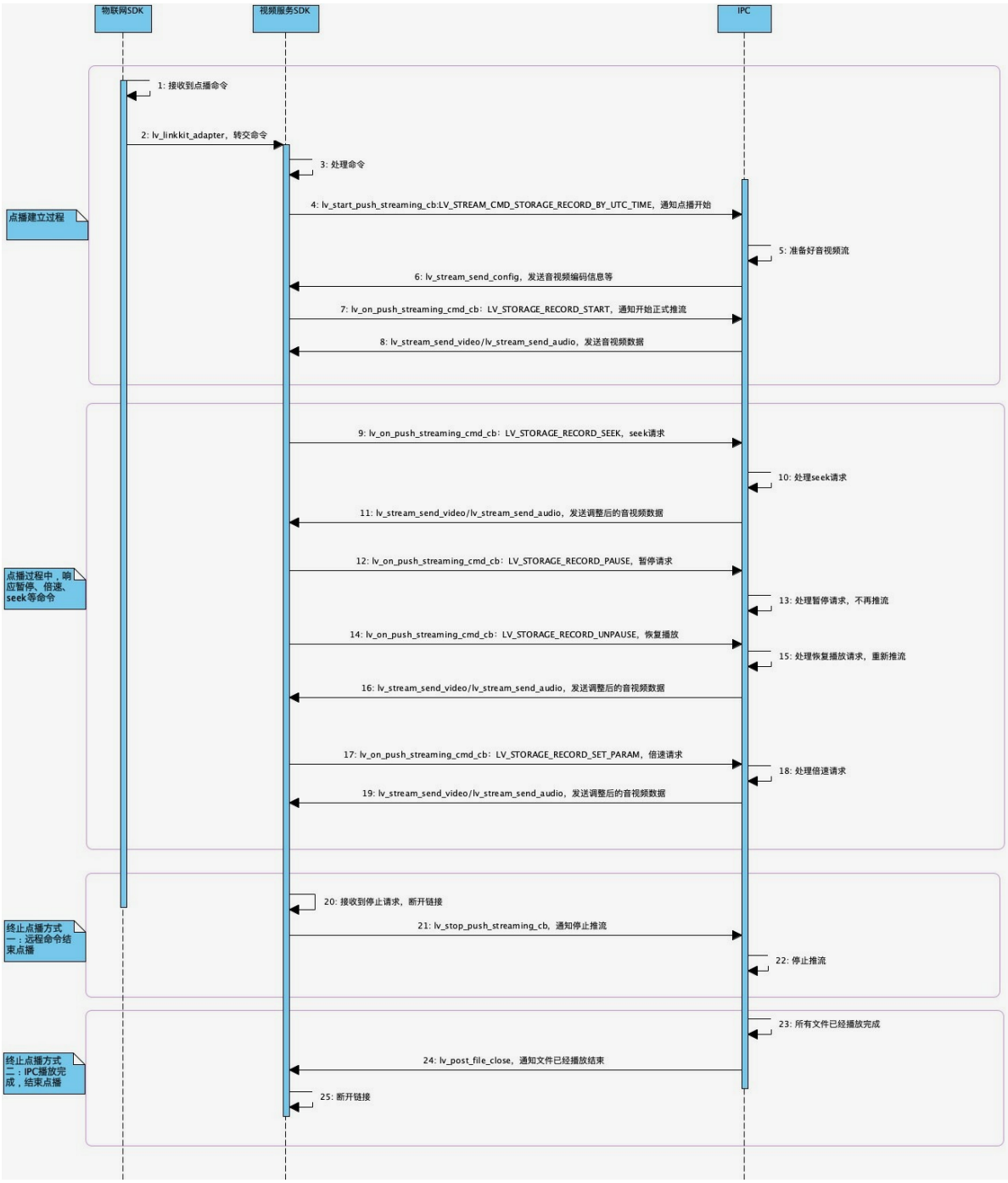
示例代码如下。

```
int dummy_ipc_report_vod_list(unsigned int start_time,
                              unsigned int stop_time,
                              int num,
                              const char *id,
                              int (*on_complete)(int num,
                                                 const char *id,
                                                 const lv_query_storage_record_response_s
*response)) {
    int answer_num = g_vod_media_file_group.size();
    auto *response = new lv_query_storage_record_response_s[answer_num];
    memset(response, 0, sizeof(lv_query_storage_record_response_s) * answer_num);
    double duration = 0;
    for (int i = 0; i < answer_num; i ++) {
        MediaParse::GetDuration(g_vod_media_file_group[i], duration);
        response[i].file_size = 0;
        response[i].record_type = LV_STORAGE_RECORD_INITIATIVE;
        snprintf(response[i].file_name, 64, g_vod_media_file_group[i].c_str(), i);//注意不
要溢出
        if (i == 0) {
            response[i].start_time = (int)start_time;
            response[i].stop_time = (int)start_time + (int)duration;
        } else {
            response[i].start_time = response[i - 1].stop_time;
            response[i].stop_time = response[i - 1].stop_time + (int)duration;
        }
    }    int result = on_complete(answer_num, id, response);
    printf("dummy_ipc_report_vod_list result: %d\n", result);
    delete[] response;
    return 0;
}
```

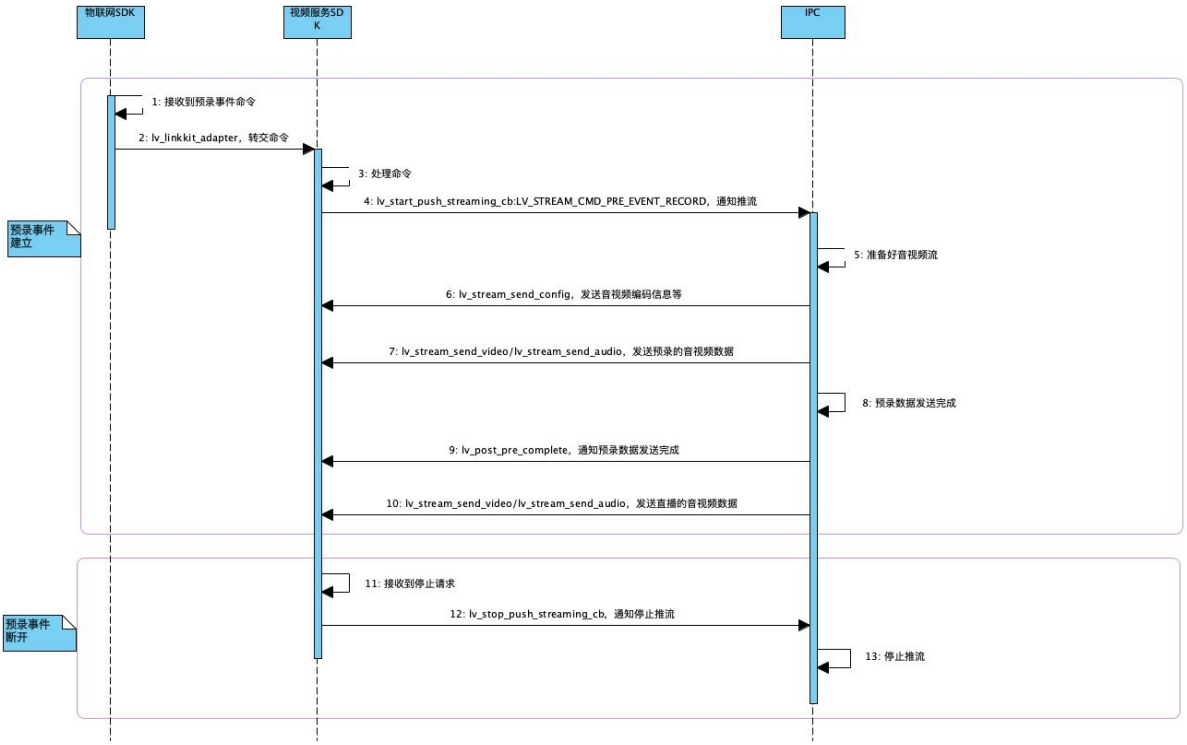
- 直播流程图


物联网SDK
视频服务SDK
IPC
1: 接收到直播命令
2: lv_linkkit_adapter，转交命令
3: 处理命令
4: lv_start_push_streaming_cb，通知推流
直播建立
5: 准备好音视频流
6: lv_stream_send_config，发送音视频编码信息等
7: lv_stream_send_video/lv_stream_send_audio，发送音视频数据
8: 接收到强制I帧请求
9: lv_on_push_streaming_cmd_cb：LV_LIVE_REQUEST_I_FRAME，强制I帧请求
直播中强制I帧请求
10: 处理强制I帧请求
11: lv_stream_send_config，重新发送音视频编码信息等
12: lv_stream_send_video/lv_stream_send_audio，发送调整后音视频数据
13: 接收到停止请求
14: lv_stop_push_streaming_cb，通知停止推流
直播断开
15: 停止推流

- 卡录像点播流程图


物联网SDK
视频服务SDK
IPC
1: 接收到点播命令
2: lv_linkkit_adapter，转交命令
3: 处理命令
4: lv_start_push_streaming_cb:LV_STREAM_CMD_STORAGE_RECORD_BY_UTC_TIME，通知点播开始
点播建立过程
5: 准备好音视频流
6: lv_stream_send_config，发送音视频编码信息等
7: lv_on_push_streaming_cmd_cb：LV_STORAGE_RECORD_START，通知开始正式推流
8: lv_stream_send_video/lv_stream_send_audio，发送音视频数据
9: lv_on_push_streaming_cmd_cb：LV_STORAGE_RECORD_SEEK，seek请求
10: 处理seek请求
11: lv_stream_send_video/lv_stream_send_audio，发送调整后的音视频数据
12: lv_on_push_streaming_cmd_cb：LV_STORAGE_RECORD_PAUSE，暂停请求
点播过程中，响应暂停、倍速、seek等命令
13: 处理暂停请求，不再推流
14: lv_on_push_streaming_cmd_cb：LV_STORAGE_RECORD_UNPAUSE，恢复播放
15: 处理恢复播放请求，重新推流
16: lv_stream_send_video/lv_stream_send_audio，发送调整后的音视频数据
17: lv_on_push_streaming_cmd_cb：LV_STORAGE_RECORD_SET_PARAM，倍速请求
18: 处理倍速请求
19: lv_stream_send_video/lv_stream_send_audio，发送调整后的音视频数据
20: 接收到停止请求，断开链接
21: lv_stop_push_streaming_cb，通知停止推流
终止点播方式一：远程命令结束点播
22: 停止推流
23: 所有文件已经播放完成
24: lv_post_file_close，通知文件已经播放结束
终止点播方式二：IPC播放完成，结束点播
25: 断开链接

- 预录事件点播流程图



- 云存

  云存分为事件触发云存和连续云存两种场景。您基于SDK实现直播功能即可同时实现云存功能，无需您额外开发。

视频播放功能的接口调用说明如下。

- 强制I帧命令发出时，为了保证尽可能快速的发出I帧，您应尽可能快的产生I帧，建议在300毫秒（ms）内完成
- 强制I帧命令发出时，您需要重新调用lv_stream_send_config发送流配置。
- 进行卡录像的定位操作时，为了尽可能快速的显示定位后的数据，您在定位操作后，应该尽可能快速的发出I帧，同时SDK在定位操作后也不在接收音频数据和视频的P帧数据，直到I帧到达。
- H264/H265的帧结构会有一定的要求，可以打印I帧的前256个字节进行查看，打印代码如下。

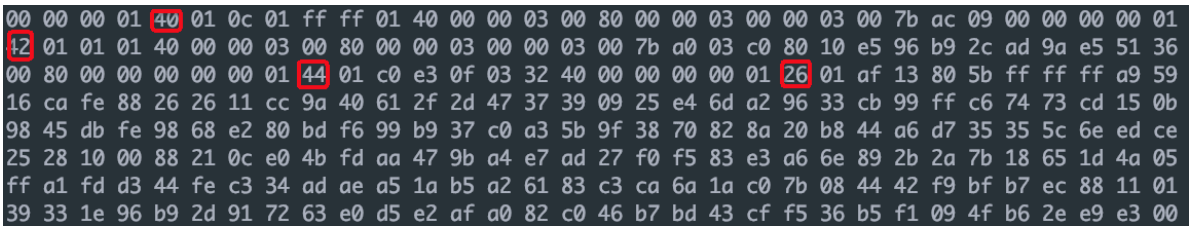
```
for (int i = 0; i < ((buffer_size > 256)?256:buffer_size); i++) {
    printf("%02x ", buffer[i]);
    if ((i + 1) % 30 == 0) {
        printf("\n");
    }
}
printf("\n");
```

H264要求I帧为：`帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR`。如下图所示。

```
00 00 01 67 42 c0 1e d9 00 a0 2f f9 3f f2 57 92 58 01 00 00 03 00 01 00 00 03 00 32 0f 16
2e 48 00 00 00 01 68 cb 8c b2 00 00 01 65 88 84 1f c9 81 0b 40 00 24 c7 19 c5 87 40 65 89
c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 ff ff
77 c2 a1 6e 00 16 c4 d5 6f e8 1b 0d 2c af b8 00 d1 72 9b 8d 81 28 7d 2c 8c 18 5d 50 9d 09
8e 79 94 b4 d8 9d e9 7e 0b ad 99 ca 06 a7 ff c7 84 56 15 91 7b 50 01 9f a0 8a f2 75 64 67
ab c2 b2 11 61 6f 23 00 03 4a d5 93 e4 1c 11 8b 2f f8 23 0d 26 cf d4 01 5a 2c c2 67 23 bd
1d 0e ff bf ff f0 42 5e 00 09 9a 88 33 36 eb 90 00 0f e0 33 ff 80 60 b0 42 4e 00 13 59 0a
ef 4a fc 89 bf ff 80 42 bc 30 00 4c e9 0c 89 ba e4 00 1f 40 8c 1a ff f0 f5 85 78 00 73 93
75 f4 bc fc 00 0f 17 21 9c e2 40 7f cc 3c c0
```

0x000001或者0x00000001是帧分隔符，0x67是SPS的开始，0x68是PPS的开始，0x65是IDR的开始。

H265要求I帧为：`帧分隔符+VPS+帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR`，如下图所示。



> **说明** 音频目前支持G711A/LC-AAC编码方式，编码参数的支持需要参考头文件*link_visual_def.h*中的宏定义值。
>
> 同一路流切换视频码流时（如主码流切为子码流、修改码流分辨率、H264/H265切换等），请保证切换后的第一帧为I帧，否则会引发花屏等现象。

七、API详述-图片服务

图片服务相关的API如下。

- lv_trigger_pic_capture_cb

接口名称	接口详情	描述
lv_trigger_pic_capture_cb	typedef int (lv_trigger_pic_capture_cb)(const char *id)	通知设备开始抓取图片。

请求参数说明如下。

参数	类型	说明
id	const char *	本次请求的ID，需要回传。

- lv_post_alarm_image

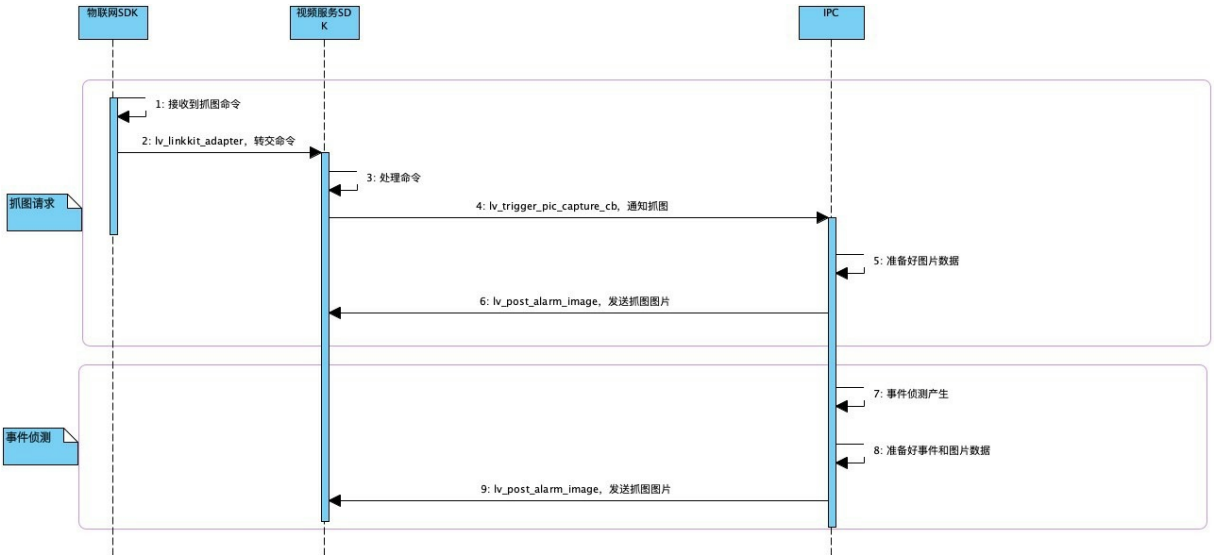
接口名称	接口详情	描述
lv_post_alarm_image	int lv_post_alarm_image(const char *buffer, int buffer_len, lv_event_type_e type, const char *id)	发送图片和报警事件，适用于抓图回调后进行上报，或者设备主动发起上报。

请求参数说明如下。

参数	类型	说明
buffer	const char *	图片数据。
buffer_len	int	图片数据长度。
type	lv_event_type_e	上报类型，分为抓图回调后进行上报和设备主动发起上报。

参数	类型	说明
id	const char *	抓图回调的ID回传，主动上报时传空字符串或者NULL。

抓图功能分为App端触发抓图请求和事件侦测触发设备端主动抓图两种场景。



调用图片服务相关接口的说明如下。

- 上传的最大图片大小为1MB。
- 图片上传的最小间隔为1秒，频繁触发的图片将会被丢弃。
- 图片会由SDK内部保存一份拷贝，并形成待发送的图片队列。图片队列的长度为5，在网络差的情况下，图片队列可能会满，满队列后新生成的图片将会被丢弃。
- SDK不限制图片的格式，只要云端或者从云端拉取图片的设备能够支持解析即可。推荐使用常见的图片格式，如*jpg*格式。

八、API详述-语音对讲服务

语音对讲服务相关的API如下。

- lv_start_voice_intercom_cb

接口名称	接口详情	描述
lv_start_voice_intercom_cb	typedef int (*lv_start_voice_intercom_cb)(int service_id)	回调函数，返回一路语音对讲服务链路已可以建立的通知。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

- lv_voice_intercom_start_service

接口名称	接口详情	描述
lv_voice_intercom_start_service	int lv_voice_intercom_start_service(int service_id, const lv_audio_param_s *audio_param)	建立一路语音对讲链路，在收到语音对讲开始的回调后，调用该接口。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
audio_param	const lv_audio_param_s *	设备的音频参数结构体。

示例代码如下。

```
//demo，start_voice_intercom_cb是lv_start_voice_intercom_cb的实际实现
lv_start_voice_intercom_cb = start_voice_intercom_cb;
int start_voice_intercom_cb(int service_id)
{
    //收到开始语音对讲请求时，主动开启对讲服务
    lv_audio_param_s audio_param;
    memset(&audio_param, 0, sizeof(lv_audio_param_s));
    audio_param.format = LV_AUDIO_FORMAT_G711A;
    audio_param.channel = LV_AUDIO_CHANNEL_MONO;
    audio_param.sample_bits = LV_AUDIO_SAMPLE_BITS_16BIT;
    audio_param.sample_rate = LV_AUDIO_SAMPLE_RATE_8000;
    int ret = lv_voice_intercom_start_service(service_id, &audio_param);
    return 0;
}
```

- lv_stop_voice_intercom_cb

接口名称	接口详情	描述
lv_stop_voice_intercom_cb	typedef int (*lv_stop_voice_intercom_cb)(int service_id)	回调函数，返回一路语音对讲服务链路已可以断开的通知。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

- lv_voice_intercom_stop_service

接口名称	接口详情	描述
lv_voice_intercom_stop_service	int lv_voice_intercom_stop_service(int service_id)	断开一路语音对讲链路，在收到语音对讲结束的回调后，调用该接口。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

示例代码如下。

```
//demo，stop_voice_intercom_cb是lv_start_voice_intercom_cb的实际实现
lv_stop_voice_intercom_cb = stop_voice_intercom_cb;
int stop_voice_intercom_cb(int service_id)
{
    lv_voice_intercom_stop_service(service_id);
}
```

- lv_voice_intercom_receive_metadata_cb

接口名称	接口详情	描述
lv_voice_intercom_receive_metadata_cb	typedef int (lv_voice_intercom_receive_metadata_cb)(int service_id, const lv_audio_param_s *audio_param)	回调函数，通知语音对讲时，App发送的音频参数格式。

请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
audio_param	const lv_audio_param_s *	APP发送的音频格式结构体。

- lv_voice_intercom_receive_data_cb

接口名称	接口详情	描述
lv_voice_intercom_receive_data_cb	typedef void (lv_voice_intercom_receive_data_cb)(const char *buffer, unsigned int buffer_len)	回调函数，语音对讲时，App发送的音频数据。

请求参数说明如下。

参数	类型	说明
buffer	const char *	APP发送的音频数据。
buffer_len	int	APP发送的音频数据长度。

- lv_voice_intercom_send_audio

接口名称	接口详情	描述
lv_voice_intercom_send_audio	int lv_voice_intercom_send_audio(int service_id, const char *buffer, int buffer_len , unsigned int timestamp_ms)	向其中一路语音对讲链路发送音频数据。

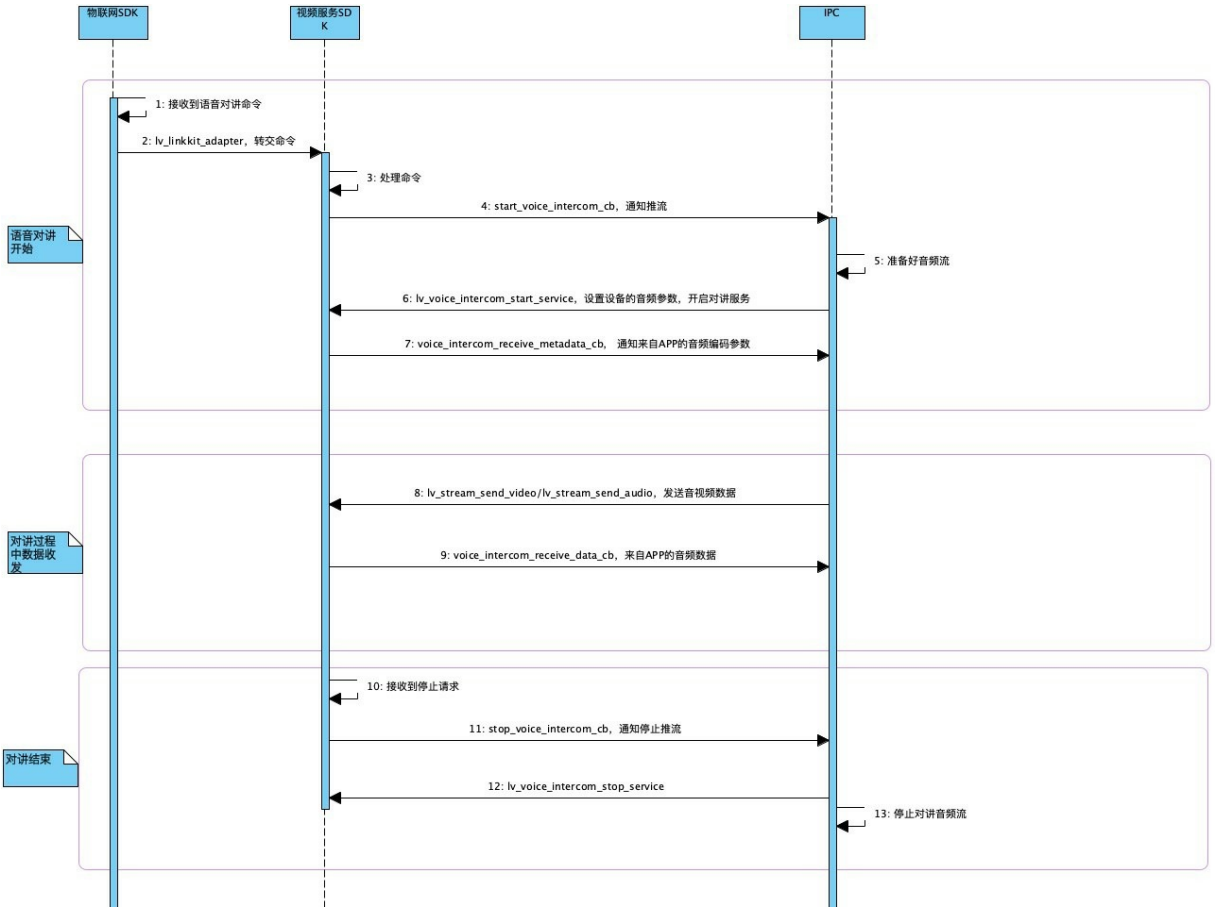
请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
buffer	const char *	设备音频数据。
buffer_len	int	设备音频数据长度。
timestamp_ms	unsigned int	设备音频时间戳。

示例代码如下。

```
void linkvisual_client_audio_handler(int service_id,
                                     lv_audio_format_e format,
                                     unsigned char *buffer,
                                     unsigned int buffer_size,
                                     unsigned int present_time) {
    lv_voice_intercom_send_audio(g_voice_intercom_service_id, buffer, buffer_size, presen
t_time);
}
```

语音对讲流程图如下：



九、后续操作

生成设备固件后，您可以将固件烧录到设备中，并使用App调试Link Visual功能。建议您直接使用公版App来调试。公版App的下载请参见云智能App介绍。

7.3. Android SDK开发指南

生活物联网平台提供Android版本的设备端Link Visual SDK，您可以基于该SDK开发Link Visual视频设备的直播、点播、语音对讲、抓图等功能。

Android设备端Link Visual SDK依赖如下。

依赖SDK	概述
Link Kit Android SDK	提供设备与云端的双向数据通道能力。

获取SDK

请您根据引入依赖的方法来获取Android版本的设备端Link Visual SDK，并在*proguard-rules.pro*文件中排除不需要被混淆的类和方法。

- 引入依赖

```
// 1. 根build.gradle添加对aliyun maven仓库的引用
allprojects {
    repositories {
        maven {
            url "http://maven.aliyun.com/nexus/content/repositories/releases"
        }
    }
}
// 2. app build.gradle中添加依赖
implementation 'com.aliyun.iotx:linkvisual-ipc:1.4.4'
```

- 混淆配置

```
# keep linkvisual
-keep class com.aliyun.iotx.linkvisualipc.** { *; }
```

初始化SDK

请在Link SDK初始化完毕后再初始化Link Visual SDK，初始化时需要传入设备证书信息。

```
LinkKit.getInstance().init(this, params, new ILinkKitConnectListener() {
            @Override
            public void onError(AError error) {
                Log.d(TAG,
                    "onError() called with: error = [" + (error == null ? "null" : (error.g
etCode() + error.getMsg()))
                        + "]");
            }
            @Override
            public void onInitDone(Object data) {
                Log.d(TAG, "onInitDone() called with: data = [" + JSON.toJSONString(data) +
"]");
                // 初始化SDK
                IPCDev.getInstance().init(context, your_productKey, your_deviceName, your_d
eviceSecret);
            }
```

其中*your_productname*、*your_devicename*、*your_devicesecret*需要替换为您自己的设备证书信息。

Link Visual SDK需借助Link kit的能力来完成消息监听和处理。注册监听的流程如下。

1. 在设备服务中注册异步服务调用监听器。

```
//注册异步服务调用监听器
LinkKit.getInstance().getDeviceThing().setServiceHandler(service.getIdentifier(),
                itResRequestHandler);
//异步服务调用监听器
private ITResRequestHandler itResRequestHandler = new ITResRequestHandler() {
        @Override
        public void onProcess(String identify, Object result, ITResResponseCallback
            itResResponseCallback) {
            Log.d(TAG,
                "ITResRequestHandler  onProcess() called with: identify = [" + identify
+ "], result = ["
                     + JSON.toJSONString(result) + "], itResResponseCallback = ["
                     + itResResponseCallback + "]");
            /**
             * 添加SDK对异步服务调用的监听
             */
            IPCDev.getInstance().notifyAsyncTopicReceived(identify, result, itResRespon
seCallback);
        }
        @Override
        public void onSuccess(Object o, OutputParams outputParams) {
            Log.d(TAG,
                "onSuccess() called with: o = [" + JSON.toJSONString(o) + "], outputPar
ams = [" + JSON
                     .toJSONString(outputParams) + "]");
        }
        @Override
        public void onFail(Object o, ErrorInfo errorInfo) {
            Log.d(TAG, "onFail() called with: o = [" + JSON.toJSONString(o) + "], error
Info = [" + JSON
                .toJSONString(errorInfo) + "]");
        }
    };
```

2. 注册同步服务调用的监听器。

```
/**
 * 注册同步服务调用的监听器
 */
LinkKit.getInstance().registerOnPushListener(connectNotifyListener);
//同步服务调用监听器
private IConnectNotifyListener connectNotifyListener = new IConnectNotifyListener() {
        @Override
        public void onNotify(String connectId, String topic, AMessage aMessage) {
            /**
             * 添加SDK对同步服务调用的监听
             */
            IPCDev.getInstance().notifySyncTopicReceived(connectId, topic, aMessage);
            if (CONNECT_ID.equals(connectId) && !TextUtils.isEmpty(topic) &&
                topic.startsWith("/sys/" + productKey + "/" + deviceName + "/rrpc/reque
st")) {
                Log.d(TAG, "IConnectNotifyListener   onNotify() called with: connectId
= [" + connectId + "], topic = ["
                    + topic + "], aMessage = ["
                    + new String((byte[])aMessage.data) + "]");
            }
        }
        @Override
        public boolean shouldHandle(String connectId, String topic) {
            return true;
        }
        @Override
        public void onConnectStateChange(String connectId, ConnectState connectState) {
        }
    };
```

开发直播功能

直播通过RTMP推流，视频支持H264、H265，音频支持G711a以及AAC_LC格式。您可以按以下步骤开发直播功能的推流。

1. 注册直播事件监听器和流错误监听器。

   Link Visual SDK收到服务端下发的开始推流指令后，会通过事先注册的直播流事件监听器（OnLiveStreamListener）来通知何时开始或结束推流、强制I帧等。该流程开发如下所示。

   i. 设置直播流事件监听。

   ```
   // 设置直播流事件监听
   IPCDev.getInstance().getIpcStreamManager().setOnLiveStreamListener(MainActivity.thi
   s);
   // 设置流错误监听
   IPCDev.getInstance().getIpcStreamManager().setOnStreamErrorListener(MainActivity.th
   is);
   ```

ii. 接收服务端发送的开始推直播流请求。

```
public interface OnLiveStreamListener {
    /**
     * 收到开始推直播流请求
     *
     * @param streamId   流ID
     * @param streamType 码流类型：0为主码流，1为辅码流
     * @param preTimeInS 预先录制时间，单位S
     */
    void onStartPushLiveStreaming(final int streamId, final int streamType, final i
nt preTimeInS);
    /**
     * 收到停止推流请求
     *
     * @param streamId 流ID
     */
    void onStopPushStreaming(final int streamId);
    /**
     * 收到强制I帧请求
     * 需立即构造一个I帧并发送
     * @param streamId 流ID
     */
    void onForceIFrame(int streamId);
}
```

推流中发生的错误也将通过流错误监听器来通知。请不要在回调接口中执行阻塞任务。

```
public interface OnStreamErrorListener {
    /**
     * 流异常时回调
     * @param streamId
     * @param error 参考StreamError定义
     */
    void onError(int streamId, StreamError error);
}
```

流错误码如下所示。

错误码	标志符	错误描述
1	StreamError.ERROR_STREAM_CREATE_FAILED	创建流实例失败。
2	StreamError.ERROR_STREAM_START_FAILED	开流失败。
3	StreamError.ERROR_STREAM_STOP_FAILED	停止流失败。
4	StreamError.ERROR_STREAM_SEND_VIDEO_FAILED	发送视频数据失败。

错误码	标志符	错误描述
5	StreamError.ERROR_STREAM_SEND_AUDIO_FAILED	发送音频数据失败。
6	StreamError.ERROR_STREAM_INVALID_PARAMS	无效的流参数。

RTMP错误码如下所示。

错误码	标志符	描述
-1	RTMP_ILLEGAL_INPUT	输入不合法，请检查输入参数。
-2	RTMP_MALLOC_FAILED	内存分配失败。
-3	RTMP_CONNECT_FAILED	RTMP建连失败。
-4	RTMP_IS_DISCONNECTED	RTMP连接未建立。
-5	RTMP_UNSUPPORT_FORMAT	不支持的音视频格式。
-6	RTMP_SEND_FAILED	RTMP数据包发送失败。
-7	RTMP_READ_MESSAGE_FAILED	RTMP消息读取失败。
-8	RTMP_READ_TIMESTAMP_ERROR	输入时间戳错误。

2. 处理开始直播推流请求。

i. 当服务端下发推流请求时，回调 `OnLiveStreamListener.onStartPushLiveStreaming(int streamId, int streamType, int preTimeInS)` 方法来通知设备端需要开始采流并推流。

一般需要开启摄像头和录音机进行采流，对摄像头采集的数据调用MediaCodec进行H264编码，对录音机采集的数据进行G711a编码，提前设置对应格式的音视频参数，分别调用发送音视频的接口来持续发送采集到编码后的数据。

```
@Override
    public void onStartPushLiveStreaming(int streamId, int streamType, int preTimeI
nS) {
        this.streamId = streamId;
        try {
            // 构造视频参数
            VideoStreamParams videoStreamParams = new VideoStreamParams();
            // 直播流该参数始终为0
            videoStreamParams.setDurationInS(0);
            videoStreamParams.setVideoFormat(VideoStreamParams.VIDEO_FORMAT_H264);
            // 构造音频参数
            AudioStreamParams audioStreamParams = new AudioStreamParams();
            audioStreamParams.setAudioChannel(AudioStreamParams.AUDIO_CHANNEL_MONO)
;
            audioStreamParams.setAudioFormat(AudioStreamParams.AUDIO_FORMAT_G711A);
            audioStreamParams.setAudioEncoding(AudioStreamParams.AUDIO_ENCODING_16B
IT);
            audioStreamParams.setAudioSampleRate(AudioStreamParams.AUDIO_SAMPLE_RAT
E_8000);
            // 设置推流参数
            IPCDev.getInstance().getIpcStreamManager().setStreamParams(streamId, vi
deoStreamParams, audioStreamParams);
            // TODO 开始采流、编码并发送音视频数据
        } catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

ii. 发送音视频数据接口（IPCStreamManager）。

```
     /**
      * 发送音频帧数据
      *
      * @param streamId         流ID
      * @param directByteBuffer 源数据
      * @param length           数据长度
      * @param timeStampInMs    音频帧时间戳，单位ms
      */
     void sendAudioData(int streamId, ByteBuffer directByteBuffer, int length, long
timeStampInMs) throws NoSuchStreamException
    /**
      * 发送视频帧数据
      *
      * @param streamId         流ID
      * @param directByteBuffer 源数据
      * @param length           数据长度
      * @param isIFrame         是否为I帧
      * @param timeStampInMs    视频帧时间戳，单位ms
      */
    void sendVideoData(int streamId, ByteBuffer directByteBuffer, int length, boole
an isIFrame, long timeStampInMs) throws NoSuchStreamException
     /**
      * 发送音频帧数据
      *
      * @param streamId      流ID
      * @param data          源数据
      * @param offset        偏移量
      * @param length        数据长度
      * @param timeStampInMs 音频帧时间戳，单位ms
      * @deprecated 使用 {@link #sendAudioData(int, ByteBuffer, int, long)}来替换
      */
    @Deprecated
    void sendAudioData(int streamId, byte[] data, int offset, int length, long time
StampInMs) throws NoSuchStreamException
    /**
      * 发送视频帧数据
      *
      * @param streamId      流ID
      * @param data          源数据
      * @param offset        偏移量
      * @param length        数据长度
      * @param isIFrame      是否为I帧
      * @param timeStampInMs 视频帧时间戳，单位ms
      * @deprecated 使用 {@link #sendVideoData(int, ByteBuffer, int, boolean, long)}
来替换
      */
    @Deprecated
    public void sendVideoData(int streamId, byte[] data, int offset, int length, bo
olean isIFrame, long timeStampInMs) throws NoSuchStreamException
```

iii. 打印I帧的前256个字节，并查看结果。

H264、H265的帧结构会有一定的要求，可以打印I帧的前256个字节查看，打印代码如下。

```
for (int i = 0; i < ((buffer_size > 256)?256:buffer_size); i++) {
    printf("%02x ", buffer[i]);
    if ((i + 1) % 30 == 0) {
        printf("\n");
    }
}
printf("\n");
```

H264、H265的帧结构的说明如下。

- H264

  H264要求I帧为：`帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR`，其中，0x000001或者0x00000001是帧分隔符，0x67是SPS的开始，0x68是PPS的开始，0x65是IDR的开始。如下图所示。

```
00 00 01 67 42 c0 1e d9 00 a0 2f f9 3f f2 57 92 58 01 00 00 03 00 01 00 00 03 00 32 0f 16
2e 48 00 00 00 01 68 cb 8c b2 00 00 01 65 88 84 1f c9 81 0b 40 00 24 c7 19 c5 87 40 65 89
c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 ff ff
77 c2 a1 6e 00 16 c4 d5 6f e8 1b 0d 2c af b8 00 d1 72 9b 8d 81 28 7d 2c 8c 18 5d 50 9d 09
8e 79 94 b4 d8 9d e9 7e 0b ad 99 ca 06 a7 ff c7 84 56 15 91 7b 50 01 9f a0 8a f2 75 64 67
ab c2 b2 11 61 6f 23 00 03 4a d5 93 e4 1c 11 8b 2f f8 23 0d 26 cf d4 01 5a 2c c2 67 23 bd
1d 0e ff bf ff f0 42 5e 00 09 9a 88 33 36 eb 90 00 0f e0 33 ff 80 60 b0 42 4e 00 13 59 0a
ef 4a fc 89 bf ff 80 42 bc 30 00 4c e9 0c 89 ba e4 00 1f 40 8c 1a ff f0 f5 85 78 00 73 93
75 f4 bc fc 00 0f 17 21 9c e2 40 7f cc 3c c0
```

- H265

  H265要求I帧为：`帧分隔符+VPS+帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR`，其中，0x000001或者0x00000001是帧分隔符，0x40是VPS的开始，0x42是SPS的开始，0x44是PPS的开始，0x26是IDR的开始。如下图所示。

```
00 00 00 01 40 01 0c 01 ff ff 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b ac 09 00 00 00 00 01
42 01 01 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b a0 03 c0 80 10 e5 96 b9 2c ad 9a e5 51 36
00 80 00 00 00 00 00 01 44 01 c0 e3 0f 03 32 40 00 00 00 00 01 26 01 af 13 80 5b ff ff ff a9 59
16 ca fe 88 26 26 11 cc 9a 40 61 2f 2d 47 37 39 09 25 e4 6d a2 96 33 cb 99 ff c6 74 73 cd 15 0b
98 45 db fe 98 68 e2 80 bd f6 99 b9 37 c0 a3 5b 9f 38 70 82 8a 20 b8 44 a6 d7 35 35 5c 6e ed ce
25 28 10 00 88 21 0c e0 4b fd aa 47 9b a4 e7 ad 27 f0 f5 83 e3 a6 6e 89 2b 2a 7b 18 65 1d 4a 05
ff a1 fd d3 44 fe c3 34 ad ae a5 1a b5 a2 61 83 c3 ca 6a 1a c0 7b 08 44 42 f9 bf b7 ec 88 11 01
39 33 1e 96 b9 2d 91 72 63 e0 d5 e2 af a0 82 c0 46 b7 bd 43 cf f5 36 b5 f1 09 4f b6 2e e9 e3 00
```

3. 处理结束推流请求。

   当服务端下发停止推流请求时，回调`OnLiveStreamListener.onStopPushLiveStreaming()`方法来通知设备端停止推流。

   一般需要停止摄像头数据和录音机数据的采集，并调用`IPCStreamManager`的`stopStreaming(int streamId)`方法。

```
/**
     * 收到停止推流请求
     *
     * @param streamId 流ID
     */
    @Override
    public void onStopPushStreaming(int streamId) {
        // TODO 停止音视频数据的发送
        try {
            // 调用停止推流接口
            IPCDev.getInstance().getIpcStreamManager().stopStreaming(streamId);
        } catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

4. 处理流错误。

   推流过程中需要处理流错误，通过 `OnStreamErrorListener.onError(int streamId, StreamError error)` 来接收和处理。

开发点播功能

点播通过RTMP推流，视频支持H264/H265，音频支持G711a以及AAC_LC格式。请根据以下步骤来开发点播功能的推流。

1. 注册点播事件监听器和流错误监听器。

   SDK收到服务端下发的开始推流指令后，会通过事先注册的点播流事件监听器（OnVodStreamListener）来通知何时开始或结束推流/暂停/恢复/seek等。

   推流中发生的错误也将通过流错误监听器来通知。请不要在回调接口中执行阻塞任务。

2. 处理查询设备端录像列表请求。

   App端发起查询设备端录像列表的请求，设备端会收到同步服务调用（rrpc/request），收到查询设备录像列表请求，响应该请求将当查询范围内的文件列表返回给App端。

```
@Override
        public void onNotify(String connectId, String topic, AMessage aMessage) {
            Log.d(TAG, "onNotify() called with: connectId = [" + connectId + "], topic
= [" + topic + "], aMessage = ["
                    + new String((byte[]) aMessage.data) + "]");
            /**
             * 添加SDK的监听。
             */
            IPCDev.getInstance().notifySyncTopicReceived(connectId, topic, aMessage);
            // 处理同步服务调用
            if (CONNECT_ID.equals(connectId) && !TextUtils.isEmpty(topic) &&
                    topic.contains("rrpc")) {
                Log.d(TAG, "IConnectNotifyListener   onNotify() called with: connectId
= [" + connectId + "], topic = ["
                        + topic + "], aMessage = ["
                        + new String((byte[]) aMessage.data) + "]");
                int code = 200;
                String data = "{}";
                JSONObject json = JSON.parseObject(new String((byte[]) aMessage.data));
                if (json != null) {
```

```
                if (json != null) {
                    String method = json.getString("method");
                    JSONObject params = json.getJSONObject("params");
                    switch (method) {
                        // 查询设备录像列表请求
                        case "thing.service.QueryRecordList":
                            int beginTime = params.getIntValue("BeginTime");
                            int endTime = params.getIntValue("EndTime");
                            int querySize = params.getIntValue("QuerySize");
                            int type = params.getIntValue("Type");
                            appendLog("收到查询设备录像列表的请求：beginTime=" + beginTime
+
                                    "\tendTime=" + endTime + "\tquerySize=" + querySize
+ "\ttype=" + type);
                            JSONArray resultArray = new JSONArray();
                            JSONObject item1 = new JSONObject();
                            item1.put("FileName", Base64.encode("file1".getBytes(), Bas
e64.DEFAULT));
                            item1.put("BeginTime", System.currentTimeMillis() / 1000 -
200);
                            item1.put("EndTime", System.currentTimeMillis() / 1000 - 10
0);
                            item1.put("Size", 1024000);
                            item1.put("Type", 0);
                            resultArray.add(item1);
                            JSONObject item2 = new JSONObject();
                            item2.put("FileName", Base64.encode("file2".getBytes(), Bas
e64.DEFAULT));
                            item2.put("BeginTime", System.currentTimeMillis() / 1000 -
100);
                            item2.put("EndTime", System.currentTimeMillis() / 1000);
                            item2.put("Size", 1024000);
                            item2.put("Type", 0);
                            resultArray.add(item2);
                            JSONObject result = new JSONObject();
                            result.put("RecordList", resultArray);
                            code = 200;
                            data = result.toJSONString();
                            break;
                        default:
                            break;
                    }
                }
                MqttPublishRequest request = new MqttPublishRequest();
                request.isRPC = false;
                request.topic = topic.replace("request", "response");
                String resId = topic.substring(topic.indexOf("rrpc/request/") + 13);
                request.msgId = resId;
                request.payloadObj = "{\"id\":\"" + resId + "\", \"code\":" + code + ",
\"data\":" + data + "}";
                LinkKit.getInstance().publish(request, new IConnectSendListener() {
                    @Override
                    public void onResponse(ARequest aRequest, AResponse aResponse) {
                        appendLog("上报成功");
                    }
```

```
                    }
                    @Override
                    public void onFailure(ARequest aRequest, AError aError) {
                        appendLog("上报失败:" + aError.toString());
                    }
                });
            }
        }
```

> 注意　文件名需使用Base64进行编码。

3. 处理开始推流指令。

   App端请求上一步返回的列表中某个设备录像文件后，服务端会下发推流指令，回调 `OnVodStreamListener.onStartPushVodStreaming(int streamId, String fileName)` 或 `OnVodStreamListener.onStartPushVodStreaming(int streamId, int beginTimeUtc, int endTimeUtc)` 方法来通知设备端需要将音视频数据推流。

   - 按文件方式播放设备端录像的响应

```
    @Override
    public void onStartPushVodStreaming(int streamId, String fileName) {
        appendLog("开始推点播流 " + streamId + " 文件名:" + new String(Base64.decode(fileName, Base64.NO_WRAP)));
        try {
            // 构造视频参数
            VideoStreamParams videoStreamParams = new VideoStreamParams();
            // 该视频文件的时长，单位S
            videoStreamParams.setDurationInS(H264_DURATION_IN_S);
            videoStreamParams.setVideoFormat(VideoStreamParams.VIDEO_FORMAT_H264);
            // 构造音频参数
            AudioStreamParams audioStreamParams = new AudioStreamParams();
            audioStreamParams.setAudioChannel(AudioStreamParams.AUDIO_CHANNEL_MONO);
            audioStreamParams.setAudioFormat(AudioStreamParams.AUDIO_FORMAT_G711A);
            audioStreamParams.setAudioEncoding(AudioStreamParams.AUDIO_ENCODING_16BIT);
            audioStreamParams.setAudioSampleRate(AudioStreamParams.AUDIO_SAMPLE_RATE_8000);
            // 设置推流参数
            IPCDev.getInstance().getIpcStreamManager().setStreamParams(streamId, videoStreamParams, audioStreamParams);
            // TODO 读取fileName文件，调用发送音视频数据接口进行推流
            // 文件推流完毕后应调用 IPCDev.getInstance().getIpcStreamManager().notifyVodComplete(streamId) 通知推流完成
        } catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

   - 按时间方式播放设备端录像的响应

```
    @Override
    public void onStartPushVodStreaming(int streamId, int beginTimeUtc, int endTimeUtc) {
        appendLog("开始推点播流 " + streamId + " beginTimeUtc: "+beginTimeUtc + " endTimeUtc:"+endTimeUtc);
        //TODO 推流逻辑需要添加：
        // 1. beginTimeUtc和endTimeUtc一般是一天的开始和结束时间
        // 2. 当收到onStartPushVodStreaming回调后，应从beginTimeUtc开始向后最近的I帧开始推流，时间戳应使用对应帧的UTC时间
        // 3. 若beginTimeUtc到endTimeUtc范围内没有录像或范围内推流已经完成了，则应调用 IPCDev.getInstance().getIpcStreamManager().notifyVodComplete(streamId) 通知推流完成
        // 4. 只要是beginTimeUtc到endTimeUtc范围内有数据，即使跨文件，推流应该持续不断
    }
```

4. 处理暂停或恢复指令。

   需响应暂停或恢复推流指令（OnVodStreamListener），相应的暂停或恢复发送音视频数据。

```
/**
     * 收到暂停推流请求
     *
     * @param streamId 流ID
     */
    void onPausePushVodStreaming(int streamId);
    /**
     * 收到恢复推流的请求
     *
     * @param streamId 流ID
     */
    void onResumePushVodStreaming(int streamId);
```

5. 处理Seek指令。

   当您需响应Seek指令（OnVodStreamListener）时，例如App端播放器进度条Seek到80秒时，对应会回调onSeekTo方法，请从该timeStampInS时间点最近的I帧开始继续推流。

```
/**
     * 收到重新定位请求
     *
     * @param streamId     流ID
     * @param timeStampInS 时间偏移量，相对于视频开始时间，单位为S
     */
    void onSeekTo(int streamId, long timeStampInS);
```

6. 处理停止推流指令。

   当服务端下发停止推流请求时，通过回调 `OnVodStreamListener.onStopPushLiveStreaming()` 方法来通知设备端停止推流。

   一般要停止调用发送音视频接口、关闭视频文件，并调用 `IPCStreamManager的stopStreaming(int streamId)` 方法。

```
/**
     * 收到停止推流请求
     *
     * @param streamId 流ID
     */
    @Override
    public void onStopPushStreaming(int streamId) {
        // TODO 停止音视频数据的发送
        try {
            // 调用停止推流接口
            IPCDev.getInstance().getIpcStreamManager().stopStreaming(streamId);
         catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

7. 处理流错误。

   推流过程中需要处理流错误，通过 `OnStreamErrorListener.onError(int streamId, StreamError error)` 来接收和处理。

开发语音对讲功能

语音对讲支持单讲、双讲模式。

- 单讲：App端采集并发送音频数据到设备端进行播放。
- 双讲：App端和设备端都需要同时做采音和放音，设备端必须支持AEC，否则不建议使用该方案。

语音对讲支持的音频格式如下。

格式	采样率	编码	解码
G711A	8Khz/16Khz	✓	✓
G711U	8Khz/16Khz	✓	✓

> **说明** 对讲提供V1、V2两套接口，除特殊说明外，请使用V2版本。
> - V1版本：需要自行实现录音采集、播放以及回声消除。
> - V2版本：支持单/双讲模式，内部已经实现录音采集、播放以及回声消除，支持设置采集音频增益大小。

1. 注册语音对讲事件监听器和错误监听器。

   SDK收到服务端下发的开始推流指令后，会通过事先注册的语音对讲事件监听器（OnLiveIntercomListener）来通知开始或结束语音对讲、接收对端的音频参数和语音数据。

   推流中发生的错误也将通过流错误监听器来通知App。请不要在回调接口中执行阻塞任务。

   i. 选择使用的语音对讲版本。

   ```
   // 设置使用对讲第二版本，默认为此版本
   IPCDev.getInstance().setLiveIntercomVersionBeforeInit(IPCDev.LiveIntercomVersion.VERSION_2);
   ```

ii. 选择对讲模式。

```
// 设置双向对讲模式，只对V2版本接口有效
IPCDev.getInstance().setLiveIntercomModeBeforeInit(IPCLiveIntercomV2.LiveIntercomMode.DoubleTalk);
```

iii. 设置监听。

```
// 设置语音对讲事件监听
IPCDev.getInstance().getIpcLiveIntercom().setOnLiveIntercomListener(MainActivity.this);
// 设置语音对讲错误回调
IPCDev.getInstance().getIpcLiveIntercom().setOnLiveIntercomErrorListener(MainActivity.this);
```

注册语音对讲事件监听器和错误监听器的完整代码如下。

```
public interface OnLiveIntercomListener {
    /**
     * 收到App端发起的开始语音对讲请求
     *
     * @return 返回当前设备端上行音频参数格式，如采样率、通道数、采样位宽、音频格式，请确保对App端能支持该音频参数配置
     */
    AudioParams onStartVoiceIntercom();
    /**
     * 收到结束语音对讲请求
     */
    void onStopVoiceIntercom();
    /**
     * 收到App端音频参数，表示与App端的通道已建立，可以开始对讲
     * @param audioParams  App端的音频参数
     */
    void onAudioParamsChange(AudioParams audioParams);
    /**
     * 收到App端发送的PCM数据，一般用来做UI展示，比如绘制音量大小
     * @param buffer
     * @param size
     */
    void onAudioBufferReceive(byte[] buffer, int size);
}
```

```
public interface OnLiveIntercomErrorListener {
    /**
     * 语音对讲发生错误
     * @param error 见{@link LiveIntercomError}
     */
    void onError(LiveIntercomError error);
}
```

2. 响应开始语音对讲请求（仅V1版本接口需要处理）。

i. 启动录音机开始音频采集并将音频数据发送给对端 。

```
@Override
    public AudioParams onStartVoiceIntercom() {
        appendLog("收到开始语音对讲指令");
        // 收到开始语音对讲请求，启动录音机
        simpleAudioRecord.setAudioRecordListener(new AudioRecordListener() {
            @Override
            public void onRecordStart() {
                // 录音开始
                appendLog("录音开始");
            }
            @Override
            public void onRecordEnd() {
                // 录音结束
                appendLog("录音结束");
            }
            @Override
            public void onBufferReceived(byte[] buffer, int offset, int size) {
                // 收到录音机抛出的PCM数据，调用发送接口发送给对端
                IPCDev.getInstance().getIpcLiveIntercom().sendAudioBuffer(buffer, o
ffset, size);
            }
            @Override
            public void onError(int error, String message) {
                appendLog("录音机错误:" + error + " " + message);
            }
        });
        simpleAudioRecord.start();
        // 通知使用G711A作为音频发送格式，内部会对PCM数据重新做编码
        return AudioParams.AUDIOPARAM_MONO_8K_G711A;
    }
```

ii. 发送音频数据接口（IPCLiveIntercom，仅V1版本接口需要处理）。

```
/**
     * 发送音频数据<br>
     * 等价{@link #sendAudioBuffer(byte[] data, int offset, int length, boolean enab
leEncode)} enableEncode=true
     */
    void sendAudioBuffer(byte[] data, int offset, int length);
    /**
     * 发送音频数据<br>
     *
     * @param data         数据buffer
     * @param offset       偏移量
     * @param length       长度
     * @param enableEncode true: 内部会依据{@link OnLiveIntercomListener#onStartVoic
eIntercom()}返回的音频格式对送入的数据（必须为PCM）重新编码并发送
     *                     false: 内部会直接发送音频数据
     */
    void sendAudioBuffer(byte[] data, int offset, int length, boolean enableEncode)
;
```

3. 处理声音播放（仅V1版本接口需要处理）。

当App端和设备端语音对讲通道建立后，设备端通过 `onAudioParamsChange(AudioParams audioParams)` 收到App端发过来的音频参数。依据该音频参数新建音频播放器，供后续接受的音频数据播放。

```
@Override
    public void onAudioParamsChange(AudioParams audioParams) {
        // 收到对端发送的音频参数
        appendLog("收到客户端的音频参数：" + audioParams.toString());
        // 初始化播放器
        if (simpleStreamAudioTrack != null) {
            simpleStreamAudioTrack.release();
            audioTrackQueue.clear();
        }
        if (acousticEchoCanceler != null) {
            acousticEchoCanceler.release();
        }
        if (noiseSuppressor != null) {
            noiseSuppressor.release();
        }
        noiseSuppressor = NoiseSuppressor.create(simpleAudioRecord.getAudioSessionId())
;
        if (noiseSuppressor != null) {
            noiseSuppressor.setEnabled(true);
        }
        simpleStreamAudioTrack = new SimpleStreamAudioTrack(audioParams, AudioManager.S
TREAM_MUSIC, audioTrackQueue,
                simpleAudioRecord.getAudioSessionId());
        if (AcousticEchoCanceler.isAvailable()) {
            acousticEchoCanceler = AcousticEchoCanceler.create(simpleAudioRecord.getAud
ioSessionId());
            if (acousticEchoCanceler != null) {
                appendLog("已开启回声消除");
                acousticEchoCanceler.setEnabled(true);
            }
        }
        simpleStreamAudioTrack.start();
    }
    @Override
    public void onAudioBufferReceive(byte[] buffer, int size) {
        // 收到对端发送的PCM数据
        audioTrackQueue.add(buffer);
    }
```

4. 处理结束语音对讲指令（仅V1版本接口需要处理）。

```
@Override
public void onStopVoiceIntercom() {
 appendLog("收到停止语音对讲指令");
 // 收到停止语音对讲请求，停止录音
 simpleAudioRecord.stop();
}
```

5. 语音对讲错误处理。

```
@Override
public void onError(LiveIntercomError error) {
 // 语音对讲发生错误
 appendLog("语音对讲错误:" + error.getCode() + " msg:" + error.getMessage());
 // 停止录音（仅V1版本接口需要处理）
 simpleAudioRecord.stop();
 // 停止播放（仅V1版本接口需要处理）
 simpleStreamAudioTrack.stop();
}
```

语音对讲错误列表如下。

错误码	错误描述
LiveIntercomError.INVALID_AUDIO_PARAMS	无效的设备端音频参数
LiveIntercomError.START_LIVE_INTERCOM_REQUEST_FAILED	无效的语音对讲请求
LiveIntercomError.CONNECTION_STREAM_FAILED	建立语音对讲流通道失败
LiveIntercomError.SEND_STREAM_DATA_FAILED	发送音频数据失败
LiveIntercomError.RECEIVE_STREAM_DATA_FAILED	接收音频数据失败
LiveIntercomError.INIT_RECORD_FAILED	录音机初始化错误
LiveIntercomError.START_RECORD_FAILED	录音机启动错误
LiveIntercomError.READ_RECORD_BUFFER_FAILED	录音机数据读取错误
LiveIntercomError.INIT_AUDIO_PLAYER_FAILED	音频播放器创建失败

开发抓图功能

抓图功能通过回调 `IUploadPicListener` 来上传图片。回调信息包括上传地址、此次上传的图片ID、上传类型等。图片上传完成后，设备端通过 `IUploadPicProcessCallback` 回调通知SDK上传任务完成。抓图功能主要分以下两种场景。

- App端发起抓图请求，设备抓图后上传至云端。
- 侦测事件触发，设备端自动抓图并上传图片至云端。

> **说明** 您还需要确认拍照的物模型TriggerPicCapture里的 `UploadUrl` 字符串长度，如果小于512字符，则需要修改到大于512字符，否则图片上传会受到影响。

1. 抓图上传。

   拍照上传任务的流程通常为：SDK通知设备拍照、设备拍照、设备上传图片、通知SDK上传结果。 详细流程如下。

i. 设备端注册上传图片监听。

```
/**
 * 上传图片监听
 */
public interface IUploadPicListener {
    /**
     * 触发上传
     * @param uploadInfo  上传信息
     * @param callback    上传结果回调
     */
    void onUpload(UploadInfo uploadInfo, IUploadPicProcessCallback callback);
}
```

ii. 当App触发拍照，设备端通过 `IUploadPicListener` 的 `onUpload` 获取到当前需要抓图并上传通知。

其中图片上传的URL从onUpload回调的uploadInfo参数中获取。

```
/**
 * 上传图片
 */
public class UploadInfo {
    /**
     * 上传url
     */
    private String uploadUrl;
    /**
     * 上传图片ID（用户生成的，只有事件上报时才有）
     */
    private String picId;
    /**
     * 上传类型，0为拍照，1为事件图片上传。
     */
    private int type;
}
```

iii. 设备端通过 `IUploadPicProcessCallback` 回调通知SDK上传结果。

```
/**
 * 上传图片结果回调
 */
public interface IUploadPicProcessCallback {
    /**
     * 上传任务成功
     */
    void onSucceed();
    /**
     * 上传任务失败
     *
     * @param errorMsg 错误信息
     */
    void onFailed(String errorMsg);
}
```

iv. 监听注册结果。

LinkKit SDK初始化成功后，设备端通过如下代码监听注册。

```
//注册上传图片监听，当App端触发拍照或者设备上报了事件以后，会在注册的回调中收到上传的回调。
IPCDev.getInstance().registerUploadPicListener(uploadPicListener);
/**
 * 上传图片回调，回调中只有是事件照片上传的时候picId不为空且为上报事件时用户上传的ID。
 * type为上传类型，0为App触发设备拍照，1为设备上传事件照片。
 */
private IUploadPicListener uploadPicListener = new IUploadPicListener() {
        @Override
        public void onUpload(final UploadInfo uploadInfo, final IUploadPicProcessCa
llback callback) {
            final String url = uploadInfo.getUploadUrl();
            final String picId = uploadInfo.getPicId();
            final int type = uploadInfo.getType();
            switch (type) {
                //拍照上报
                case 0:
                    //拍照并且上传
                    upLoadFile(url, pic, new CallBack(){
                             @Override
                             public void onSucceed() {
                                    Log.d(TAG, "uploadImage onSucceed");
                                    if (callback != null) {
                                            callback.onSucceed();
                                    }
                             }
                              @Override
                              public void onFailed(String error) {
                                     Log.e(TAG, "uploadImage onFailed:" + error);
                                     if (callback != null) {
                                            callback.onFailed(error);
                                    }
                             }
                    });
                    break;
                //事件图片上报
                case 1:
                    //上传图片
                    upLoadFile(url, pic, new CallBack(){
                             @Override
                             public void onSucceed() {
                                    Log.d(TAG, "uploadImage onSucceed");
                                    if (callback != null) {
                                            callback.onSucceed();
                                    }
                             }
                              @Override
                              public void onFailed(String error) {
                                     Log.e(TAG, "uploadImage onFailed:" + error);
                                     if (callback != null) {
                                            callback.onFailed(error);
                                    }
```

```
                                }
                            });
                            break;
                        default:
                            break;
                    }
                }});
```

2. 事件上报触发图片上传。

   事件上报流程依次为：设备触发事件、事件上报、SDK通知设备上传图片、设备上传图片、通知SDK上传结果。 具体的流程如下。

   i. 设备端注册上传图片的监听。
   ii. 当设备端触发事件，设备通过SDK提供的接口上报事件（需要上传图片ID）。
   iii. SDK会通过 `IUploadPicListener` 的 `onUpload` 通知设备上传图片。
   iv. 设备端通过 `onUpload` 中的UploadInfo获取此次上传的URL、图片ID。
   v. 设备端自行通过URL上传图片。
   vi. 设备端通过 `IUploadPicProcessCallback` 回调通知SDK上传结果。

   上报事件接口实例如下。

```
//上报报警事件分成两步：1.上传报警，2.上传图片
//上报报警事件需要设备生成一个报警图片ID，这个报警图片ID要求是设备唯一的图片ID，建议用Unix时间戳。
// 不过这边需要确认一下拍照的物模型"TriggerPicCapture"里的"UploadUrl"的字符串长度，如果小于512请增加长度到512以上。
// 上报事件以后会收到IUploadPicListener（需要注册监听）内的onUpload回调，然后请将对应的图片上传到回调中url指定地址。
final String alarmPicId = String.valueOf(System.currentTimeMillis() / 1000);
IPCDev.getInstance().reportAlarmEvent(alarmPicId, 1, new ReportAlarmEventListener() {
    @Override
    public void onSucceed() {
        Log.d(TAG, " report onSucceed  alarmPicId:" + alarmPicId);
        appendLog("上报事件成功");
    }
    @Override
    public void onFailed(String msg) {
        Log.e(TAG, " report onFailed  e:" + msg);
        appendLog("上报事件失败  e:" + msg);
    }
});
```

8.配网功能开发

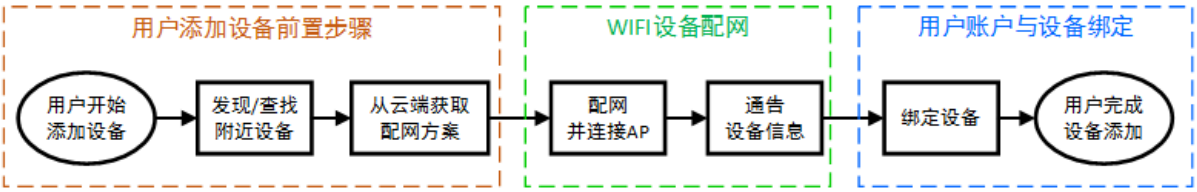
8.1. Wi-Fi设备配网方案介绍

对于无屏的设备（如插座、灯泡等），用户无法通过输入Wi-Fi热点信息让设备接入网络，此时需要对这些设备进行配网操作。生活物联网平台为Wi-Fi设备提供了多种配网技术方案，使不具备人机交互能力的设备可以借助于一些特殊方式连上网络。

配网概述

配网是将路由器的Wi-Fi SSID和PASSWORD通过某种方式传递到终端设备，让终端设备可以接入Wi-Fi网络的过程。配网示意图如下所示。

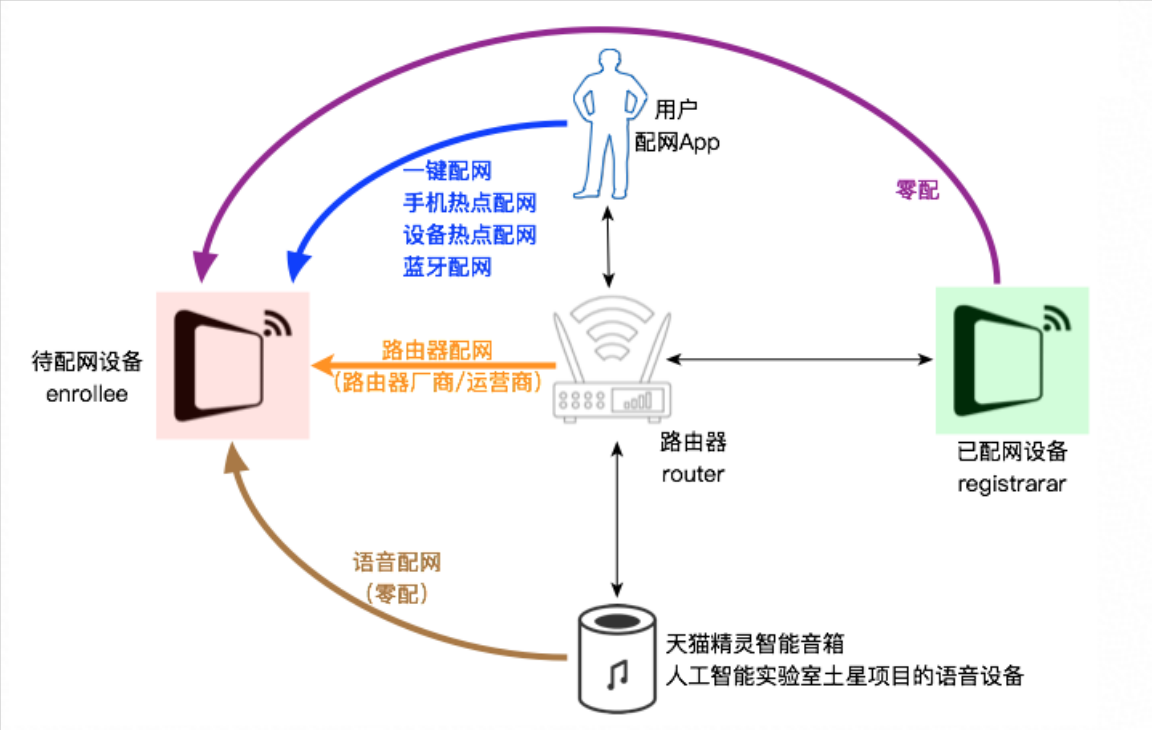
通常情况下，用户使用App添加一个Wi-Fi设备的流程如下。



1. 用户添加设备前的准备操作。
2. Wi-Fi设备配网（通过某种方式将目标AP的SSID和Password给到设备，使设备可连接上目标AP）。
3. 用户账户与设备之间绑定。

Wi-Fi配网方式对比

生活物联网平台提供了多种Wi-Fi设备的配网方式（各种方式的对比如下表所示），Wi-Fi设备配网方式的总体场景示意图如下。



配网方式	配网技术	简要说明	补充说明
设备热点配网（dev-ap-config）	手机连设备热点传数据	App连接设备起的热点传输配网信息给设备	推荐使用该配网方式
蓝牙辅助配网（ble-config）	BLE（Bluetooth）传数据	App通过蓝牙通道传输配网信息给设备	设备必须支持Wi-Fi与BLE双模通信（combo chip），推荐您使用该配网方式

设备热点配网

设备热点配网（dev-ap-config）的配网流程如下。

1. 设备开启自带的Wi-Fi热点。手机搜索并发现热点后，连接到该设备的热点。
2. 建立连接通道后，手机将Wi-Fi热点（路由器）的SSID/密码发送给设备。
3. Wi-Fi设备使用该SSID/密码连接Wi-Fi热点（路由器）。

蓝牙辅助配网

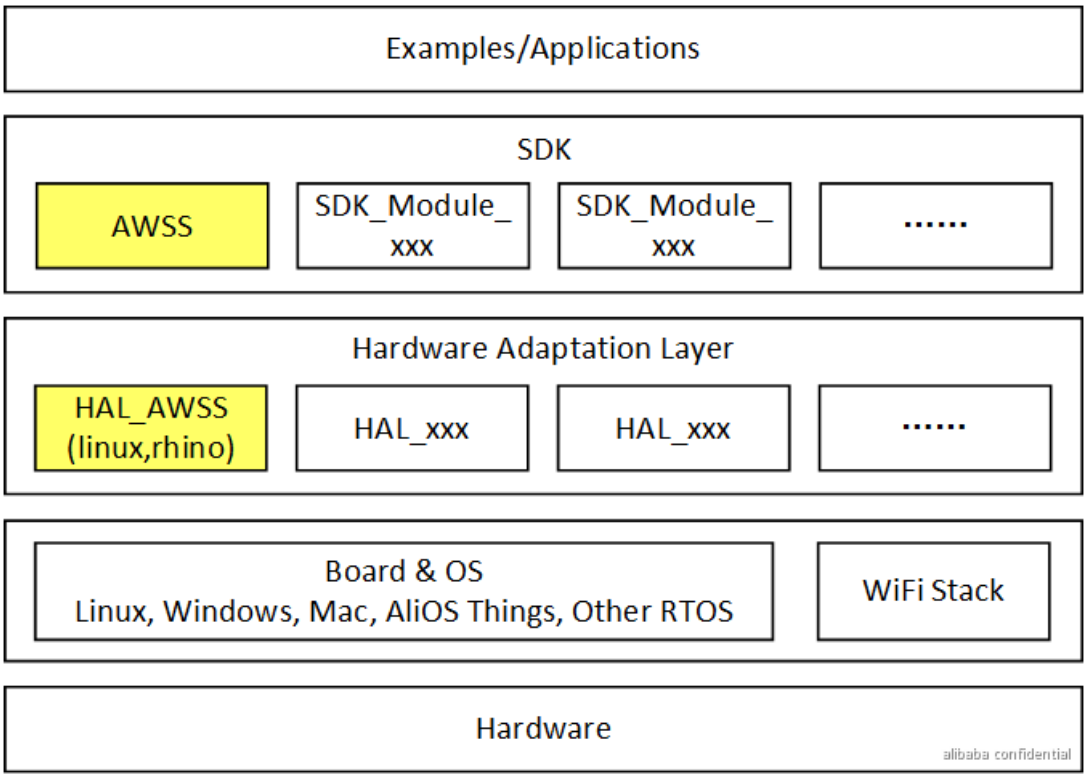
蓝牙辅助配网（ble-config）方案无需区分手机的操作系统（iOS或Android的手机体验一样），且该方案的配网成功率和可靠性较高。使用该方案，设备需支持Wi-Fi与BLE双模通信能力。配网流程如下。

1. 手机通过蓝牙连接待配网的双模设备。
2. 手机通过蓝牙将Wi-Fi热点（路由器）的SSID/密码信息传送给设备。
3. 设备连接Wi-Fi热点。

8.2. Wi-Fi设备配网适配开发

生活物联网平台提供多种Wi-Fi设备的配网技术方案，对于不具备人机交互能力的设备，可以通过借助其他设备来协助配网。为此，生活物联网平台提供了开放的配网能力，便于您在不同的设备硬件平台上快速适配和实现Wi-Fi配网功能。

下图为生活物联网平台SDK中Wi-Fi设备配网模块的分层框架。



- HAL_AWSS实现接口和OS、硬件的适配对接，为上层SDK的配网服务提供基础的Wi-Fi通信能力。

  在生活物联网平台SDK中，HAL_AWSS的适配实现，主要提供了Wi-Fi配网需要适配的接口。

- AWSS模块是SDK中提供的配网服务，可以提供多种配网方式，依赖于HAL层的HAL_AWSS部分，以及HAL层其他相关接口的正确适配。

  AWSS模块为上层的应用开发提供了配网相关的用户编程接口，Examples为调用这些接口的完整示例的实现。

  在生活物联网平台SDK中，AWSS用户编程接口，主要用来调用实现设备的Wi-Fi配网功能。

通用HAL说明

Wi-Fi配网模块中依赖生活物联网平台SDK的公共HAL接口（如下表所示）。请您参照生活物联网平台SDK中对应模块的说明完成对接。

序号	适配接口名	说明
1	HAL_MutexCreate	创建一个互斥量对象，返回指向所创建互斥量的指针，用于同步访问，对于仅支持单线程应用，可实现为空函数
2	HAL_MutexDestroy	销毁一个互斥量对象，释放资源
3	HAL_MutexLock	锁住一个互斥量
4	HAL_MutexUnlock	解锁一个互斥量
5	HAL_UptimeMs	获取设备从上电到当前时刻所经过的毫秒数
6	HAL_Malloc	申请一块堆内存
7	HAL_Free	释放参数ptr指向的一块堆内存，当传入的参数为NULL时不执行任何操作
8	HAL_SleepMs	睡眠函数，使当前执行线程睡眠指定的毫秒数
9	HAL_GetProductKey	获取设备的ProductKey，用于标识设备的品类，设备证书之一
10	HAL_GetProductSecret	获取设备的ProductSecret，用于标识设备的品类，设备证书之一
11	HAL_GetDeviceName	获取设备的DeviceName，用于标识设备单品的名字，设备证书之一
12	HAL_GetDeviceSecret	获取设备的DeviceSecret， 用于标识设备单品的密钥， 设备证书之一
13	HAL_Kv_Set	Flash中写入键值对（Key-Value）
14	HAL_Kv_Get	Flash中读取键值对的Value
15	HAL_Aes128_Init	初始化AES加密的结构体
16	HAL_Aes128_Destroy	销毁AES加密的结构体
17	HAL_Aes128_Cbc_Encrypt	以AES-CBC-128方式，根据 `HAL_Aes128_Init()` 时传入的密钥，加密指定的明文
18	HAL_Aes128_Cbc_Decrypt	以AES-CBC-128方式，根据 `HAL_Aes128_Init()` 时传入的密钥，解密指定的密文
19	HAL_Aes128_Cfb_Encrypt	以AES-CFB-128方式，根据 `HAL_Aes128_Init()` 时传入的密钥，加密指定的明文
20	HAL_Aes128_Cfb_Decrypt	以AES-CFB-128方式，根据 `HAL_Aes128_Init()` 时传入的密钥，解密指定的密文
21	HAL_Timer_Create	根据Name、TimerFunc和用户上下文创建Timer

序号	适配接口名	说明
22	HAL_Timer_Start	Timer开始计时，Timer超时时调用回调函数TimerFunc
23	HAL_Timer_Stop	停止Timer计时
24	HAL_Timer_Delete	删除Timer，释放资源

说明 带AliOS Things的生活物联网平台SDK的HAL接口文件说明，请参见以下目录中的各个文件内容。

- /Living_SDK/framework/protocol/linkkit/sdk/iotx-sdk-c_clone/include/imports/
- /Living_SDK/framework/protocol/linkkit/sdk/iotx-sdk-c_clone/include/iot_import.h

HAL适配概述

Wi-Fi配网模块需要适配的HAL接口的详细说明如下表所示。

说明 表中分类为配网通用的接口在适配时必须要实现，否则会影响Wi-Fi设备生活物联网平台SDK的正常运行。其他配网特定的分类，您可以根据自己硬件平台的情况以及产品的使用场景和需要进行选择支持，但是同一配网方式类别的所有接口需要适配完整，否则会影响此种配网方式的正常运行。

序号	适配接口名	说明	分类
1	HAL_Wifi_Get_Mac	获取设备的MAC地址， 格式应当是”XX:XX:XX:XX:XX:XX”	配网通用
2	HAL_Wifi_Get_IP	获取设备的IP地址，点分十进制格式保存在字符串数组出参，二进制格式则作为返回值，并以网络字节序（大端）表达，Station模式为连接AP时的IP地址，SoftAP模式为设备作为AP的Gateway IP地址	配网通用
3	HAL_Awss_Get_Channelscan_Interval_Ms	获取在每个信道（channel）上扫描的时间长度，单位是毫秒	配网通用
4	HAL_Awss_Get_Timeout_Interval_Ms	获取配网服务（AWSS）的超时时间长度，单位是毫秒	配网通用
5	HAL_Awss_Get_Encrypt_Type	获取一键配网服务的安全等级	一键配网
6	HAL_Awss_Get_Conn_Encrypt_Type	获取基于连接的配网服务（热点配网/零配）的安全等级	• 零配配网 • 设备热点配网 • 蓝牙辅助配网 • 手机热点配网
7	HAL_Awss_Open_Monitor	设备工作在监听（Monitor）模式，并在收到802.11帧的时候调用被传入的回调函数（包括管理帧和数据帧）	配网通用

序号	适配接口名	说明	分类
8	HAL_Awss_Close_Monitor	关闭监听（Monitor）模式	配网通用
9	HAL_Wifi_Scan	启动一次Wi-Fi的空中扫描	配网通用
10	HAL_Awss_Switch_Channel	设置Wi-Fi设备切换到指定的信道（channel）上	配网通用
11	HAL_Awss_Connect_Ap	要求Wi-Fi网卡连接指定热点（Access Point）的函数，bssid指定特定AP，另外bssid也可能为空或无效值（全0或全0xff）	配网通用
12	HAL_Wifi_Send_80211_Raw_Frame	在当前信道（channel）上以基本数据速率（1Mbps）发送裸的802.11帧（raw 802.11 frame）	配网通用
13	HAL_Wifi_Enable_Mgmt_Frame_Filter	使能或禁用对管理帧的过滤	配网通用
14	HAL_Sys_Net_Is_Ready	检查Wi-Fi网卡、芯片或模组当前的IP地址是否有效	配网通用
15	HAL_Wifi_Get_Ap_Info	获取设备所连接的热点（Access Point）的信息（SSID、Password、BSSID等）	配网通用
16	HAL_Wifi_Get_Link_Stat	获取当前与AP连接链路状态的信息（channel、RSSI等）	配网通用
17	HAL_Awss_Open_Ap	开启设备热点（SoftAP模式）	设备热点配网
18	HAL_Awss_Close_Ap	关闭当前设备热点，并把设备由SoftAP模式切换到Station模式	设备热点配网

HAL_Wifi_Get_Mac

原型

```
char *HAL_Wifi_Get_Mac(_OU_ char mac_str[HAL_MAC_LEN]);
```

接口说明

获取Wi-Fi网口的MAC地址，格式为 `XX:XX:XX:XX:XX:XX` 。

参数说明

参数	数据类型	方向	说明
mac_str	char	输出	指向缓冲区数组起始位置的字符指针

返回值说明

指向缓冲区数组起始位置的字符指针。

HAL_Wifi_Get_IP

原型

```
uint32_t HAL_Wifi_Get_IP(_OU_ char ip_str[NETWORK_ADDR_LEN], _IN_ const char *ifname);
```

接口说明

获取Wi-Fi网口的IP地址，点分十进制格式保存在字符串数组出参，二进制格式则作为返回值，并以网络字节序（大端）表达。

参数说明

参数	数据类型	方向	说明
ip_str	char[]	输出	存放点分十进制格式的IP地址字符串的数组
ifname	const char*	输入	指定Wi-Fi网络接口的名字（如果只有一个网口，可以忽略此参数，该参数可能为NULL）

返回值说明

二进制形式的IP地址，以网络字节序（大端）组织。

HAL_Awss_Get_Channelscan_Interval_Ms

原型

```
int HAL_Awss_Get_Channelscan_Interval_Ms(void);
```

接口说明

获取在每个信道（channel）上扫描的时间长度，单位是毫秒，建议设置为200ms~400ms（默认250ms）。

参数说明

void

返回值说明

时间长度，单位为毫秒。

HAL_Awss_Get_Timeout_Interval_Ms

原型

```
int HAL_Awss_Get_Timeout_Interval_Ms(void);
```

接口说明

获取配网服务（AWSS）的超时时间长度，单位是毫秒，建议配置为60s或60000ms。

参数说明

void

返回值说明

超时时长，单位是毫秒。

HAL_Awss_Get_Encrypt_Type

原型

```
int HAL_Awss_Get_Encrypt_Type(void);
```

接口说明

获取一键配网服务的安全等级。

参数说明

void

返回值说明

值	说明
0	open (no encrypt)
1	aes256cfb with default aes-key and aes-iv
2	aes128cfb with default aes-key and aes-iv
3	aes128cfb with aes-key per product and aes-iv = 0
4	aes128cfb with aes-key per device and aes-iv = 0
5	aes128cfb with aes-key per manufacture and aes-iv = 0
others	无效

HAL_Awss_Get_Conn_Encrypt_Type

原型

```
int HAL_Awss_Get_Conn_Encrypt_Type(void);
```

接口说明

获取零配，热点配网的安全等级。

参数说明

void

返回值说明

值	说明
3	aes128cfb with aes-key per product and aes-iv = random
4	aes128cfb with aes-key per device and aes-iv = random

值	说明
5	aes128cfb with aes-key per manufacture and aes-iv = random
others	无效

HAL_Awss_Open_Monitor

原型

```
void HAL_Awss_Open_Monitor(_IN_ awss_recv_80211_frame_cb_t cb);
```

接口说明

设置Wi-Fi网卡工作在监听（Monitor）模式，并在收到802.11帧的时候调用被传入的回调函数。

参数说明

参数	数据类型	方向	说明
cb	awss_recv_80211_frame_cb_t	输入	回调函数指针，当Wi-Fi接收到帧时会调用此函数

```
/**
 * @brief   802.11帧的处理函数，可以将802.11 Frame传递给这个函数
 *
 * @param[in] buf @n 80211 frame buffer, or pointer to struct ht40_ctrl
 * @param[in] length @n 80211 frame buffer length
 * @param[in] link_type @n AWSS_LINK_TYPE_NONE for most rtos HAL,
 *              and for linux HAL, do the following step to check
 *              which header type the driver supported.
   @verbatim
              a) iwconfig wlan0 mode monitor    #open monitor mode
              b) iwconfig wlan0 channel 6    #switch channel 6
              c) tcpdump -i wlan0 -s0 -w file.pacp    #capture 80211 frame & save
              d) open file.pacp with wireshark or omnipeek
                  check the link header type and fcs included or not
   @endverbatim
 * @param[in] with_fcs @n 80211 frame buffer include fcs(4 byte) or not
 * @param[in] rssi @n rssi of packet, range of [-127, -1]
 */
typedef int (*awss_recv_80211_frame_cb_t)(char *buf, int length,
        enum AWSS_LINK_TYPE link_type, int with_fcs, signed char rssi);
```

返回值说明

void

HAL_Awss_Close_Monitor

原型

```
void HAL_Awss_Close_Monitor(void);
```

接口说明

设置Wi-Fi网卡离开监听（Monitor）模式，并开始以站点（Station）模式工作。

参数说明

void

返回值说明

void

HAL_Wifi_Scan

原型

```
int HAL_Wifi_Scan(awss_wifi_scan_result_cb_t cb);
```

接口说明

启动一次Wi-Fi的空中扫描，该API是一个阻塞操作，扫描没有完成不能结束。所有的AP列表收集完成后，一个一个通过回调函数告知AWSS，最好不要限制AP的数量，否则可能导致中文GBK编码的SSID热点配网失败。

参数说明

参数	数据类型	方向	说明
cb	awss_wifi_scan_result_cb_t	输入	扫描通知回调函数

返回值说明

值	说明
0	扫描正常结束
-1	其他情况

HAL_Awss_Switch_Channel

原型

```
void HAL_Awss_Switch_Channel(
            _IN_ char primary_channel,
            _IN_OPT_ char secondary_channel,
            _IN_OPT_ uint8_t bssid[ETH_ALEN]);
```

接口说明

设置Wi-Fi网卡切换到指定的信道（channel）上。

参数说明

参数	数据类型	方向	说明
primary_channel	char	输入	首选信道

参数	数据类型	方向	说明
secondary_channel	char	输入	辅助信道，信道带宽为40MHz时才会使用，信道宽度为20MHz时忽略该参数
bssid	uint8_t	输入	生活物联网平台SDK1.3.0及以上版本中，此参数已废弃，您若兼容1.3.0以前版本使用

返回值说明

void

HAL_Awss_Connect_Ap

原型

```
int HAL_Awss_Connect_Ap(
            _IN_ uint32_t connection_timeout_ms,
            _IN_ char ssid[HAL_MAX_SSID_LEN],
            _IN_ char passwd[HAL_MAX_PASSWD_LEN],
            _IN_OPT_ enum AWSS_AUTH_TYPE auth,
            _IN_OPT_ enum AWSS_ENC_TYPE encry,
            _IN_OPT_ uint8_t bssid[ETH_ALEN],
            _IN_OPT_ uint8_t channel);
```

接口说明

要求Wi-Fi网卡连接指定热点（Access Point）的函数，bssid指定特定AP，另外bssid也可能为空或无效值（全0或全0xff）。

参数说明

参数	数据类型	方向	说明
connection_timeout_ms	uint32_t	输入	连接AP的超时时间
ssid	char	输入	目的AP的SSID
passwd	char	输入	目的AP的PASSWORD
auth	enum	输入	目的AP的加密方式，HAL可以忽略
encry	enum	输入	目的AP的认证方式，HAL可以忽略
bssid	uint8_t	输入	目的AP的BSSID，该字段可能为NULL或设置为全0
channel	uint8_t	输入	目的AP的信道，该字段可以忽略

返回值说明

值	说明
0	连接AP和DHCP成功
-1	连接AP或DHCP失败

HAL_Wifi_Send_80211_Raw_Frame

原型

```
int HAL_Wifi_Send_80211_Raw_Frame(_IN_ enum HAL_Awss_Frame_Type type,
        _IN_ uint8_t *buffer, _IN_ int len);
```

接口说明

在当前信道（channel）上以基本数据速率（1Mbps）发送裸的802.11帧（raw 802.11 frame）。

参数说明

参数	数据类型	方向	说明
type	enum HAL_Awss_Frame_Type	输入	查看 `HAL_Awss_Frame_Type_t` 定义
buffer	uint8_t *	输入	802.11裸数据帧，包括完整的MAC头和FCS域
len	int	输入	802.11裸帧字节长度

```
/* 80211 frame type */
typedef enum HAL_Awss_Frame_Type {
    FRAME_ACTION,                   // 802.11中的Action帧
    FRAME_BEACON,                  // 802.11中的Beacon帧
    FRAME_PROBE_REQ,             // 802.11中的Probe Request帧
    FRAME_PROBE_RESPONSE,        // 802.11中的Probe Response帧
    FRAME_DATA                    // 802.11中的数据帧
} HAL_Awss_Frame_Type_t;
```

返回值说明

值	说明
0	发送成功
-1	发送失败

HAL_Wifi_Enable_Mgmt_Frame_Filter

原型

```
int HAL_Wifi_Enable_Mgmt_Frame_Filter(
            _IN_ uint32_t filter_mask,
            _IN_OPT_ uint8_t vendor_oui[3],
            _IN_ awss_wifi_mgmt_frame_cb_t callback);
```

接口说明

使能或禁用对特定管理帧的过滤。

参数说明

参数	数据类型	方向	说明
filter_mask	uint32_t	输入	帧过滤参数
vendor_oui	uint8_t	输入	Wi-Fi联盟分配的厂商OUI，如果OUI为NULL，表示不对OUI过滤，反之要根据OUI过滤
callback	awss_wifi_mgmt_frame_cb_t	输入	用于接收802.11帧或者信息元素（IE）的回调函数

返回值说明

值	说明
= 0	发送成功
= -1	发送失败
= -2	不支持

HAL_Sys_Net_Is_Ready

原型

```
int HAL_Sys_Net_Is_Ready();
```

接口说明

检查系统网络是否可用（设备是否已经成功获得IP地址并且当前IP地址可用）。

参数说明

void

返回值说明

值	说明
0	网络不可用
1	网络可用

HAL_Wifi_Get_Ap_Info

原型

```
int HAL_Wifi_Get_Ap_Info(
            _OU_ char ssid[HAL_MAX_SSID_LEN],
            _OU_ char passwd[HAL_MAX_PASSWD_LEN],
            _OU_ uint8_t bssid[ETH_ALEN]);
```

接口说明

获取所连接的热点（Access Point）的信息。

参数说明

参数	数据类型	方向	说明
ssid	char	输出	AP的SSID，该参数可能为NULL
passwd	char	输出	AP的Password，该参数为NULL
bssid	uint8_t	输出	AP的BSSID，该参数可能为NULL，如果bssid不对将导致零配设备发现失败

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Wifi_Get_Link_Stat

原型

```
int HAL_Wifi_Get_Link_Stat(_OU_ int *p_rssi,
                           _OU_ int *p_channel);
```

接口说明

获取当前与AP连接链路状态的信息（channel、RSSI等）。

参数说明

参数	数据类型	方向	说明
p_rssi	char	输出	设备当前关联的AP的信号强度值
p_channel	char	输出	设备当前关联的AP所在的信道

参数	数据类型	方向	说明

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Awss_Open_Ap

原型

```
int HAL_Awss_Open_Ap(const char *ssid,
                     const char *passwd,
                     int beacon_interval,
                     int hide);
```

接口说明

开启设备热点（SoftAP模式）。

参数说明

参数	数据类型	方向	说明
ssid	const char *	输入	热点的ssid字符
passwd	const char *	输入	热点的passwd字符
beacon_interval	int	输入	热点的Beacon广播周期（广播间隔）
hide	int	输入	• 0：非隐藏 • 非0（其它值）：隐藏

返回值说明

值	说明
0	success
-1	unsupported
-2	failure with system error
-3	failure with no memory

值	说明
-4	failure with invalid parameters

HAL_Awss_Close_Ap

原型

```
int HAL_Awss_Close_Ap(void);
```

接口说明

关闭设备热点。

参数说明

void

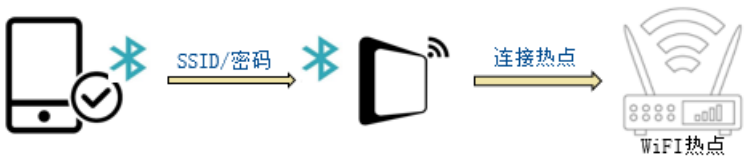
返回值说明

值	说明
0	success
-1	unsupported
-2	failure

8.3. 蓝牙辅助配网开发

蓝牙辅助配网基于已有的WiFi+BLE的Combo芯片方案，利用BLE的通信能力，将Wi-Fi连接所需的SSID、无线密码等信息传输给WiFi+BLE Combo设备，使设备顺利接入互联网与云端，继而完成设备绑定、数据上下行传输等功能。您可以根据生活物联网平台提供的设备端SDK代码、应用示例等，快速集成蓝牙辅助配网功能。

蓝牙辅助配网的方案示意图如下。



生活物联网平台设备端蓝牙辅助配网方案的特性说明如下。

- 数据加密传输：确保物联网复杂环境下的通信安全，保护用户隐私。
- 支持国内与海外环境不同区域的物联网设备接入。
- 支持设备功能自检（包括配网、连云等），并通过移动端App页面实时反馈，增强设备可运维性。
- 商业化高水准的专业测试，保障功能品质。
- 配套提供专门的示例设备应用，便于客户快速评估和验证功能。

- 示例设备应用可支持蓝牙辅助配网和一键配网功能，备用的配网功能增强用户配网的稳定性。
- 示例设备应用支持便捷的串口命令交互功能，适用于不同功能场景的操作使用。
- 支持一机一密的高安全级别的防护，每个设备需烧录生活物联网平台颁发的设备证书。

设备端资源需求

生活物联网平台的蓝牙辅助配网功能模块需运行于支持WiFi + BLE的Combo芯片或模组上，该模块完整功能的支持需要的硬件资源推荐指标列表如下，供您选型时参考。

指标类目	指标项	指标推荐要求
硬件性能类	处理器	• ≥ 32位MCU • 主频 ≥ 100MHz
	内存资源	• Flash ≥ 2MByte • RAM ≥ 256KByte
协议栈指标类	Wi-Fi协议栈	• 802.11 b/g/n 1x1 • 2.4G频段支持必选，5G频段支持可选 • Station模式 • Monitor模式
	BLE协议栈	≥BLE 4.2
生活物联网平台SDK指标类	内存资源	• Flash 300KByte • RAM 50KByte

SDK获取

设备端SDK中包含了蓝牙辅助配网的示例应用，该示例应用代码目录与编译指令在不同设备端SDK版本中区别如下。

SDK版本	版本比较	代码目录	编译指令
1.6.0	• 海外建连速度优化 • 配网时设备异常自检 • 支持全球统一激活	/Products/example/smart_outlet/	**./build.sh example smart_outlet bk7231udevkitc MAINLAND ONLINE 1**
1.3.0	支持设备身份信息设置、蓝牙辅助配网等实用cli指令	/Living_SDK/example/comboapp/	**cd Living_SDK** **aos make clean** **aos make comboapp@bk7231udevkitc btstack=vendor**

SDK版本	版本比较	代码目录	编译指令
1.1.0	初始蓝牙辅助配网版本，设备需使用一机一密的认证方式	/example/comboapp/	aos make clean aos make comboapp@bk7231udevkitc btstack=vendor

固件移植与开发

生活物联网平台SDK中，蓝牙辅助配网移植了部分芯片平台（在控制台的产品**调试设备**页面查询已认证的Combo类型的芯片或模组）。如果是没有经过认证的模组/芯片，则需要您根据开发文档进行移植开发。

- HAL层接口移植

  蓝牙辅助Wi-Fi配网是专用于同时支持BLE+WiFi的设备，因此其功能依赖于底层的BLE协议栈和Wi-Fi协议栈的正确适配。

  - BLE协议栈移植

    生活物联网平台定义了相应的BLE HAL接口，您需将BLE HAL接口根据自己使用的芯片平台进行对接和实现。蓝牙辅助配网的BLE部分是基于生活物联网平台的蓝牙设备端开发，详细请参见蓝牙设备端SDK移植接口。

  - Wi-Fi协议栈移植

    生活物联网平台定义了相应的Wi-Fi HAL接口，您需将Wi-Fi HAL接口根据自己使用的芯片平台进行对接和实现。蓝牙辅助配网的Wi-Fi部分是基于生活物联网平台设备端SDK的，移植详细说明请参见Wi-Fi芯片移植中“HAL移植”与“Wi-Fi和配网移植”的内容。

- 用户编程接口指南

  基于已移植好的BLE和Wi-Fi的协议栈后，生活物联网平台的SDK向上层应用开发提供了相应的用户编程接口，同时生活物联网平台的SDK中也提供了蓝牙辅助配网应用的示例程序，通过对相应的用户编程接口进行调用，完成蓝牙辅助配网的功能。

  蓝牙相关的编程接口说明请参见蓝牙设备端SDK用户编程接口中 `breeze_awss_xxx` 接口。

  蓝牙辅助配网的Wi-Fi部分的编程接口说明请参见基于已认证的模组开发中“Wi-Fi配网”的内容（重点关注连接路由器相关的接口）。

基于该开发指南，生活物联网平台提供了蓝牙辅助配网开发的最佳实践文档，详细请参见Combo设备蓝牙辅助配网适配最佳实践。