设备集成开发指导

文档版本 01

发布日期 2022-06-24

华为终端有限公司

注意

华为会不定期对本文档的内容进行更新。

本文档仅作为使用指导，文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为终端有限公司

地址： 广东省东莞市松山湖园区新城路 2 号

网址： https://consumer.huawei.com

目 录

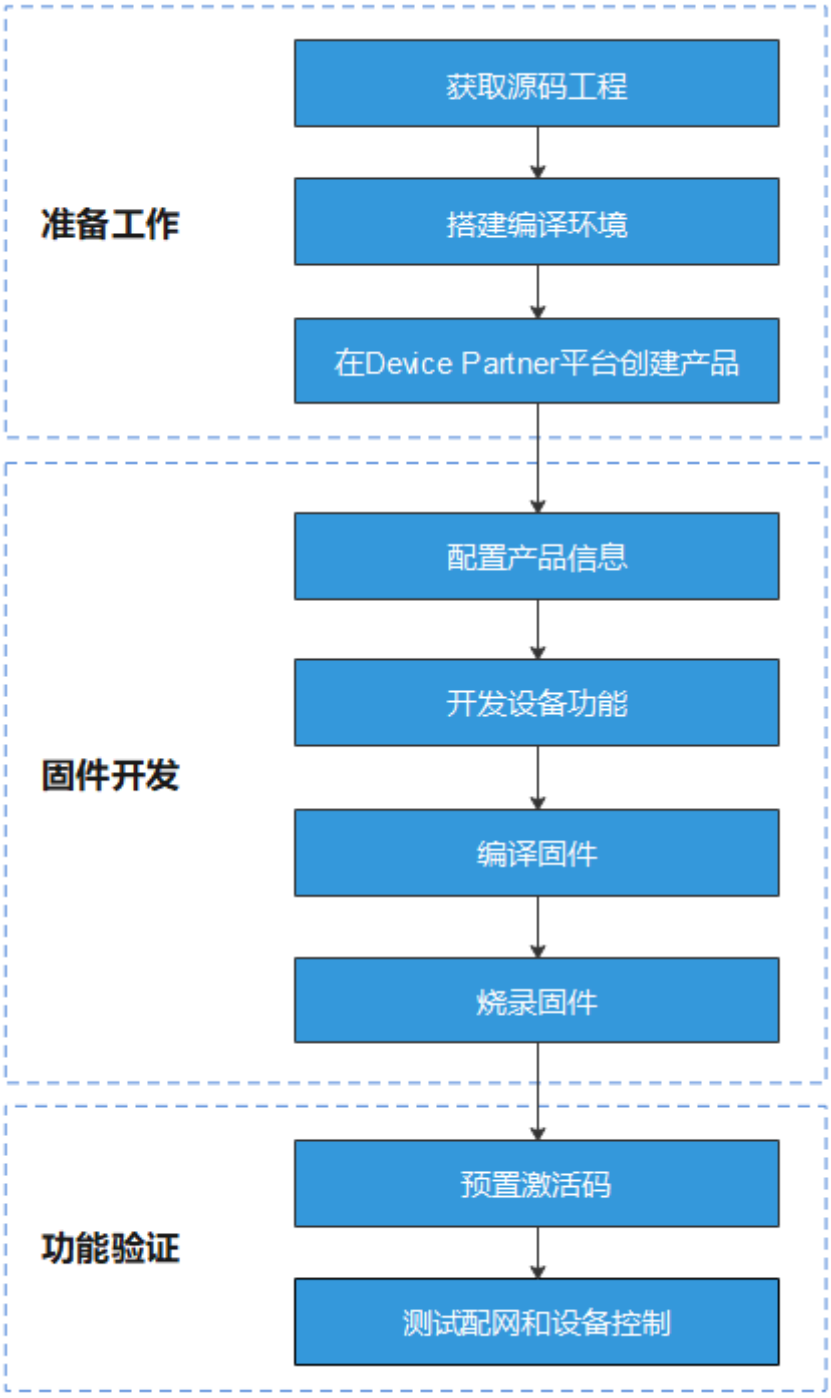
1 概述

简介

本文档为 HarmonyOS Connect 生态产品合作伙伴提供集成指导，旨在帮助伙伴快速熟悉开发流程，完成产品信息配置、配网和设备控制等功能开发，并基于智慧生活 App 进行配网测试和设备控制测试。

开发流程

图1-1 设备集成开发流程



2 文档更新记录

日期	修订版本	修改描述
2022-05-30	2.0	1.【变更】SDK 升级后，接口和产品信息配置有调整，详见 4.3.1 配置 hilink_device.c 中产品信息 和 4.3.2 配置 hilink_device.c 中的服务信息。 2.【新增】设备发现方式增加蓝牙碰一碰和蓝牙靠近发现，详见 4.3.3 配置 hilink_demo.c 中的设备发现方式信息。 3.【新增】设备控制样例代码，详见 4.3.4 实现设备控制功能。 4.【变更】存在 MCU 的情况下，软件可见版本号需要需要包含 MCU 版本，详见 4.2.1 配置 parameter_common.c 中的参数信息。 5.【变更】使用 Debug 版本智慧生活调测需要设置调测环境，详见配置智慧生活 App 测试环境。
2021-10-01	1.0	首次发布。

3 准备工作

步骤 1 联系模组商获取以下工具和信息。

表3-1 工具与信息

资料名称	说明
HarmonyOS Connect 服务包源码工程	用于开发固件，其中，所集成的 HiLink SDK 和 Kit Framework 均需要为 release 版本。
编译工具链和使用指导	用于搭建编译环境。不同芯片采用的编译工具链不同，请联系模组商获取。
烧录工具	用于烧录固件。不同芯片采用的驱动和烧录工具不同，请联系模组商获取。
串口驱动	用于固件烧录过程中 PC 和设备的通信。不同芯片采用的串口驱动不同，请联系模组商获取。
SDK 集成路径	用于配置 HiLink SDK 和 Kit Framework 所需产品信息。
激活码预置方式	不同模组商采用的激活码预置方式不同（AT 指令写入或者烧录工具烧写），请联系模组商确认。

说明

当前已通过认证的芯片和模组参见 HarmonyOS Connect > 芯片与模组。

步骤 2 搭建编译环境。

步骤 3 安装开发板编译环境。

说明

根据开发板实际型号，参考对应的章节进行配置。

步骤 4 在 Device Partner 平台创建产品。

须知

针对不同类型的模组设备，配网方式需要在 Device Partner 平台，“产品开发 > 产品定义 > 软硬件定义 > 极简连接”进行相应的配置。

- Wi-Fi 模组：Hi3861 芯片选择“极速秒控配网”，其他的选择“极速常规配网”。
- Combo 模组：推荐选择“蓝牙辅助配网”。

步骤 5 导出产品信息，用于固件开发中的信息配置。

在 Device Partner 平台的“产品开发”页面，单击“导出”可以获得产品基础信息，如图 3-1 所示。

图3-1 导出产品信息

----结束

4 固件开发

4.1 简介

工程配置项说明

为了保证设备配网、设备控制功能的实现，需要对 HarmonyOS Connect 服务包中的文件进行配置。具体文件及配置项的说明参见表 4-1。

表4-1 HarmonyOS Connect 服务包配置项说明

HarmonyOS Connect 组件名称	文件名称	配置项	用途
Kit Framework	parameter_common.c	操作系统名称、操作系统版本号以及软件版本号	用于产品认证过程中的兼容性测试。
	hal_sys_param.c	产品类型、厂商英文简称、品牌英文名、产品型号等	用于产品认证过程中的兼容性测试，以及设备添加过程中的设备信息校验。
	hal_token.c	产品 ID、AcKey 信息	用于设备添加过程中的设备激活码验证。
HiLink	hilink_d	产品 ID、设备类型 ID、	用于设备配网过程中的 Wi-Fi 信

HarmonyOS Connect 组件名称	文件名称	配置项	用途
SDK	evice.c	厂商 ID、设备类型名、厂商名称、服务信息	息获取，以及设备注册与设备上报时支持的服务列表。

附加参考：API 接口说明

表 4-2 提供了常用 API 接口的功能介绍，供开发者在固件开发时进行参考。

API 接口定义参见“base\startup\syspara_lite\interfaces\kits\parameter.h”。

表4-2 API 接口说明

API 接口	API 返回值的名称	Device Partner 平台的对应字段	举例	说明和要求
GetDeviceType() / GetProductType()	设备类型	-	linkiot ipcamera	最大 32 字符。 物联网设备取固定值 linkiot。
GetManufacture()	企业英文名简称	公司英文名简称	HUAWEI	最大 32 字符。 Kit Framework 认证关键项，和单品激活码强关联。
GetBrand()	品牌英文名称	品牌英文名	HUAWEI	最大 32 字符。 Kit Framework 认证关键项，和单品激活码强关联。
GetMarketName()	外部产品系列名称	产品名称（传播名）	Mate 30	最大 32 字符。 用户可见，Kit Framework 认证关键项。
GetProductSeries()	产品系列英文名称	产品系列	TAS	最大 32 字符。 用以对不同类型产品进行分类管理。Kit Framework 认证关键项。
GetProductModel()	型号	产品型号	TAS-AL00	最大 32 字符。 设备关键信息之一，用以标识设备的类型，用户可见，通常打印在设备铭牌上，用以区分不同产品。该值也是进行 HarmonyOS Connect 认证所需

API 接口	API 返回值的名称	Device Partner 平台的对应字段	举例	说明和要求
				的关键数据，需要采用英文描述。 Kit Framework 认证关键项，和单品激活码强关联。
GetSoftwareModel()	内部软件子型号	-	TAS-AL00	最大 32 字符。 厂商软件型号，厂商自定义，多硬件共软件时区分软件分支。
GetHardwareModel()	硬件版本号	硬件设备版本号	TASAL00 CVN1	最大 32 字符。 厂商定义填写内容。 **对应 Device Partner 平台，认证预约时填写的“硬件设备版本号”。**
GetHardwareProfile()	硬件支持配置	-	{RAM:352K,ROM:2M:WIFI:true}	最大 1000 字符。使用 json 格式字符串表示。根据芯片确定，例如[RAM:352KB,ROM:288KB,WIFI:true]。
GetSerial()	设备序列号	-	随设备差异	最大 64 字符。 SN 非配置，由厂家定义，并保证编号唯一。 Kit Framework 认证关键项，认证后，云端会记录此信息。
GetOsFullName() / GetOsName()	操作系统及版本号	操作系统和操作系统版本号	HarmonyOS-2.0.1.27	最大 64 字符。 名称与版本号之间以"-"连接。
GetDisplayVersion()	用户可见的软件版本号	软件版本号	1.0.0.6	最多 64 字符。 厂商定义填写内容。注意是整个系统的软件版本号而非 HarmonyOS 版本号。 **对应 Device Partner 平台，认证预约时填写的 “软件版本号”。**
GetBootloaderVersion()	Bootloader 版本号	-	u-boot-v2019.07	最多 64 字符。 实际情况填写，例如 XXX-bootloader-v2015.01.03。

API 接口	API 返回值的名称	Device Partner 平台的对应字段	举例	说明和要求
GetSecurityPatchTag()	安全补丁标签	安全补丁级别	2021/1/1	最多 64 字符。 标识当前 OS 的安全补丁级别。按实际情况填写，例如 2020-12-01。 **对应 Device Partner 平台，认证预约时填写的“安全补丁级别”，厂家根据实际情况填写。**
GetAbiList()	Native 接口列表	-	1. riscv-liteos 2. arma7_hard_neon-vfpv4-linux 3. arma7_soft-linux 4. arma7_softfp_neon-vfpv4-linux 5. arma7_hard_neon-vfpv4-liteos 6. arma7_soft-liteos 7. arma7_softfp_neon-vfpv4-liteos	最多 64 字符。 用逗号隔开，仅有生态且生态中包含 native 应用的系统使用。 按实际情况填写，例如 riscv-liteos 这些字段对 kit-framework 没有影响，但是做认证的时候可能会检查，看认证团队的要求。
GetSdkApiVersion()	系统软件 API version	系统软件 API Level / 系统 API Level	3	设备当前版本 API 级别，一般是整数，仅有生态的系统使用。 版本预置值，不需要修改。 **对应 Device Partner 平台，认证预约时填写的“系统 API Level”。**

API 接口	API 返回值的名称	Device Partner 平台的对应字段	举例	说明和要求
GetFirstApiLevel()	设备首版本的系统软件 API level	-	3	一般是整数，仅有生态的系统使用。 版本预置值，不需要修改。
GetIncrementalVersion()	差异版本号	-	1.0.0.6	该版本号是对整个固件实际版本的标识，每次固件更新，均需要更新该版本号。一般取软件版本号即可。 在设备型号确定的情况下，即在"设备类型"+"公司"+"品牌"+"产品系列"+"操作系统及版本号"+"型号"+"内部硬件子型号"+"内部软件子型号"均相同的情况下，可以唯一标识软件版本。 **建议与用户可见软件版本号（GetDisplayVersion()返回结果）保持一致。**
GetVersionId()	版本 ID	版本 ID	-	最大 127 字符。 由多个字段拼接而成：$(设备类型) + '/' + $(公司英文名简称) + '/' + $(品牌英文名称) + '/' + $(产品系列英文名称) + '/' + $(操作系统及版本号) + '/' + $(型号) + '/' + $(内部软件子型号) + '/' + $(系统软件 API level) + '/' + $(差异版本号) + '/' + $(构建类型)　在所有厂家的所有设备范围中，可以唯一标识版本。
GetBuildType()	构建类型	-	release:nolog	最多 32 字符。 同一基线代码的不同构建类型，比如 debug/release、log/nolog 可以用多个标识，分号分隔。
GetBuildUser()	构建 user	-	-	最多 32 字符。
GetBuildHost()	构建 host	-	-	最多 32 字符。
GetBuildTi	构建时间	-	-	最多 32 字符。

API 接口	API 返回值的名称	Device Partner 平台的对应字段	举例	说明和要求
me()				Epoch Time，自 1970 年至今的秒数；例如 1294902266。
GetBuildRootHash()	版本 Hash	版本 Hash	-	默认为空即可。对应代码中填入空串（""）即可。

4.2 配置产品信息

4.2.1 配置 parameter_common.c 中的参数信息

参见表 4-3，配置 parameter_common.c 文件中的产品信息。

```
static char g_roBuildOs[] = {"OpenHarmony-1.1.0"}; // 操作系统和操作系统版本号，中间用"-"连接，例如：OpenHarmony-1.1.0
static char g_roBuildVerShow[] = {"1.0.1"}; // 用户可见的软件版本号，需确保与 Device Partner 平台上填写的软件版本号一致。
```

表4-3 配置项说明

配置项	说明	示例
g_roBuildOs	操作系统和操作系统版本号，使用“-”连接。 获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看操作系统和操作系统版本号。	OpenHarmony-1.1.0
g_roBuildVerShow	用户可见的软件版本号，需确保与 Device Partner 平台上预约认证时填写的“软件版本号”一致。 参见图 6-2。 • 如果包含 MCU，软件版本号的格式为：*模组固件版本号*(*MCU 版本号*)。 • 如果不含 MCU，软件版本号与模组固件版本号 （FIRMWARE_VER）相同。	• **示例 1（含 MCU）：** 固件版本号是 v1.1.0，MCU 版本号是 R-1.0，用户可见软件版本号为 v1.1.0(R-1.0)。 • **示例 2（不含 MCU）：** 固件版本号是 v1.1.0，用户可见软件版本号为

配置项	说明	示例
		v1.1.0。

说明

差异版本号（GetIncrementalVersion()）的取值建议与软件版本号保持一致。

4.2.2 配置 hal_sys_param.c 中的参数信息

参考表 4-4 配置产品详细信息，对 Kit Framework 认证结果有直接影响。企业英文名简称、品牌英文名、产品型号和认证过程强相关，信息不匹配会导致认证结果失败。

```
static const char OHOS PRODUCT TYPE[] = {"linkiot"}; // 固定值
static const char OHOS MANUFACTURE[] = {"Test"}; // 企业英文名简称
static const char OHOS BRAND[] = {"HiTest"}; // 品牌英文名
static const char OHOS MARKET NAME[] = {"SmartLight"}; // 产品名称（传播名），包含汉字时建议使用 Unicode，避免乱码问题
static const char OHOS PRODUCT SERIES[] = {"Light"}; // 产品系列
static const char OHOS_PRODUCT_MODEL[] = {"HiTest0728"}; // 产品型号
static const char OHOS_SOFTWARE_MODEL[] = {"1.0.0"}; // 软件版本
static const char OHOS_HARDWARE_MODEL[] = {"1.0.0"}; // 产品硬件版本号，需要与模组硬件版本号（HARDWARE_VER）保持一致
static const char OHOS_HARDWARE_PROFILE[] = {"ROM:352K,RAM:2M,WIFI:true"}; // 系统能力，参考产品信息介绍
static const char OHOS_BOOTLOADER_VERSION[] = {"bootloader"};
static const char OHOS_SECURITY_PATCH_TAG[] = {"2020-09-01"};
static const char OHOS_ABI_LIST[] = {"riscv-liteos"};
static const char OHOS_SERIAL[] = {"1234567890"}; // 厂商自定义，保持唯一。实际使用中需要厂商产线逐个设备写入，并在 GetSerial()接口返回设备序列号。
```

如果需要动态获取信息，比如从硬件存储或者 mcu 获取信息，可以修改 HalGet 开头的对应的方法。

表4-4 配置项说明

配置项	说明	示例
OHOS_PRODUCT_TYPE	设备类型，取固定值“linkiot”。	linkiot
OHOS_MANUFACTURE	企业英文名简称，申请激活码后不可修改。获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“厂商信息”页签下，可以查看“企业英文名简称”。	HUAWEI
OHOS_BRAND	品牌英文名，申请激活码后不可修改。获取方式如下： 1. 在 Device Partner 平台的“产品开	HUAWEI

配置项	说明	示例
	发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看“品牌英文名”。	
OHOS_MARKET_NAME	外部产品名称，用户可见。获取方式如下： 在 Device Partner 平台的“产品开发”页面，单击“编辑”图标，进入产品信息界面，可以查看“产品名称（传播名）”。	Mate 30
OHOS_PRODUCT_SERIES	产品系列英文名称。获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看“产品系列”。	TAS
OHOS_PRODUCT_MODEL	型号。获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看“产品型号”。	TAS-AL00
OHOS_SOFTWARE_MODEL	内部软件子型号。厂商自定义。	TAS-AL00
OHOS_HARDWARE_MODEL	硬件版本号。厂商自定义。	TASAL00CVN1
OHOS_HARDWARE_PROFILE	硬件配置。厂商自定义，根据设备实际支持功能配置。	{RAM:352K,ROM:2M,WIFI:true}
OHOS_SECURITY_PATCH_TAG	安全补丁标签。厂商自定义。	2021/1/1
OHOS_ABI_LIST	Native 接口列表。取固定值“riscv-liteos”	riscv-liteos
OHOS_SERIAL	设备序列号。厂商自定义。实际使用中需要厂商产线逐个设备写入，并在GetSerial()接口返回设备序列号。	-

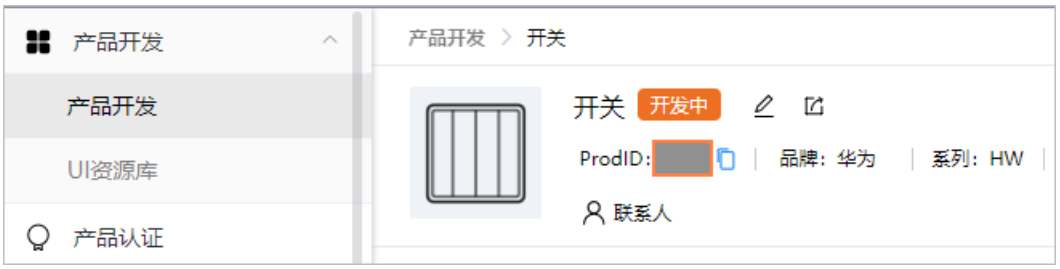
4.2.3 配置 hal_token.c 中的参数信息

4.2.3.1 配置产品 ID 信息

步骤 1 获取产品 ID 信息。

在 Device Partner 平台的“产品开发”页面，选择对应产品。在产品开发界面，可以查看 ProdID。

图4-1 产品 ID 信息



步骤 2 修改产品 ID 信息。

在函数 OEMGetProdId 中，修改产品 ID 信息。

```
static int OEMGetProdId(char *productId, unsigned int len)
```

----结束

4.2.3.2 配置 AcKey 信息

步骤 1 获取 AcKey。具体方法参考步骤 5，在导出的信息可以查找到 AcKey 的取值。

步骤 2 配置 AcKey。

1. 调整十六进制文本，每个字节以“0x”开头。

   例如，获取到的 AcKey 取值为“1122334455667788 99AABBCCDDEEFF11223344556677889 9AABBCCDDEEFF11223344556677889 9AABBCCDDEEFF112233”，调整后的取值为：

   ```
   0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD,
   0xEE, 0xFF, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99, 0xAA, 0xBB,
   0xCC, 0xDD, 0xEE, 0xFF, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
   0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x11, 0x22, 0x33
   ```

2. 在函数中，修改 acKeyFromOS，替换十六进制文本信息。

   ```
   static int OEMGetAcKey(char *acKey, unsigned int len)
   ```

----结束

4.3 开发设备功能

4.3.1 配置 hilink_device.c 中产品信息

配置产品基本信息，用于设备配网和设备注册。

```
/* 设备产品 ID */
static const char *PRODUCT_ID = "9ABC"; // 产品 ID，必须和产品真实信息一致
/* 设备产品子型号 ID */
static const char *SUB_PRODUCT_ID = ""; // 产品子型号，无需填写
/* 设备类型 ID */
static const char *DEVICE_TYPE_ID = "xxx";// 产品类型 ID，必须和产品真实信息一致
/* 设备类型英文名称 */
static const char *DEVICE_TYPE_NAME = "xxxxx"; // 品类英文名，与 Device Partner 平台的“集成开发”页面中“无线网络名称（SSID）”的品类英文名保持一致
/* 设备制造商 ID */
static const char *MANUAFACTURER_ID = "xxx"; // 厂商 ID，必须和产品真实信息一致
/* 设备制造商英文名称 */
static const char *MANUAFACTURER_NAME = "XXXX";  // 品牌名，与 Device Partner 平台的“集成开发”页面中“无线网络名称（SSID）”的品牌名保持一致
/* 设备型号 */
static const char *PRODUCT_MODEL = "test123456"; // 产品型号，必须和产品真实信息一致
/* 设备 SN */
static const char *PRODUCT_SN = ""; // 设备 SN 号，建议与 GetSerial()接口返回指保持一致
/* 设备固件版本号 */
static const char *FIRMWARE_VER = "1.0.0"; // 模组固件版本号
/* 设备硬件版本号 */
static const char *HARDWARE_VER = "1.0.0"; // 模组硬件版本号。对应智慧生活 App 中设备版本信息显示的硬件版本。需要与产品硬件版本号（OHOS_HARDWARE_MODEL）保持一致
/* 设备软件版本号 */
static const char *SOFTWARE_VER = "1.0.0"; // 对应智慧生活 App 中设备版本信息显示的 SDK 版本，即 HiLink SDK 版本。例如 12.0.0.303（默认值即可，无需修改，HiLink SDK 会自动替换该版本号）
```

图4-2 SSID 配置

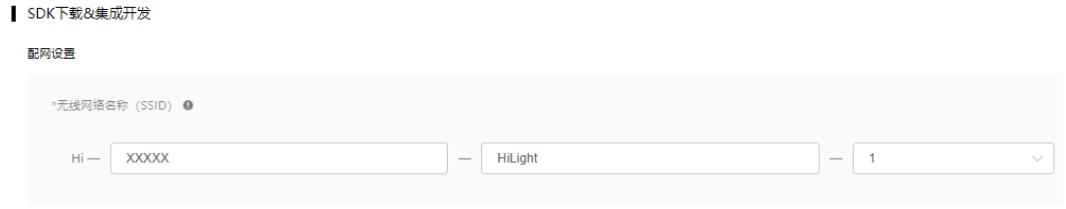


表4-5 配置项说明

配置项	说明	示例
FIRMWARE_VER	模组固件版本号。 模组固件版本号与软件版本号的关系，参见表 4-3 中用户可见版本号介绍。	-
SOFTWARE	对应智慧生活 App 中设备版本信息显示的 SDK	-

配置项	说明	示例
_VER	版本，即 HiLink SDK 版本。例如 12.0.0.303（默认值即可，无需修改，HiLink SDK 会自动替换该版本号）。	
HARDWARE_VER	模组硬件版本号。对应智慧生活 App 中设备版本信息显示的硬件版本。需要与产品硬件版本号（OHOS_HARDWARE_MODEL）保持一致。	-
PRODUCT_ID	产品 ID，参考准备工作步骤 5。	-
DEVICE_TYPE_ID	产品类型 ID，参考准备工作步骤 5。	-
MANUAFACTURER_ID	厂商 ID。获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看 ManufactureID。	-
PRODUCT_MODEL	产品型号。获取方式如下： 1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。 2. 单击右上角的“详情”，在“产品信息”页签下，可以查看产品型号。	-
DEVICE_TYPE_NAME	设备类型名。参考图 4-2，需要和网站 SSID 保持一致。	Light
MANUAFACTURER_NAME	产商名称。参考图 4-2，需要和网站 SSID 保持一致	HUAWEI

4.3.2 配置 hilink_device.c 中的服务信息

为了确保设备控制功能的正常使用，需要将 Profile 中定义的设备功能，配置在 hilink_device.c 中。Profile 中默认添加的功能（例如 ota、netinfo 等），无需在 hilink_device.c 文件中配置。

须知

请确保信息配置正确，否则会导致设备信息校验失败，出现设备反复上线/下线的现象。

步骤 1 获取产品 Profile 文件。

1. 在 Device Partner 平台的“产品开发”页面，选择对应产品。
2. 在“产品定义 > 物模型定义”页面，单击右侧的“下载 Profile”。

步骤 2 在 hilink_device.c 文件中配置设备功能。

以门锁为例介绍如何配置设备功能，假定产品 profile 信息如表 4-6 所示，则对应的工程代码示例如下：

表4-6 产品 profile 信息（节选）

服务 sid	服务(中文)	服务类型 ServiceType
lockState	门在线/离线	state
lockMode	防护模式状态	mode

```
/* 服务信息定义 */
static const HILINK SvcInfo SVC INFO[] =
{
    { "state", "lockState"},
    { "mode", "lockMode"}
};
```

须知

定义服务信息时的结构顺序必须遵循：先服务类型 **ServiceType**、后服务 **sid** 的顺序。否则，会导致功能无法生效。

关于结构体 HILINK_SvcInfo 的定义，可以在“hilink_device.h”中查看。

```
typedef struct {
    char svcType[32];  /* 服务类型，长度范围(0, 32] */
    char svcId[64];    /* 服务 ID，长度范围(0, 64] */
} HILINK_SvcInfo;
```

----结束

4.3.3 配置 hilink_demo.c 中的设备发现方式信息

4.3.3.1 设备发现方式简介

设备发现的方式包括 NFC 碰一碰、蓝牙碰一碰、蓝牙靠近发现这三种。伙伴需要根据产品定义时选择的极简交互方式不同，配置不同的信息。

不同设备发现方式适用的 SDK 和模组要求如下：

表4-7 设备发现方式的相关要求

设备发现方式	HiLinkSDK 最低版本	模组要求	已适配芯片
NFC 碰一	12.0.0.303	Combo 或 Wi-Fi	ALL

设备发现方式	HiLinkSDK 最低版本	模组要求	已适配芯片
碰		模组	
蓝牙碰一碰	12.0.5.302	Combo 模组	BL602C
蓝牙靠近发现	12.0.5.302	Combo 模组	BL602C

4.3.3.2 NFC 碰一碰

1. 确认已完成 NFC 标签认证。
2. 若选择的配网方式为蓝牙辅助配网，按照蓝牙 BLE 设备接入规范，对外发送未注册常态广播报文，报文格式参考：

表4-8 未注册常态广播报文格式

长度	类型	值	说明
0x02	0x01	0x06	BLE 可被发现

示例代码如下：

```
// BLE 设备可被发现
unsigned char myadvData_0010[] = {
    0x2, 0x1, 0x6
};
```

3. 当设备未注册时，智慧生活 App 可以扫描广播添加设备，设备侧需要发送未注册常态广播。未注册常态广播的蓝牙广播响应数据(myRspData)具体字段参考表 4-13。

示例代码如下：

```
unsigned char myrspData 0001[] = {
    0x15, 0x9,  'H', 'i', '-', 'H', 'U', 'A', 'W', 'E', 'I' , '-', 0x31,
    demoDevInfo->productId[0], demoDevInfo->productId[1], demoDevInfo-
>productId[2], demoDevInfo->productId[3],
    demoDevInfo->subProductId[0], demoDevInfo->subProductId[1],
    demoDevInfo->sn[8], demoDevInfo->sn[9], demoDevInfo->sn[10],demoDevInfo-
>sn[11]
};
```

4. 当设备注册成功后，即可停止发送广播报文。

示例代码如下：

```
void HILINK NotifyDevStatus(int status)
{
    switch (status) {
        case HILINK M2M CLOUD OFFLINE:
            /* 设备与云端连接断开，请在此处添加实现 */
```



```
        printf("----------HILINK_M2M_CLOUD_OFFLINE----------\r\n");
        break;
    case HILINK_M2M_CLOUD_ONLINE:
        /* 设备连接云端成功，请在此处添加实现 */
        printf("----------HILINK_M2M_CLOUD_ONLINE----------\r\n");
        BLE_CfgNetDeInit(NULL, 1);
        break;
    case HILINK_M2M_LONG_OFFLINE:
        /* 设备与云端连接长时间断开，请在此处添加实现 */
        printf("----------HILINK_M2M_LONG_OFFLINE----------\r\n");
        break;
    case HILINK_M2M_LONG_OFFLINE_REBOOT:
        /* 设备与云端连接长时间断开后进行重启，请在此处添加实现 */
        printf("----------HILINK_M2M_LONG_OFFLINE_REBOOT----------\r\n");
        break;
    case HILINK_UNINITIALIZED:
        /* HiLink 线程未启动，请在此处添加实现 */
        printf("----------HILINK_UNINITIALIZED----------\r\n");
        break;
    case HILINK_LINK_UNDER_AUTO_CONFIG:
        /* 设备处于配网模式，请在此处添加实现 */
        printf("----------HILINK_LINK_UNDER_AUTO_CONFIG----------\r\n");
        break;
    case HILINK LINK CONFIG TIMEOUT:
        /* 设备处于 10 分钟超时状态，请在此处添加实现 */
        printf("----------HILINK LINK CONFIG TIMEOUT----------\r\n");
        break;
    case HILINK LINK CONNECTTING WIFI:
        /* 设备正在连接路由器，请在此处添加实现 */
        printf("----------HILINK LINK CONNECTTING WIFI----------\r\n");
        break;
    case HILINK LINK CONNECTED WIFI:
        /* 设备已经连上路由器，请在此处添加实现 */
        printf("----------HILINK LINK CONNECTED WIFI----------\r\n");
        break;
    case HILINK M2M CONNECTTING CLOUD:
        /* 设备正在连接云端，请在此处添加实现 */
        printf("----------HILINK M2M CONNECTTING CLOUD----------\r\n");
        break;
    case HILINK LINK DISCONNECT:
        /* 设备与路由器的连接断开，请在此处添加实现 */
        printf("----------HILINK LINK DISCONNECT----------\r\n");
        break;
    case HILINK DEVICE REGISTERED:
        /* 设备被注册，请在此处添加实现 */
        printf("----------HILINK DEVICE REGISTERED----------\r\n");
        break;
    case HILINK DEVICE UNREGISTER:
        /* 设备被解绑，请在此处添加实现 */
        printf("----------HILINK DEVICE UNREGISTER----------\r\n");
        break;
    case HILINK REVOKE FLAG SET:
        /* 设备被复位标记置位，请在此处添加实现 */
        printf("----------HILINK REVOKE FLAG SET----------\r\n");
        reset_kitfwk();
```



```
            break;
        case HILINK_NEGO_REG_INFO_FAIL:
            /* 设备协商配网信息失败 */
            printf("----------HILINK_NEGO_REG_INFO_FAIL----------\r\n");
            break;
        case HILINK_LINK_CONNECTED_FAIL:
            /* 设备与路由器的连接失败 */
            printf("----------HILINK_LINK_CONNECTED_FAIL----------\r\n");
            break;
        default:
            break;
    }

    return;
}
```

综上，NFC 碰一碰，通过蓝牙辅助配网发现并注册设备的示例代码如下：

```
void main(void)
{
    HILINK SetNetConfigMode(HILINK NETCONFIG OTHER);

    BLE ConfPara isBlePair;
    BLE InitPara initPara;
    BLE AdvInfo ble adv info;

    memset(&isBlePair, 0x00, sizeof(BLE ConfPara));
    memset(&initPara, 0x00, sizeof(BLE InitPara));
    memset(&ble adv info, 0x00, sizeof(BLE AdvInfo));

    unsigned char myadvData 0010[] = {
        0x2, 0x1, 0x6
    };

    unsigned char myrspData 0001[] = {
        0x15, 0x9,  'H', 'i', '-', 'H', 'U', 'A', 'W', 'E', 'I' , '-', 0x31,
        demoDevInfo->productId[0], demoDevInfo->productId[1], demoDevInfo-
>productId[2], demoDevInfo->productId[3],
        demoDevInfo->subProductId[0], demoDevInfo->subProductId[1],
        demoDevInfo->sn[8], demoDevInfo->sn[9], demoDevInfo->sn[10],demoDevInfo-
>sn[11]
    };

    BLE_AdvPara advPara = {
        .advType = 0x00, //HILINK_BT_ADV_TYPE_IND
        .minInterval = 0x20,
        .maxInterval = 0x40,
        .channelMap = 0x07, //HILINK_ADV_CHNL_ALL
    };

    BLE_AdvData advData = {
            .advData = myadvData_0010,
            .advDataLen = sizeof(myadvData_0010),
            .rspData =  myrspData_0001,
            .rspDataLen = sizeof(myrspData_0001),
    };
```



```
    BLE_CfgNetCb BleCfgNetCb = {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL};

    ble_adv_info.advPara = &advPara;
    ble_adv_info.advData = &advData;

    isBlePair.isBlePair = 0;
    initPara.confPara = &isBlePair;
    initPara.advInfo = &ble_adv_info;
    BLE_CfgNetInit(&initPara, &BleCfgNetCb);
    BLE_CfgNetAdvCtrl(0XFFFFFFFF);

    HILINK_Main();
}
```

4.3.3.3 蓝牙碰一碰

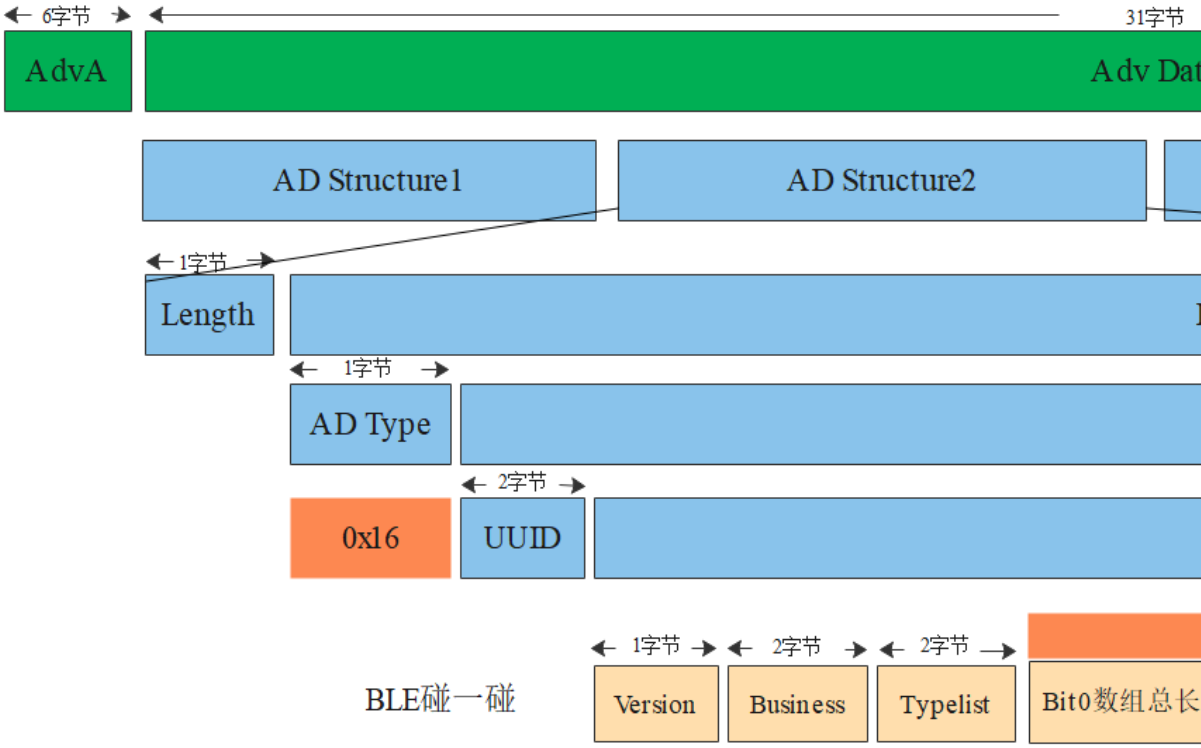
蓝牙碰一碰广播规范

在蓝牙碰一碰中，BLE 广播分为一碰广播、二碰广播。BLE 碰一碰不涉及常态广播。碰一碰广播默认发送，广播间隔根据实际设备自定义。碰一碰广播规范定义了广播包（Advertising Data）和响应包（Scan Response），响应包（Scan Response）装载了蓝牙广播名称，与蓝牙靠近发现相同。

- 一碰广播为设备未注册时发送的碰一碰广播，此时设备与手机碰一碰拉起 FA，请求 FA 进行设备注册，ProtocolD 值为 0x05。
- 二碰广播为设备注册成功后发送的碰一碰广播，此时设备与手机碰一碰后，手机 FA 直接进入设备控制页，ProtocolD 值为 0x00/0x01。
- 设备发送一碰广播时，使用智慧生活 App 可以扫描添加设备。

BLE 碰一碰广播结构和 BLE 靠近发现广播整体结构类似，在 Huawei Spec Data 之前的内容二者一致，区别在于 Huawei Spec Data 中 Version 和 Business 字段后，靠近发现协议定义多个 TV 字段，而 BLE 碰一碰则定义了一个 TypeList 后面跟一组 Value 信息。

碰一碰广播包（Advertising Data）结构如下图所示：



说明

AdvA 为 6 字节的 MAC 地址，不占用广播包的长度，此处需要使用 Public MAC，不能使用随机 MAC。

基于上述的广播包结构，我们需要首先了解蓝牙碰一碰广播包规范如表 6 蓝牙碰一碰广播包规范。

表4-9 蓝牙碰一碰广播包规范

内容		长度（Byte）	值	必选/可选	说明
AD Structure1	Length	1	0x 02	必选	类型和值的总长度
	AD Type	1	0x 01	必选	0x01 表示 flag
	Value	1	0x 06	必选	支持的 LE/BE/EDR 的 Flag 信息
AD Structure2	Length	1	0x XX	必选	类型和值的总长度
	AD Type	1	0x 16	必选	广播类型，0x16 表示蓝牙服务数据

内容				长度（Byte）	值	必选/可选	说明
	UUID			2	0xEEFD	必选	华为购买的 UUID，0xFDEE（已购买）小端存储
	Huawei Spec Data	协议版本		1	0x 01	必选	华为蓝牙广播协议版本
		business		1	0x 06	必选	06 表示碰一碰业务
		businessEx		1	0x 00	必选	默认为 0x00
		typelist		2	0x00xx	必选	根据所需业务使能对应 bit 位，碰一碰传 0x1700（使能的 bit0、bit1、bit2、bit4） • bit0：自定义数据，不定长，Length+Value 格式 • bit1：prodId • bit2：Sub prod ID • bit4：AdvPower
		bit0 Length		1	0x0A		bit0 的长度
		bit0 Value	Protocol D T	1	0x 17	必选	蓝牙协议 ID 类型
			Protocol D L	1	0x 01	必选	蓝牙协议 ID 长度
			Protocol D V	1	0x XX	必选	蓝牙 ID 值 • bit0：0 表示 member 二碰不弹框，1 表示 member 二碰弹框 • bit1：预留字段，默认填 0 • bit2~bit7 组成的 int 值标识事件类型，字段说明如下： – 0-弹设备控制框 – 1-WiFi+BLE

内容				长度（Byte）	值	必选/可选	说明
							Combo 类型设备请求首次进行配网，下发 ssid 、密码和注册信息。 – 2-直连云设备（插网线、蜂窝等）和已由 App 代理注册的 BLE 设备检测到直连云通道已建立，通过 BLE 下发注册信息 打通直连云通道。 – 3-WiFi+BLE 的 Combo 设备，已经由蓝牙辅助 WiFi 配网注册成功，目前 WiFi 连不上路由器，请求 owner 更换路由器 SSID 和密码（member 不弹框）。 – 4-设备有异常告警事件，发送告警广播。 – 5-纯蓝牙 BLE 设备请求 App 进行代理注册。
			SN T	1	0x 14	必选	设备 SN
			SN L	1	0x 02	必选	设备 SN 长度
			SN V	2	0x XXXX	必选	设备 SN 。设备 SN 最后 2 字节的 ASCII 码值，必须和 deviceinfo 上报的 SN 最后两位一致

内容				长度（Byte）	值	必选/可选	说明
			业务自定义	1	0x 91	可选	以下为业务自定义数据
			L	1	自定义 L	可选	业务自定义字段 L
			V	变长	自定义 V	可选	业务自定义字段 V
		prodId V		4	0x XXXXXXXX	必选	产品 ID 值，产品 ID 的 ASCII 码值。
		subProdId V		1	0x XX	必选	子产品 ID 值，默认 00
		AdvPower V		1	0x XX	必选	广播的实际发射功率 AdvPower TRP = TxPower （设备芯片广播发射功率） - OTA（设备天线损耗），取值【-128，127】dBm，建议用芯片可配的最小发射功率以节省功耗。例如芯片的广播 TxPower 为-6dBm，天线 OTA 是 10dBm，则 Adv TRP 配置为-16dBm，取值为 0xF0。

表4-10 BLE 设备碰一碰广播实例

适用场景	长度	类型	值	说明
以下每个场景都携带	0x02	0x01	0x06	BLE 可被发现
WiFi+BLE Combo 类型设备请求 App 进行设备注册（一碰广播）	0x16	0x16	0xEEFD0106001700071402NNNN170105XXXXXXXXSSPP	未注册，SS 表示产品子型号；PP 表示发射功率值；XXXXXXXX 表示产品 ID 的 ASCII 码值；NNNN 是 SN 后两位的 ASCII 码值。

适用场景	长度	类型	值	说明
二碰广播，owner 弹框，member 不弹框	0x16	0x16	0xEEFD01060017000714 02NNNN170100XXXXX XXXSSPP	已注册，SS 表示产品子型号；PP 表示发射功率值；XXXXXXXX 表示产品 ID 的 ASCII 码值；NNNN 是 SN 后两位的 ASCII 码值。
二碰广播，owner、member 都弹框	0x16	0x16	0xEEFD01060017000714 02NNNN170101XXXXX XXXSSPP	已注册，SS 表示产品子型号；PP 表示发射功率值；XXXXXXXX 表示产品 ID 的 ASCII 码值；NNNN 是 SN 后两位的 ASCII 码值。
二碰广播，离线更换配网信息	0x16	0x16	0xEEFD01060017000714 02NNNN170C01XXXXX XXXSSPP	已注册断网，用 PP 功率值触发离线弹窗；NNNN 是 SN 后两位，仅支持 owner 弹框不支持 member 弹框。

配置蓝牙碰一碰的设备发现方式信息

若产品定义时选择的极简交互方式为蓝牙碰一碰，产品伙伴需要调用 HiLink SDK 接口 BLE_SetAdvType，对外广播蓝牙报文。

示例代码如下：

```
static BLE_ConfPara g_isBlePair = {
    .isBlePair = 1,
};
static BLE_InitPara g_bleInitParam = {
    .confPara = &g_isBlePair,
    /* advInfo 为空表示使用 ble_sdk 默认广播参数及数据 */
    .advInfo  = NULL,
    .gattList = NULL,
};
/* APP 下发自定义指令时调用此函数，需处理自定义数据，返回 0 表示处理成功 */
static int BleRcvCustomData(unsigned char *buff, unsigned int len)
{
    printf("custom data, len: %u, data: %s\r\n", len, buff);
    /* 处理自定义数据 */
    return 0;
}

static BLE_CfgNetCb g_bleCfgNetCb = {
    .rcvCustomDataCb = BleRcvCustomData,
};
void main(void)
{
     HILINK_SetNetConfigMode(HILINK_NETCONFIG_OTHER);
     /* 设置广播类型为蓝牙碰一碰 ，此函数必须在 BLE_CfgNetInit 之前调用*/
     BLE_SetAdvType(BLE_ADV_ONEHOP);
```



```
    HILINK_Main();
    /* BLE 配网资源申请：BLE 协议栈启动、配网回调函数 */
    int ret = BLE_CfgNetInit(&g_bleInitParam, &g_bleCfgNetCb);
    if (ret != 0) {
        printf("ble cfg net init fail");
        return ret;
    }
    /* BLE 配网广播控制：参数代表广播时间，0:停止，0xFFFFFFFF:一直广播，其他：广播指定时间后停止，单位秒 */
    ret = BLE_CfgNetAdvCtrl(180);
    if (ret != 0) {
        printf("ble cfg net adv ctrl fail\r\n");
        return ret;
    }
}
```

4.3.3.4 蓝牙靠近发现

蓝牙靠近发现广播规范

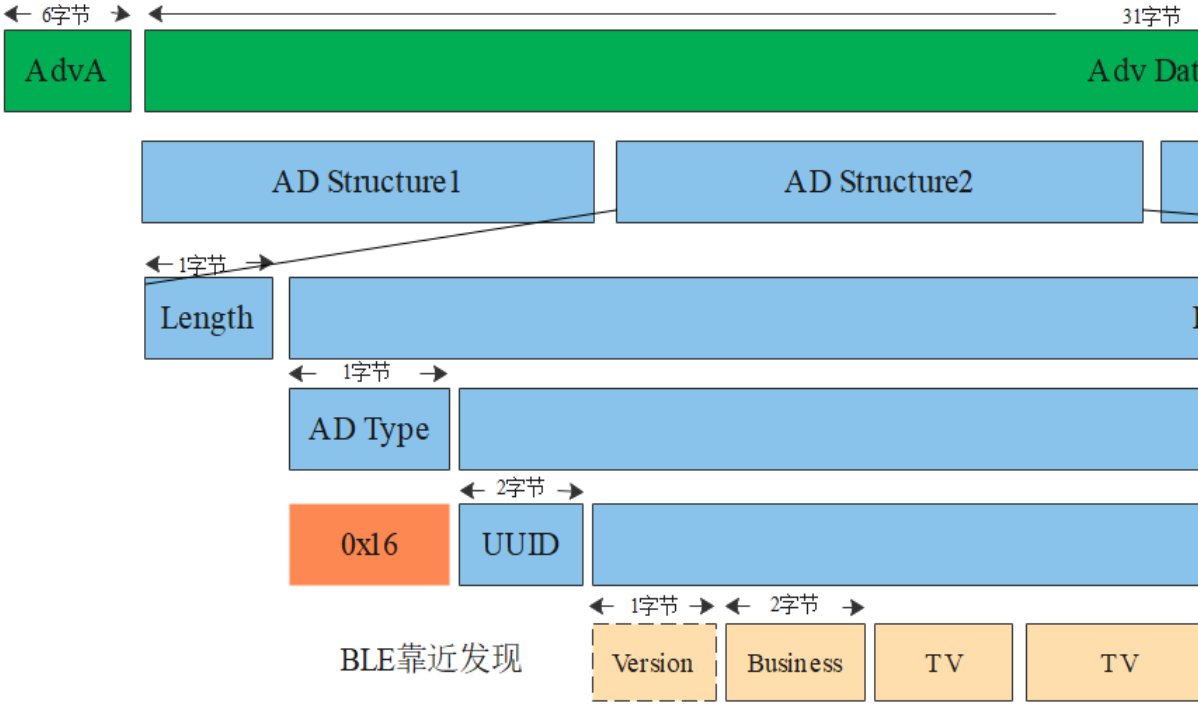
在蓝牙靠近发现中，BLE 广播分为靠近发现弹框 FA 的广播（简称靠近发现广播）和常态广播。为避免骚扰用户，靠近发现广播需要主动触发（开机/上电/双击等）才能发送。靠近发现广播规范定义了广播包（Advertising Data）和响应包（Scan Response），根据 Advertising Data 可分为一靠广播和二靠广播；常态广播规范只定义了响应包（Scan Response），也就是蓝牙广播名称。

- 一靠广播为设备未注册时发送的靠近发现广播，此时设备与手机靠近后拉起 FA，请求 FA 进行设备注册，ProtocolD 值为 0x05。
- 二靠广播为设备注册成功后发送的靠近发现广播，此时设备与手机靠近后，手机拉起 FA 直接进入设备控制页，ProtocolD 值为 0x00/0x01。
- 未注册常态广播为设备未被注册，且用户未主动触发时的广播。
- 设备发送一靠广播或者未注册常态广播时，使用智慧生活 App 可以扫描添加设备。

说明

当设备支持分享时，设备的分享者称为 owner，被分享者称为 member。ProtocolD 值为 0x00 时表示仅 owner 会弹框，member 不会弹框；ProtocolD 值为 0x01 时表示 owner 和 member 都会弹框。

靠近发现广播包（Advertising Data）结构如下图所示：



说明

AdvA 为 6 字节的 MAC 地址，不占用广播包的长度,此处需要使用 Public MAC，不能使用随机 MAC。

基于上述的广播包结构，我们需要首先了解蓝牙靠近发现广播包规范如表 2 蓝牙靠近发现广播包规范。

表4-11 蓝牙靠近发现广播包规范

内容		长度（Byte）	值	必选/可选	说明
AD Structure1	Length	1	0x 02	必选	类型和值的总长度
	AD Type	1	0x 01	必选	0x01 表示 flag
	Value	1	0x 06	必选	支持的 LE/BE/EDR 的 Flag 信息
AD Structure2	Length	1	0x XX	必选	类型和值的总长度
	AD Type	1	0x 16	必选	广播类型，0x16 表示蓝牙服务数据
	UUID	2	0xEEFD	必	华为购买的 UUID，0xFDEE（已购买）小端存

内容			长度（Byte）	值	必选/可选	说明
					选	储
	Huawei Spec Data	协议版本	1	0x 01	必选	华为蓝牙广播协议版本
		business	1	0x 01	必选	01 表示靠近发现业务
		businessEx	1	0x 0D	必选	0x0D 表示拉起 FA
		subProdIdT	1	0x 04	必选	子产品 ID 类型
		subProdIdV	1	0x XX	必选	子产品 ID 值，默认 00
		AdvPowerT	1	0x 11	必选	测距字段类型
		AdvPowerV	1	0x XX	必选	广播的实际发射功率 AdvPower TRP = TxPower（设备芯片广播发射功率） – OTA（设备天线损耗），取值【-128，127】dBm，建议用芯片可配的最小发射功率以节省功耗。 例如：芯片的广播 TxPower 为-6dBm，天线 OTA 是 10dBm，则 Adv TRP 配置为-16dBm，取值为 0xF0
		prodId T	1	0x 12	必选	产品 ID 类型
		prodId V	4	0x XXXXXXXX	必选	产品 ID 值，产品 ID 的 ASCII 码值
		0xFF	1	0x FF	必选	分隔符
		ProtocolDT	1	0x 17	必选	蓝牙协议 ID 类型
		ProtocolD	1	0x 01	必	蓝牙协议 ID 长度

内容			长度（Byte）	值	必选/可选	说明
		L			选	
		ProtocolDV	1	0x XX	必选	蓝牙 ID 值 • bit0：0 表示 member 二靠不弹框，1 表示 member 二靠弹框。 • bit1：预留字段，默认填 0。 • bit2~bit7 组成的 int 值标识事件类型，字段说明如下： – 0-弹设备控制框 – 1-WiFi+BLE Combo 类型设备请求首次进行配网，下发 ssid 、密码和注册信息 – 2-直连云设备（插网线、蜂窝等）和已由 App 代理注册的 BLE 设备检测到直连云通道已建立，通过 BLE 下发注册信息 打通直连云通道 – 3-WiFi+BLE 的 Combo 设备，已经由蓝牙辅助 WiFi 配网注册成功，目前 WiFi 连不上路由器，请求 owner 更换路由器 SSID 和密码（member 不弹框） – 4-设备有异常告警事件，发送告警广播 – 5-纯蓝牙 BLE 设备请求 App 进行代理注册 通过计算得出： • 0x05：请求 App 进行

内容			长度（Byte）	值	必选/可选	说明
						设备注册（一靠广播） • 0x01/0x00：弹设备控制框（二靠广播） • 0x11/0x10：设备有告警异常事件
		SN T	1	0x 14	必选	设备 SN
		SN L	1	0x 02	必选	设备 SN 长度
		SN V	2	0x XXXX	必选	设备 SN 。设备 SN 最后 2 字节的 ASCII 码值，必须和 deviceinfo 上报的 SN 最后两位一致
		业务自定义	1	0x 91	可选	以下为业务自定义数据
		L	1	自定义 L	可选	业务自定义字段 L
		V	变长	自定义 V	可选	业务自定义字段 V

表4-12 BLE 设备靠近发现广播实例

适用场景	长度	类型	值	说明
以下每个场景都携带	0x02	0x01	0x06	BLE 可被发现
WiFi+BLE Combo 类型设备请求 App 进行设备注册（一靠广播）	0x17	0x16	0xEEFD01010D04**SS**11**PP**12**XXXXXXXX**FF1701051402**NNNN**	未注册，**SS** 表示产品子型号；**PP** 表示发射功率值；**XXXXXXXX** 表示产品 ID 的 ASCII 码值；**NNNN** 是 SN 后两位的 ASCII 码值。
二靠广播，owner 弹框，member 不	0x17	0x16	0xEEFD01010D04**SS**11**PP**12**XXXXXXXX**FF1701001402**NNNN**	已注册，**SS** 表示产品子型号；**PP** 表示发射功率值；**XXXXXXXX** 表示产品 ID 的 ASCII 码值；**NNNN** 是 SN 后

适用场景	长度	类型	值	说明
弹框				两位的 ASCII 码值。
二靠广播，owner、member 都弹框	0x17	0x16	0xEEFD01010D04SS11PP12XXXXXXXXFF1701011402NNNN	已注册，SS 表示产品子型号；PP 表示发射功率值；XXXXXXXX 表示产品 ID 的 ASCII 码值；NNNN 是 SN 后两位的 ASCII 码值。
二靠广播，离线更换配网信息	0x17	0x16	0xEEFD01010D04SS11PP12XXXXXXXXFF17010C1402NNNN	已注册断网，用 PP 功率值触发离线弹窗；NNNN 是 SN 后两位，仅支持 owner 弹框不支持 member 弹框。

响应包（Scan Response）结构如下图所示：

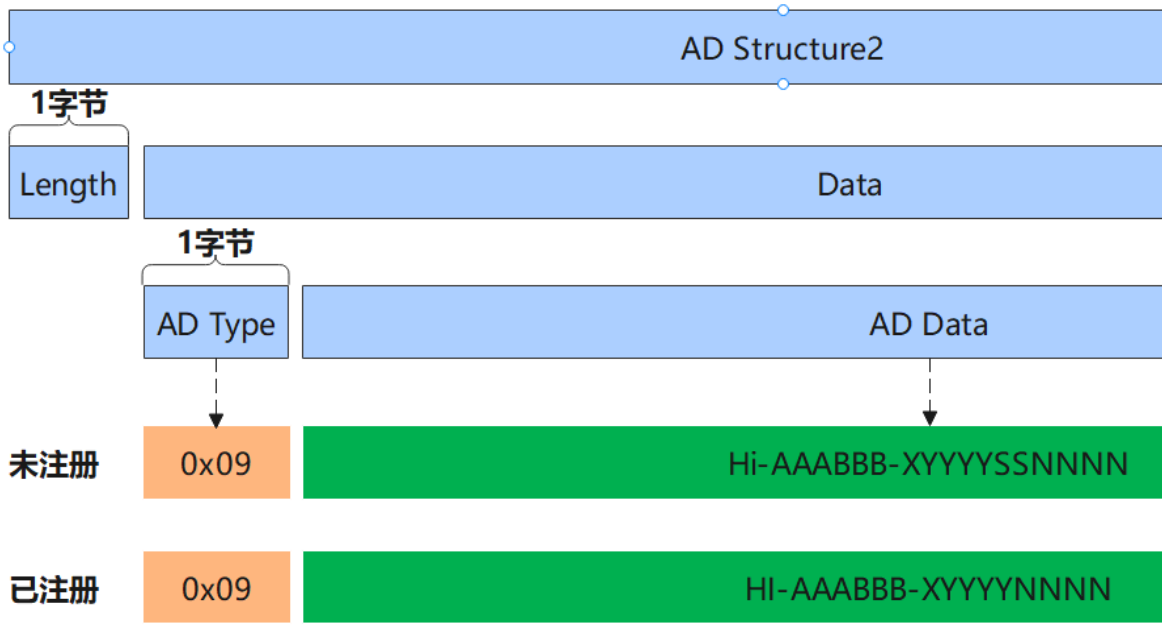


表4-13 响应包（Scan Response）广播实例

适用场景	长度	类型	值	说明
靠近发现	可变	0x09	未注册： “Hi-**AAABBB-XYYYYSSNN**	未注册： **Hi-**：未注册广播名称固定前缀，3 字节，内容为字符串‘Hi-’对应的 ASCII 码十六进制，必传；

适用场景	长度	类型	值	说明
广播、常态广播和碰一碰广播			**NN**” 已注册： “HI-**AAABBB-XYYYYNNNN**”	**AAABBB** ：设备名称+厂商名称，最长 10 字节，由厂商自定义，内容为对应字符串的 ASCII 码十六进制。可以包含字母、数字、下划线，不支持其他字符，必传； **-**：固定分割符，1 个字节，内容为‘-’对应的 ASCII 码 0x2d，必传； **X**：1 字节版本号，非 0，标识协议的版本号，当前传’1’对应的 ASCII 码十六进制 0x31,必传； **YYYY**：设备类型（ProductId），4 字节，内容为对应字符串 ASCII 码的十六进制，必传； **SS** ：设备子型号(sub ProductID)，默认值为“00”，产品配置多外观时，内容为多外观对应的编号，2 字节，内容为对应字符串 ASCII 码的十六进制，必传； **NNNN**：设备 sn 序列号后四位，4 字节，内容为对应字符串 ASCII 码的十六进制，必选； 已注册： **HI-**：固定前缀，3 字节，内容为字符串‘HI-’对应的 ASCII 码十六进制，必传； **AAABBB** ：设备名称+厂商名称，最长 10 字节，由厂商自定义，内容为对应字符串的 ASCII 码十六进制。可以包含字母、数字、下划线，不支持其他字符，必传； **-**：固定分割符，1 个字节，内容为‘-’对应的 ASCII 码 0x2d，必传； **X**：1 字节版本号，非 0，标识协议的版本号，当前传‘1’对应的 ASCII 码十六进制 0x31，必传； **YYYY**：设备类型（ProductId），4 字节，内容为对应字符串 ASCII 码的十六进制，必传； **NNNN**：设备 sn 序列号后四位，4 字节，内容为对应字符串 ASCII 码的十六进制，必选；

例如，响应包的报文为：48492d48554157454941492d313130314332383031

对应的格式如下：**48492d**（HI-） **4855415745494149**（HUAWEIAI）**2d**（**-**）**31**（X）**31303143**（YYYY 为 101C）**32383031**（NNNN 为 2801）

配置蓝牙靠近发现的设备发现方式信息

1. 若产品定义时选择的极简交互方式为蓝牙靠近发现，产品伙伴需要遵循表 2 蓝牙靠近发现广播包规范，对外广播未注册常态蓝牙报文。

表4-14 常态广播报文格式

长度	类型	值	说明
0x02	0x01	0x06	BLE 可被发现

示例代码如下：

```
// BLE 设备可被发现
unsigned char myadvData 0010[] = {
    0x2, 0x1, 0x6
};
```

未注册常态广播的蓝牙广播响应数据(myRspData)具体字段参考表 4-13，广播内容(myAdvData)厂家自定义即可。

示例代码如下：

```
//AAA，由产品品牌名与设备名称组成，伙伴自定义，1~10 位。
unsigned char myadvData 0010[] = {
    0x2, 0x1, 0x6
};
unsigned char myrspData 0001[] = {
    0x13, 0x9,  'H', 'i', '-', 'A', 'A', 'A', '-', 0x31,
    demoDevInfo->productId[0], demoDevInfo->productId[1], demoDevInfo-
>productId[2], demoDevInfo->productId[3],
    demoDevInfo->subProductId[0], demoDevInfo->subProductId[1],
    demoDevInfo->sn[8], demoDevInfo->sn[9], demoDevInfo->sn[10],demoDevInfo-
>sn[11]
};
```

2. 使用 HiLink SDK，在需要靠近发现拉起 FA 时，对外广播一靠蓝牙报文和二靠蓝牙报文。HiLink SDK 已实现报文发送的功能，发送靠近发现广播前需要调用 BLE_SetAdvType 接口，设置发送广播类型，HiLink SDK 根据当前设备注册状态对外广播一靠/二靠蓝牙报文。此外，为避免骚扰用户，建议设定发送靠近发现广播的广播时间，超时后自动切换到常态广播。

   靠近发现广播发送完整示例代码如下：

```
typedef struct
{
    void *taskName;
    int level;
    unsigned long stackSize;
} TaskParam;


#define BLE ADV CTRL TASK SLEEP 100
#define TASK STACKLEN LOW 1024
#define TASK_PRIORITY_LOW 4
```



```
#define BLE_ADV_FOREVER_START_FLAG 0XFFFFFFFF
#define BLE_ADV_60 60

//记录是否蓝牙已经 init 过
static int is_ble_init_done = 0;

typedef struct
{
    void *advTaskHandle;
    unsigned long startTime;
    unsigned long advTime;
} AdvTimeCtrl;
static AdvTimeCtrl g_advCtrl = {0};

static BLE_AdvPara g_advPara = {
    .advType = 0x00,
    .minInterval = 0x20,
    .maxInterval = 0x40,
    .channelMap = 0x07,
};

static BLE_CfgNetCb g_BleCfgNetCb = {
    NULL,
    NULL,
    NULL,
    NULL,
    NULL};

#define MS_PER_SECOND 1000
#define LOSCFG_BASE_CORE_TICK_PER_SECOND 1000
static void DEMO_BT_GetTime(unsigned long *ms)
{
    unsigned long long tickCount;
    if (ms == NULL)
    {
        return;
    }
    tickCount = (unsigned long long)osKernelGetTickCount();
    if (osKernelGetTickFreq() == 0)
    {
        *ms = 0;
        return;
    }
    *ms = (unsigned long)(tickCount * MS_PER_SECOND / osKernelGetTickFreq());
}
static void AdvCtrlTask(void *para)
{
    unsigned long currentTime = 0;
    AdvTimeCtrl *advTimeCtrl = (AdvTimeCtrl *)para;
    (void)HILINK_BT_GetTime(&currentTime);
    while (currentTime - advTimeCtrl->startTime <= advTimeCtrl->advTime)
    {
        (void)DEMO_BT_GetTime(&currentTime);
        osDelay(BLE_ADV_CTRL_TASK_SLEEP);
    }
```



```
    (void)HILINK_BT_StopAdvertise();
    advTimeCtrl->advTaskHandle = NULL;

    //靠近发现广播停止之后需要发送常态广播
    ble_adv_normal();
}

#define BLE_ADV_TIME_CTRL_TASK "adv_ctrl"
static int CreateAdvCtrlTask(AdvTimeCtrl *advCtrl)
{
    TaskParam advTaskParam = {BLE_ADV_TIME_CTRL_TASK, TASK_PRIORITY_LOW,
TASK_STACKLEN_LOW};
    int ret = osThreadNew(&advCtrl->advTaskHandle, &advTaskParam, AdvCtrlTask,
advCtrl);
    if (ret != 0)
    {
        printf("create adv ctrl task fail");
        return -1;
    }
    printf("advCtrl->advTaskHandle is [%d]\n", advCtrl->advTaskHandle);
    return 0;
}

/*
 *  蓝牙靠近发现常态广播
 */
void ble adv normal()
{
    int reg = HILINK IsRegister();
    unsigned char adv data[] = {
        0x02, 0x01, 0x06
    };
    printf("function:[%s],adv data len is [%d]\n",   FUNCTION  ,
sizeof(adv data));
    unsigned char adv rsp data[30] = {0};
    int adv rsp len = 0;
    // 只有未注册，需要发送未注册常态广播。否则，直接返回。
    if (0 != reg)
    {
        return;
    }
    printf("function:[%s],device is not register yet\n",   FUNCTION  );
    /* 未注册常态广播
     * 设备未注册时的常态广播需要能够使用智慧生活 App 扫描添加设备。蓝牙响应数据
(adv_rsp_data)需要符合蓝牙命名格式(参考准备工作 步骤 4)，
     * 广播内容(adv_data)厂家自定义即可。参考代码如下：//AAAABBBB，由产品品牌名与设备名称
组成，伙伴自定义，1~14 位。*/
    unsigned char  adv rsp data[] = {
        0x18, 0x09,
        'H', 'i', '-',
        MANUAFACTURER NAME[0], MANUAFACTURER NAME[1], MANUAFACTURER NAME[2],
MANUAFACTURER NAME[3],
        DEVICE TYPE NAME[0], DEVICE TYPE NAME[1], DEVICE TYPE NAME[2],
DEVICE TYPE NAME[3], '-', 0x31,
        PRODUCT_ID[0], PRODUCT_ID[1], PRODUCT_ID[2], PRODUCT_ID[3], 0x30, 0x30,
```



```
    PRODUCT_SN[8], PRODUCT_SN[9], PRODUCT_SN[10], PRODUCT_SN[11]};

 adv_rsp_len = sizeof(_adv_rsp_data);
 printf("adv_rsp_len is [%d]\n", adv_rsp_len);
 for (int i = 0; i < adv_rsp_len; i++)
 {
    adv_rsp_data[i] = _adv_rsp_data[i];
 }
 adv_rsp_data[adv_rsp_len] = '\0';

 BLE_AdvInfo advInfo;
 advInfo.advPara = &g_advPara;

 BLE_AdvData g_advData = {
    .advData = adv_data,
    .advDataLen = sizeof(adv_data),
    .rspData = adv_rsp_data,
    .rspDataLen = adv_rsp_len,
 };
 advInfo.advData = &g_advData;

 BLE_InitPara initPara;

 BLE ConfPara isBlePair;
 isBlePair.isBlePair = 0;
 initPara.confPara = &isBlePair;

 initPara.gattList = NULL;
 initPara.advInfo = &advInfo;

 //如果已经 init 过了，则只需要 update 就行
 if (is ble init done)
 {
    int ret = BLE CfgNetAdvUpdate(&advInfo);
    if (ret != 0)
    {
       printf("function=[%s] error,line=[%d],update advertise error\n",
FUNCTION  ,   LINE  );
       return ret;
    }
 }
 else
 {
    int ret = BLE CfgNetInit(&initPara, &g BleCfgNetCb);
    if (ret != 0)
    {
       printf("function=[%s] error,line=[%d],ble init advertise error\n",
FUNCTION  ,   LINE  );
       return ret;
    }
    is ble init done = 1;
 }

 (void)BLE CfgNetAdvCtrl(BLE ADV FOREVER START FLAG);
```



```
    //如果此时有发送靠近、碰一碰广播，需要停止 task
    AdvTimeCtrl *advCtrl = &g_advCtrl;
    if (g_advCtrl.advTaskHandle != NULL)
    {
        osThreadTerminate(g_advCtrl.advTaskHandle);
        g_advCtrl.advTaskHandle = NULL;
    }
}

/*
 *   蓝牙靠近发现功能一靠二靠广播示例函数
 *   使用场景：用户主动触发时，调用改接口，由常态广播切换到一靠二靠广播
 */
void ble_adv_nearby()
{
    BLE_ConfPara isBlePair;
    isBlePair.isBlePair = 0;
    BLE_InitPara initPara;
    BLE_AdvInfo advInfo;

    memset(&isBlePair, 0x00, sizeof(BLE_ConfPara));
    memset(&initPara, 0x00, sizeof(BLE_InitPara));
    memset(&advInfo, 0x00, sizeof(BLE_AdvInfo));

    initPara.confPara = &isBlePair;
    initPara.advInfo = NULL;
    advInfo.advPara = NULL;
    advInfo.advData = NULL;

    //如果已经 init 过了，则只需要 update 就行
    if (is ble init done)
    {
        BLE SetAdvType(BLE ADV DEFAULT);
        int ret = BLE CfgNetAdvUpdate(&advInfo);
        if (ret != 0)
        {
            printf("function=[%s] error,line=[%d],update advertise error\n",
  FUNCTION  ,   LINE  );
            return ret;
        }
    }
    else
    {
        /* BLE 配网资源申请：BLE 协议栈启动、配网回调函数挂载*/
        BLE SetAdvType(BLE ADV DEFAULT);

        int ret = BLE CfgNetInit(&initPara, &g BleCfgNetCb);
        if (ret != 0)
        {
            printf("function=[%s] error,line=[%d],ble init advertise error\n",
  FUNCTION  ,   LINE  );
            return ret;
        }
        is ble init done = 1;
    }
```



```
    (void)BLE_CfgNetAdvCtrl(BLE_ADV_60);

    AdvTimeCtrl *advCtrl = &g_advCtrl;
    if (g_advCtrl.advTaskHandle != NULL)
    {
        osThreadTerminate(g_advCtrl.advTaskHandle);
        g_advCtrl.advTaskHandle = NULL;
    }
    /* 秒折算成1000毫秒 */
    advCtrl->advTime = (unsigned long)(BLE_ADV_60 * 1000);
    (void)DEMO_BT_GetTime(&advCtrl->startTime);
    if (CreateAdvCtrlTask(advCtrl) != 0)
    {
        printf("CreateAdvCtrlTask fail");
    }
}

void main(void)
{
    HILINK_SetNetConfigMode(HILINK_NETCONFIG_OTHER);
    ble_adv_normal();

    HILINK SdkAttr sdkAttr = {0};
    HILINK SdkAttr *attr = HILINK GetSdkAttr();
    memcpy(&sdkAttr, attr, sizeof(sdkAttr));
    sdkAttr.deviceMainTaskStackSize = 8 * 1024;
    sdkAttr.monitorTaskStackSize = 2 * 1024;
    sdkAttr.otaCheckTaskStackSize = 6 * 1024;
    sdkAttr.otaUpdateTaskStackSize = 6 * 1024;
    HILINK SetSdkAttr(sdkAttr);

    HILINK Main();
}

// 当用户触发后，需要切换广播内容，发一靠或者二靠广播。只需调用 ble_adv_nearby() 即可。
```

4.3.4 实现设备控制功能

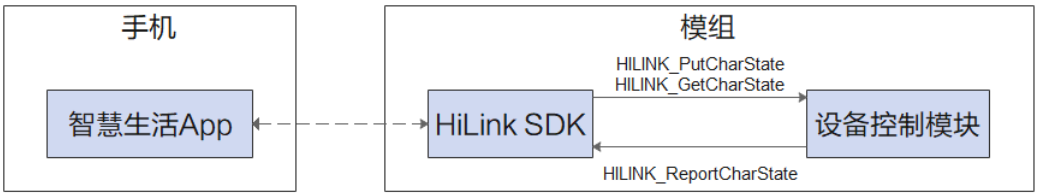
本节介绍如何开发设备控制功能，需要开发者实现 hilink_device.c 中的 HILINK_PutCharState 和 HILINK_GetCharState 两个函数，并结合 SDK 提供的接口 HILINK_ReportCharState 开发设备控制和状态上报功能。

表4-15 设备控制相关函数

函数名称	是否需要开发者实现	说明
HILINK_PutCharState	是	云端下发控制指令后，会通过 SDK 调用到这个函数。伙伴需要再对服务进行识别、分发和处理。
HILINK_GetCh	是	云端通过 HiLink SDK 获取设备状态信息，或者

函数名称	是否需要开发者实现	说明
arState		HiLink SDK 主动调用接口获取设备状态信息。
HILINK_ReportCharState	否	HiLink SDK 报文上报接口，用于上报设备状态信息。根据设备功能，按需调用。

图4-3 手机和模组交互关系图



步骤 1 根据 profile 文件中定义，开发设备控制和状态查询功能。

profile 定义了设备支持的服务，以及服务支持的操作。如：控制、查询、上报等。设备开发必须按照 profile 的定义分别实现对应的功能。

以开关的控制（handle_put_switch）和查询（handle_get_switch）功能为例，实现如下：

```
// 设备状态定义
typedef struct{
    unsigned int switch on;
    unsigned int faltDetection code;
    unsigned int faltDetection status;
} t device info;

// 分配一个对象记录设备状态
static t device info g device info = {0};

// 处理从 HILINK PutCharState 传递过来的信息
int handle put switch(const char* svc id, const char* payload, unsigned int len)
{
    cJSON* pJson = cJSON Parse(payload);
    if (pJson == NULL){
        printf("JSON parse failed in PUT cmd: ID-%s \r\n", svc id);
        return INVALID PACKET;
    }
    cJSON* item = cJSON GetObjectItem(pJson, "on");
    if (item != NULL) {
        g device info.switch on = item->valueint;
    }
    if (pJson != NULL) {
        cJSON Delete(pJson);
    }
    printf("handle func:%s, sid:%s \r\n",   FUNCTION  , svc id);
    return M2M_NO_ERROR;
```



```
}

// 处理从 HILINK_GetCharState 传递过来的信息
int handle_get_switch(const char* svc_id, const char* in, unsigned int in_len,
char** out, unsigned int* out_len)
{
    bool on = g_device_info.switch_on;
    *out_len = 20;
    *out = (char*)hilink_malloc(*out_len);
    if (NULL == *out){
        printf("malloc failed in GET cmd: ser %s in GET cmd", svc_id);
        return INVALID_PACKET;
    }
    *out_len = hilink_sprintf_s(*out, *out_len, "{\"on\":%d}", on);
    printf("hilink_device_ctr.c :%d %s svcId:%s, out:%s\r\n", __LINE__,__FUNCTION__ ,
svc_id, *out);
    return M2M_NO_ERROR;
}
```

步骤 2 注册服务处理信息。

设备开发时，需要根据“HILINK_PutCharState”函数和“HILINK_GetCharState”函数下发的服务 ID，分发指令信息到不同的函数处理。示例代码如下：

```
// 服务处理函数定义
typedef int (*handle_put_func)(const char* svc_id, const char* payload, unsigned
int len);
typedef int (*handle_get_func)(const char* svc_id, const char* in, unsigned int
in_len, char** out, unsigned int* out_len);
// 服务注册信息定义
typedef struct{
    // service id
    char* sid;
    // HILINK_PutCharState cmd function
    handle_put_func putFunc;
    // handle HILINK_GetCharState cmd function
    handle_get_func getFunc;
} HANDLE_SVC_INFO;

//不支持 HILINK_PutCharState 时，默认实现
int not_support_put(const char* svc_id, const char* payload, unsigned int len)
{
    printf("sid:%s NOT SUPPORT PUT function \r\n", svc id);
    return 0;
}
// 服务处理信息注册
HANDLE SVC INFO g device profile[] = {
    {"switch", handle put switch, handle get switch},
    // 故障不支持 HILINK_PutCharState，配置 not_support_put
    {"faultDetection", not support put, handle get faultDetection},
};
// 服务总数量
int g_device_profile_count = sizeof(g_device_profile) / sizeof(HANDLE_SVC_INFO);
```

步骤 3 分发服务。

增加服务分发处理函数“handle_put_cmd”、“handle_get_cmd”以及“fast_report”。

- “handle_put_cmd”和“handle_get_cmd”分别用于分发“HILINK_PutCharState”和“HILINK_GetCharState”传递的指令。
- “fast_report”用于快速上报设备状态信息。

示例代码如下：

```
// 辅助函数，用于查找服务注册信息
static HANDLE_SVC_INFO* find_handle(const char* svc_id)
{
    for(int i = 0; i < g_device_profile_count; i++) {
        HANDLE_SVC_INFO handle = g_device_profile[i];
        if(strcmp(handle.sid, svc_id) == 0) {
            return &g_device_profile[i];
        }
    }
    return NULL;
}
// 分发设备控制指令
int handle_put_cmd(const char* svc_id, const char* payload, unsigned intlen)
{
    HANDLE_SVC_INFO* handle = find_handle(svc_id);
    if(handle == NULL) {
        printf("no service to handle put cmd : %s \r\n", svc id);
        return INVALID PACKET;
    }
    handle put func function = handle->putFunc;
    if(function == NULL) {
        printf("put function is null for %s \r\n", svc id);
        return INVALID PACKET;
    }
    return function(svc id, payload, len);
}
// 分发服务查询直连
int handle get cmd(const char* svc id, const char* in, unsigned int in len, char**
out, unsigned int* out len)
{
    HANDLE SVC INFO* handle = find handle(svc id);
    if(handle == NULL) {
        printf("no service to handle get cmd : %s \r\n", svc id);
        return INVALID PACKET;
    }
    handle get func function = handle->getFunc;
    if(function == NULL) {
        printf("get function is null for %s \r\n", svc id);
        return INVALID PACKET;
    }
    return function(svc id, in, in len, out,out len);
}
// 快速上报函数，用于上报服务状态信息
int fast report(const char* svc id, int task id)
{
    const char* payload = NULL;
    int len;
```



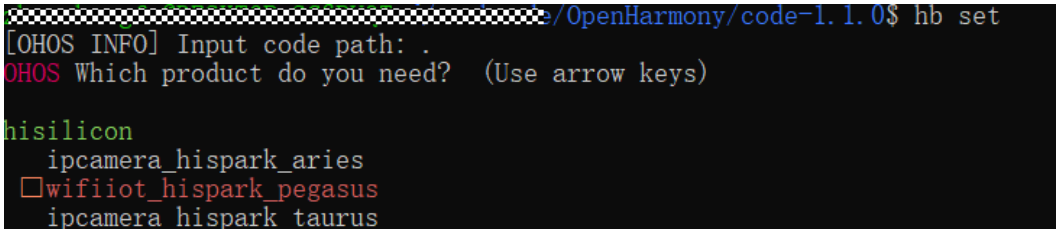
```
    int err = handle_get_cmd(svc_id, NULL, 0, &payload, &len);
    if(err != M2M_NO_ERROR) {
        printf("get msg from %s failed \r\n", svc_id);
        return err;
    }
    err = HILINK_ReportCharState(svc_id, payload, len, task_id);
    printf("report %s result is %d \r\n", svc_id, err);
    return err;
}


// ---------------------------------- //
// 以下两个函数在 hilink_device.c 中      //
// ---------------------------------- //
int HILINK_PutCharState(const char* svc_id,
        const char* payload, unsigned int len){
    int err = M2M_NO_ERROR;
    if(svc_id == NULL) {
        hilink_error("empty service ID in PUT cmd");
        return M2M_SVC_RPT_CREATE_PAYLOAD_ERR;
    }
    if (payload == NULL) {
        hilink_error("empty payload in PUT cmd");
        return M2M SVC RPT CREATE PAYLOAD ERR;
    }

    hilink debug("start handle PUT cmd: ID-%s", svc id);
    err = handle put cmd(svc id, payload, len);
    hilink debug("handle PUT cmd end: ID-%s, ret-%d", svc id, err);
    return err;
}
int HILINK GetCharState(const char* svc id, const char* in,
        unsigned int in len, char** out, unsigned int* out len){
    int err = M2M NO ERROR;
    if(svc id == NULL){
        hilink error("empty service ID in GET cmd");
        return M2M SVC RPT CREATE PAYLOAD ERR;
    }
    hilink info("start process GET cmd: ID - %s", svc id);
    err = handle get cmd(svc id, in, in len, out, out len);
    hilink debug("end process GET cmd: ID - %s, ret - %d", svc id, err);
    return err;
}
```

----结束

4.4 编译固件

步骤 1 进入工程根目录，执行“hb set”、“.”，并选择需要构建的产品。

图4-4 构建设置示例



步骤 2 在工程根目录，使用“hb build xx”命令进行版本构建。

- 在执行 XTS 测试时，需要使用 debug 版本进行构建，即执行构建命令“hb build -f”。
- 在提交认证预约时，需要使用 release 版本进行构建，即执行构建命令“hb build -f -b release"。

返回“xxxx build success”，表明构建成功。

----结束

4.5 烧录固件

步骤 1 从模组商处获取串口驱动和烧录工具。

说明

不同芯片使用的驱动和烧录工具均不同，建议联系模组商获取支撑。

步骤 2 按照模组商提供的指导文档安装驱动。

步骤 3 使用烧录工具烧写固件到模组上。

----结束

5 功能验证

5.1 预置激活码

说明

激活码是设备合法性认证的唯一标识，需要保持一机一码。认证成功后，调测激活码会自动转为商用激活码，无需重新申请。

设备中预置正确的激活码是认证成功的必要条件。

步骤 1 申请激活码。

参考设备授权申请激活码。调测阶段申请和使用调测激活码；商用阶段申请和使用商用激活码。在产品认证通过之后，调测激活码会自动更新为商用激活码，无需重新预置激活码。

步骤 2 预置激活码一般有两种方法，一种是通过 AT 指令写入，另外一种是使用版本烧录工具烧写。不同模组商提供的写入方式不同，需要和模组商确认如何进行激活码预置。

须知

如果采用烧录工具烧写激活码，需要首先把激活码转换成可以烧录的文件。同时，需要提前与模组商确认烧录的位置和长度，以免覆盖到其他信息。

激活码是设备的信任凭据，伴随整个设备的生命周期，不能被任意擦写。OTA 升级和恢复出厂设置时都不能覆盖激活码区域。

----结束

5.2 测试配网和设备控制

5.2.1 配置调测环境

步骤 1 配置调测设备。Device Partner 平台，“产品开发 > 集成开发 > 管理调测设备”，添加调测设备的 SN 号。

说明

- 添加调测设备的 SN 号，区分大小写。
- 添加了 SN 号所对应的调测设备，才能使用调测激活码对该设备进行配网和功能调测。

步骤 2 配置测试帐号。

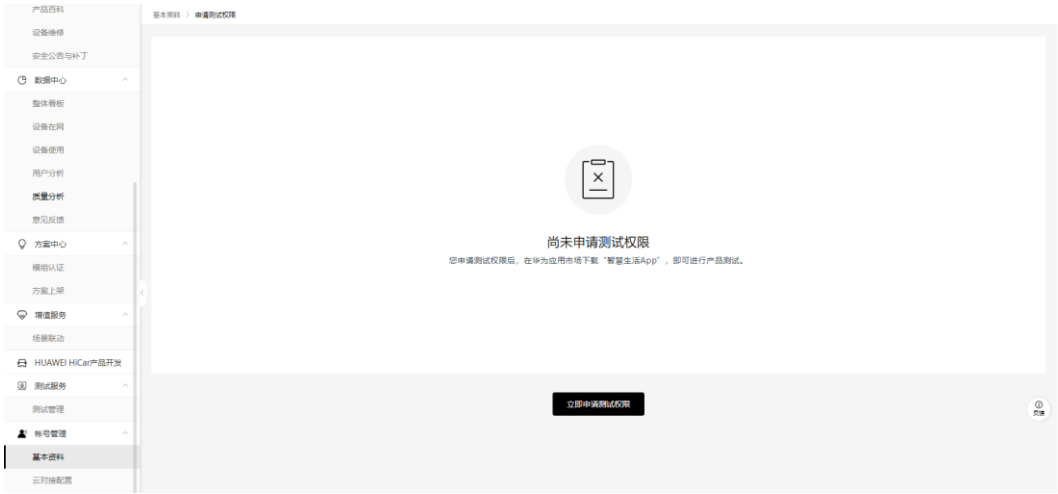
- 方式一：申请测试权限。

须知

申请权限操作仅对当前帐号生效，不会对团队的其他帐号生效。申请权限的华为帐号，必须与手机调测使用的华为帐号保持一致。

a. 登录 Device Partner 平台，进入管理中心。
b. 选择“帐号管理 > 基本资料”，单击右上角的“申请测试权限”。
c. 单击“立刻申请测试权限”申请测试权限。

图5-1 申请测试权限



- 方式二：下载智慧生活 App Debug 版本。
  a. 登录华为智能硬件合作伙伴平台，单击右上角的“管理中心”。
  b. 进入“产品开发 > 集成开发”页面，下载智慧生活 App Debug 版本。通过手机浏览器扫描二维码，或者在手机浏览器中输入链接地址下载即可。

图5-2 智慧生活 App 下载方式

c. 配置智慧生活 App 测试环境。

打开智慧生活 App 后，进入“我的 > 设置 > 关于 > 环境设置”，选择“认证沙箱”环境。

----结束

5.2.2 测试设备配网与设备控制功能

步骤 1 打开智慧生活 App，点击右上角“+”，选择“添加设备”，智慧生活 App 会扫描附近所有处于待配网状态的设备。

图5-3 智慧生活 App 首界面

步骤 2 选择需要配网的设备，点击“连接”开始配网。

步骤 3 配网成功后，设置设备位置信息（如卧室、阳台等）。

步骤 4 打开设备卡片，进入设备控制界面。

设备控制界面为交互设计环节部署的 H5 界面，展示了设备状态和功能控制服务等。

说明

调测阶段，因为产品还没有提交认证，所以会有警告窗口，点击“继续”即可。

步骤 5 点击设备控制按钮，如开关等，设备侧会收到相关指令。

----结束

5.2.3 添加设备失败问题分析

本节对添加设备过程进行拆解和说明，帮助伙伴了解添加设备的主要过程，以达成快速对问题定位定界的目的。

从执行顺序上看，添加设备过程依次经历以下三个过程阶段。

表5-1 添加设备过程介绍

阶段	作用	开始标志	结束标志	成功日志	失败日志
配网阶段	设备连接到 Wi-Fi 热点，具备联网能力	wait STA join AP	connect success	-	-
认证阶段	联网认证设备合法性，校验激活码和设备信息	INFO [KitFramework]: Device is in initialization mode	[KitFramework]: Response kit authentication status by executing callback	INFO [KitFramework]: Active symbol succeed	搜索关键字“ERROR [KitFramework]:”
注册阶段	注册设备信息，建立设备和帐号的关联关系	set dev status [2]	set dev status [4]	set dev status [4]	未打印“set dev status [4]”，或者打印“set dev status [6]”。

说明

认证阶段日志信息较多，搜索关键字“INFO [KitFramework]: ”可以查看认证的过程。

- 如果设备认证成功，会打印 “INFO [KitFramework]: Active symbol succeed”日志。
- 如果设备认证失败，将不会打印“INFO [KitFramework]: Active symbol succeed”日志，需要搜索 “ERROR [KitFramework]:”查看错误信息。

6 附录

图6-1 产品创建时需要填写的产品基本信息

表6-1 产品创建时需要填写的产品基本信息说明

参数	API 接口	说明和要求
产品名称（传播名）	GetMarketName()	最大 32 字符。 用户可见，Kit Framework 认证关键项。
品牌	-	-
品牌英文名	GetBrand()	最大 32 字符。 Kit Framework 认证关键项，和单品激活码强关联。
产品系列	GetProductSeries()	最大 32 字符。 用以对不同类型产品进行分类管理。Kit Framework 认证关键项。
产品型号	GetProductModel()	最大 32 字符。 设备关键信息之一，用以标识设备的类型，用户可见，通常打印在设备铭牌上，用以区分不同产品。该值也是进行 HarmonyOS Connect 认证所需的关键数据，需要采用英文描述。

参数	API 接口	说明和要求
		Kit Framework 认证关键项，和单品激活码强关联。

图6-2 产品预约认证时填写的信息

表6-2 产品预约认证时填写的信息说明

参数	对应 API	说明
软件版本号	GetDisplayVersion()	最多 64 字符。 厂商定义填写内容。注意是整个系统的软件版本号而非 HarmonyOS 版本号。
硬件设备版本号	GetHardwareModel()	最大 32 字符。 厂商定义填写内容。
系统 API Level	GetSdkApiVersion()	设备当前版本 API 级别，一般是整数，仅有生态的系统使用。 版本预置值，不需要修改。
版本 ID	GetVersionId()	最大 127 字符。 由多个字段拼接而成：$(设备类型) + '/' + $(企业英文名简称) + '/' + $(品牌英文名称) + '/' + $(产品系列英文名称) + '/' + $(操作系统及版本号) + '/' + $(型号) + '/' + $(内部软件子型号) + '/' + $(系统软件 API level) + '/' + $(差异版本号) + '/' + $(构建类型) 在所有厂家的所有设备范围中，可以唯一标识版本。
安全补丁级别	GetSecurityPatchTag()	最多 64 字符。 标识当前 OS 的安全补丁级别。按实际情况填写，例如 2020-12-01。
版本 Hash	GetBuildRootHash()	默认为空即可。对应代码中填入空串（""）即可。

说明

Device Partner 平台认证预约界面填写的这些字段，需要与设备 OpenHarmony 固件包保持一致（可通过接口查询）。如果不一致，将无法通过认证。

7 参考

华为智能硬件合作伙伴 > 常见问题

