设备集成开发指导

文档版本 01

发布日期 2022-04-19

华为终端有限公司

商标声明


、 

、 

、华为，以上为华为公司的商标（非详尽清单），未经华为公司书面事先明示许可，任何第三方不得以任何形式使用。

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.....1

2 准备工作.....3

3 固件开发.....5

3.1 简介.....5

3.2 配置产品信息.....11

3.2.1 配置 parameter_common.c 中的参数信息.....11

3.2.2 配置 hal_sys_param.c 中的参数信息.....12

3.2.3 配置 hal_token.c 中的参数信息.....13

3.2.3.1 配置产品 ID 信息.....13

3.2.3.2 配置 AcKey 信息.....14

3.3 开发设备功能.....14

3.3.1 配置 hilink_device.h 中产品信息.....14

3.3.2 配置 hilink_device.c 中的服务信息.....16

3.3.3 配置 hilink_demo.c 中的发现设备方式信息.....17

3.3.4 开发设备功能.....27

3.4 编译固件.....31

3.5 烧录固件.....32

4 功能验证.....33

4.1 预置激活码.....33

4.2 测试配网和设备控制.....34

4.2.1 配置调测环境.....34

4.2.2 测试设备配网与设备控制功能.....35

4.2.3 添加设备失败问题分析.....36

5 附录.....38

6 参考.....41

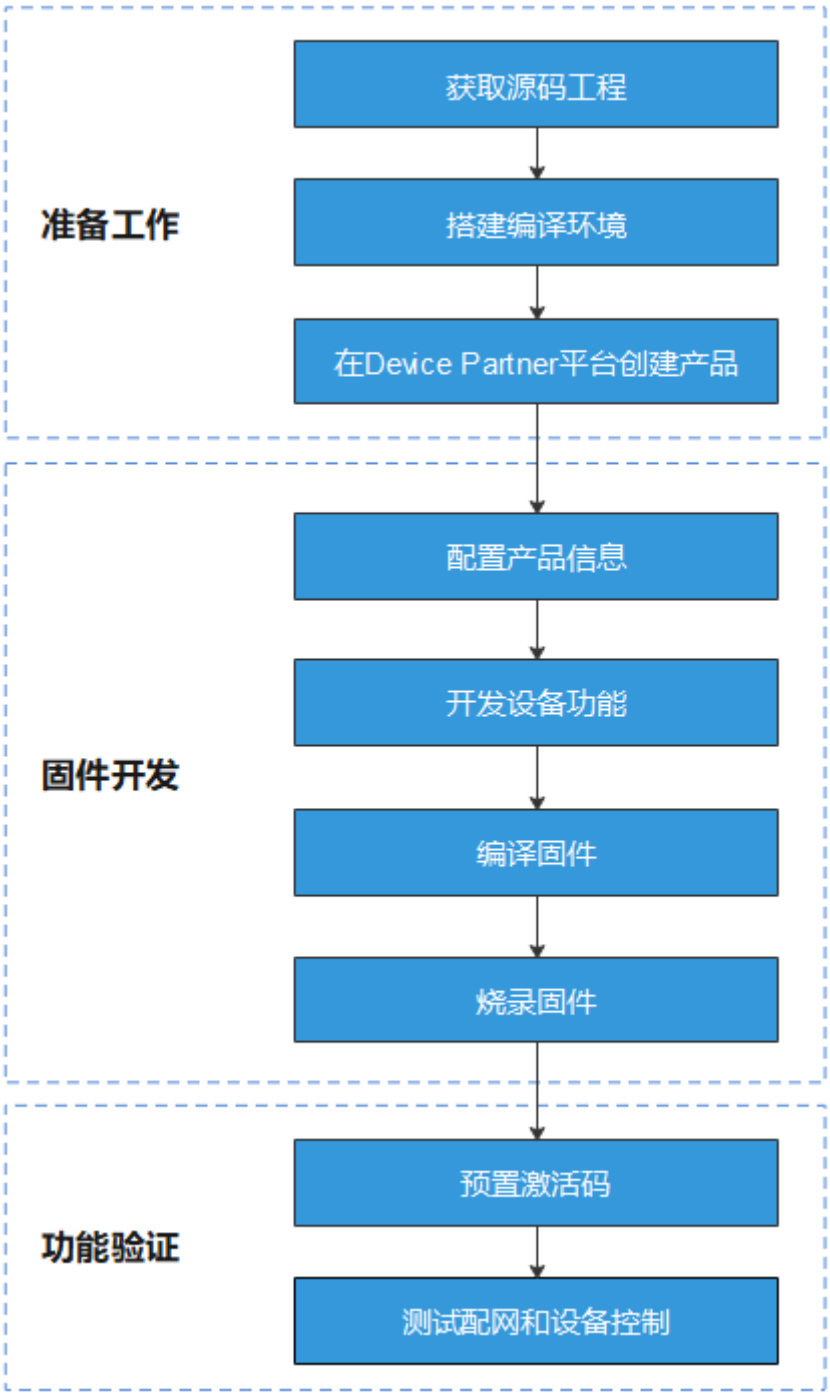
1 概述

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 准备工作

步骤 1 联系模组商获取以下工具和信息。

表2-1 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。

- Hi3861 芯片的 Wi-Fi 模组：选择“极速秒控配网”。
- 其他 Wi-Fi 模组：选择“极速常规配网”。
- Combo 模组：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 2-1 所示。

图2-1 导出产品信息

----结束

3 固件开发

3.1 简介
3.2 配置产品信息
3.3 开发设备功能
3.4 编译固件
3.5 烧录固件

3.1 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 3-1。

表3-1 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_de	产品 ID、设备类型	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	vice.h	ID、厂商 ID、设备类型名、厂商名称	息获取与设备注册。
	hilink_device_sdk.c	服务信息	用于设备上报当前支持的服务列表。

服务包说明

当前的“SDK 开发包”下载下来会包含所有的服务包，开发者需要根据产品配置的开发方案和选择的模组类型，集成不同的开发包，具体可以参考表 2

表3-2 集成开发包

标准方案类型	模组类型	开发包
直连方案	Wi-Fi 模组	ailifeclientservice-wifi-版本号
	Combo 模组	ailifeclientservice-wifi-版本号 ailifeclientservice-ble-版本号
双联方案	Wi-Fi 模组	ailifeclientservice-wifi-cloud-版本号
	Combo 模组	ailifeclientservice-ble-cloud-版本号

不同的服务包包含的静态库不同，以下表 3 为服务包静态库说明

表3-3 服务包说明

文件夹名称	包含的静态库	静态库说明
smartdeviceframework-版本号	libkitframework.a	KitFramework 认证业务全部功能静态库
ailifeclientservice-wifi-版本号	libhilinkdevicesdk.a libhilinkota.a	HiLink SDK 通用功能静态库 HiLink SDK 华为云 OTA 功能静态库
ailifeclientservice-ble-版本号	libhilinkbtsdk.a	HiLink SDK BLE 辅助配网特性静态库

文件夹名称	包含的静态库	静态库说明
ailifeclientservice-wifi-cloud-版本号	libhilinkdevicesdk.a	HiLink SDK 通用功能静态库
ailifeclientservice-ble-cloud-版本号	libhilinkbtsdk.a	HiLink SDK BLE 辅助配网特性静态库

附加参考：API 接口说明

表 3-4 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3-4 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。 具体见设备类型定义章节。物联网设备取固定值 linkiot。
GetManufacture()	公司英文名简称	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。 **需要与用户可见软件版本号（g_roBuildVerShow）、固件版本号（FIRMWARE_VER）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。 **在提交认证预约前，建议以 release 版本发布，即执行构建命令"hb build -f -b release"。**
GetBuildUser()	构建 user	-	-	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
GetBuildHost()	构建 host	-	-	最多 32 字符。
GetBuildTime()	构建时间	-	-	最多 32 字符。 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。

3.2 配置产品信息

3.2.1 配置 parameter_common.c 中的参数信息

参见表 3-5，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"}; // 用户可见软件版本号。需要与固件版本号（FIRMWARE_VER）、差异版本号（GetIncrementalVersion()）保持一致
```

表3-5 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	用户可见软件版本号。需要与固件版本号（FIRMWARE_VER）、差异版本号（GetIncrementalVersion()）保持一致。 **其中用户可见软件版本号，对应 Device Partner 平台认证预约时填写的“软件版本号”**。参见图 5-2。	-

3.2.2 配置 hal_sys_param.c 中的参数信息

参考表 3-6 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、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3-6 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	公司英文简称，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“产品信息”页签下，可以查看“公司英文名简称”。	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”。	HUAWEI
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	Mate 30

配置项	说明	示例
	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）”。	
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。	-

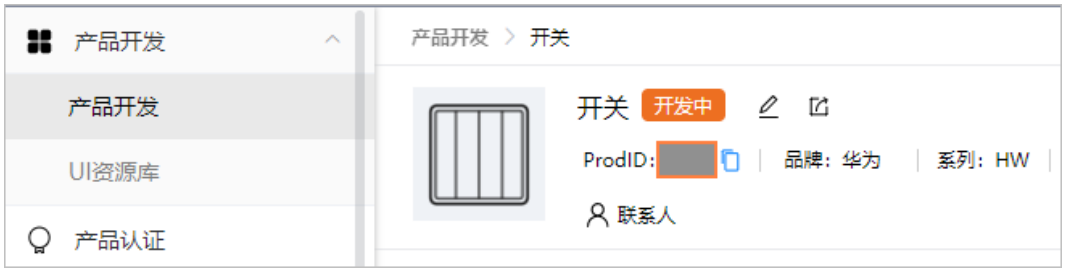
3.2.3 配置 hal_token.c 中的参数信息

3.2.3.1 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3-1 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

3.2.3.2 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

3.3 开发设备功能

3.3.1 配置 hilink_device.h 中产品信息

配置产品基本信息，用于设备配网和设备注册。

```
#define FIRMWARE_VER "1.0.1" // 固件版本号，对应智慧生活 App 中设备版本信息显示的固件版本。需要与用户可见软件版本号（g_roBuildVerShow）、差异版本号（GetIncrementalVersion()）保持一致
#define SOFTWARE_VER "12.0.0.303" // 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）
#define 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
```



```
#define PRODUCT_ID "9A8C" // 产品 ID，必须和产品真实信息一致
#define DEVICE_TYPE "046" // 产品类型 ID，必须和产品真实信息一致
#define MANUAFACTURER "xxx" // 厂商 ID，必须和产品真实信息一致
#define DEVICE_MODEL "HiTest0728" // 产品型号，必须和产品真实信息一致
#define DEVICE_TYPE_NAME "HiLight" // 设备类型名，和"集成开发环节"ssid 信息中保持一致
#define MANUAFACTURER_NAME "XXXXX"  // 厂商名称，和"集成开发环节"ssid 信息中保持一致
```

图3-2 SSID 配置

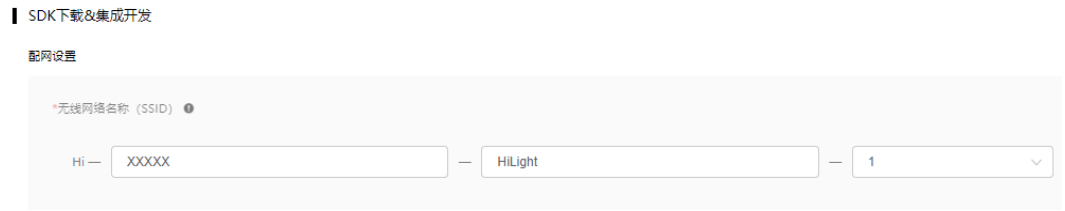


表3-7 配置项说明

配置项	说明	示例
FIRMWARE_VER	固件版本号，对应智慧生活 App 中设备版本信息显示的固件版本。需要与用户可见软件版本号（g_roBuildVerShow）、差异版本号（GetIncrementalVersion()）保持一致。 **其中用户可见软件版本号，对应 Device Partner 平台认证预约时填写的“软件版本号”**。参见图 5-2。	-
SOFTWARE_VER	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）。	-
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	产品类型 ID，参考准备工作步骤 5。	-
MANUAFACTURER	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。	-
DEVICE_M	产品型号。获取方式如下：	-

配置项	说明	示例
ODEL	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 3-2，需要和网站 SSID 保持一致。	Light
MANUAFACTURER_NAME	产商名称。参考图 3-2，需要和网站 SSID 保持一致	HUAWEI

3.3.2 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“交互设计 > 产品开发 > 物模型定义”页面，单击右侧的“下载 Profile”。

步骤 2 在 hilink_device_sdk.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 3-8 所示，则对应的工程代码示例如下：

表3-8 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

```
typedef struct {
    constchar* st; // 服务类型
    constchar* svc_id; // 服务 ID
} svc_info_t;
```



```
int gSvcNum = 2;
svc_info_t gSvcInfo[] =
{
    { "state", "lockState"},
    { "mode", "lockMode"}
};
```

说明

gSvcInfo 变量的服务类型和服务 ID 需要和结构体定义的顺序保持一致。

----结束

3.3.3 配置 hilink_demo.c 中的发现设备方式信息

发现设备的方式对应的 SDK 要求和模组要求如下：

表3-9

发现设备方式	HiLinkSDK 最低版本	模组要求	已适配芯片
NFC 碰一碰	12.0.0.303	Combo 或 Wi-Fi 模组	ALL
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

NFC 碰一碰：

需要完成 NFC 标签码流，参见 NFC 标签认证。

若产品定义时选择的配网方式为蓝牙辅助配网，需要遵循 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3-10 未注册常态广播报文格式

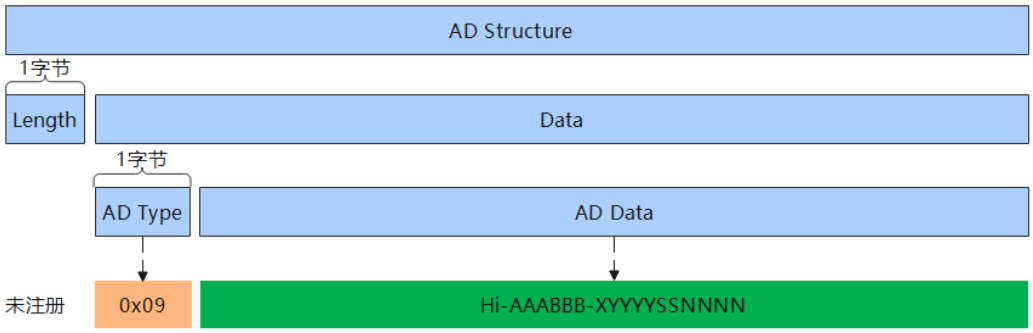
长度	类型	值	说明
0x02	0x01	0x06	BLE 可被发现

示例代码如下：

```
// BLE 设备可被发现
unsigned char myadvData_0010[] = {
```

```
    0x2, 0x1, 0x6
};
```

响应报文结构如下：



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 3。

表3-11 响应包（Scan Response）广播报文格式

长度	类型	值	说明
可变	0x09	Hi-**AAABBB-**X**YYYY**SSNNNN	**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； **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； -：固定分割符，1 字节，内容为‘-’对应的 ASCII 码 0x2d； **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； **YYYY**：设备类型（ProductId），4 字节，内容为对应字符串 ASCII 码的十六进制； **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； **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；

例如，响应包的报文为：**170948692d48554157454941492d013130314330313238303l**

对应的格式如下：**17（长度）09（类型）48692d**（Hi-） **4855415745494149**（HUAWEIAI）**2d**（-）**31**（X）**31303143**（YYYY 为 101C）**3031**（SS 为 01）**32383031**（NNNN 为 2801）

示例代码如下：

```
unsigned char myrspData_0001[] = {
    0x17, 0x9,  'H', 'i', '-', 'H', 'U', 'A', 'W', 'E', 'I', 'A', '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]
};
```

当设备注册成功后，即可停止发送广播报文。

蓝牙碰一碰：

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 HiLinkSDK 接口 BLE_SetAdvType，对外广播蓝牙报文。

示例代码如下：

```
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(&initPara, &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;
    }
}
```

蓝牙靠近发现：

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：

1. 遵循蓝牙靠近发现广播包规范，对外广播未注册常态蓝牙报文和已注册常态蓝牙报文。

当设备未注册时，为保证智慧生活 App 可以扫描广播添加设备，需要发送未注册常态广播。

当设备已注册时，为保证打开设备卡片时连接设备，或者设备连接过程中蓝牙连接断开时重新连接，需要发送已注册常态广播。

未注册常态广播和已注册常态广播的蓝牙广播响应数据(myRspData)具体字段参考表 3，广播内容(myAdvData)厂家自定义即可。

表3-12 常态广播报文格式

长度	类型	值	说明

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
};
```

响应包（Scan Response）结构如下图所示：

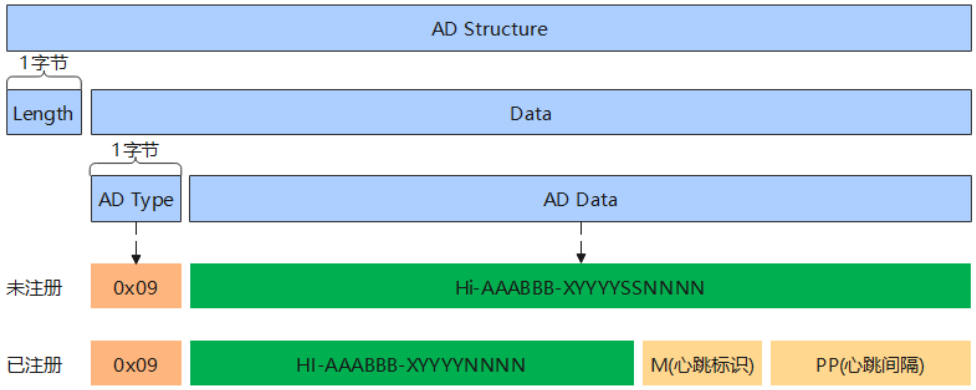


表3-13 响应包（Scan Response）广播实例

长度	类型	值	说明
可变	0x09	未注册： “Hi-**AAABBB**-XYYYYSSNNNN” 已注册： “HI-**AAABBB**-XYYYYNNNNMPP”	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； **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 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；

长度	类型	值	说明
			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，必传； 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，必传； **M**: 1 个字节：0x00—0xFF，可选，如果发送该参数，则 PP 为必传；**如果设备侧不携带 MPP 字段，则网关默认不回连该 BLE 设备。** bit0： 0 为心跳广播报文 1 为设备主动回连请求广播报文（比如设备有数据上报需要连接网关）。 bit1：心跳时长间隔单位。0 为单位毫秒，1 单位为秒。 **PP**：采用小端格式传输， 2 个字节，unsigned int 类型参数，16 进制。标识设备心跳间隔时长， 例如 5000ms，可选。

例如响应包的报文为：4892d48554157454941492d013130314332383031023c00

对应的格式如下：**48492d**（HI-） **4855415745494149**（HUAWEIAI）**2d**（-）**31**（X）**31303143**（YYYY 为 101C）**32383031**（NNNN 为 2801）**02**（M 对应为 0b00000010 为广播心跳，单位为秒）**3c00**（PP 对应 10 进制为 60 秒）

示例代码如下：

- 未注册常态广播

```
//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]
};
```

- 已注册常态广播

```
//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_0010[] = {
    0x14, 0x9,  'H', 'I', '-', 'A', 'A', 'A', '-', 0x31,
    demoDevInfo->productId[0], demoDevInfo->productId[1], demoDevInfo-
>productId[2], demoDevInfo->productId[3],
    demoDevInfo->sn[8], demoDevInfo->sn[9], demoDevInfo->sn[10],demoDevInfo-
>sn[11],
    MM,PP[0],PP[1]
};
```

2. 使用 HiLinkSDK，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。

HiLinkSDK 已实现报文发送的功能，只需调用 BLE_SetAdvType 接口，设置发送广播类型，即可对外广播蓝牙报文。

靠近发现广播发送示例代码如下：

```
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};


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)HILINK_BT_GetTime(&currentTime);
        HILINK_BT_SleepMs(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 = HILINK_BT_CreateTask(&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;
}

/*
 *   常态广播
 */
void ble_adv_normal()
{
    int reg = HILINK_IsRegister()
    unsigned char adv_data[] = {
        0x02, 0x01, 0x06, 0x11, 0xFF,
        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};
    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;
    //未注册，需要发送未注册常态广播
    if (0 == reg)
    {
        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';
    }

    //已注册，发送已注册常态广播
    else
    {
        printf("function:[%s],device has been registered\n", __FUNCTION__);
          /*  当蓝牙断开连接时，为了保持设备能够被重连，或者通过 H5 连接设备，需要发送常态广播。常态
广播，
                蓝牙响应数据(adv_rsp_data)和广播内容(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。*/
        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)
    {
        HILINK_BT_DeleteTask(g_advCtrl.advTaskHandle, BLE_ADV_TIME_CTRL_TASK);
```



```
        g_advCtrl.advTaskHandle = NULL;
    }
}

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)
    {
        HILINK_BT_DeleteTask(g_advCtrl.advTaskHandle, BLE_ADV_TIME_CTRL_TASK);
```



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

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

    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();
}
```

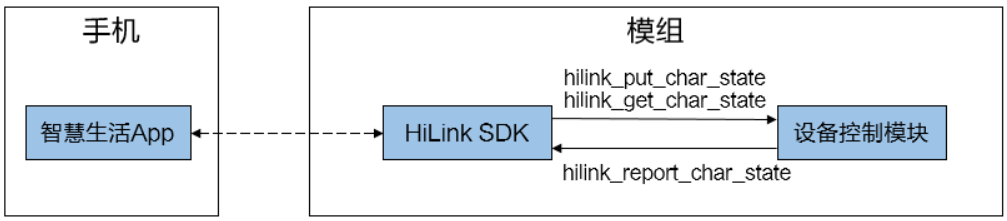
3.3.4 开发设备功能

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_put_char_state 和 hilink_get_char_state 两个函数，并结合 SDK 提供的接口 hilink_report_char_state 开发设备控制和状态上报功能。

表3-14 设备控制相关函数

函数名称	是否需要开发者实现	说明
hilink_put_char_state	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_get_char_state	是	云端通过 HiLink SDK 获取设备状态信息，或者 HiLink SDK 主动调用接口获取设备状态信息。
hilink_report_char_state	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3-3 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，以及服务支持的控制、查询、上报等操作。设备功能开发必须按照 profile 分别实现对应的功能。

以开关的控制（handle_put_switch）和查询（handle_get_switch）的功能实现为例：

```
// 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_put_char_state 传递过来的信息
int handle_put_switch(const char* svc_id, const char* payload, unsigned int len)
{
    void* pJson = hilink_json_parse(payload);
    if (pJson == NULL){
        printf("JSON parse failed in PUT cmd: ID-%s \r\n", svc_id);
        return INVALID_PACKET;
    }
    bool on;
    bool* on_p = NULL;
    if (hilink_json_get_number_value(pJson, "on", &on) == 0 &&
        (on == 0 || on == 1)) {
        on_p = &on;
    }
    g_device_info.switch_on = *on_p;
    if (pJson != NULL) {
        hilink_json_delete(pJson);
    }
    printf("handle func:%s, sid:%s \r\n", __FUNCTION__, svc_id);
    return M2M_NO_ERROR;
}

// 处理从 hilink_get_char_state 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_put_char_state”函数和“hilink_get_char_state”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;
    // handle hilink_put_char_state cmd function
    handle_put_func putFunc;
    // handle hilink_get_char_state cmd function
    handle_get_func getFunc;
} HANDLE_SVC_INFO;

//不支持 hilink_put_char_state 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_put_char_state，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_put_char_state”和“hilink_get_char_state”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_report_char_state(svc_id, payload, len, task_id);
    printf("report %s result is %d \r\n", svc_id, err);
    return err;
```



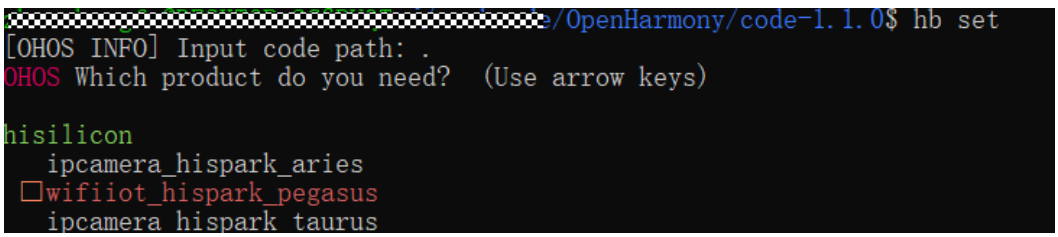
```
}


// ---------------------------------- //
// 以下两个函数在 hilink_device.c 中      //
// ---------------------------------- //
int hilink_put_char_state(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_get_char_state(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

3.4 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3-4 构建设置示例



步骤 2 在工程根目录，执行“hb build -f”命令进行版本构建。在版本最终发布时，执行“hb build -f -b release”命令进行发布版本构建。即发布 release 版本。

出现“xxxx　build success”字样，则证明构建成功。

----结束

3.5 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

4 功能验证

4.1 预置激活码
4.2 测试配网和设备控制

4.1 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

4.2 测试配网和设备控制

4.2.1 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。

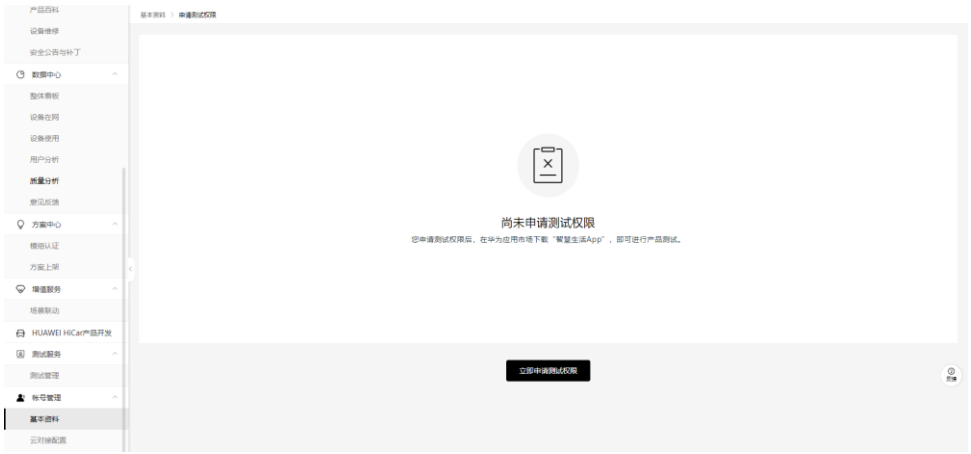
- 方式一：申请测试权限。

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4-1 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4-2 智慧生活 App 下载方式

----结束

4.2.2 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4-3 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

4.2.3 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4-1 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。

5 附录

图5-1 产品创建时需要填写的产品基本信息

表5-1 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5-2 产品预约认证时填写的信息

表5-2 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) + '/' + $(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。
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）。Device Partner 平台填写的会上传到设备云端，作为基线。

产品提交认证后，Kit Framework 测试用例中会将设备云端的与设备端的参数进行对比，如果完全一致，则通过测试；否则无法通过认证。

6 参考

- 华为智能硬件合作伙伴 > 原子化服务开发
- 华为智能硬件合作伙伴 > 常见问题

