HarmonyOS Connect 直连套餐（Wi-Fi/Combo）

文档版本 01

发布日期 2024-04-28

华为终端有限公司

商标声明


、HUAWEI、 
、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 文档更新记录.....3
3 准备工作.....4
4 固件开发.....6
4.1 简介.....6
4.2 开发设备功能.....6
4.2.1 配置 hilink_device.c 中产品信息.....6
4.2.2 配置 hilink_device.c 中的服务信息.....8
4.2.3 配置 hilink_demo.c 中的设备发现方式信息.....9
4.2.3.1 设备发现方式简介.....9
4.2.3.2 NFC 碰一碰.....10
4.2.3.3 蓝牙碰一碰.....13
4.2.3.4 蓝牙靠近发现.....18
4.2.4 实现设备控制功能.....30
4.2.5 网络优化通用适配.....34
4.3 编译固件.....39
4.4 烧录固件.....40
5 功能验证.....41
5.1 测试配网和设备控制.....41
5.1.1 配置调测环境.....41
5.1.2 测试设备配网与设备控制功能.....42
5.1.3 添加设备失败问题分析.....44
6 附录.....45
6.1 3861 网络优化工程修改示例.....45
6.1.1 工程更新.....45
6.1.2 三方工程 Wi-Fi 参数修改 demo 示例.....45
6.1.3 三方工程 Wi-Fi 参数修改示例.....46
6.2 AIW4211 网络优化工程修改实例.....47
6.2.1 工程更新.....47

6.2.2 三方工程 Wi-Fi 参数修改 demo 示例.....47
6.2.3 三方工程 Wi-Fi 参数修改实例示例.....47
6.3 8720 网络优化工程修改示例.....49
6.3.1 三方工程 Wi-Fi 参数修改 demo 示例.....49
6.3.2 三方工程 Wi-Fi 参数修改示例.....49
6.4 ASR 网络优化工程修改实例.....50
6.4.1 三方工程 Wi-Fi 参数修改 demo 示例.....50
6.4.2 三方工程 Wi-Fi 参数修改示例.....51
6.5 BK7231M 网络优化工程修改示例.....51
6.5.1 三方工程 Wi-Fi 参数修改 demo 示例.....51
6.5.2 三方工程 Wi-Fi 参数修改示例.....52
6.6 BL602C 网络优化工程修改示例.....54
6.6.1 三方工程 Wi-Fi 参数修改 demo 示例.....54
6.6.2 三方工程 Wi-Fi 参数修改示例.....54
7 参考.....56

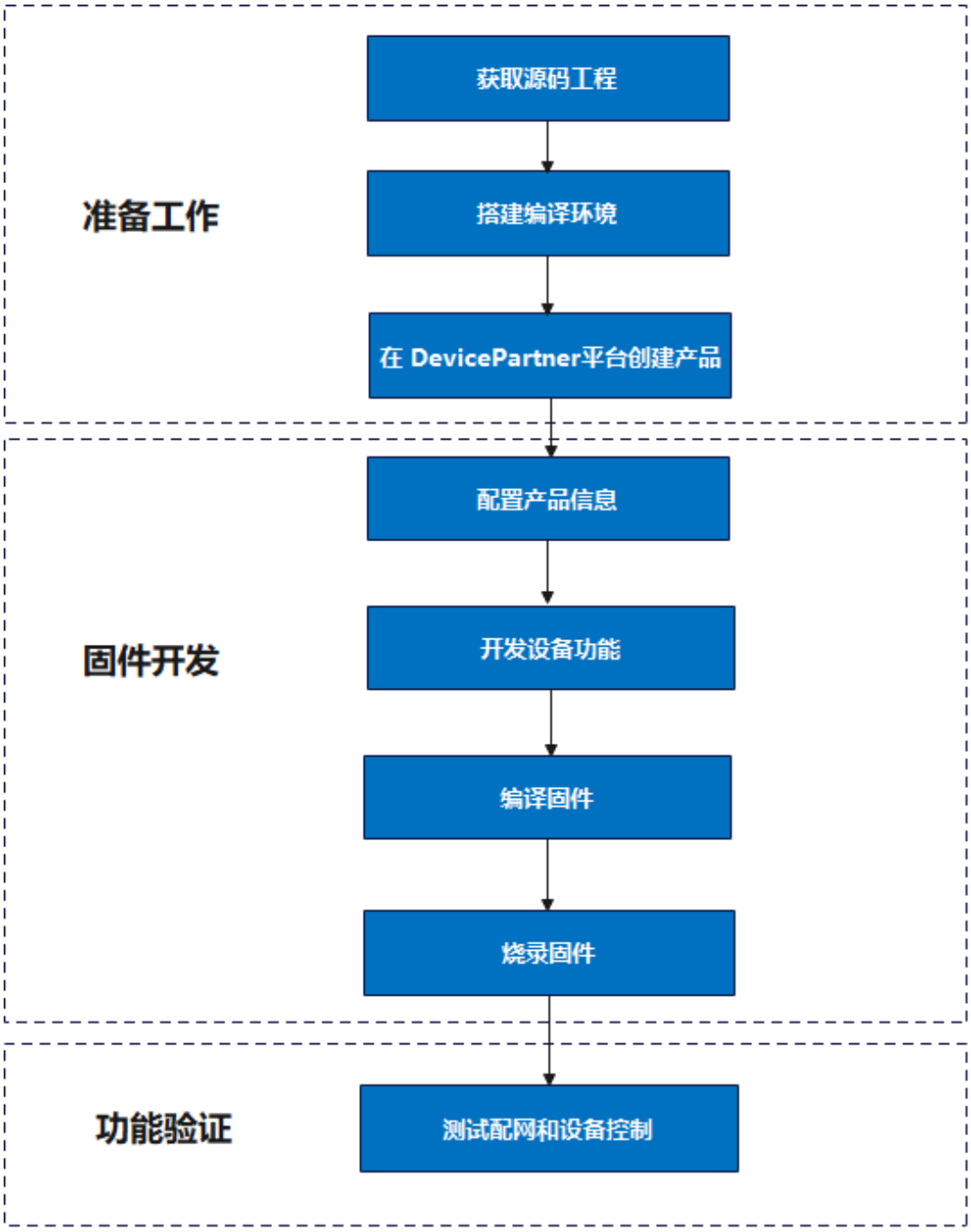
1 概述

简

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
2023-6-07	3.0	【变更】移除 Kit Framework 相关操作说明信息。
2022-5-30	2.0	1.【变更】SDK 升级后，接口和产品信息配置有调整，详见 4.2.1 配置 hilink_device.c 中产品信息 和 4.2.2 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 4.2.3 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 4.2.4 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 zh-cn_topic_0000001618863593.xml。 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，所集成的 HiLink SDK 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 所需产品信息。

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 点击右上角箭头引导处导出产品信息

----结束

4 固件开发

4.1 简介
4.2 开发设备功能
4.3 编译固件
4.4 烧录固件

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
HiLink SDK	hilink_device.c	产品 ID、设备类型 ID、厂商 ID、设备类型名、厂商名称、服务信息	用于设备配网过程中的 Wi-Fi 信息获取，以及设备注册与设备上报时支持的服务列表。

4.2 开发设备功能

4.2.1 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4-1 SSID 配置

表4-2 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 zh-cn_topic_0000001618863593.xml#table3768034573 中用户可见版本号介绍。	-
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	-

配置项	说明	示例
	（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 4-1，需要和 Device Partner 平台 SSID 保持一致。	Light
MANUAFACTURER_NAME	产商名称。参考图 4-1，需要和 Device Partner 平台 SSID 保持一致	HUAWEI

4.2.2 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

请确保工程代码实现与 DP 平台物模型定义中的信息配置一致，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 4-3 所示，则对应的工程代码示例如下：

表4-3 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

4.2.3 配置 hilink_demo.c 中的设备发现方式信息

4.2.3.1 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4-4 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
NFC 碰一碰	12.0.0.303	Combo 或 Wi-Fi 模组
蓝牙碰一碰	12.0.5.302	Combo 模组
蓝牙靠近发现	12.0.5.302	Combo 模组

4.2.3.2 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4-5 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[] = {
    0x02, 0x01, 0x06
};
```

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 4-10。

示例代码如下：

```
unsigned char myrspData 0001[] = {
    0x16, 0x09,  '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");
        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[] = {
        0x02, 0x01, 0x06
    };

    unsigned char myrspData 0001[] = {
        0x16, 0x09,  '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();
}
```

4.2.3.3 蓝牙碰一碰

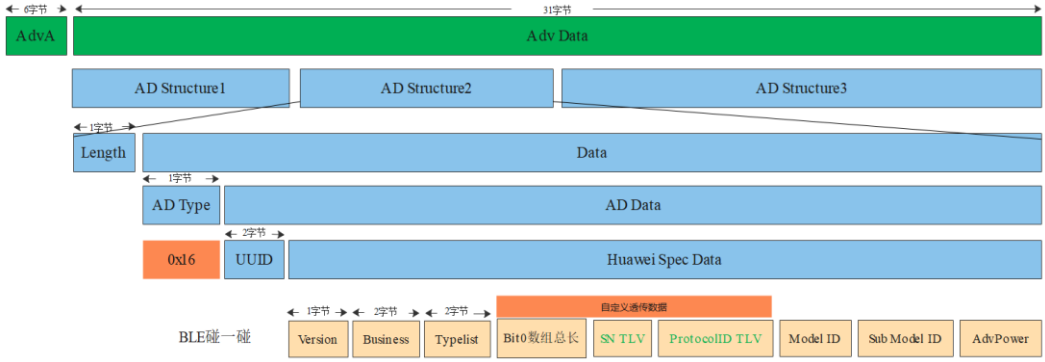
蓝牙碰一碰广播规范

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，此请求半模态进行设备注册，ProtocolID 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，手机半模态直接进入设备控制页，ProtocolID 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 4-6。

表4-6 蓝牙碰一碰广播包规范

内容	长度（Byte）	值	必选/可选	说明

内容				长度（Byte）	值	必选/可选	说明
AD Structure1	Length			1	0x02	必选	类型和值的总长度
	AD Type			1	0x01	必选	0x01 表示 flag
	Value			1	0x06	必选	支持的 LE/BE/EDR 的 Flag 信息
AD Structure2	Length			1	0xXX	必选	类型和值的总长度
	AD Type			1	0x16	必选	广播类型，0x16 表示蓝牙服务数据
	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	0x01	必选	华为蓝牙广播协议版本
		business		1	0x06	必选	06 表示碰一碰业务
		businessEx		1	0x00	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	0x17	必选	蓝牙协议 ID 类型
			Protocol D L	1	0x01	必选	蓝牙协议 ID 长度

内容				长度（Byte）	值	必选/可选	说明
			Protocol D V	1	0xXX	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-Wi-Fi+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-Wi-Fi+BLE 的 Combo 设备，已经由蓝牙辅助 Wi-Fi 配网注册成功，目前 Wi-Fi 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

内容				长度（Byte）	值	必选/可选	说明
							代理注册。
			SN T	1	0x14	必选	设备 SN
			SN L	1	0x02	必选	设备 SN 长度
			SN V	2	0xXXXX	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	0x91	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	0xXXXXXXXX	必选	产品 ID 值，产品 ID 的 ASCII 码值。
		subProdId V		1	0xXX	必选	子产品 ID 值，默认 00
		AdvPower V		1	0xXX	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4-7 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
Wi-Fi+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。
二碰广播，owner 弹框，member 不弹框	0x16	0x16	0xEEFD0106001700071402NNNN170100XXXXXXXXSSPP	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	0xEEFD0106001700071402NNNN170101XXXXXXXXSSPP	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	0xEEFD0106001700071402NNNN17010CXXXXXXXXSSPP	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;
    }
}
```

4.2.3.4 蓝牙靠近发现

蓝牙靠近发现广播规范

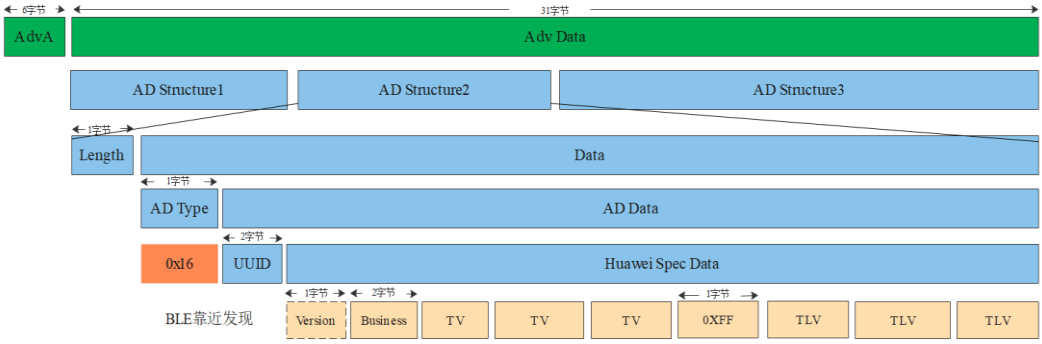
在蓝牙靠近发现中，BLE 广播分为靠近发现弹框 H5 广播（简称靠近发现广播）和常态广播。为避免骚扰用户，靠近发现广播需要主动触发（开机/上电/双击等）才能发送。靠近发现广播规范定义了广播包（Advertising Data）和响应包（Scan Response），根据 Advertising Data 可分为一靠广播和二靠广播；常态广播规范只定义了响应包（Scan Response），也就是蓝牙广播名称。

- 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 H5 半模态，请求 H5 半模态进行设备注册，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 H5 半模态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 4-8。

表4-8 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	0x02	必选	类型和值的总长度
	AD Type		1	0x01	必选	0x01 表示 flag
	Value		1	0x06	必选	支持的 LE/BE/EDR 的 Flag 信息
AD Structure2	Length		1	0xXX	必选	类型和值的总长度
	AD Type		1	0x16	必选	广播类型，0x16 表示蓝牙服务数据
	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	0x01	必选	华为蓝牙广播协议版本
		business	1	0x01	必选	01 表示靠近发现业务
		businessEx	1	0x07	必选	0x07 表示拉起 H5 半模态
		subProdIdT	1	0x04	必	子产品 ID 类型

内容			长度（Byte）	值	必选/可选	说明
					选	
		subProdIdV	1	0xXX	必选	子产品 ID 值，默认 00
		AdvPowerT	1	0x11	必选	测距字段类型
		AdvPowerV	1	0xXX	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	0x12	必选	产品 ID 类型
		prodId V	4	0xXXXXXXXX	必选	产品 ID 值，产品 ID 的 ASCII 码值
		0xFF	1	0xFF	必选	分隔符
		ProtocolDT	1	0x17	必选	蓝牙协议 ID 类型
		ProtocolDL	1	0x01	必选	蓝牙协议 ID 长度
		ProtocolDV	1	0xXX	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：

内容			长度（Byte）	值	必选/可选	说明
						– 0-弹设备控制框 – 1-Wi-Fi+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-Wi-Fi+BLE 的 Combo 设备，已经由蓝牙辅助 Wi-Fi 配网注册成功，目前 Wi-Fi 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	0x14	必选	设备 SN
		SN L	1	0x02	必选	设备 SN 长度

内容			长度（Byte）	值	必选/可选	说明
		SN V	2	0xXXXX	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	0x91	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4-9 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
Wi-Fi+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 不弹框	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 后两位的 ASCII 码值。
二靠广播，owner、member 都弹框	0x17	0x16	0xEEFD01010D04**SS**11**PP**12**XXXXXXXX**FF1701011402**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 后两位的 ASCII 码值。
二靠广播，离线更换配网信息	0x17	0x16	0xEEFD01010D04**SS**11**PP**12**XXXXXXXX**FF17010C1402**NNNN**	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：

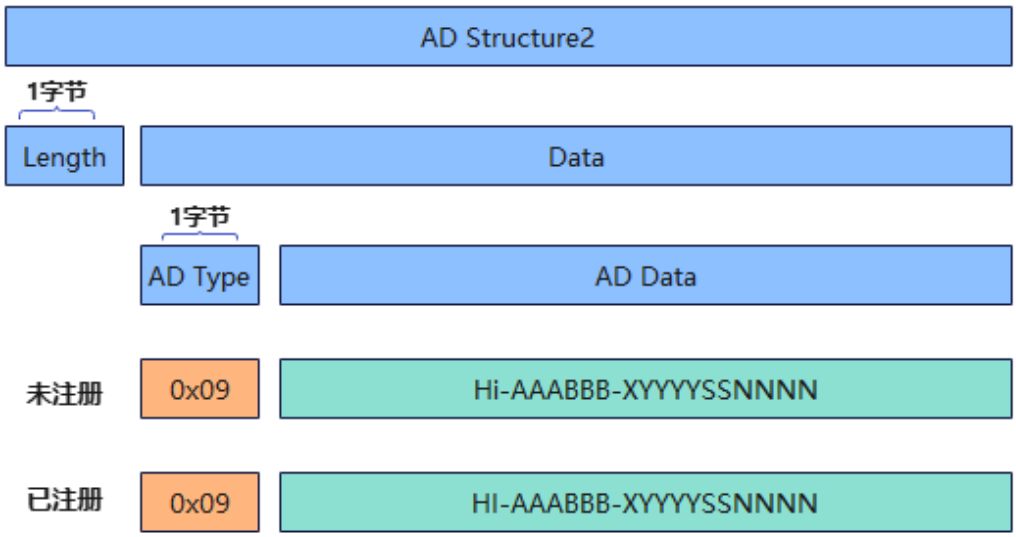


表4-10 响应包（Scan Response）广播实例

适用场景	长度	类型	值	说明
靠近发现广播、常态广播和碰一碰广播	可变	0x09	未注册： “Hi-**AAABBB-XYYYYSSNNNN**” 已注册： “HI-**AAABBB-XYYYYSSNNNN**”	**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 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：**

适用场景	长度	类型	值	说明
				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； **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；

例如，响应包的报文为：48492d48554157454941492d313130314330303032383031

对应的格式如下：**48492d**（HI-） **4855415745494149**（HUAWEIAI）**2d**（-）**31**（X）**31303143**（YYYY 为 101C）**3030**（SS 为 00）**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 4-8，对外广播未注册常态蓝牙报文。

表4-11 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[] = {
```

```
    0x02, 0x01, 0x06
};
```

未注册常态广播的蓝牙广播响应数据(myRspData)具体字段参考表 4-10，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[] = {
    0x02, 0x01, 0x06
};
unsigned char myrspData 0001[] = {
    0x13, 0x09,  '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 H5 半模态时，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。
```

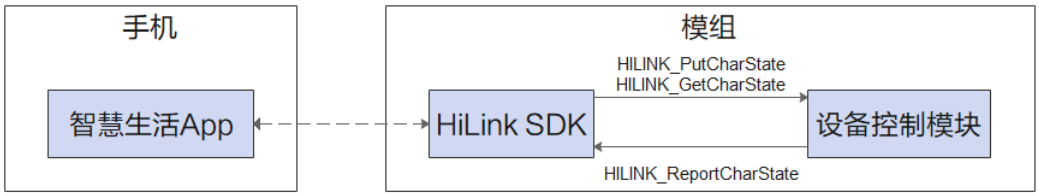
4.2.4 实现设备控制功能

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 实现开发设备控制和状态上报功能。

表4-12 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 调用此函数，伙伴需要再对服务进行识别、分发和处理。
HILINK_GetCharState	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_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4-2 手机和模组交互关系图



步骤 1 根据 profile 进行产品定义、设备控制和状态查询功能。

- profile 定义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

4.2.5 网络优化通用适配

本节介绍如何开启网络优化功能，开发者可选择开启网络优化的功能类型，通过调用 HILINK_RegWiFiRecoveryCallback 函数把网络优化相关功能接口注册给 SDK 使用。网络优化功能主要包含智能心跳，多扫多连，AP 智选。其中多扫多连和 AP 智选涉及 WiFi 连接机制，需要平台适配对应接口。

表4-13 网络优化相关接口描述，在头文件 hilink_network_adapter.h 中，表 1 接口功能在 SDK 内部实现。

接口名称	是否需要开发者实现	说明
HILINK_ScanAP	否	根据参数扫描周围 AP 信息。
HILINK_GetAPScanResult	否	SDK 调用 HILINK_ScanAP 后，调用此函数获取扫描周围 AP 信息的结果。
HILINK_ConnectWiFiByBssid	否	连接指定 bssid 与配网 ssid 同名的 Wi-Fi。可以通过 bssid 连接到指定的路由器。
HILINK_GetLastConnectResult	否	获取上一次连接 WiFi 失败原因。

接口名称	是否需要开发者实现	说明
HILINK_RestartWiFi	否	重新启动 WiFi 模块。该接口只重启 WiFi 的 STA，不重启设备。
HILINK_RegWiFiRecoveryCallback	否	伙伴实现网络优化相关功能函数，通过此接口注册给 SDK 使用 前提条件：HILINK_Main 初始化之前调用。
HILINK_SetWiFiRecoveryTimesParam	否	设置 WiFi 自恢复时的扫描、连接相关参数。scanTimes 扫描次数，默认 3 次；connectTimes 连接次数，默认 3 次。 前提条件：HILINK_Main 初始化之前调用。
HILINK_SetHeartbeatLimit	否	当环境中路由器 WiFi 信号质量不好时，设置心跳超时离线导致重连 WiFi 的阀值。一段时间内如果心跳超时次数达到该阀值，则进行优化，阀值默认为 3。该接口在 HILINK_Main 初始化之前调用。

表4-14 网络优化功能接口结构体 WiFiRecoveryApi 描述，在头文件 hilink_network_adapter.h 中，以下接口提供给 SDK 调用，由开发者实现并通过 HILINK_RegWiFiRecoveryCallback 函数注册给 SDK。

函数名称	是否需要开发者实现	说明
unsigned int (*getWifiRecoveryType)(void);	是	获取网络优化功能类型。开发者通过该接口的返回值，选择开启网络优化的功能类型。 返回 0x00：表示关闭网络优化所有功能； 返回 0x01：表示开启路由器引导设备连接至较优 AP 功能； 返回 0x02：表示开启 SDK 连接 WiFi 逻辑，此时需要设备配合关闭本身 WiFi 重连功能； 返回(0x01 | 0x02）：表示功能全开。 注：使用支持网路优化的 SDK 版本，要求返回 (0x01 | 0x02)，开启所有功能。 该接口由开发者自己实现，在初始化 WiFiRecoveryApi 时使用。
int (*scanAP)(const HILINK_APScanParam *param);	可选	设备配网阶段或断网重连阶段，SDK 调用此接口根据参数扫描周围 AP 信息。

函数名称	是否需要开发者实现	说明
		该接口开发者可以自己实现，也可使用 SDK 内部提供。如果使用 SDK 内部提供，在初始化 WiFiRecoveryApi 时，可直接使用表 1 中的 HILINK_ScanAP 函数。
int (*getAPScanResult)(HILINK_APList *scanList);	可选	scanAP 接口扫描完成后，SDK 通过此接口获取目标 AP 扫描结果。如果环境中有多个同名目标 AP 会全部返回。 该接口开发者可以自己实现，也可使用 SDK 内部提供。如果使用 SDK 内部提供，在初始化 WiFiRecoveryApi 时，可直接使用表 1 中的 HILINK_GetAPScanResult 函数。
int (*connectWiFiByBssid)(int securityType, const unsigned char *bssid, unsigned int bssidLen);	可选	SDK 在扫描结果中找到信号质量最优的 AP，调用此接口连接到最优 AP。 该接口开发者可以自己实现，也可使用 SDK 内部提供。如果使用 SDK 内部提供，在初始化 WiFiRecoveryApi 时，可直接使用表 1 中的 HILINK_ConnectWiFiByBssid 函数。
int (*lastConnResult)(int *result);	可选	WiFi 连接过程中如果出现异常，SDK 通过此接口获取连接失败错误码。 该接口开发者可以自己实现，也可使用 SDK 内部提供。如果使用 SDK 内部提供，在初始化 WiFiRecoveryApi 时，可直接使用表 1 中的 HILINK_GetLastConnectResult 函数。
int (*restartWiFi)(void);	可选	设备离线一段时间无法恢复，SDK 调用此接口重启 WiFi。该接口只重启 WiFi 的 STA，不重启设备。 该接口开发者可以自己实现，也可使用 SDK 内部提供。如果使用 SDK 内部提供，在初始化 WiFiRecoveryApi 时，可直接使用表 1 中的 HILINK_RestartWiFi 函数。

表4-15 实现模组重启前的设备操作，在头文件 hilink_device.c 中。

函数名称	是否需要开发者实现	说明

函数名称	是否需要开发者实现	说明
HILINK_ProcessBeforeRestart	是	开启网络优化功能后，设备长时间离线无法恢复，SDK 通过该接口参数 flag=2 的状态通知设备要重启。开发者需实现 flag=2 时模组重启前的操作(如:保存系统状态等)。

操作步骤如下：

步骤 1 设备长时间离线重启通知处理。

需要开发者实现 hilink_device.c 中的 HILINK_ProcessBeforeRestart 函数，对新增的 flag=2 条件分支，实现模组重启前的操作(如:保存系统状态等)。

代码如下：

```
/*
* 功能：实现模组重启前的设备操作
* 参数：flag 入参，触发重启的类型
*          0 表示 HiLink SDK 线程看门狗触发模组重启；
*          1 表示 APP 删除设备触发模组重启；
*          2 表示设备长时间离线无法恢复而重启；
* 返回值：0 表示处理成功，系统可以重启，使用硬重启；
*         1 表示处理成功，系统可以重启，如果通过 HILINK_SetSdkAttr()注册了软重启
(sdkAttr.rebootSoftware），使用软重启；
*         负值表示处理失败，系统不能重启
* 注意：(1) 此函数由设备厂商实现；
*       (2) 若 APP 删除设备触发模组重启时，设备操作完务必返回 0，否则会导致删除设备异常；
*       (3) 设备长时间离线无法恢复而重启，应对用户无感，不可影响用户体验，否则不可以重启；
*/
int HILINK ProcessBeforeRestart(int flag)
{
    /* HiLink SDK 线程看门狗超时触发模组重启 */
    if (flag == 0) {
        /* 实现模组重启前的操作(如:保存系统状态等) */
        return -1;
    }
    /* APP 删除设备触发模组重启 */
    if (flag == 1) {
        /* 实现模组重启前的操作(如:保存系统状态等) */
        return 1;
    }
    /* 设备长时间离线触发模组重启，尝试恢复网络 */
    if (flag == 2) {
        /* 实现模组重启前的操作(如:保存系统状态等) */
        return -1;
    }
    return -1;
}
```


步骤 2 定义网路优化功能类型选择接口，使用支持网络优化功能的 SDK 版本，要求网络优化功能全部打开。

示例代码：

```
int GetWifiRecoveryType(void)
{
    /* (0x01 | 0x02)表示网络优化功能全开，返回 0 则关闭，具体定义详见 WiFiRecoveryApi 结构体 */
    return (0x01 | 0x02);
}
```

步骤 3 注册网络优化相关功能函数。

如果使用 SDK 内部提供的接口，除 getWifiRecoveryType 需要开发者实现外，scanAP、getAPScanResult、restartWiFi、connectWiFiByBssid、lastConnResult 直接使用 SDK 提供的相关接口初始化即可。

示例代码：

```
/* 网络优化开启 */
WiFiRecoveryApi recApi = {
    .getWifiRecoveryType = GetWifiRecoveryType,
    .scanAP = HILINK ScanAP,
    .getAPScanResult = HILINK GetAPScanResult,
    .restartWiFi = HILINK RestartWiFi,
    .connectWiFiByBssid = HILINK ConnectWiFiByBssid,
    .lastConnResult = HILINK GetLastConnectResult,
};
ret = HILINK RegWiFiRecoveryCallback((const WiFiRecoveryApi *)&recApi,
sizeof(WiFiRecoveryApi));
if (ret != 0) {
    printf("reg wifi recovery api failed\r\n");
}
/* 启动 Hilink SDK */
if (HILINK Main() != 0) {
    printf("HILINK Main start error");
}
```

步骤 4 开启路由引导设备连接指定 AP 功能（GetWifiRecoveryType 返回值包含 0x01）。

修改 wifi 单信道扫描时长为 210ms，信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

步骤 5 开启 WiFi 多扫多连并连接至最优 AP 功能。（GetWifiRecoveryType 返回值包含 0x02）。

1. 关闭 WiFi 自动重连，WiFi 自动重连关闭作用于设备运行整个生命周期（注：在 SDK 运行前执行该代码）；

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
```



```
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2.修改 WiFi 单信道扫描时长为 210ms，信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

3.增加 WiFi 连接结果状态码获取，详见 WiFiRecoveryApi 结构体的回调函数 lastConnResult。SDK 通过鸿蒙 RegisterWifiEvent 接口注册的 OnWifiConnectionChanged 回调获取 lastConnResult 状态码。

⚠ 注意

因密码错误导致 WIFI 连接失败，错误码要求返回 14 或 15 或 19 任意一个。其它场景的错误码无约束，根据实际情况传递即可。

为保证 WiFi 连接可靠性，只有确实是密码错误导致的连接失败才可以返回该错误码。

----结束

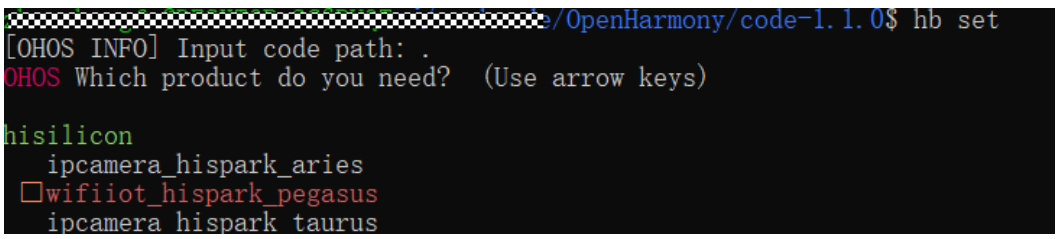
离线优化不同芯片代码适配示例

6 附录

4.3 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4-3 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

4.4 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 测试配网和设备控制

5.1 测试配网和设备控制

5.1.1 配置调测环境

步骤 1 配置测试帐号。

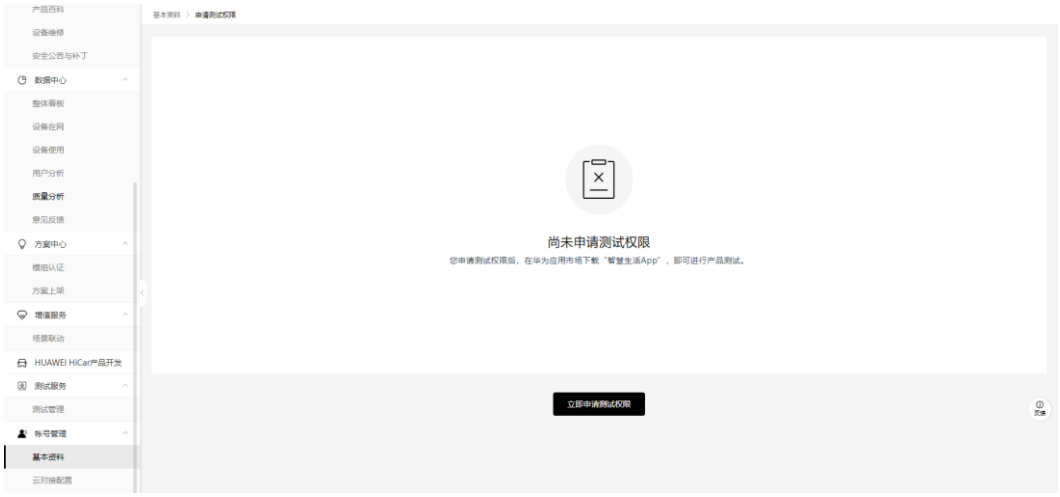
- 方式一：申请测试权限。

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

5.1.2 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

5.1.3 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两个过程阶段。

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	-	-
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]”。

6 附录

6.1 3861 网络优化工程修改示例
6.2 AIW4211 网络优化工程修改实例
6.3 8720 网络优化工程修改示例
6.4 ASR 网络优化工程修改实例
6.5 BK7231M 网络优化工程修改示例
6.6 BL602C 网络优化工程修改示例

6.1 3861 网络优化工程修改示例

6.1.1 工程更新

工程请联系海思 FAE 获取，否则编译报错，“hi_wifi_scan_strategy_stru”结构体和“hi_wifi_set_scan_strategy”接口未定义。

说明

“hi_wifi_scan_strategy_stru”：扫描策略结构体；

“hi_wifi_set_scan_strategy”：设置扫描策略。

6.1.2 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

   Wifi 自动重连关闭作用于设备运行整个生命周期。

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
```



```
    ...
}
```

2. 修改扫描信道停留时间。

   信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.1.3 三方工程 Wi-Fi 参数修改示例

1. 关闭 Wifi 自动重连。

   接口 WifiErrorCode EnableWifi(void)，在 WiFi 使能时增加修改：

```
/* 定义 int 型变量，接收接口返回值 */
int hiRet;
/* 如果开启了网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02) {
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 关闭 WiFi 自动重连 */
hiRet = hi wifi sta set reconnect policy(WIFI RECONN POLICY DISABLE,
WIFI RECONN POLICY TIMEOUT, WIFI RECONN POLICY PERIOD,
WIFI RECONN POLICY MAX TRY COUNT);
} else {
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 如果没有开启网络优化功能 2，则开启 WiFi 自动重连 */
hiRet = hi wifi sta set reconnect policy(WIFI RECONN POLICY ENABLE,
WIFI RECONN POLICY TIMEOUT, WIFI RECONN POLICY PERIOD,
WIFI RECONN POLICY MAX TRY COUNT);
}
/* 接口返回异常处理：以下为复制的代码段，以实际为准 */
if (hiRet != HISI OK) {
    printf("[wifi service]:EnableWifi set reconn policy fail\n");
    if (UnlockWifiGlobalLock() != WIFI SUCCESS) {
        return ERROR WIFI UNKNOWN;
    }
    return ERROR WIFI UNKNOWN;
}
```

2. 修改扫描信道停留时间。

   接口 WifiErrorCode EnableWifi(void)，在 Wifi 使能时增加修改：

```
#define WIFI SCAN CNT 7 /* wifi 扫描时长，取值范围 1~10，单位 30ms。7 表示 210ms */
/* 如果开启了网络优化 */
if (GetWifiRecoveryType() != 0) {
    printf("wifi recovery: scan interval [%d]\r\n", (30 * WIFI SCAN CNT));
    /* 设置 wifi 扫描策略，关闭 STA/AP 再启动 STA/AP 恢复默认值 2 */
    hi wifi scan strategy stru scanStrategy;
scanStrategy.scan cnt = WIFI SCAN CNT;
/* 调用"设置扫描策略"接口修改 WiFi 扫描策略，修改单个信道停留时长为 210ms，保证环境中的 AP 尽
可能扫全 */
    int ret = hi wifi set scan strategy(ifName, &scanStrategy);
    /* 接口返回异常处理：以下为复制的代码段，以实际为准 */
```



```
if (ret != HISI OK) {
       printf("set wifi scan strategy fail [%d]\r\n", ret);
       return ERROR WIFI UNKNOWN;
   }
}
```

3. BUG 修复：WiFi 扫描结果中 rssi 的值被放大 100 倍。

   rssi 正常取值范围在(-100, 0)之间。WiFi 扫描结果中 rssi 返回值为正常值 100 倍，在接口 WifiErrorCode GetScanInfoList(WifiScanInfo* result, unsigned int* size)修改如下：

```
/* WiFi 信号指令 rssi 范围(-99, 0)dB，如果获取到的值被放大 100 倍需除 100 */
result[i].rssi = pstResults[i].rssi / 100;
```

6.2 AIW4211 网络优化工程修改实例

6.2.1 工程更新

请联系爱旗模组厂商获取支持网络优化工程，否则编译报错，
“ext_wifi_scan_strategy_stru”结构体和“aich_wifi_set_scan_strategy”接口未定义。

6.2.2 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

   Wifi 自动重连关闭作用于设备运行整个生命周期。

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2. 修改扫描信道停留时间。

   信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.2.3 三方工程 Wi-Fi 参数修改实例示例

1. 关闭 Wifi 自动重连。

   接口 WifiErrorCode EnableWifi(void)，在 Wifi 使能是增加修改。


```
/* 定义 int 型变量，接收接口返回值 */
int ret;

/* 如果开启了网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02) {
    printf("wifi recovery: autoconnect disable\r\n");
    /* 关闭 WiFi 自动重连 */
ret = aich wifi sta set reconnect policy(WIFI RECONN POLICY DISABLE,
WIFI RECONN POLICY TIMEOUT, WIFI RECONN POLICY PERIOD,
WIFI RECONN POLICY MAX TRY COUNT);
} else {
    printf("wifi recovery: autoconnect enable\r\n");
    /* 如果没有开启网络优化功能 2，则开启 WiFi 自动重连 */
ret = aich wifi sta set reconnect policy(WIFI RECONN POLICY ENABLE,
WIFI RECONN POLICY TIMEOUT, WIFI RECONN POLICY PERIOD,
WIFI RECONN POLICY MAX TRY COUNT);
}
/* 接口返回异常处理：以下为复制的代码段，以实际为准 */
if (ret != SOC OK) {
    printf("[wifi service]:EnableWifi set reconn policy fail\n");
    if (UnlockWifiGlobalLock() != WIFI SUCCESS) {
        return ERROR WIFI UNKNOWN;
    }
    return ERROR WIFI UNKNOWN;
}
```

2. 修改扫描信道停留时间。

   接口 WifiErrorCode EnableWifi(void)，在 Wifi 使能时增加修改。

```
#define WIFI_SCAN_CNT 7 /* wifi 扫描时长，取值范围 1~10，单位 30ms。7 表示 210ms */

/* 如果开启了网络优化 */
if (GetWifiRecoveryType() != 0) {
    printf("wifi recovery: scan interval time [%d]\r\n", (30 * WIFI SCAN CNT));
    ext wifi scan strategy stru scanStrategy;
    scanStrategy.scan cnt = WIFI SCAN CNT;
    /* 调用“aich_wifi_set_scan_strategy”接口修改 WiFi 扫描策略，修改单个信道停留时长为
210ms，保证环境中的 AP 尽可能扫全 */
int ret = aich wifi set scan strategy(ifName, &scanStrategy);
    /* 接口返回异常处理：以下为复制的代码段，以实际为准 */
if (ret != SOC OK) {
        printf("set wifi scan strategy fail [%d]\r\n", ret);
        return ERROR WIFI UNKNOWN;
    }
}
```

3. BUG 修复：Wifi 扫描结果中 rssi 的值被放大 100 倍。

   rssi 正常取值范围在(-100, 0)之间。WiFi 扫描结果中 rssi 返回值为正常值 100 倍，在接口 WifiErrorCode GetScanInfoList(WifiScanInfo* result, unsigned int* size)修改如下：

```
/* WiFi 信号指令 rssi 范围(-99, 0)dB，如果获取到的值被放大 100 倍需除 100 */
result[i].rssi = pstResults[i].rssi / 100;
```


6.3 8720 网络优化工程修改示例

6.3.1 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

   Wifi 自动重连关闭作用于设备运行整个生命周期。

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2. 修改扫描信道停留时间。

   信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.3.2 三方工程 Wi-Fi 参数修改示例

1. 关闭 WiFi 自动重连

   接口 WiFiErrorCode ConnectToNoLock(int networkId)，在 WiFi 连接之前修改两处：

   - 修改在 WiFi 连接前：

```
int connect_result, max_retry = 1; /* max_retry 改之前为 5。减少重连次数，
ConnectToNoLock 是阻塞的，连接次数太多，影响其它业务流程 */
/* 注：8720 在 WiFi 连接前会先调用 wifi_set_autoreconnect(0);禁用重连模式，所以网络
优化功能开启后，保证不会打开 WiFi 重连模式即可 */
WIFI_CNT:/* 截取的代码片段，以实际为准 */
    /* 首次连接 WiFi 且网络优化功能 2 没有开启 */
    if((max retry == 1) && ((GetWifiRecoveryType() & 0x02) != 0x02)) {
        printf("wifi recovery: enable autoreconnect\r\n");
        /* 启用无限重连模式，设置 2 */
wifi set autoreconnect(2);
    }
```

   - 修改在 Wifi 连接后：

```
   /* 网络优化功能 2 没有开启 */
if((GetWifiRecoveryType() & 0x02) != 0x02) {
        printf("wifi recovery: ebable autoreconnect\r\n");
        /* 启用无限重连模式，设置 2 */
wifi set autoreconnect(2);
}
/* 以下为截取的代码片段，以实际为准 */
```



```
    if(config->ipType == DHCP)
    {
        LwIP DHCP(0, DHCP START);
    }
```

2. 修改扫描信道停留时间。

   接口 WifiErrorCode AdvanceScanNoLock(WifiScanParams *params)，在 WiFi 扫描之前增加修改：

```
/* 网络优化功能开启 */
if (GetWifiRecoveryType() != 0) {
    printf("wifi recovery: scan interval [210]\r\n");
    /* 网络优化功能开启，修改 WiFi 单信道停留时长为 210ms，保证环境中的 AP 尽可能扫全 */
wifi set partial scan chan interval(210);
}
```

3. BUG 修复：Wifi 断开状态码返回。

   Wifi 连接失败错误码通过回调接口“void (*OnWifiConnectionChanged)(int state, WifiLinkedInfo* info);”传递给 SDK，WiFi 连接错误码在第二个参数“WifiLinkedInfo* info”中携带。“WifiLinkedInfo* info”通过接口“WifiErrorCode GetLinkedInfo(WifiLinkedInfo* result)”获取时，对 WiFi 连接失败场景，没有传递连接失败错误码。下述修改是 WiFi 连接失败场景 WiFi 错误码获取方法。

   接口 WifiErrorCode GetLinkedInfoNoLock(WifiLinkedInfo* result)，增加 WiFi 断开时错误码获取如下：

```
/* WiFi 连接成功 */
if ((wifi is connected to ap() == RTW SUCCESS) && (wifi get setting(WLAN0 NAME,
&setting) == RTW SUCCESS))
{
    /* WiFi 连接成功代码逻辑，已实际为准 */
    ...
}
/* WiFi 连接失败：以下代码为新增 */
else
{
    /* 获取 WiFi 连接失败错误码 */
result->disconnectedReason = wifi get last error();
/* 获取 WiFi 连接状态 */
    result->connState = WIFI DISCONNECTED;
}
```

6.4 ASR 网络优化工程修改实例

6.4.1 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

```
WiFi 自动重连关闭作用于设备运行整个生命周期。
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 实现关闭自动重连 */
```



```
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2. 修改扫描信道停留时间。

- ASR 如果不支持 WiFi 停留时间修改。为了尽可能把环境中的 AP 扫全，建议通过接口修改扫描参数，增加扫描次数达到扫全目的：

接口：int HILINK_SetWiFiRecoveryTimesParam(unsigned int scanTimes, unsigned int connectTimes);

scanTimes：扫描次数，默认 3 次，可以把该参数改大以提高 AP 扫全的概率。增大扫描次数会增加配网时长，如果产品有配网时长规格，建议扫描时长维持在 8s 左右，但要平衡好配网时长和 AP 扫全的关系，可以通过多次测试来确定扫描次数。

connectTimes：连接次数，默认 3 次，如果使用默认连接次数可以直接传入 3。

使用：HILINK_SetWiFiRecoveryTimesParam 在 HILINK_Main 前调用即可。

- 信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.4.2 三方工程 Wi-Fi 参数修改示例

1. 关闭 Wifi 自动重连。

   接口 WifiErrorCode ConnectTo(int networkId)，在 WiFi 连接之前增加修改如下：

```
/* 如果开启网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02) {
    printf("wifi recovery: disable autoconnect\r\n");
    /* 关闭 WiFi 自动重连 */
lega_wlan_set_sta_autoconnect(0); // 1 开 0 关
}
```

6.5 BK7231M 网络优化工程修改示例

6.5.1 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

   Wifi 自动重连关闭作用于设备运行整个生命周期。

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
```



```
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2. 修改扫描信道停留时间。

   信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.5.2 三方工程 Wi-Fi 参数修改示例

1. 支持 BSSID 连接。

   接口 WifiErrorCode ConnectTo(int networkId)，Wifi 连接时新增修改如下：

```
/* 如果开启网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02)
{
    /* WiFi 连接时指定 BSSID 连接 */
    user bssid app init(g wifiConfigs[networkId].bssid,
g wifiConfigs[networkId].preSharedKey);
}
/* 如果没有开启网络优化功能 2 */
else
{
    /* 使用默认连接方式 */
    los wlan start sta(&wNetConfig,
g wifiConfigs[networkId].preSharedKey,64,chan);
}
```

2. 关闭 WiFi 自动重连。

   修改扫描信道停留时间　接口 WifiErrorCode AdvanceScan(WifiScanParams *params)，在 Wifi 断开时增加修改如下：

```
/* 如果开启网络优化功能 */
if(GetWifiRecoveryType() != 0)
{
    /* 释放 hw_list_ssid 内存 */
    if(hw list ssid)
    {
        free(hw list ssid);
}
/* 给 hw_list_ssid 重新申请内存 */
    hw list ssid = malloc(WIFI MAX SSID LEN);
    if(hw list ssid == NULL)
    {
        printf("hw list ssid malloc fail\r\n");
    }
```



```
    else
    {
   /* hw_list_ssid 初始化 */
os memset(hw list ssid, 0, WIFI MAX SSID LEN);
}
/* 设置扫描类型 */
    scan type = NORMAL SCAN TYPE;
}
/* 如果开启网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02)
{
    extern void user set auto reconnect switch(int value);
    /* 1 自动重连关，0 自动重连开 */
    user set auto reconnect switch(1);
    printf("wifi recovery: autoconnect disable\r\n");
}
if(scan type== NORMAL SCAN TYPE)
{
    /* 外部声明 */
    extern void user set chan time(uint32 t scan time);
    /* 调用"user_set_chan_time"接口修改 WiFi 扫描策略，修改单个信道停留时长为 210ms，保证
环境中的 AP 尽可能扫全 */
user set chan time(210*1024);
    printf("wifi recovery: scan interval time [210]\r\n");
    if(hw list ssid != NULL)
{
    /* ssid 拷贝 */
        strcpy(hw list ssid, params->ssid);
        /* 1 底层过滤指定 SSID 的 AP 存入缓存；0 不过滤存放所有扫描结果 */
        user set scan ssid(hw list ssid, 1);
}
mhdr scanu reg cb(los scan cb, 0);
/* 启动扫描 */
    bk wlan start scan();
}
else
{
    /* 设置扫描参数，并启动扫描 */
    ssid array = params->ssid;
    mhdr scanu reg cb(los scan cb, 0);
    bk wlan start assign scan(&ssid array, 1);
}
```

3. WiFi 连接状态码返回。

   接口 WifiErrorCode ConnectTo(int networkId)，在 Wifi 连接返回时修改如下：

```
/* 原代码段，以实际为准 */
while(time out cnt --)
{
    /* 原代码段，以实际为准 */
    osDelay(500);//4//1s
    /* 原代码段，以实际为准 */
GetLinkedInfo(&info);
/* 如果 WiFi 已连接或 WiFi 连接完成并返回了错误码 */
    if((info.connState == WIFI CONNECTED) || hw get wifi disc reason code())
    {
```



```
        ret = WIFI SUCCESS;
        break;
}
/* 如果开启网络优化功能 2 且 WiFi 连接完成并返回了错误码 */
    if (((GetWifiRecoveryType() & 0x02) == 0x02) &&
(hw get wifi disc reason code()))
    {
        ret = WIFI SUCCESS;
        break;
    }
}
return ret;
```

6.6 BL602C 网络优化工程修改示例

6.6.1 三方工程 Wi-Fi 参数修改 demo 示例

1. 关闭 Wifi 自动重连。

   Wifi 自动重连关闭作用于设备运行整个生命周期。

```
if((GetWifiRecoveryType() & 0x02) == 0x02) {
    /* 开启功能 2，关闭自动重连功能 */
    printf("wifi recovery: disable autoreconnect\r\n");
    /* 实现关闭自动重连 */
    ...
} else {
    /* 开启自动重连功能 */
    printf("wifi recovery: enable autoreconnect\r\n");
    /* 实现开启自动重连 */
    ...
}
```

2. 修改扫描信道停留时间。

   信道停留时间修改作用于设备运行整个生命周期。

```
if(GetWifiRecoveryType() != 0) {
    /* 开启网络优化 */
    printf("wifi recovery: scan interval [210]\r\n");
    /* 设置单信道扫描时长 210ms */
    ...
}
```

6.6.2 三方工程 Wi-Fi 参数修改示例

1. 关闭 Wifi 自动重连 。

   接口 WifiErrorCode ConnectTo(int networkId)，在 Wifi 连接之前增加修改：

```
/* 开启网络优化功能 2 */
if ((GetWifiRecoveryType() & 0x02) == 0x02) {
    printf("wifi recovery: autoconnect disable\r\n");
    /* 关闭 WiFi 自动重连 */
wifi mgmr sta autoconnect disable();
}
```


2. 修改扫描信道停留时间 。

   接口 WifiErrorCode AdvanceScan(WifiScanParams *params)，在 Wifi 扫描之前增加修改：

```
/* 网络优化功能开启 */
if (GetWifiRecoveryType() != 0) {
   printf("wifi recovery: scan interval time [210]\r\n");
   /* 调用“WifiSetScanIntervalTime”接口修改 WiFi 扫描策略，修改单个信道停留时长为 210ms，保证环境中的 AP 尽可能扫全 */
WifiSetScanIntervalTime(210);
}
```

3. BUG 修复。

   WiFi 断开后 sta 状态不是空闲，无法扫描到 AP。在 WiFi 断开时调用“wifi_mgmr_sta_disable(NULL);”，让 WiFi 状态变为空闲。接口 static void turbox_event_cb_wifi_event(input_event_t *event, void *private_data)，修改如下：

```
/* 原代码段，已实际代码为准 */
case CODE WIFI ON DISCONNECT:
{
   /* 原代码段，已实际代码为准 */
   TURBOX PRINTF("[APP] [EVT] disconnect %lld, Reason: %s\r\n", aos now ms(), wifi mgmr status code str(event->value));
/* 原代码段，已实际代码为准 */
hf gpio nlink(0);
/* 原代码段，已实际代码为准 */
hfwifi sta set connected state(0);
/* 如果网络优化功能 2 开启 */
   if ((GetWifiRecoveryType() & 0x02) == 0x02) {
      printf("wifi recovery: sta disable\r\n");
      /* sta disable */
wifi mgmr sta disable(NULL);
}
/* 以下为原代码段，已实际代码为准 */
   extern void WifiConnectionChangedCallback(int state, unsigned short disreason);
   WifiConnectionChangedCallback(0, event->value);
}
break;
```


7 参考

华为智能硬件合作伙伴 > 常见问题

