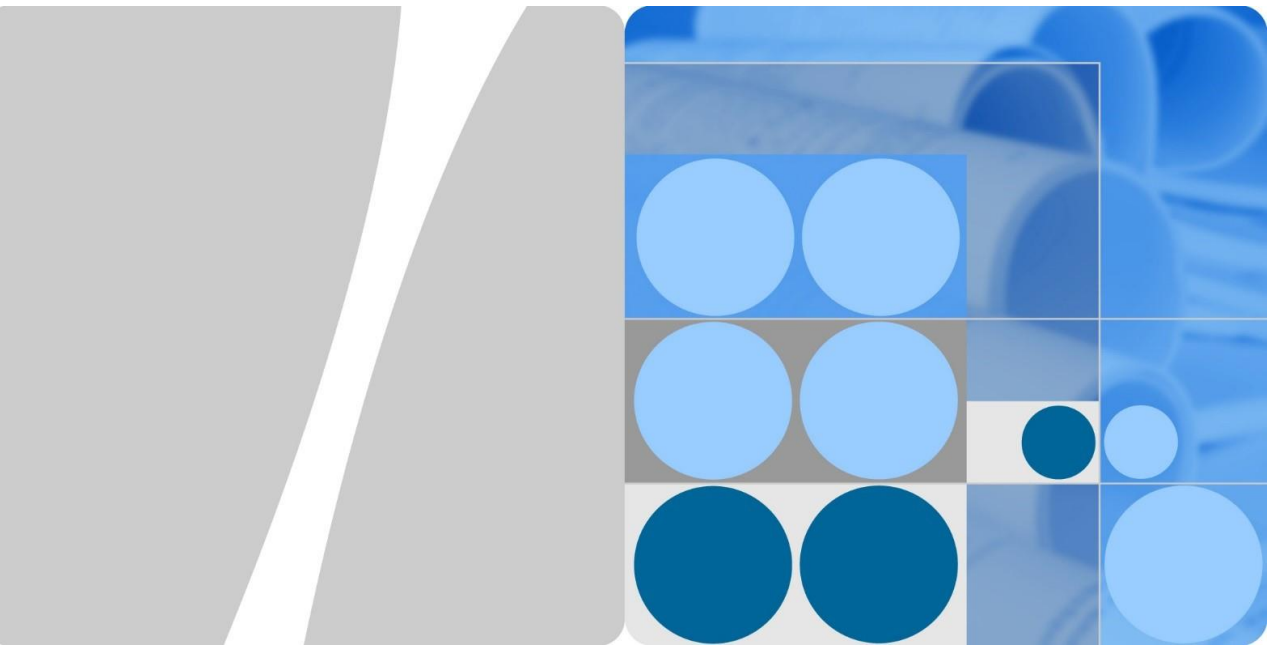


智能家居 HiLink SDK

适配开放集成调测指导

文档版本 01

发布日期 2020-08-31

华为技术有限公司

HUAWEI

商标声明

 和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼

邮编： 518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

目录

1 概述

HiLink SDK 是运行在设备 WiFi 模组上的程序模块，用于实现设备的联网，以及设备与智能家居云和智能家居 App 的互联互通。HiLink SDK 调用 WiFi 模组底层的网络和操作系统等相关 API，在上层为设备提供了统一的业务逻辑功能，包括设备联网、互联互通、设备升级等。

本文档用于指导开发者在智能设备中适配、集成和调测 HiLink SDK，实现和验证 HiLink 设备的远程控制、设备状态上报和 OTA 升级等功能。

由于不同 WiFi 模组底层的网络和操作系统等 API 存在差异，因此模组在集成 HiLink SDK 前，需要进行适配。HiLink SDK 提供了一组适配 API，用于屏蔽不同模组底层实现差异，由开发者进行具体的适配实现，具体包括四个方面的适配，如下图所示。

<table>
<tr><td colspan="4">设备主入口程序</td></tr>
<tr><td colspan="4">HiLink SDK业务逻辑</td></tr>
<tr><td>操作系统适配</td><td>网络适配</td><td>配置信息适配</td><td>OTA适配</td></tr>
</table>

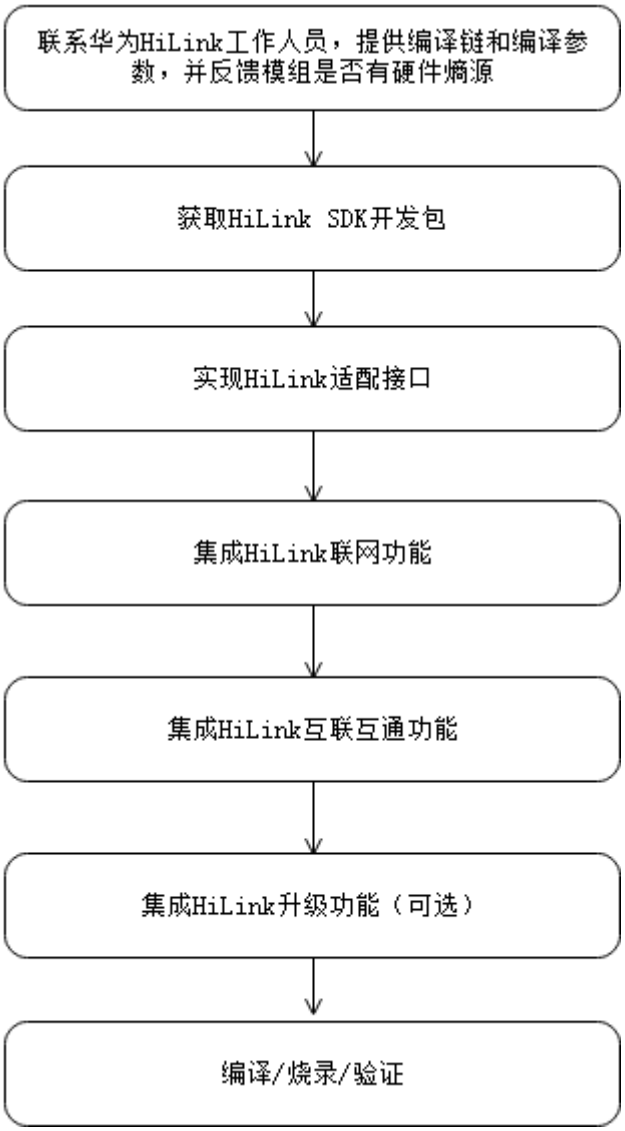
1) 操作系统适配：用于封装不同操作系统的内存、字符串、Socket、线程等接口，向 HiLink SDK 业务层提供统一接口；

2) 网络适配：用于封装不同模组 SDK WiFi 驱动 API，向业务层提供统一接口；

3) 配置信息适配：HiLink SDK 运行时会产生配置信息，需要对配置信息进行读写。配置信息适配层用于封装不同模组在读写配置信息的差异，向业务层提供统一接口；

4) OTA 适配：模组固件升级时，需要通过 HiLink SDK 下载固件并写入模组。OTA 适配层用于封装不同模组在读写固件的差异，向业务层提供统一接口。

开发者适配完后，就可以集成 HiLink SDK，使用 HiLink SDK 提供的业务逻辑接口。适配和集成的区别在于，适配的接口偏底层，目的在于 HiLink SDK 屏蔽不同模组底层的差异；集成的接口偏上层业务逻辑，使得不同模组可以使用 HiLink SDK 提供的业务逻辑接口。开发者的具体工作流程如下：



2 开发包结构介绍

HiLink 根据开发者提供的编译链和模组是否有硬件熵源生成设备的 HiLink SDK 开发包，其结构及文件说明如下表

目录	文件名	说明
doc	智能家居 HiLink SDK 适配开放集成调测指导	智能家居 HiLink SDK 适配开放集成调测指导
	智能家居HiLink基本功能测试用例	HiLink 智能家居设备自验用例集
	智能家居 HiLink SDK 适配开放集成 FAQ	HiLink SDK 集成中常见问题 FAQ
lib	libhilinkdevicesdk.a	HiLink SDK 静态库文件 Debug 版本，用于设备调测时集成 Release 版本，用于商用设备发布时集成
	libhilinkota.a	HOTA 升级特性静态库文件

		Debug 版本，用于设备调测时集成 Release 版本，用于商用设备发布时集成
|-adapter |-include	hilink_open_network_adapter.h	网络适配层接口头文件，开发者需要实现该头文件中的接口
	hilink_open_softap_adapter.h	softAp 适配层接口头文件，开发者需要实现该头文件中的接口
	hilink_open_mem_adapter.h	操作系统适配层内存接口头文件，开发者需要实现该头文件中的接口
	hilink_open_str_adapter.h	操作系统适配层字符串接口头文件，开发者需要实现该头文件中的接口
	hilink_open_sys_adapter.h	操作系统适配层基础能力抽象层接口头文件，开发者需要实现该头文件中的接口
	hilink_open_timer_adapter.h	操作系统适配层定时器接口头文件，开发者需要实现该头文件中的接口
	hilink_open_socket_adapter.h	操作系统适配层网络 Socket 接口头文件，开发者需要实现该头文件中的接口
	hilink_open_config_adapter.h	配置信息读写适配层接口头文件，开发者需要实现该头文件中的接口
	hilink_open_ota_adapter.h	OTA 升级适配层接口头文件，开发者需要实现该头文件中的接口
	hilink_sec_random.h	HiLink SDK 提供的安全函数接口，如果模组有硬件随机源，可以调用这些接口生成安全随机数。开发者可以直接调用该头文件中的接口，无需实现
adapter	hilink_open_network_adapter.c	网络适配层待实现函数
	hilink_open_softap_adapter.c	softAp 适配层待实现函数
	hilink_open_mem_adapter.c	操作系统适配层内存待实现函数
	hilink_open_str_adapter.c	操作系统适配层字符串待实现函数
	hilink_open_sys_adapter.c	操作系统适配层基础能力抽象层待实现函数
	hilink_open_timer_adapter.c	操作系统适配层定时器待实现函数
	hilink_open_socket_adapter.c	操作系统适配层网络 Socket 待实现函数
	hilink_open_config_adapter.c	配置信息读写适配层待实现函数
	hilink_open_ota_adapter.c	OTA 升级适配层待实现函数
include	hilink.h	HiLink SDK 静态库主头文件，包含 HiLinkSDK 入口函数和时间获取函 数
	hilink_device.h	产品功能适配头文件，包含： ● 设备信息、设备模型定义

		● 待实现设备服务控制、查询响应 ● 函数声明 ● 供调用设备服务状态上报函数声明
	hilink_ota.h	OTA 功能定义头文件
	hilink_netconfig_mode_mgt.h	产品配网模式选择相关函数
	hilink_log.h	HiLink SDK 系统提供的日志打印接口函数声明头文件
	hilink_typedef.h	系统类型定义头文件
	hilink_cjson.h	json 操作接口头文件，开发者可以直接调用该头文件中的接口，无需实现
	hilink_secfun.h	内存和字符串安全函数，开发者可以直接调用该头文件中的接口，无需实现
根目录	hilink_device.c	产品功能适配源文件，包含设备模型相关待实现函数
	hilink_ota.c	HiLink 设备 OTA 升级功能待实现函数

说明：开发包中的代码文件(.c 和.h)中，中文注释采用 GB2313 编码，开发者查看代码时请将对应 IDE 开发环境编码格式调整为 GBK 格式。

3 开发包使用概述

本章节介绍如何使用开发包中的文件，在正式实现适配接口前，需要进行哪些准备工作和使用步骤。具体步骤如下：

（1）准备工作 1：集成 mbedtls

HiLink SDK 内部使用 mbedtls，但不提供实现，使用开发者提供的 mbedtls 库。开发者在选择 mbedtls 时，需要保持和 HiLink SDK 内部使用的 mbedtls 库版本一致。当前 HiLink SDK 使用 mbedtls-2.16.8.

开发者在编译 mbedtls 时，需要在从官网下载的 mbedtls-2.16.8 的基础上，在 mbedtls 中的 config.h 头文件中打开附录 2 中的功能模块。然后按照 mbedtls 中的 README.md 指导书进行编译，在编译时，注意需要修改编译链和编译参数。

编译完后，将 mbedtls 的.a 库文件链接到三方工程中。

（2）准备工作 2：添加 HiLink SDK 到主程序工程中

解压 HiLink SDK 开发包到本地智能设备主程序源代码工程目录。通过修改 Makefile 文件，将 HiLink SDK 开发包中的源代码文件（*.c）、头文件（*.h）和静态库文件（*.a）添加到智能设备程序编译中。

（3）准备工作 3：启动 HiLink SDK 主程序

在设备程序中直接调用 hilink_main()函数，启动 HiLink SDK 主流程，示例如下：

```
include "hilink.h"
...
```

```
void main(void)
{
    ...
    hilink_main();
    ...
}
```

注意：由于 hilink_main()接口内部会创建线程，所以直接调用 hilink_main 接口即可，不要添加到循环或其他线程中调用。这一步需要复制开发包中的 hilink.h 头文件到其工程中。

（4）实现 HiLink SDK 适配层接口

完成上述两个步骤后，就可以开始实现适配接口。适配接口的头文件位于开发包/adapter/inlucde 目录下。开发包的 adapter 目录中提供了这些适配接口的待实现.c 文件。此时开发者应先编译固件版本，如果上述步骤正确，此时编译通过，如果编译不通过，需要先排除问题，再进行接口适配。

（5）集成 HiLink SDK

实现完 HiLink SDK 适配层接口后，就可以集成 HiLink SDK，包括集成 HiLink 设备联网功能、集成 HiLink 设备互联互通功能和集成 HiLink 设备升级功能，实现 HiLink 设备的远程控制、设备状态上报和 OTA 升级等功能。

4 HiLink SDK 适配层接口说明

本章节介绍了 HiLink SDK 适配层接口，这些接口位于开发包/adapter/include 目录中。按照功能划分，可以分为操作系统层接口适配，网络层适配，配置信息读写适配和 OTA 适配。

4.1 操作系统适配

操作系统适配主要包括内存接口、字符串接口、定时器接口、网络 socket 接口和基础能力抽象层接口适配。

4.1.1 内存接口适配

内存接口适配包括申请内存空间、释放内存空间和内存比较。

详见 hilink_open_mem_adapter.h 头文件中接口注释说明。

4.1.2 字符串接口适配

字符串接口适配包括计算字符串长度，字符查找，字符串打印和字符串比较。

详见 hilink_open_str_adapter.h 头文件中接口注释说明。

4.1.3 定时器接口适配

定时器接口适配包括设置定时器。

详见 hilink_open_timer_adapter.h 头文件中接口注释说明。

4.1.4 网络 socket 层适配

网络 socket 层适配主要包括 UDP 和 TCP 套接字的创建，数据的发送与接收，获取主机名等相关操作。适配实现时，注意 udp 和 tcp 需要设置为非阻塞模式，而且 UDP 创建后需要加入组播组。

详见 hilink_open_socket_adapter.h 头文件中接口注释说明。

4.1.5 基础能力抽象层接口适配

基础能力抽象层接口适配主要包括线程和锁和安全随机数生成的相关接口。

详见 hilink_open_sys_adapter.h 头文件中接口注释说明。安全随机数适配将在第 4.5 章节中详细说明。

4.2 网络适配

4.2.1 网络基础接口适配

不同模组底层的网络实现存在差异，HiLink SDK 抽象了一系列网络适配接口，封装了这些差异。这些接口位于 hilink_open_network_adapter.h 中，主要包括网卡信息和网络状态读写。详见头文件中接口注释说明。

4.2.2 SoftAp 适配

SoftAp 适配的接口位于 hilink_open_softap_adapter.h 中，主要包括开启和关闭 softAp 热点。

1. 开启 SofAp 热点

```
int HILINK_StartSoftAp(const char *ssid, unsigned int ssidLen)
{
    // 1.设置网卡为 SoftAp 模式
    // 2.设置 softap id，加密方式，最多支持连接数
    // 3.设置网卡 IP 信息，包括 IP，网关和子网掩码
    // 4.设置 DHCP 信息
    // 5.启动 soft ap
}
```

2. 获取广播 IP 接口 int HILINK_GetBroadcastIp(char *broadcastIp, unsigned char len)。

比如对于 IP192.168.5.2 的广播 IP 为 192.168.5.255

3. 关闭 SoftAp 热点接口 int HILINK_StopSoftAp(void)。

关闭 SoftAp 热点后，并将网卡切回 station 模式

4.3 配置信息读写适配

开发者需要适配配置信息读写接口，实现 HiLink SDK 配置数据的读写。HiLink SDK 配置数据包括 HiLink 运行配置数据，HiLink 定时器配置数据，桥设备配置参数和端到端安全 hichain 数据。

配置数据大小方面：

sdk 的配网结构体 1 片 4KB

sdk 的时间管理 1 片 4KB

sdk 的网桥功能 2 片 8KB(64 路子设备)

sdk 支持端到端功能 1 片 4KB

sdk 支持 IFTTT 功能 2 片 8KB（预留）

sdk 支持 SoC 功能 1 片 4KB（预留）

每次以 4KB 为读写单元

开发者需要实现如下接口：

1. 初始化配置保存分区接口 int HILINK_InitConfig()：用于初始化 flash 操作分区地址或者创建相关目录文件。

2. 保存 HiLink 配置信息接口。适配该接口的伪代码示例如下：

```
int HILINK_WriteConfig(enum HILINK_ConfigType configType,
const unsigned char *buf, unsigned int len)
{
    switch (configType) {
        case HILINK_RUNNING_CONFIG:
            // 保存 HiLink 运行配置数据，必须适配
            break;
        case HILINK_TIMER_CONFIG:
            // 保存 HiLink 定时任务配置数据，需要定时功能需要适配
            break;
        case HILINK_BRIDGE_CONFIG:
            // 保存 HiLink 桥设备配置数据，当前为桥设备时需要适配
            break;
        case HILINK_HKS_CONFIG:
```

```
                // 端到端安全 hichain 组件需要存储数据，需要端到端功能时适配
                break;
            default:
                return -1;
        }
        return 0;
    }
```

3 读取 HiLink 配置数据接口 int HILINK_ReadConfig(enum HILINK_ConfigType configType, unsigned char *buf, unsigned int len)，和保存 HiLink 配置数据类似，参考上述示例。

4.4 OTA 适配

模组使用 HiLink SDK 提供的 OTA 功能进行升级，然后 HiLink SDK 调用 OTA 适配接口将升级固件写入模组中。如何将升级固件写入模组，不同模组存在差异。因此 HiLink SDK 抽象了 OTA 适配接口，由不同模组提供具体实现。开发者需要提供两个升级固件，分别对应模组存储设备上的两个分区。

这些接口位于 hilink_open_ota_adapter.h 中，具体有如下接口：

1. 初始化升级固件保存分区接口 bool HILINK_OtaAdapterFlashInit(void)：对设备 flash 分区等进行初始化，确定下载的位置和长度等信息；
2. 判断需要升级的分区接口 unsigned int HILINK_OtaAdapterGetUpdateIndex(void)：返回值是 UPGRADE_FW_BIN1 时，表示升级固件到分区 1，返回值是 UPGRADE_FW_BIN2 时，表示升级固件到分区 2；
3. 擦除需要升级的分区接口 int HILINK_OtaAdapterFlashErase(unsigned int size)：擦除 Flash 分区或者关闭相关文件，size 表示擦除分区的大小。擦除 flash 时注意 block 对齐，只能一个 block 全部擦除；
4. 升级数据写入升级的分区接口 int HILINK_OtaAdapterFlashWrite(const unsigned char *buf, unsigned int bufLen)：将升级固件内容 buf 写入升级的分区
5. 读取升级分区数据接口 int HILINK_OtaAdapterFlashRead(unsigned int offset, unsigned char *buf, unsigned int bufLen)：读取升级分区的数据
6. 判断是否正常分区升级结束接口 bool HILINK_OtaAdapterFlashFinish(void)：如果正常结束，则返回 true，如果异常结束，则返回 false
7. 获取升级区间最大长度接口 unsigned int HILINK_OtaAdapterFlashMaxSize(void)
8. 重启模组接口 void HILINK_OtaAdapterRestart(int flag)：如果 flag 是 RESTART_FLAG_NOW 时，表示只有 MCU 升级时立即重启；如果 flag 是 RESTART_FLAG_LATER 时，表示有模组时切换分区后再重启。

本节可以配合 5.3 节华为 HOTA 升级功能集成进行理解。HOTA 升级功能集成的接口位于 hilink_ota.h 中。HOTA 升级功能集成与本节的 OTA 适配的区别在于：前者偏向上层 OTA 业务逻辑，后者偏向底层 OTA 固件读写。上述接口的具体调用逻辑如下：

1. 设备检测到新版本，用户点击升级，设置执行升级命令，创建一个升级线程启动升级

2. 设备先向 App 上报一个 1%的升级进度，表示升级已经启动
3. 调用 HilinkOtaStartProcess 函数通知用户升级已启动
4. 调用 HILINK_OtaAdapterFlashInit 对设备 flash 分区等进行初始化，确定下载的位置和长度等信息
5. 调用 HILINK_OtaAdapterGetUpdateIndex 确定是下载固件到分区 1 还是分区 2
6. 下载 filelist.json 文件
7. 下载 filelist.json.asc 文件（filelist.json 的签名）
8. 校验 filelist.json 文件的签名
9. 开始下载升级固件 image2_all_ota1.bin 或者 image2_all_ota2.bin
10. 调用 HILINK_OtaAdapterFlashMaxSize 函数获取升级区间最大长度，判断下载的固件是否大于下载分区的大小
11. 调用 HILINK_OtaAdapterFlashErase 擦除下载分区
12. 调用 HILINK_OtaAdapterFlashWrite 函数将下载的固件写入到 flash 里，每次写入不超过 512 字节
13. 校验写入 flash 的数据的完整性，调用 HILINK_OtaAdapterFlashRead 将写入的内容读出进行完整性校验
14. 如果完整性校验成功，调用 HILINK_OtaAdapterFlashFinish 结束升级模式，进行分区切换等动作
15. 调用 HilinkOtaEndProcess 函数通知升级结束
16. 上报升级结果给 OTA 服务器和云端
17. 调用 HILINK_OtaAdapterRestart 重启模组

4.5 安全随机数适配

HiLink SDK 在加密和通信时需要使用安全随机数。根据模组是否有硬件熵源，在适配安全随机数时，分为如下两种情况：

a) 模组有硬件熵源

如果模组有硬件熵源，需要进行下面几个步骤处理：

1) 首先调用接口 HILINK_RegisterRandomEntropy 注册熵源

```
int HILINK_RegisterRandomEntropy(RandEntropy randEntropy);
```

其中 RandEntropy 接口原型为

```
typedef int (*RandEntropy)(unsigned int *entropyValue);
```

该接口由设备厂商实现，返回 32 位的熵源。

返回的熵源需要通过熵源测试，测试方法见附录 1。如果熵源通过不了测试，开发者需优化熵源，如通过增加熵产生源的随机参数或者更换熵产生源等方式，直到通过熵源测试。

2) 在注册熵源成功后，设备可通过华为提供的 HILINK_SecRandom 来生成安全随机数。

```
int HILINK_SecRandom(unsigned int *securityRandom);
```

b) 模组没有硬件熵源

如果模组没有硬件熵源，则开发者自行生成随机数（要保证随机数要尽量随机），同时向华为 HiLink 工作人员报备。

5 HiLink SDK 集成

5.1 联网功能集成

联网功能是指 HiLink 智慧生活 App 扫描添加设备、绑定设备到用户账号的功能。设备可以通过集成 HiLink SDK 的联网功能，实现连接路由网络、注册及登录智能家居云、添加到智慧生活 App。

5.1.1 修改设备信息

本小节介绍开发者需要根据具体产品完成的相关信息的修改和配置。

（1）版本信息

在 hilinksdk/include/hilink_device.h 文件中，默认版本号均设置为“1.0.0”，开发者可根据实际情况修改设备的固件版本、软件版本、硬件版本。例如：

```
#define FIRMWARE_VER "1.1.1"
#define SOFTWARE_VER "2.2.2"
#define HARDWARE_VER "3.3.3"
```

（2）厂商及设备名称修改

hilinksdk/include/hilink_device.h 文件中有厂商和设备的名称定义的宏，例如：

```
#define DEVICE_TYPE_NAME "Switch" // 设备类型名称
#define MANUAFACTURER_NAME "Huawei" // 设备厂商名称
```

如果两个名称字符串的长度之和超过了 17 字符，SDK 内部会进行截断，导致显示不全。建议开发者手动对名称修改，改用简短单词或单词缩写，确保不超过 17 个字符。

（3）设备 SN 录入(可选)

设备默认使用 MAC 地址作为 SN 号，如果开发者需要根据实际情况录入 SN，可以实现 hilinksdk/hilink_device.c 中的 HilinkGetDeviceSn 接口，将 SN 数据传递给 HiLink SDK。示例如下

```
void HilinkGetDeviceSn(unsigned int len, char *sn)
{
    char str[] = "This is SN"; // (调试用)
    memcpy(sn, str, strlen(str)); // (调试用)
}
```

注意：SN 限制的最大长度是 39 字节。

（4）设备 PIN 录入(可选)

HiLink SDK 提供了 PIN 码配网功能，开发者可以实现 hilinksdk/hilink_device.c 中的如下接口，将 PIN 码传递给 HiLink SDK。

```
int HiLinkGetPinCode(void)
{
    return 12345678; // (调试用)
}
```

注意：返回值是 8 位数字；若返回-1 则使用 HiLink SDK 的默认 PIN 码。

（5）设备子型号适配(可选)

如果设备定义有子型号，则 HiLink SDK 在组装 SoftAp 的 SSID 阶段、向云端注册阶段、登录录后设备信息同步阶段，三个阶段均会使用设备子型号，开发者需要适配 adapter\profile_adapter\hilink_profile_adapter.c 中的如下接口，将设备的子型号 ID 传递给 HiLink SDK。

```
int HILINK_GetSubProdId(char *subProdId, int len);
```

参数说明:

(1) char *subProdId: 保存子型号的缓冲区;

(2) int len: 缓冲区的长度;

如果设备定义有子型号，则子型号长度固定为 2 字节，取值为十六进制的字符串表示，范围 00 ~FF，其中有字母则字母大写，并以’\0’结束，函数返回 0。

如果未定义有设备子型号，函数返回-1。

示例如下：

```
int HILINK_GetSubProdId(char *subProdId, int len)
{
    strcpy(subProdId, "0A");
    return 0;
}
```

5.1.2 设置 HiLink SDK 属性（可选）

参考 include\hilink.h 中结构体 HILINK_SdkAttr 的描述，使用 HILINK_SetSdkAttrs 接口设置 HiLink SDK 的属性；使用 HILINK_GetSdkAttr 接口查询当前 HiLink SDK 的属性。各个属性值的说明如下。

1) monitorTaskStackSize：HiLink SDK 监控任务栈大小，开发者根据具体情况调整，默认为 1024 字节，在异常监控任务中会调用重启前处理接口 hilink_process_before_restart、软

重启接口 rebootSoftware 和硬重启接口 rebootHardware，若在以上接口的实现中使用了较大的栈空间，调整此接口；

2) deviceMainTaskStackSize：普通设备形态，HiLink SDK 运行主任务栈大小，开发者根据具体情况调整，如果当前设备形态是普通设备，可以使用此接口调整栈大小；

3) bridgeMainTaskStackSize：网桥设备形态，HiLink SDK 运行主任务栈大小，开发者根据具体情况调整，如果当前设备形态是网桥设备，可以使用此接口调整栈大小；

4) otaCheckTaskStackSize：HiLink OTA 检查升级版本任务栈大小，开发者根据具体情况调整，如果使用 HiLink 提供的 OTA 升级，可以使用此接口调整栈大小；

5) otaUpdateTaskStackSize：HiLink OTA 升级任务栈大小，开发者根据具体情况调整，如果使用 HiLink 提供的 OTA 升级，可以使用此接口调整栈大小；

6) rebootSoftware：HiLink SDK 异常、OTA 升级、设备删除等场景软重启接口，不影响设备硬件状态，如果用户注册此接口，首先使用；

7) rebootHardware：HiLink SDK 异常、OTA 升级、设备删除等场景硬重启接口，影响硬件状态，如果开发者未注册软重启接口，使用此接口重启。

示例：如果要调整 HiLink SDK 监控任务栈大小，由默认 1024 字节改为 2048 字节，该如何处理？

(1) 在调整该栈大小前先调用 HILINK_SdkAttr *HILINK_GetSdkAttr(void)函数接口获取当前各任务栈的大小；

(2) 调用 int HILINK_SetSdkAttr(HILINK_SdkAttr sdkAttr)函数接口来设置要修改的任务栈的大小。

注：调用(1) (2)中两个函数都必须在 hilink_main()函数之前，否则不会生效。

```
void main(void)
{
    HILINK_SdkAttr *sdkAttr = NULL;

    /* 获取 HiLink SDK 当前属性值 */
    sdkAttr = HILINK_GetSdkAttr();
    if (sdkAttr == NULL) {
        printf("sdk attr is null\r\n");
        return;
    }

    /* 设置新的属性值 */
    sdkAttr->monitorTaskStackSize = 2048;
    HILINK_SetSdkAttr(*sdkAttr);

    /* 启动 HiLink 任务 */
    hilink_main();
```

```
        return;
}
```

5.1.3 设置配网方式（可选）

开发者可以设置设备要使用的配网方式(4G 或网线连网、WiFi 配网或其他配网)。开发者可以参考 include\hilink_netconfig_mode_mgt.h 中 enum HILINK_NetConfigMode 的描述，使用 HILINK_SetNetConfigMode 接口通知 HiLink SDK 设备使用的配网方式。若不设置，HiLink SDK 将默认使用 WiFi 配网方式。

5.1.4 启动 HiLink SDK 主程序

在设备程序中直接调用 hilink_main()函数，启动 HiLink SDK 主流程，示例如下：

```
include "hilink.h"
...
void main(void)
{
        ...
        hilink_main();
        ...
}
```

注意：

(1) 由于 hilink_main()接口内部会创建线程，所以直接调用 hilink_main 接口即可，不要添加到循环或其他线程中调用。

(2) 开发者完成联网功能集成后，先不要继续下面的互联互通和升级功能集成，也不要添加设备业务功能的实现。开发者先编译固件版本，烧录验证设备联网功能（App 添加设备）是否正常。待验证联网功能正常后，再继续完整后续集成和业务功能开发。

5.2 互联互通功能集成

互联互通功能是 HiLink SDK 提供的设备与智慧生活 App 之间的命令交互、状态同步的功能。功能包括：智能设备响应华为智慧生活 App 的服务状态控制命令；智能设备上报服务状态给华为智慧生活 App；智能设备处理华为智慧生活 App 的服务状态查询命令。

集成互联互通功能，请参考如下步骤：

5.2.1 实现设备服务状态控制功能

开发者需要根据产品功能定义，在 hilinksdk/hilink_device.c 中实现 hilink_put_char_state 函数。

5.2.2 实现设备服务状态查询功能

开发者需要根据产品功能定义，在 hilinksdk/hilink_device.c 中实现 hilink_get_char_state 函数。

5.2.3 实现设备服务状态上报功能

开发者需要根据产品功能定义，在 hilinksdk/hilink_device.c 中添加实现某些服务的状态上报接口。在设备状态主动发生变化，智慧生活 App 控制下发后设备状态没有立刻改变、过段时间完成更改后主动上报设备的状态。请开发者调用 include\hilink.h 中的 hilink_report_char_state 接口实现。

5.2.4 实现设备重启预处理功能

在 hilinksdk/hilink_device.c 中定义了 int hilink_process_before_restart(int flasg)接口提供开发者实现。HiLink SDK 在异常、OTA 升级、设备删除等情况下，会请求重启 WiFi 模组，重启前会调用本接口。
开发者可以实现本接口完成模组重启前的一些必要操作，如数据备份等，以确保模组重启后的运行状态与重启前一致。其中返回值可以参考如下描述实现：
函数返回 0 表示处理成功，系统可以重启，使用硬重启接口重启;
函数返回 1 表示处理成功，系统可以重启，使用软重启（软重启接口需要开发者提前适配并注册）;
函数返回负值表示处理失败，系统不能重启，HiLink SDK 将继续等待。
这里的参数 flag 表示了不同的重启原因：
当 flag 为 0 时，表示 HiLink SDK 线程看门狗触发导致即将重启，此时函数返回值由开发者根据具体业务状态决定；
当 flag 为 1 时，表示 App 删除设备即将重启，此时函数返回值必须为 0 或 1(允许重启)，否则将导致删除设备功能异常。
注意：请将该接口实现为非阻塞函数，建议该接口在 1 秒内返回，避免阻塞 SDK 主流程。

5.2.5 获取设备在线状态（可选）

开发者可以通过接口获取设备当前状态，当前支持状态列表如下：

```
/* 设备与云端连接断开(版本向前兼容) */
#define HILINK_M2M_CLOUD_OFFLINE 0
/* 设备连接云端成功，处于正常工作态(版本向前兼容) */
#define HILINK_M2M_CLOUD_ONLINE 1
/* 设备与云端连接长时间断开(版本向前兼容) */
#define HILINK_M2M_LONG_OFFLINE 2
/* 设备与云端连接长时间断开后进行重启(版本向前兼容) */
#define HILINK_M2M_LONG_OFFLINE_REBOOT 3
/* HiLink 线程未启动 */
#define HILINK_UNINITIALIZED 4
/* 设备处于配网模式 */
#define HILINK_LINK_UNDER_AUTO_CONFIG 5
/* 设备处于 10 分钟超时状态 */
#define HILINK_LINK_CONFIG_TIMEOUT 6
/* 设备正在连接路由器 */
#define HILINK_LINK_CONNECTTING_WIFI 7
/* 设备已经连上路由器 */
#define HILINK_LINK_CONNECTED_WIFI 8
/* 设备正在连接云端 */
#define HILINK_M2M_CONNECTTING_CLOUD 9
```

```
/* 设备与路由器的连接断开 */
#define HILINK_M2M_CLOUD_DISCONNECT 10
/* 设备被注册 */
#define HILINK_DEVICE_REGISTERED 11
/* 设备被解绑 */
#define HILINK_DEVICE_UNREGISTER 12
```

1. 查询接口

hilinksdk/include/hilink.h 中声明了 int hilink_get_devstatus(void)函数，函数返回设备当前状态，值对应上面不同的状态。开发者可以调用本接口查询当前设备状态。

```
int devStatus = hilink_get_devstatus();
```

2. 状态变化回调函数

hilinksdk/hilink_device.c 中定义了虚函数 hilink_notify_devstatus(int status)。HiLink SDK 内部在设备状态变化时会调用本接口将状态通知给开发者。开发者可以实现这个接口，根据参数 status 的值，添加相应状态下的业务处理。

```
void hilink_notify_devstatus(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
            break;
        case HILINK_M2M_CLOUD_ONLINE:
            // 设备连接云端成功，请在此处添加实现
            break;
        …
        default:
            break;
    }
}
```

5.2.6 实现恢复出厂设置（可选）

用户手动操作设备(如按键、长按、组合操作等)进行恢复出厂设备，此时需要清除设备 Flash 存储的配网信息及华为智能家居云记录的设备绑定信息，使设备重新进入配网状态。

开发者实现上述功能，可以在手动操作触发恢复出厂的接口中，调用如下接口实现。

```
hilink_restore_factory_settings();
```

本接口声明在 hilinksdk/include/hilink.h 文件中。

5.2.7 存取设备状态配置（可选）

HiLink SDK 提供了接口供开发者将一些简单状态信息保存到 Flash，Flash 空间大小 32 字节。对没有电控板 MCU 的设备(业务功能实现在模组上)，可调用下面的接口保存或获取 Flash 中的设备状态。

需要注意：

1)App 删除设备或设备手动恢复出厂后，保存的状态会被清除。

2)接口仅提供数据存储 Flash 的能力，不对数据进行加密。若用户保存重要信息请自行加密后再调用本接口保存。

存取接口均声明在 hilinksdk/include/hilink.h 文件中。

（1）保存设备状态

调用以下函数保存设备状态配置到模组 Flash，最长 32 字节。注意：每次保存都会覆盖之前旧的内容。

```
int HilinkSetUserConfig(const char* config, unsigned short len);
```

使用示例：

```
char inConfig[32] = { /* 用户配置或状态 */ };
int ret = HilinkSetUserConfig(inConfig, 32);
if (ret == 0) {
    /* 保存成功 */
} else {
    /* 保存失败 */
}
```

（2）获取用户设备状态

调用以下函数获取之前保存的设备状态配置，最长 32 字节。

```
int HilinkGetUserConfig(unsigned short len, char* config);
```

参数 config 是出参，调用者需要预先分配好内存，获取到的设备状态配置会被保存到 config 对应的内存中。使用示例：

```
char outConfig[32] = {0};
int ret = HilinkGetUserConfig(32, outConfig);
if (ret == 0) {
    /* 获取成功，信息被获取到 outConfig 中 */
} else {
    /* 获取失败 */
}
```

5.2.8 设备离线场景在 App 删除设备 SDK 处理适配（可选）

设备离线时，如果在 App 上删除了该设备，设备下次上线后云端会给设备下发 Errcode=5(登录错误)或 Errcode=6(设备已被删除)错误码。此时设备接收到这两个错误码后，会根据接口 HILINK_EnableProcessDelErrCode (enable)的设置结果来判断是否需要清除注册信息进入配网。

enable 为 0 表示 SDK 不处理云端下发的 Errcode=5 或 Errcode=6 错误码，此时 SDK 不会清除设备端注册信息，需要用户手动硬件恢复出厂设置，设备才能重新进入配网状态。

enable 为非 0 表示 SDK 处理云端下发的 Errcode=5 或 Errcode=6 错误码，此时 SDK 会清除设备端注册信息，并重新进入配网状态。

如果不设置，默认 enable 为 0。

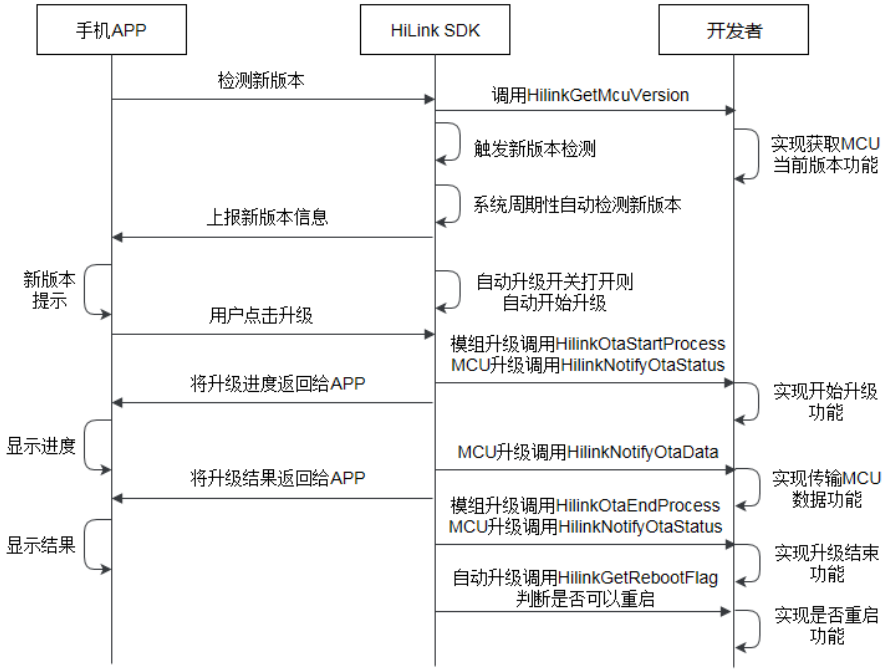
对于配网跟注册一体的设备无需设置此接口。

对于配网跟注册分离的设备，如果具备单独的本地清除注册的功能，无需设置此接口。

对于配网跟注册分离的设备，如果不具备单独的本地清除注册的功能，需要设置 enable 为非 0。

5.3 华为 HOTA 升级功能集成（可选）

本小节介绍使用华为 HOTA 升级功能的模组，适配升级功能的实现。升级分为模组升级和电控板 MCU 升级。HiLink 设备的升级通过华为智能家居 App 触发或者用户打开自动升级功能自动触发，由 HiLink SDK 实现模组和 MCU 升级业务，开发者只需要实现几个接口，整体流程如下图所示：



5.3.1 实现升级接口函数

实现 hilinksdk/hilink_ota.c 中升级接口函数：

1.获取 MCU 当前版本信息函数

实现获取 MCU 当前版本号 HilinkGetMcuVersion (char *version, unsigned int inLen, unsigned int *outLen)，开发者实现获取 MCU 的当前版本号。

如果获取不到 MCU 的版本，则不对 MCU 进行升级。建议开发者在 MCU 正常启动后，或升级启动后，就将 MCU 的版本号传递给模组，确保模组可以获取到 MCU 的版本。如果没有 MCU 的返回 RETURN_ERROR_NO_MCU，获取成功则返回 RETURN_OK，获取失败则返回 RETURN_ERROR。

2.模组开始升级函数

实现模组开始升级函数 HilinkOtaStartProcess (int type)，开发者可在此函数中实现升级开始时需要添加的功能。

在手动升级场景下，HiLink SDK 在接收到用户发出的升级指令后，将直接调用此接口；在自动升级场景下，当 HiLink SDK 在调用 HilinkGetRebootFlag 接口返回值是 MODULE_CAN_REBOOT 时，HiLink SDK 将调用此接口。厂商可在此接口中完成和升级流程相关的处理。自动升级流程在凌晨进行，因此厂商在实现升级流程相关功能时，确保

在升级的下载安装固件和重启设备时避免对用户产生影响，比如发出声音、光亮等。如果处理正常就返回 RETURN_OK，处理异常请返回 RETURN_ERROR。

3.模组结束升级函数

实现模组结束升级函数 HilinkOtaEndProcess (int status)，开发者可在此函数中实现在升级结束时需要添加的功能。

HiLink SDK 在将固件写入到 OTA 升级区后，且完整性校验通过后，将调用厂商适配的此接口；厂商可在此接口中完成和升级流程相关的处理。自动升级流程在凌晨进行，因此厂商在实现升级流程相关功能时，确保在升级的下载安装固件和重启设备时避免对用户产生影响，比如发出声音、光亮等；升级类型是否为自动升级可参考接口 HilinkOtaStartProcess 的参数 type 的描述。如果处理正常就返回 RETURN_OK，处理异常请返回 RETURN_ERROR。

4.通知 MCU 升级状态函数

实现通知 MCU 升级状态函数 HilinkNotifyOtaStatus (int flag, unsigned int len, unsigned int type)，开发者可在此函数中实现 MCU 升级状态改变时需要添加的功能。没有 MCU 可不用实现此函数。

HiLink SDK 调用开发者适配的此接口通知 MCU 固件传输的状态。当 flag 是 STOP_SEND_DATA 时，此接口需返回 MCU 侧固件升级的结果；当 flag 是其它值时，需返回接口接收到此消息后的处理结果。自动升级流程在凌晨进行，因此厂商在实现升级流程相关功能时，确保在升级的下载安装固件和重启设备时避免对用户产生影响，比如发出声音、光亮等。

5.MCU 数据传输函数

实现 MCU 数据传输函数 HilinkNotifyOtaData(unsigned char *data, unsigned int len, unsigned int offset)，开发者可在此函数中实现传输 MCU 固件数据给 MCU 的功能。没有 MCU 可不用实现此函数。

HiLink SDK 调用开发者适配的此接口通知开发者发送 MCU 固件文件数据。HiLink SDK 通知发送 MCU 固件文件时，将 MCU 固件文件拆分成若干个数据包通知给模组。开发者适配的此接口需要返回 MCU 接收这部分数据的处理结果。当此接口返回 RETURN_OK 时，HiLink SDK 继续正常处理后续流程；当此接口返回 RETURN_ERROR 时，HiLink SDK 将终止此次的 MCU 固件升级。如果处理正常就返回 RETURN_OK，处理异常请返回 RETURN_ERROR。

6.判断模组是否能重启函数

实现判断模组是否能重启函数 HilinkGetRebootFlag(void)，开发者可在此函数中实现重启前保存数据之类的功能。

在用户同意设备可以自动升级的情况下，HiLink SDK 调用此接口获取设备当前业务状态下，模组是否可以立即升级并重启的标志。只有当设备处于业务空闲状态时，接口才可以返回 MODULE_CAN_REBOOT。当设备处于业务非空闲状态时，接口返回 MODULE_CANNOT_REBOOT。

5.3.2 部署华为 HOTA 服务器的升级包格式

开发者需提供模组升级固件，联系华为相关支撑人员进行版本的 HOTA 服务器部署。

5.4 DHCP Option 60 功能实现

DHCP Option 60 用于智能设备向 DHCP 服务器上报自身厂商以及配置信息，设备开发者需要自行实现 DHCP Option 60 上报功能。HiLink 要求在 DHCP 的 discover 报文中上报 DHCP Option60 信息，上报格式为：“厂商标识”:“终端类型”:“终端型号”。

厂商标识”、“终端类型”、“终端型号”三个属性可以通过 hilink_device.h 文件中的“MANUAFACTURER_NAME”、“DEVICE_TYPE_NAME”、“DEVICE_MODEL”三个宏定义获取。

6 功能验证

6.1 概述

根据集成开发包中的认证测试用例《HiLink 智能家居解决方案 SDK 基本功能测试用例》验证智能设备基本功能。

根据产品定义的功能，验证业务功能。使用配套的华为智慧生活 App（实现该设备对应的添加、注册、控制功能）与设备共同验证。

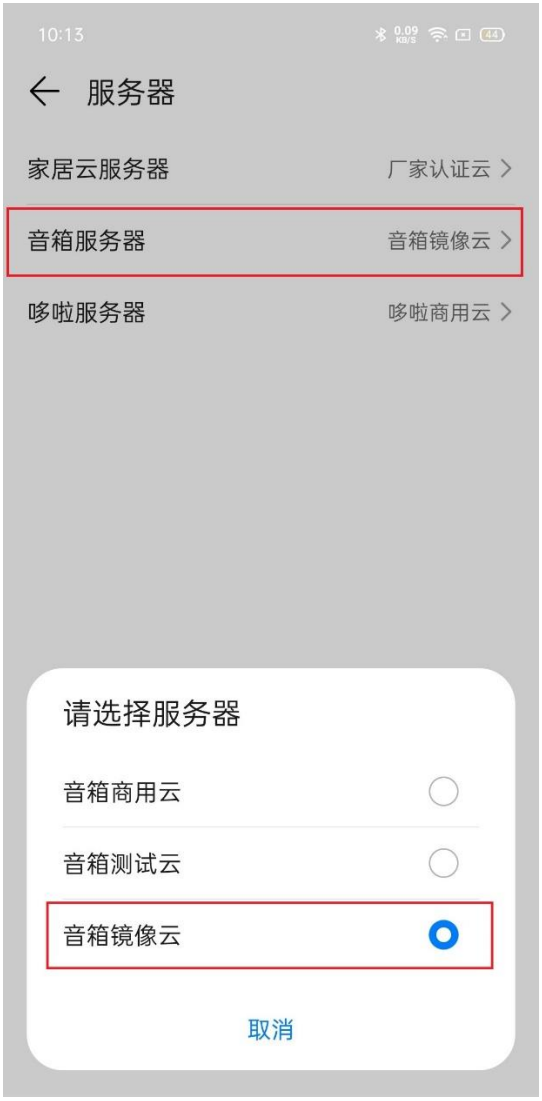
特别注意，请关闭手机应用市场中华为智慧生活 App 的自动升级功能。该 App 为测试专用 App，请勿升级至其他版本，否则 App 无法进行正常调试工作。

6.2 App 调试环境设置

1.运行华为智慧生活 App，可以直接选择商用环境、认证环境和开发测试环境。

1.1 可根据自己需求自定义环境类型，选择自定义，选择家居云服务器，选择“厂家认证云”。

1.2 选择音箱云服务器，任选一个即可，点击“确认”按钮。



2. 阅读相关用户协议和隐私声明，勾选加入用户体验计划（也可后面在 App 中打开），点击“同意”按钮。

3. 允许 App 获取的权限。

4.登录华为账号（若之前手机已登录过，App 会自动登录），观看使用引导。

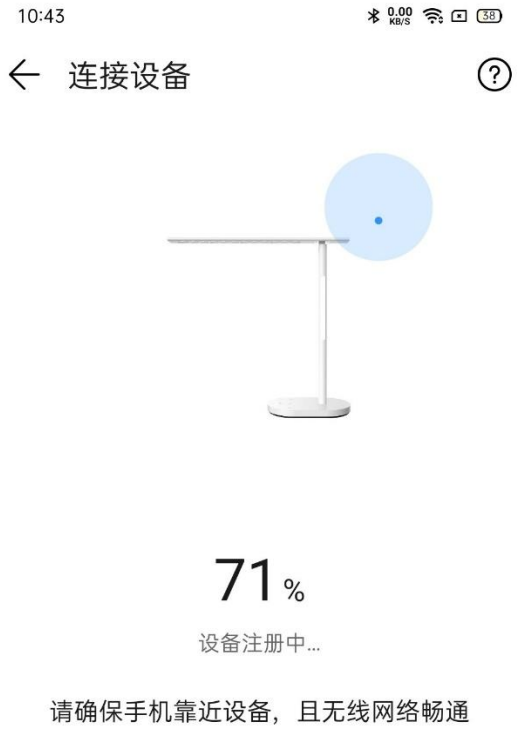
6.3 搜索添加待测设备

1. 在智慧生活 App“家居”页面，点击右上角的“+”号按钮，选择“添加设备”。

2. 华为智慧生活 App 开始自动扫描周围设备。

3. 选择扫描出的对应设备，点击“连接”按钮，进入配网流程。

4. 在“连接设备”页面输入手机当前连接的 WLAN 热点的密码，单击“下一步”，设备开始自动连网注册。



说明

1、如果设备实现了定制 PIN 码配网，还需要手动输入 PIN 码。

5. 在“设备设置”页面，设置设备名称及选择设备所属房间。如果设备需要使用自动升级功能，可以打开自动升级开关。点击“完成”按钮，设备即添加成功。

6. 设备添加完成后，可在华为智慧生活 App“家居”页面查看到已添加设备。点击设备图标进入设备页面，即可完成设备功能的控制等操作。

6.4 验证设备控制功能

1. 进入设备页面，操作相关控件，控制设备功能。
2. 查看设备状态是否按预期改变，以及 App 状态与设备侧状态是否一致。

具体页面和操作，与具体设备类型和功能相关，此处不再赘述。

7 附录 1：熵源符合度测试方法

7.1 环境准备

a) Linux 机器一台

b) 安装好 GCC 编译器

c) 安装好 bzlib 和 divsufsort 库。

7.2 软件安装

获取软件: 从 GitHub 获取软件 SP800-90B_EntropyAssessment-master

下载成功后将获取到名称为 SP800-90B_EntropyAssessment-master.zip 的文件。

使用 ftp 将文件上传到准备好的 linux 机器上，并在文件所在目录执行如下命令：

```
unzip SP800-90B_EntropyAssessment-master.zip
chmod -R 750 SP800-90B_EntropyAssessment-master
cd SP800-90B_EntropyAssessment-master/cpp
make
```

7.3 测试步骤

a) 在设备上采集熵源的二进制数据保存至二进制文件中，记录设备每次产生熵源的比特位数作为文件名后缀。为保证测试准确性，要求熵源数据的比特数不大于 8（超过 8 位只采集 8 位），采集设备多次重启后的数据，重启次数不小于 1000 次，采集数据总量不小于 1000 000 字节（多次重启采集的文件要求不能相同）。二进制文件命名可参考: <filename_xbit>.bin 其中 filename 为二进制文件名，xbit 表示每次产生熵源的比特位数。

b) 执行如下命令测试熵源数据的最小熵:

./ea_non_iid <binary_file> <bits_per_word> <-i|-c> <-a|-t> [-v]

参数说明：

binary_file 表示采集到的待测试熵源二进制文件

bits_per_word 表示采集熵源的 bit 位数，选择为 1-8

-i 测试熵源数据没有经过加工处理，为初始的熵值（默认）

-c 熵源数据已经经过调节处理

-a 计算二进制文件中所有数据的熵（默认）

-t 只计算前 100 万位数据的熵

-v 可选参数，用于打印测试过程中的详细过程

计算说明：最小熵记录了在产生的一组熵源中的有效数据比率，如: 产生的熵源长度为 4，最小熵大小为 0.1，则一组熵源数据中只有 4*0.1=0.4 的有效熵源数据，需要产生 10 组熵源数据，才能产生一个 4 位的随机数符合要求的随机数。

例如：

./ea_non_iid ../bin/testrand_8bit.bin 8 -i -a –v

执行返回数据

…

min(H_original, 8 X H_bitstring): 0.000298

其中红色数字为返回的最小熵的值，当该值大于 0.1 时认为是测试通过，否则认为不通过。在上面例子中因为 0.000298 < 0.1 所以测试不通过。

8 附录 2：mbedtls 需要开启的模块

```
#define _CRT_SECURE_NO_DEPRECATE 1
#define MBEDTLS_HAVE_TIME
#define MBEDTLS_HAVE_TIME_DATE
#define MBEDTLS_PLATFORM_MEMORY
#define MBEDTLS_PLATFORM_NO_STD_FUNCTIONS
#define MBEDTLS_PLATFORM_TIME_ALT
#define MBEDTLS_AES_ROM_TABLES
#define MBEDTLS_CIPHER_MODE_CBC
#define MBEDTLS_CIPHER_MODE_CFB
#define MBEDTLS_CIPHER_MODE_CTR
#define MBEDTLS_CIPHER_MODE_OFB
#define MBEDTLS_CIPHER_MODE_XTS
#define MBEDTLS_CIPHER_PADDING_PKCS7
#define MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS
#define MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN
#define MBEDTLS_CIPHER_PADDING_ZEROS
#define MBEDTLS_REMOVE_3DES_CIPHERSUITES
#define MBEDTLS_ECP_DP_SECP192R1_ENABLED
#define MBEDTLS_ECP_DP_SECP224R1_ENABLED
#define MBEDTLS_ECP_DP_SECP256R1_ENABLED
#define MBEDTLS_ECP_DP_SECP384R1_ENABLED
```

```
#define MBEDTLS_ECP_DP_SECP521R1_ENABLED
#define MBEDTLS_ECP_DP_SECP192K1_ENABLED
#define MBEDTLS_ECP_DP_SECP224K1_ENABLED
#define MBEDTLS_ECP_DP_SECP256K1_ENABLED
#define MBEDTLS_ECP_DP_BP256R1_ENABLED
#define MBEDTLS_ECP_DP_BP384R1_ENABLED
#define MBEDTLS_ECP_DP_BP512R1_ENABLED
#define MBEDTLS_ECP_DP_CURVE25519_ENABLED
#define MBEDTLS_ECP_DP_CURVE448_ENABLED
#define MBEDTLS_ECP_NIST_OPTIM
#define MBEDTLS_KEY_EXCHANGE_ECDHE_RSA_ENABLED
#define MBEDTLS_ERROR_STRERROR_DUMMY
#define MBEDTLS_GENPRIME
#define MBEDTLS_NO_PLATFORM_ENTROPY
#define MBEDTLS_PK_RSA_ALT_SUPPORT
#define MBEDTLS_PKCS1_V15
#define MBEDTLS_SSL_ALL_ALERT_MESSAGES
#define MBEDTLS_SSL_ENCRYPT_THEN_MAC
#define MBEDTLS_SSL_EXTENDED_MASTER_SECRET
#define MBEDTLS_SSL_FALLBACK_SCSV
#define MBEDTLS_SSL_CBC_RECORD_SPLITTING
#define MBEDTLS_SSL_RENEGOTIATION
#define MBEDTLS_SSL_MAX_FRAGMENT_LENGTH
#define MBEDTLS_SSL_PROTO_TLS1
#define MBEDTLS_SSL_PROTO_TLS1_1
#define MBEDTLS_SSL_PROTO_TLS1_2
#define MBEDTLS_SSL_SESSION_TICKETS
#define MBEDTLS_SSL_EXPORT_KEYS
#define MBEDTLS_SSL_SERVER_NAME_INDICATION
#define MBEDTLS_SSL_TRUNCATED_HMAC
#define MBEDTLS_AES_C
#define MBEDTLS_ARC4_C
#define MBEDTLS_ASN1_PARSE_C
#define MBEDTLS_BASE64_C
#define MBEDTLS_BIGNUM_C
#define MBEDTLS_CHACHA20_C
#define MBEDTLS_CHACHAPOLY_C
```

```
#define MBEDTLS_CIPHER_C
#define MBEDTLS_CTR_DRBG_C
#define MBEDTLS_ECDH_C
#define MBEDTLS_ECP_C
#define MBEDTLS_ENTROPY_C
#define MBEDTLS_GCM_C
#define MBEDTLS_HKDF_C
#define MBEDTLS_MD_C
#define MBEDTLS_MD5_C
#define MBEDTLS_MEMORY_BUFFER_ALLOC_C
#define MBEDTLS_NET_C
#define MBEDTLS_OID_C
#define MBEDTLS_PEM_PARSE_C
#define MBEDTLS_PEM_WRITE_C
#define MBEDTLS_PK_C
#define MBEDTLS_PK_PARSE_C
#define MBEDTLS_PKCS5_C
#define MBEDTLS_PLATFORM_C
#define MBEDTLS_POLY1305_C
#define MBEDTLS_RSA_C
#define MBEDTLS_SHA1_C
#define MBEDTLS_SHA256_C
#define MBEDTLS_SHA512_C
#define MBEDTLS_SSL_CACHE_C
#define MBEDTLS_SSL_COOKIE_C
#define MBEDTLS_SSL_TICKET_C
#define MBEDTLS_SSL_CLI_C
#define MBEDTLS_SSL_TLS_C
#define MBEDTLS_VERSION_C
#define MBEDTLS_X509_USE_C
#define MBEDTLS_X509_CRT_PARSE_C
#define MBEDTLS_SSL_MAX_CONTENT_LEN    16384
#define MBEDTLS_SSL_IN_CONTENT_LEN     16384
#define MBEDTLS_SSL_OUT_CONTENT_LEN 16384
#define MBEDTLS_TLS_DEFAULT_ALLOW_SHA1_IN_CERTIFICATES
```