

智能生活网关产品参考应用程序说明

修订记录

修改时间	修改内容	修改人	备注
2020-05-21	第一个发布版本	远情	针对智能生活 SDKV1.6.0 版本

AOS 版本

应用程序参考代码：Products/example/linkkit_gateway

代码文件：

gateway_cmds.c
gateway_cmds.h
gateway_entry.c
gateway_entry.h
gateway_main.c
gateway_main.h
gateway_ut.c
gateway_ut.h
gateway_api.c
gateway_api.h
linkkit_gateway.mk
makefile
make.settings

代码文件说明：

gateway_entry.c：应用程序主入口，入口函数

```
int application_start(int argc, char **argv)
{
    .....

    设置默认的日志级别
#ifdef DEFAULT_LOG_LEVEL_DEBUG
    IOT_SetLogLevel(IOT_LOG_DEBUG);
#else
#ifdef CSP_LINUXHOST
    IOT_SetLogLevel(IOT_LOG_DEBUG);
#else
    IOT_SetLogLevel(IOT_LOG_INFO);
#endif
#endif

    加载四元组信息，用户需要实现该函数
    load_device_meta_info();

    .....

    注册连云成功回调函数，执行 ota 初始化
    IOT_RegisterCallback(ITE_MQTT_CONNECT_SUCC,
mqtt_connected_event_handler);

    注册串口命令，代码在 gateway_cmds.c 中
#ifdef CONFIG_AOS_CLI
    gateway_register_cmds();
#endif

    .....

    main 函数的 loop 循环
    aos_loop_run();

    return 0;
}
```

gateway_main.c

这个 c 文件里面主要创建了两个线程，这里需要两个线程的原因是子设备的入网操作某些步骤是同步阻塞式的。

user_dispatch_yield 线程：linkkit SDK 代码及注册的用户回调函数都在这个线程中执行；

gateway_main 主线程：这个线程主要进行 linkkit SDK 接口的调用，还有针对子设备的通信和操作；

```
res = HAL_ThreadCreate(&gateway_ctx->g_user_dispatch_thread,
user_dispatch_yield, NULL, &hal_os_thread_param, NULL);
if (res < 0)
{
    gateway_info("HAL_ThreadCreate Failed\n");
    IOT_Linkkit_Close(gateway_ctx->master_devid);
    return -1;
}
```

gateway_cmds.c

这里主要是命令行相关的功能代码，方便测试和调试操作

gateway_ut.c

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

gateway_api.c

这个 c 文件是对子设备操作的 SDK 接口函数的一次封装，支持的功能有添加子设备、删除子设备、复位子设备、查询子设备 ID 等操作，开发者可以直接使用这里的接口函数：

1. 添加一个子设备

添加一个子设备需要对 SDK 进行三次接口调用：

- ☐ IOT_Linkkit_Open 函数将子设备添加到网关本地的数据结构
- ☐ IOT_Linkkit_Connect 将子设备注册并添加拓扑关系到云端
- ☐ IOT_Linkkit_Report 告诉云端子设备上线了

```

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

```

2. 批量添加子设备

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

```

GATEWAY_API int gateway_add_multi_subdev(int master_devid,
iotx_linkkit_dev_meta_info_t *subdev_list, int subdev_num)
{
    ... ..
    connect_times = subdev_num /
GATEWAY_SUBDEV_ONE_TIME_CONNECT_MAX_NUM + (((subdev_num %
GATEWAY_SUBDEV_ONE_TIME_CONNECT_MAX_NUM) == 0) ? 0 : 1);

    for (index = 0; index < connect_times; index++)
    {
        ... ..
        res = IOT_Linkkit_Report(master_devid,
ITM_MSG_CONNECT_SUBDEV, (unsigned char *)p_cur_subdev,
sizeof(iotx_linkkit_dev_meta_info_t) * cur_subdev_num);
    }
}

```

```

        if (res == FAIL_RETURN)
        {
            gateway_err("add multi subdev index:%d Failed\n", index);
        }
        else
        {
            ....
            操作子设备上线
        }
    }

    return res;
}

```

3. 删除一个子设备


- 调用 SDK 函数 `IOT_Linkkit_Report(subdev_id, ITM_MSG_DELETE_TOPO, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t))` 删除一个子设备和网关的拓扑关系，同时会释放网关中子设备的资源。这里需要注意会优先根据子设备的 ProductKey 和 DeviceName 查找子设备，如果没有找到就根据第一个参数 subdev_id 查找子设备，找到就删掉对应的子设备

```

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


```

4. 复位一个子设备

- 调用 SDK 函数 IOT_Linkkit_Report(subdev_id, ITM_MSG_SUBDEV_RESET, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t))复位一个子设备，复位子设备除了会删除和网关的拓扑关系之外，还会删除子设备和手机用户账号的绑定关系，同时会释放网关中子设备的资源。这里需要注意会优先根据子设备的 ProductKey 和 DeviceName 查找子设备，如果没有找到就根据第一个参数 subdev_id 查找子设备，找到就复位对应的子设备

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

5. 根据 ProductKey 和 DeviceName 信息查询一个子设备的 DeviceID

- 调用 IOT_Linkkit_Query(master_devid, ITM_MSG_QUERY_SUBDEV_ID, (unsigned char *)subdev_mate, sizeof(iotx_linkkit_dev_meta_info_t)); 函数根据子设备的 ProductKey 和 DeviceName 信息查询一个子设备的 DeviceID，这个 DeviceID 是一个大于 0 的整数，只是网关里面设备列表数据结构的一个索引

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

```
}
```

宏定义说明：

1. 文件 gateway_ut.h 中的宏定义

宏定义名称	功能说明
GATEWAY_UT_TESTING	这个是 User Test 的参考程序，用户可以根据自己产品的功能进行修改，最好不要直接使用

2. 文件 gateway_api.h 中的宏定义

宏定义名称	功能说明
GATEWAY_SUBDEV_ONE_TIME_CONNECT_MAX_NUM	这个是配置批量上线子设备的最大个数，目前只支持五个，请不要修改

3. 项目根目录下面的文件 make.settings 中配置功能

Feature 名称	功能说明
FEATURE_ALCS_ENABLED	开启本地通信功能，让设备在断网的时候可以通过手机进行本地局域网操控
FEATURE_WIFI_PROVISION_ENABLED	开启 WiFi 配网相关功能，有线网络产品可以不需要这个功能
FEATURE_AWSS_SUPPORT_SMARTCONFIG	一键配网功能，手机发送 WiFi 广播数据包进行配网
FEATURE_AWSS_SUPPORT_ZEROCONFIG	零配配网功能，两个使用场景，一种是通过一个已经配网的设备给进入配网状态的设备进行配网，另外一种是天猫精灵音箱说：天猫精灵找队友 触发天猫进行给设备配网

FEATURE_AWSS_SUPPORT_DEV_AP	设备热点配网功能，设备开启 WiFi 热点，手机连接这个热点进行配网，热点名称是 adh_PK_MAC，其中 PK 是产品型号，MAC 是设备 MAC 地址的后三个字节
-----------------------------	--

主要功能函数说明：

函数名称	功能说明
IOT_SetLogLevel	设置日志级别
load_gateway_meta_info	从 Flash 中读取四元组信息到内存中，其他需要使用四元组的地方从这个内存中读取
gateway_main_init	初始化部分，主要是注册用户回调函数
gateway_main	主线程，主要是循环调用 IOT_Linkkit_Yield
user_connected_event_handler	连云成功
user_disconnected_event_handler	断开云端连接
user_service_request_event_handler	收到云端推送的异步服务事件
user_property_set_event_handler	收到云端设置属性事件，app_parse_property 是属性解析参考函数
user_event_notify_handler	收到云端事件推送，例如手机解绑网关或者子设备等

其他网关功能相关函数

函数名称	功能说明
<code>user_permit_join_event_handler</code>	手机扫描子设备添加二维码之后，网关会触发下面的函数执行将对应 ProductKey 的子设备进行上线操作
<code>gateway_ut_update_subdev</code>	获取网关下面所有绑定的子设备列表，在实际产品中当网关上线后或者网关断网重连后都应该将网关绑定的子设备重新上线一次
<code>user_topolist_reply_handler</code>	在调获取子设备列表函数之后，SDK 会通过这个回调函数告诉用户有哪些子设备信息
<code>IOT_Linkkit_Report(devid, ITM_MSG_LOGOUT, NULL, 0)</code>	如果某个子设备已经添加到云端，网关可以直接调用 SDK 的这个接口函数对 devid 对应的子设备执行下线操作
<code>IOT_Linkkit_Report(devid, ITM_MSG_LOGIN, NULL, 0);</code>	对于通过调用 LOGOUT 接口下线的子设备可以通过调用这个函数再次恢复子设备的上线状态

不带有 AOS 的版本

应用程序参考代码：`examples/linkkit/gateway`

代码文件：

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

```
gateway_api.c
```

```
gateway_api.h
```

这里的源代码相比于基于 AOS 的代码没有 gateway_cmd 文件，其他代码基本一样。

关于一个依赖库文件的说明

在使用品类配网的时候会用到一个库文件进行路由器密码的解密，在 example/iot.mk 里面链接了这个库文件，库文件路径如下，linux 系统使用的这个库文件路径在 lib/linux/libawss_security.a 文件。

```
lib
```

```
└── ANDES_N10
    └── libawss_security.a
└── ARM968E-S
    └── libawss_security.a
└── Cortex-M4
    └── libawss_security.a
└── linux
    └── libawss_security.a
└── xtensa
    └── libawss_security.a
```

代码中自带的五个库文件如果不能满足需要，厂家需要提供对应的编译器给阿里智能生活技术支持人员编译对应的库文件，可以通过智能生活工单系统反馈问题。