文档版本	V1.0
发布日期	

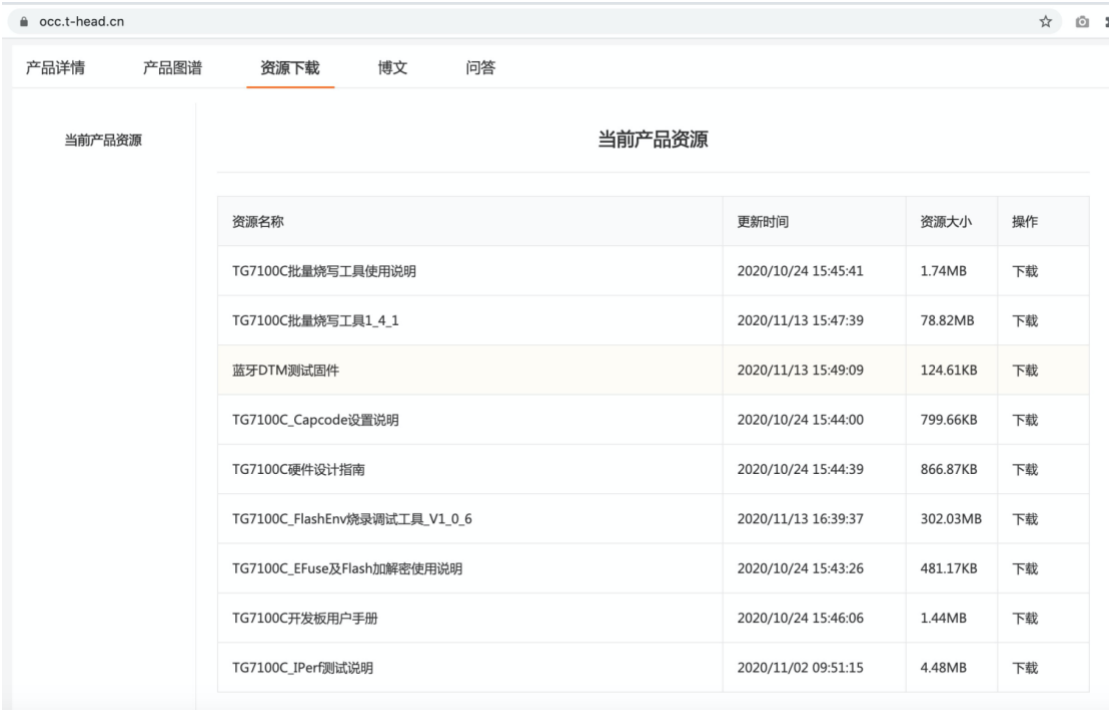
TG7100C 应用开发手册

目录

1.简介

本文档介绍如何基于 TG7100C 芯片和智能生活物联网平台做应用开发。

TG7100C 芯片相关文档和软件工具可以访问平头哥芯片开放社区资源下载页面获取。



2.开发环境与代码编译

2.1 准备开发环境

搭建 SDK 的开发环境。建议您在 64 位 Ubuntu 下搭建设备端 SDK 的开发环境，并使用 vim 编辑代码。该部分的操作请自行查阅网络相关文档完成。

说明 暂不支持在 Windows 系统（含 Windows 子系统）下编译生活物联网平台 SDK。

安装 Ubuntu（版本 16.04 X64）程序运行时库。请您按顺序逐条执行命令。

```
sudo apt-get update
sudo apt-get -y install libssl-dev:i386
sudo apt-get -y install libncurses-dev:i386
sudo apt-get -y install libreadline-dev:i386
```

安装 Ubuntu（版本 16.04 X64）依赖软件包。请您按顺序逐条执行命令。

```
sudo apt-get update
```

```
sudo apt-get -y install git wget make flex bison gperf unzip
sudo apt-get -y install gcc-multilib
sudo apt-get -y install libssl-dev
sudo apt-get -y install libncurses-dev
sudo apt-get -y install libreadline-dev
sudo apt-get -y install python python-pip
```

安装 Python 依赖包。请您按顺序逐条执行命令。

```
python -m pip install setuptools
python -m pip install wheel
python -m pip install aos-cube
python -m pip install esptool
python -m pip install pyserial
python -m pip install scons
```

说明 安装完成后，请您使用 aos-cube --version 查看 aos-cube 的版本号，需确保 aos-cube 的版本号大于等于 0.5.11。

如果在安装过程中遇到网络问题，可使用国内镜像文件。

```
### 安装/升级 pip
python -m pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple/ --upgrade pip
### 基于 pip 依次安装第三方包和 aos-cube
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple/ setuptools
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple/ wheel
pip install --trusted-host=mirrors.aliyun.com -i https://mirrors.aliyun.com/pypi/simple/ aos-cube
```

2.2 下载 SDK

下载方式如下：

https://code.aliyun.com/living_platform/ali-smartliving-device-alios-things/tree/rel_1.6.6

或者 git clone git@code.aliyun.com:living_platform/ali-smartliving-device-alios-things.git -b rel_1.6.6

注意：当您通过 git 命令获取 SDK 时，您需先设置 SSH Key。详细请参见 SSH 介绍。或者如何设置 SSH。

2.3 编译 SDK

请根据以下步骤来编译 SDK。

1. 在开发环境中解压下载的 SDK 压缩包。如果您通过 git 命令的方式下载 SDK，则无需解压。
2. 将开发的业务代码存放到 SDK 相应的目录下。例 Products/example/smart_outlet 为单路智能插座参考实现。
3. 编译 SDK。

快速编译 smart_outlet 示例。

```
./build.sh example smart_outlet tg7100cevb MAINLAND ONLINE 1
```

SDK 根目录 build.sh 文件说明如下，根据硬件使用的模组型号和要编译的应用，修改文件中的如下参数。

```
default_type="example"          //配置产品类型
default_app="smart_outlet"    //配置编译的应用名称
default_board="tg7100cevb"    //配置编译的模组型号
default_region=MAINLAND    //配置设备的连云区域,配置为 MAINLAND 或 SINGAPORE 都可以，设备可以全球范围内激活
default_env=ONLINE            //配置连云环境，默认设置为线上环境（ONLINE）
default_debug=0                 // Debug log 等级
default_args=""                 //配置其他编译参数
//更多介绍请参见 README.md
```

编译完成后，在 out/smart_outlet@tg7100cevb/目录下会生成 smart_outlet@tg7100cevb.bin 文件。该文件为需要烧录到真实设备中的固件。tg7100cevb_ota.bin 文件为 OTA 使用的固件。

说明 build.sh 脚本会自动判断模组的 toolchain（交叉编译工具链）是否已经安装，如果没有安装，脚本会自动安装。

编译出错常见问题：

- 不要使用 Windows 下 Ubuntu 子系统，建议使用虚拟机软件安装 Ubuntu。
- 不要在 Windows 下载解压代码再拷贝到 Ubuntu 系统中，建议直接在 Ubuntu 系统内下载和解压代码，建议使用 git clone 下载代码。
- 不要把代码存放在 Windows 共享目录下，然后通过 mount 挂载到 Ubuntu 系统里。建议直接在 Ubuntu 系统内下载和解压代码。

3.固件烧录

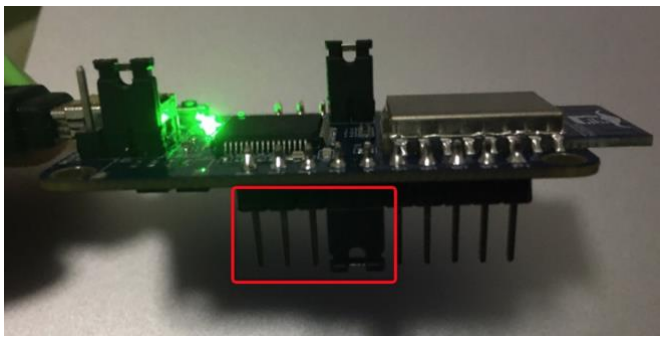
3.1 固件工具与开发板

请在平头哥芯片开放社区资源下载页面获取 TG7100C_FlashEnv 烧录调试工具。

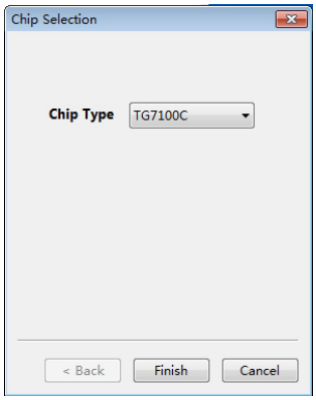
开发板如下图，红色框的地方是 Reset 按键：

3.2 固件烧录步骤

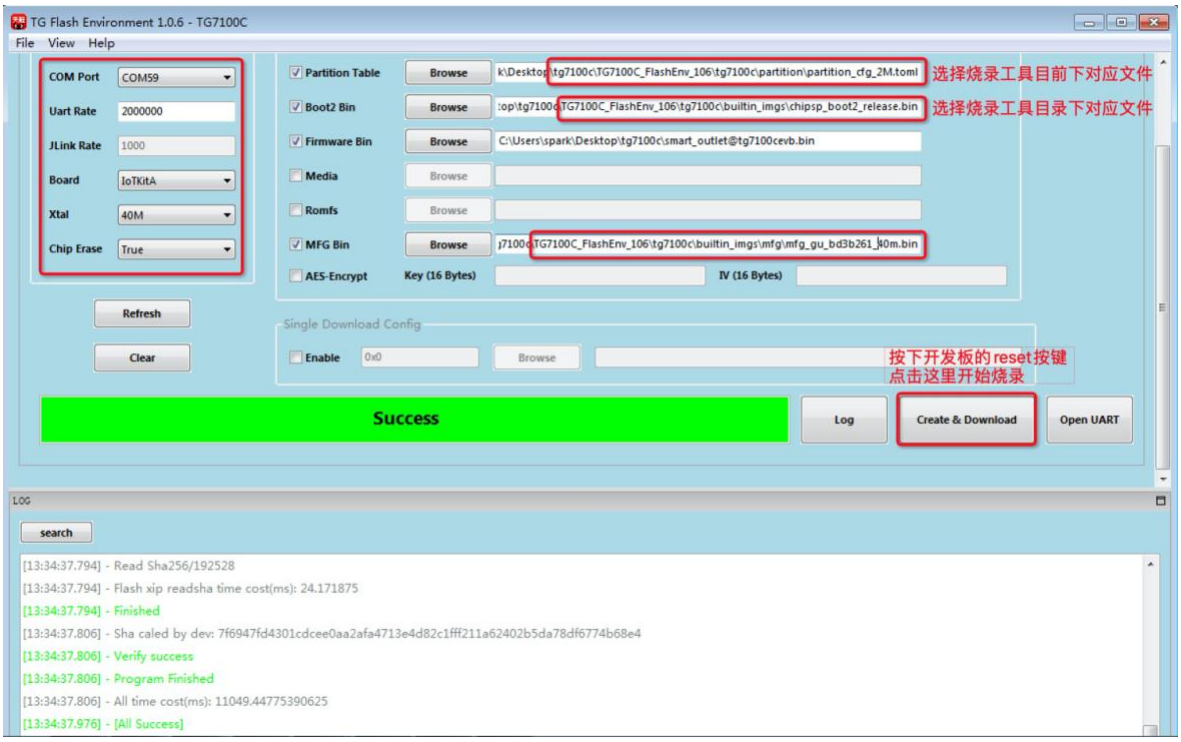
- 首先切换开发板短路帽为烧录模式，如下图所示，短路帽前面会有 3 根空出的引脚：



- 打开下载好的烧录工具目录中的 TGFlashEnv.exe，点击 Finish：

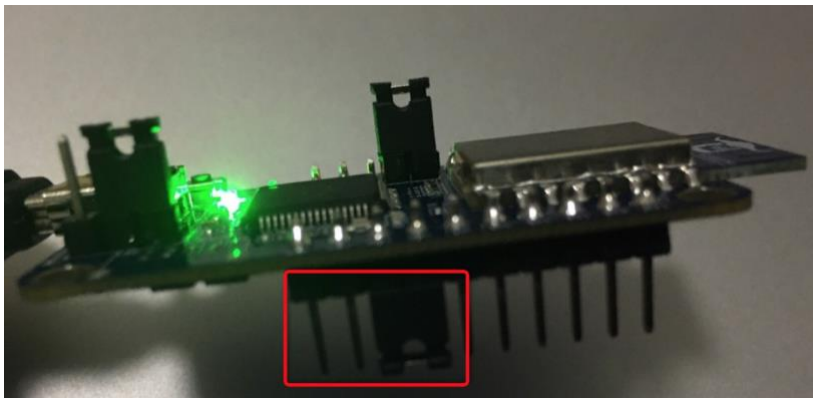


- 然后会进入烧录界面，Interface 选择 Uart，并点击 Refresh 按键，然后如下图所示设置好串口参数，以及选择好对应的烧录文件。确认好配置后，点击“Download”按钮，并同时按下开发板上的“Reset 按键”，即可开始烧录固件。



3.3 设备重启与固件运行

需要切换开发板的短路帽为设备使用模式，如下图所示，短路帽前面会有 2 根空出的引脚：



此时串口工具设置波特率为 2000000，然后按下开发板的 Reset 按键，就看到设备复位开始运行的日志了。

4.创建产品

在生活物联网平台控制台创建产品，创建一个 WiFi 插座。并在设备调试页面生成测试五元组。

4.1 天猫精灵项目

- 创建项目

注意在新建项目时选择天猫精灵生态项目。

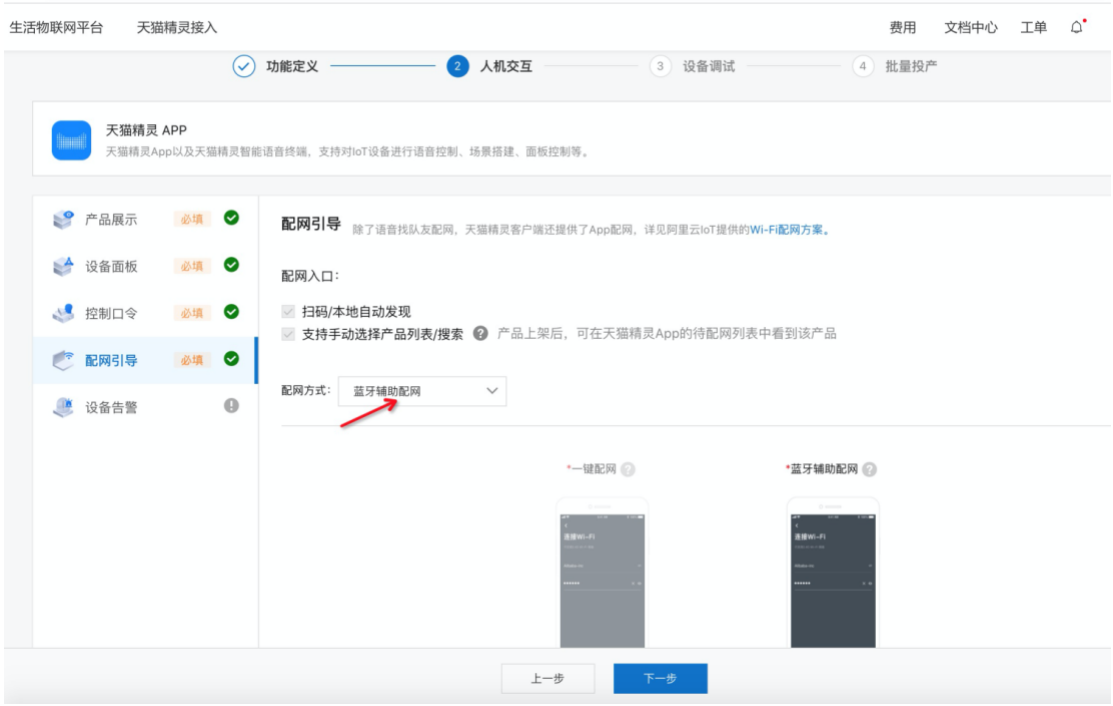
- 创建产品

详细说明请参考[创建产品并定义产品功能]。

- 配置人机交互

完成必填配置项，详细说明请参考配置 App。

注意在配网引导页面将配网方式配置为蓝牙辅助配网。



- 生成测试五元组

目前的平台方案设计要求 WiFi MAC 地址与五元组中的 Device Name 保持一致。如基于开发板开发，可以在固件运行通过串口输入 mac 指令，查询到芯片当前的 WiFi MAC 地址，并用此 MAC 地址作为 Device Name 在设备调试页面申请测试五元组。

查询 WiFi MAC 地址示例。

```
# mac     输入mac指令读取芯片 WiFi MAC
MAC address: 18-b9-05-c0-6e-d5
```

4.2 自有品牌项目

- 创建项目

注意在新建项目时选择自有品牌项目。

- 创建产品

详细说明请参考[创建产品并定义功能](#)。

- 配置人机交互

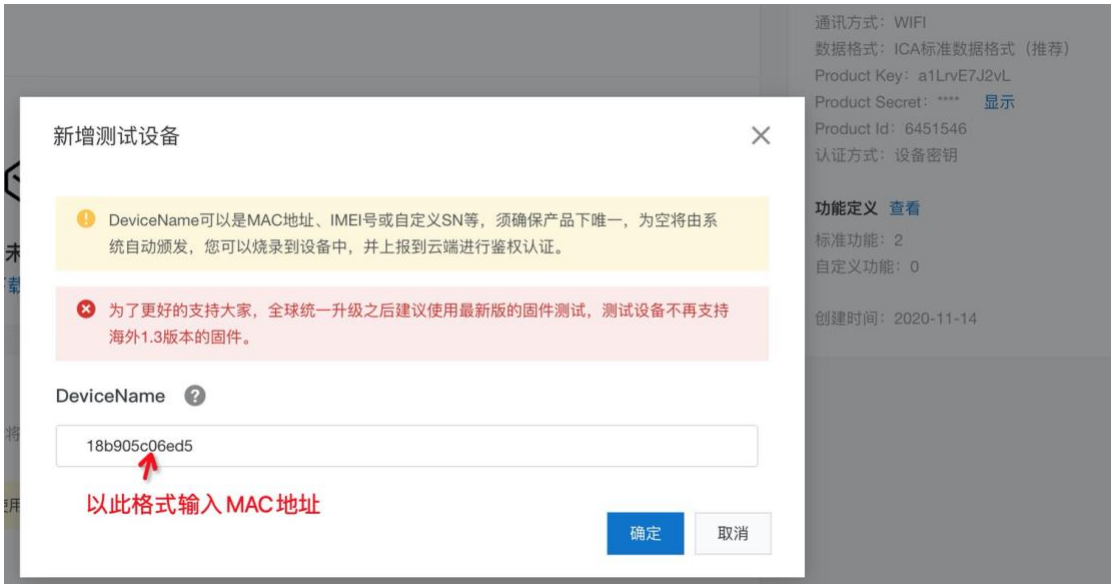
完成必填配置项，详细说明请参考[配置 App]。注意打开使用云智能公版 App 的开关和标准配网类型选择 BLE+Wi-Fi（蓝牙辅助配网）。

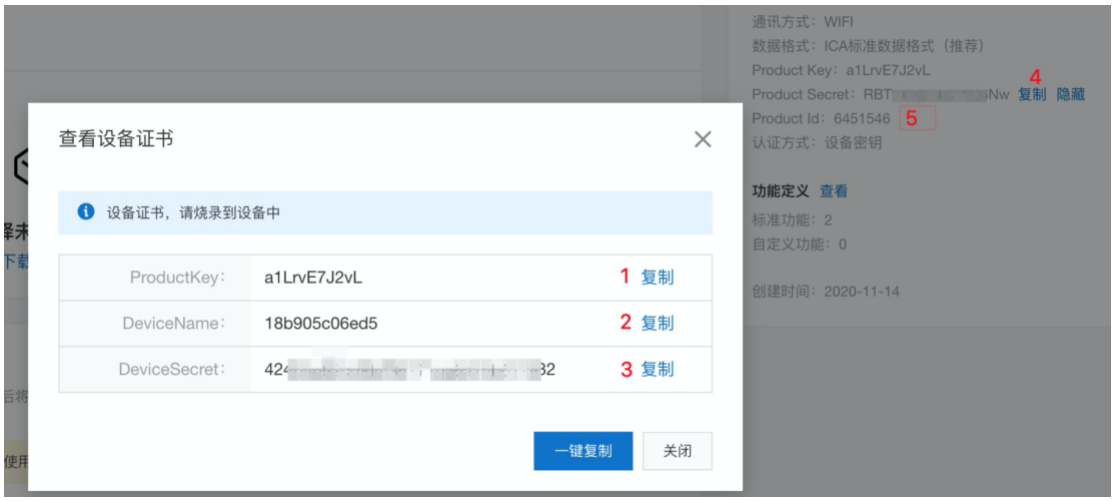
- 生成测试五元组

目前的平台方案设计要求 WiFi MAC 地址与五元组中的 Device Name 保持一致。如基于开发板开发，可以在固件运行通过串口输入 mac 指令，查询到芯片当前的 WiFi MAC 地址，并用此 MAC 地址作为 Device Name 在设备调试页面申请测试五元组。

查询 WiFi MAC 地址示例。

```
# mac     输入mac指令读取芯片 WiFi MAC
MAC address: 18-b9-05-c0-6e-d5
```





5.调试设备连云

5.1 设置五元组

在按照前面的步骤，将设备固件烧录到开发板之后，可以通过 linkkey 命令设置五元组，然后通过 reset 命令重置设备。命令说明如下。

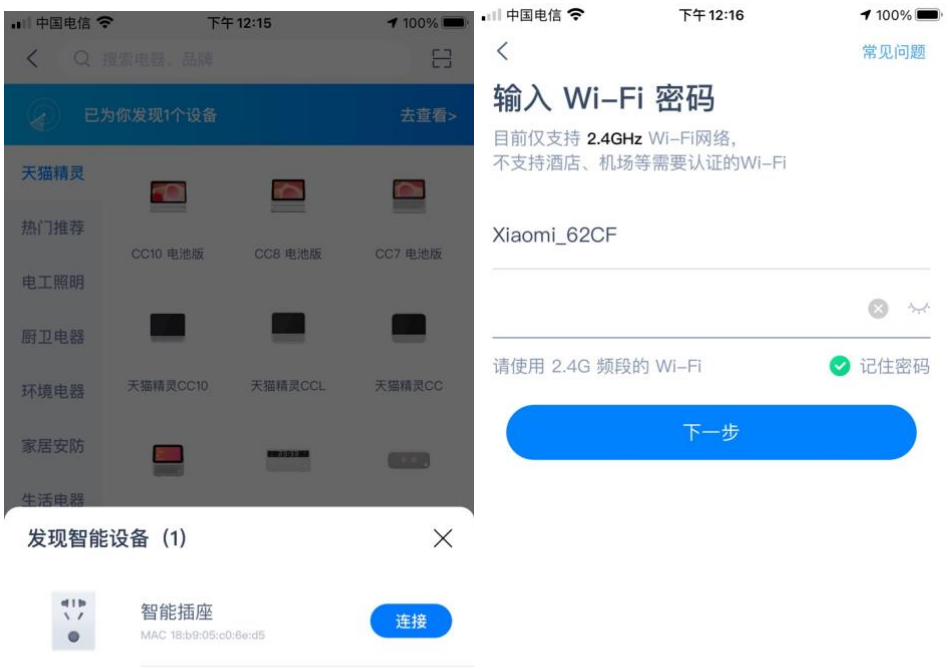
- 设置五元组信息，该五元组信息在飞燕控制台创建的产品与设备信息中找到，在设备上电初始后就需要设置此设备信息：
  - linkkey ProductKey DeviceName DeviceSecret ProductSecret ProductID
- 设备重置，清除设备配网信息：
  - reset

5.2 天猫精灵 App 配网

在开发板上设置天猫精灵产品的五元组，并重置设备。设备正常启动后会处于待配网状态，设备会通过 BLE 广播自己的蓝牙辅助配网相关的设备信息。

天猫精灵 App 可在其设备发现页面发现到处于待配网状态的设备，通过 App 界面可发起对设备的蓝牙辅助配网。

注意支持 TG7100C 蓝牙辅助配网的天猫精灵 App 版本为 V4.13.0 以上版本，请下载更新版本。

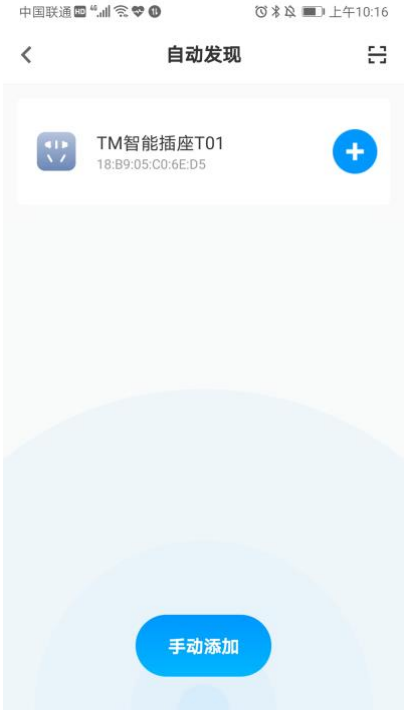


5.3 云智能 App 配网

注意连接云智能 App 的产品基于自有品牌项目，和连接天猫精灵音箱和天猫精灵 App 的产品会有不同的五元组，在开发板上设置自有品牌产品的五元组，并重置设备后，设备会处于待配网状态，会通过 BLE 广播自己的蓝牙辅助配网相关的设备信息。通过 App 界面可发起对设备的蓝牙辅助配网。配网过程、配网结果、配网过程中发生的异常等信息会通过 App 界面展示。

- 支持 TG7100C 蓝牙辅助配网的云智能 App 版本为 V3.5.5 以上版本，请注意在 App Store 下载更新版本。
- 对于在批量投产页面，完成发布产品流程第一步，确认“开发完成”的产品，可以在云智能 App“自动发现”页面发现到处于待配网状态的设备。如下图示。

- 如果完成上述操作，可以在云智能 App“自动发现”页面发现到处于待配网状态的设备。



- 如果产品未确认“开发完成”，可以点击上图“手动添加”按键进入手动添加页面，选择对应品类的“蓝牙+Wi-Fi”连接方式进入配网。

- 配网过程如下图所示。

5.4 配网连云设备端关键日志

5.4.1 蓝牙辅助配网

配网状态	命令/动作	预期日志

设备重置	输入 cli 指令：reset	start----------hal
开启蓝牙辅助配网	输入 cli 指令：ble_awss	ble_awss_open
Parse SSID/PWD	设备解析出热点信息	IOTX_AWSS_GOT_SSID_PASSWD
AP Connect	设备连接 AP	IOTX_AWSS_CONNECT_ROUTER
DHCP Get IP	设备获取 IP 地址	IOTX_AWSS_GOT_IP
Cloud Connect	连云成功	Cloud Connected

5.4.2 零配（天猫精灵音箱）

配网状态	命令/动作	预期日志
设备重置	输入 cli 指令：reset	start----------hal
Dev Scan	输入 cli 指令：awss	IOTX_AWSS_START
Awss Process	输入 cli 指令：active_awss	IOTX_AWSS_ENABLE
Parse SSID/PWD	设备解析出热点信息	IOTX_AWSS_GOT_SSID_PASSWD
AP Connect	设备连接 AP	IOTX_AWSS_CONNECT_ROUTER
DHCP Get IP	设备获取 IP 地址	IOTX_AWSS_GOT_IP
Cloud Connect	连云成功	Cloud Connected

5.4.3 一键配网

配网状态	命令/动作	预期日志
设备重置	输入 cli 指令：reset	start----------hal
Dev Scan	输入 cli 指令：awss	IOTX_AWSS_START
Awss Process	输入 cli 指令：active_awss	IOTX_AWSS_ENABLE
Parse SSID/PWD	设备解析出热点信息	IOTX_AWSS_GOT_SSID_PASSWD
AP Connect	设备连接 AP	IOTX_AWSS_CONNECT_ROUTER

DHCP Get IP	设备获取 IP 地址	IOTX_AWSS_GOT_IP
Cloud Connect	连云成功	Cloud Connected

5.4.4 设备热点配网

TG7100C 芯片上暂不推荐使用。

6.智能插座应用开发

6.1 smart_outlet 介绍

SDK 中提供了单路智能插座的参考实现，路径为 Products/example/smart_outlet。

实现的功能包括：

- 支持云智能 App（V3.5.5 以上）与天猫精灵 App（4.13.0 以上）蓝牙辅助配网。
- 支持通过云端、本地通信（目前仅云智能 App 支持）对设备进行控制的能力。
- 支持通过生活物联网平台进行设备 OTA 的能力。
- 支持恢复工厂设置。
- 支持断电用户设置记忆。

6.2 固件代码说明

6.2.1 代码结构

```
├── Products
│   │   ├── example/smart_outlet
│   │   │   ├── app_entry.c
│   │   │   ├── app_entry.h
│   │   │   ├── combo_net.c
│   │   │   ├── combo_net.h
│   │   │   ├── device_state_manger.c
│   │   │   ├── device_state_manger.h
│   │   │   ├── factory.c
│   │   │   ├── factory.h
│   │   │   ├── makefile
│   │   │   ├── make.settings
│   │   │   ├── msg_process_center.c
```

```
│   │   │   ├── msg_process_center.h
│   │   │   ├── property_report.c
│   │   │   ├── property_report.h
│   │   │   ├── smart_outlet.h
│   │   │   ├── smart_outlet.json
│   │   │   ├── smart_outlet_main.c
│   │   │   ├── smart_outlet.mk
│   │   │   ├── vendor.c
│   │   │   └── vendor.h
```

厂家需要适配的文件（设备初始化等）：vendor.c、vendor.h

应用程序主入口：app_entry.c smart_outlet_main.c

配网和连云状态管理：device_state_manger.c

设备控制指令处理：msg_process_center.c

设备属性上报：property_report.c

厂测模式：factory.c

蓝牙辅助配网：combo_net.c

6.2.2 设备端通用功能说明

以下功能在 smart_outlet 固件已有相关实现，作为设备端的通用功能做一些补充介绍。

- 事件回调

在 smart_outlet_main.c 文件中定义了系统的各种事件处理函数，在 linkkit_main 函数中注册了回调函数。

```
int linkkit_main()
{
    ...

    /* Register Callback */
    IOT_RegisterCallback(ITE_CONNECT_SUCC, user_connected_event_handler);
    IOT_RegisterCallback(ITE_DISCONNECTED, user_disconnected_event_handler);
    IOT_RegisterCallback(ITE_SERVICE_REQUEST, user_service_request_event_handler);
    IOT_RegisterCallback(ITE_PROPERTY_SET, user_property_set_event_handler);
#ifdef ALCS_ENABLED
```

```
    /*Only for local communication service(ALCS) */
    IOT_RegisterCallback(ITE_PROPERTY_GET, user_property_get_event_handler);
#endif
    IOT_RegisterCallback(ITE_REPORT_REPLY, user_report_reply_event_handler);
    IOT_RegisterCallback(ITE_TRIGGER_EVENT_REPLY,
user_trigger_event_reply_event_handler);
    IOT_RegisterCallback(ITE_INITIALIZE_COMPLETED, user_initialized);
    IOT_RegisterCallback(ITE_EVENT_NOTIFY, user_event_notify_handler);
    ...
}
```

用到的回调函数的说明如下。

事件	事件触发条件说明
ITE_CONNECT_SUCC	与云端连接成功时
ITE_DISCONNECTED	与云端连接断开时
ITE_RAWDATA_ARRIVED	SDK 收到 raw data 数据时
ITE_SERVICE_REQUEST	SDK 收到服务（同步/异步）调用请求时
ITE_PROPERTY_SET	SDK 收到属性设置请求时
ITE_PROPERTY_GET	SDK 收到属性获取的请求时
ITE_REPORT_REPLY	SDK 收到上报消息的应答时
ITE_TRIGGER_EVENT_REPLY	SDK 收到事件上报消息的应答时
ITE_EVENT_NOTIFY	SDK 收到事件通知时
ITE_INITIALIZE_COMPLETED	设备初始化完成时

- 属性上报

  产品的属性发生变化时，需要将变化后的数值上报到物联网平台。可以根据产品需求增加属性变化的检测以及上报逻辑。

```
void user_post_property(property_report_msg_t * msg)
{
    …
#ifdef TSL_FY_SUPPORT
    //兼容旧版本开关 PowerSwitch 属性
    cJSON_AddNumberToObject(response_root, "PowerSwitch", msg->powerswitch);
#endif
    //处理新版本物模型开关 powerstate 属性
    cJSON_AddNumberToObject(response_root, "powerstate", msg->powerswitch);
    //处理新版物模型 allPowerstate 属性
    cJSON_AddNumberToObject(response_root, "allPowerstate", msg->all_powerstate);
    property_payload = cJSON_PrintUnformatted(response_root);
    cJSON_Delete(response_root);

    char *property_formated;
    uint32_t len;
    res = user_property_format(property_payload,strlen(property_payload),&property_formated,&len);
#ifdef EN_COMBO_NET
    //对于 WiFi-BLE Combo 设备可以同时通过蓝牙控制链路上报属性值。
    if (combo_ble_conn_state()) {
        if (0 == res) {
            combo_status_report(property_formated, strlen(property_formated));
            LOG_TRACE("Post Property Message ID: %d Payload %s", res, property_formated);
        } else {
            combo_status_report(property_payload, strlen(property_payload));
            LOG_TRACE("Post Property Message ID: %d Payload %s", res, property_payload);
        }
    }
#endif
    if (0 == res) {
        if (msg->seq != NULL && strcmp(msg->seq, SPEC_SEQ)) {
```

```
            res = IOT_Linkkit_Report_Ext(user_example_ctx->master_devid,
                        ITM_MSG_POST_PROPERTY,
                        (unsigned char *)property_formated, strlen(property_formated), msg->flag);
        } else {
            res = IOT_Linkkit_Report(user_example_ctx->master_devid,
                        ITM_MSG_POST_PROPERTY,
                        (unsigned char *)property_formated, strlen(property_formated));
        }
        LOG_TRACE("Post Property Message ID: %d Payload %s", res, property_formated);
        example_free(property_formated);
    } else {
        if (msg->seq != NULL && strcmp(msg->seq, SPEC_SEQ)) {
            res = IOT_Linkkit_Report_Ext(user_example_ctx->master_devid,
                        ITM_MSG_POST_PROPERTY,
                        (unsigned char *)property_payload, strlen(property_payload), msg->flag);
        } else {
            res = IOT_Linkkit_Report(user_example_ctx->master_devid,
                        ITM_MSG_POST_PROPERTY,
                        (unsigned char *)property_payload, strlen(property_payload));
        }
        LOG_TRACE("Post Property Message ID: %d Payload %s", res, property_payload);
    }
    example_free(property_payload);
}
```

- 属性设置

  smart_outlet 按对 ITE_PROPERTY_SET 注册的回调函数，在回调函数 user_property_set_event_handler 中获取云端设置的属性值，并原样将收到的数据发回给云端，这样可以更新在云端的设备属性值，用户可在此处对收到的属性值进行处理。

```
static int user_property_set_event_handler(const int devid, const char *request, const int request_len)
{
    ...
```

```
    property_setting_handle(request, request_len, &msg);
    ...
}

static int property_setting_handle(const char *request, const int request_len, recv_msg_t * msg)
{
    ...
    if ((item = cJSON_GetObjectItem(root, "setPropsExtends")) != NULL &&
        cJSON_IsObject(item)) {
        ...
    }
    if ((item = cJSON_GetObjectItem(root, "powerstate")) != NULL && cJSON_IsNumber(item)) {
        //设置 powerstate 属性处理
        msg->powerswitch = item->valueint;
        msg->all_powerstate = msg->powerswitch;
        ret = 0;
    }
#ifdef TSL_FY_SUPPORT /*  支持旧版本开关 PowerSwitch 属性  */
    else if ((item = cJSON_GetObjectItem(root, "PowerSwitch")) != NULL &&.
        cJSON_IsNumber(item)) {
        msg->powerswitch = item->valueint;
        ret = 0;
    }
#endif
    else if ((item = cJSON_GetObjectItem(root, "allPowerstate")) != NULL &&.
        cJSON_IsNumber(item)) {
        //设置 allPowerstate 属性处理
        msg->powerswitch = item->valueint;
        msg->all_powerstate = msg->powerswitch;
        ret = 0;
    }
#ifdef AOS_TIMER_SERVICE
```

```
    else if (((item = cJSON_GetObjectItem(root, "LocalTimer")) != NULL &&.
            cJSON_IsArray(item))|| \
            ((item = cJSON_GetObjectItem(root, "CountDownList")) != NULL &&.
            cJSON_IsObject(item)) || \
            ((item = cJSON_GetObjectItem(root, "PeriodTimer")) != NULL &&.
            cJSON_IsObject(item)) || \
            ((item = cJSON_GetObjectItem(root, "RandomTimer")) != NULL &&.
            cJSON_IsObject(item)))
    {
            // Timer service  定时、倒计时相关属性设置的处理
            cJSON_Delete(root);                 // Before LocalTimer Handle, Free Memory
            timer_service_property_set(request);
            user_example_ctx_t *user_example_ctx = user_example_get_ctx();
            IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                    (unsigned char *)request, request_len);
            return 0;
    }
#endif
    else {
            LOG_TRACE("property set payload is not JSON format");
            ret = -1;
    }

    cJSON_Delete(root);
    if (ret != -1)
            send_msg_to_queue(msg);

    return ret;
}
```

- 本地通信功能（目前仅云智能 App 支持）

  本地定时功能介绍请参考 最佳实践 > 本地控制 > 本地通信开发实践。

本地定时功能在文件 smart_outlet_main.c 中通过宏 ALCS_ENABLED 来管理。使用 IOT_RegisterCallback 函数注册 ITE_PROPERTY_GET 事件，对应回调函数实现为 user_property_get_event_handler。

此函数中目前已实现的本地通信请求的设备属性如下所示，如果产品需要增加功能，可以相应的增加新属性的处理 case。

```
#ifdef ALCS_ENABLED
static int user_property_get_event_handler(const int devid, const char *request, const int request_len,
char **response,
            int *response_len)
{
    ...
    for (int index = 0; index < cJSON_GetArraySize(request_root); index++) {
        item_propertyid = cJSON_GetArrayItem(request_root, index);
        ...
        LOG_TRACE("Property ID, index: %d, Value: %s", index, item_propertyid->valuestring);
        if (strcmp("powerstate", item_propertyid->valuestring) == 0) {
            //处理新版物模型开关 powerstate 属性
            cJSON_AddNumberToObject(response_root, "powerstate",
                    device_status->powerswitch);
        }
        else if (strcmp("allPowerstate", item_propertyid->valuestring) == 0) {
            //处理新版物模型 allPowerstate 属性
            cJSON_AddNumberToObject(response_root,"allPowerstate",
                    device_status->all_powerstate);
        }
#ifdef TSL_FY_SUPPORT /* support old feiyan TSL */
        else if (strcmp("PowerSwitch", item_propertyid->valuestring) == 0) {
            //兼容旧版本开关 PowerSwitch 属性
            cJSON_AddNumberToObject(response_root, "PowerSwitch",
                    device_status->powerswitch);
        }
#endif
```

```
#ifdef AOS_TIMER_SERVICE
            else if (strcmp("LocalTimer", item_propertyid->valuestring) == 0) {
                ...   //处理本地定时 LocalTimer
            } else if (strcmp("CountDownList", item_propertyid->valuestring) == 0) {
                ...   //处理倒计时
#endif
            }
        }
        ...
}
#endif
```

- 云端解绑与恢复出厂默认设置通知

  设备被解绑后，云端会下发一个解绑事件通知：
  {"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}} 设备收到此消息可以做重置配网、清空本地数据等处理。
  如果通过 App 将设备恢复出厂默认设置，云端会下发一个 Reset 事件通知：
  {"identifier":"awss.BindNotify","value":{"Operation":"Reset"}} 设备收到此消息可以做重置配网、清空本地数据等处理。设备开发者可以结合具体产品类型，决定收到解绑和恢复出厂默认设置通知后做哪些清空操作。
  可以参考示例代码 example/smart_outlet/smart_outlet_main.c 中 notify_msg_handle 函数。

```
static int notify_msg_handle(const char *request, const int request_len)
{
    ....
    if (!strcmp(item->valuestring, "awss.BindNotify")) {
        cJSON *value = cJSON_GetObjectItem(request_root, "value");
        if (value == NULL || !cJSON_IsObject(value)) {
            cJSON_Delete(request_root);
            return -1;
        }
        cJSON *op = cJSON_GetObjectItem(value, "Operation");
        if (op != NULL && cJSON_IsString(op)) {
```

```
            if (!strcmp(op->valuestring, "Bind")) {       //绑定通知
                LOG_TRACE("Device Bind");
                vendor_device_bind();       //设备绑定时需要完成的操作，设备应用可定义
            } else if (!strcmp(op->valuestring, "Unbind")) {   //解绑通知
                LOG_TRACE("Device unBind");
                vendor_device_unbind();     //设备解绑时需要完成的操作，设备应用可定义
            } else if (!strcmp(op->valuestring, "Reset")) {   //重置通知
                LOG_TRACE("Device reset");
                vendor_device_reset();      //设备重置时需要完成的操作，设备应用可定义
            }
        }
    }
    ....
}
```

- 蓝牙辅助配网

  蓝牙辅助配网设备端开发请参考文档 最佳实践 > 配网开发最佳实践 > 蓝牙辅助配网最佳实践 > 设备端开发。

- 本地定时功能

  本地定时功能设备端开发请参考文档 最佳实践 > 本地定时功能开发实践 > 开发设备端本地定时功能。

6.3 产品功能开发

对单路智能插座应用，只需要较小的修改，就可以完成产品固件的输出。根据产品的不同需求，涉及到的调整项介绍如下。

6.3.1 GPIO 适配

单路插座需要 3 个 GPIO，用途分别为：

- 控制 LED 亮灭；
- 控制继电器开关；
- 读取按键状态。

那么只需要修改 vendor.c 中定义，实例如下：

```
......
#elif (defined (TG7100CEVB))
#define LED_GPIO       1                    // 控制 LED 亮灭
#define RELAY_GPIO    5                     // 控制继电器开关
#define KEY_GPIO       3                    // 读取按键状态
......
```

产品开发时，可以根据具体的原理图设计配置对应的 GPIO。

6.3.2 状态 LED 指示

目前 smart_outlet 中默认实现的状态显示如下，可以根据产品的不同需求做调整。

设备状态定义在文件 Products/example/smart_outlet/device_state_manger.h 中。

```
typedef enum {
    RECONFIGED = 0,                 //reconfig with netconfig exist
    UNCONFIGED,                     //awss start
    AWSS_NOT_START,                 //standby mode(not start softAP)
    GOT_AP_SSID,                    //connect AP successfully
    CONNECT_CLOUD_SUCCESS,          //connect cloud successfully
    CONNECT_CLOUD_FAILED,           //connect cloud failded
    CONNECT_AP_FAILED,              //connect ap failed
    CONNECT_AP_FAILED_TIMEOUT, //connect ap failed timeout
    APP_BIND_SUCCESS,               //bind sucessfully, wait cmd from APP
    ...
    UNKNOW_STATE
} eNetState;
```

状态显示的处理代码在文件 Products/example/smart_outlet/device_state_manger.c 中，indicate_net_state_task 函数负责指示设备处于不同状态。

```
static void indicate_net_state_task(void *arg)
{
    uint32_t nCount = 0;
    uint32_t duration = 0;
    int pre_state = UNKNOW_STATE;
    int cur_state = UNKNOW_STATE;
```

```
int switch_stat = 0;

while (1) {
    pre_state = cur_state;
    cur_state = get_net_state();
    switch (cur_state) {
        case RECONFIGED:
            ...
            break;
        case UNCONFIGED:
            ...
            break;
        case AWSS_NOT_START:
            ...
            break;
        case GOT_AP_SSID:
        case CONNECT_CLOUD_FAILED:
            ...
            break;
        case CONNECT_AP_FAILED_TIMEOUT:
            ...
            break;
        case CONNECT_AP_FAILED:
            ...
            break;
        case CONNECT_CLOUD_SUCCESS:
            ...
            break;
        case APP_BIND_SUCCESS:
            ...
            break;
        ...
```

```
                default:
                    break;
            }
            aos_msleep(100);
        }
        ...
}
```

状态	默认 LED 显示
配网模式	插座 LED 反复闪烁，亮 0.8 秒，灭 0.8 秒。
恢复出厂设置	插座 LED 反复闪烁，亮 0.2 秒，灭 0.2 秒。
连接 AP 超时 连接 AP 认证失败 （超时时间 2 分钟）	插座 LED 反复闪烁的模 式更改为，亮 0.5 秒、灭 0.5 秒，闪烁两分钟之后停止闪烁。停止闪烁之后，如 果插座配电使能则 LED 灯点亮，否则 LED 灯灭掉。
连接 AP 成功、尝试连云	插座 LED 反复闪烁，亮 0.8 秒，灭 0.8 秒，然后开始尝试连接云端。
连云失败	连接云端失败后，需要再次尝试连接，其间 LED 的显示与"连接 AP 成功、尝试连云"模式一样。
连云成功	当设备连接云端成功，则停止 LED 闪烁，若插座配电打开则 LED 点亮，若插座配电未打开则 LED 灭掉。

6.3.3 设备按键处理

根据用户按下按键的时长，确定用户的行为，目前按键有三种用户行为处理。代码 Products/example/smart_outlet/device_state_manger.c 中 key_detect_event_task 函数负责按键处理。如下定义了各种行为的时间，如果需要调整不同行为的按键时长，可以自行修改。

```
//长按 4S 进入网络配置模式，开始重新配网
#define AWSS_REBOOT_TIMEOUT (4 * 1000)
//长按 6S 进入恢复出厂设置，在设备已进入网络配置模式下
#define AWSS_RESET_TIMEOUT (6 * 1000)
//按键按下超过 100ms，小于 500ms，表示有按键按下
```

```
#define KEY_PRESSED_VALID_TIME_MIN 100
#define KEY_PRESSED_VALID_TIME_MAX 500
#define KEY_DETECT_INTERVAL 50    //按键按下的检测时间间隔  50ms
#define AWSS_REBOOT_CNT AWSS_REBOOT_TIMEOUT /KEY_DETECT_INTERVAL
#define AWSS_RESET_CNT AWSS_RESET_TIMEOUT /KEY_DETECT_INTERVAL
#define KEY_PRESSED_CNT KEY_PRESSED_VALID_TIME /KEY_DETECT_INTERVAL

// 此函数处理插座按键检测
void key_detect_event_task(void *arg)
{
    int nCount = 0, awss_mode = 0;
    int timeout = (AWSS_REBOOT_CNT < AWSS_RESET_TIMEOUT)?
        AWSS_REBOOT_CNT : AWSS_RESET_TIMEOUT;

    while (1) {
        if (!product_get_key()) {
            nCount++;
            LOG("nCount :%d", nCount);
        } else {
            if (nCount >= KEY_PRESSED_CNT && nCount < timeout) {    // 按键控制
                if (product_get_switch() == ON) {     // 按键控制插座关闭继电器
                    product_set_switch(OFF);
                    user_post_powerstate(OFF);
                } else {                              // 按键控制插座打开继电器
                    product_set_switch(ON);
                    user_post_powerstate(ON);
                }
            }
            if ((awss_flag == 0) && (nCount >= AWSS_REBOOT_CNT)) {
                LOG("do awss reboot");          // 长按 4S 进入网络配置模式，开始重新配网
                do_awss_reboot();
                break;
```

```
            } else if ((awss_flag == 1) && (nCount > AWSS_RESET_CNT)) {
                LOG("do awss reset");              // 长按 6S 进入恢复出厂设置
                do_awss_reset();                   // 实际执行设备重置
                break;
            }
            nCount = 0;
        }
        if ((awss_flag == 0) && (nCount >= AWSS_REBOOT_CNT && awss_mode == 0)) {
            set_net_state(RECONFIGED);          // 设置相应的设备状态
            awss_mode = 1;
        } else if ((awss_flag == 1) && (nCount > AWSS_RESET_CNT && awss_mode == 0)) {
            set_net_state(UNCONFIGED);          // 设置相应的设备状态
            awss_mode = 1;
        }
        aos_msleep(KEY_DETECT_INTERVAL); // 检测按键间隔为 50ms
    }
    aos_task_exit(0);
}
```

短按

如果按键按下时长长于 100ms，小于 500ms， 认为用户是进行按键开关。

长按

如果用户按下时间超过 5s，认为用户触发设备进入网络配置模式。如果用户确认设备已经进入网络配置模式，此时，继续按键，设备进入恢复出厂模式。

6.4 自有品牌项目出海产品说明

控制台创建项目和产品的操作，此处不再重复，对于要出货海外的自有品牌项目的产品需要做的操作说明如下。

6.4.1 物模型

在物模型定义上请注意按以下介绍操作。

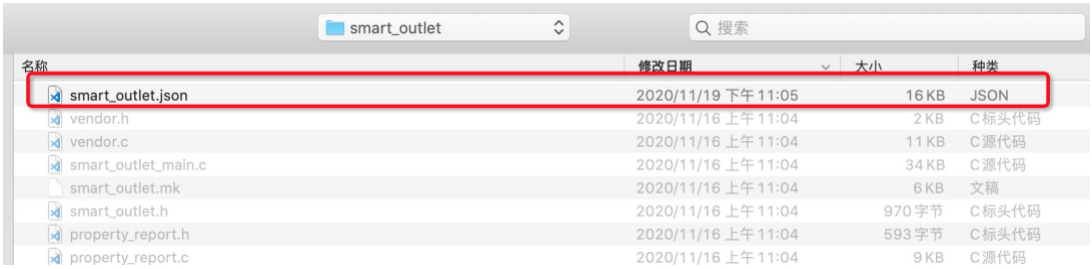
在 smart_outlet 示例代码里面提供了一份导出的完整的物模型文件 Products/example/smart_outlet/smart_outlet.json，建议在创建产品后通过导入物模型复制到新建产品中（目前只有自有品牌项目支持导入物模型）。

新产品导入物模型有两种方式，即从已有产品拷贝一份物模型，或者导入一份 json 文件。

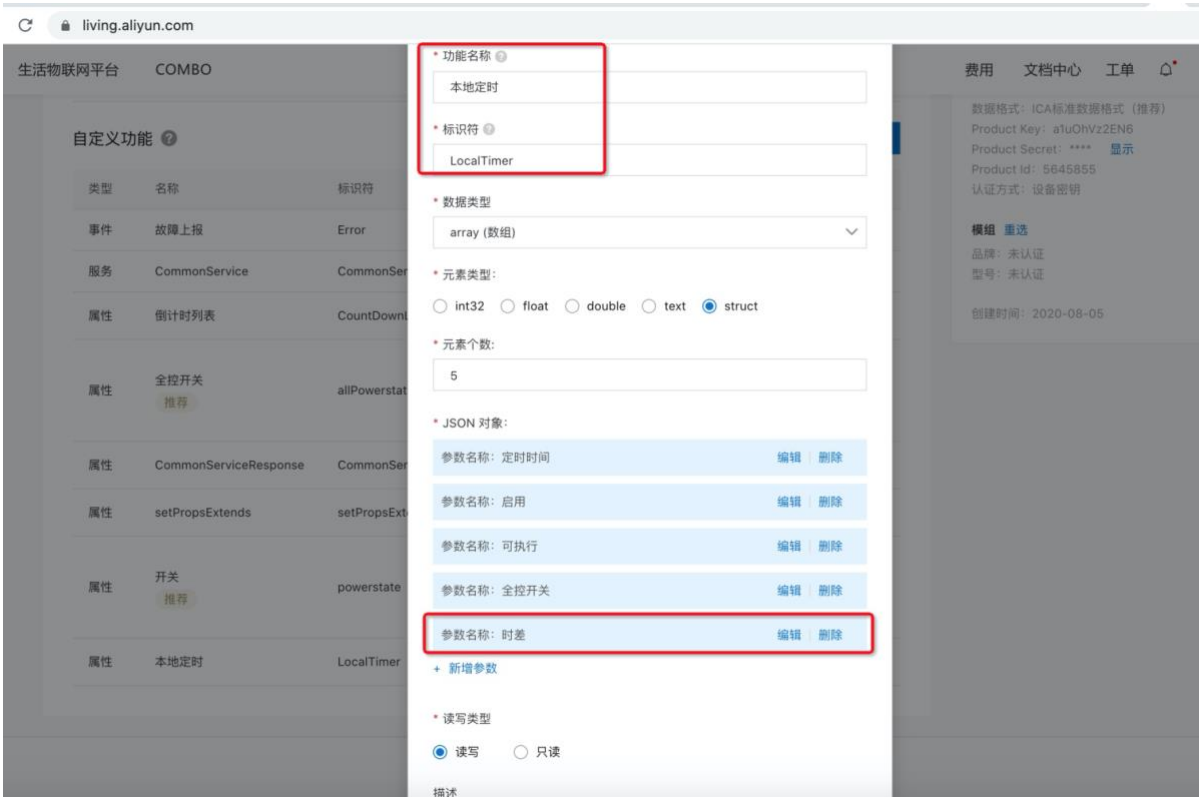
如果没有可拷贝产品，可以采用上传 json 格式物模型文件的方式。

- 拷贝新建产品的 Product Key，编辑文件 Products/example/smart_outlet/smart_outlet.json，将 productKey 字段替换成新的，并保存。

- 选择并上传 json 格式物模型文件。



- 确定本地定时属性的 JSON 对象定义内包括时差（TimezoneOffset）。这个会影响在海外出货时不同时区下的本地定时功能能否正常工作，请务必做确认。



6.4.2 海外语音平台

三方语音平台 Amazon Alexa 和 Google Home 的对接请参考以下文档，如遇问题可提工单请技术支持协助处理。

[最佳实践 > 三方语音平台 > 自有 App 定制 Amazon Alexa 技能]

[最佳实践 > 三方语音平台 > 公版 App 使用 Amazon Echo 音箱控制设备]

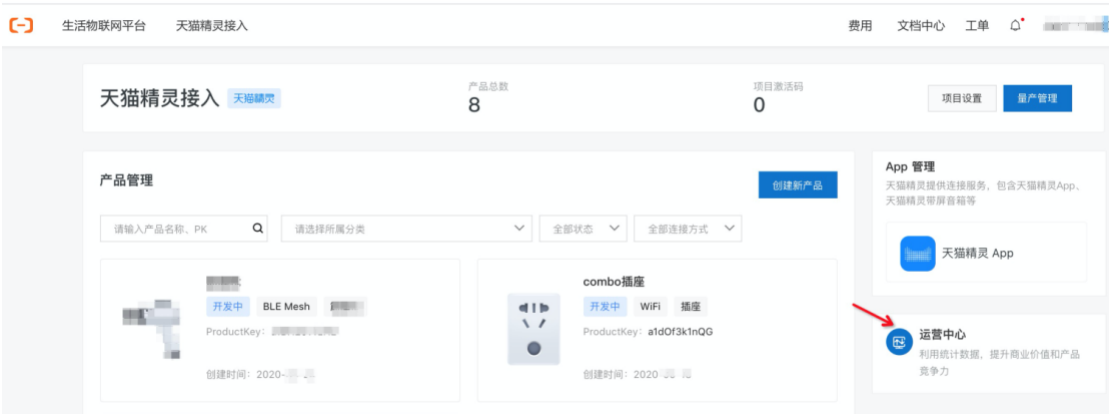
[最佳实践 > 三方语音平台 > 自有 App 定制 Google Assistant 技能]

[最佳实践 > 三方语音平台 > 公版 App 使用 Google Home 音箱控制设备]

7.设备 OTA 说明

下面以一个天猫精灵项目产品为例说明 OTA 的过程，自有品牌项目的产品过程类似。

- 登陆生活物联网平台运营中心。



- 在“设备运维-固件升级”页面为对应项目下的对应产品添加新版本固件。

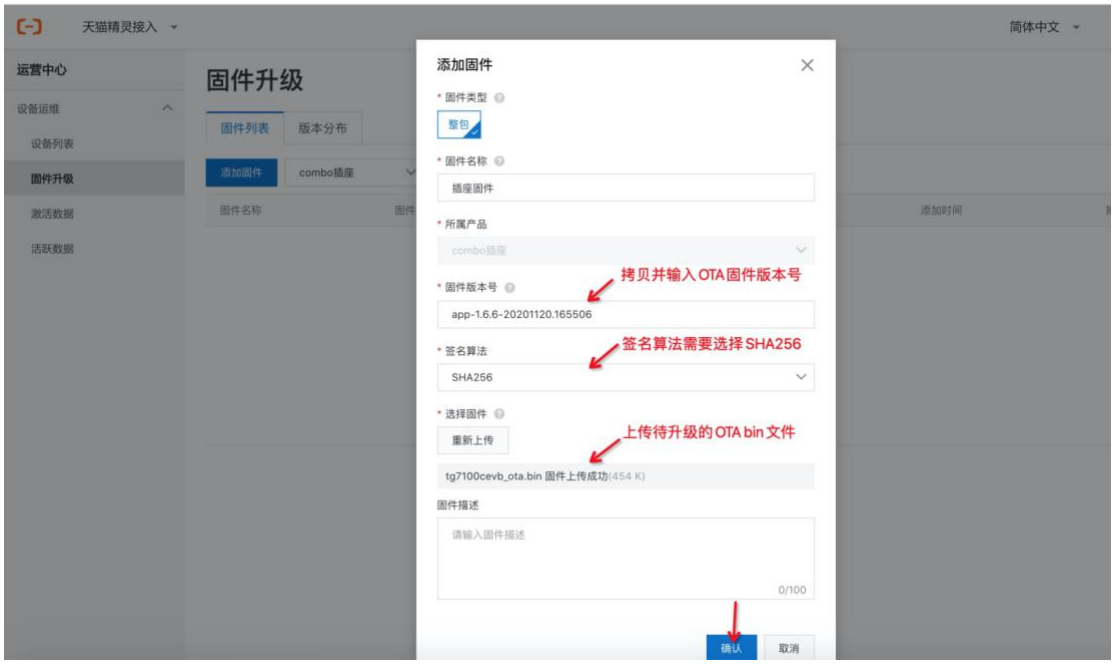
- 获取待升级固件文件和版本号。

```
spark.yb@l[redacted] /home/spark.yb/rel_1.6.6/ali-smartliving-device-alios-things/out/smart_outlet@tg7100cevb
$ls -l
total 16764
-rw-r--r-- 1 spark.yb users       38 Nov 20 16:55 readme.txt
-rwxr-xr-x 1 spark.yb users   853672 Nov 20 16:55 smart_outlet@tg7100cevb.bin
-rwxr-xr-x 1 spark.yb users 10885620 Nov 20 16:55 smart_outlet@tg7100cevb.elf
-rw-r--r-- 1 spark.yb users  4078887 Nov 20 16:55 smart_outlet@tg7100cevb.map
-rw-r--r-- 1 spark.yb users     5823 Nov 20 16:55 smart_outlet@tg7100cevb_map.csv
-rwxr-xr-x 1 spark.yb users   861200 Nov 20 16:55 smart_outlet@tg7100cevb.stripped.elf
-rw-r--r-- 1 spark.yb users   464876 Nov 20 16:55 tg7100cevb_ota.bin
 spark.yb@l[redacted] /home/spark.yb/rel_1.6.6/ali-smartliving-device-alios-things/out/smart_outlet@tg7100cevb
$cat readme.txt
version : app-1.6.6-20201120.165506
```

版本号

OTA的bin文件

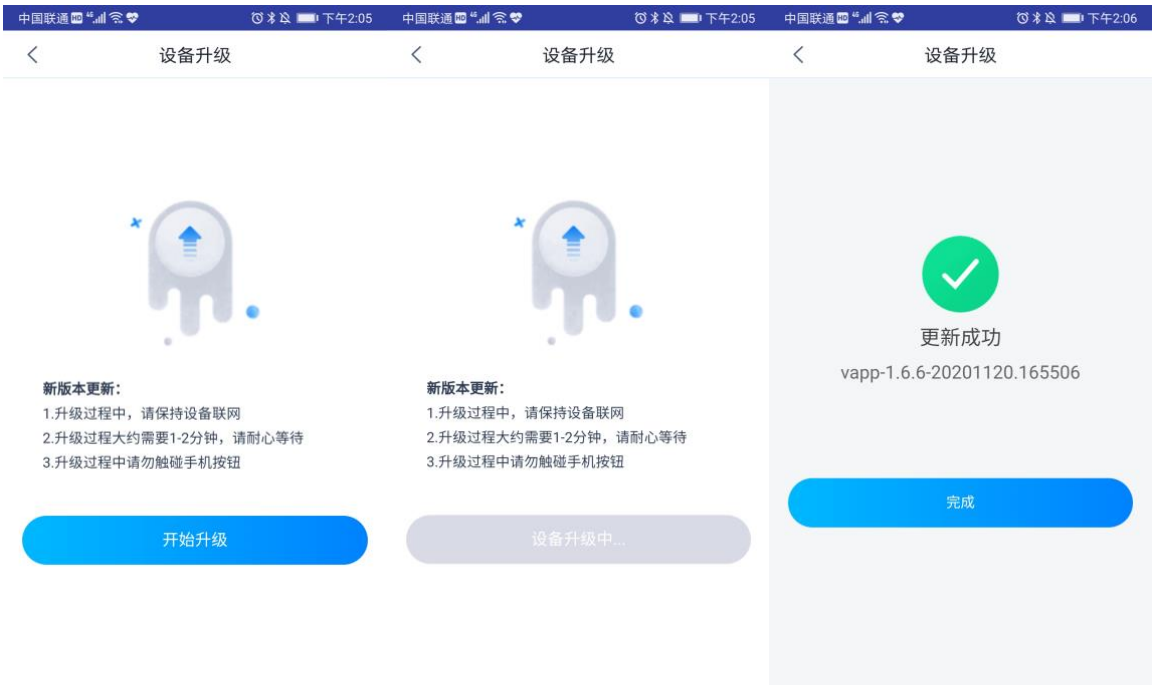
- 添加固件，注意签名算法需要选择 SHA256。



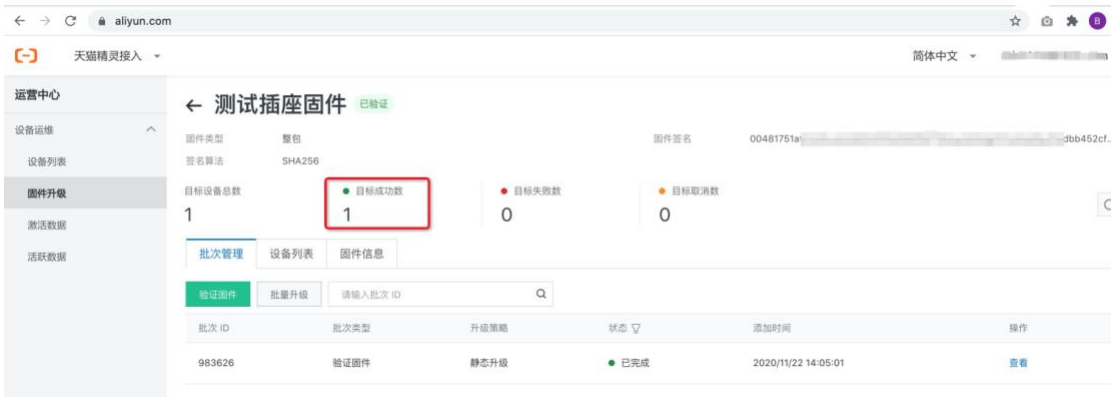
- 验证固件

- 进入天猫精灵 App，查看待验证设备的设备详情页面，可以看到已有设备升级的提示，点击“立即更新”即可开始 OTA。

- 等待 OTA 完成。



- 查看运营中心升级状态。



8.量产五元组说明

前面介绍了测试五元组的获取，本节介绍量产阶段如何获取五元组。天猫精灵项目和自有品牌项目会有区别。

生活物联网平台设备量产详细说明请参考量产流程介绍与量产设备。

8.1 天猫精灵项目三元组

对于天猫精灵项目的产品，天猫精灵有 MAC 地址段，在人机交互-配网引导里面选择蓝牙辅助配网后，生成三元组时可以分配合法 MAC 地址作为 Device Name。

烧录时通过量产烧录工具写入 Device Name 同时覆盖芯片 WiFi MAC 即可实现 WiFi MAC 与 Device Name 保持一致的要求。

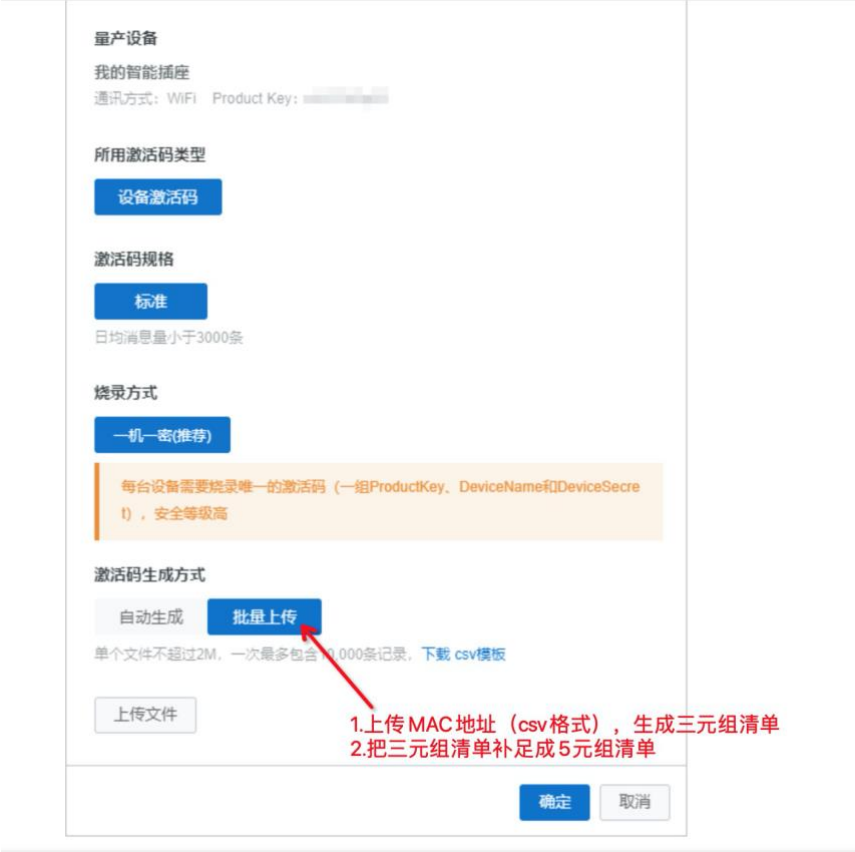
天猫精灵项目产品在量产设备时，选择自动生成即可。

8.2 自有品牌项目三元组

对于自有品牌项目的产品，智能生活物联网平台没有 MAC 地址段提供，需要厂商自己上传 MAC 地址，作为 Device Name 来生成三元组。

烧录时通过量产烧录工具写入 Device Name 同时覆盖芯片 WiFi MAC 即可实现 WiFi MAC 与 Device Name 保持一致的要求。

自有品牌项目产品在量产设备时，需要准备好符合模版要求的 MAC 地址段，并选择批量上传。

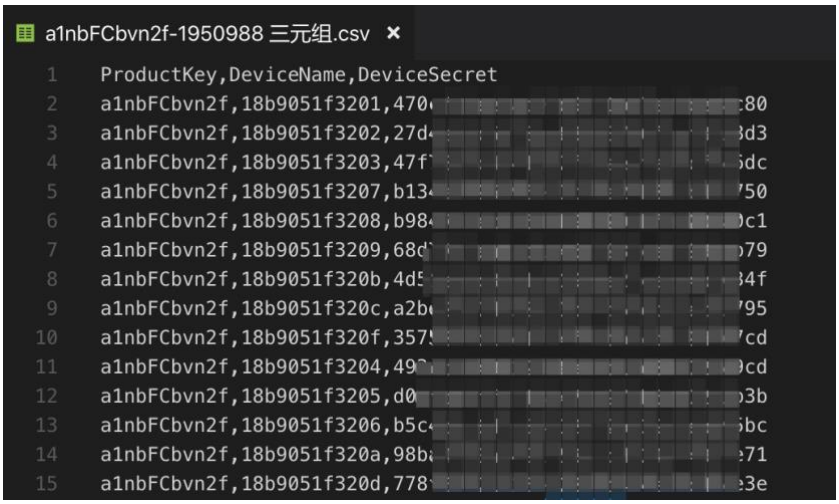


8.3 三元组扩展五元组

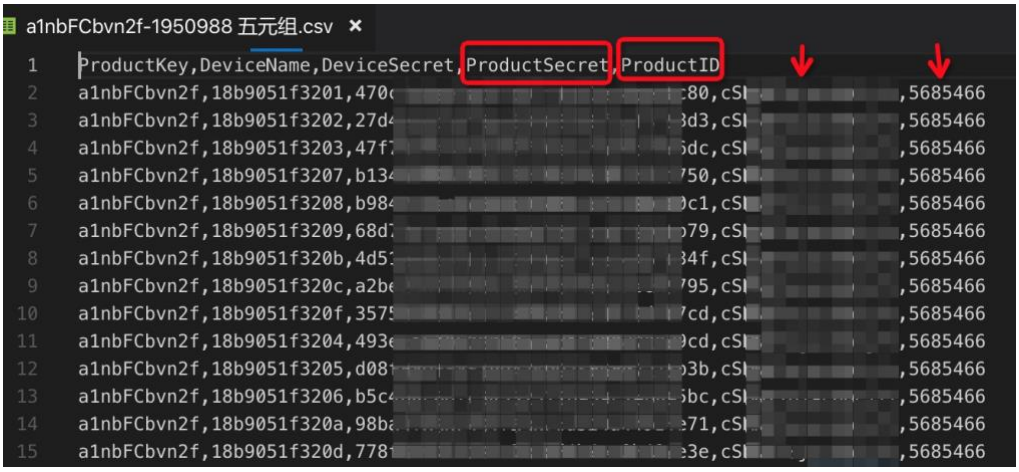
按以上操作步骤下载的 csv 格式文件都是三元组清单，需要通过文本处理才能生成符合 TG7100C 批量烧录工具需要的五元组清单的格式。

注意 csv 格式文件的处理要保持文本格式，通过文本编辑器可编辑。建议编写一个程序来处理。**不要通过 excel 编辑，这会将其转为 excel 的格式。**

下载后的三元组 csv 清单示例如下：



处理后的五元组 csv 清单示例如下，按文本格式增加 ProductSecret,ProductID 的内容。其中第一行的"ProductSecret,ProductID"拼写（含大小写）也严格保持一致。



8.4 五元组烧录

在平头哥芯片开放社区资源下载页面下载 TG7100C 批量烧写工具与文档《TG7100C 五元组量产工具使用说明》，完成包含五元组的 csv 文件导入到数据库后进行烧录。