ALIBABA CLOUD

阿里云

生活物联网平台（飞燕平台）

最佳实践

文档版本：20220531

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.Link Visual视频开发

1.1. 什么是Link Visual

Link Visual是生活物联网平台针对视频产品推出的增值服务，提供视频数据上云、存储、转发、AI计算等能力。

功能介绍

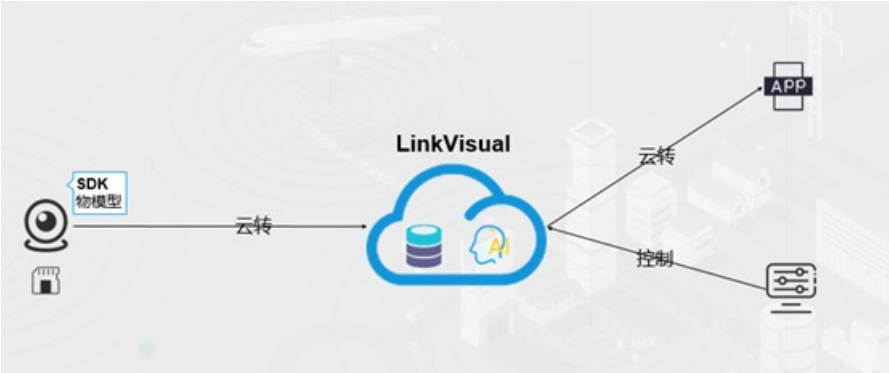
Link Visual提供的设备端SDK（支持各种标准的适配与统一），配合阿里云物联网标准化物模型，可实现最轻量级的设备上云连接。同时提供自有品牌App开发需要的API和SDK，可以为您打造一整套的设备连接云端、App开发控制等能力，再辅助云端转发、存储、视频AI等增值扩展服务，帮助您快速完成视频方案搭建，满足客户远程视频观看、存储、报警与控制需求等。

Link Visual主要提供以下功能。

- 云端摄像头视频直播
- 摄像头云端视频存储
- 云端、本地TF存储录像播放
- 语音对讲
- 远程摄像头控制
- 摄像头规则管理（报警、抓图、录像、检测识别等）
- 设备数据分析、云存储资源购买等运营管理功能

产品架构

Link Visual产品的功能链路及结构如下。



应用场景

Link Visual主要面向的场景包括：家用级别的安防监控、视频采集直播存储场景等。

1.2. Link Visual收费策略

本节介绍Link Visual服务的收费方式。以下内容仅供参考，实际收费请以账单为准。

设备

设备使用生活物联网平台的视频接入服务时，根据实际预计占用的带宽，您需要购买不同规格的视频激活码。详细参见激活码计费。

> **说明** 阿里云会统计实际带宽占用，如设备平均带宽超过其激活码的限制，阿里云将采取包括但不限于限制带宽、丢帧等措施。所以请根据实际使用场景选择合适规格的视频激活码。

消息通信

当设备日均消息超过3000条时，生活物联网平台对超出部分的消息收取消息通信费用，且按消息数计费。详细参见消息通信计费。（平台暂时不收取费用）

智能云存储

智能云存储是阿里云Link Visual为视频设备提供的增值服务，包括录像、图片的存储和查看，以及告警消息的提醒。对接使用智能云存储要求您在智能生活物联网平台上签署阿里云Link Visual视频云存储服务条款，相关API请参见视频服务云存储购买。

- 支付方式

  您在自有App端完成用户付款结算逻辑，比如集成支付宝移动端支付SDK。自有App调用云端购买接口，为您的设备开通智能云存储服务的同时，阿里云会从您的阿里云账号里扣款，并为设备开通智能云存储服务。因此，需要您确保阿里云账号的余额充足。如果扣款时阿里云账号余额不足，会出现开通服务失败的情况。

- 套餐规格

  智能云存储套餐分为标准套餐（适用于常供电设备）和门铃猫眼低功耗套餐两种。

  - 标准套餐

套餐类型	存储周期	月套餐		年套餐	
		中国内地（大陆）	中国香港、中国台湾、中国澳门及海外地区	中国内地（大陆）	中国香港、中国台湾、中国澳门及海外地区
事件云存储	1天	1元	2元	10元	20元
	3天	3元	6元	30元	60元
	7天	4元	8元	40元	80元
	14天	无	20元	无	200元
	30天	18元	36元	180元	360元

- 门铃猫眼低功耗套餐

套餐类型	存储周期	月套餐		年套餐	
		中国内地（大陆）	中国香港、中国台湾、中国澳门及海外地区	中国内地（大陆）	中国香港、中国台湾、中国澳门及海外地区
事件云存储	3天	0.9	1.8	9	18
	7天	1.8	3.6	18	36
	14天	2.8	5.6	28	56
	30天	5元	10元	50元	100元

> **说明** 门铃猫眼低功耗套餐会限制每个设备每天录制的事件录像个数以及时长。

智能云存储按照服务周期分为月套餐（按30天计）和年套餐（按365天计）。

智能云存储主要指事件云存储，事件云存储服务是指设备接入阿里云且网络良好的情况下，当设备检测到监控画面变化，包括但不限于有人走过，物品移动，光线变化等事件的时候，设备会自动录像并上传到阿里云。

事件云存储按照存储周期分为1天、3天、7天、14天、和30天。以7天循环事件年套餐举例，2019年1月1日，购买并立即生效，系统自动开始存储2019年1月1日至2019年1月7号时间段内产生的所有录像和图片。在2019年1月8号，系统会删除2019年1月1日的数据，并开始存储2019年1月8号产生的视频数据，依次类推。至2020年1月1日，服务到期，将不再存储新的数据（已存数据仍然可以查看）。至2020年1月8号，已存储数据将全部删除并不可访问。

- 赠送规则

  阿里云可以为购买视频激活码的每个量产设备赠送一份智能云存储套餐，但原则上每个设备只能领用一次，换一个用户账号绑定不可以再领取。考虑到您的消费类产品可能发生退回二次销售，允许一个设备最多在三个不同用户账号下各领用一次赠送套餐。您不得向用户宣传可以通过更换账号骗取赠送资源，否则阿里云有权立即收回赠送资源，并要求您支付所有赠送资源对应款项的三倍作为违约金。对于每个购买视频激活码的设备，阿里云可以赠送3个月7天的事件云存储套餐。赠送活动阿里云可随时停止。

- 购买和激活规则

  您自建的用户体系下，单个用户账号下可以同时购买多份智能云存储的月套餐和年套餐，多个有效套餐可以同时存在于一个用户账号下，但同一时刻只能有一个激活套餐。智能云存储套餐的激活可以选择立即生效还是延迟生效。立即生效会保留原套餐（原套餐按剩余天数保留，不满一天按使用了一天计算），但是立即切换到购买套餐上。延迟生效方式是新套餐在原套餐自然结束以后再生效。您可以基于用户需要允许同时拥有多个套餐的用户自行调整套餐激活顺序，默认是按照购买下单顺序激活。

  智能云存储套餐一旦购买立即生效。云存储作为虚拟商品，不支持退货，换货和退款，不退差价，可以停用。

  设备解绑以后，新用户账号绑定后不可以使用和访问原云存储的数据，新用户只能向您重新购买智能云存储。

  一个用户账号下的多个设备只能分别为每个设备购买智能云存储服务。

- 转移规则

您自建用户体系的用户，如果为设备购买了智能云存储服务，不管是否解绑了该设备，智能云存储服务的剩余期限（非内容）都可以转移到任何该用户账号绑定的其他购买了视频激活码的设备上。转移的条件如下：

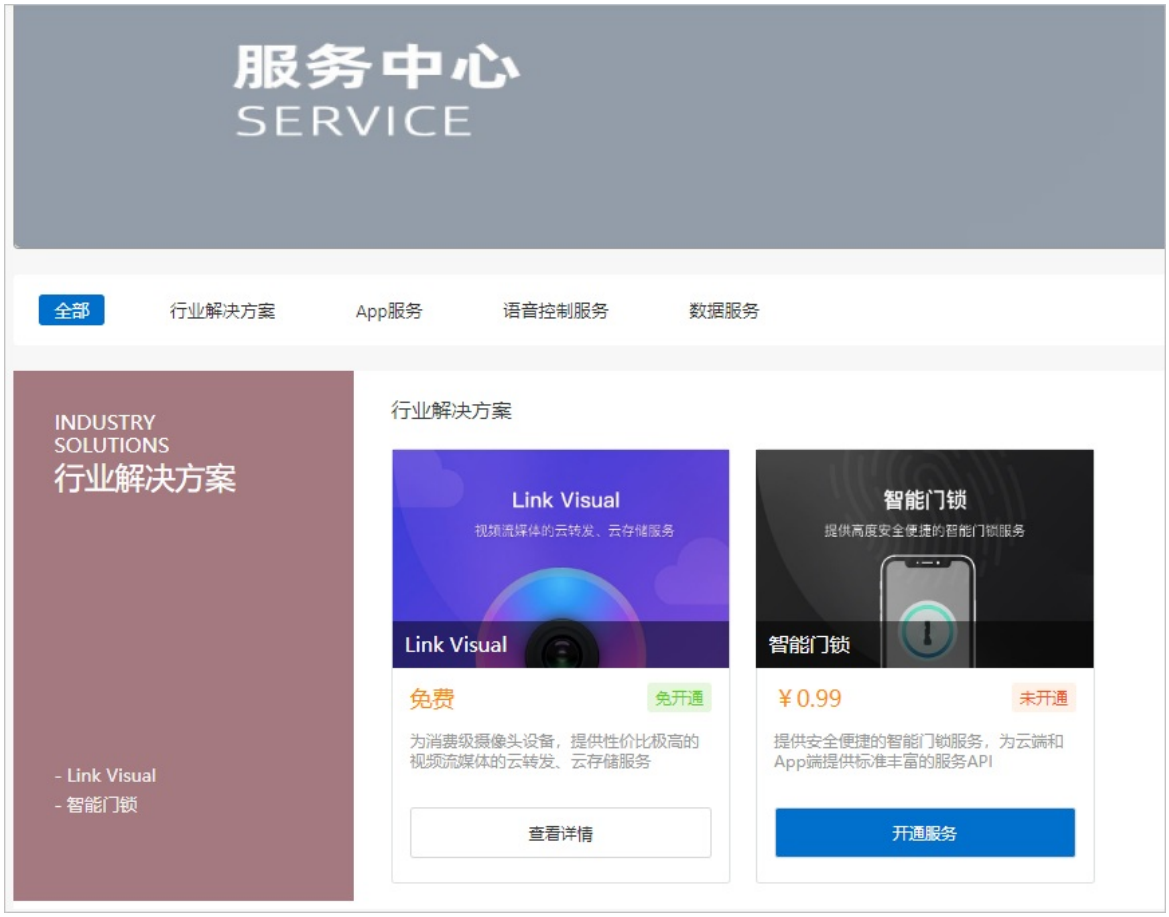
- 能转移的智能云存储服务必须是购买的并且在有效期内，赠送的智能云存储服务不可以转移。
- 能接受转移智能云存储服务的设备必须购买该用户账号绑定的LV设备，不可以在您自建用户体系内跨用户转移。

1.3. 快速体验Link Visual

您可以根据本文档快速体验Link Visual服务的开发流程，并熟悉Link Visual的能力。

一、开发产品

1. 登录生活物联网控制台。
2. 单击控制台主页面右下角的**服务中心**，并单击Link Visual对应的**查看详情**。

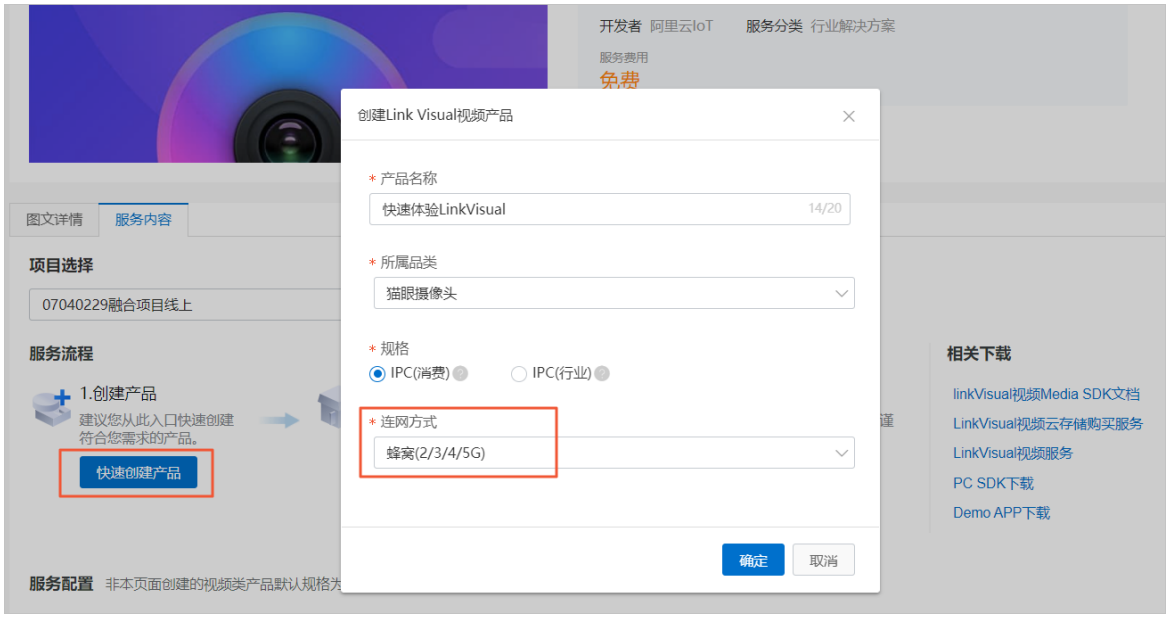


3. 从项目选择列表中，选择一个项目。

   若您当前没有已存在的项目，请单击**创建项目**来创建一个新项目。详细操作请参见创建项目。
4. 在**服务内容**页签中，下拉选择项目名称，并单击服务流程中的**快速创建产品**。

5. 配置视频产品的相关参数。

   快速体验阶段您需要将连网方式设置为蜂窝（2G/3G/4G）（正式接入设备时请按设备的实际情况配置）。



   产品创建成功后，页面自动跳转至产品的**功能定义**页面。

6. 定义产品的功能属性。

   生活物联网平台为视频产品提供了默认的功能定义，快速体验阶段您直接使用默认属性即可。

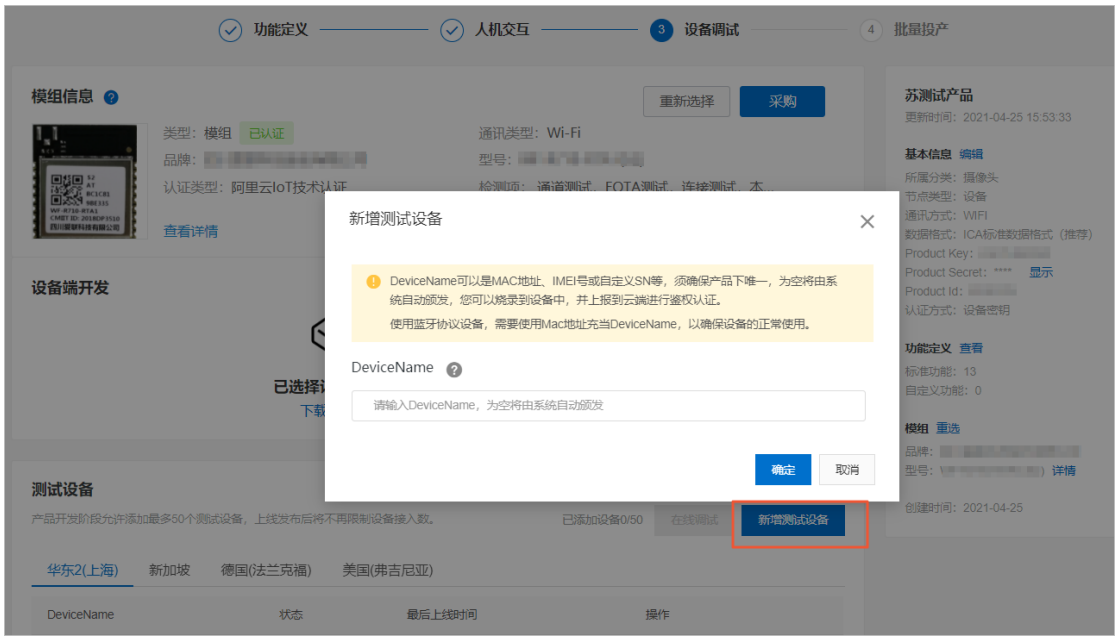
7. 单击**下一步：设备调试**，添加调试设备。

   i. 选择模组信息。

      快速体验阶段，您可以选择任意的模组信息（真正接入设备时请按实际情况选择）。

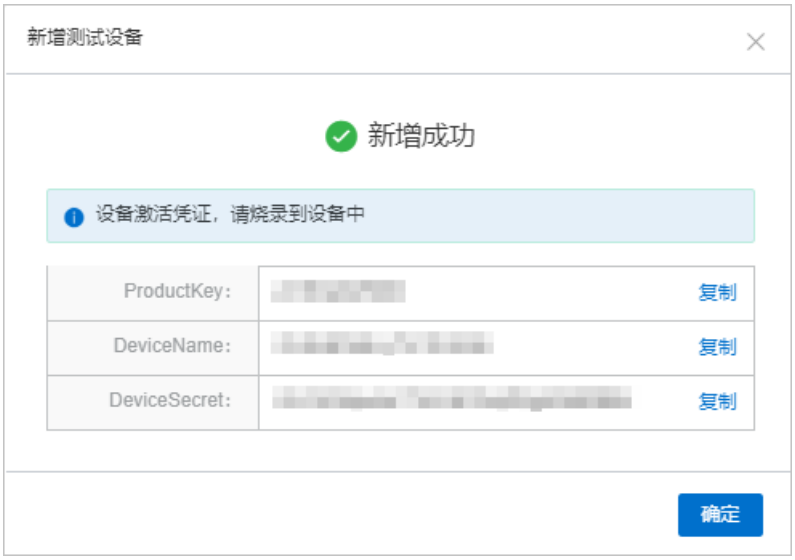
ii. 单击**新增测试设备**，弹出**新增测试设备**对话框。

添加测试设备后，可以免费使用平台提供的测试设备证书调试设备（每个产品最多可添加50个免费测试设备，测试设备的证书不能用于量产，仅供调试使用）。



iii. 单击**确定**，对话框中显示测试设备的激活凭证。

测试设备的激活凭证



8. 单击**下一步：人机交互**，选择App版本并获取配网二维码。

快速体验阶段，建议您使用公版App（云智能）来快速体验设备控制。

i. 打开使用公版App控制产品的开关。

ii. 单击**配网+App下载二维码**，获取配网二维码。

iii. 在文本框中，输入测试设备激活凭证图中的DeviceName，单击**生成二维码**。

配网+App下载的二维码

二、运行Link Visual Demo

生活物联网平台为您提供了Linux语言的Link Visual Demo，您可以根据以下步骤来运行该Demo，从而使用虚拟摄像头来体验Link Visual服务。

1. 下载Link Visual Demo。

   生活物联网平台提供了两种Link Visual Demo，请根据您的开发环境选择。

   - 基于Ubuntu的Link Visual Demo

     该Demo基于x86 64位Ubuntu 16.04系统上编译，在其他Linux版本上尚未验证过，推荐您安装相同的Ubuntu版本以规避兼容性问题。单击下载基于Ubuntu的Link Visual Demo

   - 基于Docker的Link Visual Demo

     Docker镜像提供更好的跨平台能力，您可以在Windows、Mac、Linux等操作系统上安装Docker软件，并在Docker系统上运行该Demo。单击下载基于Docker的Link Visual Demo

   > 说明 下载本Link Visual Demo，将默认您已同意本软件许可协议。

2. 运行Link Visual Demo程序。

   - Ubuntu的Link Visual Demo

```
# 下载得到文件link_visual_ipc_ubuntu_1.2.2.tar.gz
# 解压缩文件，并进入解压后的文件夹
$ tar -xf link_visual_ipc_ubuntu_1.2.2.tar.gz
$ cd link_visual_ipc_ubuntu_1.2.2
# 确认文件内容
$ ls
aac_h265_640
aac_h265_640.index
aac_h265_640.meta
aac_h265_768
aac_h265_768.index
aac_h265_768.meta
link_visual_demo
# 传入设备的激活凭证信息，并运行
$ ./link_visual_demo -p your_product_name -n your_device_name -s your_device_secret
```

- Docker的Link Visual Demo

```
# 下载得到文件link_visual_ipc_docker_1.2.2.tar.gz
# 导入docker镜像
$ docker load -i link_visual_ipc_docker_1.2.2.tar.gz
Loaded image: ubuntu:lv_1.2.2
# 运行镜像，此时会进入到镜像生成的容器中
$ docker run -it --rm ubuntu:lv_1.2.2 bash
# 进入link_visual目录
$ cd /link_visual
# 解压缩内容并进入
$ tar -xf sample.tar.gz
$ cd sample
# 确认文件内容
$ ls
aac_h265_640
aac_h265_640.index
aac_h265_640.meta
aac_h265_768
aac_h265_768.index
aac_h265_768.meta
link_visual_demo
# 传入设备的激活凭证信息，并运行
$ ./link_visual_demo -p your_product_name -n your_device_name -s your_device_secret
```

> **说明** 命令中的*your_productname*、*your_devicename*、*your_devicesecret*，需要替换为您的设备激活凭证信息 。请参见测试设备激活凭证。

3. 查看Link Visual Demo运行效果。
   - App上触发直播、点播等功能，观看App的播放情况。
   - App触发功能时，在运行Link Visual Demo的Docker或Ubuntu内查看设备实时日志。

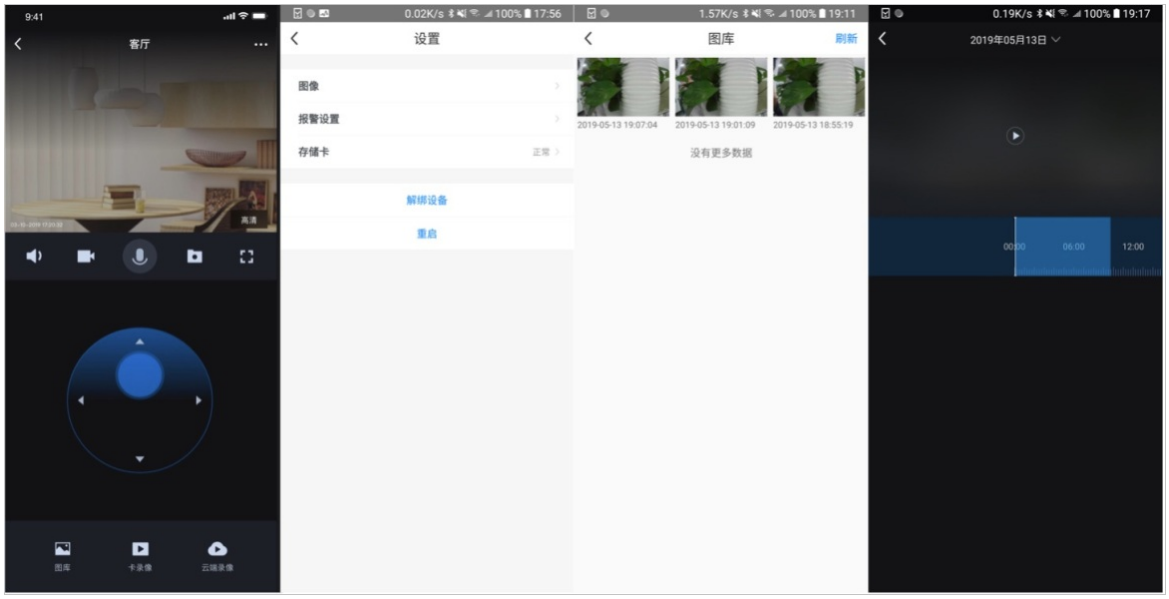
三、使用公版App体验Link Visual服务

1. 扫描生成的配网+App下载二维码图，下载公版App（云智能）。更多公版App的下载方式请参见云智能App介绍。

2. 使用下载的公版App（云智能），扫描生成的配网+App下载的二维码，绑定虚拟设备。

   虚拟设备绑定后，您就可以体验Link Visual的能力了。

   从公版App（云智能）设备列表进入到对应的摄像头设备后，您将看到如下界面（依次为直播、设置、图库和本地录像播放的界面）。



当前公版App中摄像头产品包含以下功能。

功能	描述
摄像头设置	包含摄像头日夜模式、视频画面翻转、报警开关（设备报警的总开关，当触发报警时设备将上传一张报警照片到云端，用户可以在图库中进行查看）、移动侦测灵敏度设置、报警频率设置（控制摄像头上报报警事件的频率）、报警时段设置（摄像头在哪些时段进行报警）、存储卡状态、容量展示和格式化存储卡（格式化掉摄像头内存储卡上的所有文件）等。
直播清晰度切换	根据您设备支持的种类进行清晰度切换。
直播声音开关	该开关只控制手机上直播的音量，并不影响摄像头采集声音。
直播录屏	从当前直播流上截取，您可以根据需求直播录屏（录制一段时间的直播录像），产生的文件可以在手机相册内查看。
直播截图	从当前直播流上截取，您可以根据需求直播截图（获取当前直播画面的图片），产生的文件可以在手机相册内查看。
语音对讲	您可以通过这个功能跟摄像头方人员进行语音聊天。
摄像头转动控制	通过直播页面的转向盘，控制摄像头进行上下左右转动，由此您可以看到需要的直播画面。
图库	内展示的设备报警产生的报警图片（具体的上报开关，灵敏度，频率以及时段在设置中“报警设置”中设置）。
卡录像	展示了一定时间内的摄像头内存储卡保存的历史录像，您可以看到任意时段设备保存的卡录像

功能	描述
云端录像	查看由设备端产生报警而生成并上传到云端的录像。

1.4. 开发指南

1.4.1. 直播功能

本文档主要介绍Link Visual的直播功能在控制台上定义的物模型，以及设备端与App端的工作流程。

功能介绍

直播视频支持H264/H265，音频支持G711A/AAC_LC。采用RTMP云转+P2P混合方式，在保证直播稳定性的同时降低成本。

- 视频播放

  Link Visual App SDK提供直播播放器，集成请参见Android Link Visual Media SDK和iOS Link Visual Media SDK。

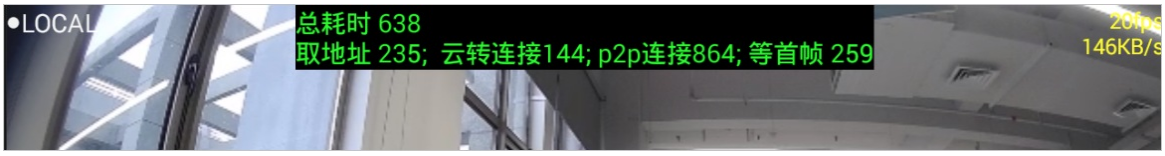
- P2P支持

  P2P支持需要同时接入Link Visual Device SDK和Link Visual App SDK。播放时会尽可能的尝试P2P连接，以减小成本开销。后台会对P2P的成功率进行统计，发现P2P成功率出现异常会通知开发者检查原因，若不修复有可能拒绝服务。

  App端P2P依赖生活物联网平台的长连接通道，需要App初始化长连接通道SDK，并完成长连接通道与账号绑定。请参见Android长连接SDK和iOS长连接SDK。

  - 确认设备端P2P是否集成成功

    设备端SDK集成完毕后，确保云智能App（开发版）和IPC设备连在同一个WiFi下，将云智能App（开发版）的测试信息开关打开后，进入直播页面，左上角RELAY代表RTMP云转，LOCAL代表P2P直连（局域网直连），SRFLX代表P2P穿透（公网穿透）。

  - 确认自有App P2P是否集成成功

    在首帧后5S，调用播放器的播放功能，获取播放器当前流的连接类型接口，来获取当前流类型。如RELAY代表RTMP云转；LOCAL代表P2P直连；SRFLX代表P2P穿透。

- 视频加密

  为保证视频数据安全，SDK支持对音视频帧的加密，推荐App端开启全链路加密。App端可调用物模型API（Android/iOS）设置物模型属性（EncryptSwitch）开启或关闭加密。

- 清晰度切换

  App端可通过调用物模型API（Android/iOS）查询主辅码流清晰度物模型属性（StreamVideoQuality/SubStreamVideoQuality）用于UI上展示当前播放码流清晰度，App端设置该物模型属性来改变码流清晰度，设备收到物模型属性设置后，切换码流清晰度重新推流，播放器SDK已支持码流自适应。

- 截图和录屏

播放器SDK提供了截图和录屏接口，详细参见Android Link Visual Media SDK和iOS Link Visual Media SDK。

- 减少首帧延迟（强制I帧）

  在有新的播放端观看时，会要求设备立即编码I帧，从而减少直播首帧延迟。参考设备端和App的文档响应和发起强制I帧。

物模型介绍

开发直播功能时，您需要配置的物模型如下。

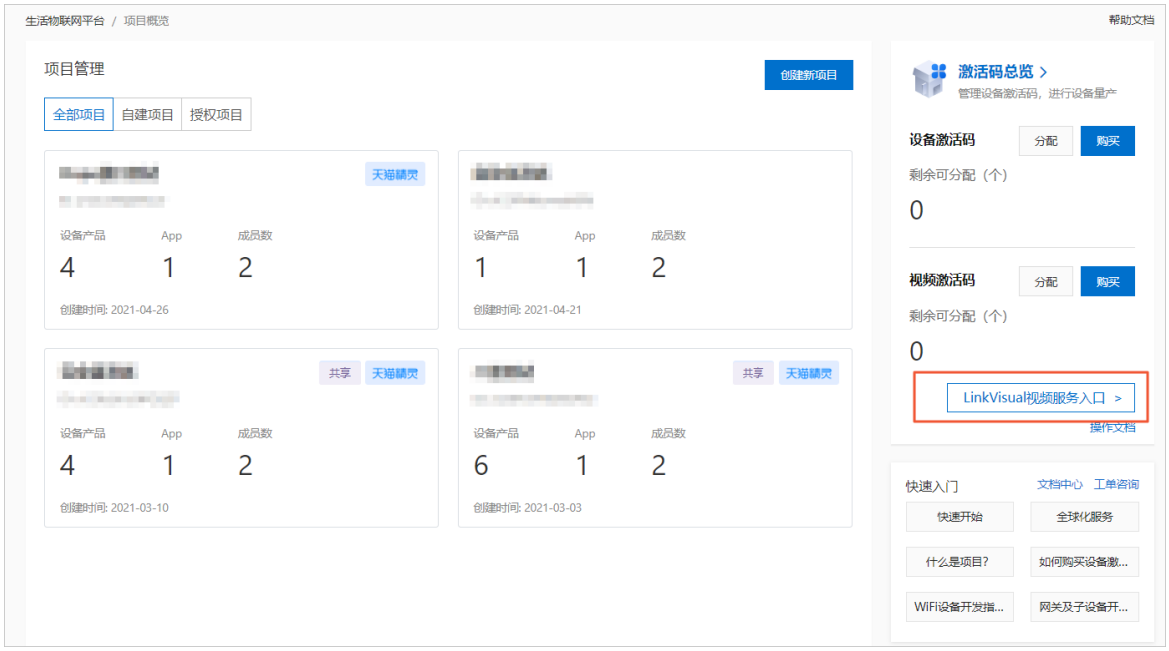
标识符	功能类型	功能名称	描述	控制台勾选	开发指南
StartPushStreaming	服务	开始直播	通知设备开始直播RTMP推流，当有播放端观看或者需要做云存录制时触发，同一码流已有推流则不再触发	是	该物模型无需额外开发
StopPushStreaming	服务	停止直播	通知设备停止直播RTMP推流	是	该物模型无需额外开发
StartP2PStreaming	服务	开始P2P直播	通知设备开始P2P直播	是	该物模型无需额外开发
EncryptTypeList	属性	加密类型列表	设备端上报支持的加密类型	是	该物模型无需额外开发
StreamVideoQuality	属性	主码流视频质量	用于做主码流清晰度切换，要求开机及状态变更时上报	按需	App和设备开发者需要关注并处理该物模型
SubStreamVideoQuality	属性	辅码流视频质量	用于做辅码流清晰度切换，要求开机及状态变更时上报	按需	App和设备开发者需要关注并处理该物模型
EncryptSwitch	属性	加密开关	加密开关，建议设备实现时默认开启，要求开机及状态变更时上报	是	App和设备开发者需要关注并处理该物模型

1.4.2. 创建产品并配置App

正式接入视频设备的第一步是在生活物联网平台上创建产品，并完成产品的开发工作。

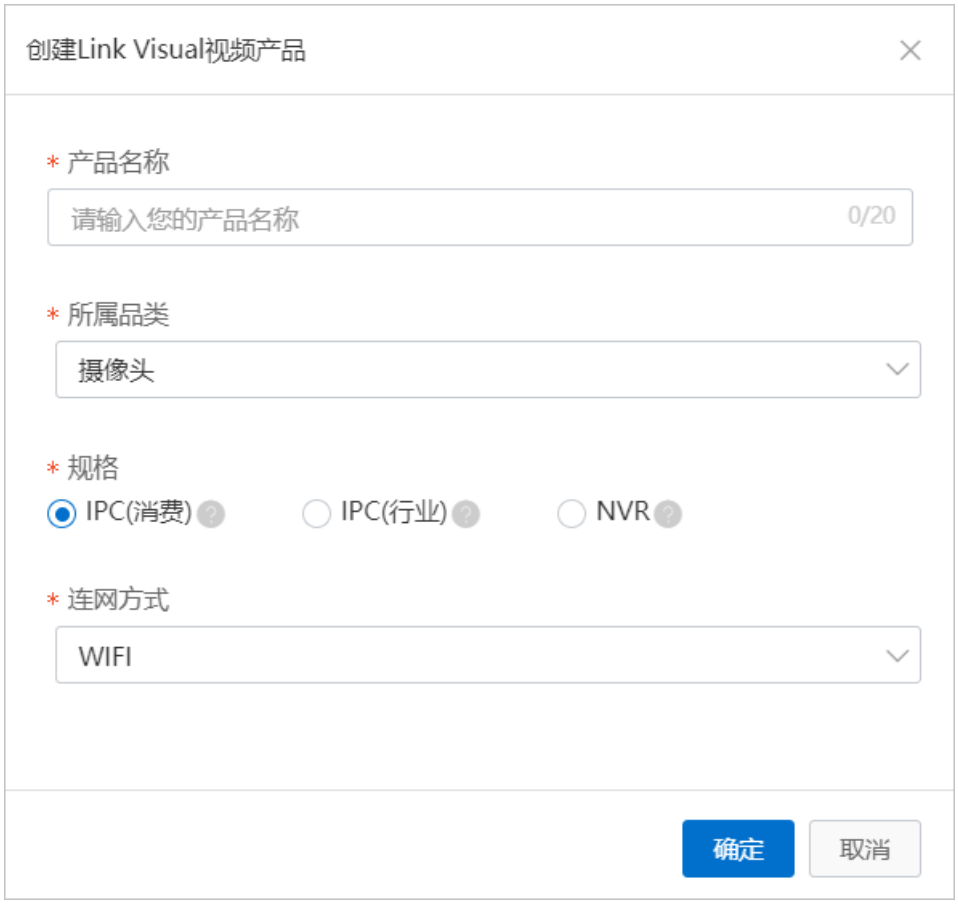
一：创建产品

1. 登录生活物联网控制台。
2. 在控制台主页面，单击**LinkVisual视频服务入口**。



3. 在**服务内容**页签中，下拉选择项目名称，并单击服务流程中的**快速创建产品**。

4. 配置视频产品名称，并根据实际情况选择产品的**所属品类**、**规格**、**连网方式**。



视频产品分三种品类：摄像头、可视门铃、猫眼摄像头。其中摄像头包括IPC消费、IPC行业、NVR三种规格，可视门铃以及猫眼摄像头包括IPC消费、IPC行业两种规格。规格的介绍如下。

- IPC消费：用于个人消费场景（家居）的视频监控摄像头接入

  该规格创建的摄像头设备，不接入网关，默认连网方式为WiFi，可修改连网方式（WiFi、蜂窝、以太网、其他）。

- IPC行业：用于SMB场景（如店铺监控、别墅监控等）的视频监控摄像头接入

  该规格创建的摄像头设备，不接入网关，默认连网方式为以太网，可修改连网方式（WiFi、蜂窝、以太网、其他）。

- NVR：用于NVR（Network Video Recorder，网络硬盘录像机）设备的视频接入，NVR为网关设备
  - 仅品类为摄像头的产品，支持NVR规格。
  - NVR连网方式固定为蜂窝（2G/3G/4G），不可修改。
  - NVR的通道数最大为128。

- NVR创建成功后，系统还会自动创建一个IPC的摄像头产品（该IPC连网方式固定为以太网），且连接在NVR网关节点下（如下图所示）。

> **注意** 自动创建的子产品有量产数量限制（上限为50万），一旦超过限制则没法继续量产。所以一般建议您创建了NVR产品后，通过工单向我们提供自动创建的子产品的ProductKey，我们为您调整量产限额。

产品名称	品类	激活码规格	连网方式	节点类型	接入网关	已量产	告警消息	产品开发
文档视频NVR	网络硬盘录像机	NVR	蜂窝(2G/3G/4G)	网关	否	0	/	查看详情
└ IPC(文档视频NV...	摄像头	IPC(NVR连接)	--	设备	是	0	配置	查看详情

> **说明** 不是通过LinkVisual视频服务入口创建的视频产品，也会显示在服务配置的产品列表中，此时激活码规格默认为无（如下图所示）。您可以将产品切换到IPC（消费）或者IPC（行业）规格。一旦切换则无法修改。

二：定义产品功能

1. 单击**查看详情**，进入产品的**功能定义**页面。

2. 根据产品的功能属性，添加物模型。

   您可以在标准功能中添加标准功能，也可以在自定义功能中自行定义。具体操作，请参见定义产品功能。

三：配置人机交互

1. 单击**下一步**，在**人机交互**页面配置人机交互的参数。
2. 选择App版本。
   - 如果您选择公版App，打开使用公版App控制产品的开关。

- 如果您选择自己开发自有品牌App，保持默认即可。

3. 配置人机交互。详细操作，请参见配置介绍。

   如果您需要使用告警功能，除了在**人机交互**页面配置设备告警（具体操作，请参见配置设备告警）外，您还需要配置视频告警规则。

   当触发视频告警后，平台向终端用户发送消息提醒。视频告警规则分为以下四类。

   - 云存储到期告警

     当云存储快到期时触发告警，可设置时间为：到期前7天、3天、1天、当天。

   - 侦测告警

     当AI侦测到以下变化时触发告警，变化包括：移动、声音、宠物、越界、区域入侵、跌倒、人形、人脸、笑脸、异响、哭声、笑声等。

   - 智能告警

     智能告警相比侦测告警提升了检测精度，同时支持多种告警同时上报，且侦测的变化类型更丰富，例如：物品遗留、人群密度估计、非机动车乱停、吸烟检测等，共计39种。

     > **说明** 使用该告警规则，您需使用1.4.0及以上版本的Link Visual设备端SDK。

   - 连续云存异常告警

     终端用户已经购买云存或已经领取免费云存时，如果连续云存持续一小时及以上出现录像丢失，则触发告警。

   > **说明** 只有通过**服务中心**中的**快速创建产品**入口，创建的IPC产品（包括IPC消费、IPC行业、创建NVR时自动创建的IPC三种），才支持设置视频告警规则。

   请您根据以下步骤配置视频告警规则。

   i. 在**服务配置**页签中，单击产品列表中告警消息对应的**配置**。

   ii. 单击**新建告警规则**。

iii. 配置视频告警规则。

请您根据实际情况配置告警规则。参数描述，请参见配置设备告警。

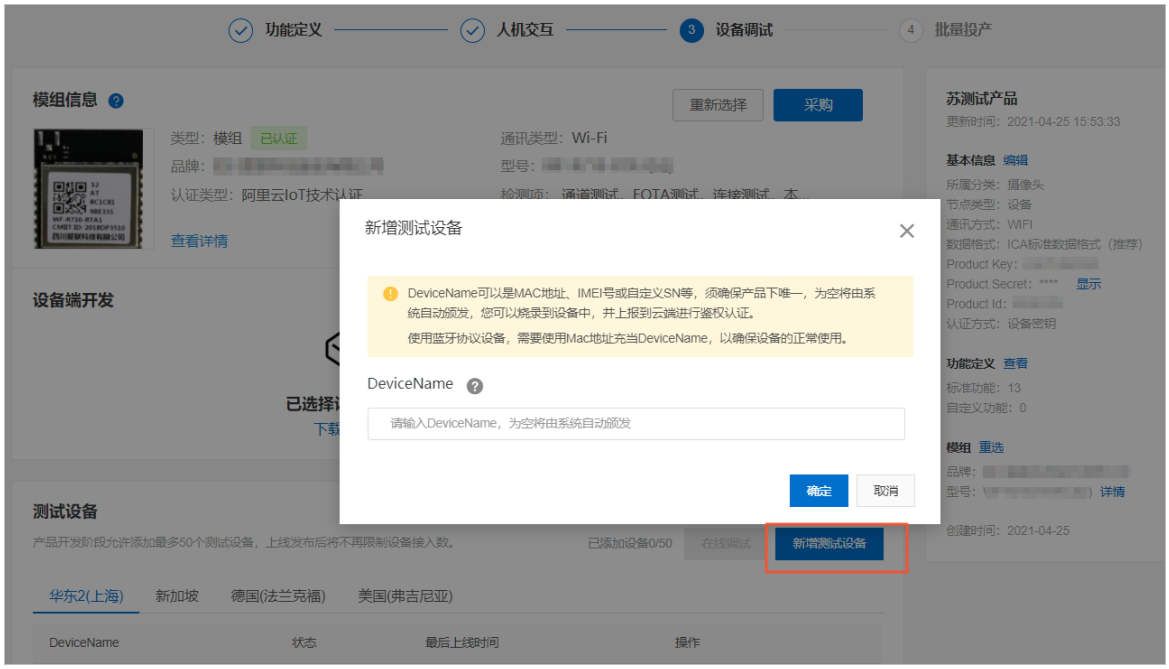
说明

- 相比设备告警，视频告警消息不支持推送到生活物联网的控制台。
- 如果您在**人机交互**页面已配置了侦测类告警（即事件AlarmEvent），建议您在**视频告警消息**页面重新配置，并在启用**视频告警消息**的告警后，将原来**人机交互**页面中的侦测告警停用，否则可能会同时收到两条告警消息。

四：添加测试设备

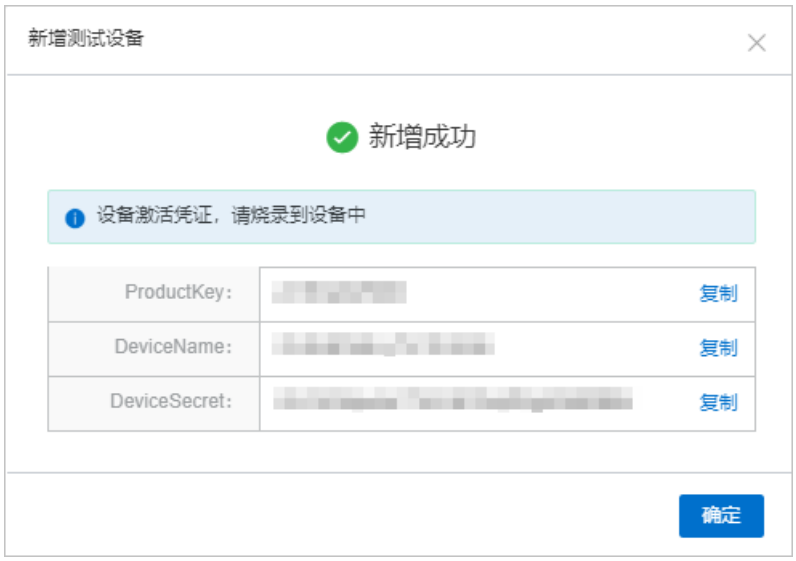
在设备开发调试阶段时，您可以免费使用平台提供的测试设备证书，来调试设备（每个产品最多可添加50个免费测试设备，测试设备的证书不能用于量产，仅供调试使用）。

1. 单击**下一步**，在**设备调试**页面添加测试设备。
2. 根据实际情况，选择模组信息。
3. 单击**新增测试设备**，弹出**新增测试设备**对话框。



> **说明** 如果调试NVR规格的产品时，您需要添加一个NVR产品的测试设备和一个IPC（NVR直连）产品的测试设备。

4. 单击**确定**，对话框中显示测试设备的激活凭证。



1.4.3. 设备端开发

您可以根据生活物联网平台的设备端SDK开发视频设备。

前提条件

已在控制台上创建产品，并完成产品的功能定义和功能参数配置，请参见创建产品并配置App。

操作步骤

1. 获取设备端SDK。
   - Linux版本设备端SDK：请参见Linux SDK开发指南。
   - Android版本设备端SDK：请参见Android SDK开发指南。
2. 结合以下文档，完成摄像头设备侧开发。
   - 功能开发文档：直播功能
   - Link Visual设备端SDK文档：IPC（Linux）、IPC（Android）
   - 常见问题：Link Visual的常见问题

1.4.4. 自有App开发

您可以根据生活物联网平台的客户端SDK开发自有品牌App。

前提条件

- 已完成Link Visual设备端开发，请参见设备端开发。
- 已在控制台上创建产品，并完成产品的功能定义和功能参数配置，请参见创建产品并配置App。

操作步骤

1. 登录生活物联网控制台。
2. 单击Link Visual产品所在的项目名称，进入项目主页面。
3. 单击**创建自有品牌App**，创建自有App并下载SDK，详细操作请参见创建自有App。
4. 结合以下文档，开发自有App。
   - Demo App文档：Android、iOS
   - 功能开发文档：直播功能
   - 云存储购买对接文档：LinkVisual视频云存储购买服务
   - Link Visual客户端SDK文档：Android、iOS、PC
   - Link Visual API文档：Link Visual视频服务
   - 常见问题：Link Visual的常见问题

1.4.5. 量产设备

产品功能开发调试完成后，进入量产阶段。请您根据以下步骤量产Link Visual设备。

前提条件

- 已在控制台上创建产品，并完成产品的功能定义和功能参数配置，请参见创建产品并配置App。
- 已完成Link Visual设备端开发，请参见设备端开发。

一、购买激活码

1. 登录生活物联网控制台。

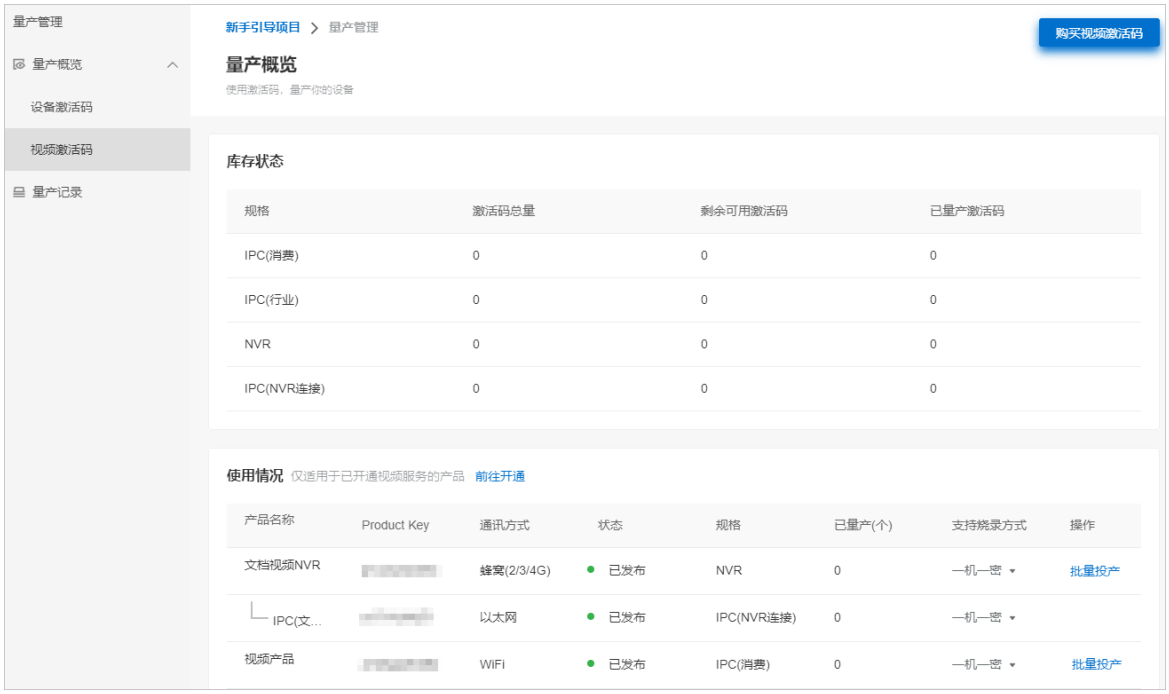
2. 选择Link Visual产品所在的项目，并单击**服务中心**。
3. 找到Link Visual服务，单击**查看详情**。

4. 单击服务流程中的**购买视频激活码**，为每一台摄像头设备，需购买一个视频激活码。详细介绍参见激活码计费。

二、量产设备

1. 在**Link Visual服务详情**页面的服务流程中，单击**量产管理**。

2. 在**视频激活码**的使用情况中，单击产品对应的**批量投产**。



说明

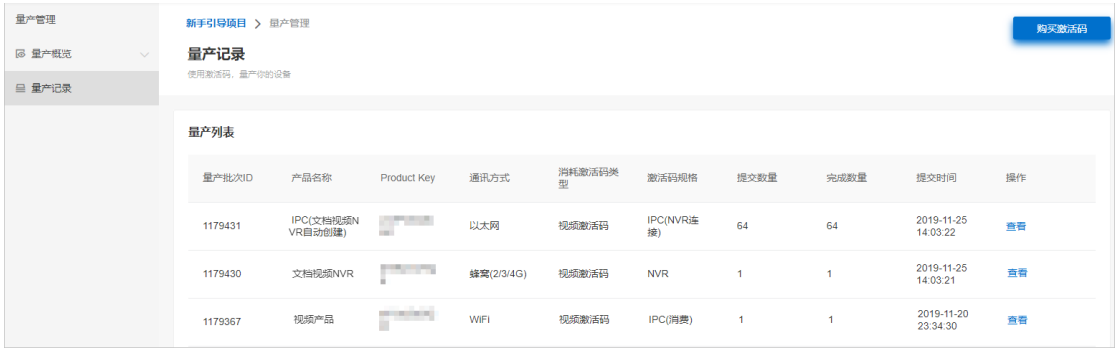
- 生活物联网平台支持一机一密（默认方式）和一型一密两种烧录方式，如果您要更换烧录方式请参见量产设备。
- 量产NVR产品前，需完成NVR产品的发布和对应的IPC（NVR连接）产品的发布。
- 量产NVR产品时，同时量产对应的IPC（NVR连接）产品。不可以单独量产IPC（NVR连接）产品。

3. 配置量产数额。

单击**确定**后，页面跳转至**量产记录**。

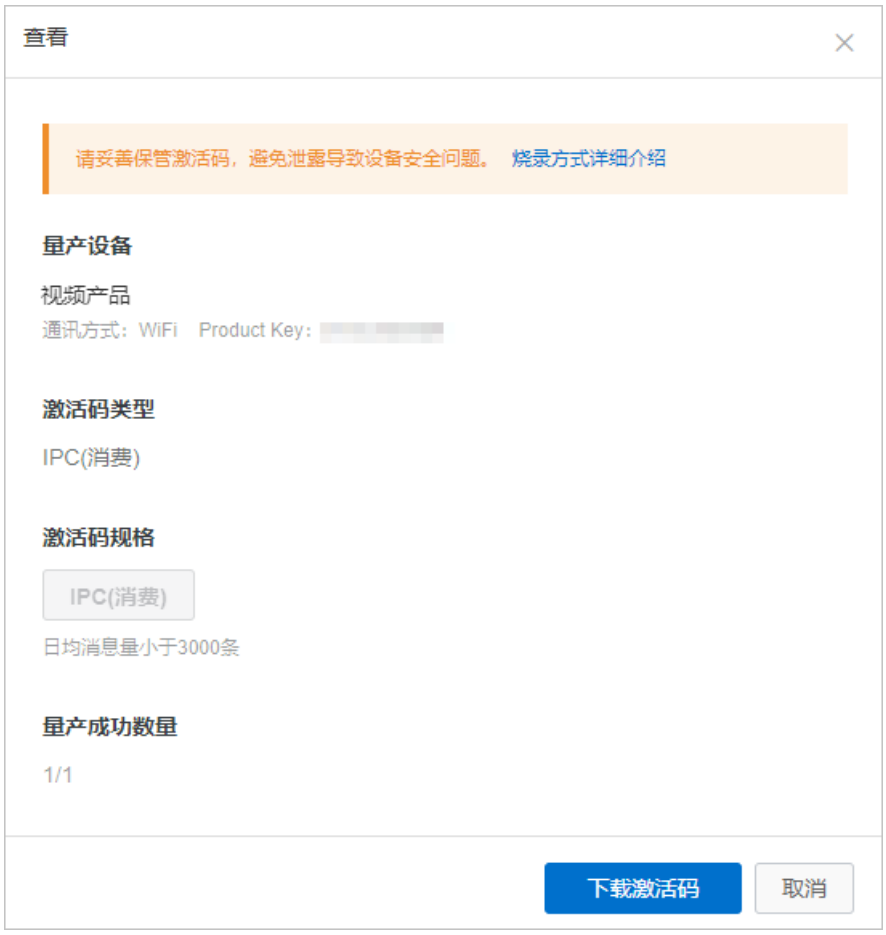
4. 下载设备证书至本地。

i. 在**量产记录**，单击产品对应的**查看**。



> **说明** 如果是NVR产品，您只需选择NVR产品对应的查看，下载NVR产品与IPC（NVR连接）合并的设备证书。

ii. 单击**下载激活码**，将设备证书的Excel文件下载到本地。



说明 如果是NVR产品，单击**下载合并设备证书**，将NVR产品与IPC（NVR相连）产品的设备证书合并下载到本地。

后续步骤

设备量产后，您便可以将设备凭证烧录到设备中。具体烧录方法请您联系设备所选模组的厂商获取。

2.配网开发最佳实践

2.1. App端Native配网开发实践

2.1.1. iOS App Native开发配网

生活物联网平台已提供了一套完整的配网页面，如果您希望自己使用Native开发自己的定制化配网页面，可以阅读本文，使用配网SDK进行开发。

背景信息

配网SDK提供了把Wi-Fi设备配置上家庭路由器以及局域网内已联网设备的发现能力，具体方案包括一键广播配网（P2P）、手机热点配网、智能路由器配网和设备间相互配网（以下简称零配）等。各配网方式介绍参见Wi-Fi设备配网方案介绍和配网SDK。

设备热点配网

1. 设置待配网设备信息。
   i. 选择配网方式。

      设备热点支持指定产品型号productKey进行配网，也支持不指定型号直接开始配网（详细参见下方代码）。

      获取产品型号productKey的方式如下。

      - 通过直接调用云端接口获取产品列表（非配网SDK提供接口）或App直接展示产品列表，用户选择后确定待配设备productKey。
      - 通过扫描产品二维码，获得设备ProductKey。

   ii. 调用SDK setDevice设置待配信息。
2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeSoftap，并调用SDK startAddDevice接口开始配网。
3. 配网中，处理回调信息。

   配网中，收到notifyProvisionPrepare回调后，提醒用户手动切换到设备热点，切换完成后调用SDK toggleProvision方法传入当前连接路由器的ssid和password。
4. 监听配网结果。

```
/**
* 第一步：设置待配网设备信息
*/
//方式一：指定productKey方式
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
model.productKey = @"xxxx";
model.linkType = ForceAliLinkTypeSoftap;
[kLkAddDevBiz setDevice:model];
//方式二：不指定型号
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
model.protocolVersion = @"2.0";
model.linkType = ForceAliLinkTypeSoftap;
[kLkAddDevBiz setDevice:model];
//若希望自定义设备热点前缀，例如前缀为"demo_"
```

```
model.softApSsidPrefix = @"demo_";
/**
* 第二步：开始配网
*/
[kLkAddDevBiz startAddDevice:self];
#pragma mark - 回调方法
- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
}
/*
 设备热点状态回调，根据不同状态进行提醒。
status 状态码
1=提示应该切换到设备热点;
2=已经切换到设备热点
3=已发送数据（dic里面会有"token"）
4=应该切换回路由器
5=已经切换回路由器
*/
- (void)notifyProvisioningNoticeForSoftAp:(int)status withInfo:(NSDictionary *)dic
{
    NSLog(@"notifyProvisioningNoticeForSoftAp,%d,%@",status,dic);
}
- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode
{
    NSLog(@"notifyProvisionPrepare callback guide code : %ld", guideCode);
    if(guideCode == LKPGuideCodeWithUserGuideForSoftAp){
        /**
        * 第三步：提醒用户输入切换到设备热点，切换完成后调用toggleProvision，参考inputSsidAndP
assword
        */
    }
}
- (void)inputSsidAndPassword
{
    NSSstring *ssid = @"example ssid";
    NSString *password = @"1qaz@WSX";
    NSInterger timeout = 60;(单位秒,s);
    [kLkAddDevBiz toggleProvision:ssid pwd:password timeout:timeout];
}
-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback(正在进行配网...) ");
}
/**
* 第四步：监听结果回调
*/
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:
(NSError *)provisionError
{
    NSLog(@"配网结果：%@",candDeviceModel);
}
```

蓝牙辅助配网

1. 配网SDK版本。

   确认Podfile依赖的配网SDK版本在 1.11.5及以上，并接入了蓝牙SDK。

```
# 配网SDK依赖
pod 'IMSDeviceCenter', '1.11.5'
# 蓝牙SDK依赖
pod 'IMSBreezeSDK', '1.6.9'
```

2. 设置待配网设备信息。

   i. 获取并设置待配设备的型号ProductKey和productID信息。

   获取方式如下。

   - 通过SDK本地发现接口IMLLocalDeviceMgr搜索周边蓝牙辅助配网设备，取得设备信息。
   - 通过直接调用云端接口获取产品列表（非配网SDK提供接口）或App直接展示产品列表，用户选择后确定待配设备productKey及 productID。
   - 通过扫描二维码获得待配设备信息，包含ProductKey及productID。

   ii. 调用SDK setDevice设置待配信息。

3. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeBLE，并调用SDK startAddDevice接口开始配网。

4. 配网中，处理回调信息。

   配网中，收到notifyProvisionPrepare回调后，提醒用户手动切换到设备热点，切换完成后调用SDK toggleProvision方法传入当前连接路由器的ssid和password。

5. 监听配网结果。

   示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
//方式一：通过本地发现获取
[[IMLLocalDeviceMgr sharedMgr] startDiscovery:^(NSArray *devices, NSError *err) {
        //devices 为 IMLCandDeviceModel 对象array，
        // 可根据 IMLCandDeviceModel 中的 devType 区分待配网设备联网类型：
        // devType 为@"ble_subtype_2" 代表蓝牙辅助配网设备*/
      //过滤获取出需要的蓝牙辅助设备数据 productId,通过prductId可通过 thing/productInfo/quer
yProductKey 接口查询到productKey
        IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
        model.productKey = @"xxxx";
        model.productId = @"xxxx";
        model.linkType = ForceAliLinkTypeBLE;
        [kLkAddDevBiz setDevice:model];
    }];
//方式二：其他获取方式设置
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
model.productKey = @"xxxx";
model.productId = @"xxxx";
model.linkType = ForceAliLinkTypeBLE;
[kLkAddDevBiz setDevice:model];
/**
* 第二步：开始配网
*/
```

```
[kLkAddDevBiz startAddDevice:self];
#pragma mark - 回调方法
- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
}
- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode
{
    NSLog(@"notifyProvisionPrepare callback guide code : %ld", guideCode);
    if(guideCode == LKPGuideCodeOnlyInputPwd){
        /**
        * 第三步：配网中传入Wi-Fi信息
        */
        NSSstring *ssid = @"example ssid";
        NSString *password = @"example pwd";
        NSInterger timeout = 60;(单位秒,s);
        [kLkAddDevBiz toggleProvision:ssid pwd:password timeout:timeout];
    }
}
-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback(正在进行配网...) ");
}
/**
* 第四步：监听结果回调
*/
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:
(NSError *)provisionError
{
    NSLog(@"配网结果：%@",candDeviceModel);
}
```

一键配网

1. 设置待配网设备信息。
   i. 获取并设置待配设备的型号ProductKey信息。

      获取方式如下。

      - 通过直接调用云端接口获取产品列表（非配网SDK提供接口）或App直接展示产品列表，用户选择后确定待配设备productKey及 productID。
      - 通过扫描二维码获得设备ProductKey。

   ii. 调用SDK setDevice设置待配信息。
2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeBroadcast，并调用SDK startAddDevice接口开始配网。
3. 配网中，处理回调信息。

   配网中，收到notifyProvisionPrepare回调后，提醒用户手动切换到设备热点，切换完成后调用SDK toggleProvision方法传入当前连接路由器的ssid和password。
4. 监听配网结果。

   示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
model.productKey = @"xxx";
model.linkType = ForceAliLinkTypeBroadcast;
[kLkAddDevBiz setDevice:model];
/**
* 第二步：开始配网
* 设置待配信息，开始配网
*/
[kLkAddDevBiz startAddDevice:self];
#pragma mark - 回调方法
- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
}
- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode
{
    NSLog(@"notifyProvisionPrepare callback guide code : %ld", guideCode);
    if(guideCode == LKPGuideCodeOnlyInputPwd){
        /**
        * 第三步：配网中传入Wi-Fi信息
        */
        NSSstring *ssid = @"example ssid";
        NSString *password = @"example pwd";
        NSInterger timeout = 60;(单位秒,s);
        [kLkAddDevBiz toggleProvision:ssid pwd:password timeout:timeout];
    }
}
-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback(正在进行配网...) ");
}
/**
* 第四步：监听结果回调
*/
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:
(NSError *)provisionError
{
    NSLog(@"配网结果：%@",candDeviceModel);
}
```

零配配网

1. 设置待配网设备信息。
   i. 获取并设置待配设备的信息。

      获取方式为通过SDK本地发现接口 IMLLocalDeviceMgr 搜索周边待配零配设备，取得设备信息。

   ii. 调用SDK setDevice设置待配信息。
2. 开始配网。

   调用SDK startAddDevice接口开始配网。

3. 配网中，处理回调信息。

   配网中，收到notifyProvisionPrepare回调后，提醒用户手动切换到设备热点，切换完成后调用SDK toggleProvision方法传入当前连接路由器的ssid和password。

4. 监听配网结果。

   示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
[[IMLLocalDeviceMgr sharedMgr] startDiscovery:^(NSArray *devices, NSError *err) {
        //devices 为 IMLCandDeviceModel 对象array，
        // 可根据 IMLCandDeviceModel 中的 addDeviceFrom 为"ZERO_DEVICE" 过滤出零配待配设备信息 model
        [kLkAddDevBiz setDevice:model];
    }];
/**
* 第二步：开始配网
* 设置待配信息，开始配网
*/
[kLkAddDevBiz startAddDevice:self];
#pragma mark - 回调方法
- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
}
-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback(正在进行配网...) ");
}
/**
* 第四步：监听结果回调
*/
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:(NSError *)provisionError
{
    NSLog(@"配网结果：%@",candDeviceModel);
}
```

设备绑定

本SDK提供的获取绑定token的接口。绑定接口非本SDK提供。

1. 获取绑定token。

   可通过以下方式获取绑定使用的token。

   - 调用本地发现接口，返回的已配设备列表设备信息中包含token。

     该方式请参照零配的本地发现接口调用示例。

   - 主动调用SDK接口获取token。

     绑定token。使用设备的 ProductKey、DeviceName、Token 调用绑定接口进行绑定。样例代码如下。

```
/**
* 第一步：获取绑定token
*/
// self.productKey 和 self.deviceName 是配网成功后返回的物模型中的 productKey 和 deviceNa
me
[[IMLLocalDeviceMgr sharedMgr] getDeviceToken:self.productKey deviceName:self.deviceN
ame timeout:20 resultBlock:^(NSString *token, BOOL boolSuccess) {
    NSLog(@"主动获取设备token：%@，boolSuccess：%d", token, boolSuccess);
    if(token){
        // 拿到绑定需要的token
        /**
        * 第二步：调用绑定接口
        */
        //TODO 用户根据具体业务场景调用
    } else{
        NSLog(@"获取token失败(超时)");
    }
}];
```

- 使用本地发现的设备信息进行绑定的时候，如果出现超时（如获取token之后过了很久才去绑定），可以主动调用getDeviceToken接口更新绑定token后，重新调用绑定接口。
- 绑定token有一定的有效时限，失败的时候可以主动重试。

2.1.2. Android App Native开发配网

平台已提供了一套完整的配网页面，如果您希望自己使用Native开发自己的定制化配网页面，可以阅读本文，使用配网SDK进行开发。

背景信息

配网SDK提供了将Wi-Fi设备连接上家庭路由器和发现局域网内已联网设备的能力，包括一键广播配网（P2P）、设备热点配网、蓝牙辅助配网和设备间相互配网（以下简称零配）等。各配网方式介绍参见Wi-Fi设备配网方案介绍和配网SDK。

设备热点配网

> **说明** 需确保App已打开位置权限、GPS服务。

1. 调用配网SDK的setDevice接口设置待配网设备信息。

   待配网设备信息分为指定产品型号productKey和不指定型号productKey两种。

   指定产品型号productKey的待配信息有三种获取方式：

   - 通过直接调用云端接口获取产品列表（非配网SDK提供接口），选择待配网设备对应的产品获取ProductKey。
   - 通过扫描二维码获得待配设备信息，包含设备ProductKey。
   - 调用SDKLocalDeviceMgr、startDiscovery接口，并根据回调获取SOFT_AP_DEVICE类型的待配设备。
2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeSoftap，并调用配网SDK的startAddDevice接口开始配网。
3. 配网准备阶段，传递Wi-Fi的SSID、password。

App收到 `onProvisionPrepare prepareType=1` 回调后，调用SDK的toggleProvision方法传入当前连接路由器的SSID、password。

4. 配网中，根据回调的参数，提示终端用户连接设备热点或恢复Wi-Fi连接。

   App收到onProvisionStatus回调后，根据以下回调的参数来提示终端用户。

   - 回调参数中 `status=ProvisionStatus.SAP_NEED_USER_TO_CONNECT_DEVICE_AP` ：引导终端用户连接设备热点，如引导用户连接 `adh_{pk}_{mac}` 格式的设备热点。

     Android 10及以上版本或发现不到设备热点或者连接不上设备热点时会发生该回调。

   - 回调参数中 `status=ProvisionStatus.SAP_NEED_USER_TO_RECOVER_WIFI` ：引导终端用户恢复Wi-Fi连接。

     一般当设备Wi-Fi未自动连接或者连接到无效Wi-Fi时，会发生该回调。

5. 监听配网结果。

   可以在配网完成后调用LocalDeviceMgr getDeviceToken接口获取绑定token，并调用 基于token方式设备绑定完成设备的绑定。

完整示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
DeviceInfo deviceInfo = new DeviceInfo();
//方式一：指定productKey方式
deviceInfo.productKey = "xx";
deviceInfo.id = "xxx";// 通过startDiscovery发现的设备会返回该信息，在配网之前设置该信息，其它方式不需要设置
deviceInfo.linkType = "ForceAliLinkTypeSoftAP";
//方式二：不指定型号
deviceInfo.productKey = null;
deviceInfo.protocolVersion = "2.0";
deviceInfo.linkType = "ForceAliLinkTypeSoftAP";
//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(deviceInfo);
/**
* 第二步：开始配网
* 前置步骤，设置待配信息并开始配网
*/
AddDeviceBiz.getInstance().startAddDevice(context,  new IAddDeviceListener(){
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
        // 参数检测回调
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
        /**
         * 第三步：配网准备阶段，传入Wi-Fi信息
         * TODO 修改使用手机当前连接的Wi-Fi的SSID和password
        */
        if (prepareType == 1) {
            AddDeviceBiz.getInstance().toggleProvision("Your Wi-Fi ssid", "Your Wi-Fi passw
ord", 60);
        }
```

```
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
        /**
         * 第四步：配网中，配网UI引导
         * TODO 根据配网回调做 UI 引导
        */
        if (provisionStatus == ProvisionStatus.SAP_NEED_USER_TO_CONNECT_DEVICE_AP) {
            // 比如android 10，或者非android 10发现或连接设备热点失败。
            // 需要引导用户连接设备热点，否则会配网失败
            return;
        }
        if (provisionStatus == ProvisionStatus.SAP_NEED_USER_TO_RECOVER_WIFI) {
            // 引导用户恢复手机Wi-Fi连接，否则会配网失败
            return;
        }
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCode e
rrorCode) {
        /**
         * 第五步：监听配网结果
        */
// 如果配网结果包含token，请使用配网成功带的token做绑定。
    }
});
```

蓝牙辅助配网

需要新增以下蓝牙相关SDK依赖。

```
compile ('com.aliyun.alink.linksdk:breeze-biz:1.1.4')
```

> **说明** 需确保App已打开位置权限、GPS服务、蓝牙。

1. 调用SDK的setDevice接口设置待配网设备信息。

   蓝牙辅助配网支持指定产品型号productKey进行配网，也支持不指定型号直接开始配网。

   指定产品型号productKey的待配信息有以下三种获取方式：

   - 通过直接调用云端接口获取产品列表（非配网SDK提供接口），选择待配网设备对应的产品获取ProductKey。
   - 通过扫描二维码获得待配设备信息，包含设备ProductKey。
   - 调用SDKLocalDeviceMgr、startDiscovery接口，并根据回调获取BLE_ENROLLEE_DEVICE类型的待配设备。

2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeBLE，并调用配网SDK的startAddDevice接口开始配网。

3. 配网准备阶段，传递Wi-Fi的SSID和password。

   App收到 `onProvisionPrepare prepareType=1` 回调后，调用SDK的toggleProvision方法传入当前连接路由器的SSID和password。

4. 监听配网结果。

   可以在配网完成后调用LocalDeviceMgr getDeviceToken接口获取绑定token，并调用 基于token方式设备绑定完成设备的绑定。

完整代码示例如下。

```
/**
* 第一步：设置待配网设备信息
*/
DeviceInfo deviceInfo = new DeviceInfo();
//方式一：指定productKey和productId方式
deviceInfo.productKey = "xx"; //必填
deviceInfo.productId = "xxx"; //必填，可通过发现接口返回或者根据productKey或云端换取
deviceInfo.id = "xxx";// 通过startDiscovery发现的设备会返回该信息，在配网之前设置该信息，其它方式不需要设置
deviceInfo.linkType = "ForceAliLinkTypeBLE";
//方式二：不指定型号
deviceInfo.productKey = null;
deviceInfo.productId = null;
deviceInfo.protocolVersion = "2.0";
deviceInfo.linkType = "ForceAliLinkTypeBLE";
//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(deviceInfo);
/**
* 第二步：开始配网
* 前置步骤，设置待配信息并开始配网
*/
AddDeviceBiz.getInstance().startAddDevice(context,  new IAddDeviceListener(){
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
        // 参数检测回调
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
        /**
         * 第三步：配网准备阶段，传入Wi-Fi信息
         * TODO 修改使用手机当前连接的Wi-Fi的SSID和password
        */
        if (prepareType == 1) {
            AddDeviceBiz.getInstance().toggleProvision("Your Wi-Fi ssid", "Your Wi-Fi passw
ord", 60);
        }
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCode e
rrorCode) {
        /**
         * 第四步：监听配网结果
        */
        // 如果配网结果包含token，请使用配网成功带的token做绑定。
    }
});
```

一键配网

1. 调用SDK的setDevice接口设置待配网设备信息。

   一键配网需要指定产品型号productKey进行配网。 获取指定产品型号productKey的待配信息有以下两种方式。

   - 通过直接调用云端接口获取产品列表（非配网SDK提供接口），选择待配网设备对应的产品获取ProductKey。
   - 通过扫描二维码获得待配设备信息，包含设备ProductKey。

2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeBroadcast，并调用SDK的startAddDevice接口开始配网。

3. 配网准备阶段，传递Wi-Fi的SSID和password。

   App收到 `onProvisionPrepare prepareType=1` 回调后，调用配网SDK的toggleProvision方法，传入当前连接路由器的SSID和password。

4. 监听配网结果。

   可以在配网完成后调用LocalDeviceMgr getDeviceToken接口获取绑定token，并调用基于token方式设备绑定完成设备的绑定。

完整示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = "xx"; //必填
deviceInfo.linkType = "ForceAliLinkTypeBroadcast";
//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(deviceInfo);
/**
* 第二步：开始配网
* 前置步骤，设置待配信息并开始配网
*/
AddDeviceBiz.getInstance().startAddDevice(context,  new IAddDeviceListener(){
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
        // 参数检测回调
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
        /**
         * 第三步：配网准备阶段，传入Wi-Fi信息
         * TODO 修改使用手机当前连接的Wi-Fi的SSID和password
        */
        if (prepareType == 1) {
            AddDeviceBiz.getInstance().toggleProvision("Your Wi-Fi ssid", "Your Wi-Fi passw
ord", 60);
        }
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCode e
rrorCode) {
        /**
         * 第四步：监听配网结果
        */
        // 如果配网结果包含token，请使用配网成功带的token做绑定。
    }
});
```

零配配网

零配配网需要依赖当前手机Wi-Fi下有一个已配网绑定的设备和一个已上电并进入配网状态的设备，已配网设备发现待配设备后会上报到云端，App端可以获取待配的设备列表。

1. 调用SDK的setDevice接口设置待配网设备信息。

   零配需要通过调用配网SDK的发现接口，来获取待配的设备信息。

调用SDK的LocalDeviceMgr、startDiscovery接口，并根据回调获取CLOUD_ENROLLEE_DEVICE类型的待配设备。

2. 开始配网。

   指定配网方式linkType为ForceAliLinkTypeZeroAP，并调用配网SDK的startAddDevice接口开始配网。

3. 监听配网结果。

   可以在配网完成后调用LocalDeviceMgr getDeviceToken接口获取绑定token，并调用基于token方式设备绑定接口完成设备的绑定。

完整示例代码如下。

```
/**
* 第一步：设置待配网设备信息
*/
DeviceInfo toProvisionDeviceInfo = null;
EnumSet<DiscoveryType> enumSet = EnumSet.of(DiscoveryType.CLOUD_ENROLLEE_DEVICE);
LocalDeviceMgr.getInstance().startDiscovery(this, enumSet, null, new IDeviceDiscoveryListen
er() {
    @Override
    public void onDeviceFound(DiscoveryType discoveryType, List<DeviceInfo> list) {

        if (list != null && !list.isEmpty() && discoveryType == DiscoveryType.CLOUD_ENROLLE
E_DEVICE) {
            // 建议将发现的设备在UI上做展示，让用户触发对某个设备进行配网
            // 发现的待配设备列表缓存在内存中，用户触发配网的时候将待配信息设置到SDK配网接口
        }
    }
}
// list是通过发现接口发现的零配待配设备
toProvisionDeviceInfo = list.get(0);
toProvisionDeviceInfo.linkType = "ForceAliLinkTypeZeroAP";
// 设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(toProvisionDeviceInfo);
/**
* 第二步：开始配网
* 前置步骤，设置待配信息并开始配网
*/
AddDeviceBiz.getInstance().startAddDevice(context,  new IAddDeviceListener(){
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
        // 参数检测回调
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCode e
rrorCode) {
        /**
         * 第三步：监听配网结果
        */
        // 如果配网结果包含token，请使用配网成功带的token做绑定。
    }
});
```

设备绑定

配网SDK提供了获取设备端token的接口，该token可用于账号和设备的绑定（绑定接口非配网SDK提供，请参见基于token方式设备绑定）。

1. 获取绑定token。

   可通过以下方式获取绑定使用的token。

   - 调用本地发现接口，返回的已配设备列表设备信息中包含token。
   - 主动调用SDK接口获取token。

   第一种方式可以参考零配或者路由器配网的本地发现接口调用示例，本文档主要介绍第二种方式。

2. 绑定token。

   使用设备的ProductKey、DeviceName、Token调用基于token方式设备绑定接口进行绑定。

   样例代码如下。

```
/**
* 第一步：获取绑定token
*/
LocalDeviceMgr.getInstance().getDeviceToken(productKey, deviceName, 60*1000, 10*1000,
new IOnDeviceTokenGetListener() {
    @Override
    public void onSuccess(String token) {
        // 获取绑定需要的token
        /**
        * 第二步：调用绑定接口
        */
        //TODO 用户根据具体业务场景调用
    }
    @Override
    public void onFail(String reason) {
    }
});
```

> **说明**
>
> - 使用本地发现的信息绑定设备时，如果出现超时（如获取token之后过了很久才去绑定），可以主动调用getDeviceToken接口更新绑定token后，重新调用绑定接口。
> - 绑定token有一定的有效时限，失败的时候可以主动重试。

2.2. 蓝牙辅助配网最佳实践

2.2.1. 设备端开发

为提升您基于新开发的Combo设备（同时支持Wi-Fi和BLE）硬件平台移植生活物联网平台SDK提供的蓝牙辅助Wi-Fi配网功能的效率，本文档将选择一款硬件开发板，进行实际的移植示例，将整个功能移植、应用开发、功能调试等过程串联起来供您参考。

Combo设备移植蓝牙辅助配网功能的主要流程如下。

1. 选择硬件设备

设备研发生产厂商、模组厂商、芯片厂商等，根据您自己的产品与场景需要，选择合适的硬件平台（需有Combo芯片或模组）。

2. 控制台创建产品

   设备硬件选择好后，需在生活物联网平台控制台，创建项目和产品，新增测试设备，并配置好App的各项参数。

3. 获取SDK

   下载的生活物联网平台SDK中包含了所需的配网模块。

4. 移植蓝牙辅助配网HAL

   移植设备端生活物联网平台SDK（包括其中的Wi-Fi配网模块和蓝牙Breeze模块）到您的硬件平台上，并进行编译和调试。

   > **说明** 蓝牙辅助配网开发相关的介绍，请参见蓝牙辅助配网开发。如果您选择生活物联网平台认证的硬件平台，则不需要重新移植。硬件平台的选择和介绍请参见选择认证模组芯片。

5. 生成设备固件

   基于完整功能的示例应用，编译能运行于您硬件平台的蓝牙辅助配网设备固件。

6. 验证蓝牙辅助配网功能

   使用生活物联网平台提供的云智能App，验证蓝牙辅助配网功能。

一、准备硬件设备

请您根据自身产品选择合适的硬件设备，具体资源请参见蓝牙辅助配网开发（本文档以同时支持Wi-Fi和BLE的Combo芯片，BK7231U为示例）。

二、在控制台开发产品

1. 登录生活物联网控制台。
2. 选择已有项目或创建一个新项目。请参见创建项目。
3. 创建产品，并定义产品功能。请参见创建产品并定义功能。

   > **说明** 创建产品时，连网方式需配置为WiFi。支持蓝牙辅助配网的Wi-Fi + BLE的Combo设备，实质上还是一个可以直连网络的Wi-Fi设备，BLE的功能是Wi-Fi模块辅助配网。

4. 添加测试设备。请参见添加设备。
5. 选择App版本并配置App功能参数，请参见人机交互概述。

   配置配网引导时，默认配网方式或备选配网方式需配置为蓝牙辅助配网，并配置相应的配网文案，请参见配置配网引导。

三、获取SDK

获取生活物联网平台SDK时，建议您使用最新版本的含AliOS Things的SDK开发设备端。SDK下载地址请参见获取SDK。

四、移植蓝牙辅助配网HAL

蓝牙辅助配网同时使用了Wi-Fi和BLE的通信能力，因此该功能模块的移植，包括Wi-Fi配网模块和蓝牙Breeze模块的移植。

1. 移植BLE协议栈。

   蓝牙辅助配网中的BLE通信部分，使用了生活物联网平台的蓝牙Breeze（通过上层通信Profile的规则定义和实现）协议，基于蓝牙协议栈HAL的移植（需移植的HAL接口请参见蓝牙辅助配网开发）后，可以运行在不同厂商的蓝牙协议栈上。

   以移植BK7231U型号的芯片为示例，在含AliOS Things的SDK代码包中，蓝牙Breeze模块及其需要移植的HAL位于*/Living_SDK/framework/bluetooth/breeze/*目录下，该目录中的内容说明如下。

内容	说明
breeze.mk	蓝牙Breeze模块的makefile
core/	蓝牙Breeze模块的协议实现，移植时不用了解其实现细节
include/	蓝牙Breeze模块的协议实现的头文件，移植时不用了解其实现细节
hal/	蓝牙Breeze模块的HAL实现，移植时需要重点实现，此处默认的实现是使用AliOS Things提供的开源BLE协议栈的移植实现

内容	说明
api/	供上层应用开发调用的用户编程接口，移植时可以不用了解其细节

该部分的移植，主要是实现breeze.mk（与编译控制相关）和HAL（与芯片蓝牙协议栈相关）的部分。

i. 实现breeze.mk。

> **说明** 示例BK7231U基于GCC交叉编译工具链，采用 `.mk` 的makefile的编译方式。如果您使用其他类型的编译工具，类似 `.mk` 的实现需完整移植到您所使用的编译工具环境下。

```
NAME := breeze
$(NAME)_MBINS_TYPE := kernel
$(NAME)_VERSION := 1.0.0
$(NAME)_SUMMARY := breeze provides secure BLE connection to Alibaba IoT cloud and s
ervices.
$(NAME)_SOURCES += core/core.c
$(NAME)_SOURCES += core/transport.c
$(NAME)_SOURCES += core/ble_service.c
$(NAME)_SOURCES += core/sha256.c
$(NAME)_SOURCES += core/utils.c
GLOBAL_INCLUDES += api include hal/include
$(NAME)_COMPONENTS := chip_code
# Breeze安全广播功能，用于增强广播数据的安全性，蓝牙辅助配网中未使用
secure_adv ?= 0
ifeq ($(secure_adv), 1)
GLOBAL_DEFINES += CONFIG_AIS_SECURE_ADV
endif
# 是否已移植并使用AliOS Things提供的开源BLE协议栈
# 如果厂商驱动中已包含自己的BLE协议栈，此项功能不用选择
# BK7231U使用的是厂商自己的BLE协议栈，因此这里不会使能
btstack ?= zephyr
ifeq (zephyr, $(btstack))
$(NAME)_COMPONENTS += framework.bluetooth.breeze.hal.ble
endif
$(NAME)_SOURCES += api/breeze_export.c
# Breeze安全认证功能，蓝牙辅助配网中必须打开
bz_en_auth ?= 1
ifeq ($(bz_en_auth), 1)
GLOBAL_DEFINES += EN_AUTH
$(NAME)_SOURCES += core/auth.c
endif
# Breeze辅助配网功能，蓝牙辅助配网中必须打开
bz_en_awss ?= 1
ifeq ($(bz_en_awss), 1)
ifeq ($(bz_en_auth), 0)
$(error awss need authentication, please set "bz_en_auth = 1")
endif
GLOBAL_DEFINES += EN_COMBO_NET
GLOBAL_DEFINES += AWSS_REGION_ENABLE
$(NAME)_SOURCES += core/extcmd.c
$(NAME)_SOURCES += api/breeze_awss_export.c
endif
```

ii. 实现*hal*目录下的文件。

*hal*目录下面需要移植的文件分别是：*breeze_hal_ble.h*、*breeze_hal_os.h*、*breeze_hal_sec.h*。三个文件的实现请参见SDK代码包中*/Living_SDK/platform/mcu/bk7231u/hal/breeze_hal/*下的内容。

- breeze_hal_ble.h

  蓝牙协议栈移植接口，涉及BLE的广播、连接、GATT相关的内容，需实现*breeze_hal_ble.c*。

```
/**
 * API to initialize ble stack.
 * @parma[in]  ais_init  Bluetooth stack init parmaters.
```

```
 * @return     0 on success, error code if failure.
 */
// 对您使用的硬件平台的BLE协议栈初始化，协议栈初始化成功后，BLE功能才能正常使用
// 蓝牙辅助配网阶段，先初始化BLE协议栈，接着基于协议栈注册GATT Service，完成后才能通信
// Breeze的BLE通信通道是基于GATT Profile设计的，在设备端实现了Breeze的GATT Service
// 生活物联网平台称之为AIS（Alibaba IoT Service）
// 该GATT Service中是一组分别支持Read、Write、Notify、Indicate操作的Characteristic，
// BLE协议栈初始化函数传入的是该GATT Primary Service的描述，BLE协议栈初始化函数需要将该S
ervice
// （也可以称为Attribute Table）注册到BLE stack，并将BLE链路的connect、disconnect与传
入Service描述中的元素对应
// 通过该方式实现了BLE stack和AIS Service之间的联系，
// 当BLE连接建立或断开，会通过初始化注册的connect、disconnect回调通知到Breeze模块中，
// 同时在连接状态下对端对于AIS Service的操作，也会通过Read，Write等回调通知到Breeze模块中
。
// 关于AIS Service的定义如下，供您移植和代码调试时参考
// Attribute Type                |UUID   |Properties        |Permission
// AIS Primary Service           |0xFEB3
// Read Characteristic           |0xFED4 |Read              |Read
// Write Characteristic          |0xFED5 |Read/Write        |Read/Write
// Indicate Characteristic       |0xFED6 |Read/Indicate     |Indication
// WriteWithNoRsp Characteristic |0xFED7 |Read/WriteCommand |Read/Write
// Notify Characteristic         |0xFED8 |Read/Notify       |Notification
ais_err_t ble_stack_init(ais_bt_init_t *ais_init);
/**
 * API to de-initialize ble stack.
 * @return     0 on success, error code if failure.
 */
// 对您使用的硬件平台的BLE协议栈反初始化，如果在蓝牙辅助配网结束，设备将不会再使用BLE通信能
力，
// 对于此种BLE使用频率低的场景，可以在蓝牙通信功能完成后，
// 将BLE协议栈反初始化，避免BLE协议栈持续开启但不使用，导致的不必要功耗和Wi-Fi、BLE共存问
题
ais_err_t ble_stack_deinit();
/**
 * API to send data via AIS's Notify Characteristics.
 * @parma[in]  p_data  data buffer.
 * @parma[in]  length  data length.
 * @return     0 on success, error code if failure.
 */
// 通过前面注册的AIS Service的Notify Characteristic向连接链路的对端设备发送数据
ais_err_t ble_send_notification(uint8_t *p_data, uint16_t length);
/**
 * API to send data via AIS's Indicate Characteristics.
 * @parma[in]  p_data  data buffer.
 * @parma[in]  length  data length.
 * @parma[in]  txdone  txdone callback.
 * @return     0 on success, erro code if failure.
 */
// 通过注册的AIS Service的Indicate Characteristic向连接链路的对端设备发送数据
ais_err_t ble_send_indication(uint8_t *p_data, uint16_t length, void (*txdone)(ui
nt8_t res));
/**
 * API to disconnect BLE connection.
```

```
 * @param[in]  reason  the reason to disconnect the connection.
 */
// 从设备端主动断开已经建立的BLE连接
void ble_disconnect(uint8_t reason);
/**
 * API to start bluetooth advertising.
 * @return     0 on success, erro code if failure.
 */
// 开始广播特定的内容，该内容为Manufacturer Specific Data，是Breeze填充，
// 以便对端设备能够识别Breeze蓝牙设备所提供的服务，以及进行相关的身份校验
ais_err_t ble_advertising_start(ais_adv_init_t *adv);
/**
 * API to stop bluetooth advertising.
 * @return     0 on success, erro code if failure.
 */
// 停止广播Breeze填充的特定的广播包，使支持Breeze协议的对端设备不要再发现此设备
ais_err_t ble_advertising_stop();
/**
 * API to start bluetooth advertising.
 * @parma[out]  mac  the uint8_t[BD_ADDR_LEN] space the save the mac address.
 * @return     0 on success, erro code if failure.
 */
// 获取设备的Bluetooth MAC地址，会用于身份识别等用途
ais_err_t ble_get_mac(uint8_t *mac);
```

- breeze_hal_os.h

  OS系统移植接口，需实现*breeze_hal_os.c*。

```
/**
 * This function will create a timer.
 *
 * @param[in]  timer   pointer to the timer.
 * @param[in]  fn      callbak of the timer.
 * @param[in]  arg     the argument of the callback.
 * @param[in]  ms      ms of the normal timer triger.
 * @param[in]  repeat  repeat or not when the timer is created.
 * @param[in]  auto_run  run auto or not when the timer is created.
 *
 * @return  0: success.
 */
// 创建一个系统的software timer，参数可配置该timer的定时时长，定时触发的回调，
// 是否反复定时，是否创建时立即运行timer等
// 下面几个接口是software timer操作相关，Breeze模块中会用来做定时触发和超时计算的用途，
// 运行过程中可能会创建多个software timer
int os_timer_new(os_timer_t *timer, os_timer_cb_t cb, void *arg, int ms);
/**
 * This function will start a timer.
 *
 * @param[in]  timer  pointer to the timer.
 *
 * @return  0: success.
 */
// 运行前面创建好的software timer
int os_timer_start(os_timer_t *timer);
```

```
/**
 * This function will stop a timer.
 *
 * @param[in]  timer  pointer to the timer.
 *
 * @return  0: success.
 */
// 停止正在运行中的software timer，与前面的os_timer_start相对应
int os_timer_stop(os_timer_t *timer);
/**
 * This function will delete a timer.
 *
 * @param[in]  timer  pointer to a timer.
 */
// 删除前面创建的一个系统的software timer，与os_timer_new相对应
void os_timer_free(os_timer_t *timer);
/**
 * Reboot system.
 */
// 设备系统重启，一般在OTA一类的服务中，需要重启系统以便执行相关的固件搬移和系统初始化，
// 类似这样的Breeze模块中的服务会需要用到该接口
void os_reboot();
/**
 * Msleep.
 *
 * @param[in]  ms  sleep time in milliseconds.
 */
// 系统睡眠和延时，有的操作需要等待某个动作发生才能执行下一步，就会用到该接口。
// 该接口在多线程实现中，一般会让所在线程休眠指定的时间，而不影响其他线程的执行
void os_msleep(int ms);
/**
 * Get current time in mini seconds.
 *
 * @return  elapsed time in mini seconds from system starting.
 */
// 获取系统的当前时间，该时间是一个相对系统启动点的相对时间，单位为ms
long long os_now_ms();
/**
 * Add a new KV pair.
 *
 * @param[in]  key    the key of the KV pair.
 * @param[in]  value  the value of the KV pair.
 * @param[in]  len    the length of the value.
 * @param[in]  sync   save the KV pair to flash right now (should always be 1).
 *
 * @return  0 on success, negative error on failure.
 */
// 下面几个接口是Key-Value存储相关的接口，Breeze模块会用于一些数据的固化存储，目前蓝牙辅助
配网
// 中暂时未固化存储数据，但考虑后续的功能扩展，Key-Value存储相关的接口也需要进行移植
// os_kv_set为将指定的数据存入Key-Value存储中
int os_kv_set(const char *key, const void *value, int len, int sync);
/**
 * Get the KV pair's value stored in buffer by its key.
```

```
 *
 * @note: the buffer_len should be larger than the real length of the value,
 *        otherwise buffer would be NULL.
 *
 * @param[in]      key         the key of the KV pair to get.
 * @param[out]     buffer      the memory to store the value.
 * @param[in-out]  buffer_len  in: the length of the input buffer.
 *                             out: the real length of the value.
 *
 * @return  0 on success, negative error on failure.
 */
// 从Key-Value存储中获取相应的数据
int os_kv_get(const char *key, void *buffer, int *buffer_len);
/**
 * Delete the KV pair by its key.
 *
 * @param[in]  key  the key of the KV pair to delete.
 *
 * @return  0 on success, negative error on failure.
 */
// 将Key-Value存储中某个Key对应的数据删除
int os_kv_del(const char *key);
/**
 * Generate random number.
 *
 * @return  random value implemented by platform.
 */
// 返回一个随机值
int os_rand(void);
```

- breeze_hal_sec.h

  安全算法移植接口，需实现*breeze_hal_sec.c*。

```
/**
 * @brief  Initialize the aes context, which includes key/iv info.
 *         The aes context is implementation specific.
 *
 * @param[in] key:
 * @param[in] iv:
 * @param[in] dir: AIS_AES_ENCRYPTION or AIS_AES_DECRYPTION
 * @return p_ais_aes128_t
   @verbatim None
   @endverbatim
 * @see None.
 * @note None.
 */
// 此部分为AES128算法的实现，Breeze模块通信是使用AES128 CBC加密的，
// 因此务必保证该实现正常，否则会导致对端和设备间的通信异常
void *ais_aes128_init(const uint8_t *key, const uint8_t *iv);
/**
 * @brief   Destroy the aes context.
 *
 * @param[in] aes: the aex context.
 * @return
```

```
    @verbatim
      = 0: succeeded
      = -1: failed
    @endverbatim
 * @see None.
 * @note None.
 */
int ais_aes128_destroy(void *aes);
/**
 * @brief   Do aes-128 cbc encryption.
 *          No padding is required inside the implementation.
 *
 * @param[in] aes: AES handler
 * @param[in] src: plain data
 * @param[in] block_num: plain data number of 16 bytes size
 * @param[out] dst: cipher data
 * @return
    @verbatim
      = 0: succeeded
      = -1: failed
    @endverbatim
 * @see None.
 * @note None.
 */
int ais_aes128_cbc_encrypt(void *aes, const void *src, size_t block_num,
                           void *dst);
/**
 * @brief   Do aes-128 cbc decryption.
 *          No padding is required inside the implementation.
 *
 * @param[in] aes: AES handler
 * @param[in] src: cipher data
 * @param[in] block_num: plain data number of 16 bytes size
 * @param[out] dst: plain data
 * @return
    @verbatim
      = 0: succeeded
      = -1: failed
    @endverbatim
 * @see None.
 * @note None.
 */
int ais_aes128_cbc_decrypt(void *aes, const void *src, size_t block_num,
                           void *dst);
```

2. 移植Wi-Fi协议栈。

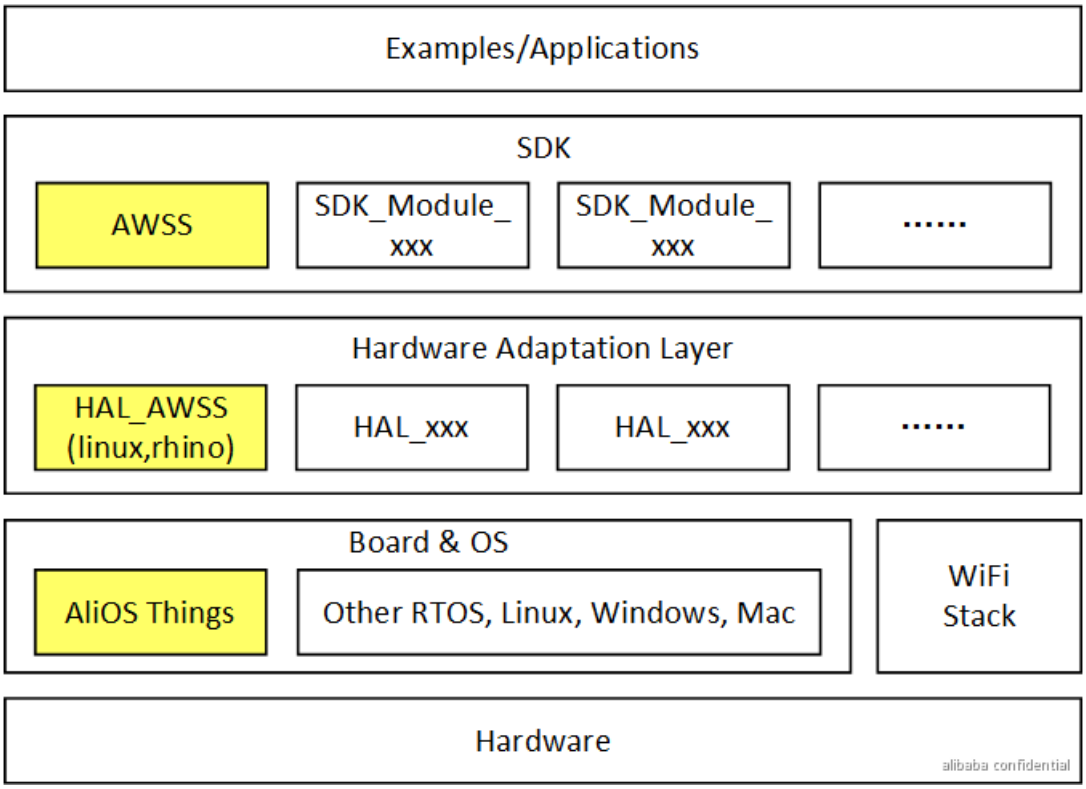
   蓝牙辅助配网中的BLE通信部分，使用了生活物联网平台的蓝牙Breeze协议，蓝牙Breeze属于上层通信Profile的规则定义和实现，基于蓝牙协议栈HAL的正确移植，可以运行于不同厂商的蓝牙协议栈之上。需移植的HAL接口（参见蓝牙辅助配网开发）。

   以BK7231U的移植实现为示例，在含AliOS Things的生活物联网平台SDK的代码包中，Wi-Fi及其需要移植的HAL位于以下目录下。

   - /Living_SDK/framework/protocol/linkkit/sdk/iotx-sdk-c_clone/include/imports/

- /Living_SDK/framework/protocol/linkkit/sdk/iotx-sdk-c_clone/include/iot_import.h

依赖的通用HAL的接口移植（如OS，LwIP，Security等）请参见Wi-Fi设备配网适配开发，基于AliOS Things实现这些HAL的方法请参见Wi-Fi芯片移植。对Wi-Fi HAL接口在BK7231U（基于AliOS Things）的移植实现示例说明如下。



HAL_AWSS中的移植接口，有些是针对某种配网方式才需要实现的接口，对HAL_AWSS的所有HAL接口分类如下。

分类	说明
配网通用接口	所有配网方式都必须要实现的接口，包括蓝牙辅助配网、设备热点配网、一键配网、零配配网等
设备热点配网专用	需要支持设备热点配网方式时需实现的接口，如HAL_Awss_Open_Ap、HAL_Awss_Close_Ap等打开和关闭设备热点的接口
蓝牙辅助配网设备热点配网零配配网手机热点配网	需要支持这几种配网方式中的一种或几种时，需实现的接口HAL_Awss_Get_Conn_Encrypt_Type
一键配网专用	需要支持一键配网方式时需实现的接口，如HAL_Awss_Get_Encrypt_Type

i. 编译相关控制。

编译配置文件请参见*/Living_SDK/framework/protocol/linkkit/sdk/iotx-sdk-c_clone/make.settings*。

```
#
# Automatically generated file; DO NOT EDIT.
# Main Menu
#
#
# Configure Link Kit SDK for IoT Embedded Devices
#
FEATURE_SRCPATH="."
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_ALCS_ENABLED=y
#
# MQTT Configurations
#
# FEATURE_MQTT_SHADOW is not set
# FEATURE_MQTT_LOGPOST is not set
FEATURE_MQTT_PREAUTH_SUPPORT_HTTPS_CDN=y
FEATURE_DEVICE_MODEL_ENABLED=y
FEATURE_MQTT_AUTO_SUBSCRIBE=y
#
# Device Model Configurations
#
# FEATURE_DEVICE_MODEL_GATEWAY is not set
# 设备绑定的Feature需要打开，在设备配网完成后会和用户账户之间进行绑定
FEATURE_DEV_BIND_ENABLED=y
# FEATURE_DEVICE_MODEL_RAWDATA_SOLO is not set
# FEATURE_COAP_COMM_ENABLED is not set
FEATURE_OTA_ENABLED=y
# FEATURE_HTTP2_COMM_ENABLED is not set
# FEATURE_HTTP_COMM_ENABLED is not set
FEATURE_SUPPORT_TLS=y
# FEATURE_SAL_ENABLED is not set
# 设备Wi-Fi配网的Feature必须打开
FEATURE_WIFI_PROVISION_ENABLED=y
#
# AWSS Configurations
#
# 设备Wi-Fi配网可以支持的配网方式：一键配网、零配配网、设备热点配网（蓝牙辅助配网目前无需在此设置）
FEATURE_AWSS_SUPPORT_SMARTCONFIG=y
FEATURE_AWSS_SUPPORT_ZEROCONFIG=y
FEATURE_AWSS_SUPPORT_DEV_AP=y
```

ii. 实现配网通用接口。

```
/**
 * @brief   获取Wi-Fi设备的MAC地址，格式应当是"XX:XX:XX:XX:XX:XX"
 *
 * @param   mac_str ：用于存放MAC地址字符串的缓冲区数组
 * @return  指向缓冲区数组起始位置的字符指针
 */
```

```
char *HAL_Wifi_Get_Mac(_OU_ char mac_str[HAL_MAC_LEN])
{
    uint8_t mac[6] = { 0 };
    // 调用驱动层的接口获取设备的MAC地址，并转为字符串的格式
    hal_wifi_get_mac_addr(NULL, mac);
    snprintf(mac_str, HAL_MAC_LEN, "%02x:%02x:%02x:%02x:%02x:%02x", mac[0],
             mac[1], mac[2], mac[3], mac[4], mac[5]);
    return mac_str;
}
extern void wifi_get_ip(char ips[16]);
/**
 * @brief   获取Wi-Fi网口的IP地址，点分十进制格式保存在字符串数组出参，
 * 二进制格式则作为返回值，并以网络字节序(大端)表达
 *
 * @param   ifname : 指定Wi-Fi网络接口的名字
 * @param   ip_str : 存放点分十进制格式的IP地址字符串的数组
 * @return  二进制形式的IP地址，以网络字节序(大端)组织
 */
uint32_t HAL_Wifi_Get_IP(_OU_ char ip_str[NETWORK_ADDR_LEN],
                         _IN_ const char *ifname)
{
    //(void *)ifname;
    // 调用驱动层的接口，获取设备的IP地址。该IP地址分为两种情况:
    // 1.一般情况，设备作为Station连接到AP，被分配的IP地址
    // 2.设备支持设备热点，开启SoftAP模式时作为gateway的IP地址
    wifi_get_ip(ip_str);
    return 0;
}
/**
 * @brief   获取在每个信道(`channel`)上扫描的时间长度，单位是毫秒
 *          该接口主要是一键配网和零配配网会使用到，因为这两者在配网时
 *          需要在Wi-Fi信道列表上进行轮询扫描
 *
 * @return  时间长度，单位是毫秒
 * @note    推荐时长是200毫秒到400毫秒
 */
int HAL_Awss_Get_Channelscan_Interval_Ms(void)
{
    // 一般都设置为该默认的值
    return 250;
}
/**
 * @brief   获取配网服务(`AWSS`)的超时时间长度，单位是毫秒
 *
 * @return  超时时长，单位是毫秒
 * @note    推荐时长是3分钟
 */
int HAL_Awss_Get_Timeout_Interval_Ms(void)
{
    // 一般都设置为该默认的值
    return 3 * 60 * 1000;
}
/**
 * @brief   802.11帧的处理函数，可以将802.11 Frame传递给这个函数
```

```
 *
 * @param[in] buf @n 80211 frame buffer, or pointer to struct ht40_ctrl
 * @param[in] len @n 80211 frame buffer length
 * @param[in] link_type @n AWSS_LINK_TYPE_NONE for most rtos HAL,
 *                         and for linux HAL, do the following step to check
 *                         which header type the driver supported.
 * @verbatim
 *             a) iwconfig wlan0 mode monitor    #open monitor mode
 *             b) iwconfig wlan0 channel 6    #switch channel 6
 *             c) tcpdump -i wlan0 -s0 -w file.pacp    #capture 80211 frame
 * & save d) open file.pacp with wireshark or omnipeek check the link header
 * type and fcs included or not
 * @endverbatim
 * @param[in] with_fcs @n 80211 frame buffer include fcs(4 byte) or not
 * @param[in] rssi @n rssi of packet
 */
awss_recv_80211_frame_cb_t g_ieee80211_handler;
static void monitor_data_handler(uint8_t *buf, int len,
                                 hal_wifi_link_info_t *info)
{
    int with_fcs  = 0;
    int link_type = AWSS_LINK_TYPE_NONE;
    (*g_ieee80211_handler)((char *)buf, len, link_type, with_fcs,
                           info->rssi);
}
/**
 * @brief   设置Wi-Fi网卡工作在监听（Monitor或Sniffer）模式，
 *          并在收到802.11帧的时候调用被传入的回调函数，回调函数的格式如上
 *          必须要将802.11帧Buffer、长度、HAL类别，是否带有FCS、RSSI等信息提供给上层
 *
 * @param[in] cb @n A function pointer, called back when wifi receive a
 * frame.
 */
void HAL_Awss_Open_Monitor(_IN_ awss_recv_80211_frame_cb_t cb)
{
    // 这里BK7231U是移植了AliOS Things的驱动移植层的，因此是hal_wifi_module
    // 风格的实现。这里主要是将回调先注册到驱动层，并开启Monitor模式，
    // 在监听到802.11的帧时，都通过回调函数上报上去
    hal_wifi_module_t *module = hal_wifi_get_default_module();
    if (module == NULL) {
        return;
    }
    g_ieee80211_handler = cb;
    hal_wifi_register_monitor_cb(module, monitor_data_handler);
    hal_wifi_start_wifi_monitor(module);
    HAL_Awss_Switch_Channel(6, 0, NULL);
}
/**
 * @brief   设置Wi-Fi网卡离开监听（Monitor或Sniffer）模式，
 * 并开始以站点(Station)模式工作
 */
void HAL_Awss_Close_Monitor(void)
{
    // 将原来注册的回调函数取消（设置为NULL），并关闭设备的Monitor模式，
    hal_wifi_module_t *module;
```

```
    hal_wifi_module_t *module;
    module = hal_wifi_get_default_module();
    if (module == NULL) {
        return;
    }
    hal_wifi_register_monitor_cb(module, NULL);
    hal_wifi_stop_wifi_monitor(module);
}
/**
 * @brief handle one piece of AP information from Wi-Fi scan result
 *
 * @param[in] ssid @n name of AP
 * @param[in] bssid @n mac address of AP
 * @param[in] channel @n AP channel
 * @param[in] rssi @n rssi range[-127, -1].
 *          the higher the RSSI number, the stronger the signal.
 * @param[in] is_last_ap @n this AP information is the last one if
 * is_last_ap > 0. this AP information is not the last one if is_last_ap ==
 * 0.
 * @return 0 for Wi-Fi scan is done, otherwise return -1
 * @see None.
 * @note None.
 */
typedef int (*awss_wifi_scan_result_cb_t)(const char ssid[HAL_MAX_SSID_LEN],
                                          const uint8_t bssid[ETH_ALEN],
                                          enum AWSS_AUTH_TYPE auth,
                                          enum AWSS_ENC_TYPE  encry,
                                          uint8_t channel, signed char rssi,
                                          int is_last_ap);
/**
 * @brief   启动一次Wi-Fi的空中扫描(Scan)
 *          该模式需要与前面的Monitor（或Sniffer）模式区别，
 *          Monitor(Sniffer):持续开启802.11帧监听，并实时上报监听到的帧，直到该模式被关闭
 *          Scan:扫描AP(通过Beacon和Probe Request帧)，并记录一次扫描到的多个AP的结果
 *
 * @param[in] cb @n pass ssid info(scan result) to this callback one by one
 * @return 0 for Wi-Fi scan is done, otherwise return -1
 * @see None.
 * @note
 *      This API should NOT exit before the invoking for cb is finished.
 *      This rule is something like the following :
 *      HAL_Wifi_Scan() is invoked...
 *      ...
 *      for (ap = first_ap; ap <= last_ap; ap = next_ap){
 *        cb(ap)
 *      }
 *      ...
 *      HAL_Wifi_Scan() exit...
 */
int HAL_Wifi_Scan(awss_wifi_scan_result_cb_t cb)
{
    // 注册Scan到的AP列表的回调函数
    // 并启动Scan(对周边AP的扫描，建议实现为active scan，扫到的效率更高)
    // 该函数是同步执行的方式，即调用后线程会被其阻塞，直到本次扫描结束
    netmgr_register_wifi_scan_result_callback(
```

```
      (netmgr_wifi_scan_result_cb_t)cb);
    hal_wifi_start_scan_adv(NULL);
    while (netmgr_get_scan_cb_finished() != true) { // block
        aos_msleep(50);
    }
    return 0;
}
/**
 * @brief    设置Wi-Fi网卡切换到指定的信道(channel)上
 *
 * @param[in] primary_channel @n Primary channel.
 * @param[in] secondary_channel @n Auxiliary channel if 40Mhz channel is
 * supported, currently this param is always 0.
 * @param[in] bssid @n A pointer to Wi-Fi BSSID on which awss lock the
 * channel, most HAL may ignore it.
 */
void HAL_Awss_Switch_Channel(_IN_ char     primary_channel,
                             _IN_OPT_ char secondary_channel,
                             _IN_OPT_ uint8_t bssid[ETH_ALEN])
{
    hal_wifi_module_t *module;
    module = hal_wifi_get_default_module();
    if (module == NULL) {
        return;
    }
    // 调用驱动层接口，设置设备此时工作在指定的信道，对于某些应用需要，指定信道
    // 会使一些操作更加高效
    hal_wifi_set_channel(module, (int)primary_channel);
}
/**
 * @brief    要求Wi-Fi网卡连接指定热点(Access Point)的函数
 *
 * @param[in] connection_timeout_ms @n AP connection timeout in ms or
 HAL_WAIT_INFINITE
 * @param[in] ssid @n AP ssid
 * @param[in] passwd @n AP passwd
 * @param[in] auth @n optional(AWSS_AUTH_TYPE_INVALID), AP auth info
 * @param[in] encry @n optional(AWSS_ENC_TYPE_INVALID), AP encry info
 * @param[in] bssid @n optional(NULL or zero mac address), AP bssid info
 * @param[in] channel @n optional, AP channel info
 * @return
   @verbatim
     = 0: connect AP & DHCP success
     = -1: connect AP or DHCP fail/timeout
   @endverbatim
 * @see None.
 * @note
 *      If the STA connects the old AP, HAL should disconnect from the old
 AP firstly.
 */
int HAL_Awss_Connect_Ap(_IN_ uint32_t connection_timeout_ms,
                        _IN_ char     ssid[HAL_MAX_SSID_LEN],
                        _IN_ char     passwd[HAL_MAX_PASSWD_LEN],
                        _IN_OPT_ enum AWSS_AUTH_TYPE auth,
```

```
                        _IN_OPT_ enum AWSS_ENC_TYPE  encry,
                        _IN_OPT_ uint8_t bssid[ETH_ALEN],
                        _IN_OPT_ uint8_t channel)
{
    int                ms_cnt = 0;
    netmgr_ap_config_t config = { 0 };
    if (ssid != NULL) {
        strncpy(config.ssid, ssid, sizeof(config.ssid) - 1);
    }
    if (passwd != NULL) {
        strncpy(config.pwd, passwd, sizeof(config.pwd) - 1);
    }
    if (bssid != NULL) {
        memcpy(config.bssid, bssid, ETH_ALEN);
    }
    // 将要连接的AP的信息暂存下来
    netmgr_set_ap_config(&config);
    // 在正式连接AP前，Suspend Station，防止设备受上一次操作未结束的干扰
    hal_wifi_suspend_station(NULL);
    // LOGI("aos_awss", "Will reconnect wifi: %s %s", ssid, passwd);
    // 实际会调用驱动层接口，向指定的AP发起连接
    netmgr_reconnect_wifi();
    // 在调用驱动层接口去连接AP后，在此阻塞线程一断时间，该段时间内持续检查设备当前是否
    // 已经连接AP成功且获取到了IP地址，如果获取到IP地址，该函数结束，并返回连接成功的结果
    while (ms_cnt < connection_timeout_ms) {
        if (netmgr_get_ip_state() == false) {
            LOGD("[waitConnAP]", "waiting for connecting AP");
            aos_msleep(500);
            ms_cnt += 500;
        } else {
            LOGI("[waitConnAP]", "AP connected");
            return 0;
        }
    }
    // if AP connect fail, should inform the module to suspend station
    // to avoid module always reconnect and block Upper Layer running
    // 如果在连接AP超时时间到达，都没能成功连上AP，或者连上了AP没能获取到IP地址，说明设备
    // 本次发起连接是失败的(可能是AP的ssid,密码等信息有误，或者是AP本身工作异常，或者是
    // 干扰太强或信号太弱导致没法顺利连接上)，那么此时Suspend Station(驱动层可能还在
    // 处于对AP发起连接的状态，Suspend Station将该状态终止，使设备不再向AP发起连接)，
    // 最后返回连接失败的结果给上层处理
    hal_wifi_suspend_station(NULL);
    return -1;
}
#define FRAME_ACTION_MASK (1 << FRAME_ACTION)
#define FRAME_BEACON_MASK (1 << FRAME_BEACON)
#define FRAME_PROBE_REQ_MASK (1 << FRAME_PROBE_REQ)
#define FRAME_PROBE_RESP_MASK (1 << FRAME_PROBE_RESPONSE)
#define FRAME_DATA_MASK (1 << FRAME_DATA)
/**
 * @brief    在当前信道(channel)上以基本数据速率(1Mbps)发送裸的802.11帧(raw
 *           802.11 frame)
 *
 * @param[in] type @n see enum HAL_Awss_frame_type, currently only
```

```
 *                      FRAME_BEACON, FRAME_PROBE_REQ is used
 * @param[in] buffer @n 80211 raw frame, include complete mac header & FCS
 field
 * @param[in] len @n 80211 raw frame length
 * @return
   @verbatim
   =  0, send success.
   = -1, send failure.
   = -2, unsupported.
   @endverbatim
 * @see None.
 * @note awss use this API send raw frame in Wi-Fi monitor mode & station
 mode
 */
int HAL_Wifi_Send_80211_Raw_Frame(_IN_ enum HAL_Awss_Frame_Type type,
                                  _IN_ uint8_t *buffer, _IN_ int len)
{
    // 调用驱动层的接口发送802.11的帧，类型必须要支持上面定义的5种
    return hal_wlan_send_80211_raw_frame(NULL, buffer, len);
}
/**
 * @brief    管理帧的处理回调函数
 *
 * @param[in] buffer @n 80211 raw frame or ie(information element) buffer
 * @param[in] len @n buffer length
 * @param[in] rssi_dbm @n rssi in dbm
 * @param[in] buffer_type @n 0 when buffer is a 80211 frame,
 *                           1 when buffer only contain IE info
 * @return None.
 * @see None.
 * @note None.
 */
typedef void (*awss_wifi_mgmt_frame_cb_t)(_IN_ uint8_t *buffer,
                                          _IN_ int len,
                                          _IN_ signed char rssi_dbm,
                                          _IN_ int buffer_type);
static awss_wifi_mgmt_frame_cb_t monitor_cb = NULL;
static void mgnt_rx_cb(uint8_t *data, int len, hal_wifi_link_info_t *info)
{
    if (monitor_cb) {
        monitor_cb(data, len, info->rssi, 0);
    }
}
/**
 * @brief    使能或禁用对管理帧的过滤
 *
 * @param[in] filter_mask @n see mask macro in enum HAL_Awss_frame_type,
 *                       currently only FRAME_PROBE_REQ_MASK &
 FRAME_BEACON_MASK is used
 * @param[in] vendor_oui @n oui can be used for precise frame match,
 optional
 * @param[in] callback @n see awss_wifi_mgmt_frame_cb_t, passing 80211
 *                       frame or ie to callback. when callback is NULL
 *                       disable sniffer feature, otherwise enable it.
```

```
 * @return
   @verbatim
   =  0, success
   = -1, fail
   = -2, unsupported.
   @endverbatim
 * @see None.
 * @note awss use this API to filter specific mgnt frame in Wi-Fi station
 mode
 */
int HAL_Wifi_Enable_Mgmt_Frame_Filter(
  _IN_ uint32_t filter_mask, _IN_OPT_ uint8_t vendor_oui[3],
  _IN_ awss_wifi_mgmt_frame_cb_t callback)
{
    monitor_cb = callback;
    // 管理帧的过滤开启与关闭，都在此接口中实现，开启时需传入有效的回调函数，而传入NULL时
    // 表示将管理帧的过滤功能关闭。
    // 管理帧过滤的开启时机，可能在设备处于Station模式，也可能在设备处于设备热点模式，
    // 因此只要设备Wi-Fi stack能获取到周边的管理帧，都需要能支持管理帧的过滤开启
    if (callback != NULL) {
        hal_wlan_register_mgnt_monitor_cb(NULL, mgnt_rx_cb);
    } else {
        hal_wlan_register_mgnt_monitor_cb(NULL, NULL);
    }
    return 0;
}
/**
 * @brief check system network is ready(get ip address) or not.
 *
 * @param None.
 * @return 0, net is not ready; 1, net is ready.
 * @see None.
 * @note None.
 */
int HAL_Sys_Net_Is_Ready()
{
    // 调用接口判断设备当前的IP地址是否有效
    return netmgr_get_ip_state() == true ? 1 : 0;
}
/**
 * @brief    获取所连接的热点(Access Point)的信息
 *
 * @param[out] ssid: array to store ap ssid. It will be null if ssid is not
 required.
 * @param[out] passwd: array to store ap password. It will be null if ap
 password is not required.
 * @param[out] bssid: array to store ap bssid. It will be null if bssid is
 not required.
 * @return
   @verbatim
     = 0: succeeded
     = -1: failed
   @endverbatim
 * @see None.
 * @note None.
```

```
 * @note None.
 */
int HAL_Wifi_Get_Ap_Info(_OU_ char ssid[HAL_MAX_SSID_LEN],
                         _OU_ char passwd[HAL_MAX_PASSWD_LEN],
                         _OU_ uint8_t bssid[ETH_ALEN])
{
    netmgr_ap_config_t config = { 0 };
    netmgr_get_ap_config(&config);
   if (ssid) {
        strncpy(ssid, config.ssid, HAL_MAX_SSID_LEN - 1);
    }
    if (passwd) {
#ifdef DISABLE_SECURE_STORAGE
        strncpy(passwd, config.pwd, HAL_MAX_PASSWD_LEN - 1);
#else
        extern int iotx_ss_decrypt(const char* in_data, int in_len, char* out_data,
int out_len);
        iotx_ss_decrypt(config.pwd, MAX_PWD_SIZE, passwd, MAX_PWD_SIZE);
#endif
    }
    if (bssid) {
        memcpy(bssid, config.bssid, ETH_ALEN);
    }
    return 0;
}
/**
 * @brief    获取当前Station模式与AP连接状态的信息
 *
 * @param[out] p_rssi: rssi value of current link
 * @param[out] p_channel: channel of current link
 *
 * @return
   @verbatim
     = 0: succeeded
     = -1: failed
   @endverbatim
 * @see None.
 * @note None.
 * @note awss use this API to get rssi and channel of current link
 */
int HAL_Wifi_Get_Link_Stat(_OU_ int *p_rssi,
                           _OU_ int *p_channel)
{
    int ret;
    hal_wifi_link_stat_t link_stat;
    if (netmgr_get_ip_state() == true)
    {
        ret = hal_wifi_get_link_stat(NULL, &link_stat);
        if ((ret == 0) && link_stat.is_connected) {
            *p_rssi = link_stat.wifi_strength;
            *p_channel = link_stat.channel;
        } else {
            return -1;
        }
    } else {
```

```
        return -1;
    }
    return 0;
}
```

iii. 实现蓝牙辅助配网关联的专用接口。

```
/**
 * @brief    Get Security level for Wi-Fi configuration with connection.
 *           Used for AP solution of router and App.
 *
 * @param None.
 * @return The security level:
   @verbatim
    3: aes128cfb with aes-key per product and aes-iv = random
    4: aes128cfb with aes-key per device and aes-iv = random
    5: aes128cfb with aes-key per manufacture and aes-iv = random
    others: invalid
   @endverbatim
 * @see None.
 */
int HAL_Awss_Get_Conn_Encrypt_Type()
{
    char invalid_ds[DEVICE_SECRET_LEN + 1] = {0};
    char ds[DEVICE_SECRET_LEN + 1] = {0};
    // 用于区分该种配网方式的加密方式是使用"一机一密"还是"一型一密"
    // 如果DeviceSecret可以获取到，则使用"一机一密"的高安全级别加密方式
    // 如果设备本地未找到DeviceSecret，则降级使用"一型一密"的次高安全级别加密方式
    HAL_GetDeviceSecret(ds);
    if (memcmp(invalid_ds, ds, sizeof(ds)) == 0)
        return 3;
    memset(invalid_ds, 0xff, sizeof(invalid_ds));
    if (memcmp(invalid_ds, ds, sizeof(ds)) == 0)
        return 3;
    return 4;
}
```

> **说明** 以上Wi-Fi HAL层实现基于AliOS Things封装的Wi-Fi Driver，BK7231U移植了AliOS Things的Wi-Fi Driver。这部分的实现说明可参见Wi-Fi芯片移植，具体的代码实现请参见以下目录中内容。
>
> - /Living_SDK/framework/netmgr/
> - /Living_SDK/platform/mcu/bk7231u/hal/

五、生成设备固件

生活物联网平台SDK提供了蓝牙辅助配网的示例应用，完成移植后，可以基于示例应用编译蓝牙辅助配网设备固件，并对蓝牙辅助配网的整体功能进行快速验证。

SDK版本	编译指令

SDK版本	编译指令
1.3.0以上版本	./build.sh example smart_outlet bk7231udevkitc MAINLAND ONLINE 1
1.3.0及以下的版本	cd Living_SDK aos make clean aos make comboapp@bk7231udevkitc btstack=vendor

1. 初始化与启动应用。

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

   示例应用目录中的*app_entry.c*文件的 `application_start` 函数，该函数主要实现如下功能。

   - 系统初始化与启动
   - 设备调试日志等级、设备信息、设备诊断模块设置
   - Wi-Fi模块初始化，以及相关的事件订阅
   - 串口交互命令 `cli` 的初始化和注册
   - Wi-Fi模块的启动，与相关任务的创建

2. 动态开启或关闭设备的蓝牙辅助配网模式。

   实现设备系统与各模块的初始化后，通过如下代码实现，可以动态开启或关闭设备的蓝牙辅助配网模式。

   ```
   // 关闭蓝牙辅助配网功能
   breeze_awss_stop();
   // 开启蓝牙辅助配网功能
   breeze_awss_start();
   ```

3. 实现蓝牙辅助配网工作流程。

   示例应用的其核心流程实现在*示例应用*目录中的*combo_net.c*文件的 `combo_net_init` 函数。该函数主要实现以下功能。

   - 注册应用层的回调，会触发去连接路由器。
   - 将设备信息设置到下层的Breeze SDK中。
   - 初始化并开启蓝牙辅助配网的BLE通信通道。

   设备发出BLE广播（广播里面会携带蓝牙辅助配网功能的标识），处于BLE广播状态的设备可以被移动端App扫描发现。此时App上的操作过程以及设备的状态变化说明如下。

   i. 设备进入蓝牙辅助配网状态时，开始持续发出BLE广播，广播里携带蓝牙辅助配网功能的标识。
   ii. 移动端App扫描发现该待配网的Combo设备，并从移动端App发起与该设备建立连接的请求。

      建立连接时，移动端App与设备之间通过安全认证，确保建立的BLE连接是安全可靠的。

   iii. App通过BLE安全连接通道下发配网信息给设备端。

iv. 设备端通过Breeze SDK，接收和解析配网信息。

设备端在获取到配网信息后，触发注册的 `combo_service_event` 事件处理函数（此时底层SDK已经获取到了Wi-Fi联网需要的路由器的SSID和密码等信息）。 `combo_service_event` 事件处理函数时，会检测路由器的SSID和信号强度等情况，并向路由器发起连接请求。

v. 连接路由器成功后，获取IP地址，启动设备连云。

> **说明** 整个过程中如果出现异常，设备端会通过异常自检生成关键错误码信息，通过和移动端App之间的BLE连接传回，并在移动端App界面显示。

```
int combo_net_init()
{
    breeze_dev_info_t dinfo = { 0 };
    // 注册获取到App端传过来的配网信息时的回调，会触发去连接路由器的动作
    aos_register_event_filter(EV_BZ_COMBO, combo_service_event, NULL);
    if ((strlen(g_combo_pk) > 0) && (strlen(g_combo_ps) > 0) \
    && (strlen(g_combo_dn) > 0) && (strlen(g_combo_ds) > 0) && g_combo_pid > 0) {
        // 设备信息设置到下层的SDK中，蓝牙辅助配网与一般的Wi-Fi设备相比，
        // 会多一个PID的设备信息，专用于蓝牙的通信握手等功能之用
        dinfo.product_id = g_combo_pid;
        dinfo.product_key = g_combo_pk;
        dinfo.product_secret = g_combo_ps;
        dinfo.device_name = g_combo_dn;
        dinfo.device_secret = g_combo_ds;
        // 初始化蓝牙辅助配网所需的BLE协议栈、Breeze SDK等，并注册获取到配网信息的回调
        // 同时正式启动蓝牙辅助配网
        breeze_awss_init(apinfo_ready_handler, &dinfo);
        breeze_awss_start();
    } else {
        // 如果设备信息设置有误，则无法进行蓝牙辅助配网
        printf("combo device info not set!\n");
    }
    return 0;
}
```

4. 调试蓝牙辅助配网设备。

编译生成设备固件，并烧录到相应的开发板之后，可以通过如下的串口命令，触发设备的相应动作（生活物联网平台SDK 1.3.0及之后的版本才支持动态串口命令交互）。

i. 烧录设备证书信息。

蓝牙相关的设备证书包括ProductKey、DeviceName、DeviceSecret、ProductSecret、ProductID（您在生活物联网控制台创建的产品与设备后平台自动颁发的），在设备上电初始执行以下命令烧录设备证书信息。

```
linkkey ProductKey DeviceName DeviceSecret ProductSecret ProductID
```

ii. 开启一键配网功能。

此时如果蓝牙辅助配网进行中，会先自动关闭蓝牙辅助配网功能。

```
awss
active_awss
```

iii. 开启蓝牙辅助配网功能。

在设备正常启动后，默认会进入信道扫描的状态。此时如果一键配网进行中，会先自动关闭一键配网功能。

```
ble_awss
```

启动蓝牙辅助配网后，设备会通过BLE广播自己的蓝牙辅助配网相关的设备信息，移动端的App可在其设备发现页面发现到处于待配网状态的设备。通过在移动端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移动端App界面实时展示。

iv. 清除设备配网信息。

```
reset
```

六、验证蓝牙辅助配网功能

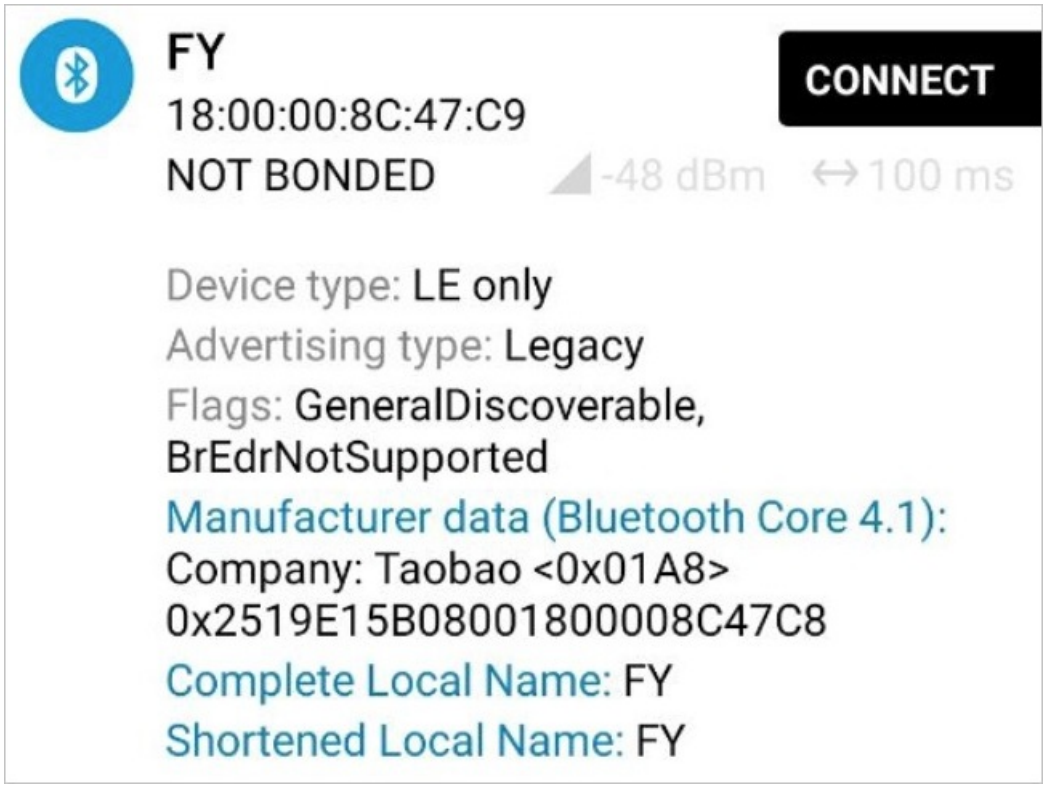
您可以使用生活物联网平台提供的公版App（云智能）来验证蓝牙辅助配网功能。

1. 下载公版App（2.7.5或以上的版本）。下载方式请参见云智能App介绍。
2. 登录公版App。
3. 打开手机系统的蓝牙开关。
4. 进入App的设备发现界面，开始扫描发现蓝牙辅助配网状态的设备。

   如果设备无法发现，请检查确保以下各项是否设置有误。

   - 设备是否已处于蓝牙广播状态（可使用相关工具搜索，如nRF Connect App）。

     蓝牙辅助配网广播内容示例说明如下。



广播数据内容主要位于广播的Manufacturer data字段中，其中包含Company ID和一些服务支持标识，以及MAC地址。其中MAC地址是表示Wi-Fi MAC（Combo实际上是Wi-Fi设备，此处蓝牙只是作为辅助配网，设备和云端通信链路通过Wi-Fi），此处Wi-Fi MAC地址为 `C8:47:8C:00:00:18` 。

- 设备证书信息是否设置正确（使用**linkkey**命令设置的内容）。
- App的账号环境与创建设备产品的站点是否对应。
- 控制台**人机交互**中是否配置了蓝牙辅助配网的配网方式。

5. App界面发现蓝牙辅助配网设备后，点击即可开始蓝牙辅助配网流程，其间需要输入手机连着的路由器的SSID和密码，并且在App UI界面确认设备已处于配网状态，用户界面确认完毕后，App端会去和设备建立蓝牙连接。

   如果蓝牙连接在短时间内断开，请检查确保以下项是否正常。

   - 设备证书信息是否设置正确（使用**linkkey**命令设置的内容）。
   - 设备BLE协议栈移植以及示例应用的实现是否检查确认无误
   - 手机连着的路由器的网络是否能正常使用

6. 手机将路由器信息传输给设备端，设备收到信息后去连接目标路由器。

   此时设备与手机之间的蓝牙连接不会断开，设备连接路由器、连接云端过程中如果有失败情况发生，设备会启动自检（需升级至生活物联网平台SDK 1.3.0之后的版本才支持失败详情自检的能力），并将自检结果通过蓝牙连接返回给手机，并在App界面上显示。

7. App界面跳转出设备的控制界面（如“灯”产品的控制界面可以开关灯，调整灯的亮度等）。

   此时，蓝牙辅助配网的功能调试成功。接下来您需要对设备产品的完整功能、稳定性能、成功率等进行严格测试把控，最终完成整个方案的量产和发布。

2.2.2. iOS App端开发

蓝牙辅助配网可以让手机App将WiFi热点信息通过蓝牙传递给设备，配网错误信息也可以通过蓝牙通道反馈给手机用户来定位问题。若您希望通过iOS系统对您的设备使用蓝牙进行配网，请根据本文档进行功能的开发。

前提条件

已完成控制台上产品开发工作，以及蓝牙辅助配网的设备端开发，请参见设备端开发。

操作步骤

1. 创建一个自有App，详细操作请参见创建自有App。
2. 选择下面任一方式获取SDK。
   - 从控制台下载并集成SDK（需勾选配网套餐包），请参见下载并集成SDK。
   - 在App工程中更新如下代码。

```
# SDK最低支持版本为iOS 9.0
platform :ios, '9.0'
# github官方pod源
source 'https://github.com/CocoaPods/Specs.git'
# 阿里云pod源
source 'https://github.com/aliyun/aliyun-specs.git'
# 需要替换下述"IMSDemoApp"为开发者App的target名称
target "IMSDemoApp" do
  ...
  # 配网SDK依赖
  pod 'IMSDeviceCenter', '1.11.5'
  # 蓝牙SDK依赖
  pod 'IMSBreezeSDK', '1.6.9'
  ...
end
```

3. 初始化SDK。

   蓝牙辅助配网依赖和云端的交互，在调用配网SDK之前需要先完成SDK初始化，可参见SDK初始化（API网关接口使用示例请参见API通道SDK）。

4. （可选）开发设备发现。

   设备发现接口提供发现附近的蓝牙辅助配网、零配、设备热点配网的待配设备和局域网已配设备的能力。这里介绍蓝牙辅助配网，示例代码会只选发现蓝牙辅助配网待配设备。

   > 说明 Wi-Fi&BLE Combo蓝牙模块广播的subType必须设置为2（表示蓝牙Wi-Fi双模设备），才能保障设备被蓝牙辅助配网方式发现。

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>
// 本地发现入口
[[IMLLocalDeviceMgr sharedMgr] startDiscovery:^(NSArray *devices, NSError *err) {
        //devices 为 IMLCandDeviceModel 对象array，
        // 可根据 IMLCandDeviceModel 中的 devType 区分待配网设备联网类型；
        // devType 为@"ble_subtype_2" 代表蓝牙辅助配网设备*/
    }];
```

5. 开发设备配网。

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>
/**
 * 第一步，设置待添加设备信息
 * 包含设备的productKey、productId信息
 * 配网的信息可以通过上面的发现接口来获取，也可以直接通过扫码的方式拿到配网所需要的信息开始配网。
 * 不是通过发现接口进行蓝牙辅助配网的需要确保当前确实有进入配网状态的combo设备，并能被发现。
*/
// 自行创建待配设备信息
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
// 商家后台注册的 productKey，不可为空
model.productKey = @"xxxx";
// 产品 ID， 蓝牙辅助配网必须
model.productId = @"xxx";
// 设置为蓝牙辅助配网模式
```

```
model.linkType = ForceAliLinkTypeBLE;
//设置待添加设备的基本信息
[kLkAddDevBiz setDevice:model];
/**
 * 第二步，开始配网
 * 设置配网信息回调
*/
//其中 self 为配网过程中 notifier 监听回调对象（代理）
[kLkAddDevBiz startAddDevice:self];
- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
}
// 用户引导页（一键配网和热点配网会有相关回调，指引用户接入相关操作）
- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode
{
    NSLog(@"notifyProvisionPrepare callback guide code : %ld", guideCode);
    if(guideCode == LKPGuideCodeOnlyInputPwd){
        // 输入WiFi及密码相关引导，执行第三步
        /**
         * 第三步，设置 Wi-Fi 信息
         * 设置需要设备连接的 Wi-Fi 的ssid、password、配网超时时间信息
        */
        // 需要设备连接的 Wi-Fi ssid，一般为当前 Wi-Fi
        NSSstring *ssid = @"example ssid";
        // 需要设备连接的 Wi-Fi password，一般为当前 Wi-Fi
        NSString *password = @"1qaz@WSX";
        NSInterger timeout = 60; （单位秒,s);//单位秒，目前最短只能设置60S
        [kLkAddDevBiz toggleProvision:ssid pwd:password timeout:timeout];
    }
}
-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback（正在进行配网...）");
}
/**
 通知上层UI：配网完成结果回调
 @param candDeviceModel 配网结果设备信息返回：配网失败时为 nil
 @param provisionError 错误信息
 */
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:
(NSError *)provisionError
{
    NSLog(@"配网结果：%@",candDeviceModel);
    /**
     * 第四步，接收配网结果
    */
    if(candDeviceModel != nil){
        NSLog(@"配网成功：%@",candDeviceModel);
    } else{
        NSLog(@"配网失败，错误信息；%@", provisionError);
    }
    // 配网结果：如果配网成功之后包含token，请使用配网成功带的token做绑定
}
```

6. 获取设备绑定Token。

   获取设备绑定Token需要productKey、deviceName、超时时间、轮询间隔。该接口提供从设备端获取绑定token的功能。配网SDK不包含绑定的逻辑，实现绑定逻辑可参见基于token方式设备绑定。

```
// self.productKey 和 self.deviceName 是配网成功后返回的物模型中的 productKey 和 deviceName
[[IMLLocalDeviceMgr sharedMgr] getDeviceToken:self.productKey deviceName:self.deviceNam
e timeout:20 resultBlock:^(NSString *token, BOOL boolSuccess) {
    NSLog(@"主动获取设备token：%@，boolSuccess：%d", token, boolSuccess);
    if(token){
    // 调用绑定接口进行设备绑定
    } else{
        NSLog(@"获取token失败（超时）");
    }
}];
```

2.2.3. Android App端开发

蓝牙可以辅助纯Wi-Fi方式来配网，当Wi-Fi配网出现错误时，还可以用来反馈错误信息，帮助用户定位问题。当您的自有App需要蓝牙辅助配网功能时，请根据本文档开发Android自有App中蓝牙辅助配网功能。

前提条件

已完成控制台上产品开发工作，以及蓝牙辅助配网的设备端开发，请参见设备端开发。

1. 创建一个自有App，详细操作请参见创建自有App。
2. 选择下面任一方式获取SDK。
   - 从控制台下载并集成SDK（需勾选配网套餐包），请参见下载并集成SDK。
   - 在App工程中添加如下代码。
     - maven仓库

```
maven {
    url "http://maven.aliyun.com/nexus/content/repositories/releases/"
}
```

     - gradle依赖

```
// 配网SDK依赖
api('com.aliyun.alink.linksdk:ilop-devicecenter:1.7.5')
// api通道 依赖
api('com.aliyun.alink.linksdk:api-client-biz:1.0.1')
// 蓝牙SDK依赖
api 'com.aliyun.alink.linksdk:breeze-biz:1.1.4'
```

3. 初始化SDK。

   蓝牙辅助配网依赖和云端的交互，在调用配网SDK之前需要先完成SDK初始化，请参见SDK初始化（API网关接口的使用示例请参见API通道SDK）。

4. （可选）开发发现设备。

   设备发现接口提供发现附近的蓝牙辅助配网、零配、设备热点配网的待配设备和局域网已配设备的能力。这里介绍蓝牙辅助配网，示例代码会只选发现蓝牙辅助配网待配设备。

> **说明** Wi-Fi&BLE Combo蓝牙模块广播的subType必须设置为2（表示蓝牙Wi-Fi双模设备），才能保障设备被蓝牙辅助配网方式发现。

```
// 开始发现蓝牙辅助配网待配设备
// 第三个参数是其它发现方式需要的，这里直接置为null
final EnumSet<DiscoveryType> discoveryTypes = EnumSet.of(DiscoveryType.BLE_ENROLLEE_DEV
ICE);
LocalDeviceMgr.getInstance().startDiscovery(context, enumSet, null, new IDeviceDiscover
yListener() {
    @Override
    public void onDeviceFound(DiscoveryType discoveryType, List<DeviceInfo> list) {
        // 发现的设备
        // discoveryType=BLE_ENROLLEE_DEVICE 表示发现的是蓝牙Wi-Fi双模设备
        //如果发现到蓝牙辅助配网的待配设备，返回的设备信息只包含productId，
        //用户需要根据productId获取产品对应的productKey相关信息，然后开始后续的配网。
    }
});
```

> **说明** 在联调蓝牙辅助配网发现和配网的时候，需要确保App具有位置权限，且蓝牙处于打开状态； 在app开发时需要做相应的权限检测和权限申请。

5. 开发设备配网。

   设备配网主要包含四个流程（设置配网设备信息、开始配网、设置Wi-Fi信息、接收配网结果），详细介绍参见以下代码示例。

```
/**
 * 第一步 设置配网设备信息
 * 包含设备的productKey、productId信息
 * 配网的信息可以通过上面的发现接口来获取，也可以直接通过扫码的方式拿到配网所需要的信息开始配网。
 * 不是通过发现接口进行蓝牙辅助配网的需要确保当前确实有进入配网状态的combo设备，并能被发现。
*/
DeviceInfo deviceInfo = new DeviceInfo();
// 商家后台注册的 productKey，不可为空
deviceInfo.productKey = "xx";
// 产品 ID， 蓝牙辅助配网必须
deviceInfo.productId = "xxx";
deviceInfo.linkType = "ForceAliLinkTypeBLE"; // 默认一键配网
//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(deviceinfo);
/**
 * 第二步 开始配网
 * 设置配网信息回调
*/
AddDeviceBiz.getInstance().startAddDevice(context, new IAddDeviceListener(){
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
        // 参数检测回调
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
        // prepareType = 1 执行第三步
        /**
```

```
        /**
         * 第三步 设置Wi-Fi信息
         * 设置需要设备连接的Wi-Fi的ssid、password、配网超时时间信息
        */
         // 需要设备连接Wi-Fi的ssid，一般为当前Wi-Fi
        String ssid = "ssid";
        // 需要设备连接Wi-Fi的password，一般为当前Wi-Fi
        String ssid = "xxxxxxxx";
        int timeout = 60;//单位秒 目前最短只能设置60S
        AddDeviceBiz.getInstance().toggleProvision(ssid, password, timeout);
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCo
de errorCode) {
        /**
         * 第四步 接收配网结果
         * isSuccess = true 表示配网成功，deviceInfo会包含设备的productKey、deviceName相关
信息
         * isSuccess = false 表示配网失败，errorCode会包含配网失败的错误code和原因;
        */
        // 配网结果 如果配网成功之后包含token，请使用配网成功带的token做绑定
    }
});
```

6. 获取设备绑定Token。

   获取设备绑定Token需要productKey、deviceName、超时时间、轮询间隔。该接口提供从设备端获取绑定token的功能。配网SDK不包含绑定的逻辑，实现绑定逻辑可参见基于token方式设备绑定。

```
/**
 * 获取设备端绑定的token
 * 该接口需要设备配网成功，且设备连云成功后，才会出现成功返回。
 * 60*1000 ms表示总超时时间，5*1000表示每5s去查询一次
*/
LocalDeviceMgr.getInstance().getDeviceToken(context, productKey, deviceName, 60*1000, 5
*1000,  new IOnDeviceTokenGetListener() {
            @Override
            public void onSuccess(String token) {
                // TODO 设备绑定
            }
            @Override
            public void onFail(String reason) {
            }
        });
```

2.3. Combo设备快速配网实践

2.3.1. Android App端开发

对于Combo设备（同时支持Wi-Fi和BLE），除了可以先配网再绑定设备（即蓝牙辅助配网开发实践），还可以先绑定设备再配网，且该方式配网速度更快。当您的业务场景需要快速配网时，可根据本文档来进行App端的Android系统开发。

操作步骤

1. 创建一个自有App，详细操作请参见创建自有App。
2. 获取App端SDK。详细操作请参见下载并集成SDK。
3. 初始化SDK。详细操作请参见SDK初始化。
4. 开发Combo设备发现。

   Combo设备通过调用LocalDeviceMgr中的startDiscovery方法来实现设备发现。如果您已实现过设备发现业务，则在该基础上，在EnumSet<DiscoveryType>入参中新增一个DiscoveryType.BLE_ENROLLEE_DEVICE类型即可。

   i. 发起本地发现。

   ```
   EnumSet<DiscoveryType> discoveryTypes=EnumSet.of(DiscoveryType.CLOUD_ENROLLEE_DEVIC
   E, DiscoveryType.SOFT_AP_DEVICE, DiscoveryType.LOCAL_
   ONLINE_DEVICE, DiscoveryType.BLE_ENROLLEE_DEVICE);
   LocalDeviceMgr.getInstance().startDiscovery(AApplication.getInstance
   (), discoveryTypes, null, new IDeviceDiscoveryListener() {
    @Override
    public void onDeviceFound(DiscoveryType discoveryType, L
   ist<DeviceInfo> list) {
    }
    });
   ```

   ii. 调用获取产品productKey接口来使用productId换取设备的ProductKey。

   iii. 调用本地发现设备列表信息过滤接口来过滤掉非法设备（如已绑定的设备等）。

   > **说明** 对于Combo设备，该接口的请求参数中deviceName需配置为Combo设备的MAC地址。

5. 开发Combo设备绑定。

   下面以公版App为示例介绍Combo快速配网流程，您可以根据自己的业务逻辑来实现。

   i. 调用蓝牙上线接口DevService.breezeSubDevLogin，传入ProductKey、MAC等参数。

   示例代码如下。

   ```
   DevService.breezeSubDevLogin(pk, mac, new DevService.ServiceListener() {
       @Override
       public void onComplete(boolean b, Object o) {
       }
   });
   ```

   调用接口后SDK会通过onComplete接口返回蓝牙上线结果。如果蓝牙上线成功则继续绑定操作；蓝牙上线失败，则流程结束。

   ii. 调用基于时间窗口方式的绑定设备接口在云端绑定设备。

iii. 调用DevService.notifySubDeviceBinded接口通知设备绑定结果。

示例代码如下。

```
private void tmpNotify(String iotId){
        ALog.d(TAG,"tmpNotify->"+iotId);
        SubDevInfo subDevInfo=new SubDevInfo();
        subDevInfo.iotId=iotId;
        if (netType==ILopNetTypeCodes.NET_TYPE_BT){
            ALog.d(TAG,"mac->"+mac);
            subDevInfo.deviceName=mac;
        }else {
            ALog.d(TAG,"dn->"+dn);
            subDevInfo.deviceName=dn;
        }
        subDevInfo.productKey=pk;
// 调用DevService.notifySubDeviceBinded接口通知设备已绑定成功
        DevService.notifySubDeviceBinded(subDevInfo,new DevService.ServiceListener(
){
            @Override
            public void onComplete(boolean b, @Nullable Object o) {
                ThreadTools.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        gotoStepFinish();
                    }
                });
            }
        });
    }
```

6. 开发Combo设备配网。

> **说明**
>
> Combo设备类型为DiscoveryType.COMBO_SUBTYPE_0X04_DEVICE，表示已配网，此时您可忽略该步骤的以下操作。

i. 开始配网时，设置wifistate为DeviceWifiStatus_Setting，表示Wi-Fi正在配置中。

示例代码如下。

```
DevService.setWifiStatus(iotID, TmpEnum.DeviceWifiStatus.DeviceWifiStatus_Setting,
new DevService.ServiceListenerEx() {
   @Override
public void onComplete(boolean b, @Nullable String s) {
}});
```

ii. 调用AddDeviceBiz.getInstance().setDevice设置配网信息。

示例代码如下。

```
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.devType = ILopNetTypeCodes.DEV_TYPYE_BLE_SUBTYPE_3;
deviceInfo.mac = mac;
deviceInfo.linkType = LinkType.ALI_BLE.getName();
RegionInfo regionInfo = new RegionInfo();
regionInfo.shortRegionId = 0;
deviceInfo.regionInfo = regionInfo;
ProvisionConfigParams params = new ProvisionConfigParams();
params.ignoreSoftAPRecoverWiFi = true;
params.enableGlobalCloudToken = true;
ProvisionConfigCenter.getInstance().setProvisionConfiguration(params);
AddDeviceBiz.getInstance().setDevice(deviceInfo);
```

iii. 调用AddDeviceBiz.getInstance().startAddDevice方法发起配网。

示例代码如下。

```
AddDeviceBiz.getInstance().startAddDevice(AApplication.getInstance(), new IAddDevic
eListener() {
    @Override
    public void onPreCheck(boolean b, DCErrorCode dcErrorCode) {
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
        if (prepareType == 1) {
//传入SSID和Wi-Fi密码开始配网
           AddDeviceBiz.getInstance().toggleProvision(ssid, password, 60);
        }
    }
    @Override
    public void onProvisioning() {
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
        if ((provisionStatus != null) && provisionStatus.code() == ProvisionStatus.
BLE_DEVICE_SCAN_SUCCESS.code()) {
            ALog.d(TAG, "BLE_DEVICE_SCAN_SUCCESS");
            if (provisionStatus.getExtraParams() != null) {
        //扫描到该设备
                String devType = (String) provisionStatus.getExtraParams().get(ILop
NetTypeCodes.KEY_DEV_TYPE);
                String bleMac = (String) provisionStatus.getExtraParams().get(ILopN
etTypeCodes.KEY_BLE_MAC);
                String prouctID = (String) provisionStatus.getExtraParams().get(ILo
pNetTypeCodes.KEY_PRODUCT_ID);
                ALog.d(TAG, "devType->" + devType + " bleMac->" + bleMac + " prouct
ID->" + prouctID);
                if (ILopNetTypeCodes.DEV_TYPYE_BLE_SUBTYPE_3.equals(devType)){
            //继续配网
                    AddDeviceBiz.getInstance().continueProvision(null);
                }
            }
```

```
            }
        }
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErr
orCode dcErrorCode) {
        String message = "onProvisionedResult. isSuccess:" + isSuccess + " deviceIn
fo:" + deviceInfo + " dcErrorCode:" + dcErrorCode;
        ALog.d(TAG, message);
        ThreadTools.runOnUiThread(new Runnable() {
            @Override
            public void run() {
               AddDeviceBiz.getInstance().stopAddDevice();//配网完成，结束当前配网流程
                if (isSuccess) {
                //配网成功
                } else {
                  //配网失败
                }
            }
        });
    }
});
```

7. 调用DevService#setWifiStatus向TMP通知设备的配网结果。

   示例代码如下。

```
DevService.setWifiStatus(iotID, TmpEnum.DeviceWifiStatus.state, new DevService.ServiceL
istenerEx() {
   //配网成功state为DeviceWifiStatus_Set；配网不成功state为DeviceWifiStatus_NotSet
@Override
    public void onComplete(boolean b, @Nullable String s) {
        }
});
```

2.3.2. iOS App端开发

对于Combo设备（同时支持Wi-Fi和BLE），除了可以先配网再绑定设备（即蓝牙辅助配网开发实践），还可以先绑定设备再配网，且该方式配网速度更快。当您的业务场景需要快速配网时，可根据本文档来进行App端的iOS系统开发。

操作步骤

1. 创建一个自有App，详细操作请参见创建自有App。
2. 获取App端SDK。

   在代码工程中添加以下依赖。

```
pod 'IMLDeviceCenter'
pod 'IMLDeviceKit'
pod 'IMSThingCapability'
```

3. 初始化SDK。具体操作，请参见SDK初始化。
4. 开发Combo设备发现。

i. 调用SDK发现接口startDiscoveryWithFilter，发现周围的Combo设备。

调用示例代码如下。

```
[[IMLLocalDeviceMgr sharedMgr] startDiscoveryWithFilter:nil
                              discoverBlcok:^(NSArray *devices, NSError *err) {
}];
//Combo设备的类型，ble_subtype_3表示未配置Wi-Fi；此时您需要绑定设备和配网设备。
// ble_subtype_4表示已配置过Wi-Fi，此时您只需绑定设备即可。
```

其中，NSArray的示例如下。

```
(
    "{\n    bindState = 0;\n    cipherRandomStr = 00000000000000000000000000000000;
\n    cipherType = 0;\n    devType = \"ble_subtype_12\";\n    deviceEncrypt = 1;\n
deviceIsReset = 0;\n    enableGlobalCloudToken = 0;\n    ignoreLocationPermisionChe
ck = 0;\n    inputNetType = 0;\n    isBindSucc = 0;\n    isBinded = 0;\n    isFinis
hLocalFindCombo = 0;\n    linkType = 0;\n    mac = a4cdxxxx0e9;\n    productId = 59
xxx0;\n    productInfoModel = \"<IMSNativeProductInfoModel: 0x2802b7e20>\";\n    pr
oductKey = a1GxxxxZK;\n    regionNode = 0;\n    softApNoNeedSwitchBackRouter = 0;\n
}"
)
```

ii. 调用获取产品productKey获取productKey。

iii. 调用本地发现设备列表信息过滤接口来过滤掉非法设备（如已绑定的设备等）。

> **说明** 对于Combo设备，该接口的请求参数中deviceName需配置为Combo设备的MAC地址。

5. 开发Combo设备绑定。

i. 将App作为网关，Combo设备以子设备方式登录云端，并使用设备MAC地址换取DeviceName。

示例代码如下。

```
[IMSSubDeviceService subDeviceAuthenLogin:@{@"productKey":productKey ?: "",
                   @"deviceName”:mac ?: ""} ///这里请务必填写设备的MAC地址
        resultBlock:^(NSDictionary * _Nullable object, NSError * _Nullable error) {
      // 成功则读取到DeviceName
        NSString *deviceName = object[@"deviceName"];
    }];
```

ii. 调用基于时间窗口方式的绑定设备接口在云端绑定设备。

该接口的请求参数中，groupIds默认为 `@[]` （表示空）即可，homeId配置为当前家的ID。

iii. 调用notifySubDeviceBinded接口通知设备绑定结果。

示例代码如下。

```
[IMSSubDeviceService notifySubDeviceBinded:YES deviceInfo:deviceInfo completeBlock:
^(BOOL succeeded, NSError * _Nullable error) {
     // 此处succeeded == yes为绑定成功，流程完全结束
 }];
```

其中，`deviceInfo:deviceInfo` 的示例如下。

```
NSDictionary *deviceInfo = @{@"deviceName": mac ?: @"",
                                                    @"productKey": productKey ?: @
"",
                                                    @"iotId": iotId ?: @""};
```

iv. 设置Wi-Fi状态。

配网前，将wifistate设置为DeviceWifiStatus_NotSet，表示Wi-Fi的状态为未配置。示例代码如下。

```
IMSThing *thing = [kIMSThingManager buildThing:iotId];
[[thing getThingActions] setWifiStatus:DeviceWifiStatus_NotSet responseHandler:^(IM
SThingActionsResponse * _Nullable response) {
}];
```

6. 开发Combo设备配网。

> **说明**
>
> Combo设备的类型为ble_subtype_4时，表示已配网，此时您可忽略该步骤的操作。

i. 传入物模型，以及SSID、密码、超时时间等。设置代理发起配网。

示例代码如下。

```
[kLkAddDevBiz setDevice:self.currentModel];    // self.currentModel 为发现逻辑完成之
后的model
[kLkAddDevBiz startAddDevice:self];
[kLkAddDevBiz toggleProvision:ssid pwd:pwd timeout:60];
```

ii. 开始配网时，设置wifistate为DeviceWifiStatus_Setting，表示Wi-Fi正在配置中。

示例代码如下。

```
IMSThing *thing = [kIMSThingManager buildThing:iotId];
[[thing getThingActions] setWifiStatus:DeviceWifiStatus_Setting responseHandler:^(I
MSThingActionsResponse * _Nullable response) {
}];
```

iii. 手动继续配网。

由于设备先绑定再配网，当找到蓝牙设备时，您还需要在配网环节的蓝牙配置中继续手动配网。示例代码如下。

```
/**
蓝牙辅助配网模式相关回调
通知上层UI：蓝牙辅助配网相关回掉提示
@param status 状态码：
1表示还未找到蓝牙设备，请检查设备是否初始化，
2表示已经找到蓝牙设备
@param dic 状态信息；
status=2，dic 可能为：{@"devType":@"ble_subtype_3"} 代表支持先绑定控制后配置Wi-Fi信息的蓝牙Combo设备，可先执行绑定控制，之后调用continueProvision接口继续配网流程。
*/
- (void)notifyProvisioningNoticeForBleConfig:(int)status withInfo:(NSDictionary *)dic {
if (status == 2 && [dic[@"devType"] isEqualToString:@"ble_subtype_3"]) {
[kLkAddDevBiz continueProvision:nil];
}
}
```

iv. 监听代理回调，处理配网结果。

示例代码如下。

```
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:(NSError *)provisionError {
    // 处理配网结果，candDeviceModel有数据返回，provisionError中无error则为配网成功
}
```

7. 配网结束后设置配网结果。

示例代码如下。

```
IMSThing *thing = [kIMSThingManager buildThing:iotId];
[[thing getThingActions] setWifiStatus:state responseHandler:^(IMSThingActionsResponse * _Nullable response) {
  //配网成功state为DeviceWifiStatus_Set；如配网不成功state为DeviceWifiStatus_NotSet
}];
```

8. （可选）当您需要判断设备是否为Combo设备以及wifistate状态时，可调用以下代码查看。

```
IMSThing *thing = [kIMSThingManager buildThing:iotId];
[[thing getThingActions] getDeviceNetTypesSupported:^(IMSThingActionsResponse * _Nullab
le response) {
        NSInteger netTypes = [(NSNumber *)response.dataObject integerValue];
    DeviceWifiStatusType type = [[thing getThingActions] getWifiStatus];
         // DeviceWifiStatus_Init表示初始化Wi-Fi
         // DeviceWifiStatus_Setting表示初始化设置中
         // DeviceWifiStatus_NotSupport表示不支持设置Wi-Fi
         // DeviceWifiStatus_NotSet表示Wi-Fi设置失败
         // DeviceWifiStatus_Set表示Wi-Fi设置成功
    BOOL isComboDevice = (netTypes & NET_WIFI) && (netTypes & NET_BT) && (type != Devic
eWifiStatus_NotSupport);
         // 返回YES表示Combo设备，返回NO表示非Combo的其他设备
}];
```

2.4. 批量配网开发实践

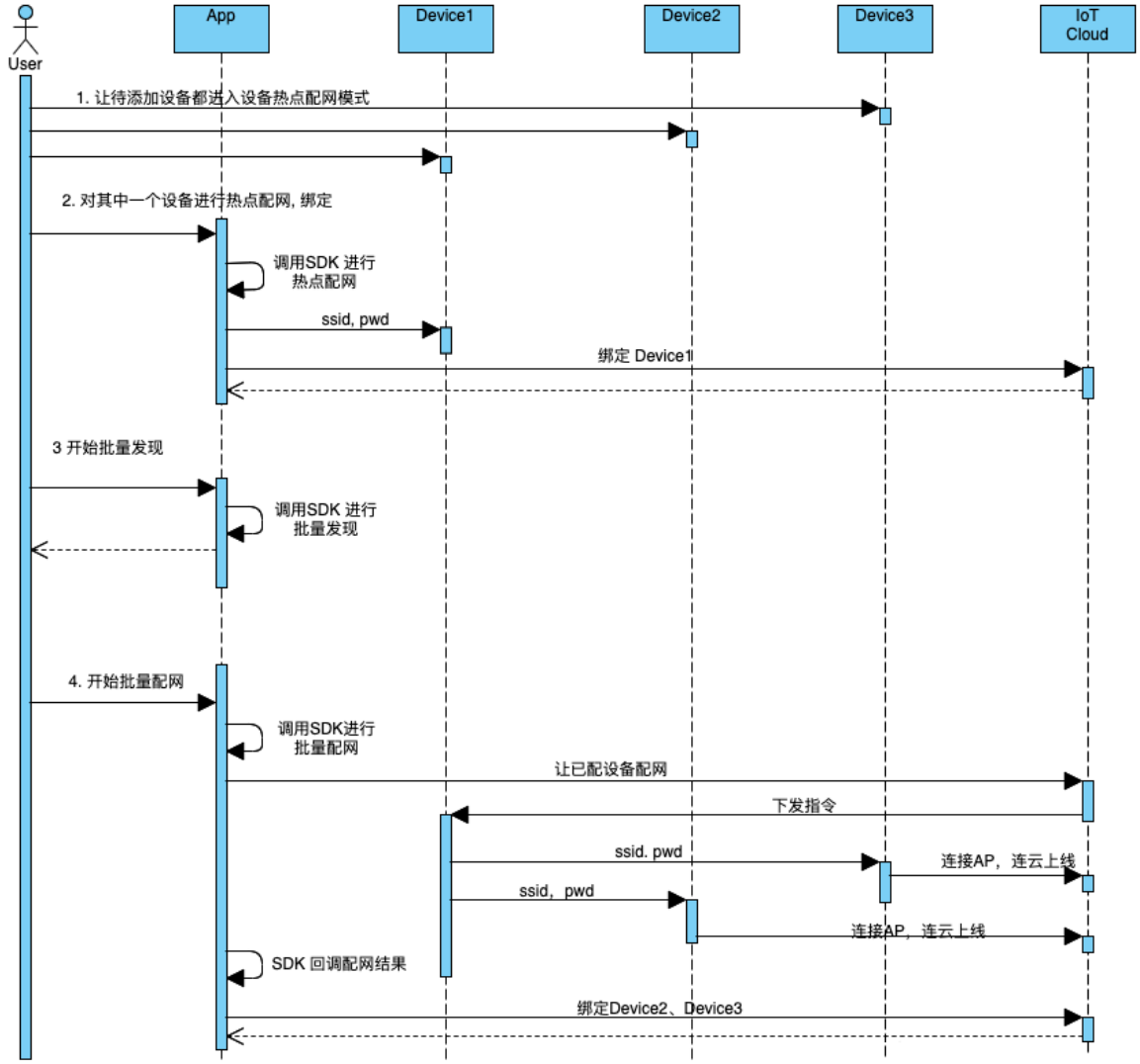
生活物联网平台支持您在自有App中集成批量配网功能，可以实现一次性为多个相同型号的设备配网，单次最多可批量配网20个设备。

前提条件

- 已在控制台创建产品，并完成产品的开发工作。
- 产品的默认配网方案需要设置为设备热点配网。详细请参见配置配网引导。
- 已完成设备热点配网和零配配网功能的开发。详细请参见Wi-Fi设备配网方案介绍。
- 已完成App端配网功能的开发。详细请参见配网开发指南。

方案介绍

批量配网方案是基于设备热点配网方案的基础上开发的方案，因此设备需要支持设备热点配网和零配配网。批量配网方案的时序图如下所示。



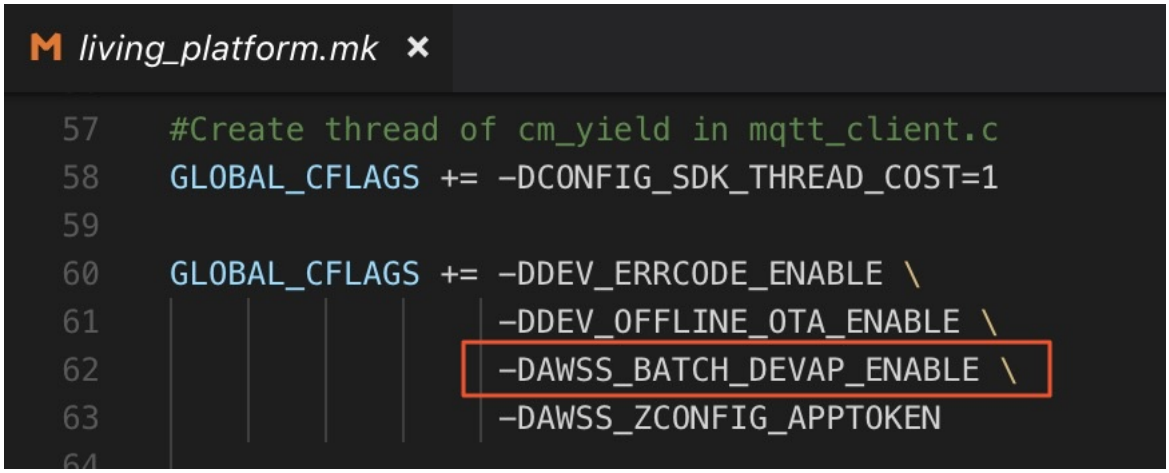
1. 设备上电或操作设备按键，将所有待添加的设备进入设备热点配网模式。
2. 通过App对其中一个设备进行设备热点配网，并完成配网及绑定操作。调用的SDK请参见Android App SDK和iOS App SDK。
3. 指定2中完成配网的设备为主设备，并通过主设备去发现其他待配网设备。

   主设备发送广播消息，其他待配网设备接收到广播消息后切换至零配模式。此时，主设备可以发现进入零配模式的待配网设备，并在App SDK回调中反馈搜索到的设备信息。
4. 3中返回的其他待配网设备，通过零配配网方式完成配网，并在App SDK回调中会反馈配网结果。

集成设备端

1. 获取生活物联网平台SDK V1.3.0及以上版本。请参见获取SDK。
2. 使能SDK中功能模块。

   在应用的mk文件中增加配置项 `GLOBAL_CFLAGS += -DAWSS_BATCH_DEVAP_ENABLE` 。如您基于living_platform开发，在*living_platform.mk*文件中已包含该配置项，如下图所示。



设备端SDK以及芯片需要支持的基本能力如下。

- SDK的基本能力要求
  - 支持设备热点配网功能
  - 支持零配配网功能
  - 支持设备热点批量配网功能
- 芯片或模组的能力要求
  - 支持设备热点开启状态下接收Wi-Fi管理帧能力
  - 支持连接AP状态下切信道发送Wi-Fi管理帧能力
  - 支持设备热点批量配网的设备，设备热点必须设置在1信道、6信道或11信道中

3. 处理应用中调用SDK功能模块接口以及回调。
   i. 在应用示例中的 `awss_open_dev_ap()` 函数中增加 `awss_dev_ap_reg_modeswit_cb()` 函数的调用。

```
void awss_open_dev_ap(void *p)
{
    iotx_event_regist_cb(linkkit_event_monitor);
    LOG("%s\n", __func__);
    if (netmgr_start(false) != 0) {
        aos_msleep(2000);
#ifdef AWSS_BATCH_DEVAP_ENABLE
        awss_dev_ap_reg_modeswit_cb(awss_dev_ap_modeswitch_cb); //增加此调用
#endif
        awss_dev_ap_start();
    }
    aos_task_exit(0);
}
```

   ii. 实现应用层的回调函数 `awss_dev_ap_modeswitch_cb()` 。

```
#ifdef AWSS_BATCH_DEVAP_ENABLE
#define DEV_AP_ZCONFIG_TIMEOUT_MS  120000 // (ms)
extern void awss_set_config_press(uint8_t press);
extern uint8_t awss_get_config_press(void);
extern void zconfig_80211_frame_filter_set(uint8_t filter, uint8_t fix_channel);
void do_awss_dev_ap();
```

```
static aos_timer_t dev_ap_zconfig_timeout_timer;
static uint8_t g_dev_ap_zconfig_timer = 0;   // this timer create once and can rest
art
static uint8_t g_dev_ap_zconfig_run = 0;
static void timer_func_devap_zconfig_timeout(void *arg1, void *arg2)
{
    LOG("%s run\n", __func__);
    if (awss_get_config_press()) {
        // still in zero Wi-Fi provision stage, should stop and switch to dev ap
        do_awss_dev_ap();
    } else {
        // zero Wi-Fi provision finished
    }
    awss_set_config_press(0);
    zconfig_80211_frame_filter_set(0xFF, 0xFF);
    g_dev_ap_zconfig_run = 0;
    aos_timer_stop(&dev_ap_zconfig_timeout_timer);
}
static void awss_dev_ap_switch_to_zeroconfig(void *p)
{
    LOG("%s run\n", __func__);
    // Stop dev ap Wi-Fi provision
    awss_dev_ap_stop();
    // Start and enable zero Wi-Fi provision
    iotx_event_regist_cb(linkkit_event_monitor);
    awss_set_config_press(1);
    // Start timer to count duration time of zero provision timeout
    if (!g_dev_ap_zconfig_timer) {
        aos_timer_new(&dev_ap_zconfig_timeout_timer, timer_func_devap_zconfig_timeo
ut, NULL, DEV_AP_ZCONFIG_TIMEOUT_MS, 0);
        g_dev_ap_zconfig_timer = 1;
    }
    aos_timer_start(&dev_ap_zconfig_timeout_timer);
    // This will hold thread, when awss is going
    netmgr_start(true);
    LOG("%s exit\n", __func__);
    aos_task_exit(0);
}
int awss_dev_ap_modeswitch_cb(uint8_t awss_new_mode, uint8_t new_mode_timeout, uint
8_t fix_channel)
{
    if ((awss_new_mode == 0) && !g_dev_ap_zconfig_run) {
        g_dev_ap_zconfig_run = 1;
        // Only receive zero provision packets
        zconfig_80211_frame_filter_set(0x00, fix_channel);
        LOG("switch to awssmode %d, mode_timeout %d, chan %d\n", 0x00, new_mode_tim
eout, fix_channel);
        // switch to zero config
        aos_task_new("devap_to_zeroconfig", awss_dev_ap_switch_to_zeroconfig, NULL,
2048);
    }
}
#endif
```

4. 编译固件，并烧录到设备中。

集成iOS App端

1. 获取SDK。

   确保配网SDK版本升级到1.11.0及以上，建议使用官网最新版本。

   您可以从生活物联网平台的控制台下载SDK（请参见下载并集成SDK），也可以集成以下代码。

```
pod 'IMSDeviceCenter', '~>1.11.0'
```

2. 开始或停止批量发现。

```
/**
 批量发现功能，通过已配网设备发现周边其他设备，需特定设备支持
 @param targetProductKey 搜索目标设备ProductKey，若为nil，则目标为所有设备
 @param searcherProductKey 作为搜索者的已配网设备的ProductKey
 @param searchDeviceName 作为搜索者的已配网设备的DeviceName
 @param batchResultBlock 搜索到设备时触发的回调，会调用多次
 */
- (void)startBatchDiscoveryForTargetProductKey:(NSString *) targetProductKey bySearcher
ProductKey:(NSString *) searcherProductKey deviceName:(NSString *) searchDeviceName res
ultBlock:(void(^)(NSArray * devices, NSError * err))batchResultBlock;
/**
 停止已配网设备搜索周边设备
 */
- (void)stopBatchDiscovery;
```

3. 开始批量配网。

```
    IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
    model.productKey = productKey; //待配网设备的ProductKey
    model.regProductKey = productKey; // 已配好设备的ProductKey
    model.regDeviceName = deviceName; // 已配好设备的DeviceName
    model.linkType = ForceAliLinkTypeZeroInBatches; //指定配网模式为批量配网
    [[IMLAddDeviceBiz sharedInstance] setDevice:model];
    [[IMLAddDeviceBiz sharedInstance] startAddDevice:notifier];
```

4. 批量配网成功回调。

   与普通配网类似，回调会通过notifyProvisionResult反馈。每成功配网一个设备就回调一次。若发生回调失败，则表示此次批量配网全部失败或者整体超时。

```
/**
 通知上层UI：配网完成结果回调
 @param candDeviceModel 配网结果设备信息返回：配网失败时为nil
 @param provisionError 错误信息
 */
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:
(NSError *)provisionError;
```

集成Android App端

1. 获取SDK。

   您可以从生活物联网平台的控制台下载SDK（请参见下载并集成SDK），也可以集成以下代码。

```
// maven仓库地址
maven {
    url "http://maven.aliyun.com/nexus/content/repositories/releases"
}
api('com.aliyun.alink.linksdk:ilop-devicecenter:1.7.3')
```

2. 开始或停止批量发现。

   API reference 参见 LocalDeviceMgr。

```
/**
 * 参数无效会抛出 IllegalArgumentException
 * @param context context
 * @param params {@link BatchDiscoveryParams}
 * @param listener discovery results, callback at least once
 */
void startBatchDiscovery(Context context, BatchDiscoveryParams params, IDiscovery liste
ner);
/**
 * 停止批量配网
 * @param productKey 已配网设备ProductKey
 * @param deviceName 已配网设备DeviceName
 */
void stopBatchDiscovery(String productKey, String deviceName);
```

3. 开始批量配网并监听回调结果。

   startAddDevice各回调说明参见配网SDK。

```
DeviceInfo info = new DeviceInfo();
info.linkType = LinkType.ALI_ZERO_IN_BATCHES.getName();
info.productKey = pk; // 待配网设备ProductKey
info.regProductKey = regPk; // 已配网设备ProductKey
info.regDeviceName = regdn; // 已配网设备DeviceName
// 设置配网信息
AddDeviceBiz.getInstance().setDevice(info);
// 开始批量配网
AddDeviceBiz.getInstance().startAddDevice(context, new IAddDeviceListener() {
    @Override
    public void onPreCheck(boolean isSuccess, DCErrorCode dcErrorCode) {
        // 预检查
    }
    @Override
    public void onProvisionPrepare(int prepareType) {
    }
    @Override
    public void onProvisioning() {
        // 配网中
    }
    @Override
    public void onProvisionStatus(ProvisionStatus provisionStatus) {
    }
    @Override
    public void onProvisionedResult(boolean isSuccess, DeviceInfo deviceInfo, DCErrorCo
de dcErrorCode) {
        // 配网结果回调，每成功配网一个设备就回调一次。请您根据配网开始时发现的设备列表和返回的配
网成功的次数来判断是否配网结束。
    }
});
```

2.5. 摄像头设备配网开发实践

生活物联网平台为摄像头设备提供一种不依赖App和设备本地通信即可完成绑定的解决方法。摄像头设备连接路由器后，当发生本地局域网不通或路由器设置问题，导致App无法发现并绑定设备时，您可以将该解决办法作为摄像头设备的辅助配网方案。

配网开发流程

该解决方法的配网流程如下。

1. 手机App生成二维码图片。

   二维码中包含当前手机连接Wi-Fi的SSID、password、BSSID、random（3个字节的随机数）。

2. 摄像头设备扫描并解析二维码。

   设备端调用生活物联网平台SDK提供的 `awss_connect` 接口（详细介绍参见本文档下方“设备端接口与实现”），获取二维码中传递的SSID、password、BSSID、random，并按一定规则生成token。具体操作，请参见生成token的规则。

3. 设备端将生成的token上报云端。

4. App通过根据token查询设备的证书信息接口向云端发送轮询消息，当收到的返回消息为请求成功后，向设备发起绑定。

设备端接口与实现

设备端开发时，您需要实现通过调用 `awss_connect` 函数（头文件为#include "connect_ap.h"），来解析二维码信息和生成token的能力，并将生成的token上报到云端。

`awss_connect` 函数的原型如下。

```
awss_connect(char ssid[HAL_MAX_SSID_LEN], char passwd[HAL_MAX_PASSWD_LEN], uint8_t *bssid,
uint8_t bssid_len, uint8_t *token, uint8_t token_len, bind_token_type_t token_type)
```

函数的参数说明如下。

参数	类型	描述
ssid	char	路由器的SSID。
passwd	char	路由器的密码。
bssid	uint8_t *	路由器的BSSID，即MAC地址。
bssid_len	uint8_t	BSSID的长度。
token	uint8_t *	生活物联网平台SDK按一定规则生成的token。
token_len	uint8_t	生成的token的长度。
token_type	bind_token_type_t	生成的token的类型。 **说明** 该参数仅在生活物联网平台SDK 1.6.0及以上版本的该函数中存在，SDK 1.6.0以下版本请忽略。

以下为您提供 `awss_connect` 函数的使用示例。设定函数配置值如下。

```
char* ref_ssid = "ssid";
char* ref_passwd = "passwd";
uint8_t ref_bssid[6] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66};
uint8_t ref_token[3] = {0xaa, 0xbb, 0xcc};
```

则函数调用示例如下。

```
awss_connect(ref_ssid, ref_passwd, ref_bssid, 6, ref_token, 3);
```

生成token的规则

二维码中通常包含BSSID、password、random三个字段。生成token的规则为拼接二维码中字段，并取前16个字节。

根据二维码规范random字段必有，生成token的规则存在以下几种情形。

场景	拼接规则
二维码中有BSSID且有password	sha256(bssid(hex)+random(hex)+password(hex))
二维码中无BSSID有password	sha256(random(hex)+password(hex))
二维码中无BSSID且无password	sha256(random(hex))

以下为您提供token生成规则的示例。设定二维码中字段值如下。

```
BSSID: C8:3A:35:23:08:31
SSID：myname
password：12345678
RandomHex：911182
```

即该示例的二维码规范为：`{"v":"Ali_1","s":"myname","p":"12345678","b":"C83A35230831","t":"911182"}`。

则此时设备端调用信息如下。

```
char* ref_ssid = "myname";
char* ref_passwd = "12345678";
uint8_t ref_bssid[6] = {0xc8, 0x3a, 0x35, 0x23, 0x08, 0x31};
uint8_t ref_token[3] = {0x91, 0x11, 0x82};
awss_connect(ref_ssid, ref_passwd, ref_bssid, 6, ref_token, 3);
shar origin data = c83a352308319111823132333435363738,
sha = 8f2f5a6476bfabb52de0ea644b469eb5c70c2f4c6c05a51b8e70cdbe54ae8cb7
token = 8F2F5A6476BFABB52DE0EA644B469EB5
```

跨站点绑定摄像头设备

1. 获取App当前所在的站点。

   具体操作，请参见：

   - Android：获取App当前连接的服务器ID
   - iOS：获取App当前连接的服务器ID

2. 生成摄像头的配网二维码。

   二维码中需携带SSID、password、BSSID、random、regionid等信息。

3. 摄像头设备解析配网二维码。

   解析二维码的流程如下：

   i. 调用`iotx_guider_set_dynamic_region(info->region_id);`接口，让设备连上对应的站点。

   ii. 调用`awss_connect`接口，解析其他信息。

4. 绑定摄像头设备。

3.本地定时功能开发实践

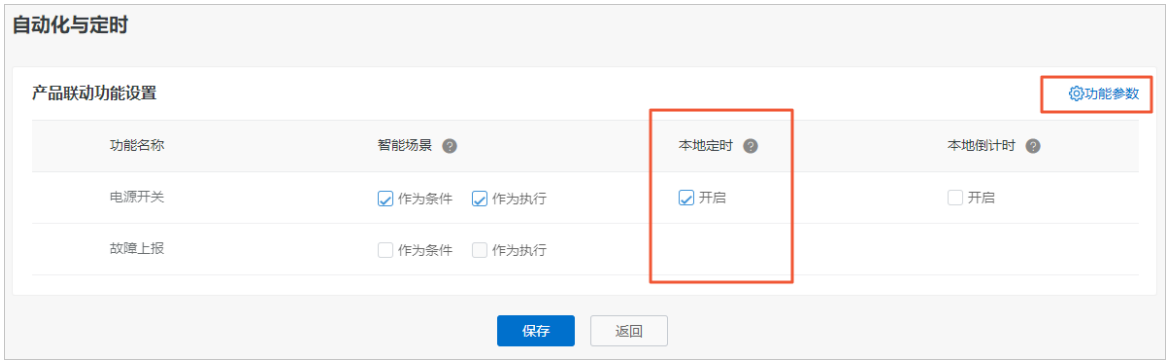
3.1. 开发设备端本地定时功能

本地定时相对云端定时而言，是指当设备离线时，也能自动执行定时任务。本文提供了一个插座本地定时功能的开发示例，您可以根据本文实现任意设备的本地定时功能。

一、配置控制台参数

1. 登录生活物联网控制台。
2. 创建一个项目。更多操作，请参见创建项目。
3. 创建产品，并定义本地定时（LocalTimer）产品功能。更多操作，请参见创建产品并定义功能。

4. 在产品的**人机交互 > 自动化和定时**页面，勾选本地定时的功能，并在功能参数中设置本地定时的最大条数（与设备端的存储、性能有关，默认为5）。



二、开发设备端的本地定时功能

1. 开发定时功能。

   在控制台上定义LocalTimer的功能属性后，设备端可以接收从云端下来的property set消息，从而获取定时任务的具体信息。详细开发步骤如下。

   i. 获取V1.3.0生活物联网平台SDK。更多操作，请参见获取SDK。

      - 智能插座实例代码，位于*Products/example/smart_outlet/smart_outlet_main.c*
      - 定时功能的开关配置代码，位于*Products/example/smart_outlet/smart_outlet.mk*

   ii. 基于设备端SDK开发定时功能。

      确认以下宏开关为打开状态。

      ```
      GLOBAL_CFLAGS += -DENABLE_LOCALTIMER  //本地定时的宏，默认为打开状态
      ```

   iii. 更改*build.sh*中的默认变量参数（如下所示，更多介绍请参见*README.md*）。

      ```
      产品类型：default_type="example"
      应用名称：default_app="smart_outlet"
      模组型号：default_board="uno-91h"  //根据实际型号配置
      连云区域：default_region=MAINLAND  //连接国外为SINGAPORE
      连云环境：default_env=ONLINE
      Debug log：default_debug=1   //0：release   1:debug
      其他参数：default_args=""     //可配置其他编译参数
      ```

   iv. 执行 `./build.sh` 编译生成固件。

      编译成功后，即可获得烧录所需的固件。

   v. 烧录固件。

      各模组的烧录方式略有差异，请向模组厂商获取详细烧录方法。

2. 调试设备。

   设备收到定时任务的属性时，在*user_property_set_event_handler*中查看日志。

```
static int user_property_set_event_handler(const int devid, const char *request, const
int request_len)
{
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    cJSON *root = NULL, *item = NULL;
    LOG_TRACE("Property Set Received, Devid: %d, Request: %s", devid, request);
    if ((root = cJSON_Parse(request)) == NULL) {
        LOG_TRACE("property set payload is not JSON format");
        return -1;
    }
    if ((item = cJSON_GetObjectItem(root, "PowerSwitch")) != NULL && cJSON_IsNumber(ite
m)) {
        if (item->valueint == 1) {
            product_set_switch(ON);
        } else {
            product_set_switch(OFF);
        }
    } else {
#ifdef AOS_TIMER_SERVICE
        timer_service_property_set(request);
#endif
    }
    cJSON_Delete(root);
    res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                             (unsigned char *)request, request_len);
    LOG_TRACE("Post Property Message ID: %d", res);
    return 0;
}
```

设备端接收到的示例如下。

```
{
  "LocalTimer": [
    {
      "PowerSwitch": 1,
      "Timer": "5 4 1,2,3",
      "TimezoneOffset": 43200,
      "Enable": 1,
      "Targets": "PowerSwitch",
      "IsValid": 1
    },
    {
      "PowerSwitch": 0,
      "Timer": "",
      "TimezoneOffset": 43200,
      "Enable": 0,
      "Targets": "",
      "IsValid": 0
    },
    {
      "PowerSwitch": 0,
      "Timer": "",
      "TimezoneOffset": 43200,
      "Enable": 0,
      "Targets": "",
      "IsValid": 0
    },
    {
      "PowerSwitch": 0,
      "Timer": "",
      "TimezoneOffset": 43200,
      "Enable": 0,
      "Targets": "",
      "IsValid": 0
    },
    {
      "PowerSwitch": 0,
      "Timer": "",
      "TimezoneOffset": 43200,
      "Enable": 0,
      "Targets": "",
      "IsValid": 0
    }
  ]
}
```


以上示例为JSON数组格式结构，LocalTimer中共有5条定时记录（在**人机交互 > 自动化和定时**页面的功能参数中设置的值）。每条数组中的每个JSON为一个定时任务，参数解释如下。

标识符	参数名称	数值类型	是否必选	参数描述
PowerSwitch	电源开关	布尔	是	产品**功能定义**页面的标准功能

标识符	参数名称	数值类型	是否必选	参数描述
Timer	定时时间	字符串	是	用于表示定时时间，使用cron格式（参见开发App端的本地定时功能中的通用说明）
Enable	启用	布尔	是	定义该条定时任务是否启用
IsValid	可执行	布尔	是	定义该条定时任务是否有效，设备端会将所有数据传给App端，App端根据该字段判定是否给用户展示该条定时任务
Targets	定时动作	字符串	否	表示当次设置的定时任务的具体动作，添加Targets属性可以设置任意数量的动作，最大值为：2048
TimezoneOffset	时差	整数	否	表示本地事件与UTC时间的差值 ○ 单位：秒 ○ 取值范围为：-43200到50400 ○ 步长：3600

三、开发App定时功能

生活物联网平台提供两种App的定时功能的开发方式。更多信息，请参见开发App端的本地定时功能。

3.2. 开发App端的本地定时功能

本地定时相对云端定时而言，是指当设备离线时，也能自动执行定时任务。本文档介绍自有App中的定时功能的开发实践。

前提条件

已完成开发设备端本地定时功能中的配置控制台参数和设备端开发。

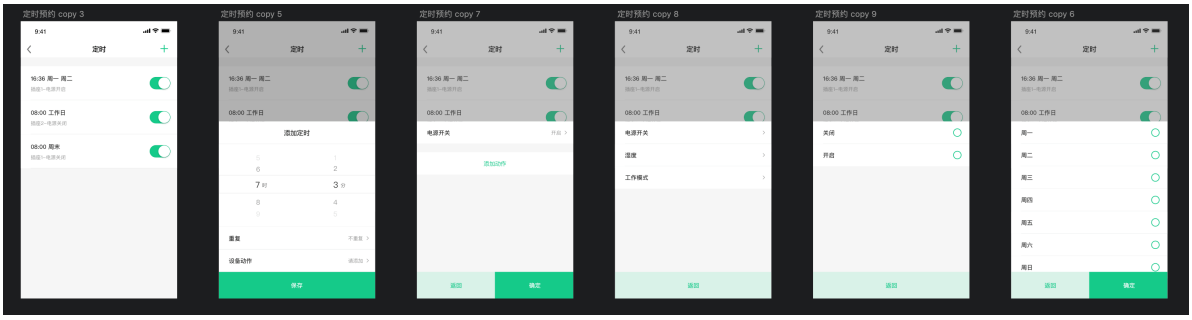
概述

生活物联网平台提供两种App定时功能的开发方式。

- 集成定时插件（动态方式）

  App调用本地定时插件实现本地定时任务的创建、编辑、删除等操作。该方式无需您开发定时业务逻辑，只需要启动本地定时插件即可，且支持动态更新。但由于定时插件在云端，该方式App的定时功能，在流畅度上的体验会稍差。

  定时插件集成到App的界面如下图所示。



- Native开发定时功能（静态方式）

  静态方式开发即开发native的本地定时功能。该方式开发的App定时功能用户体验更好。但对技术要求较高，且无法动态更新，需要通过发布App版本来更新。

集成定时插件

1. 创建一个自有App，并完成SDK下载。详细操作请参见创建自有App。
2. 在自有品牌App的**App界面**中，单击本地定时对应的**生成代码**。

3. 单击**复制**，将代码粘贴到您的App代码中，调用定时插件。

   示例代码片段如下。

   - iOS

     ```
     NSString *url = @"link://router/localTimer";
     [[IMSRouterService sharedService] openURL:url options:@{@"iotId":iotId}];
     ```

   - Android

```
Intent intent = new Intent();
intent.putExtra("iotId",iotId);
Router.getInstance().toUrl(context, "link://router/localTimer");
```

Native开发定时功能（iOS）

1. 创建一个自有App，并完成SDK下载。详细操作请参见创建自有App。
2. 获取本地定时的属性字段。

   本地定时是属于TSL的一个属性，定时就是操作TSL的LocalTimer。根据物模型SDK获取对应设备的TSL，并从中获取本地定时的个数、Targets、Timezoneoffset属性字段。

```
// 获取本地定时的个数，是否支持 Targets、Timezoneoffset
NSUInteger size = 0;
IMSThing *thing = [[IMSThingManager sharedManager] buildThing:iotId];
IMSThingProfile *profile = [thing getThingProfile];
NSArray<IMSThingTslProperty *> *proList = [profile allPropertiesOfModel];
__block BOOL hasTargets = NO;
__block BOOL hasTimezoneOffset = NO;
for (NSUInteger i = 0; i<proList.count; i++) {
    IMSThingTslProperty * _Nonnull obj = proList[i];
    NSLog(@"%lul", (unsigned long)i);
    if ([obj.identifier isEqualToString:@"LocalTimer"]) {
        NSString *sizeTmp = obj.dataType[@"specs"][@"size"];
        size = sizeTmp.integerValue;
        NSArray *itemList = obj.dataType[@"specs"][@"item"][@"specs"];
        [itemList enumerateObjectsUsingBlock:^(id  _Nonnull item, NSUInteger idx, BOOL
* _Nonnull stop) {
            if ([item[@"identifier"] isEqualToString:@"Targets"]) {
                hasTargets = YES;
            } else if ([item[@"identifier"] isEqualToString:@"TimezoneOffset"]){
                hasTimezoneOffset = YES;
            }
        }];
    }
}
```

3. 获取设备端的定时器列表数据。

```
// 本地定时model数据结构
@interface IMSLocalTimerModel : NSObject
@property (nonatomic, copy)   NSString *timer;
@property (nonatomic, assign) BOOL enable;
@property (nonatomic, assign) BOOL isValid;
@property (nonatomic, strong) NSMutableDictionary *propertyValueDic;
@property (nonatomic, strong) NSString *iotId;
@property (nonatomic, strong, nullable) NSString *targets;
@property (nonatomic, assign) NSInteger timezoneOffset;
@end
// 获取本地定时列表数据
id<IMSThingActions> thingObj = [thing getThingActions];
NSMutableArray<IMSLocalTimerModel *> *list = [NSMutableArray array];
[thingObj getPropertiesFull:^(IMSThingActionsResponse * _Nullable response) {
    // 回调不在主线程，请注意
```

```
    NSDictionary *dic = (NSDictionary *)response.dataObject;
    NSDictionary *data = dic[@"data"];
    for (NSString *key in data){
        if ([key isEqualToString:@"LocalTimer"]){
            NSDictionary *localTimer = data[key];
            NSArray *value = localTimer[@"value"];
            for (NSDictionary *item in value) {
                IMSLocalTimerModel *timer = [IMSLocalTimerModel new];
                timer.isValid = NO;
                timer.propertyValueDic = [NSMutableDictionary dictionary];
                for (NSString *a in item) {
                    if ([a isEqualToString:@"IsValid"]) {
                        NSNumber *isValid = item[a];
                        timer.isValid = isValid.boolValue;
                    } else if ([a isEqualToString:@"Enable"]) {
                        NSNumber *enable = item[a];
                        timer.enable = enable.boolValue;
                    } else if ([a isEqualToString:@"Timer"]) {
                        timer.timer = item[a];
                    } else if ([a isEqualToString:@"Targets"]){
                        timer.targets = item[a];
                    } else if ([a isEqualToString:@"TimezoneOffset"]){
                        NSNumber *timezoneOffset = item[a];
                        timer.timezoneOffset = timezoneOffset.integerValue;
                    } else {
                        timer.propertyValueDic[a] = item[a];
                    }
                }
                if (!timer.isValid) {
                    timer.propertyValueDic = [NSMutableDictionary dictionary];
                    timer.timer = @"";
                    timer.targets = @"";
                    timer.timezoneOffset = 0;
                }
                timer.hasTargets = hasTargets;
                timer.hasTimezoneOffset = hasTimezoneOffset;
                [list addObject:timer];
            }
            break;
        }
    }
    // 如果get到的属性值的timer个数没有达到tsl要求的定时器数，需要补齐，因为set的时候，tsl要求返回的个数和tsl要求的一样
    for (NSInteger i = list.count; i < size; i++) {
        IMSLocalTimerModel *timer = [IMSLocalTimerModel new];
        timer.isValid = NO;
        timer.timer = @"";
        timer.iotId = iotId;
        timer.hasTargets = hasTargets;
        timer.hasTimezoneOffset = hasTimezoneOffset;
        timer.propertyValueDic = [NSMutableDictionary dictionary];
        timer.modelList = list;
        [list addObject:timer];
    }
}
```

4. 设置（编辑/创建/删除等）一个定时器。

```
@implementation IMSLocalTimerModel
// 单个定时转换为json接口，供参考
- (NSMutableDictionary *)toJson{
    NSMutableDictionary *json = [NSMutableDictionary dictionary];
    [json addEntriesFromDictionary:self.propertyValueDic];
    [json addEntriesFromDictionary:@{@"Enable":@(self.enable?1:0),@"IsValid":@(self.isV
alid?1:0), @"Timer":self.timer?:@""}];
    if (self.hasTargets) {
    [json addEntriesFromDictionary:@{@"Targets":self.targets?:@""}];
    }
    if (self.hasTimezoneOffset) {
    [json addEntriesFromDictionary:@{@"TimezoneOffset":@(self.timezoneOffset)}];
    }
    return json;
}
/* 如果 TimezoneOffset 字段存在，这个字段是为了让设备端能获取到app端设置定时时所处时区*/
propertyValues[@"TimezoneOffset"] = @([NSTimeZone localTimeZone].secondsFromGMT);
/*
    假设当前设备有3个属性可以通过本地定时进行控制，这3个属性的identifier分别为id1、id2、id3
    如果本地定时存在Targets字段，则允许1个定时只控制3个属性中部分属性，假设这个定时是控制属性id1
和属性id2，
    则 Targets = @"id1, id2"
    如果本地定时不存在Targets字段，则要求改定时创建或者编辑的时候，必须对3个属性都设置属性值（定时
时间到的是属性需要生效的值）
*/
// list 是属性的 identifier 的列表
- (NSString *)setTargetList:(NSArray<NSString *> *)list{
    if (list.count == 0) {
        return @"";
    }
    NSString *targets = list.firstObject;
    for (NSUInteger i = 1; i < list.count; i++) {
        targets = [targets stringByAppendingString:@","];
        targets = [targets stringByAppendingString:list[i]];
    }
    return targets;
}
propertyValues[@"TimezoneOffset"] = [obj setTargetList:@[id1, id2]];
/* 假设tsl中获取到设备可以设置3个本地定时，则timerList=@[[timer1 toJson], [timer2 toJson], [
timer3 toJson]] :
    每个定时器的json生成参考 [IMSLocalTimerModel toJson]
[
1. 同时控制属性1和属性2的定时
{"id1":1,"id2":2,"id3":1,"Timer":"0 8 * * *","Enable":1,"IsValid":1, "Targets":"id1, id
2", "TimezoneOffset":""},
2. 只控制属性1的定时
 {"id1":1,"id2":2,"id3":1,"Timer":"2 6 * * 3","Enable":1,"IsValid":1, "Targets":"id1",
"TimezoneOffset":""},
3. 设置一个定时的IsValid为0，则就是删除该定时器，但是这里建议把IsValid 也设置为0，因为设备端的实
现可能只认IsValid字段
 {"id1":1,"id2":2,"id3":1,"Timer":"2 6 * * 3","Enable":0,"IsValid":0, "Targets":"", "Ti
```

```
mezoneOffset":""}
]
*/
[thingActions setProperties:@{@"LocalTimer":timerList} responseHandler:^(IMSThingAction
sResponse * _Nullable response) {
}
```

5. 订阅本地定时属性值。

   以监听定时的Enable状态为例，设定定时的有效性是一次，当定时超时后，App端会收到定时的Enable属性变为0的通知，并进行界面刷新等相关操作。相应示例代码如下。

```
// 代码供参考
a.注册订阅
IMSThing *thing = [[IMSThingManager sharedManager] buildThing:controller.iotId];
[thing registerThingObserver:(id<IMSThingObserver>)obj];
b.订阅通知处理
- (void)onPropertyChange:(NSString *)iotId params:(NSDictionary *)params{
    IMSDeviceLogDebug(@"onPropertyChange %@ %@", iotId, params);
    NSArray *list = params[@"items"][@"LocalTimer"][@"value"];
    for (NSInteger i = 0; i < list.count; i++) {
        NSDictionary *timer = list[i];
        if (i >= self.list.count) {
            [self.list addObject:[IMSLocalTimerModel new]];
        }
        IMSLocalTimerModel *model = self.list[i];
        model.timezoneOffset = NO;
        model.hasTargets = NO;
        model.propertyValueDic = [NSMutableDictionary dictionary];
        [timer enumerateKeysAndObjectsUsingBlock:^(NSString *_Nonnull key, NSNumber*  _
Nonnull obj, BOOL * _Nonnull stop) {
            if ([key isEqualToString:@"Enable"]) {
                model.enable = obj.boolValue;
            } else if ([key isEqualToString:@"IsValid"]){
                model.isValid = obj.boolValue;
            } else if ([key isEqualToString:@"Targets"]){
                model.targets = (NSString *)obj;
                model.hasTargets = YES;
            } else if ([key isEqualToString:@"Timer"]){
                model.timer = (NSString *)obj;
            } else if ([key isEqualToString:@"TimezoneOffset"]){
                model.timezoneOffset = obj.integerValue;
                model.hasTimezoneOffset = YES;
            } else {
                model.propertyValueDic[key] = obj;
            }
        }];
    }
    // 刷新界面等
}
c.注销订阅
IMSThing *thing = [[IMSThingManager sharedManager] buildThing:self.iotId];
[thing unregisterThingObserver:self];
```

Native开发定时功能（Android）

1. 创建一个自有App，并完成SDK下载。详细操作请参见创建自有App。
2. 获取设备TSL模型和定时属性。
   i. 创建 `com.aliyun.alink.linksdk.tmp.device.panel.PanelDevice` 对象。

```
PanelDevice panelDevice = new PanelDevice(iotId);
panelDevice.init(null, new IPanelCallback() {
  @Override
  public void onComplete(boolean succeed, Object json) {
      // 根据自己的业务逻辑实现相关代码
  }
})
```

   ii. 获取TSL模型。

   本地定时是属于TSL的一个属性，更多TSL模型的介绍请参见物模型SDK。

```
panelDevice.getTslByCache(new IPanelCallback() {
    @Override
    public void onComplete(boolean succeed, Object json) {
  // 根据自己的业务逻辑实现相关代码
    }
});
```

   iii. 获取设备属性。

```
panelDevice.getPropertiesByCache(new IPanelCallback() {
  @Override
  public void onComplete(boolean succeed, @Nullable Object json) {
     // 根据自己的业务逻辑实现相关代码
    }
}, null);
```

   返回数据中的JSON格式示例如下。

```
{
    "items":{
    "LocalTimer":  // 以下为定时数据的示例
[{
            "LightSwitch": 1,
            "ColorTemperature": 2000,
            "Timer": "5 4 1,2,3",
            "TimezoneOffset": 43200,
            "Brightness": 0,
            "Enable": 1,
            "Targets": "LightSwitch",
            "WorkMode": 0,
            "IsValid": 1
}]
            }
}
```

3. 解析定时数据。

根据TSL模型解析定时数据的方法如下，更详细的数据结构请参见物模型SDK。

```
   static private void parseLocalTimer(JSONObject dataObj, LocalTimerData localTimerDat
a) {
        for (String key : dataObj.keySet()) {
            if (key.equals("LocalTimer")) {
                JSONArray value = dataObj.getJSONObject(key).getJSONArray("value");
                for (int i = 0; i < value.size(); i++) {
                    JSONObject valueItem = value.getJSONObject(i);
                    LocalTimer localTimer = new LocalTimer();
                    for (String valueItemkey : valueItem.keySet()) {
                        if (valueItemkey.equals("Timer")) {
                            localTimer.timer = (String) valueItem.get(valueItemkey);
                        } else if (valueItemkey.equals("Enable")) {
                            localTimer.enable = (int) valueItem.get(valueItemkey);
                        } else if (valueItemkey.equals("IsValid")) {
                            localTimer.valid = (int) valueItem.get(valueItemkey);
                        } else if (valueItemkey.equals("Targets")) {
                            localTimer.targets = (String) valueItem.get(valueItemkey);
                        } else if (valueItemkey.equals("TimezoneOffset")) {
                            localTimer.timezoneOffset = (int) valueItem.get(valueItemke
y);
                        } else {
                            localTimer.property.put(valueItemkey, valueItem.get(valueIt
emkey));
                        }
                    }
                    localTimerData.items.add(localTimer);                }
            }
     }
```

4. 设置一个定时器。
   - 单设备

     使用 `PanelDevice#setProperties()` 方法更新设备属性。

     ```
     panelDevice.setProperties(json, new IPanelCallback() {
                    @Override
                    public void onComplete(final boolean succeed, final Object json) {
                  }
                });
     ```

     入参JSON格式示例如下。

```
{
    "iotId":"",
    "items":{
        "LocalTimer":[  // 以下为定时数据的示例
            {
                "LightSwitch":1,
                "ColorTemperature":2000,
                "Timer":"5 4 1,2,3",
                "TimezoneOffset":43200,
                "Brightness":0,
                "Enable":1,
                "Targets":"LightSwitch",
                "WorkMode":0,
                "IsValid":1
            }
        ]
    }
}
```

- 组控设备

  使用 `PanelGroup#setGroupProperties()` 方法更新设备属性。

  PanelGroup的初始化方法与panelDevice的初始化类似，区别是要传入组控的groupId。

```
panelGroup.setGroupProperties(json, new IPanelCallback() {
               @Override
               public void onComplete(final boolean succeed, final Object json) {
             }
});
```

  入参JSON格式示例如下。

```
{
    "controlGroupId":"",
    "items":{
        "LocalTimer":[   // 以下为定时数据的示例
            {
                "LightSwitch":1,
                "ColorTemperature":2000,
                "Timer":"5 4 1,2,3",
                "TimezoneOffset":43200,
                "Brightness":0,
                "Enable":1,
                "Targets":"LightSwitch",
                "WorkMode":0,
                "IsValid":1
            }
        ]
    }
}
```

5. 订阅本地定时属性值。

   实时更新常用来处理多端同时设置设备属性的情况。

```
// 接口：/app/down/thing/properties 设备端上报属性
panelDevice.subAllEvents(new IPanelEventCallback() {
                    @Override
                    public void onNotify(String id, String path, Object json) {
                        if (!id.equals(iotId)) {
                            return;
                        }
                        if (!"/app/down/thing/properties".equals(path)) {
                            return;
                        }
                    }
                }, new IPanelCallback() {
                    @Override
                    public void onComplete(boolean b, Object o) {
                    }
                });
```

返回数据中的JSON格式示例如下。

```
{
    "items":{
    "LocalTimer":  // 以下为定时数据的示例
[{
            "LightSwitch": 1,
            "ColorTemperature": 2000,
            "Timer": "5 4 1,2,3",
            "TimezoneOffset": 43200,
            "Brightness": 0,
            "Enable": 1,
            "Targets": "LightSwitch",
            "WorkMode": 0,
            "IsValid": 1
}]
            }
}
```

通用说明

- corn 表达式说明（即LocalTimer结构中的Timer字段 ）

  定时属性中的 CronTrigger 配置完整格式为： `[分] [小时] [日] [月] [周]`

  - `*` 表示所有值。在分钟里表示每一分钟触发。如在小时、日期、月份里面表示每一小时、每一日、每一月。
  - `-` 表示区间。小时设置为10-12表示10、11、12点均会触发。
  - `,` 表示多个值。 周设置成 2、3、4、5、6 表示在周一至周五工作日会触发。
  - `/` 表示递增触发。 5/15表示从第5秒开始，每隔15秒触发。
  - `L` 表示最后的意思。 日上表示最后一天。星期上表示星期六或7。 L前加数据，表示该数据的最后一个。 星期上设置6L表示最后一个星期五（6表示星期五）。
  - `W` 表示离指定日期最近的工作日触发。15W离该月15号最近的工作日触发。表示每月的第几个周几（6#3表示该月的第三个周五）。

- 时区说明（即LocalTimer结构中的TimeOffset字段 ）

  由于Android中自带的Calendar对于 `Daylight Saving Time (DST)` 的处理有问题。在需要处理冬令时和夏令时的地区，请使用Java 8提供的Instant类或者其他方法来计算时差。代码示例如下。

```
 private int timezoneOffset() {
        try {
            Instant instant = Instant.now();
            Calendar calendar = new GregorianCalendar();
            TimeZone timezone = calendar.getTimeZone();
            ZoneId zone = ZoneId.of(timezone.getID());
            ZonedDateTime z = instant.atZone(zone);
            int offset = z.getOffset().getTotalSeconds();
            ALog.d(TAG, "timezoneOffset(): ZoneId:" + timezone.getID() + ", getTotalSecon
ds: " + offset);
            return offset;
        } catch (Exception ignored) {
            return 0;
        }
    }
```

- Targets字段说明

  如果在LocalTimer里添加了多个动作， 则必须在Target字段里面添加您本次修改的字段。否则，您必须完整设置所有的动作，本地定时才能正常保存。

```
{
    "LightSwitch":0,
    "Timer":"45 12 * * *",
    "Enable":0,
    "Targets":"LightSwitch",
    "IsValid":1
}
```

3.3. 开发天猫精灵项目Wi-Fi产品本地定时功能

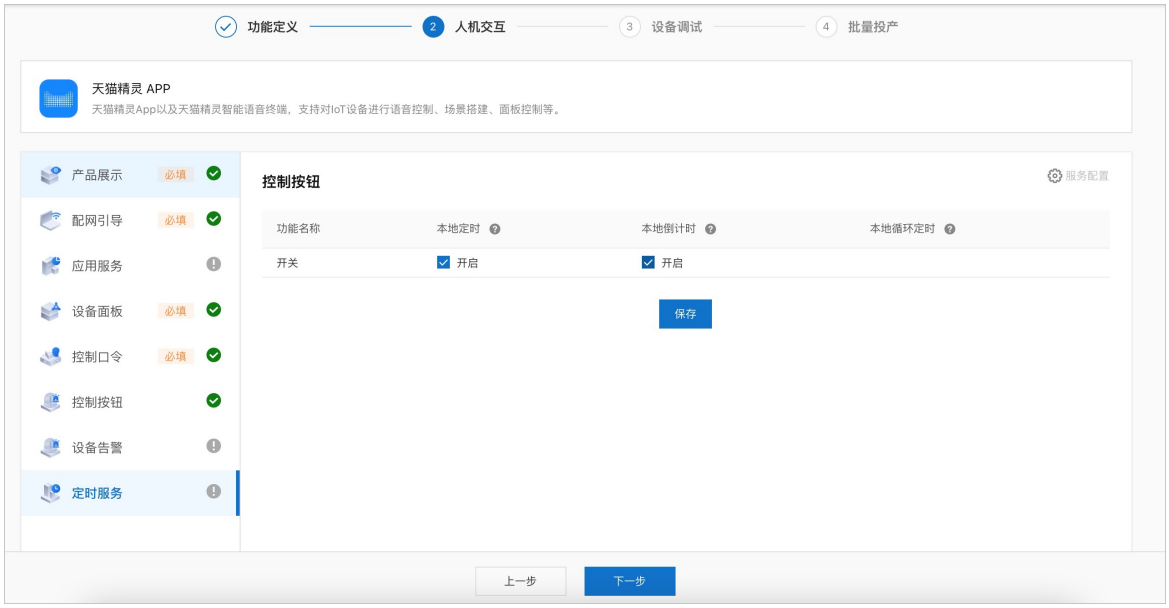
天猫精灵项目新增了DeviceTimer属性，整合了本地定时、循环定时、倒计时等定时相关的功能。本文提供了一个插座本地定时功能的开发示例，作为基于DeviceTimer属性开发定时功能的参考示例。

配置控制台参数

1. 登录生活物联网控制台。
2. 创建一个项目。更多操作，请参见创建项目。
3. 创建产品，并定义设备端上定时（DeviceTimer）产品功能。更多操作，请参见创建产品并定义产品功能，注意选择联网方式为Wi-Fi。


新建产品
产品信息
* 产品名称
智能插座
* 所属品类
电工照明 / 插座
功能定义
节点类型
* 节点类型
设备
网关
* 是否接入网关
是
否
连网与数据
* 连网方式
WiFi
确认
取消

4. 在产品的**人机交互 > 定时服务**页面，勾选本地定时与本地倒计时的功能，并在服务配置中设置本地定时的最大条数（与设备端的存储、性能有关，默认为13）。



5. 在产品的**人机交互 > 设备面板**页面，选择或者配置产品的面板，可以选择宜控面板，或者自己编辑面板。

如果选择编辑面板，注意要选上**预约**组件。

开发设备端的本地定时功能

1. 开发定时功能。

   在控制台上定义DeviceTimer的功能属性后，设备端可以接收从云端下来的property set消息，从而获取定时任务的具体信息。详细开发步骤如下。

   i. 必须下载生活物联网平台SDKV1.6.6-5以上的版本。更多操作，请参见获取SDK。

   - 智能插座示例代码，位于*Products/example/smart_outlet/smart_outlet_main.c*
   - 定时功能的配置代码，位于*Products/example/smart_outlet/smart_outlet.mk*

ii. 基于设备端SDK开发定时功能。

确认*Products/example/smart_outlet/smart_outlet.mk*文件中以下宏开关的打开与关闭状态。

```
GLOBAL_CFLAGS += -DAIOT_DEVICE_TIMER_ENABLE     //新版设备端DeviceTimer支持的宏开关，默认为打开状态
# GLOBAL_CFLAGS += -DAOS_TIMER_SERVICE          //老版本定时服务的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_COUNTDOWN_LIST      //老版本本地倒计时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_LOCALTIMER          //老版本本地定时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_PERIOD_TIMER        //老版本周期定时的宏，默认为关闭状态
# GLOBAL_CFLAGS += -DENABLE_RANDOM_TIMER        //老版本随机定时的宏，默认为关闭状态
```

iii. 查看确认*build.sh*中的默认变量参数，是否符合项目实际需求（如下所示，更多介绍请参见*README.md*）。

```
产品类型：default_type="example"
应用名称：default_app="smart_outlet"
模组型号：default_board="tg7100cevb"  //根据实际型号配置
连云区域：default_region=MAINLAND
连云环境：default_env=ONLINE
Debug log：default_debug=1            //0：release   1:debug
其他参数：default_args=""             //可配置其他编译参数
```

iv. 执行 `./build.sh` 编译生成固件。

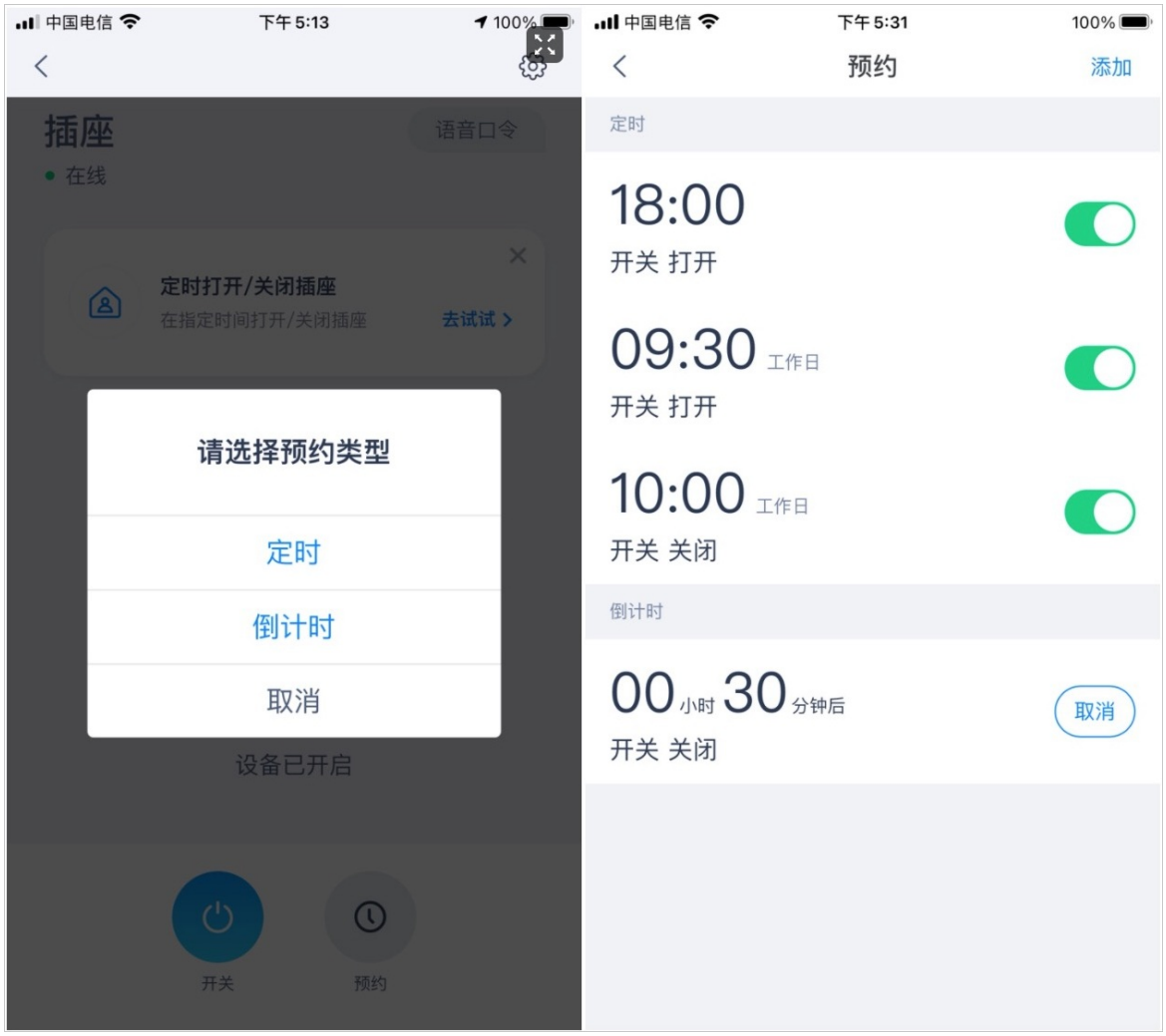
编译成功后，即可获得烧录所需的固件。

v. 烧录固件。

各模组的烧录方式略有差异，请向模组厂商获取详细烧录方法。

2. 调试设备。

用天猫精灵App或者天猫精灵音箱找队友添加设备后，通过面板预约定时。



设备收到定时任务的属性时，在*user_property_set_event_handler*中查看日志。

```
static int user_property_set_event_handler(const int devid, const char *request, const
int request_len)
{
    int ret = 0;
    recv_msg_t msg;
#ifdef CERTIFICATION_TEST_MODE
    return ct_main_property_set_event_handler(devid, request, request_len);
#endif
    LOG_TRACE("property set,  Devid: %d, payload: \"%s\"", devid, request);
    msg.from = FROM_PROPERTY_SET;
    strcpy(msg.seq, SPEC_SEQ);
    property_setting_handle(request, request_len, &msg);
    return ret;
}
static int property_setting_handle(const char *request, const int request_len, recv_msg
_t * msg)
{
    cJSON *root = NULL, *item = NULL;
    int ret = -1;
    if ((root = cJSON_Parse(request)) == NULL) {
        LOG_TRACE("property set payload is not JSON format");
```

```
        return -1;
    }
    ...
#ifdef AIOT_DEVICE_TIMER_ENABLE
    else if ((item = cJSON_GetObjectItem(root, DEVICETIMER)) != NULL && cJSON_IsArray(i
tem))
    {
        // Before LocalTimer Handle, Free Memory
        cJSON_Delete(root);
        ret = deviceTimerParse(request, 0, 1);
        user_example_ctx_t *user_example_ctx = user_example_get_ctx();
        if (ret == 0) {
            IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                (unsigned char *)request, request_len);
        } else {
            char *report_fail = "{\"DeviceTimer\":[]}";
            IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                    (unsigned char *)report_fail, strlen(report_fail));
            ret = -1;
        }
        // char *property = device_timer_post(1);
        // if (property != NULL)
        //     HAL_Free(property);
        return 0;
    }
    #ifdef MULTI_ELEMENT_TEST
    else if (propertys_handle(root) >= 0) {
        user_example_ctx_t *user_example_ctx = user_example_get_ctx();
        cJSON_Delete(root);
        IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                (unsigned char *)request, request_len);
        return 0;
    }
    #endif
#endif
    ...
 }
```

设备端接收到的示例如下。

```
"{"DeviceTimer":[
    {"A":"powerstate:0","R":0,"S":0,"T":"01 18 22 05 ? 2021","E":1,"Y":1,"Z":28800,"N":
""},
    {"A":"powerstate:1","R":0,"S":0,"T":"00 18 22 05 ? 2021","E":1,"Y":2,"Z":28800,"N":
""},
    {"A":"powerstate:1","R":0,"S":0,"T":"30 09 ? * 1,2,3,4,5 *","E":1,"Y":2,"Z":28800,"
N":""},
    {"A":"powerstate:0","R":0,"S":0,"T":"00 10 ? * 1,2,3,4,5 *","E":1,"Y":2,"Z":28800,"
N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""},
    {"A":"","R":0,"S":0,"T":"","E":0,"Y":0,"Z":28800,"N":""}
]
}"
```

以上示例为JSON数组格式结构，DeviceTimer中共有13条定时记录（在**人机交互 > 定时服务**页面的服务配置中设置的值）。每条数组中的每个JSON为一个定时任务，参数解释如下。

缩写	全名	字段名称	数值类型	参数描述
A	Targets	定时动作	字符串	表示当次设置的定时任务的具体动作，如字符串里包含"|"，则"|"前面的是RunTime需执行的action，"|"后面的是SleepTime需执行的action
R	RunTime	运行时间	整数	单位：秒
S	SleepTime	睡眠时间	整数	单位：秒
T	Timer	开始时间	字符串	用于表示定时任务开始时间，使用cron格式
E	Enable	启用	布尔	定义该条定时任务是否启用
Y	Type	定时类型	整数	定时类型 ◦ 0：未配置 ◦ 1：倒计时 ◦ 2：本地定时 ◦ 3：循环定时
Z	TimeZone	时区	整数	表示本地事件与UTC时间的差值 ◦ 单位：秒 ◦ 取值范围为：-43200到50400 ◦ 步长：3600
N	EndTime	结束时间	字符串	参考格式为"18:30"

cron格式定义与示例如下：

```
cron格式：分 时 日 月 周 年
周重复：   22 10 * * 1,2,3,4,5,6,7 *      // 每周一到周日，10点22分
指定日期：22 10 28 12 * 2021              // 2021年12月28日10点22分
单次定时：22 10 * * * *                   // 10点22分
```

所以上面示例中，默认13条定时任务，下发了4条启用的任务。

```
{ //类型为倒计时，2021年5月22日18时01分，执行开关关闭（powerstate属性值设为0）
  "A":"powerstate:0",
  "R":0,
  "S":0,
  "T":"01 18 22 05 ? 2021",
  "E":1,        //本任务启用
  "Y":1,        //类型为倒计时
  "Z":28800,    //东八区
  "N":""
},
{ //类型为本地计时，2021年5月22日18时00分单次执行，执行开关打开（powerstate属性值设为1）
  "A":"powerstate:1",
  "R":0,
  "S":0,
  "T":"00 18 22 05 ? 2021",
  "E":1,        //本任务启用
  "Y":2,        //类型为本地定时
  "Z":28800,    //东八区
  "N":""
},
{ //类型为本地计时，每周一到周五9：30分执行，执行开关打开（powerstate属性值设为1）
  "A":"powerstate:1",
  "R":0,
  "S":0,
  "T":"30 09 ? * 1,2,3,4,5 *",
  "E":1,        //本任务启用
  "Y":2,        //类型为本地定时
  "Z":28800,    //东八区
  "N":""
},
{ //类型为本地计时，每周一到周五10：00分执行，执行开关关闭（powerstate属性值设为0）。
  "A":"powerstate:0",
  "R":0,
  "S":0,
  "T":"00 10 ? * 1,2,3,4,5 *",
  "E":1,        //本任务启用
  "Y":2,        //类型为本地定时
  "Z":28800,    //东八区
  "N":""
},
{ //未设置的任务
  "A":"",
  "R":0,
  "S":0,
  "T":"",
  "E":0,        //未启用
  "Y":0,        //类型为未配置
  "Z":28800,    //东八区
  "N":""
},
```

> **说明** 对于没有RTC的设备，会有两个问题需要注意。第一，配置定时后，如果长时间离线，时钟偏差会逐渐变大；第二，配置定时后，发生设备重启，如果设备未成功联网并更新UTC时间，定时功能将无法工作。

开发多孔插座的定时功能

如果要基于智能插座示例开发多孔插座，设备端在开发定时功能时需要注意以下事项。

- 将单孔插座中默认关闭的宏MULTI_ELEMENT_TEST打开，可以使能多element功能。
- 通过宏NUM_OF_TIMER_PROPERTYS定义定时控制的element数量。
- 把各element对应的物模型字段名，填入数组propertys_list[NUM_OF_TIMER_PROPERTYS]，并在数组propertys_type[NUM_OF_TIMER_PROPERTYS]填写对应属性的数据类型（布尔型，枚举型，整型统一填T_INT，浮点型填T_FLOAT）
- 示例如下：

```
#ifdef AIOT_DEVICE_TIMER_ENABLE
    #define MULTI_ELEMENT_TEST      //此处使能多element功能
    #ifndef MULTI_ELEMENT_TEST
        #define NUM_OF_TIMER_PROPERTYS 3 /*  */
        const char *propertys_list[NUM_OF_TIMER_PROPERTYS] = { "PowerSwitch", "powerstate
", "allPowerstate" };
    #else // only for test
        #define NUM_OF_TIMER_PROPERTYS 14 /*  */
        // const char *propertys_list[NUM_OF_TIMER_PROPERTYS] = { "testEnum", "testFloat"
, "testInt", "powerstate", "allPowerstate" };
        const char *propertys_list[NUM_OF_TIMER_PROPERTYS] = { "powerstate", "allPowersta
te", "mode", "powerstate_1", "brightness", "windspeed", "powerstate_2",
                    "powerstate_3", "heaterPower", "windspeed", "angleLR", "testEnum", "t
estFloat", "testInt" };
        typedef enum {
            T_INT = 1,
            T_FLOAT,
            T_STRING,
            T_STRUCT,
            T_ARRAY,
        }  data_type_t;
        const data_type_t propertys_type[NUM_OF_TIMER_PROPERTYS] = { T_INT,T_INT,T_INT,T_
INT,T_INT,T_INT,T_INT,T_INT,T_FLOAT,T_INT,T_INT,T_INT,T_FLOAT,T_INT };
        static int propertys_handle(cJSON *root) {
            cJSON *item = NULL;
            int ret = -1, i = 0;
            for (i = 0; i < NUM_OF_TIMER_PROPERTYS; i++) {
                if (propertys_type[i] == T_STRUCT && (item = cJSON_GetObjectItem(root, pr
opertys_list[i])) != NULL && cJSON_IsObject(item)) { //structs
                    printf(" %s\r\n", propertys_list[i]);
                    ret = 0;
                } else if (propertys_type[i] == T_FLOAT && (item = cJSON_GetObjectItem(ro
ot, propertys_list[i])) != NULL && cJSON_IsNumber(item)){ // float
                    printf(" %s %f\r\n", propertys_list[i], item->valuedouble);
                    ret = 0;
                } else if (propertys_type[i] == T_INT &&(item = cJSON_GetObjectItem(root,
propertys_list[i])) != NULL && cJSON_IsNumber(item)){ // int
                    printf(" %s %d\r\n", propertys_list[i], item->valueint);
                    ret = 0;
                } else if (propertys_type[i] == T_STRING &&(item = cJSON_GetObjectItem(ro
ot, propertys_list[i])) != NULL && cJSON_IsString(item)){ // string
                    printf(" %s %s\r\n", propertys_list[i], item->valuestring);
                    ret = 0;
                } else if (propertys_type[i] == T_ARRAY &&(item = cJSON_GetObjectItem(roo
t, propertys_list[i])) != NULL && cJSON_IsArray(item)){ // array
                    printf(" %s \r\n", propertys_list[i]);
                    ret = 0;
                }
            }
            return ret;
        }
    #endif
```

各element属性设置处理入口示例如下。

```
static int propertys_handle(cJSON *root) {
    cJSON *item = NULL;
    int ret = -1, i = 0;
    for (i = 0; i < NUM_OF_TIMER_PROPERTYS; i++) {
        if (propertys_type[i] == T_STRUCT && (item = cJSON_GetObjectItem(root, propertys_li
st[i])) != NULL && cJSON_IsObject(item)) { //structs
            printf(" %s\r\n", propertys_list[i]);
            ret = 0;
        } else if (propertys_type[i] == T_FLOAT && (item = cJSON_GetObjectItem(root, proper
tys_list[i])) != NULL && cJSON_IsNumber(item)){ // float
            printf(" %s %f\r\n", propertys_list[i], item->valuedouble);
            ret = 0;
        } else if (propertys_type[i] == T_INT &&(item = cJSON_GetObjectItem(root, propertys
_list[i])) != NULL && cJSON_IsNumber(item)){ // int
            printf(" %s %d\r\n", propertys_list[i], item->valueint);
            ret = 0;
        } else if (propertys_type[i] == T_STRING &&(item = cJSON_GetObjectItem(root, proper
tys_list[i])) != NULL && cJSON_IsString(item)){ // string
            printf(" %s %s\r\n", propertys_list[i], item->valuestring);
            ret = 0;
        } else if (propertys_type[i] == T_ARRAY &&(item = cJSON_GetObjectItem(root, propert
ys_list[i])) != NULL && cJSON_IsArray(item)){ // array
            printf(" %s \r\n", propertys_list[i]);
            ret = 0;
        }
    }
    return ret;
}
```

定时执行，完成相关操作，会执行回调函数，参考 `timer_service_cb` 函数回调实现。

```
static void timer_service_cb(const char *report_data, const char *property_name, const char
*data)
{
    uint8_t value = 0;
    char property_payload[128] = {0};
    // if (report_data != NULL) /* post property to cloud */
    //     user_post_property_json(report_data);
    if (property_name != NULL) {  /* set value to device */
        LOG_TRACE("timer event callback=%s val=%s", property_name, data);
        #ifdef MULTI_ELEMENT_TEST
        user_example_ctx_t *user_example_ctx = user_example_get_ctx();
        if (strcmp(propertys_list[0], property_name) != 0 && strcmp(propertys_list[1], prop
erty_name) != 0) {
            snprintf(property_payload, sizeof(property_payload), "{\"%s\":%s}", property_na
me, data);
            IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERTY,
                    property_payload, strlen(property_payload));
            return;
        }
        else
        #endif
        {
            // data is int; convert it.
            value = (uint8_t)atoi(data);
        }
        recv_msg_t msg;
        msg.powerswitch = value;
        msg.all_powerstate = value;
        msg.flag = 0x00;
        strcpy(msg.seq, SPEC_SEQ);
        send_msg_to_queue(&msg);
    }
    return;
}
```

4.UI定制开发实践

4.1. 定制Android App的OA UI

前提条件

已完成账号及用户SDK的开发，详细请参见账号及用户SDK。

定制项

Android App的OA UI定制项如下。

App界面图示	可定制内容
	修改原生元素 • 显示手机区号（图示中②） • 修改登录和注册按钮的颜色（图示中③） • 修改现有控件的事件 • 修改登录超限提示信息样式（图示中④） 新增元素 • 新增控件和单击事件 • 新增邮箱登录方式（图示中①）

说明 Android不支持客户端修改失败提示信息，且不支持对现有控件增加自定义事件。您仅可以修改现有控件的事件或新增控件来自定义事件。

显示手机区号

登录和注册页面中的手机区号默认不显示，如果您需要显示手机区号，如+86，请根据以下内容来操作。

在App工程中添加以下代码，并通过设置supportForeignMobileNumbers参数的值来控制手机区号的显示和隐藏。

```
//登录页显示手机区号
OpenAccountUIConfigs.AccountPasswordLoginFlow.supportForeignMobileNumbers=true; //true为显示，false为隐藏
//忘记密码页显示手机区号
OpenAccountUIConfigs.MobileResetPasswordLoginFlow.supportForeignMobileNumbers=true; //true为显示，false为隐藏
```

修改登录和注册按钮的颜色

登录和注册按钮默认为浅灰色，如果您需要修改App登录和注册按钮的颜色，请根据以下步骤来操作。

1. 在代码工程的*styles.xml*中新增一个style。
2. 修改style中登录和注册按钮的颜色，其对应的参数为ali_sdk_openaccount_attrs_next_step_bg。

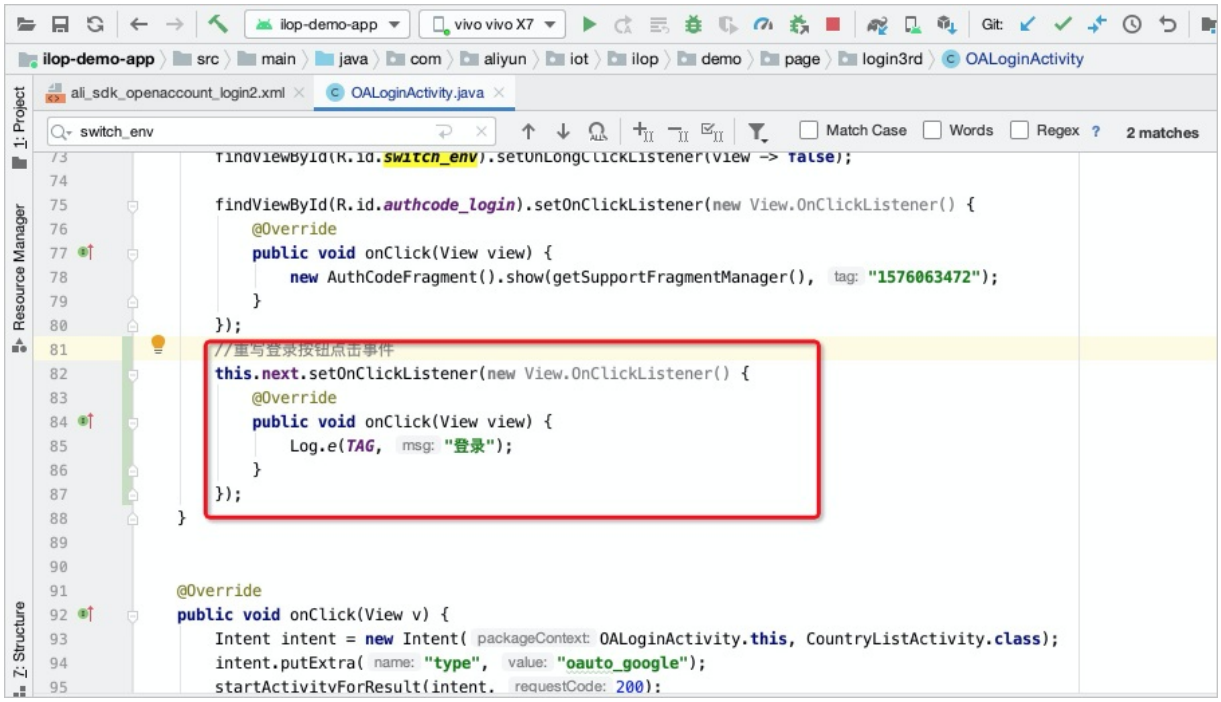
```
<style name="NewLogin" parent="@style/Login">
        //修改item属性的值
        <item name="ali_sdk_openaccount_attrs_next_step_bg">@color/color_1FC88B</item>
</style>
```

3. 修改AndroidManifest中的activity的主题配置，使设置的颜色生效。

```
//AndroidManifest中的activity的主题配置修改
<activity
            android:name="com.aliyun.iot.ilop.demo.page.login3rd.OALoginActivity"
            android:configChanges="orientation|screenSize|keyboardHidden|locale|layoutDirection|keyboard"
            android:theme="@style/NewLogin" />
```

修改现有控件的事件

如果您需要修改现有控件的事件，只需重新设置控件事件即可。以重写登录界面中的登录事件为例，即重新定义下图所示内容。



修改登录超限提示信息样式

登录App时，如果输入登录密码的次数达到上限，App会限制继续操作，并提示重置密码。如果您需要修改该失败提示对话框的样式，只需修改 `toastUtils.alert()` 即可。

1. 重写 `toastUtils.alert()` 的代码。

```
//OALoginActivity中的LoginTask
protected class LoginTask extends TaskWithDialog<Void, Void, Result<LoginResult>> {
        private String loginId;
        private String password;
        private String sig;
        private String nocToken;
        private String cSessionId;
        public LoginTask(Activity activity, String loginId, String password, String sig
, String nocToken, String cSessionId) {
            super(activity);
            this.loginId = loginId;
            this.password = password;
            this.sig = sig;
            this.nocToken = nocToken;
            this.cSessionId = cSessionId;
        }
        protected Result<LoginResult> asyncExecute(Void... params) {
            Map<String, Object> loginRequest = new HashMap();
            if (this.loginId != null) {
                loginRequest.put("loginId", this.loginId);
            }
            if (this.password != null) {
                try {
                    String rsaKey = RSAKey.getRsaPubkey();
                    if (TextUtils.isEmpty(rsaKey)) {
                        return null;
```

```
                    }
                    loginRequest.put("password", Rsa.encrypt(this.password, rsaKey));
                } catch (Exception var4) {
                    return null;
                }
            }
            LoginActivity.this.hideSoftInputForHw();
            Result result;
            if (!CommonUtils.isNetworkAvailable()) {
                if (ConfigManager.getInstance().isSupportOfflineLogin()) {
                    return LoginActivity.this.tryOfflineLogin(this.loginId, this.passwo
rd);
                } else {
                    result = new Result();
                    result.code = 10014;
                    result.message = MessageUtils.getMessageContent(10014, new Object[0
]);
                    return result;
                }
            } else {
                if (this.sig != null) {
                    loginRequest.put("sig", this.sig);
                }
                if (!TextUtils.isEmpty(this.cSessionId)) {
                    loginRequest.put("csessionid", this.cSessionId);
                }
                if (!TextUtils.isEmpty(this.nocToken)) {
                    loginRequest.put("nctoken", this.nocToken);
                }
                result = OpenAccountUtils.toLoginResult(RpcUtils.pureInvokeWithRiskCont
rolInfo("loginRequest", loginRequest, "login"));
                return ConfigManager.getInstance().isSupportOfflineLogin() && result.co
de == 10019 ? LoginActivity.this.tryOfflineLogin(this.loginId, this.password) : result;
            }
        }
        protected void doWhenException(Throwable t) {
            this.executorService.postUITask(new Runnable() {
                public void run() {
                    ToastUtils.toastSystemError(LoginTask.this.context);
                }
            });
        }
        protected void onPostExecute(Result<LoginResult> result) {
            this.dismissProgressDialog();
            super.onPostExecute(result);
            try {
                if (result == null) {
                    if (ConfigManager.getInstance().isSupportOfflineLogin()) {
                        ToastUtils.toastNetworkError(this.context);
                    } else {
                        ToastUtils.toastSystemError(this.context);
                    }
                } else {
                    android.net.Uri.Builder builder;
                    Intent h5Intent;
```

```
                    String accountName;
                    switch(result.code) {
                    case 1:
                        if (result.data != null && ((LoginResult)result.data).loginSucc
essResult != null) {
                            SessionData sessionData = OpenAccountUtils.createSessionDat
aFromLoginSuccessResult(((LoginResult)result.data).loginSuccessResult);
                            if (sessionData.scenario == null) {
                                sessionData.scenario = 1;
                            }
                            LoginActivity.this.sessionManagerService.updateSession(sess
ionData);
                            accountName = ((LoginResult)result.data).userInputName;
                            if (TextUtils.isEmpty(accountName)) {
                                accountName = this.loginId;
                            }
                            if (ConfigManager.getInstance().isSupportOfflineLogin()) {
                                OpenAccountSDK.getSqliteUtil().saveToSqlite(this.loginI
d, this.password);
                            }
                            boolean isExist = LoginActivity.this.loginIdEdit.saveInputH
istory(accountName);
                            if (AccountPasswordLoginFlow.showTipAlertAfterLogin && !isE
xist) {
                                String message = ResourceUtils.getString(LoginActivity.
this.getApplicationContext(), "ali_sdk_openaccount_dynamic_text_alert_msg_after_login")
;
                                LoginActivity.this.showTipDialog(String.format(message,
this.loginId));
                            } else {
                                LoginActivity.this.loginSuccess();
                            }
                            return;
                        }
                        break;
                    case 2:
                        SessionData sessionData1 = OpenAccountUtils.createSessionDataFr
omLoginSuccessResult(((LoginResult)result.data).loginSuccessResult);
                        if (sessionData1.scenario == null) {
                            sessionData1.scenario = 1;
                        }
                        LoginActivity.this.sessionManagerService.updateSession(sessionD
ata1);
                        LoginActivity.this.loginSuccess();
                        break;
                    case 4037:
                        if (AccountPasswordLoginFlow.showAlertForPwdErrorToManyTimes) {
                            String postive = LoginActivity.this.getResources().getStrin
g(string.ali_sdk_openaccount_text_confirm);
                            accountName = LoginActivity.this.getResources().getString(s
tring.ali_sdk_openaccount_text_reset_password);
                            final ToastUtils toastUtils = new ToastUtils();
                            android.content.DialogInterface.OnClickListener postiveList
ener = new android.content.DialogInterface.OnClickListener() {
```

```
                                public void onClick(DialogInterface dialog, int which)
{
                                    toastUtils.dismissAlertDialog(LoginActivity.this);
                                }
                            };
                            android.content.DialogInterface.OnClickListener negativeLis
tener = new android.content.DialogInterface.OnClickListener() {
                                public void onClick(DialogInterface dialog, int which)
{
                                    LoginActivity.this.forgetPassword((View)null);
                                }
                            };
        //重写toastUtils类的alert()方法
                            toastUtils.alert(LoginActivity.this, "", result.message, po
stive, postiveListener, accountName, negativeListener);
                        } else {
                            ToastUtils.toast(this.context, result.message, result.code)
;
                        }
                        break;
                    case 26053:
                        if (result.data != null && ((LoginResult)result.data).checkCode
Result != null && !TextUtils.isEmpty(((LoginResult)result.data).checkCodeResult.clientV
erifyData)) {
                            builder = Uri.parse(((LoginResult)result.data).checkCodeRes
ult.clientVerifyData).buildUpon();
                            builder.appendQueryParameter("callback", "https://www.alipa
y.com/webviewbridge");
                            h5Intent = new Intent(LoginActivity.this, LoginDoubleCheckW
ebActivity.class);
                            h5Intent.putExtra("url", builder.toString());
                            h5Intent.putExtra("title", result.message);
                            h5Intent.putExtra("callback", "https://www.alipay.com/webvi
ewbridge");
                            LoginActivity.this.startActivityForResult(h5Intent, Request
Code.NO_CAPTCHA_REQUEST_CODE);
                            return;
                        }
                        break;
                    case 26152:
                        if (result.data != null && ((LoginResult)result.data).checkCode
Result != null && !TextUtils.isEmpty(((LoginResult)result.data).checkCodeResult.clientV
erifyData)) {
                            builder = Uri.parse(((LoginResult)result.data).checkCodeRes
ult.clientVerifyData).buildUpon();
                            builder.appendQueryParameter("callback", "https://www.alipa
y.com/webviewbridge");
                            h5Intent = new Intent(LoginActivity.this, LoginIVWebActivit
y.class);
                            h5Intent.putExtra("url", builder.toString());
                            h5Intent.putExtra("title", result.message);
                            h5Intent.putExtra("callback", "https://www.alipay.com/webvi
ewbridge");
                            LoginActivity.this.startActivityForResult(h5Intent, Request
```

```
Code.RISK_IV_REQUEST_CODE);
                        }
                        break;
                    default:
                        if (TextUtils.equals(result.type, "CALLBACK") && LoginActivity.
this.getLoginCallback() != null) {
                            LoginActivity.this.getLoginCallback().onFailure(result.code
, result.message);
                            return;
                        }
                        LoginActivity.this.onPwdLoginFail(result.code, result.message);
                    }
                }
            } catch (Throwable var8) {
                AliSDKLogger.e("oa", "after post execute error", var8);
                ToastUtils.toastSystemError(this.context);
            }
        }
    }
```

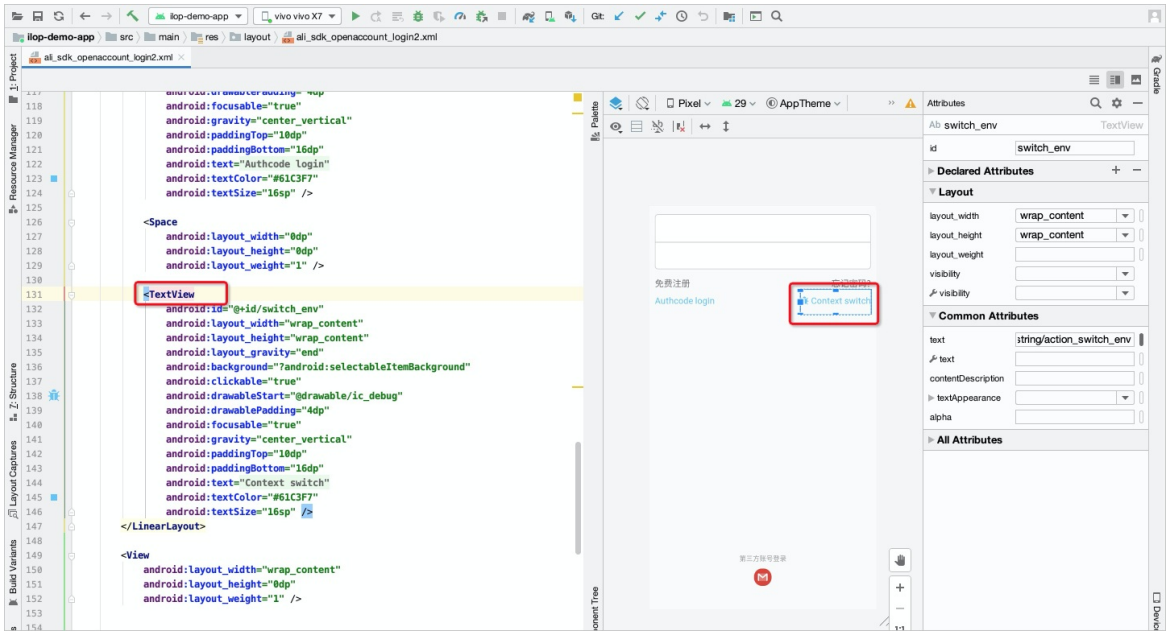
2. 定制弹出的对话框，并显示 `result.message` 的错误信息。

```
toastUtils.alert(LoginActivity.this, "", result.message, postive, postiveListener, acco
untName, negativeListener);
```

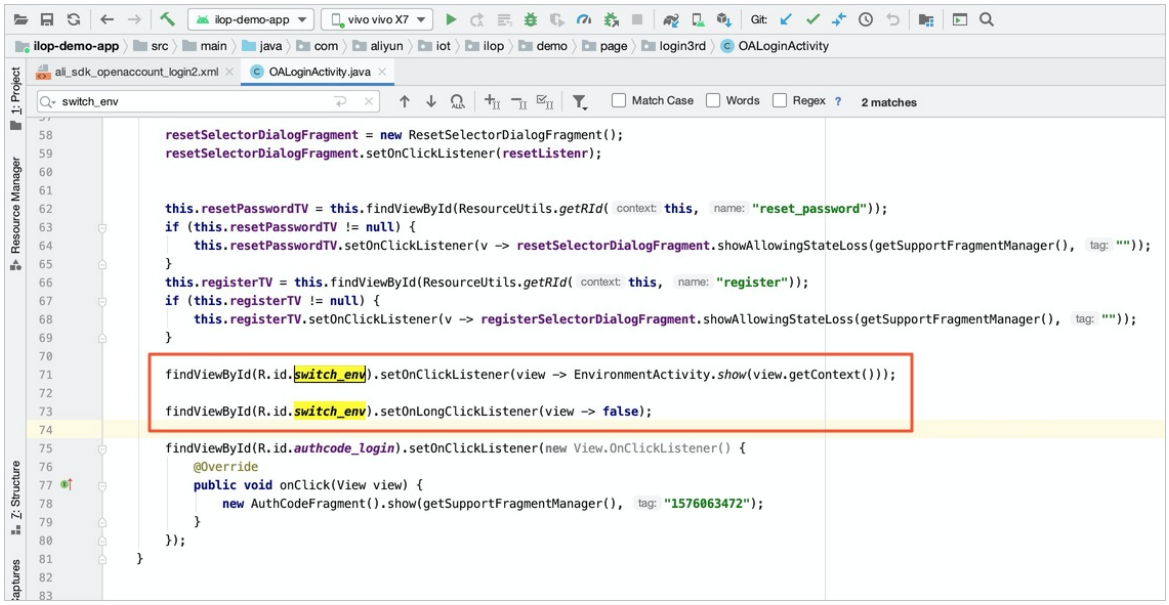
新增控件和单击事件

当您需要在App中新增一个控件，并添加相应的单击事件时，可以根据以下步骤来操作。

1. 在界面上新增一个控件，并在重写的xml文件中添加控件信息。

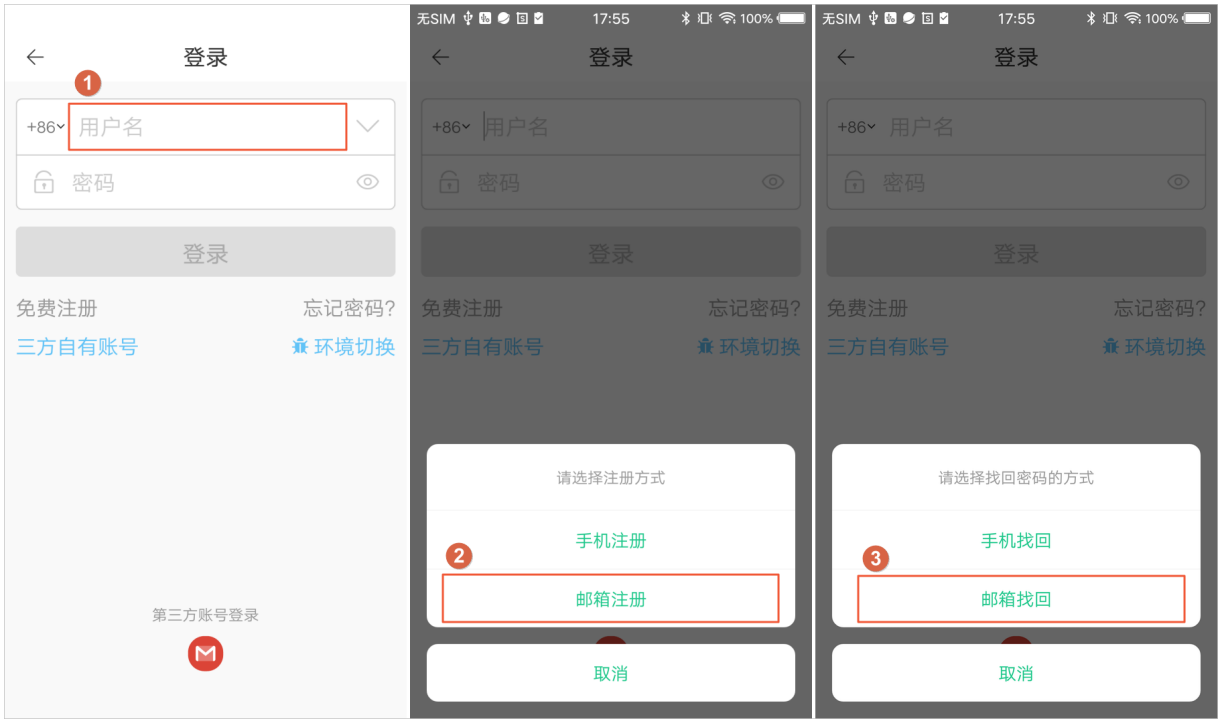


2. 在activity中添加该控件的单击事件。



新增邮箱登录方式

如果您需要App支持通过邮箱方式登录，您需要开发邮箱相关的功能，包括邮箱登录（图示中①）、邮箱注册（图示中②）、邮箱重置密码（图示中③）等。



1. 在App登录页面增加邮箱登录的方式。

   Android账号及用户SDK的OALoginActivity已默认支持手机号登录和邮箱登录。此时您只需要将手机区号隐藏，并修改登录提示即可。

   i. 隐藏手机区号。

   ```
   //隐藏手机区号
   OpenAccountUIConfigs.AccountPasswordLoginFlow.supportForeignMobileNumbers = false;
   ```

ii. 修改邮箱登录信息。

```
//修改登录提示信息
this.loginIdEdit.getEditText().setHint("请输入手机号或邮箱账号");
```

2. 新增邮箱注册功能。

i. 在登录页面增加**邮箱注册**按钮。

请参见本文档中新增控件和单击事件来实现。

ii. 定义注册的回调函数。

```
//注册的回调函数
 private EmailRegisterCallback getEmailRegisterCallback() {
        return new EmailRegisterCallback() {
            @Override
            public void onSuccess(OpenAccountSession session) {
                //注册成功
            }
            @Override
            public void onFailure(int code, String message) {
                //注册失败
            }
            @Override
            public void onEmailSent(String email) {
            }
        };
    }
```

iii. 跳转到邮箱注册界面。

```
//跳转到邮箱注册界面
OpenAccountUIService openAccountUIService = OpenAccountSDK.getService(OpenAccountUI
Service.class);
openAccountUIService.showEmailRegister(OALoginActivity.this, getEmailRegisterCallba
ck());
```

3. 新增邮箱重置密码功能。

通过邮箱方式登录App时，如果忘记了邮箱注册的密码，需要通过邮箱重置密码的功能来找回密码。

i. 在登录页面增加**邮箱找回**按钮。

请参见本文档中新增控件和单击事件来实现。

ii. 定义邮箱密码找回的回调方法。

```
private EmailResetPasswordCallback getEmailResetPasswordCallback() {
     return new EmailResetPasswordCallback() {
         @Override
         public void onSuccess(OpenAccountSession session) {
             //修改成功
         }
         @Override
         public void onFailure(int code, String message) {
            //修改失败
         }
         @Override
         public void onEmailSent(String email) {
             //发送验证码邮件
         }
     };
 }
```

iii. 跳转到修改密码界面。

```
OpenAccountUIService openAccountUIService = (OpenAccountUIService) OpenAccountSDK.g
etService(OpenAccountUIService.class);
openAccountUIService.showEmailResetPassword(this, this.getEmailResetPasswordCallbac
k());
```

更多UI定制参考

如果您还需定制更多的UI，例如修改更多原生元素，您可以根据以下内容来自行实现。

- 修改更多组件的样式

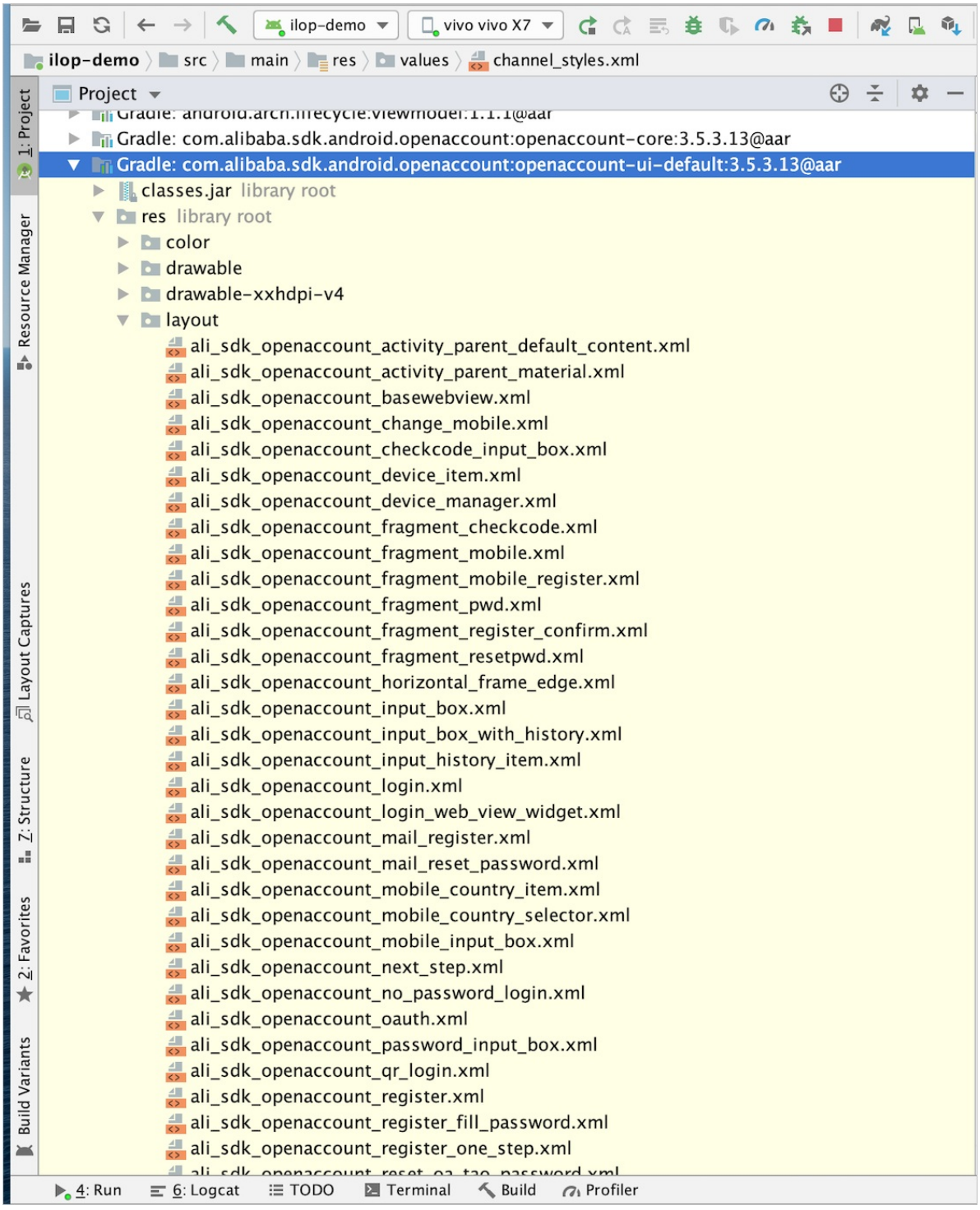
  账号及用户SDK开放了所有组件的样式，在SDK的*deali_sdk_openaccount_styles.xml*里定义了可以直接覆盖的所有样式。您可以继承对应的style再修改AndroidManifest.xml中activity的配置。详细操作请参见本文档中**修改登录和注册按钮的颜色**来实现。

- 修改OA界面布局

  控件或Activity通过LayoutMapping类来加载layout文件。建议您不要修改现有控件的ID，直接新增控件来加载自定义的layout文件。

  ```
  com.alibaba.sdk.android.openaccount.ui.LayoutMapping.put(控件(或activity).class,R.layout.x
  xx);
  ```

  您可以在下图所示目录下查找所需的layout文件。



- 定制OA模块界面

  如果您需要自定义整个界面，建议您集成原先界面的activity后，再替换SDK中整个界面的内容。您可以参照以下示例代码来实现。

```
//以登录界面为例
//1.继承LoginActivity，重写登录界面为OALoginActivity
   public class OALoginActivity extends LoginActivity{}
//2.在 OALoginActivity中重写getLayoutName方法，将布局文件的名称返回
@Override
    protected String getLayoutName() {
        //替换成自己的布局文件名称
        return "ali_sdk_openaccount_login2";
    }
//3.在账号及用户SDK初始化后，将登录界面替换到SDK中
OALoginAdapter adapter = (OALoginAdapter) LoginBusiness.getLoginAdapter();
adapter.setDefaultLoginClass(OALoginActivity.class);
```

> **说明** 在重写界面时，需将原界面中的控件全部添加上，且保持ID不变。不需用的控件隐藏处理，请勿删除，否则可能出现找不到控件的错误。

如果您还需定制除原生元素以外的内容，可以参考以下信息来实现。

- 快速集成
- UI界面自定义
- 云账号与三方账号集成
- 云账号国际化处理

4.2. 定制iOS App的OA UI

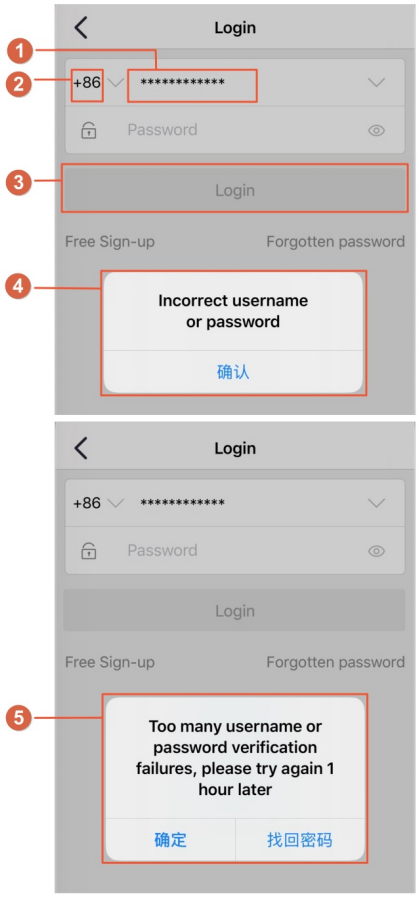
基于生活物联网平台的账号及用户SDK，您可以自定义自有品牌App的开放账号OA（Open Account）模块相关的用户界面UI（User Interface），主要包括登录页面、注册页面、密码重置页面等。

前提条件

已完成账号及用户SDK的开发，详细操作，请参见账号及用户SDK。

定制项

iOS App的OA UI定制项如下。

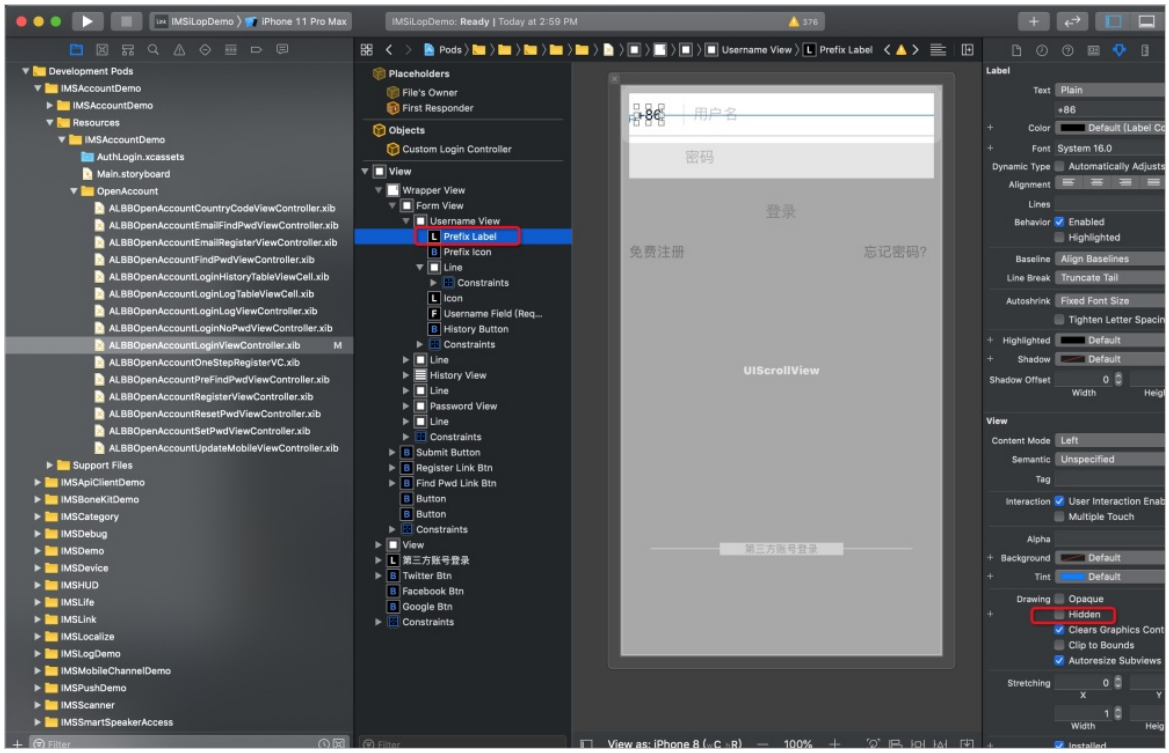
App界面图示	可定制内容
	修改原生元素 • 显示手机区号（图示中②） • 修改登录和注册按钮的颜色（图示中③） • 修改失败提示样式（图示中④） 新增元素 • 新增控件和单击事件 • 现有控件增加自定义的事件 • 新增邮箱登录方式（图示中①） • 登录密码输入次数超限后提示找回密码（图示中⑤）

显示手机区号

登录和注册页面中默认不显示手机区号，如果您需要显示手机区号，如+86，请根据以下步骤来操作。

1. 打开*ALBBOpenAccountLoginViewController.xib*文件，选择**View > Wrapper View > Form View > Username View > Prefix Label**。
2. 打开右侧控制面板，选择Prefix Label，并取消选中**Hidden**复选框。

   此时即可显示手机区号。



3. 在命令区域，执行**pod Update**命令，查看显示效果。

   如果页面中出现内容重叠，请在*ALBBOpenAccountLoginViewController.xib*中更改相关约束，通过显示和隐藏其他控件来调节显示效果。

修改登录和注册按钮的颜色

App的登录和注册按钮默认为浅灰色，如果您需要修改App登录和注册按钮的颜色，请根据以下步骤来操作。

1. 注册ALBBOpenAccountLoginViewDelegate相关代理。
2. 生成并替换按钮颜色的图片。

   您可以通过放置图片的方法来替换按钮颜色，以下为您提供颜色生成图片的扩展方法。

```
#pragma mark - ALBBOpenAccountLoginViewDelegate
- (void)loginViewDidLoad:(ALBBOpenAccountLoginViewController *)viewController {
    // 本示例中按钮的三种状态（正常、高亮、禁用）的颜色一致，您如需不一致，传入不同图片即可
    UIImage *bgImage = [UIImage ims_imageWithColor:[UIColor whiteColor];
    [viewController.submitButton setBackgroundImage:bgImage forState:UIControlStateNorm
al];
    [viewController.submitButton setBackgroundImage:bgImage forState:UIControlStateHigh
lighted];
    [viewController.submitButton setBackgroundImage:bgImage forState:UIControlStateDisa
bled];
}
// 分类方法
+ (UIImage *)ims_imageWithColor:(UIColor *)color {
  CGRect rect = CGRectMake(0.0f, 0.0f, 1.0f, 1.0f);
  UIGraphicsBeginImageContext(rect.size);
  CGContextRef context = UIGraphicsGetCurrentContext();
  CGContextSetFillColorWithColor(context, [color CGColor]);
  CGContextFillRect(context, rect);
  UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
  UIGraphicsEndImageContext();
  return image;
}
```

修改提示信息样式

当OA模块发生业务错误（如密码错误）时，App会弹出相关提示信息的对话框。如果您需要修改该提示对话框的样式，可以通过设置回调来实现。

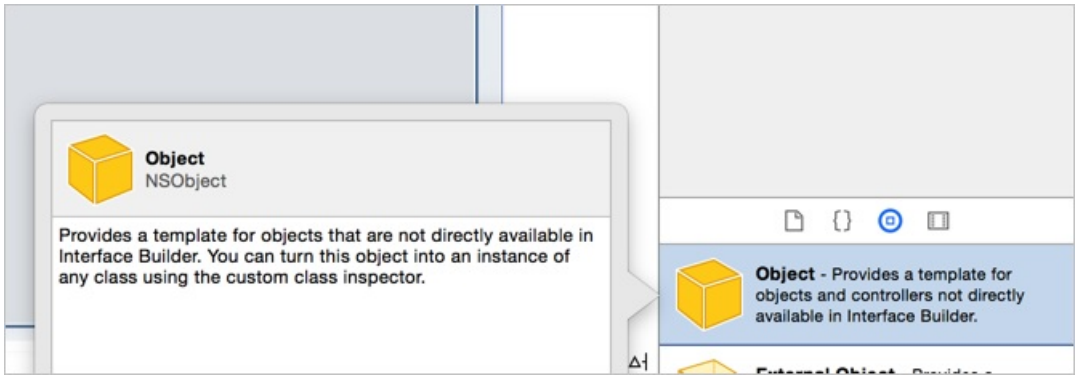
```
[ALBBService(ALBBOpenAccountUIService) setHandleBizErrorCallback:^(NSString *errMsg) {
     UIAlertController *controller = [UIAlertController alertControllerWithTitle:nil messag
e:errMsg preferredStyle:UIAlertControllerStyleAlert];
    [controller addAction:[UIAlertAction actionWithTitle:@"确认" style:UIAlertActionStyleDef
ault handler:nil]];
    [[self getCurrentVC] presentViewController:controller animated:true completion:nil];
}];
```

其中，`getCurrentVC`方法可参见本文档**登录密码输入超限后提示找回密码**中`getCurrentVC`的方法。

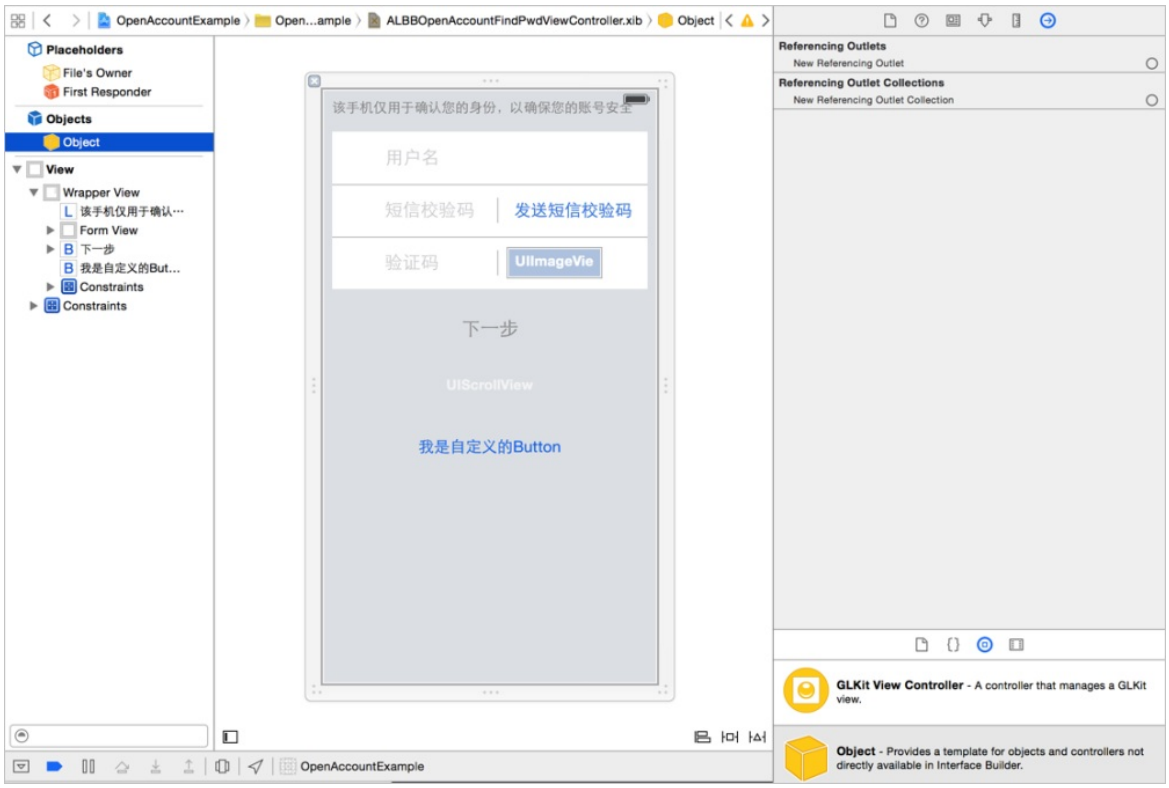
新增控件和单击事件

当您需要在App中增加新的单击事件时，例如新增一个Button控件的单击事件，您可以根据以下步骤来操作。

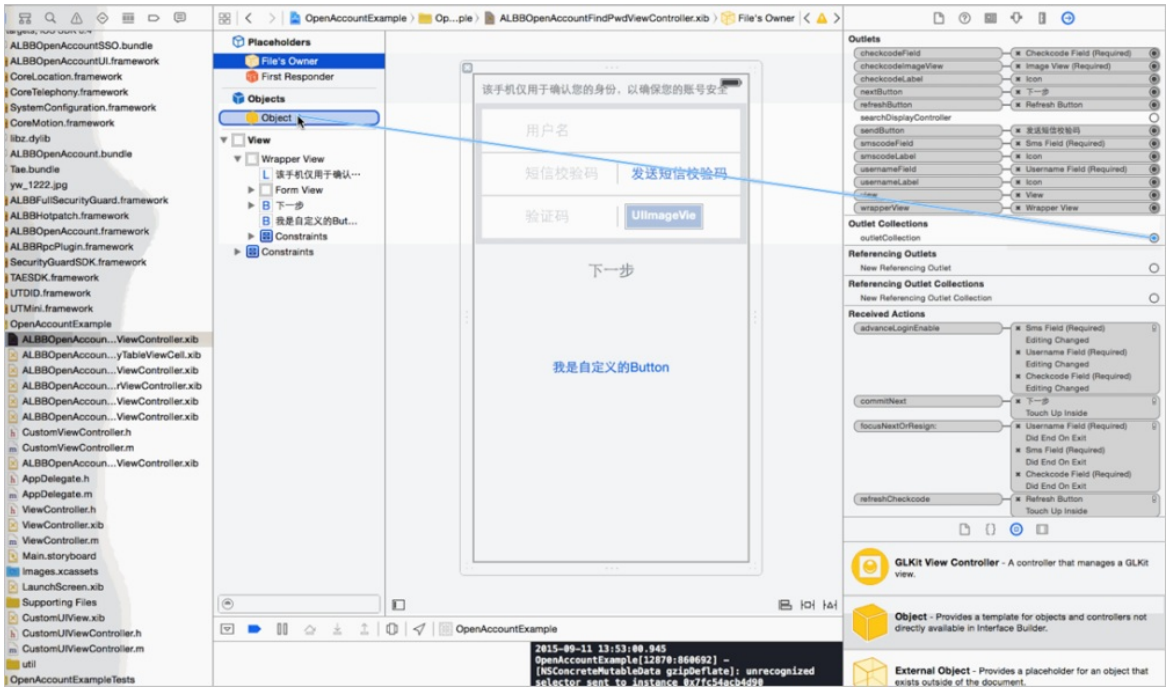
1. 在*ALBBOpenAccountFindPwdViewController.xib*中添加一个Button元素。例如，Button标题命名为：我是自定义的Button。
2. 在*xib*目录中添加一个如图所示的Object控件。



添加成功后，如下图所示。



3. 选择**File's Owner**，并单击outletCollection对应的加号（+），添加一个控制器，例如命名为Object。
4. 按住鼠标从如图所示的右下角拖至左上角，关联Object控制器和outletCollection。

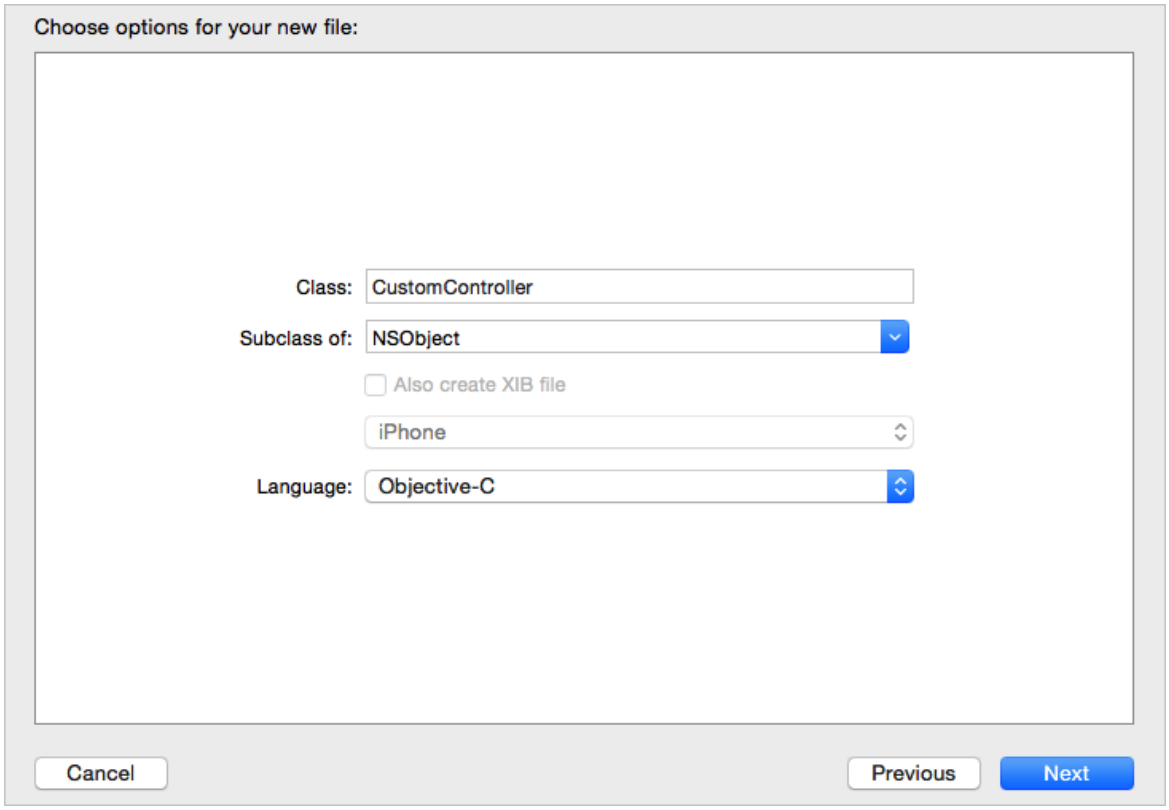


此处outletCollection是一个NSArray，您也参照以下代码，一次关联多个Object控件。

```
@property (nonatomic, strong) IBOutletCollection(NSObject)? NSArray *outletCollection;
```

5. 新建一个类，例如命名为CustomController。

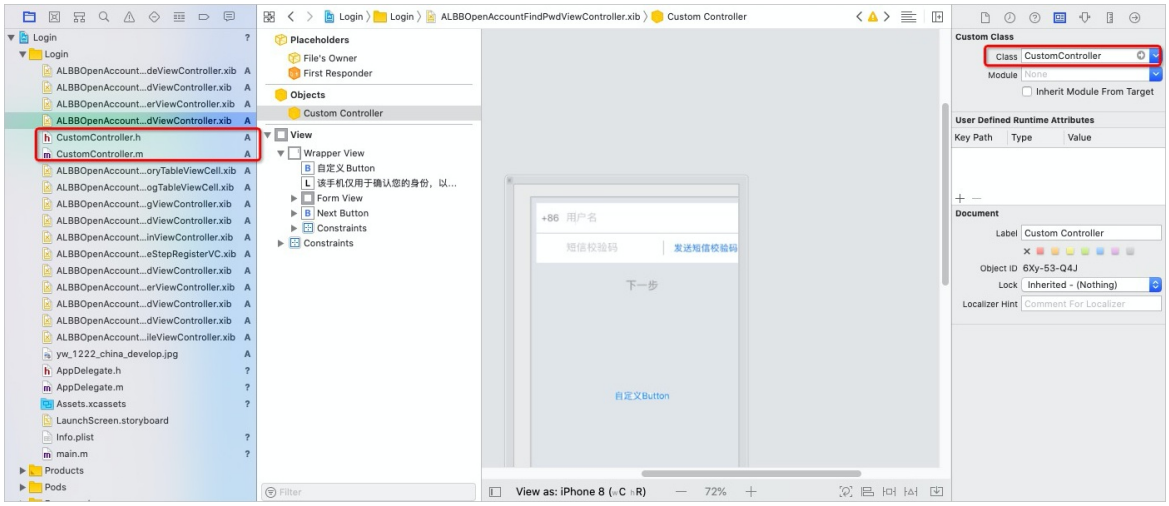
   该类继承NSObject，设置Objec控制器的Class参数值为CustomController。



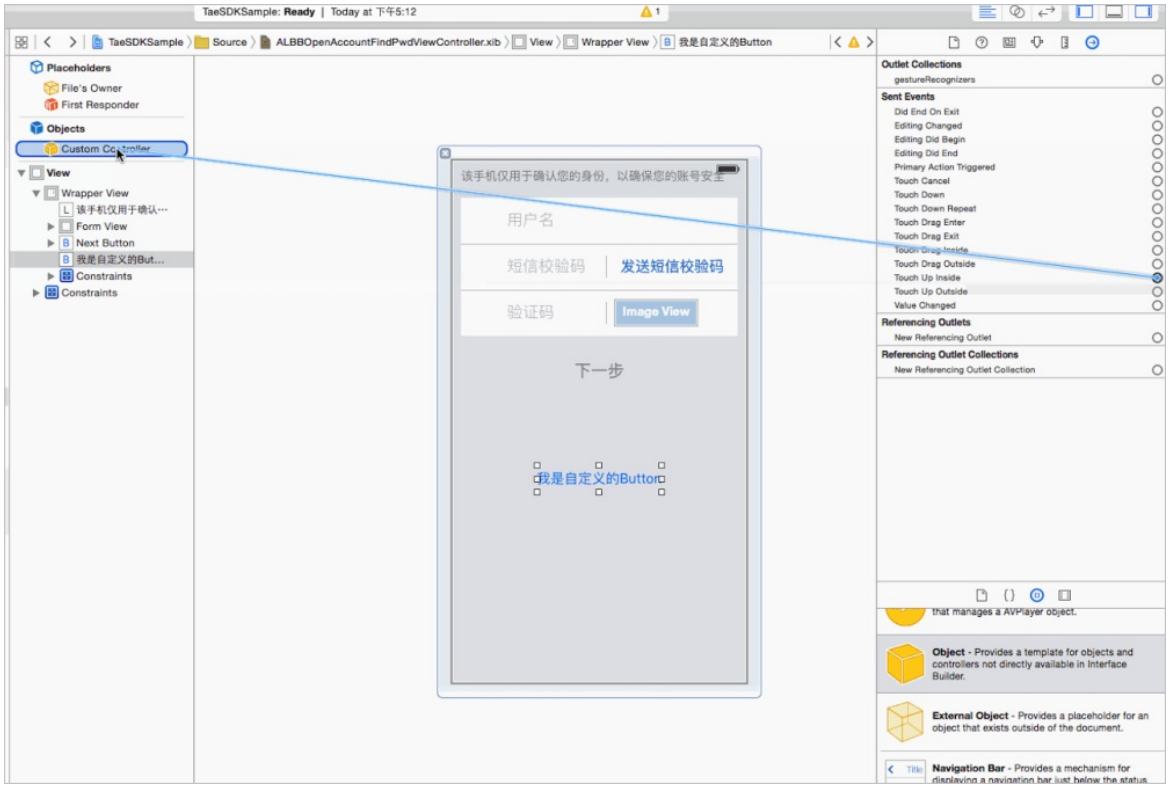
6. 在CustomController类中添加一个响应函数，例如命名为IBAction。

```
-(IBAction)CustomButtonClick {
    NSLog(@"来自 Custom Button 的Click");
}
```

7. 将新建的Object与新建的Class相关联，如下图所示。

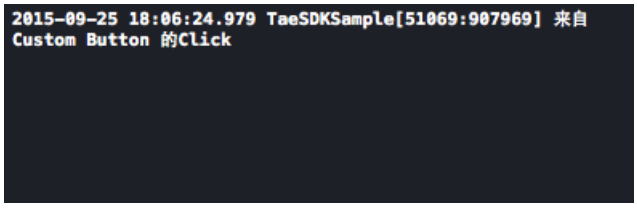


8. 按住鼠标从如图所示的右下角拖至左上角，关联Button的单击事件和CustomController类，即选择CustomButtonClick函数作为Button单击事件的响应函数。



9. 单击CustomController对应的Button，验证单击事件。

   如果打印出以下日志则表示单击事件可正常使用。如果您需要单击Button后跳转其他页面，可自行实现。



现有控件增加自定义事件

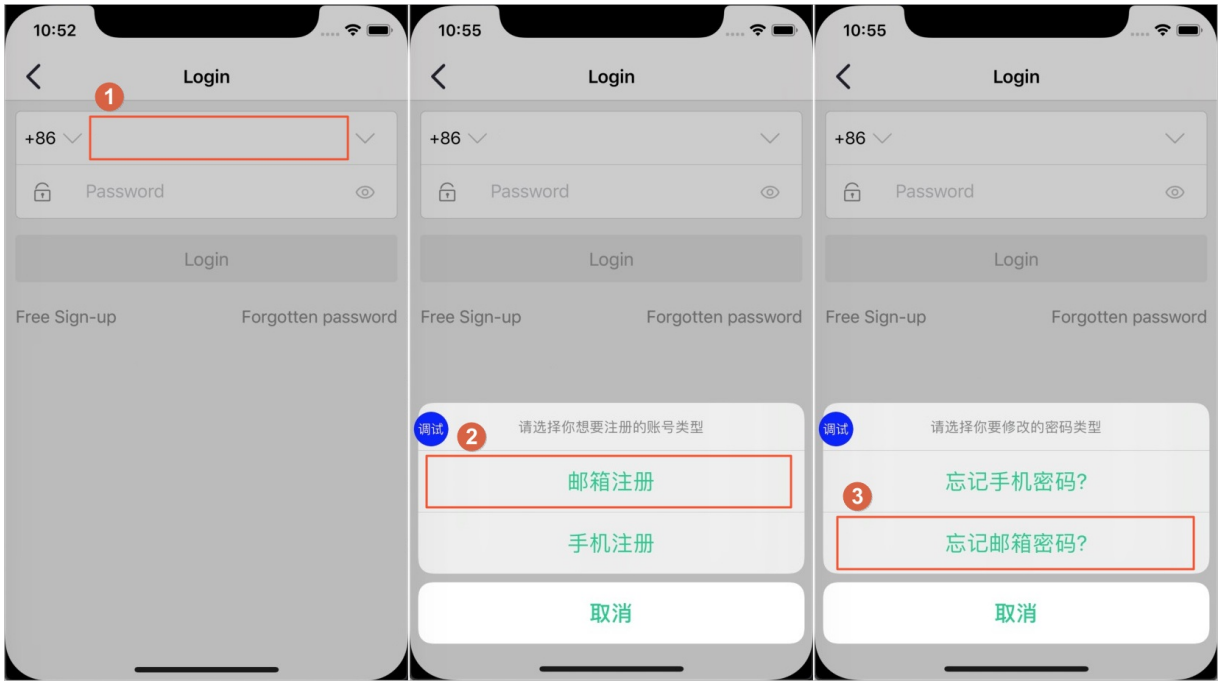
当您需要对现有控件增加自定义事件时，例如统计某控件的单击次数等，您可以参考以下示例方法来更改OA内部逻辑并增加自定义事件。

> **说明** 通常不建议您通过Runtime方式来修改非主动暴露的业务逻辑，建议您优先联系技术支持，确认是否有新版本SDK支持该功能，或者提交该功能的需求申请。

```
//以统计按钮单击次数的逻辑为例
@implementation ALBBOpenAccountSetPwdViewController (aspect)
+ (void)load {
    Method originalMethod = class_getInstanceMethod(self, @selector(submitPassword));
    Method newMethod = class_getInstanceMethod(self, @selector(newSubmitPassword));
    BOOL addMethod = class_addMethod(self, @selector(submitPassword), method_getImplementation(newMethod), method_getTypeEncoding(newMethod));
    if (addMethod) {
        class_replaceMethod(self, @selector(newSubmitPassword), method_getImplementation(originalMethod), method_getTypeEncoding(originalMethod));
    } else {
        method_exchangeImplementations(originalMethod, newMethod);
    }
}
- (void)newSubmitPassword {
    [self newSubmitPassword];
   //处理统计次数逻辑
}
@end
```

新增邮箱登录方式

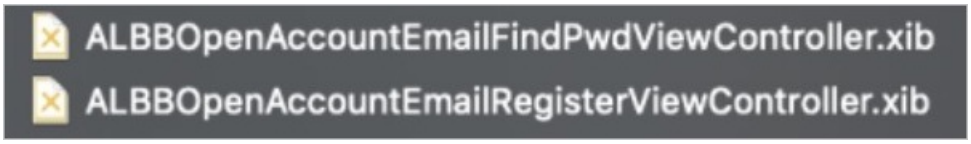
如果您需要App支持邮箱方式登录，您需要开发邮箱相关的功能，包括邮箱登录（图示中①）、邮箱注册（图示中②）、忘记邮箱密码（图示中③）等。



1. （可选）从账号及用户SDK的Bundle中拷贝邮箱相关的两个*xib*文件到工程目录下。

   如果您之前开发过邮箱相关的功能，可直接跳过该步骤的操作。

   邮箱相关的*xib*文件如下图所示，其中头文件为 `#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>` 。

2. 在App登录页面增加邮箱登录的方式。

   邮箱登录与手机号登录可以共用同一个登录框，此时您只需要将手机区号隐藏，并在代码中判断输入的内容是邮箱还是手机号即可。如果您需要在UI上区分，可自行实现。

3. 新增邮箱注册功能。

   i. 确认账号及用户SDK版本。

   ```
   pod 'AlicloudALBBOpenAccount', '3.4.0.39' //3.4.0.39及以上版本都支持
   ```

   ii. 在登录页面增加**邮箱注册**按钮。

   请参见本文档中新增控件和单击事件来实现。

   iii. 调用控制器弹出邮箱注册页面。

   ```
   id uiService = ALBBService(ALBBOpenAccountUIService);
   [uiService showEmailRegisterInNavigationController:self.navigationController succes
   s:nil failure:nil];
   ```

4. 新增忘记邮箱密码功能。

   通过邮箱方式登录App时，如果忘记了邮箱注册的密码，需要通过邮箱找回密码。

i. 确认账号及用户SDK版本。

```
pod 'AlicloudALBBOpenAccount', '3.4.0.39' //3.4.0.39及以上版本都支持
```

ii. 在登录页面增加**忘记邮箱密码**按钮。

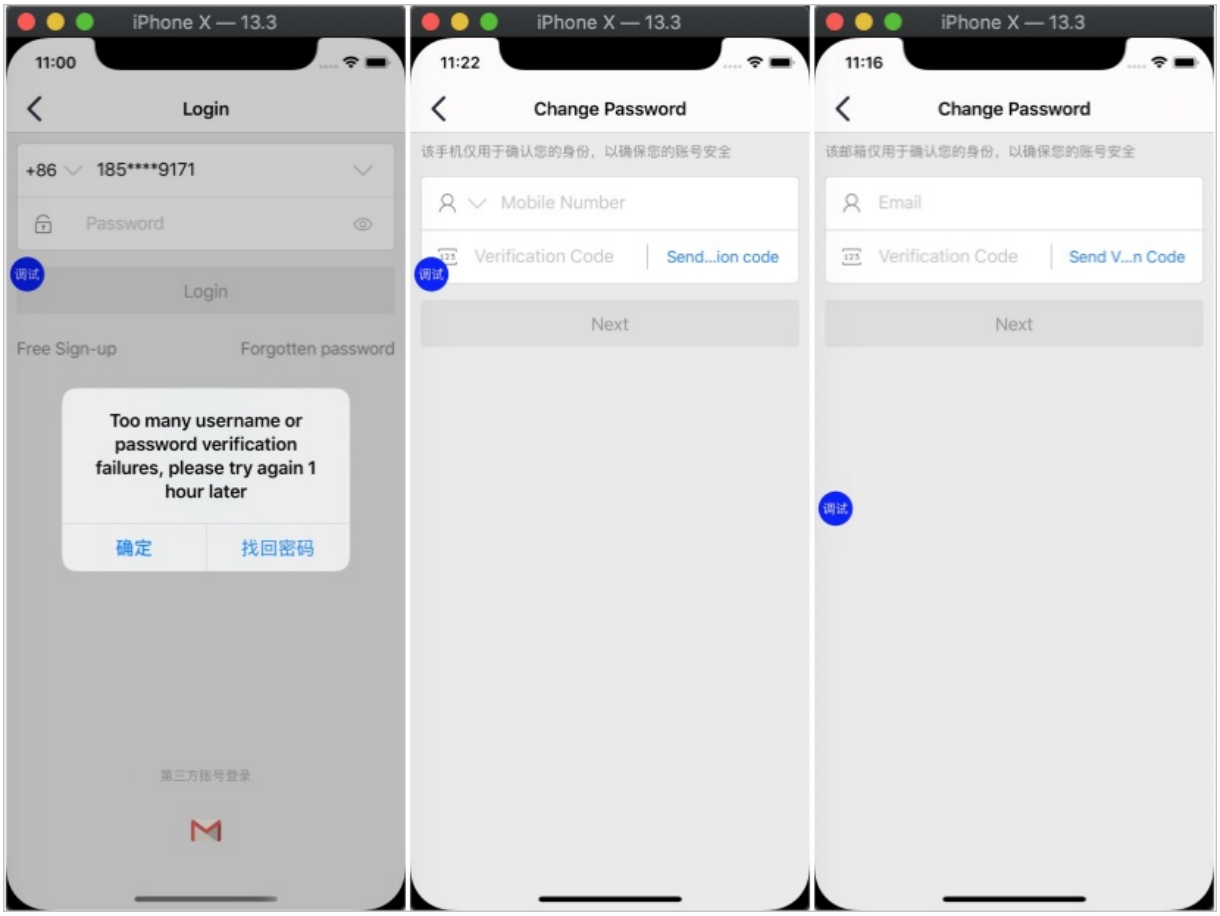
请参见本文档中新增控件和单击事件来实现。

iii. 调用控制器弹出忘记密码页面。

```
id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUIService);
[uiService showFindPasswordInNavigationController:self.navigationController success
:nil failure:nil];
```

登录密码输入次数超限后提示找回密码

登录App时，如果输入登录密码的次数达到上限，App会限制继续操作，并提示找回密码。登录账号如果是手机号码则跳转至手机忘记密码页面；如果是邮箱则跳转至邮箱忘记密码页面。



有两种方案可实现登录密码输入次数超限后提示找回密码，您任选一种即可。

> **说明** 如果您同时添加了以下两种方案，则仅生效第二种方案。运行App后，添加功能代码逻辑的方式（即反射+Runtime自定义跳转的方式）会修改SDK中的业务逻辑。

- 升级SDK

  最新版本账号及用户SDK中已默认支持该能力，您可以通过升级SDK来实现该功能。完成SDK升级后，您无需额外操作。

```
pod 'AlicloudALBBOpenAccount', '3.4.0.39'  //3.4.0.39及以上版本都支持
```

- 添加该功能代码逻辑

  您还可以通过反射+Runtime自定义跳转的方式来实现该功能。

```
// 通过runtime方式做一个方法来替换，将ALBBOpenAccountLoginViewController的私有方法showFindPassw
ordView转到自己定义的方法findPwdStyleChoose中
+ (void)load {
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        Method findPwdMethod = class_getInstanceMethod([self class], @selector(findPwdStyl
eChoose));
        IMP findPwdNewImp = method_getImplementation(findPwdMethod);
        const char * typeEncodeing = method_getTypeEncoding(findPwdMethod);
        class_replaceMethod([ALBBOpenAccountLoginViewController class], NSSelectorFromStri
ng(@"showFindPasswordView"), findPwdNewImp, typeEncodeing);
    });
}
/**
 显示找回密码方式sheet
 */
- (void)findPwdStyleChoose {
    __weak typeof(self) weakSelf = self;
    UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"请选择你要修改
的密码类型" message:nil preferredStyle:UIAlertControllerStyleActionSheet];
    UIAlertAction *cancelAction = [UIAlertAction actionWithTitle:@"取消" style:UIAlertActio
nStyleCancel handler:^(UIAlertAction * action) {
    }];
    UIAlertAction *phoneAction = [UIAlertAction actionWithTitle:@"忘记手机密码?" style:UIAl
ertActionStyleDefault handler:^(UIAlertAction * action) {
        id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUIService);
        [uiService showFindPasswordInNavigationController:[weakSelf getCurrentVC].navigat
ionController success:nil failure:nil];
    }];
    UIAlertAction *emailAction = [UIAlertAction actionWithTitle:@"忘记邮箱密码?" style:UIAl
ertActionStyleDefault handler:^(UIAlertAction * action) {
        id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUIService);
        [uiService showEmailFindPasswordInNavigationController:[self getCurrentVC].naviga
tionController success:nil failure:nil];
    }];
    [alert addAction:cancelAction];
    [alert addAction:phoneAction];
    [alert addAction:emailAction];
    [[self getCurrentVC] presentViewController:alert animated:YES completion:nil];
}
- (UIViewController *)getCurrentVC {
    UIViewController *result = nil;
    UIWindow *window = [[UIApplication sharedApplication] keyWindow];
    if (window.windowLevel != UIWindowLevelNormal) {
        NSArray *windows = [[UIApplication sharedApplication] windows];
        for (UIWindow *temp in windows) {
            if (temp.windowLevel == UIWindowLevelNormal) {
                window = temp;
                break;
```

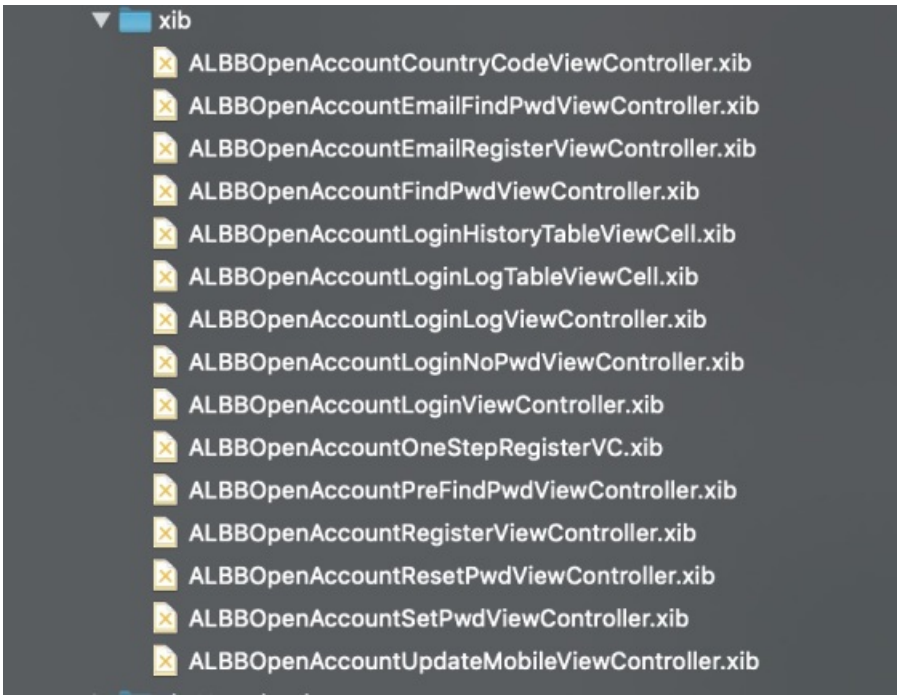
```
                }
            }
        }
    result = window.rootViewController;
    while (result.presentedViewController) {
        result = result.presentedViewController;
    }
    if ([result isKindOfClass:[UITabBarController class]]) {
        result = [(UITabBarController *)result selectedViewController];
    }
    if ([result isKindOfClass:[UINavigationController class]]) {
        result = [(UINavigationController *)result visibleViewController];
    }
    return result;
}
```

更多UI定制

如果您还需定制更多的UI，例如修改更多原生元素，您可以根据以下内容自行实现。

- 账号及用户SDK开放了所有xib组件，每个xib对应相应的功能页。

  您可以在保证云账号交互逻辑的基础上，通过修改以下任意*xib*文件，实现改变页面布局、调整控件样式、新增控件等一系列自定义UI的操作。



- SDK开放了所有ViewController，以及每一个ViewController的UI控件的引用。

  以登录页面对应的UI控件ALBBOpenAccountLoginViewController为例，控件中包括以下元素。

```
@interface ALBBOpenAccountLoginViewController : ALBBOpenAccountBaseController
@property (assign, nonatomic) BOOL isNeedBackButtonHidden;
//预留的外挂引用
@property (nonatomic, strong) IBOutletCollection(NSObject)  NSArray *outletCollection;
// wrapper
@property (weak, nonatomic) IBOutlet ALBBOpenAccountWrapperView *wrapperView;
// form
@property (weak, nonatomic) IBOutlet NSLayoutConstraint *heightOfFormView;
//locale
@property (weak, nonatomic) IBOutlet UILabel *prefixLabel;
@property (weak, nonatomic) IBOutlet UIButton *prefixIcon;
// username
@property (weak, nonatomic) IBOutlet UILabel *usernameLabel;
@property (weak, nonatomic) IBOutlet UITextField *usernameField;
@property (weak, nonatomic) IBOutlet UIButton *historyButton;
@property (weak, nonatomic) IBOutlet UITableView *historyView;
@property (weak, nonatomic) IBOutlet NSLayoutConstraint *heightOfHistoryView;
// password
@property (weak, nonatomic) IBOutlet UIView *passwordView;
@property (weak, nonatomic) IBOutlet UILabel *passwordLabel;
@property (weak, nonatomic) IBOutlet UITextField *passwordField;
@property (weak, nonatomic) IBOutlet UIButton *visibleButton;
// control
@property (weak, nonatomic) IBOutlet UIButton *submitButton;
- (IBAction)submitLogin;
// 前往国家列表
- (IBAction)prefixNumberChoose:(id)sender;
// sso
- (IBAction)taobaoSSO:(id)sender;
#ifdef WECHAT_SSO
- (IBAction)weChatSSO:(id)sender;
#endif
- (IBAction)weiBoSSO:(id)sender;
- (IBAction)qqSSO:(id)sender;
@property (weak, nonatomic) IBOutlet UIButton *registerLinkBtn;
@property (weak, nonatomic) IBOutlet UIButton *findPwdLinkBtn;
- (IBAction)showRegisterView;
- (IBAction)showFindPasswordView;
@end
```

- SDK开放了每一个ViewController的生命周期回调，便于您定制ViewController的UI控件。

  以ALBBOpenAccountLoginViewDelegate为例，使用方式如下。

  i. 设置一个登录界面的代理。

  ```
  [ALBBService(ALBBOpenAccountUIService) setLoginViewDelegate:self];
  ```

  ii. 设置ALBBOpenAccountLoginViewDelegate代理方式。

```
@protocol ALBBOpenAccountLoginViewDelegate <NSObject>
@optional
- (void)loginViewDidLoad:(ALBBOpenAccountLoginViewController *) viewController;
- (void)loginViewWillAppear:(ALBBOpenAccountLoginViewController *) viewController;
- (void)loginViewDidAppear:(ALBBOpenAccountLoginViewController *) viewController;
- (void)loginViewWillDisappear;
- (void)loginViewDidDisappear;
- (void)loginViewWillLayoutSubviews:(ALBBOpenAccountLoginViewController *) viewContro
ller;
- (void)loginViewDidLayoutSubviews:(ALBBOpenAccountLoginViewController *) viewControl
ler;
@end
```

iii. 使用代理方法改变内容。

```
- (void)loginViewDidLoad:(ALBBOpenAccountLoginViewController *) viewController{
viewController.navigationItem.rightBarButtonItem=[[UIBarButtonItem alloc]   initWithT
itle:@"login" style:UIBarButtonItemStyleDone target:nil action:nil];
}
```

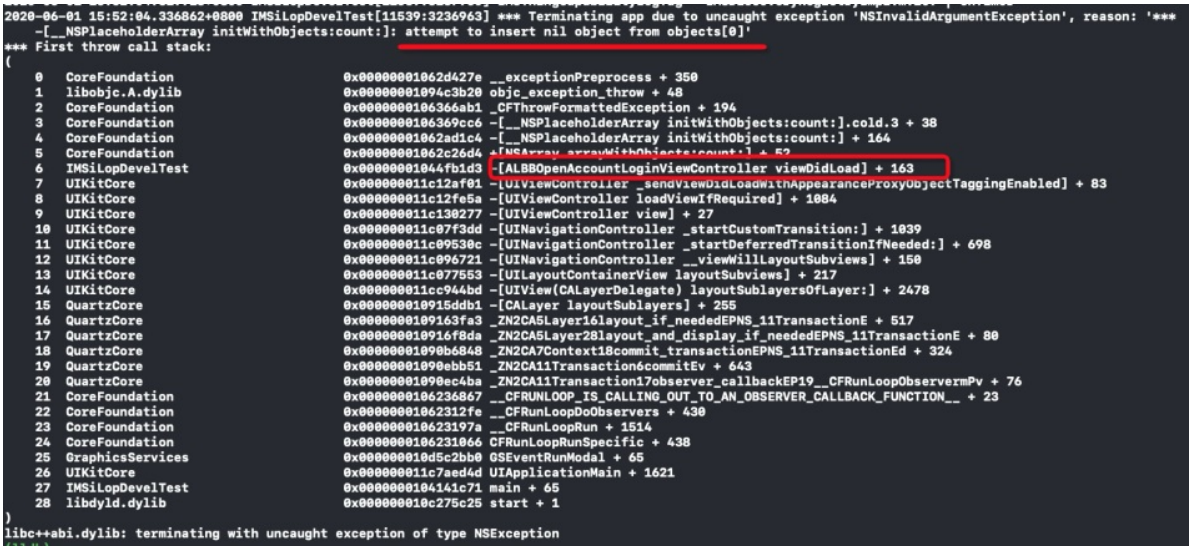
如果您还需定制除原生元素以外的内容，可以参考以下信息来实现。

- 快速集成
- UI界面自定义
- 云账号与三方账号集成
- 云账号支持国外手机号
- 自建UI登录
- 云账号国际化处理

常见问题

- 问题一

  Q：使用账号及用户SDK时，Crash出现以下提示。



  A：将ALBBOpenAccountUI.framework中的*xib*目录放置到主工程目录下即可。

- 问题二

Q：账号及用户SDK初始化成功后，单击App登录按钮，没有成功跳转至相应的页面。

```
#import "OtherViewController.h"
#import "ViewController.h"

@interface OtherViewController ()

@end

@implementation OtherViewController



- (void)viewDidLoad {
    self.view.backgroundColor = [UIColor redColor];
}

- (void)touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent *)event {
    id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUIService);

    ViewController *VC = [[ViewController alloc] init];

    [uiService presentLoginViewController:VC success:^(ALBBOpenAccountSession *currentSession) {
        // 登录成功，currentSession为当前会话信息
        // 获取当前会话标识
        NSLog(@"sessionId:%@", currentSession.sessionID);
        // 获取当前用户信息
        ALBBOpenAccountUser *currentUser = [currentSession getUser];
        NSLog(@"mobile:%@", [currentUser mobile]);
        NSLog(@"avatarUrl:%@", [currentUser avatarUrl]);
        NSLog(@"accountId:%@", [currentUser accountId]);
        NSLog(@"displayName:%@", [currentUser displayName]);
    } failure:^(NSError *error) {
        // 登录失败对应的错误；取消登录同样会返回一个错误码
    }];
```

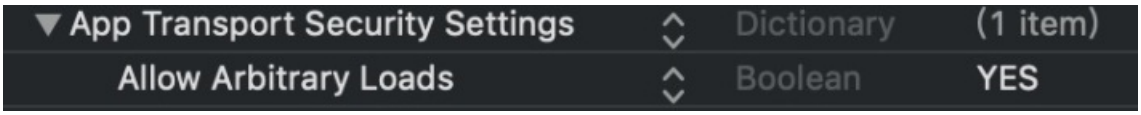
A：示例中VC未加入到导航栈，所以推送登录页面会失败。可以用如下方式推出登录页面。

```
[uiService presentLoginViewController:self success:^(ALBBOpenAccountSession *currentSessi
on) {
        } failure:^(NSError *error) {
        }];
```

- 问题三

  Q：SDK接口报错，提示：Http load fail。

  A：打开工程的*info.plist*文件，设置ATS配置即可。



5.App开发

5.1. 场景自动化最佳实践

生活物联网平台提供的场景自动化是基于物模型TSL（Thing Specification Language）设计的。按照一定的逻辑规律，将设备的物模型、设备动作等按照场景联动，达到自动化执行的效果。

概述

平台针对自动化场景定义一组API协议，API使用Oauth2.0协议进行鉴权。API请参见场景服务。

下面介绍几种创建场景常用的模式。

场景模式	规则描述	备注
TCA（Trigger&Condition&Action）	• type：规则类型，默认为“IFTTT”。 • trigger：（即This）规则的触发器部分，支持单触发器，或通过or模式，集成多个触发器。 • conditon：规则的条件部分，可为空。支持单个条件，或者通过and模式，集成多个条件。 • action：（即That）规则的动作部分，支持多个动作。	IFTTT（If This Than That）规则，是一个JSON描述，包含type、trigger、conditon、action四个部分。即设置一个触发器，当满足条件时，自动触发去执行某动作。
CA（Condition&Action）	• type：规则类型，默认为“CA”。 • conditon：规则的条件部分，不可为空。支持单个条件，或者通过and模式，集成多个条件。 • action：规则的动作部分，支持多个动作。 设备同一属性不能重复设置，条件和动作不能设置相同的设备属性	以CA模式创建场景还分为以下两种情况： • all：当所有conditon都满足，执行操作 conditon支持时间点、设备属性、时间段、等，其中时间段不能作为唯一的条件，需要与其他条件（除时间点外）一起设置。 • any：任一conditon满足，执行操作 conditon支持时间点、设备属性等（不支持时间段作为触发条件）。

Conditon介绍

conditon通常支持时间点、设备属性、时间段，下面介绍这几种的样例代码。

- 时间点

  指定在某个时间点允许执行对应的任务。

```
{
  "uri": "condition/timer",
  "params": {
    "cron": "",          //定时表达式，参考http://crontab.org/
    "cronType": "",      //表达式类型，支持"linux"、"quartz_cron"；"linux"：表示crontab类型，5位；"quartz_cron"：表示quartz类型，7位，支持年。当前只能最小设置分钟级别，第一位秒级，必须是 0
    "timezoneID": ""     //时区
  }
}
```

- 时间段

  指定在一周内的某天的某个时间段中允许执行对应的任务。

```
{
  "uri": "condition/timeRange",
  "params": {
    "format": "",        //时间格式为"HH:mm"
    "beginDate": "",     //开始时间，例如 "08:30" 指condition生效期从一天的早上8点30开始
    "endDate": "",       //结束时间，例如 "23:00" 指condition的生效期到晚上23点00结束
    "repeat": "",        //重复日期，例如 "1,2,3"，一周内的周1周2周3允许执行，支持1~7表示周一到周日，用','分割
    "timezoneID": ""     //时区
  }
}
```

- 设备属性

  当设备属性变更到对应值，触发对应的任务。

```
{
  "uri": "condition/device/property",
  "params": {
    "productKey": "",            //产品productKey
    "deviceName": "",            //设备deviceName
    "propertyName": "",         //属性唯一标识符（产品下唯一），TSL模型中properties的identifier
    "compareType": "",         //比较类型，支持 ">"、"<"、"=="
    "compareValue": ""         //比较的属性值，这里值的类型用TSL模型中properties的type，如果type是int、float等compareValue是数字，type是text时 compareValue是String，以此类推
  }
}
```

Action介绍

Action支持以下几种。

- 设备属性

  满足条件时设置设备属性。

```
{
  "uri": "action/device/setProperty",
  "params": {
    "iotId": "",         //设备ID
    "propertyValue": "",  //比较的属性值
    "propertyName": ""    //属性唯一标识符（产品下唯一），TSL模型中properties的identifier
  }
}
```

- 设备服务

  满足条件时执行设备服务，设备服务需要全部属性都设置才可以创建。

```
{
  "uri": "action/device/invokeService",
  "params": {
    "iotId": "",     //设备ID
    "serviceName": "",     //服务名称，TSL模型中services的identifier
    "serviceArgs": {}     //服务参数集，key为TSL模型中services的inputData某一项的identifier，value为inputData某一项的值
  }
}
```

- 场景

  满足条件时执行场景。

```
{
  "uri": "action/scene/trigger",
  "params": {
    "sceneId": ""     //场景Id
  }
}
```

- 发送通知

  满足条件时向当前用户发送通知。

```
{
  "uri": "action/mq/send",
  "params": {
    "msgTag": "IlopBusiness_CustomMsg",         //通知消息的一个固定规则tag
    "customData": {
      "message": ""             //通知内容，最多60个字符
    }
  }
}
```

创建自动化（示例一）

以设备开关作为condition，发送通知作为action为例，介绍自动化的开发流程如下。

1. 获取设备的属性。

   获取设备属性的接口为获取设备上支持TCA配置的功能属性列表，示例入参如下。

```
"params" : {
    "iotId" : "dhS41YnxxxxYx000101",    //设备ID
    "flowType" : 1    // 1代表condition，2 代表action，云端会根据flowType过滤掉不支持的属性
、服务
  }
```

选择主灯开关属性作为condition（设置属性值为1），对应的返回结果如下。其中，propertyName是选择属性的identifier。通知的内容设置为“演示自动化创建”。

```
"data" : {
    "abilityDsl" : {   //abilityDsl是设备的TSL数据
      "properties" : [   //properties是设备TSL中的属性列表
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "LightSwitch",
          "dataType" : {
            "specs" : {   //specs是可选的值和值的名称
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"  //type是属性值的类型
          },
          "name" : "主灯开关"
        }
      ],
      "services" : [
        {
          "method" : "thing.service.property.set",
          "callType" : "async",
          "desc" : "属性设置",
          "outputData" : [
           ],
          "identifier" : "set",
          "required" : true,
          "inputData" : [
            {
              "dataType" : {
                "specs" : {
                  "0" : "关闭",
                  "1" : "开启"
                },
                "type" : "bool"
              },
              "name" : "主灯开关",
              "identifier" : "LightSwitch"
            }
          ],
          "name" : "set"
        },
        {
          "method" : "thing.service.property.get",
          "callType" : "async",
          "desc" : "属性获取",
```

```
      "outputData" : [
        {
          "dataType" : {
            "specs" : {
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"
          },
          "name" : "主灯开关",
          "identifier" : "LightSwitch"
        }
      ],
      "identifier" : "get",
      "required" : true,
      "inputData" : [
        "LightSwitch"
      ],
      "name" : "get"
    }
  ],
  "schema" : "https:\/\/iotx-tsl.oss-ap-southeast-1.aliyuncs.com\/schema.json",
  "profile" : {
    "productKey" : "a1xxxxG"
  },
  "events" : [
    {
      "method" : "thing.event.property.post",
      "outputData" : [
        {
          "dataType" : {
            "specs" : {
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"
          },
          "name" : "主灯开关",
          "identifier" : "LightSwitch"
        }
      ],
      "identifier" : "post",
      "type" : "info",
      "required" : true,
      "name" : "post",
      "desc" : "属性上报"
    },
    {
      "type" : "error",
      "required" : true,
      "method" : "thing.event.Error.post",
      "identifier" : "Error",
      "name" : "故障上报",
      "outputData" : [
```

```
            {
              "dataType" : {
                "specs" : {
                  "0" : "正常"
                },
                "type" : "enum"
              },
              "name" : "故障代码",
              "identifier" : "ErrorCode"
            }
          ]
        }
      ]
    },
    "simplifyAbilityDTOs" : [
      {
        "type" : 1,    // simplifyAbilityDTOs属性列表的值，1表示属性，2表示服务，3表示事件
        "identifier" : "LightSwitch",   //propertyName是设备属性的identifier，此处设置的值
为LightSwitch
        "categoryType" : "Light",
        "name" : "主灯开关"
      }
    ]
  }
```

2. 创建自动化。

   创建自动化的接口为创建场景， 示例如下。

```
"params" : {
    "enable" : true,
    "icon" : "http://gaic.alicdn.com/ztms/cloud-intelligence-icons/icons/scene_img_xxxx
_1.png",
    "mode" : "all",
    "catalogId" : "1",
    "caConditions" : [
      {
        "uri" : "condition/device/property",
        "params" : {
          "propertyName" : "LightSwitch",
          "productKey" : "a1xxxx7G",
          "compareValue" : 1,
          "compareType" : "==",
          "deviceName" : "VDxxxxx15RH"
        }
      }
    ],
    "sceneType" : "CA",
    "actions" : [
      {
        "uri" : "action/mq/send",
        "params" : {
          "customData" : {
            "message" : "演示自动化创建"    //设置通知的内容
          },
          "msgTag" : "IlopBusiness_CustomMsg"
        }
      }
    ],
    "name" : "演示自动化创建",
    "iconColor" : "#A86AFB"
  }
```

场景自动化（示例二）

下面以时间点（中国内地地区周一~周五13:35 重复执行）作为condition，设备服务作为action为例，介绍自动化的开发流程如下。

1. 设置时间点。

```
{
    "uri" : "condition/timer",
    "params" : {
        "timezoneID" : "Asia/Shanghai",
        "cron" : "35 13 * * 1,2,3,4,5",
        "cronType" : "linux"
       }
}
```

2. 获取设备的属性。

   获取设备属性的接口为获取设备上支持TCA配置的功能属性列表，示例入参如下。

```
"params" : {
    "iotId" : "z2v7EHHxxxxoWs000100",
    "flowType" : 2        // 2 代表action，云端会根据flowType过滤掉不支持的属性、服务
}
```

对应的返回结果如下。

```
"data" : {
    "abilityDsl" : {    //abilityDsl是设备的TSL数据
      "properties" : [     //properties是设备TSL中的属性列表
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "LightSwitch",
          "dataType" : {
            "specs" : {    //specs是可选的值和值的名称
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"    //type是属性值的类型
          },
          "name" : "主灯开关"
        },
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "WIFI_Band",
          "dataType" : {
            "specs" : {
              "length" : "255"
            },
            "type" : "text"
          },
          "name" : "频段"
        },
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "WiFI_RSSI",
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "-127",
              "max" : "-1",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信号强度"
        },
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "WIFI_AP_BSSID",
```

```
          "dataType" : {
            "specs" : {
              "length" : "255"
            },
            "type" : "text"
          },
          "name" : "热点BSSID"
        },
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "WIFI_Channel",
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "1",
              "max" : "255",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信道"
        },
        {
          "required" : true,
          "accessMode" : "rw",
          "identifier" : "WiFI_SNR",
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "-127",
              "max" : "127",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信噪比"
        }
      ],
      "services" : [
        {
          "method" : "thing.service.property.set",
          "callType" : "async",
          "desc" : "属性设置",
          "outputData" : [
          ],
          "identifier" : "set",
          "required" : true,
          "inputData" : [
            {
              "dataType" : {
                "specs" : {
                  "0" : "关闭",
                  "1" : "开启"
```

```
      },
      "type" : "bool"
    },
    "name" : "主灯开关",
    "identifier" : "LightSwitch"
  },
  {
    "dataType" : {
      "specs" : {
        "length" : "255"
      },
      "type" : "text"
    },
    "name" : "频段",
    "identifier" : "WIFI_Band"
  },
  {
    "dataType" : {
      "specs" : {
        "unitName" : "无",
        "min" : "-127",
        "max" : "-1",
        "step" : "1"
      },
      "type" : "int"
    },
    "name" : "信号强度",
    "identifier" : "WiFI_RSSI"
  },
  {
    "dataType" : {
      "specs" : {
        "length" : "255"
      },
      "type" : "text"
    },
    "name" : "热点BSSID",
    "identifier" : "WIFI_AP_BSSID"
  },
  {
    "dataType" : {
      "specs" : {
        "unitName" : "无",
        "min" : "1",
        "max" : "255",
        "step" : "1"
      },
      "type" : "int"
    },
    "name" : "信道",
    "identifier" : "WIFI_Channel"
  },
  {
    "dataType" : {
```

```
            "specs" : {
              "unitName" : "无",
              "min" : "-127",
              "max" : "127",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信噪比",
          "identifier" : "WiFI_SNR"
        }
      ],
      "name" : "set"
    },
    {
      "method" : "thing.service.property.get",
      "callType" : "async",
      "desc" : "属性获取",
      "outputData" : [
        {
          "dataType" : {
            "specs" : {
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"
          },
          "name" : "主灯开关",
          "identifier" : "LightSwitch"
        },
        {
          "dataType" : {
            "specs" : {
              "length" : "255"
            },
            "type" : "text"
          },
          "name" : "频段",
          "identifier" : "WIFI_Band"
        },
        {
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "-127",
              "max" : "-1",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信号强度",
          "identifier" : "WiFI_RSSI"
        },
        {
          "dataType" : {
```

```
            "specs" : {
              "length" : "255"
            },
            "type" : "text"
          },
          "name" : "热点BSSID",
          "identifier" : "WIFI_AP_BSSID"
        },
        {
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "1",
              "max" : "255",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信道",
          "identifier" : "WIFI_Channel"
        },
        {
          "dataType" : {
            "specs" : {
              "unitName" : "无",
              "min" : "-127",
              "max" : "127",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "信噪比",
          "identifier" : "WiFI_SNR"
        }
      ],
      "identifier" : "get",
      "required" : true,
      "inputData" : [
        "LightSwitch",
        "WIFI_Band",
        "WiFI_RSSI",
        "WIFI_AP_BSSID",
        "WIFI_Channel",
        "WiFI_SNR"
      ],
      "name" : "get"
    },
    {
      "method" : "thing.service.ToggleLightSwitch",
      "callType" : "async",
      "outputData" : [
      ],
      "identifier" : "ToggleLightSwitch",
      "required" : false,
```

```
      "inputData" : [
        {
          "dataType" : {
            "specs" : {
              "min" : "1",
              "max" : "100",
              "unit" : "count",
              "step" : "1"
            },
            "type" : "int"
          },
          "name" : "测试1",
          "identifier" : "test1"
        },
        {
          "dataType" : {
            "specs" : {
              "min" : "1",
              "max" : "10",
              "unit" : "mm\/s",
              "step" : "0.1"
            },
            "type" : "float"
          },
          "name" : "测试2",
          "identifier" : "test2"
        }
      ],
      "name" : "翻转主灯开关"
    },
    {
      "method" : "thing.service.SetLightSwitchTimer",
      "callType" : "async",
      "outputData" : [
      ],
      "identifier" : "SetLightSwitchTimer",
      "required" : false,
      "inputData" : [
        {
          "dataType" : {
            "specs" : {
              "unitName" : "分",
              "min" : "0",
              "max" : "1440",
              "unit" : "min",
              "step" : "0.01"
            },
            "type" : "double"
          },
          "name" : "计时器",
          "identifier" : "Timer"
        },
        {
          "dataType" : {
```

```
            "specs" : {
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"
          },
          "name" : "主灯开关",
          "identifier" : "LightSwitch"
        }
      ],
      "name" : "设置主灯开关倒计时"
    },
    {
      "method" : "thing.service.ControlDevice",
      "callType" : "sync",
      "outputData" : [
        {
          "dataType" : {
            "specs" : {
              "min" : "0",
              "max" : "99999",
              "unitName" : "无"
            },
            "type" : "int"
          },
          "name" : "返回码",
          "identifier" : "Code"
        },
        {
          "dataType" : {
            "specs" : {
              "length" : "128"
            },
            "type" : "text"
          },
          "name" : "返回消息",
          "identifier" : "Message"
        }
      ],
      "identifier" : "ControlDevice",
      "required" : false,
      "inputData" : [
        {
          "dataType" : {
            "specs" : {
              "length" : "64"
            },
            "type" : "text"
          },
          "name" : "主机ID",
          "identifier" : "HostId"
        },
        {
          "dataType" : {
```

```
            "specs" : {
              "length" : "64"
            },
            "type" : "text"
          },
          "name" : "围栏ID",
          "identifier" : "EfenceId"
        },
        {
          "dataType" : {
            "specs" : {
              "length" : "64"
            },
            "type" : "text"
          },
          "name" : "动作",
          "identifier" : "Action"
        }
      ],
      "name" : "控制设备"
    }
  ],
  "schema" : "https:\/\/iotx-tsl.oss-ap-southeast-1.aliyuncs.com\/schema.json",
  "profile" : {
    "productKey" : "a1xxxxxska"
  },
  "events" : [
    {
      "method" : "thing.event.property.post",
      "outputData" : [
        {
          "dataType" : {
            "specs" : {
              "0" : "关闭",
              "1" : "开启"
            },
            "type" : "bool"
          },
          "name" : "主灯开关",
          "identifier" : "LightSwitch"
        },
        {
          "dataType" : {
            "specs" : {
              "length" : "255"
            },
            "type" : "text"
          },
          "name" : "频段",
          "identifier" : "WIFI_Band"
        },
        {
          "dataType" : {
            "specs" : {
```

```
            "unitName" : "无",
            "min" : "-127",
            "max" : "-1",
            "step" : "1"
          },
          "type" : "int"
        },
        "name" : "信号强度",
        "identifier" : "WiFI_RSSI"
      },
      {
        "dataType" : {
          "specs" : {
            "length" : "255"
          },
          "type" : "text"
        },
        "name" : "热点BSSID",
        "identifier" : "WIFI_AP_BSSID"
      },
      {
        "dataType" : {
          "specs" : {
            "unitName" : "无",
            "min" : "1",
            "max" : "255",
            "step" : "1"
          },
          "type" : "int"
        },
        "name" : "信道",
        "identifier" : "WIFI_Channel"
      },
      {
        "dataType" : {
          "specs" : {
            "unitName" : "无",
            "min" : "-127",
            "max" : "127",
            "step" : "1"
          },
          "type" : "int"
        },
        "name" : "信噪比",
        "identifier" : "WiFI_SNR"
      }
    ],
    "identifier" : "post",
    "type" : "info",
    "required" : true,
    "name" : "post",
    "desc" : "属性上报"
  },
  {
    "type" : "error",
```

```
        "required" : true,
        "method" : "thing.event.Error.post",
        "identifier" : "Error",
        "name" : "故障上报",
        "outputData" : [
          {
            "dataType" : {
              "specs" : {
                "0" : "恢复正常"
              },
              "type" : "enum"
            },
            "name" : "故障代码",
            "identifier" : "ErrorCode"
          }
        ]
      }
    ]
  },
  "simplifyAbilityDTOs" : [
    {
      "type" : 1,
      "identifier" : "LightSwitch",
      "categoryType" : "Light",
      "name" : "主灯开关"
    },
    {
      "type" : 1,
      "identifier" : "WiFI_RSSI",
      "categoryType" : "Light",
      "name" : "信号强度"
    },
    {
      "type" : 1,
      "identifier" : "WIFI_Channel",
      "categoryType" : "Light",
      "name" : "信道"
    },
    {
      "type" : 1,
      "identifier" : "WiFI_SNR",
      "categoryType" : "Light",
      "name" : "信噪比"
    },
    {
      "type" : 2,
      "identifier" : "ToggleLightSwitch",
      "categoryType" : "Light",
      "name" : "翻转主灯开关"
    },
    {
      "type" : 2,
      "identifier" : "SetLightSwitchTimer",
      "categoryType" : "Light",
```

```
        "name" : "设置主灯开关倒计时"
      },
      {
        "type" : 2,                    //simplifyAbilityDTOs属性列表的值，1表示属性，2表示服务，3表示事件
        "identifier" : "ControlDevice",  //propertyName是设备属性的identifier，此处设置的值为ControlDevice
        "categoryType" : "Light",
        "name" : "控制设备"
      }
    ]
}
```

abilityDsl是设备的TSL数据，services是设备TSL的服务列表，services中inputData是服务需要设置的properties列表。还需要配置以下TSL信息。

```
// 以下是TSL中的一个服务
{
  "method" : "thing.service.SetLightSwitchTimer",
  "callType" : "async",
  "outputData" : [
  ],
  "identifier" : "SetLightSwitchTimer",
  "required" : false,
      "inputData" : [     //inputData是服务需要设置的properties列表
      {
          "dataType" : {
            "specs" : {
                "unitName" : "分",
                "min" : "0",
                "max" : "1440",
                "unit" : "min",
                "step" : "0.01"
            },
            "type" : "double"
        },
        "name" : "计时器",
        "identifier" : "Timer"
      },
      {
          "dataType" : {
            "specs" : {
                "0" : "关闭",
                "1" : "开启"
            },
             "type" : "bool"
         },
         "name" : "主灯开关",
         "identifier" : "LightSwitch"
       }
  ],
  "name" : "设置主灯开关倒计时"
 }
```

返回结果如下所示。

```
{
  "method" : "thing.service.SetLightSwitchTimer",
  "callType" : "async",
  "outputData" : [
  ],
  "identifier" : "SetLightSwitchTimer",
  "required" : false,
      "inputData" : [
      {
          "dataType" : {
            "specs" : {
                "unitName" : "分",
                "min" : "0",
                "max" : "1440",
                "unit" : "min",
                "step" : "0.01"
            },
            "type" : "double"
        },
        "name" : "计时器",
        "identifier" : "Timer"
      },
      {
          "dataType" : {
            "specs" : {
                "0" : "关闭",
                "1" : "开启"
            },
             "type" : "bool"
         },
         "name" : "主灯开关",
         "identifier" : "LightSwitch"
       }
  ],
  "name" : "设置主灯开关倒计时"  //服务名称为"设置主灯开关倒计时"作为action
 }
```

选择服务作为action，还需要给services包含的所有属性“计时器”、“主灯开关”都设置值后才可以生效，计时器设置值为0.08，主灯开关设置值为1，代码如下。

```
{
    "uri" : "action/device/invokeService",
    "params" : {
        "iotId" : "z2v7EHHfExxxxWs000100",
        "serviceName" : "SetLightSwitchTimer",        // TSL中服务的SetLightSwitchTimer
        "serviceArgs" : {                // {属性identifier: 属性值}
          "Timer" : 0.08,   //计时器设置值为0.08
          "LightSwitch" : 1   //主灯开关设置值为1
        }
    }
}
```

3. 创建自动化。

   创建自动化的接口为创建场景，示例代码如下。

```
"params" : {
    "enable" : true,
    "icon" : "http://gaic.alicdn.com/ztms/cloud-intelligence-icons/icons/scene_xxxx_15.
png",
    "mode" : "all",
    "catalogId" : "1",
    "caConditions" : [
      {
        "uri" : "condition/timer",
        "params" : {
          "timezoneID" : "Asia/Shanghai",
          "cron" : "35 13 * * 1,2,3,4,5",
          "cronType" : "linux"
        }
      }
    ],
    "sceneType" : "CA",
    "actions" : [
      {
        "uri" : "action/device/invokeService",
        "params" : {
          "iotId" : "z2v7EHHfxxxxoWs000100",
          "serviceName" : "SetLxxxxxTimer",    // TSL中服务的SetLightSwitchTimer
          "serviceArgs" : {
            "Timer" : 0.08,
            "LightSwitch" : 1
          }
        }
      }
    ],
    "name" : "小庆灯我的小灯泡xiaoqingdeng-设置主灯开关倒计时",
    "iconColor" : "#FF454F"
  }
```

创建手动触发场景

自动化是用户创建后会按照响应的触发条件自动触发动作的任务；而手动触发场景则需要用户自己手动执行才会触发执行任务。因此，手动触发场景只有一系列的Action。

- 创建手动触发场景

  下面以执行风扇电源开关（打开）为例，介绍如何创建手动触发场景。

  使用创建场景接口来创建场景，示例代码如下。

```
"params":{
   "enable":true,
   "icon":"http://gaic.alicdn.com/ztms/cloud-intelligence-icons/icons/scene_img_xxxx_13.p
ng",
   "iconColor":"#738DE1",
   "name":"家家风扇01-电源开关 - 开启",
   "actions":[
       {
           "uri":"action/device/setProperty",
           "params":{
               "propertyName":"PowerSwitch",
               "iotId":"bx6lzXLxxxxLNOO000101",
               "propertyValue":1
           }
       }
   ],
   "catalogId":"0"
}
```

> **说明** 手动触发场景与自动化的创建很类似，区别在于没有caConditions，且不传sceneType mode这两个非必填参数。

- 执行手动触发场景

  执行场景的接口为执行场景，示例代码如下。

```
params:{"sceneId":"7823be10xxxx8790f36bdc629e"}
```

5.2. 小组件开发最佳实践（Android）

应用程序小组件是一个微型的应用程序视图，可以嵌入其他应用程序（例如主屏幕）中并接收定期更新。本文档介绍了如何使用App Widget provider发布Android小组件。

概述

应用程序视图在用户界面中称为组件（Widget），您可以使用小组件提供程序（App Widget provider）发布这些视图。能够容纳其他小组件的应用程序组件称为App Widget宿主（如Launcher），下图为时钟和天气的小组件示例，更多有关小组件设计规范的内容，请参见 App Widgets Overview。

为了创建小组件，您还需要先了解以下信息。

- AppWidgetProviderInfo

  描述应用程序小组件的元数据，例如小组件的布局、更新频率、指定AppWidgetProvider类等。

- AppWidgetProvider

  用来处理小组件的广播事件，当更新、启用、禁用、删除小组件时，您可以收到广播。

- View layout

以XML定义的小组件的初始布局

- 其他

  还可以为您的小组件配置活动。启动活动后，允许用户在该活动里修改小组件设置。

创建小组件

1. 在应用程序的清单文件*AndroidManifest.xml*中声明AppWidgetProvider类，示例代码如下。

```
<receiver android:name="ExampleAppWidgetProvider" >
    <intent-filter>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>
    <meta-data android:name="android.appwidget.provider"
               android:resource="@xml/example_appwidget_info" />
</receiver>
```

   代码配置说明如下。

   - receiver元素需要设置android：name属性，该属性指定App Widget使用的是AppWidgetProvider类。
   - intent-filter元素必须包含具有android：name属性的action元素。此属性指定AppWidgetProvider类接受的广播类型为ACTION_APPWIDGET_UPDATE（这是您须明确声明的唯一广播）。AppWidgetManager根据用户需要，自动将所有其他App Widget广播发送到AppWidgetProvider。
   - meta-data元素指定AppWidgetProviderInfo资源，并需要设置以下属性。
     - android：name指定元数据名称，使用android.appwidget.provider将数据标识为AppWidgetProviderInfo描述符。
     - android：resource 指定AppWidgetProviderInfo的资源位置。

2. 添加AppWidgetProviderInfo元数据。

   AppWidgetProviderInfo定义了小组件的基本配置（例如最小布局尺寸、初始布局资源、更新频率、（可选）在创建时启动的配置Activity等）。使用单个元素在XML资源中定义AppWidgetProviderInfo对象，并将其保存在项目的*res/xml*文件夹中。

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="40dp"
    android:minHeight="40dp"
    android:updatePeriodMillis="86400000"
    android:previewImage="@drawable/preview"
    android:initialLayout="@layout/example_appwidget"
    android:configure="com.example.android.ExampleAppWidgetConfigure"
    android:resizeMode="horizontal|vertical"
    android:widgetCategory="home_screen">
</appwidget-provider>
```

   代码配置说明如下。

   - minWidth和minHeight属性指定默认情况下App Widget占用的最小尺寸（为了使您App的小组件适配多种屏幕，此处的最小尺寸不得大于4 x 4单元）。更多信息请参见应用程序小组件设计指南。
   - updatePeriodMillis属性定义App Widget框架通过调用`onUpdate()`回调方法从AppWidgetProvider请求更新的频率。使用该值不能保证实际的更新会准时进行，我们不建议频繁地更新（每小时不超过一次），以便节省电池电量。

- initialLayout属性指向定义App Widget布局的布局资源。
- configure属性（可选属性）定义了用户添加App Widget时要启动的活动，以便配置App Widget属性。
- previewImage属性指定配置后的应用小组件的预览，用户在选择应用小组件时可见。如果未提供，用户看到的为您应用程序的启动器图标。
- resizeMode属性指定可以调整窗口小组件大小的规则，例如水平、垂直或双向调整大小等。
- minResizeHeight和minResizeWidth属性指定窗口小组件可以调整大小的最小高度与最小宽度（单位为dp）。

> **说明** 更多元素属性的介绍请参见AppWidgetProviderInfo。

3. 创建小组件布局。

   使用XML为您的小组件定义初始布局，并将其保存在工程的*res/layout*目录中。

   小组件布局基于RemoteViews，一个RemoteViews对象（通常就是一个小组件）可以支持以下布局类和视图组件。

   - 布局类
     - FrameLayout
     - LinearLayout
     - RelativeLayout
     - GridLayout

     > **说明** 仅支持这些类，不支持这些类的子类。

   - 视图组件
     - AnalogClock
     - Button
     - Chronometer
     - ImageButton
     - ImageView
     - ProgressBar
     - TextView
     - ViewFlipper
     - ListView
     - GridView
     - StackView
     - AdapterViewFlipper
   - 其他

     RemoteViews还支持ViewStub，这是一个不可见的零尺寸视图，可用于在运行时延迟布局资源的渲染。

   下面以设计布局类FrameLayout为例介绍，更多信息请参见App Widget设计指南。

i. 进入*res/layout/*目录，创建一个xml的布局文件（例如：appwidget_provider_layout.xml）。

ii. 添加布局类FrameLayout，相关代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="OK"
        tools:ignore="HardcodedText" />
</FrameLayout>
```

4. 添加小组件之间的距离。

   为了更好地提升用户体验，从Android 4.0及以上版本，系统会自动在小组件框架和应用小组件的边界框之间提供填充。Android 4.0以下则需要开发者自行设置（由于目前市场上Android 4.0以下机型较少，此处不作介绍，可自行查找资料）。

5. 创建AppWidgetProvider类。

   最后您还需要创建在清单中声明的ExampleAppWidgetProvider类。例如，如果您想要一个用于单击的按钮小组件，使用AppWidgetProvider实现的示例代码如下。

```
/*
 * Copyright (C) 2008 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
public class ExampleAppWidgetProvider extends AppWidgetProvider {
    public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appW
idgetIds) {
        final int N = appWidgetIds.length;
        // Perform this loop procedure for each App Widget that belongs to this provide
r
        for (int i=0; i<N; i++) {
            int appWidgetId = appWidgetIds[i];
            // Create an Intent to launch ExampleActivity
            // Intent intent = new Intent(context, ExampleActivity.class);
            Intent intent = new Intent();
            PendingIntent pendingIntent = PendingIntent.getActivity(context, 0, intent,
0);
            // Get the layout for the App Widget and attach an on-click listener
            // to the button，其中appwidget_provider_layout为创建的xml布局文件名称
            RemoteViews views = new RemoteViews(context.getPackageName(), R.layout.appw
idget_provider_layout);
            views.setOnClickPendingIntent(R.id.button, pendingIntent);
            // Tell the AppWidgetManager to perform an update on the current app widget
            appWidgetManager.updateAppWidget(appWidgetId, views);
        }
    }
}
```

代码配置说明如下。

此AppWidgetProvider仅定义 `onUpdate()` 方法，并定义启动活动的PendingIntent，使用 `setOnClickPendingIntent（int，PendingIntent）` 将其添加到小组件的按钮上。

关于AppWidgetProvider的使用说明如下。

- AppWidgetProvider类是BroadcastReceiver 的扩展，用来处理小组件的广播。 AppWidgetProvider仅接收与小组件相关的事件广播，例如何时更新，删除，启用和禁用小组件。当发生这些广播事件时，AppWidgetProvider会收到以下方法调用：
  - onUpdate(android.content.Context, android.appwidget.AppWidgetManager, int[])
  - onAppWidgetOptionsChanged(android.content.Context, android.appwidget.AppWidgetManager, int, android.os.Bundle)
  - onDeleted(android.content.Context, int[])

- onEnabled(Context)
- onDisabled(Context)
- onReceive(Context, Intent)

- AppWidgetProvider回调通过 `onUpdate()` 方法。如果您的小组件需要接受用户的交互事件（此时您需要在此回调中注册事件处理程序），并且您的小组件不需要创建临时文件或数据库，也不需要执行其他清理的工作时，您无需关心除 `onUpdate()` 以外的生命周期回调。

> **说明** 由于AppWidgetProvider是BroadcastReceiver的子类，因此不能保证您的进程在回调方法返回后仍能继续运行（有关广播生命周期的信息，请参见BroadcastReceiver）。如果您的小组件设置过程需要花费几秒钟的时间（可能是在执行Web请求时），并且您要求设置过程继续进行，请使用 `onUpdate()` 方法，通过在方法中启动服务来处理耗时操作。您可以从服务内部对小组件执行自己的更新，从而不必担心AppWidgetProvider会由于Application Not Responding（ANR）而关闭。

请求接口

小组件开发过程中常用到的请求接口如下。

- 获取设备列表（已添加到小组件）

```
path: /iotx/ilop/queryComponentProduct
version: 1.0.0
params: @{}
```

- 获取设备属性

```
path: /iotx/ilop/queryComponentProperty
version: 1.0.0
params = @{@"productKey":productKey,@"iotId":iotId,@"query":@{@"dataType":@"BOOL”, @"I18Language":@"zh-CN"}}
```

- 更新设备属性

```
 path: /iotx/ilop/updateComponentProduct
version: 1.0.0
params: 更改后的设备list
```

- 获取场景列表（已添加到小组件）

```
path: /living/appwidget/list
version: 1.0.0
params: @{}
```

- 执行场景

```
path: /scene/fire
version:1.0.1
params: @{@"sceneId":sceneId}
```

- 更新小组件场景

```
path: /living/appwidget/create
version: 1.0.0
params = @{@"sceneIds": @[]}
```

更多关于接口的说明请参见场景服务和物的模型服务。

监听设备属性更新

如果您需要开发控制设备的小组件，通过小组件实现手机对设备的查看和控制。那么您还需要了解App端物模型（属性、事件、服务）相关的知识点。下面以如何用物模型SDK监听设备属性变更事件为例，提供相关的代码示例供您参考。

下面以使用物模型SDK监听设备属性变更事件为例，提供相关的代码示例供您参考。更多内容请参见物模型SDK。

```
PanelDevice panelDevice = new PanelDevice(iotId);
panelDevice.subAllEvents(new IPanelEventCallback() {
    @Override
    public void onNotify(String s, String s1, Object o) {
        Log.d(TAG, "onNotify: " + s);
        Log.d(TAG, "onNotify: " + s1);
        Log.d(TAG, "onNotify: " + JSON.toJSONString(o));
        // 更新界面
    }
}, new IPanelCallback() {
    @Override
    public void onComplete(boolean b, Object o) {
        /* */
    }
});
panelDevice.init(this, new IPanelCallback() {
    @Override
    public void onComplete(boolean b, Object o) {
        Log.e(TAG, "panelDevice.init:" + b);
    }
});
```

5.3. 小组件开发最佳实践（iOS）

应用程序小组件是一个微型的应用程序视图，可以嵌入其他应用程序（例如主屏幕）中并接收定期更新。本文档介绍了开发iOS小组件。

创建证书

iOS中小组件（Widget）是一个独立的应用，可以看做是一个独立的App（宿主App的拓展程序），所以我们需要对Widget单独创建证书。

1. 创建宿主App证书。

   该部分操作资料很丰富，此处不做介绍，可自行查找相关资料。

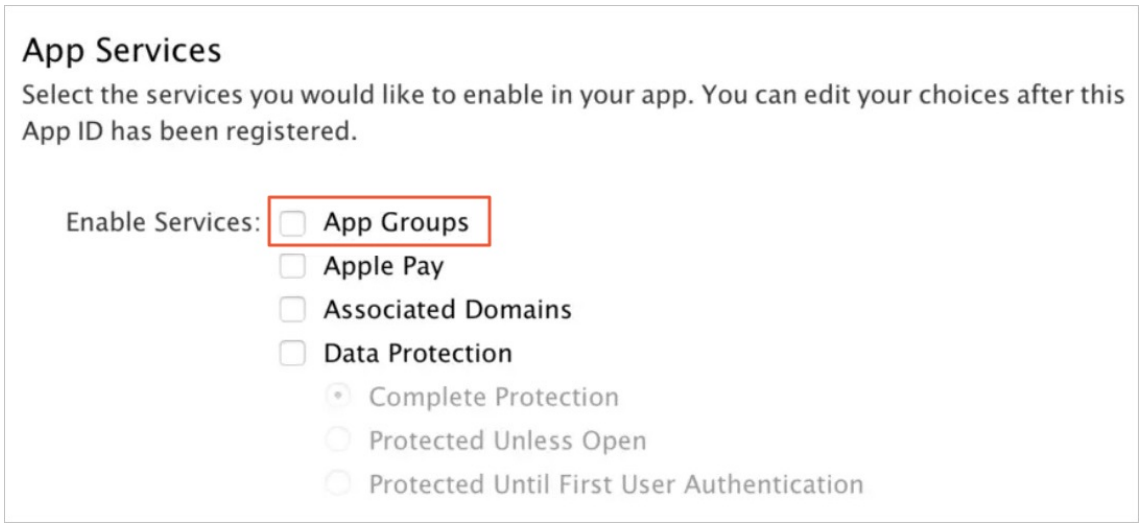
2. 创建Widget证书。

   创建Widget证书的操作与创建宿主App证书的操作类似，但需注意以下几点。

   - Widget的Bundle Id是以宿主App为基础扩展的，例如宿主App为 `com.companyName.AppName` ，则

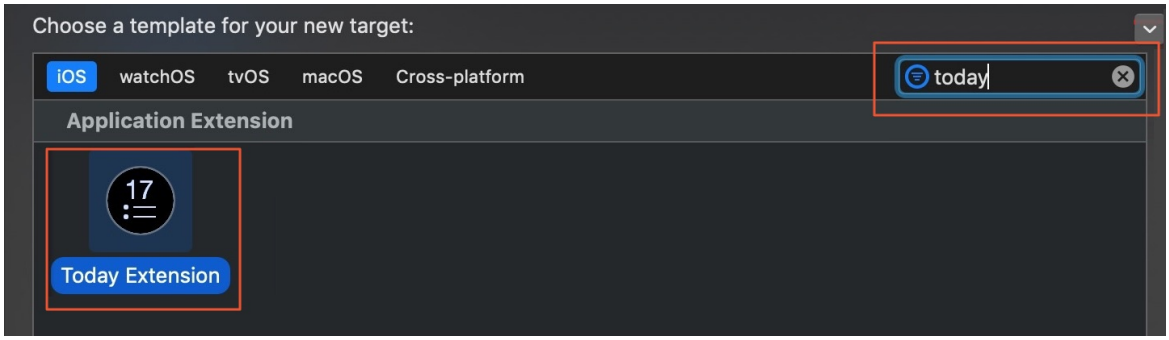
Widget的格式应该为 `com.companyName.AppName.WidgetName` 。

- 创建证书的时候，需勾选App Group配置项。

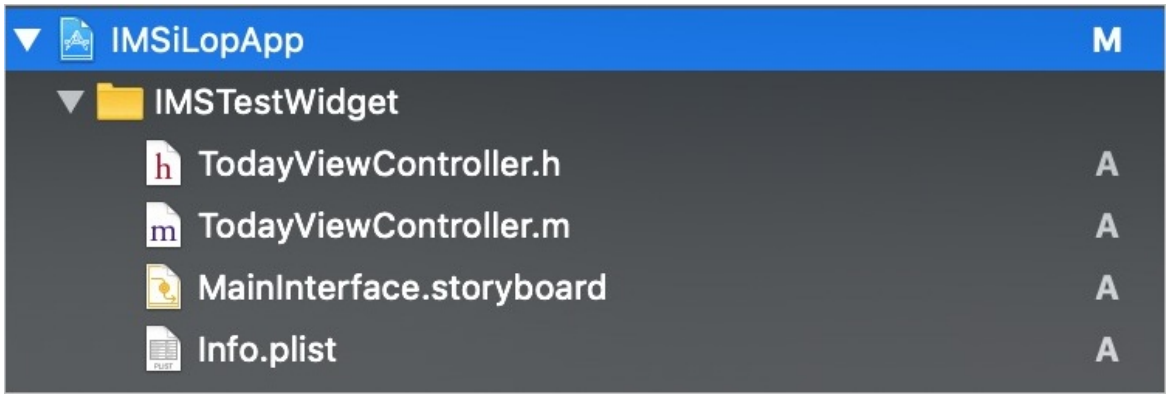


创建Widget

1. 在xcode中，选择**File > New > Target > Today Extension**，创建Today。



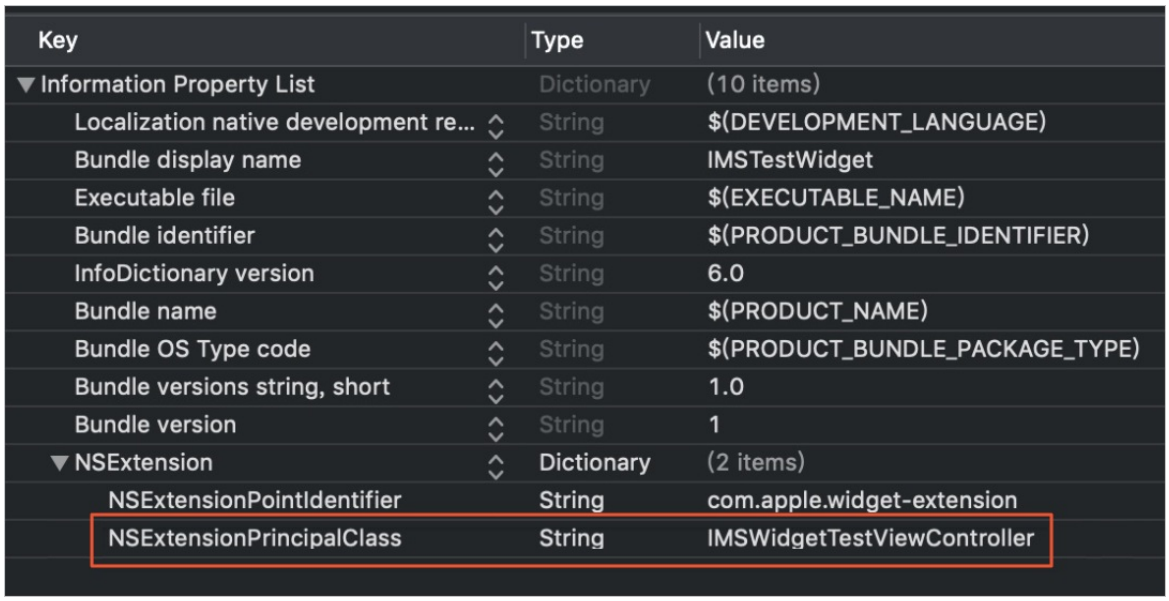
2. 查看创建后的目录结构。



3. 设置Widget工程的开发方式。

   工程默认为storyboard开发方式，如果想使用纯代码方式，则需要进行以下操作。

   在**TodayWidget > Info.plist > Extension**中，删除NSExtensionMainStoryboard选项，增加NSExtensionPrincipalClass选项，value为类的名字IMSWidgetTestViewController，如下图所示。



Key	Type	Value
Information Property List	Dictionary	(10 items)
Localization native development re...	String	$(DEVELOPMENT_LANGUAGE)
Bundle display name	String	IMSTestWidget
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
NSExtension	Dictionary	(2 items)
NSExtensionPointIdentifier	String	com.apple.widget-extension
NSExtensionPrincipalClass	String	IMSWidgetTestViewController

4. 设置Widget的展开与折叠效果，示例如下。

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    if (/*折叠展开判断 */) {
        self.extensionContext.widgetLargestAvailableDisplayMode = NCWidgetDisplayModeExpanded;
    } else {
        self.extensionContext.widgetLargestAvailableDisplayMode = NCWidgetDisplayModeCompact;
    }
}
- (void)widgetActiveDisplayModeDidChange:(NCWidgetDisplayMode)activeDisplayMode withMaximumSize:(CGSize)maxSize {
    switch (activeDisplayMode) {
        case NCWidgetDisplayModeCompact: {
            self.preferredContentSize = maxSize;
            break;
        }
        case NCWidgetDisplayModeExpanded: {
            self.preferredContentSize = CGSizeMake(self.view.bounds.size.width, 210);
            break;
        }
        default:
            break;
    }
}
```

说明

- 展开的高度可以自行设置，不超过系统最大值即可。
- 系统不支持折叠高度的修改。

5. 刷新数据，建议使用系统提供的方法，示例如下。

```
- (void)widgetPerformUpdateWithCompletionHandler:(void (^)(NCUpdateResult))completionHandler {
    completionHandler(NCUpdateResultNewData);
}
```

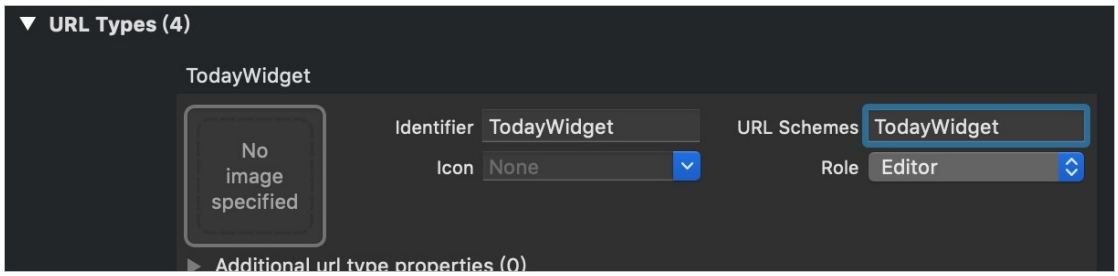
说明 此处刷新有可能执行失败，这是目前Apple存量的问题，可通过延迟来解决。详细请参见延时的原因和临时解决方案。

6. 配置小组件与宿主App的跳转功能。

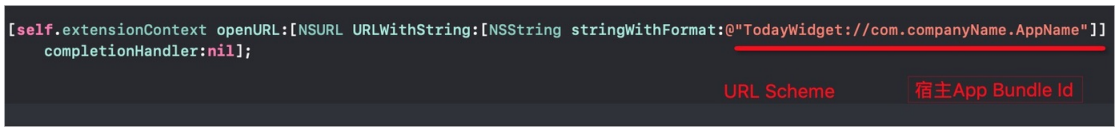
   Extension和宿主App是两个完全独立的进程，它们之间不能直接通信（即无法通过单击应用内部按钮跳转到指定页面）。为了实现Widget调起宿主App，这里通过openURL的方式来启动宿主App。

i. 在宿主App里选择**Targets > MCWidgetDemo > Info > Url Types**，添加URL Schemes。

下图为设置示例，设置URL Schemes为TodayWidget。



ii. 配置代码跳转地址（openURL）。完整地址为：”URL Schemes” + “://” + “宿主App Bundle Id”，如下图所示。

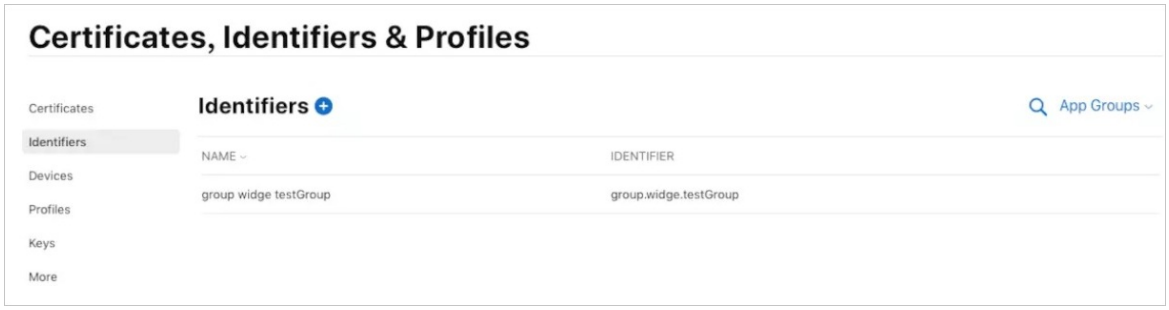


设置Widget和宿主App交互通信

因为Widget的独立性，宿主App要与Widget之间相互通信，需要通过App Group来实现。

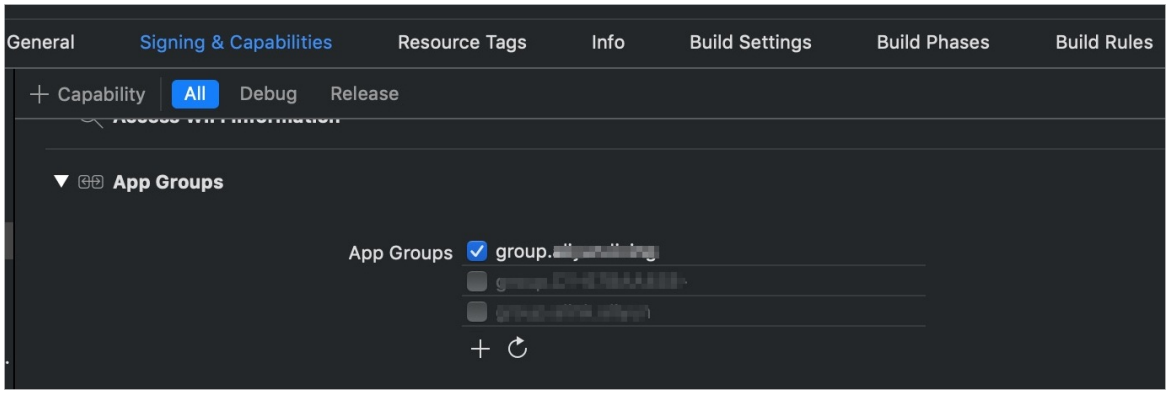
1. 创建App Gruop。

   前往开发者网站注册一个App Group，填入名字和id，并根据界面提示操作，即可得到下图类似的App Group。



2. 在**Target > Signing & Capabilities > App Group**下，配置App Group。

   在宿主App和扩展程序（Widget）的App Group中，分别设置group名称。需确保宿主App和Widget的groupName相同，并且与在开发者网站注册的App Groups保持一致。



3. 配置Widget和宿主App之间交互通信。

   使用NSUserDefaults或者NSFileManager方式都可以实现Widget和宿主App之间交互通信。此处介绍如

何使用NSUserDefaults方式实现交互通信。

- 存数据

```
NSUserDefaults *sharedData = [[NSUserDefaults alloc]
    initWithSuiteName:@"group.widge.testGroup"];
[sharedData setValue:@"我叫小组件" forKey:@"name"];
[sharedData synchronize];
```

存数据

- 取数据

```
NSUserDefaults *sharedData = [[NSUserDefaults alloc]
    initWithSuiteName:@"group.widge.testGroup"];
NSString *name = [sharedData objectForKey:@"name"];
```

取数据

生活物联网平台SDK使用指导

下面主要介绍TodayExtension的开发过程，其余Widget开发请参照Apple官方文档自行完成。

1. 引入SDK。

   i. 设置Profile。

   iOS推荐使用Cocoapods引入，分别对宿主App Target和Widget Target引入SDK。因为Widget是独立的应用，所以两个Target都需要各自引入编译所需的SDK，多个小组件，就配置多份Profile。配置示例如下。

```
target “WidgetTargetName1” do
    pod 'IMSApiClient', '1.6.0'
    pod 'IMSAuthentication', '1.4.1'
end
target “WidgetTargetName2” do
    pod 'IMSApiClient', '1.6.0'
    pod 'IMSAuthentication', '1.4.1'
end
```

   ii. 查看小组件开发必备SDK列表。

```
小组件开发必备SDK列表
【1】通用请求SDK
    pod 'IMSApiClient', '1.6.0'
    pod 'IMSAuthentication', '1.4.1'
【2】设备小组件相关SDK
    # 物
    pod 'IMSThingCapability', '1.7.5'
    # 长连接
    pod 'IMSMobileChannel', '1.6.7'
```

   iii. 执行pod update，并编译工程。

   编译成功后，选择Widget的target，运行小组件工程。

> **说明** 因为Widget的独立，安全图片也需要导入一份到Widget的Target下，否则会报错。

2. 初始化宿主App配置。

   i. 初始化宿主App的IMSAuthentication，示例代码如下。

   ```
   // 设置需要更新的Credential至AppGroup中
   [[IMSCredentialManager sharedManager] addCredentialStoreWithAppGroupName:AppGroupName];
   ```

   ii. 把ApiClient的信息，写入对应的AppGroup共享区域中。

   ```
   // 宿主App初始化IMSApiClient完成后，把ApiClient的信息，写入到对应的AppGroup共享区域中
   [[IMSConfiguration sharedInstance] storeConfigToAppGroup:AppGroupName];
   ```

3. 配置TodayExtension，配置示例代码如下。

```
+ (void)initialize {
    // 初始化APIClient
        [IMSConfiguration initWithAppGroupName:AppGroupName];
        // 初始化身份认证
        [IMSCredentialManager initWithAppGroupName:AppGroupName];
    // 注册RequestClient的代理
    IMSIoTAuthentication *iotAuthDelegate = [[IMSIoTAuthentication alloc] initWithCredentialManager:IMSCredentialManager.sharedManager];
    [IMSRequestClient registerDelegate:iotAuthDelegate forAuthenticationType:IMSAuthenticationTypeIoT];
}
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    // 根据AppGroup共享区域存储的信息
    // 防止出现未打开宿主App、已经初始化Extension的APIClient
    [[IMSConfiguration sharedInstance] synchronizeConfigFromAppGroup];
    // 从UserDefaults更新Credential
    [[IMSCredentialManager sharedManager] synchronizeCredentialFromAppGroup];
}
```

4. 调用接口，示例代码如下。

```
 IMSIoTRequestBuilder *builder = [[IMSIoTRequestBuilder alloc] initWithPath:@"/uc/path/
xxxx"
                                                                    apiVersion:@"1.0.0"
                                                                        params:@{}];
    [builder setScheme:@"https"];
    IMSRequest *request = [[builder setAuthenticationType:IMSAuthenticationTypeIoT] bui
ld];
    __weak typeof(self) weakSelf = self;
    [IMSRequestClient asyncSendRequest:request responseHandler:^(NSError * _Nullable er
ror, IMSResponse * _Nullable response) {
          if (response.code == 401) {
                [self loginOut];
            }
            if (error) {
                NSLog(@"request error = %@",error);
            } else {
                NSLog(@"request success");
            }
        }];
    }];
```

5. 实时判断宿主App登录状态，示例代码如下。

```
- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    // 根据AppGroup共享区域存储的信息、配置Host、环境、语言、安全图片
    // 防止出现未打开宿主App、已经初始化Extension的APIClient
        [[IMSConfiguration sharedInstance] synchronizeConfigFromAppGroup];
        // 从UserDefaults更新Credential
        [[IMSCredentialManager sharedManager] synchronizeCredentialFromAppGroup];
    // 通过Credential是否存在来判断登录态
    if ([IMSCredentialManager sharedManager].credential) {
        // 已登录
    } else {
        // 未登录
    }
}
```

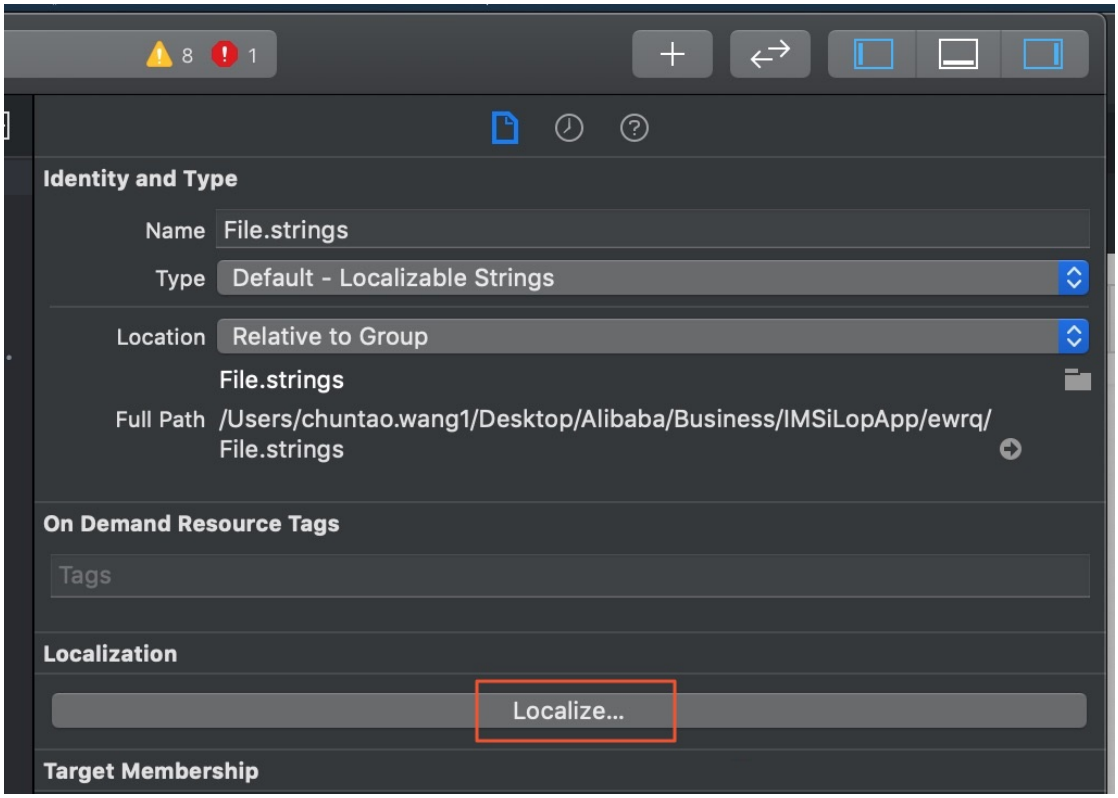
6. 配置小组件显示名称的多语言。
   i. 使用宿主App的 `[IMSConfiguration sharedInstance].language` 更新语言信息，信息需要重新保存到group中。

   ```
   // 把ApiClient的信息，写入到对应的AppGroup共享区域中
   [[IMSConfiguration sharedInstance] storeConfigToAppGroup:AppGroupName];
   ```

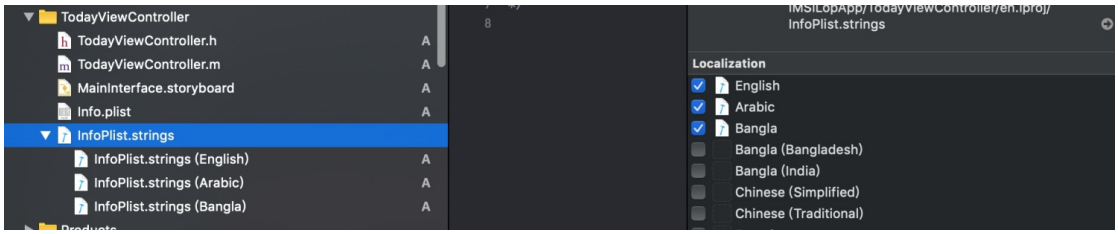
ii. 设置多语言。

选中TodayExtension的Target，选择 **New > File > String File**，新建 strings 文件（名称请使用InfoPlist）。

创建完成后，选中InfoPlist.strings文件，单击**Localize**，添加多语言。

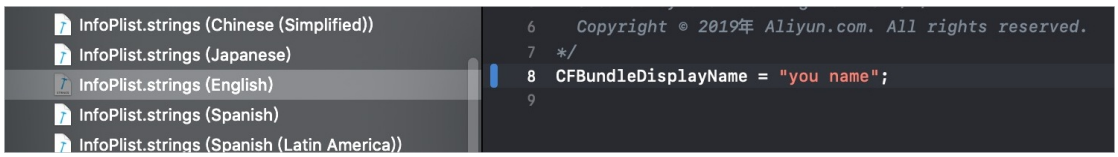


多语言设置后的界面如下。



iii. 更改小组件的显示名称。

选中某种语言，修改该语言下小组件的显示名称（小组件名字是系统语言控制的，这个不随App更改）。



设备小组件&场景小组件接口文档和调用过程

设备小组件和场景小组件在开发过程中使用的接口文档（参见场景服务）和调用示例如下。

- 宿主App相关的接口

- 场景小组件

```
【1】获取已经被添加到小组件的场景list
   path：/living/appwidget/list
   version：1.0.0
   params：@{}
【2】全量场景查询
   path：/living/scene/query
   version：1.0.1
   params = @{@"catalogId": @"0",
                           @"pageNo": @(pageNo),
                           @"pageSize": @(pageSize)
                           }
【3】更新场景小组件
 path：/living/appwidget/create
 version：1.0.0
 params = @{@"sceneIds": @[]}
```

- 设备小组件

```
【1】获取已经被加到小组件的设备list
   path：/iotx/ilop/queryComponentProduct
   version：1.0.0
   params：@{}
【2】获取设备的属性列表（目前属性多语言需要入参时传递）
   path：/iotx/ilop/queryComponentProperty
   version：1.0.0
   params = @{@"productKey":productKey,
                           @"iotId":iotId,
                           @"query":@{@"dataType":@"BOOL”, @"I18Language":@"zh-CN"}
                           }
【3】小组件列表更新
   path：/iotx/ilop/updateComponentProduct
   version：1.0.0
   params：更改后的设备list
```

- TodayExtension相关接口
  - 场景小组件

```
【1】获取已经被添加到小组件的场景list
   path：/living/appwidget/list
   version：1.0.0
   params：@{}
【2】执行场景
   path：/scene/fire
   version:1.0.1
   params：@{@"sceneId":sceneId}
```

- 设备小组件

```
【1】获取已经被加到小组件的设备list
    path：/iotx/ilop/queryComponentProduct
    version：1.0.0
    params：@{}
【2】设备小组件，有本地通信和云端通信逻辑，需要集成宿主APP中的长连接绑定 & 订阅，监听长连接正常连接
【3】设备状态变更，需要自行定位/thing/properties 和  /thing/status 的topic，监听状态变更，刷新UI
【4】选中设备，指定ThingShell设置设备属性，通过物的模型，变更属性
【5】如果订阅过Topic，设置【4】成功后，也会受到云端的状态变更通知
```

- 设备小组件核心参考代码

```
【1】长连接绑定 & 订阅（相关SDK参见长连接通道SDK）
  IMSConfiguration * imsconfig = [IMSConfiguration sharedInstance];
  LKAEConnectConfig * config = [LKAEConnectConfig new];
  config.appKey = imsconfig.appKey;
  config.authCode = imsconfig.authCode;
  // 指定长连接服务器地址。 （默认不填，SDK会使用默认的地址及端口。默认为国内华东节点。不要带 "协议://"，如果置为空，底层通道会使用默认的地址）
  config.server = @""
// 开启动态选择Host功能。 （默认 NO，海外环境请设置为 YES。此功能前提为 config.server 不特殊指定。）
  config.autoSelectChannelHost = NO;
  [[LKAppExpress sharedInstance]startConnect:config connectListener:self];// self 需要实现 LKAppExpConnectListener 接口
}
【2】注册下行Listener
#pragma mark - 注册下行Listener
static NSString *const IMSiLopExtensionDidReceiveUpdateAttributeSuccess = @"LAMPPANEL_DIDRECEIVE_UPDATE_ATTRIBUTE_SUCCESS";
static NSString *const IMSiLopExtensionDidReceiveUpdateDeviceStateSuccess = @"LAMPPANEL_DIDRECEIVE_UPDATE_DEVICE_STATE_SUCCESS";
@class TodayViewController;
@interface IMSWidgetDeviceListener : NSObject <LKAppExpDownListener>
@end
@implementation IMSWidgetDeviceListener
- (void)onDownstream:(NSString * _Nonnull)topic data:(id  _Nullable)data {
    IMSAppExtensionLogVerbose(@"小组件 onDownstream topic : %@", topic);
    IMSAppExtensionLogVerbose(@"小组件 onDownstream data : %@", data);
    NSDictionary * replyDict = nil;
    if ([data isKindOfClass:[NSString class]]) {
        NSData * replyData = [data dataUsingEncoding:NSUTF8StringEncoding];
        replyDict = [NSJSONSerialization JSONObjectWithData:replyData options:NSJSONReadingMutableLeaves error:nil];
    } else if ([data isKindOfClass:[NSDictionary class]]) {
        replyDict = data;
        //这里添加云端处理!
        if (data) {
            if ([topic isEqualToString:@"/thing/properties"]) {
                [[NSNotificationCenter defaultCenter] postNotificationName:IMSiLopExtensionDidReceiveUpdateAttributeSuccess object:self userInfo:data];
            }
```

```
            if ([topic isEqualToString:@"/thing/status"]) {
                [[NSNotificationCenter defaultCenter] postNotificationName:IMSiLopExten
sionDidReceiveUpdateDeviceStateSuccess object:self userInfo:data];
            }
        }
    }
    if (replyDict == nil) {
        return;
    }
}
- (BOOL)shouldHandle:(NSString * _Nonnull)topic {
    // 需要设什么topic，返回什么topic
    if ([topic isEqualToString:@"/thing/properties"] || [topic isEqualToString:@"/thing
/status"]) {
        return YES;
    }
    return NO;
}
@end
```

【3】增加代理监听、增加属性

```
@interface IMSWidgetDeviceController () < LKAppExpConnectListener>
// 本地控制
@property (nonatomic, strong) IMSWidgetDeviceListener *imsWidgetDeviceListener;
```

【4】在ViewDidLoad 中增加监听

```
- (void)viewDidLoad {
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
    // 监听云端设备属性变更
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(dididRece
iveUpdateAttributeNoti:) name:IMSiLopExtensionDidReceiveUpdateAttributeSuccess object:n
il];
    // 监听云端设备状态变更
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(dididRece
iveUpdateDeviceStateNoti:) name:IMSiLopExtensionDidReceiveUpdateDeviceStateSuccess obje
ct:nil];
}
```

【5】自行处理下行通知

```
// 云端属性数据下发
- (void)dididReceiveUpdateAttributeNoti:(NSNotification *)info {
}
// 云端状态数据下发
- (void)dididReceiveUpdateDeviceStateNoti:(NSNotification *)info {
}
```

【6】更改属性方法

```
IMSThing *thingShell = [kIMSThingManager buildThing:iotId];
[[thingShell getThingActions] setProperties:@{propertyIdentifierName:value}
                                responseHandler:^(IMSThingActionsResponse * _Nullable
response) {
                                    if (response.success) {
                                        // 成功
                                    } else {
                                        // 失败
                                    }
}];
```

```
【7】释放资源
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    // 移除长链接相关
    [[LKAppExpress sharedInstance] removeConnectListener:self];
    [[LKAppExpress sharedInstance] removeDownStreamListener:self.imsWidgetDeviceListene
r];
}
- (void)dealloc {
    [kIMSThingManager destroyThing:self.thingShell];
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

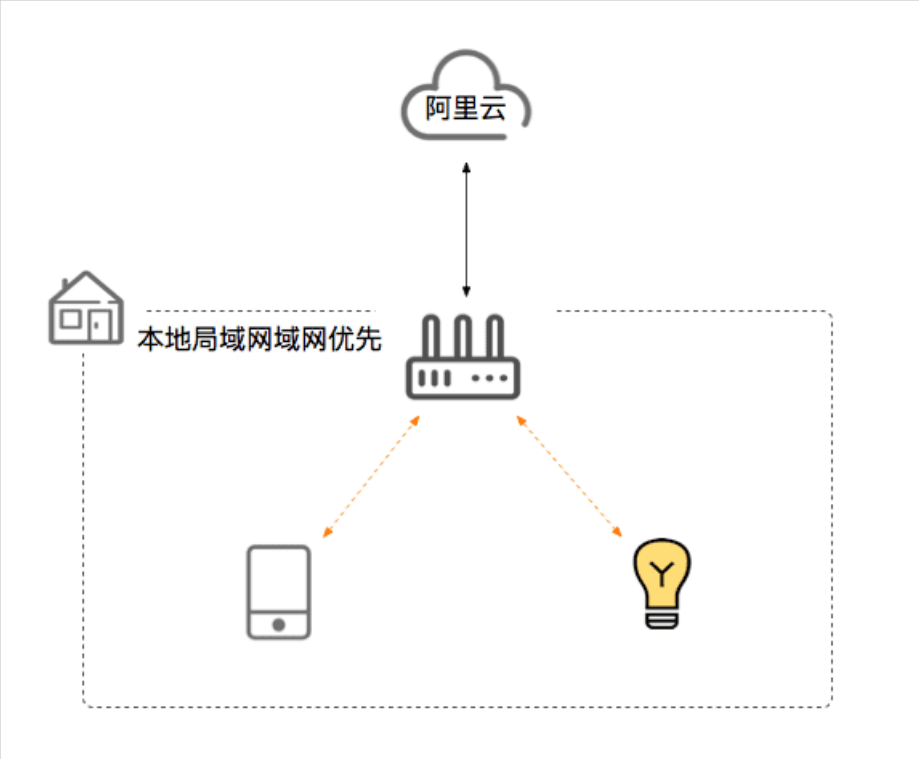
6.本地控制

6.1. 本地通信开发实践

APP和设备处于局域网环境中，未和外网连接，往往需要通过本地通信的方式，对设备进行控制和查询。阿里云IoT提供了一种本地通信方案，可以达到该目标。

背景信息

设备端的Link Kit SDK中已内置本地通信功能（ALCS），这样集成Link Kit SDK的设备，即可使用本地通信能力。集成了ALCS client功能的android app、IOS app、边缘网关等终端都可以通过本地控制与芯片模组产品交互。

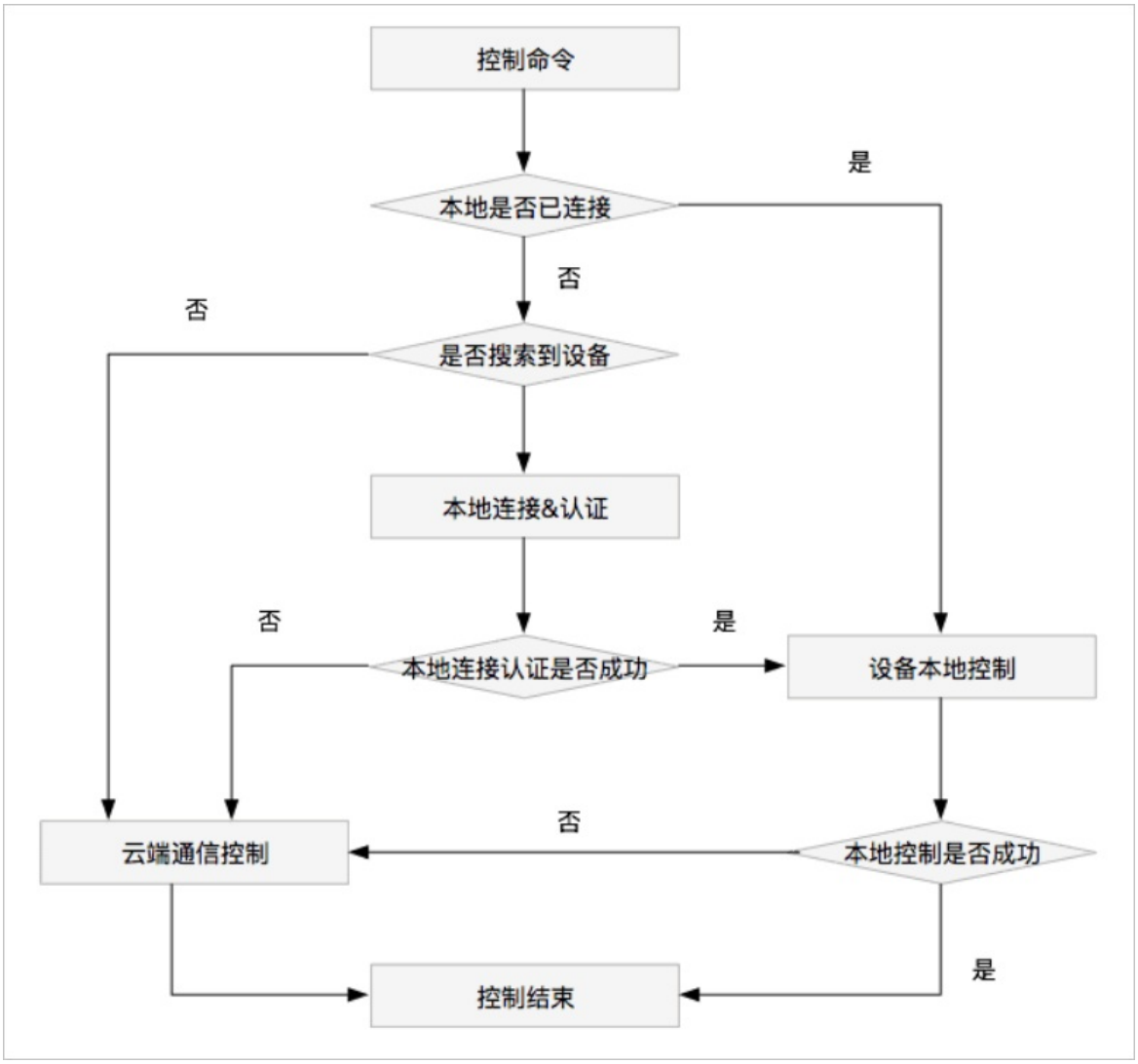


第一次连接和认证阶段 APP 会和云端有一次数据交互来获取认证信息，连接认证成功之后会将认证数据保存在本地，后续的连接认证优先使用本地缓存的认证数据。

操作步骤

1. App端开发。

   App端流程如下。



- iOS

  iOS端开发依赖 IMSThingCapability.framework。

  ```
  pod 'IMSThingCapability'
  ```

- Android

  安卓端开发请依赖 public-tmp-ext-boneplugin SDK 包。

  ```
  // 设备模型 SDK
  api ('com.aliyun.alink.linksdk:public-tmp-ext-boneplugin:1.3.5.1'){

  }
  ```

2. 设备端开发。

   对于本地通信功能，设备端只需要使用IOT_RegisterCallback函数注册ITE_PROPERTY_GET事件，实现对应回调函数即可。

   接口说明如下。

   - 当该事件被触发时，用户收到的payload格式为本地通信请求的设备属性列表，以json格式描述。例如：`["property1","property2","property3"]`。

- 当用户回复该本地通信请求时，需按如下格式回复消息：`{"property1":...,"property2":...,"property3":...}`。

  示例如下。

  property1为int型，property2为字符串类型，property3为复合类型，包含int型的item1和字符串类型的item2：

  ```
  {"property1":23,"property2":"hello,world","property3":{"item1":23,"item2":"hello"}}
  ```

  具体代码可参考SDK Example中的user_property_get_event_handler回调函数的实现。

6.2. 本地倒计时功能开发实践

本文提供了一个插座本地倒计时功能的开发案例，开发者可以参考本文，实现任意设备的本地倒计时功能。

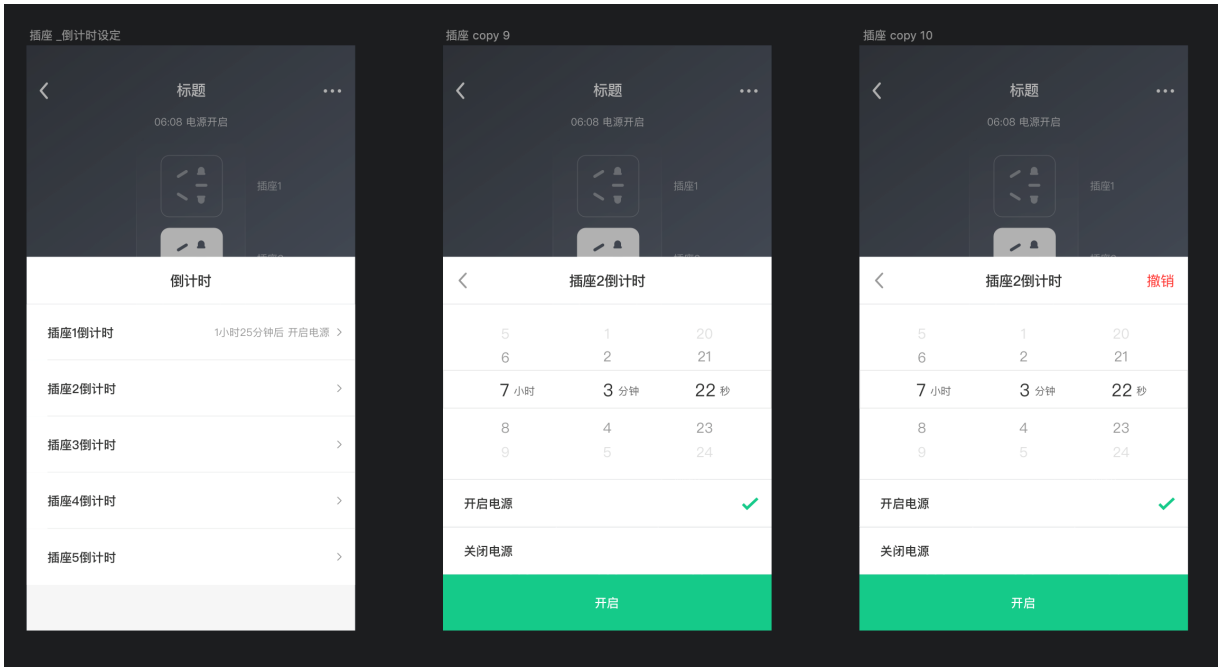
背景信息

“本地倒计时”是指，由App页面向设备端下发“开始倒计时”的任务后，设备端开始根据本地时钟，执行倒计时任务。

要实现本地倒计时的效果，设备端和APP端需要按照本文推荐的方式进行实现。

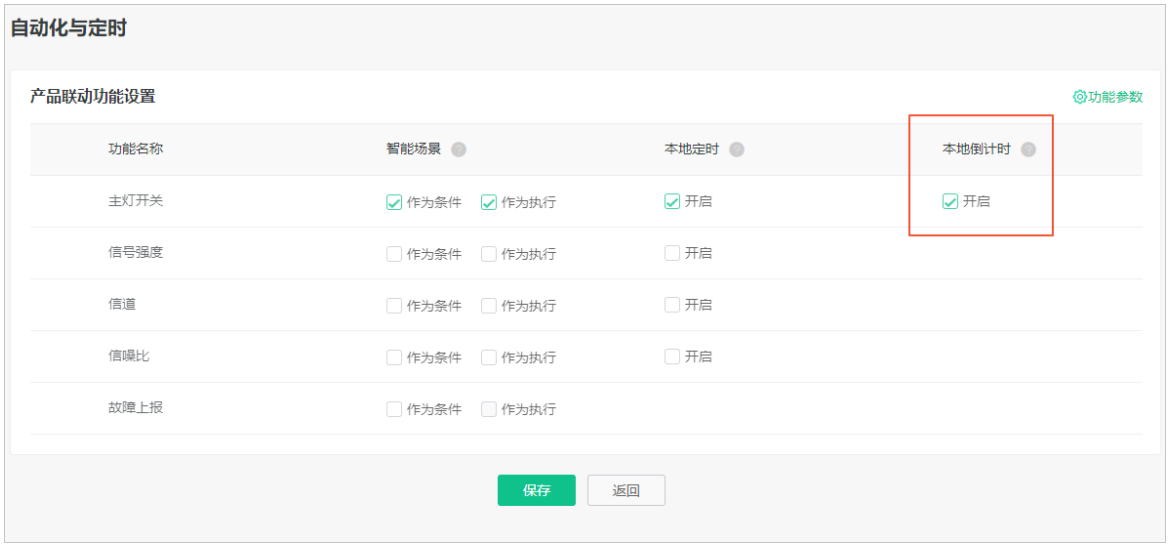
- 设备端：按照平台标准数据格式，实现倒计时任务的增删改。
- APP端：在公版App中，或平台给自定义App提供的设备界面中，如“灯”“开关”，可以直接使用倒计时功能。

与单路倒计时不同的是，多路倒计时可以同时对多个设备上的布尔值设定本地倒计时任务，如多路插座。



操作步骤

1. 在控制台**人机交互 > 自动化和定时**处，勾选本地定时功能，并在功能参数中设置本地定时的最大条数（与设备端的存储、性能有关）。



平台会在功能定义中自动插入一个本地倒计时的标准功能（CountDownList），并支持多个定时任务。

数据结构说明如下。

- 功能名称：倒计时列表
- 标识符：CountDownList
- 类型：JSON

```
CountDownList: {
Target: "PowerSwitch"(string，当次设置操作指定的布尔值的identifier),
Contents: "PowerSwitch-1-1-123-1535644800000,LightSwitch-0-1-456-1535644800000"(string
，该设备的所有倒计时任务字符串，具体格式说明见下方)
XXX1:0(bool，该产品已有的布尔类型标准属性),
XXX2:0(bool，该产品已有的布尔类型标准属性)
}
```

CountDownList是可选的标准属性，其中Target和Contents为CountDownList的必选属性二者都是string类型（最大长度为2048字节）。

*XXX1*和*XXX2*为当前产品已有的标准布尔属性，可以由开发者随意增删，默认值为0。

> **说明** 功能定义里如果含有CountDownList，则CountDownList里除了Target和Contents外，还需要含有其他的布尔属性，否则运行会报错。

参数名称	类型	属性名	是否必选	长度	解释
Target	string	操作对象	是	2048	当次操作的布尔值类型的标准属性，用于表时倒计时的地点位置。

参数名称	类型	属性名	是否必选	长度	解释
Contents	string	倒计时命令	是	2048	当同时设置多个布尔值属性的倒计时任务，且当前没有数组类型的数据，需采用该参数来存放设备的所有倒计时任务。

2. 功能设计。
   - 设计倒计时功能。
     - 如果当前产品的功能定义里有CountDownList属性，那么此时走新的倒计时列表逻辑，在当前组件展示倒计时列表相关界面。
     - 如果当前产品的功能定义里没有CountDownList属性，但是有CountDown属性，那么则走以前的倒计时插件逻辑。
   - 撤销某个倒计时任务。
     - App删除Contents字段里对应identifier的任务，同时设置Target为这个identifier，上报云端，下发给设备端。
     - 设备端收到新的CountDownList属性，发现Target指定了一个identifier，但是Contents里却没有这个identifier，那么删除正在执行的identifier对应的倒计时任务，不影响其余的倒计时任务。

   场景举例如下。

   > **说明** 整个插件和设备端上报云端的过程中，CountDownList里的*XXX1*和*XXX2*等布尔属性可以设置为任意符合布尔属性的值。

   i. 单击倒计时按钮。
   ii. 检查TSL，发现CountDownList里还有额外的*XXX1*，*XXX2*属性。
   iii. 倒计时列表弹层展示*XXX1*、*XXX2*的中文名称列表。
   iv. 选择其中一个属性，例如PowerSwitch，展示设置时间和动作界面，设定好时间（1000s）和动作（打开）。
   v. 设置 `Target: "PowerSwitch"` 和 `Contents: "PowerSwitch-1-1-1000-1535644800000"` ，下发给云端。
   vi. 设备端收到CountDown后，解析Contents和Target内容，设置实际的定时任务。
   vii. 设置了第二个属性XXX2。Contents为： `PowerSwitch-1-1-1000-1535644800000,XXX2-0-1-2000-1535644800000` ，Target为：XXX2。

   1000s到了之后，第一个PowerSwitch倒计时任务结束，设备端删除Contents里PowerSwitch这一条任务，同时设置Target为PowerSwitch，将整个CnountDownList属性上报到云端，插件侧查询CountDown属性，发现Contents里没有Target指定的布尔属性，代表PowerSwitch倒计时任务已执行，进而提示PowerSwitch倒计时任务执行完毕。

6.3. 第三方蓝牙通信插件适配指南

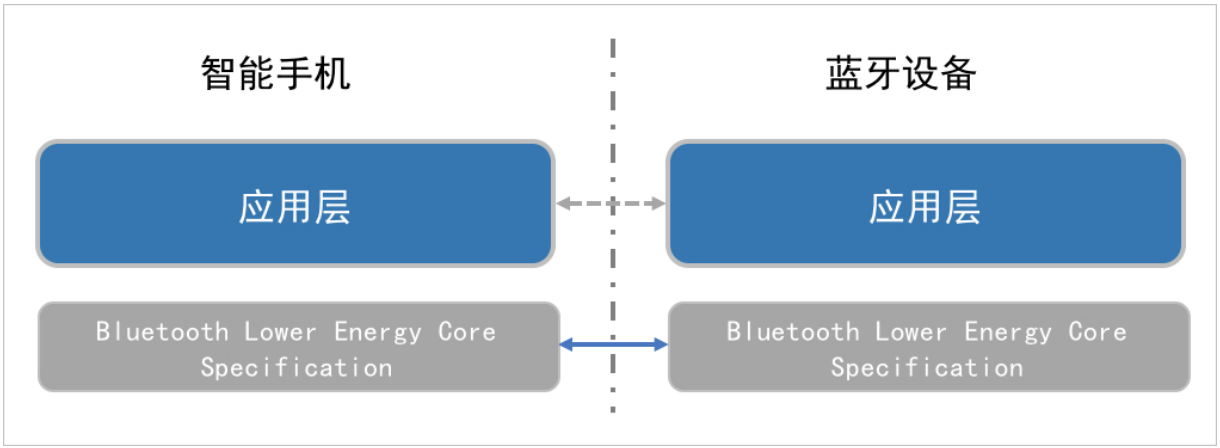
由于各家蓝牙应用层协议千差万别，为了将不同的蓝牙应用层协议接入到统一框架，势必需要有一个适配层来抹平各个协议的差异。如阿里巴巴IoT就定义了一套应用层的通信协议：Breeze。

背景信息

通常，各家的蓝牙应用层协议都会定义如下功能。

- 蓝牙广播：广播设备相关信息，如厂家信息，设备型号信息，设备MAC地址等。
- 数据通信：包括数据分段，组包等功能。

因此，阿里云IoT在蓝牙接入框架中引入了蓝牙通信插件管理系统（即Link Protocol Bridge System，简称LPBS）。



蓝牙通信插件管理系统（LPBS）定义了一套抽象的接口，可以将不同的蓝牙协议设备接入到统一框架中，从而可以按阿里云IoT定义的物模型对蓝牙设备进行控制以及感知。每一个具体的实现我们称为：插件。

操作步骤

1. 获取身份信息。

   智能设备需要在阿里云IoT平台获得一个身份后，才能将某个智能设备归属到某一个用户账号下，进一步地可以实现设备数据的上下云以及设备地相关控制。在阿里云IoT平台，设备身份由以下二者确定。

   - ProductKey是一串随机字符串，可以理解为设备的产品型号，在生活物联网平台注册产品时由阿里云IoT平台颁发。
   - DeviceName是阿里云IoT平台特有的设备的ID。与ProductKey共同确定一个IoT平台的唯一设备。DeviceName可以是阿里云IoT平台生成的一串随机字符串，也可以是厂家在阿里云IoT平台录入设备时指定的字符串。

   智能设备在出厂时，厂家会给设备颁发一个全球唯一的ID来标识该设备。此ID一般由两部分组成。

   - ProductModel，即产品型号，如某冰箱型号：BCD-320WGPZM。
   - DeviceId，即设备ID，一般为Mac或者SN。

   智能设备在接入阿里云IoT平台时需要完成私有ID跟阿里云IoT平台颁发的ID的映射。

   i. 创建产品。

      在生活物联网平台上，厂家（或ISV）可以新增一个产品，在注册产品时阿里云IoT平台会给该产品颁发一个ProductKey来标识此产品。

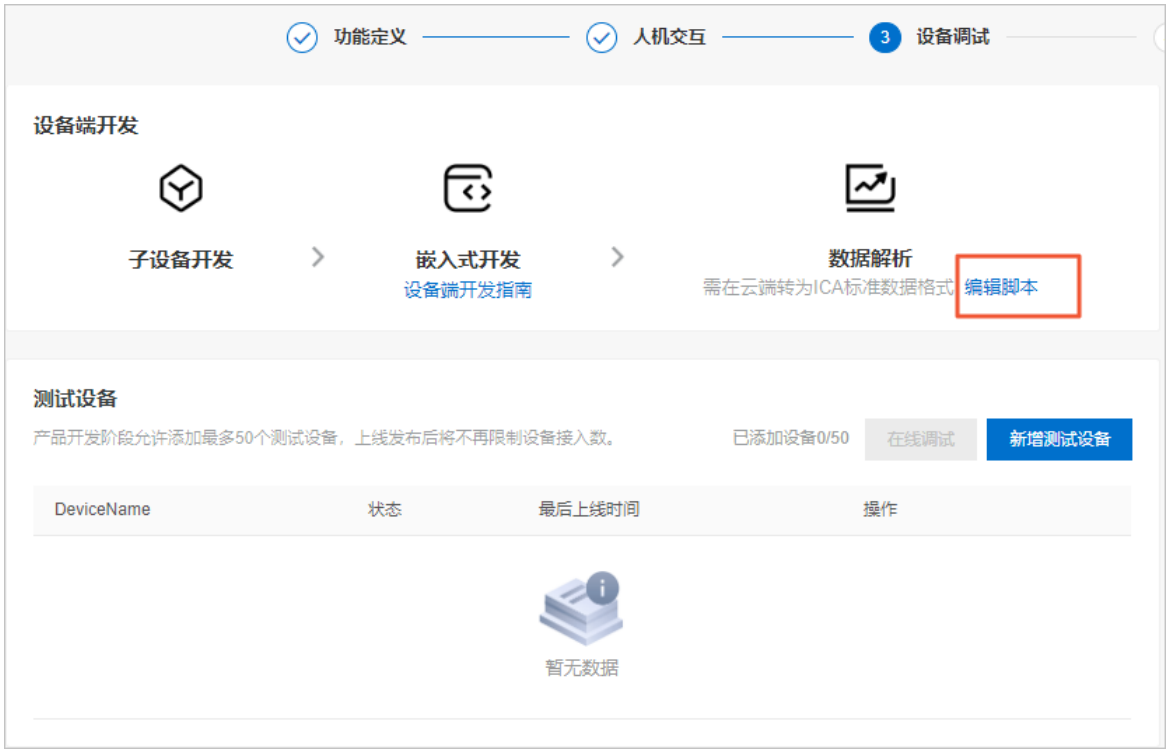
ii. 录入设备ID。

厂家（或ISV）可以将该产品下的设备批量录入DeviceId。阿里云IoT平台会为该批次设备颁发设备身份，并以录入的DeviceId作为设备的DeviceName使用。

2. 数据格式转换脚本。

第三方蓝牙设备在开发时并没有按物模型来定义设备的功能，因此需要翻译成蓝牙设备能够理解的数据格式，否则，蓝牙设备会无法识别。例如以开灯这一场景举例，按物模型封装的消息格式如下所示。

```
{
"id":123,
"version":"1.0.0",
"method":"thing.service.property.set",
"params":{
"LightSwitch":1
},
}
```



3. 开发蓝牙通信插件管理系统中的插件。

   通信插件抽象接口分为三个部分：

   - 蓝牙设备发现接口

     开发者需要将广播中的ProductModel转换成产品注册时获取的ProductKey，Mac地址可以直接当做DeviceName使用。

     在生活物联网的控制台的**量产管理**中购买激活码时，选择批量上传的方式生成激活码，同时上传Mac地址作为DeviceName。

- 设备连接接口

  这个接口需要实现手机（即蓝牙Central）跟蓝牙设备（即蓝牙Peripheral）建立连接。

- 数据发送/接收接口

  需要完成将业务数据发送到蓝牙设备，同时将蓝牙设备的响应数据返回给上层的蓝牙接入框架。另外，还需要实现事件订阅功能，可以订阅蓝牙设备的事件，在蓝牙设备发生事件时，App侧可以收到。

4. 绑定及控制蓝牙设备。

   在插件实现后，第三方蓝牙设备的后续流程跟阿里云IoT蓝牙设备是一样的，蓝牙设备的绑定以及控制等流程可以参见开发指南中移动端SDK介绍。

相关代码

- iOS端：插件抽象接口头文件基于阿里蓝牙协议（Breeze）实现的通信插件，可以参考代码BreezeBridgeSample。

- android端：插件抽象接口头文件

6.4. 使用OTA控制台升级固件

使用生活物联网平台设备端SDK的内置OTA功能，用户只需要适配所使用的模组和新品，即可使用生活物联网平台的云端OTA服务。

前提条件

设备已烧录了一个固件版本，且已成功连接到生活物联网平台。

背景信息

总体OTA升级流程如下：

1. 设备端开发升级所需的固件

   编译设备端新固件，固件中需要内置版本号信息，设备完成固件烧写并重启后，会上报该新版本号到OTA服务端。

2. 控制台（云端）执行升级过程

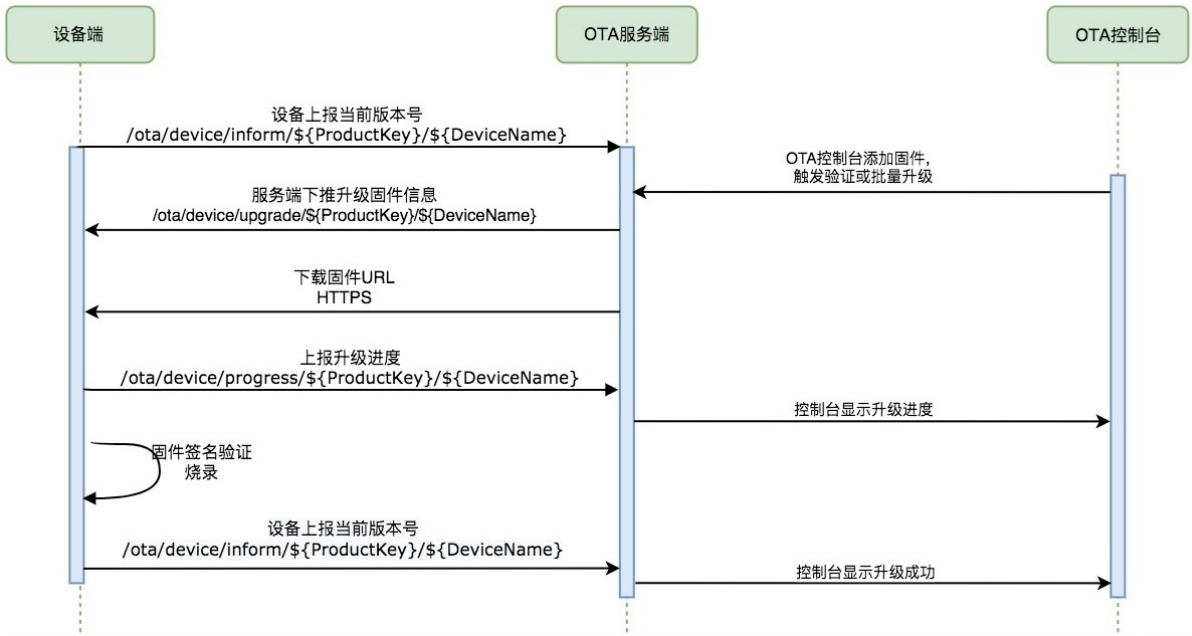
   用户在OTA控制台执行操作，包括上传固件、验证固件、批量升级、管理固件等。

3. 手机App端确认升级和验证升级结果

   如果固件升级开启App确认功能，则需要用户在手机App上确认升级任务后，控制台才会执行固件升级。升级完成后，在手机App上验证升级结果。

一、设备端开发升级所需的固件

设备端（以Wi-Fi设备为例）主要有3个topic来与云端进行交互，流程如下图所示。



1. 设备端通过以下topic向云端上报当前版本号。通常设备上电开机后需要上报一次当前固件版本号。

   当设备完成OTA固件下载并烧录，重启后该topic上报的则为新固件的版本信息。

   ```
   /ota/device/inform/${productKey}/${deviceName}
   ```

2. 当有新版本固件时，云端通过以下topic向设备端推送新固件的版本信息。

```
/ota/device/upgrade/${productKey}/${deviceName}
```

3. 设备端通过以下topic向云端上报OTA升级进度。该进度可以在控制台以及手机App中查看。

```
/ota/device/progress/${productKey}/${deviceName}
```

1. 根据设备端开发文档和设备端与云端交互的流程图，开发待升级的设备固件。详细的OTA开发编程请参见设备OTA编程。
2. 编译生成新的设备固件，并确认设备端新固件的版本号。

   在SDK根目录，执行编译命令。

```
aos make clean
aos make living_platform@<模组名>
```

   以下为编译过程中的日志片段，其中app_version_new对应的即为新固件的版本号。

```
Check if required tools for esp8266 exist
Making config file for first time
processing components: living_platform esp8266
platform/mcu/esp8266 vcall init auto_component
server region: MAINLAND
server env: ONLINE
APP: living_platform
Board: esp8266host
user: liuese
branch: rel_1.1.0
hash: 0f1596cffffd9a4646a3cce4d3c7ba9b4f14649d
app_version_new:app-1.0.0-20191107142256  //新固件的版本号
firmware type: RELEASE
FEATURE_SUPPORT_ITLS != y, so using normal TLS
app_version:app-1.0.0-20191107142256
kernel_version:AOS-R-1.3.4
server region: MAINLAND
server env: ONLINE
APP: living_platform Board: esp8266
```

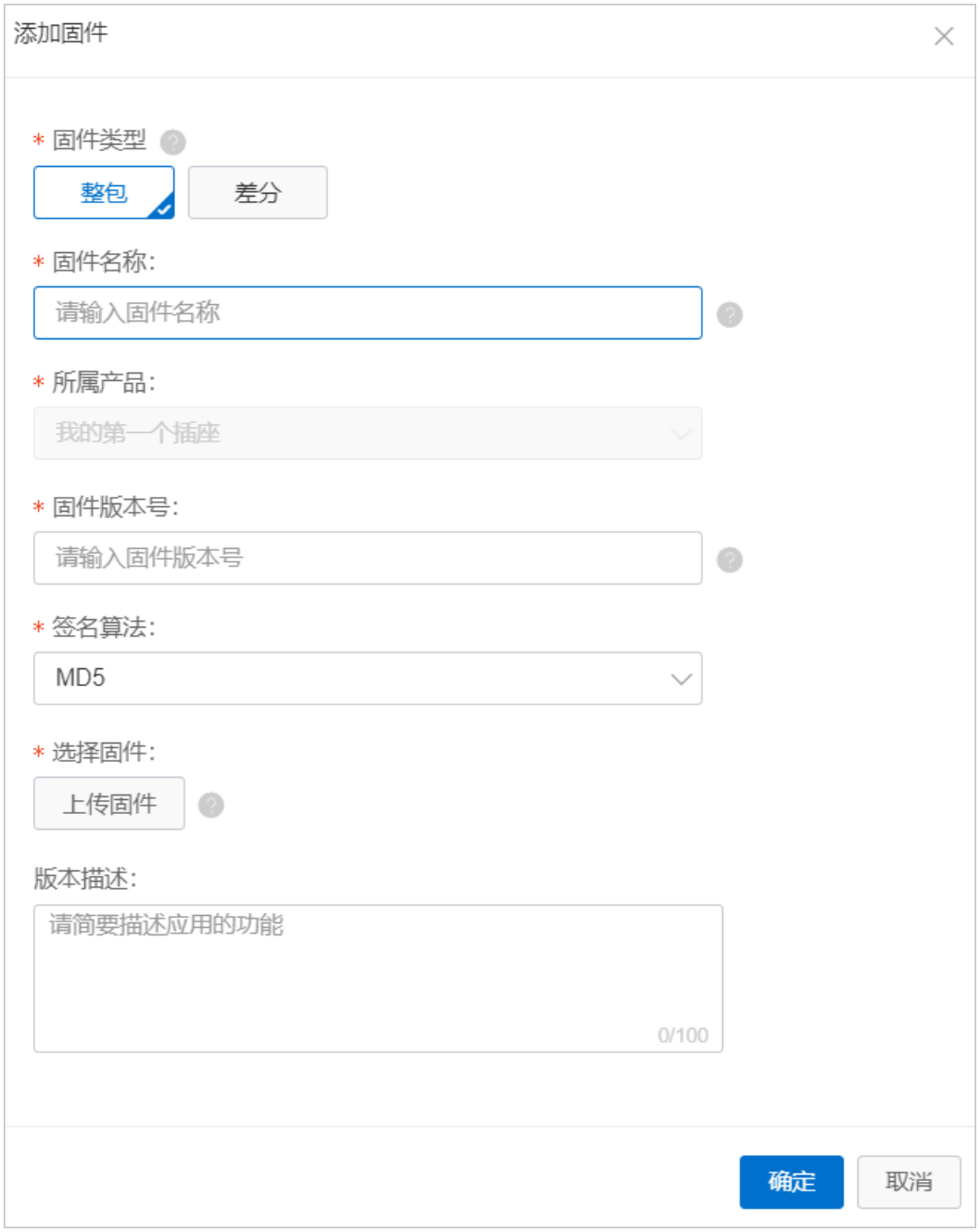
二、在控制台执行升级过程

1. 登录生活物联网平台的控制台。
2. 上传固件。
   i. 进入**运营中心 > 设备运维 > 固件升级**页面。

ii. 在固件列表的下拉框中选择产品名称，并单击**新增固件**。

iii. 输入固件相关参数，并单击**确定**完成。

通常使用容易识别的固件名称和版本号来标识固件，建议将固件版本号与设备固件自身版本号保持一致。详细参数解释请参见固件升级。



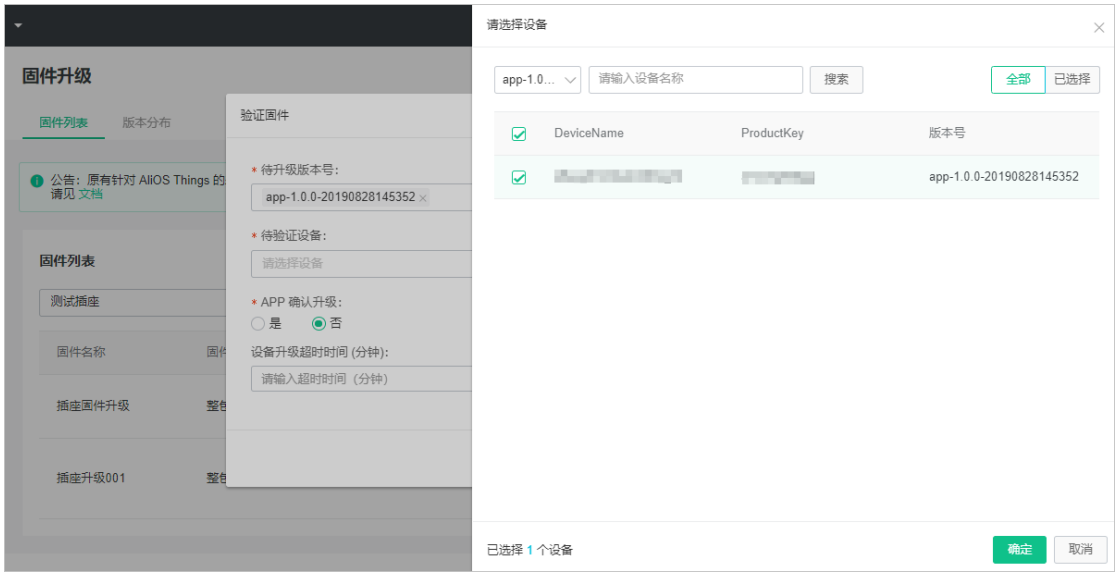
3. 上传固件后，使用测试设备验证固件。

   固件验证的目的是为了保证固件质量，并且，通常建议选择3～5个测试设备来执行固件验证。

   - 确保新固件版本正确，固件运行正常。
   - 确保OTA升级服务可以正常执行，设备可以正常完成升级任务。
     i. 返回**运营中心 > 设备运维 > 固件升级**页面。
     ii. 单击固件列表中操作列中的**验证固件**。

iii. 配置验证固件的相关参数。

- 待升级版本号：现有设备版本号，即原版本号
- 待验证设备：选择待验证设备，支持单选、多选、全选
- APP确认升级：选择是否支持App升级确认功能，如果通过公版App连接设备，可以使用公版App确认升级该固件
- 设备升级超时时间：选择超时时间，达到超时时间后仍未完成固件验证，则验证固件失败



iv. 单击**确定**，开始执行固件验证。

如果APP确认升级配置为是，需登录手机公版App，进入**我的 > 设置 > 固件升级**中，单击**确认升级**后，才开始执行固件验证。

v. 选择新的测试设备，执行以上步骤。

4. 批量升级固件。

固件验证通过后，进入批量升级阶段，可以支持全量、分批向设备推送升级。

i. 返回**运营中心 > 设备运维 > 固件升级**页面。

ii. 单击固件列表中操作列中的**批量升级**。

iii. 配置批量升级的相关参数，并单击**确定**完成。

根据您自身需要选择合理参数，控制OTA升级节奏，提高升级成功率。批量升级的参数介绍请参见固件升级。

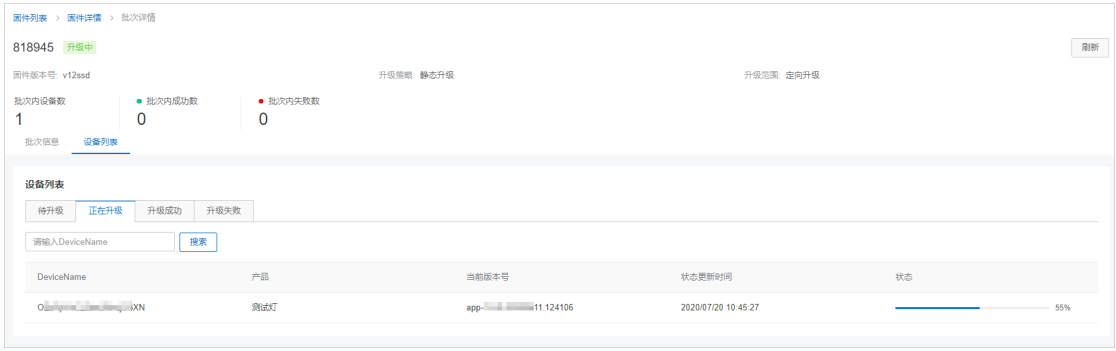
5. 管理固件升级。

   控制台支持对固件的管理，包括了新增、产品维度固件枚举、查看指定固件详情、删除等操作。在固件详情界面中，可以查看当前固件详细信息，还包括了目标设备总数、升级成功数、失败数等信息，有利于厂商掌握该版本升级的结果信息。下面介绍查看固件升级结果的操作为例，更多操作可参见固件升级。

   i. 进入**运营中心 > 设备运维 > 固件升级**页面。

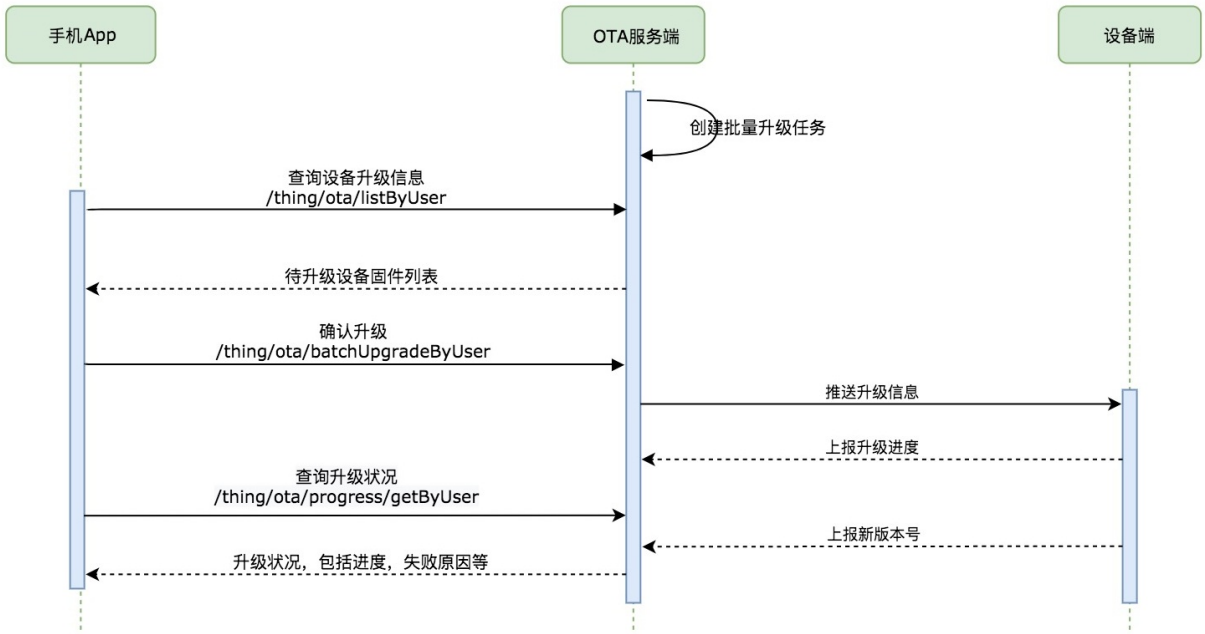
   ii. 单击固件列表中操作列中的**查看**。

iii. 在**批次管理**页签下的**设备列表**页签中，查看升级状态。



三、手机App确认升级和验证升级结果

App端主要有4个topic来与云端进行交互，流程如下图所示。



1. 查询待升级固件

   App端通过以下topic向云端查询用户绑定设备的待升级信息，根据返回信息展示待升级设备名称列表。在控制台添加批量升级任务后，云端会将相应待确认升级固件列表信息会返回给用户（App端）。

   ```
   /thing/ota/listByUser
   ```

2. 确认设备升级

   当APP确认升级配置为是时，云端需要收到App端通过该topic上报的消息后，才能开始执行升级任务，并将该执行任务推送到设备端。

   ```
   /thing/ota/batchUpgradeByUser
   ```

3. 获取正在升级的设备列表（公版App中目前没有使用该topic）

   App端选择周期性或者下拉刷新等UI交互方式，通过以下topic向云端查询当前正在升级的设备信息。云端返回升级的设备列表信息等。

   ```
   /thing/ota/upgrade/listByUser
   ```

4. 获取指定固件的升级信息

   根据指定设备以及固件版本号，App端通过以下topic向云端查询某固件的升级结果、进度等信息，App端根据云端的返回结果，提示用户该固件升级的相关信息。

   ```
   /thing/ota/progress/getByUser
   ```

下面以生活物联网公版App云智能为例，介绍App端的操作。

1. 登录公版App。
2. 选择**我的 > 设置 > 固件升级**。
3. 查看固件升级，展示待升级信息、发起固件升级、展示升级结果等信息。

7.行业应用

7.1. 灯的App免开发解决方案2.0

为了满足用户对于App界面不同的体验需求，生活物联网平台对灯的界面风格以及场景功能进行了重新设计，提供了一套新的免开发的App标准界面，如果您对界面体验要求比较高，想做一款与众不同的产品，我们推荐您使用App免开发解决方案2.0。

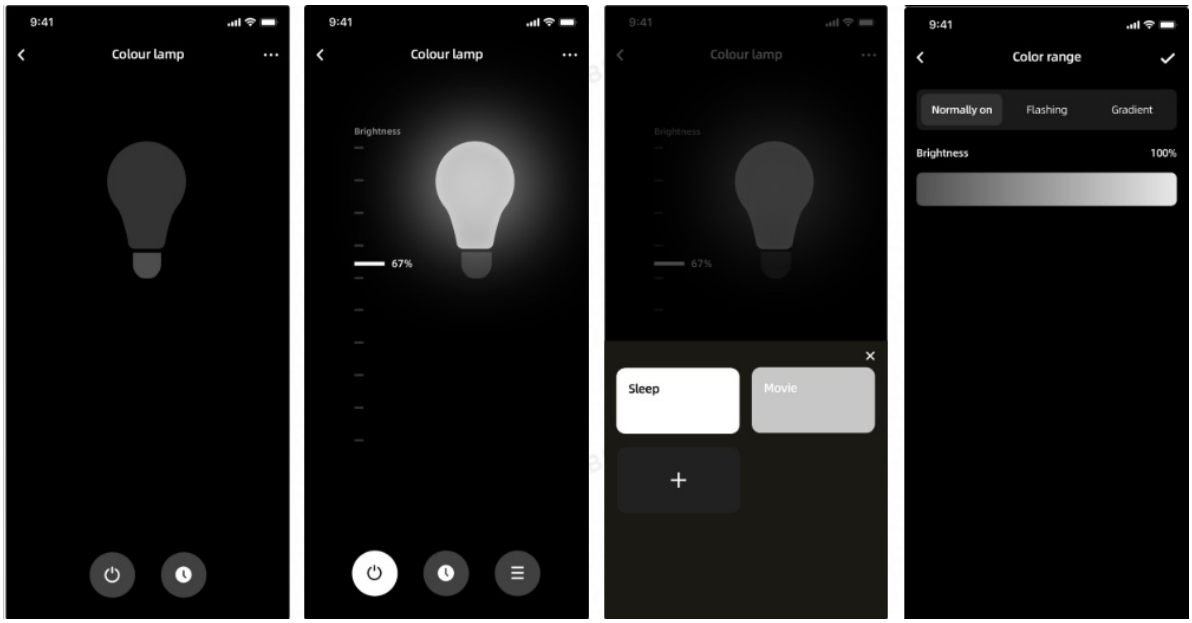
方案概述

搭配生活物联网平台提供的公版App，灯的解决方案提供以下基本功能。

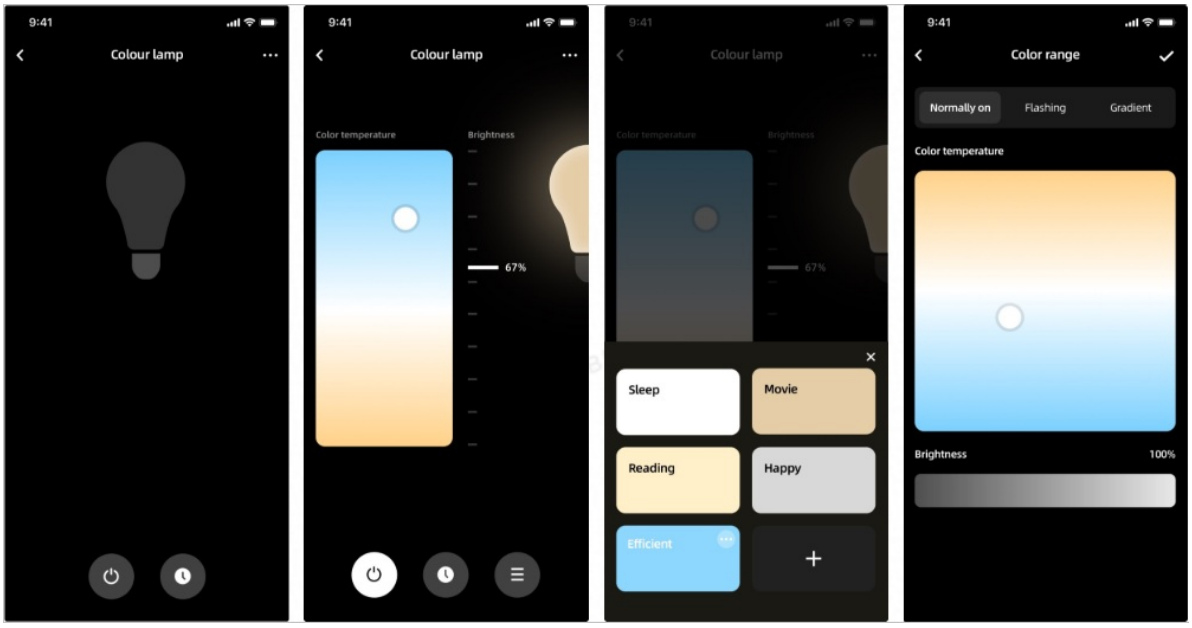
- 白灯的亮度和色温调节
- 彩灯的颜色、亮度和饱和度调节
- 智能场景，支持白光常亮、白光闪烁、白光渐变、彩光常亮、彩光闪烁、彩光渐变6种场景模式
- 定时，支持云端定时或本地定时

根据设备类型不同展示不同的界面。

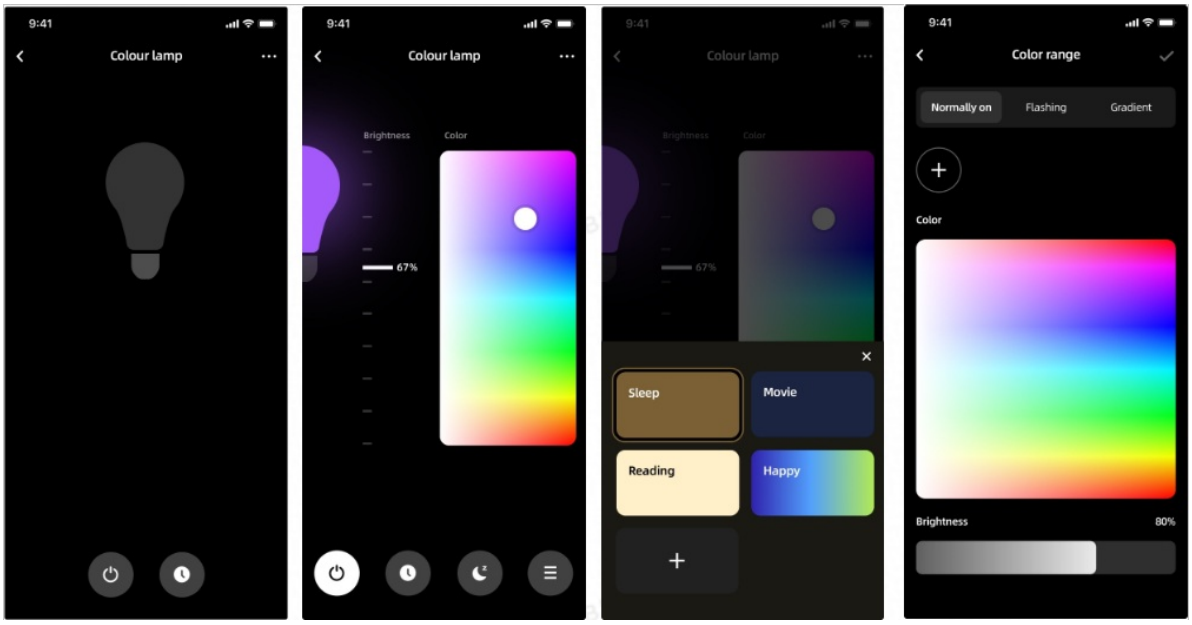
- 一路灯



- 两路灯


9:41
Colour lamp
Color temperature
Brightness
67%
Sleep
Movie
Reading
Happy
Efficient
Color range
Normally on
Flashing
Gradient
Color temperature
Brightness
100%

- 三路灯


9:41
Colour lamp
Brightness
Color
67%
Sleep
Movie
Reading
Happy
Color range
Normally on
Flashing
Gradient
Color
Brightness
80%

- 四路灯

- 五路灯

在控制台开发产品

1. 登录生活物联网控制台。
2. 创建一个产品，并定义产品功能。

   建议您添加以下灯的标准功能，操作请参见创建产品并定义功能。

功能名称	标识符	数据类型	是否必选	功能描述
开关	powerstate	布尔型	是	灯开关

功能名称	标识符	数据类型	是否必选	功能描述
HSV调色	HSVColor	复合型	否	3路灯、4路灯、5路灯必要属性，JSON对象 ○ 色调：Hue ■ 数据类型：整数型 ■ 取值范围：0~360 ■ 步长：1 ■ 单位：度（°） ○ 饱和度：Saturation ■ 数据类型：整数型 ■ 取值范围：0~100 ■ 步长：1 ■ 单位：百分比（%） ○ 明度：Value ■ 数据类型：整数型 ■ 取值范围：0~100 ■ 步长：1 ■ 单位：百分比（%）
明暗度	brightness	整数型	否	1路灯、2路灯、4路灯、5路灯必要属性 ○ 取值范围：0~100 ○ 单位：百分比（%） ○ 步长：1
色温_开尔文	colorTemperatureInKelvin	整数型	否	2路灯、5路灯必要属性 ○ 取值范围：2000~7000 ○ 单位：开尔文（K） ○ 步长：1
本地定时	LocalTimer	数组型	否	元素类型：JSON数组
灯模式	LightMode	枚举型	否	取值：0 - mono（白光）；1 - color（彩光） **说明** 支持两种模式的灯需要此属性，添加完页面会出现白光、彩光两种模式

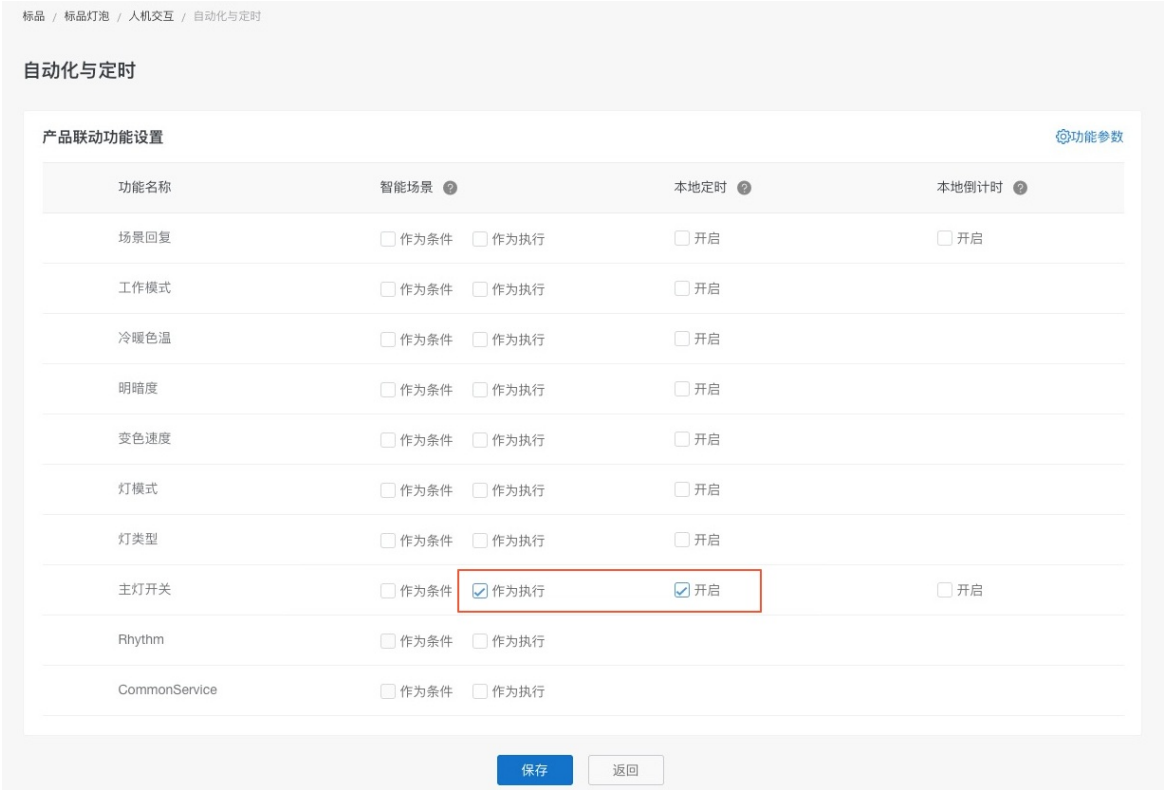
功能名称	标识符	数据类型	是否必选	功能描述
灯类型	LightType	枚举型	否	当定义了灯类型以后，设备上电需要自己上报当前灯的类型，如没有上报则以TSL为准 ○ 0：C ○ 1：CW ○ 2：RGB ○ 3：RGBC ○ 4：RGBCW

功能名称	标识符	数据类型	是否必选	功能描述
灯场景（新增）	LightScene	复合型	否	JSON对象 ○ 灯模式：LightMode 数据类型：枚举型 ■ 0 - mono（白光） ■ 1 - color （彩光） ○ 变色速度：ColorSpeed ■ 数据类型：整数型 ■ 取值范围：0~100 ■ 步长：1 ■ 单位：百分比（%） ○ 场景模式：SceneMode 数据类型：枚举型 ■ 0：常亮 ■ 1：闪烁 ■ 2：渐变 ○ 灯颜色：ColorArr ■ 数据类型：字符型 ■ 数据长度：2048 ○ 明暗度：Brightness ■ 数据类型：字符型 ■ 数据长度：2048 ○ 运行状态：Enable 数据类型：布尔型 ■ 0：已停止 ■ 1：运行中 ○ 场景标识：SceneId ■ 数据类型：字符型 ■ 数据长度：100 ○ 场景参数：SceneItems ■ 数据类型：字符型 ■ 数据长度：2048
心跳（新增）	Heartbeat	布尔型	否	○ 0：停止 ○ 1：正常

> **说明** 当使用LightType属性时，一定要配置以上的功能定义，否则可能会造成面板画面显示不正常。

3. 添加测试设备，参见添加设备。
4. 配置App的功能参数项，参见配置人机交互。

   该解决方案默认使用本地定时功能，您还需要在**人机交互**页面设置定时的功能属性。



开发界面

由于灯的界面较复杂，请您根据以下灯的界面与设备功能之间的逻辑来开发设备。

- 白灯模式

  1路灯、2路灯、4路灯、5路灯支持此模式。

  - 当功能定义里定义了ColorTemperture，或者通过LightType设置了2路灯、5路灯时，白灯模式支持冷暖调节。
  - 白灯模式的亮度Brightness，设置为1~100。
  - 单击白光按钮时，依次下发当前灯光的模式 `LightMode=0` ，以及云端备份的Brightness和ColorTemputure（支持冷暖的灯会下发）。

- 彩灯模式

  3路灯、4路灯、5路灯支持此模式。

  - 单击彩光按钮时，依次下发灯光的模式 `LightMode=1` 以及云端存的HSVColor。

- 彩光模式的色调为H（Hue）、饱和度为S（Saturation）、亮度为V（Value），各取值范围如下。
  - Hue：0~360，与实际灯的是一一对应的
  - Saturation：0~100，与实际灯的是一一对应的
  - Value：1~100，因为实际调节灯的亮度不会变成0，所以在App上调节的范围是1~100
- 每次调节彩光的亮度、饱和度或色调时，都会下发对应的HSVColor。

开发灯场景

方案2.0的灯场景一共支持6种场景模式，分别为：白光常亮、白光闪烁、白光渐变、彩光常亮、彩光闪烁、彩光渐变。各路灯的支持情况如下。

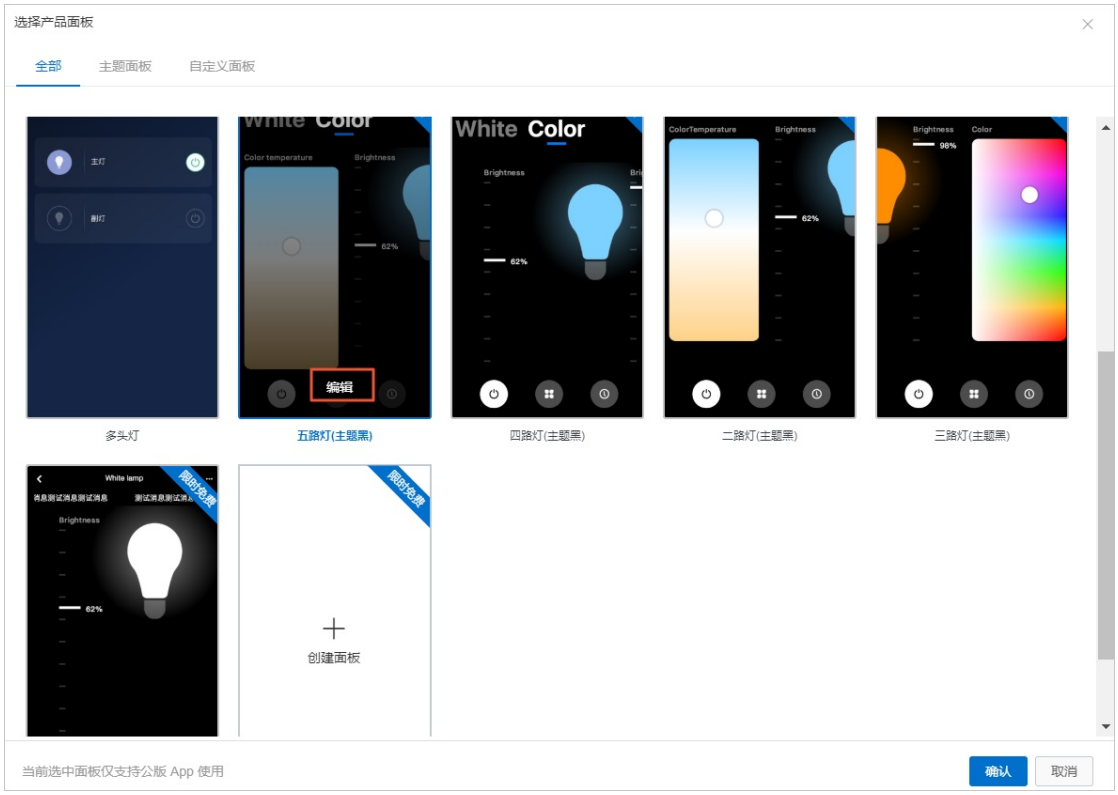
类型	支持的灯场景
1路灯	白光常亮、白光闪烁、白光渐变
2路灯	
3路灯	彩光常亮、彩光闪烁、彩光渐变
4路灯	白光常亮、白光闪烁、白光渐变、彩光常亮、彩光闪烁、彩光渐变
5路灯	

请您根据以下操作开发灯的场景。

1. 登录生活物联网控制台。
2. 预设场景。
   i. 进入产品的**人机交互**页面。

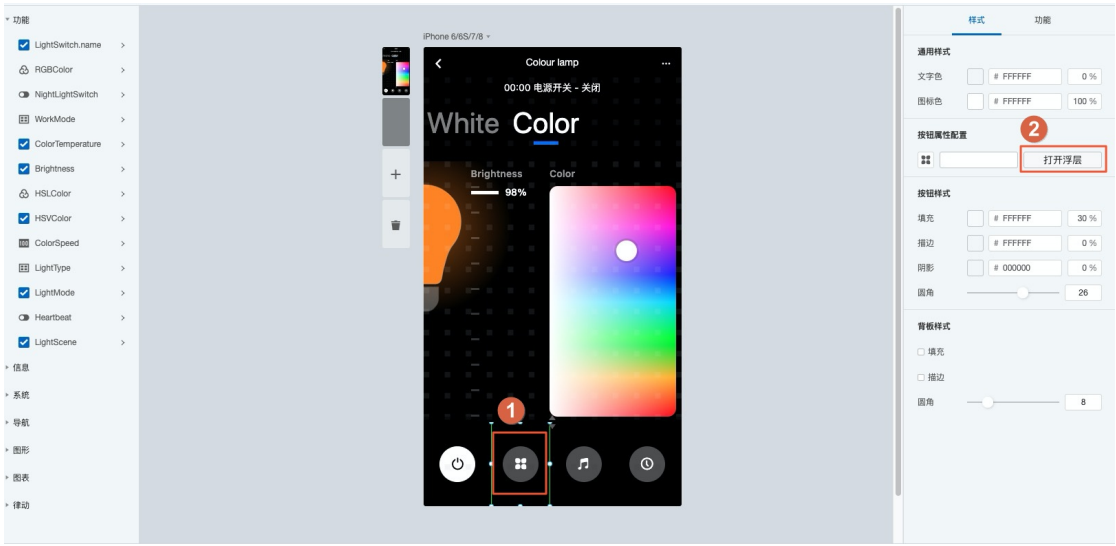
ii. 打开公版App开关，单击选择产品面板处的**未设置**。

iii. 选择**X路灯**（根据预置的lightType来选择几路灯），并单击**编辑**。

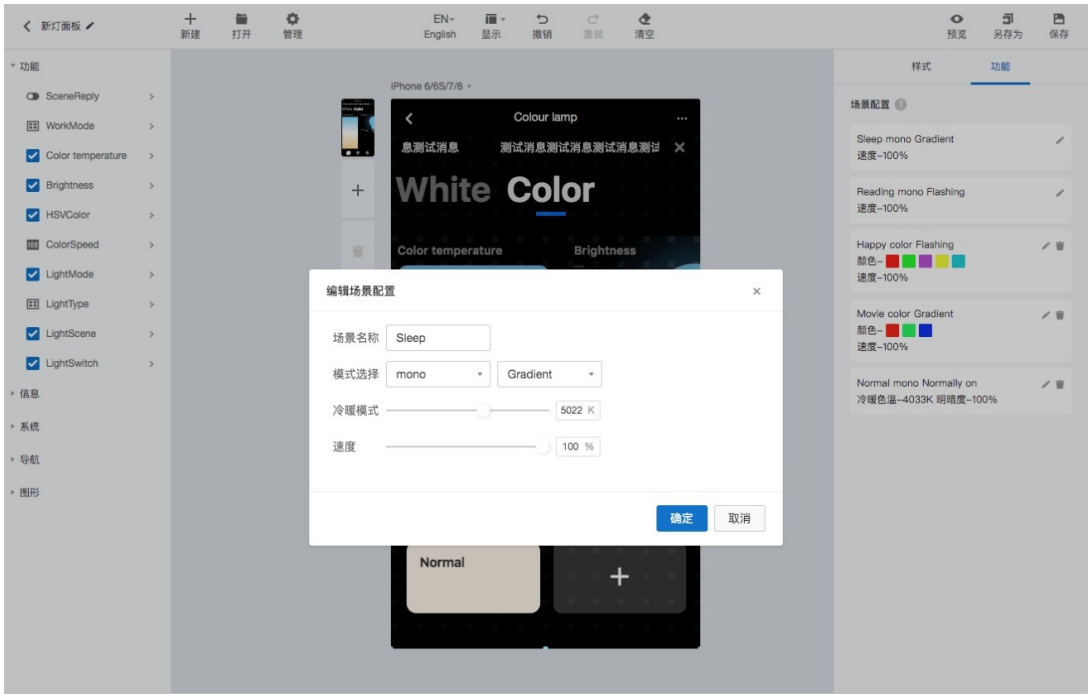


iv. 配置灯界面。

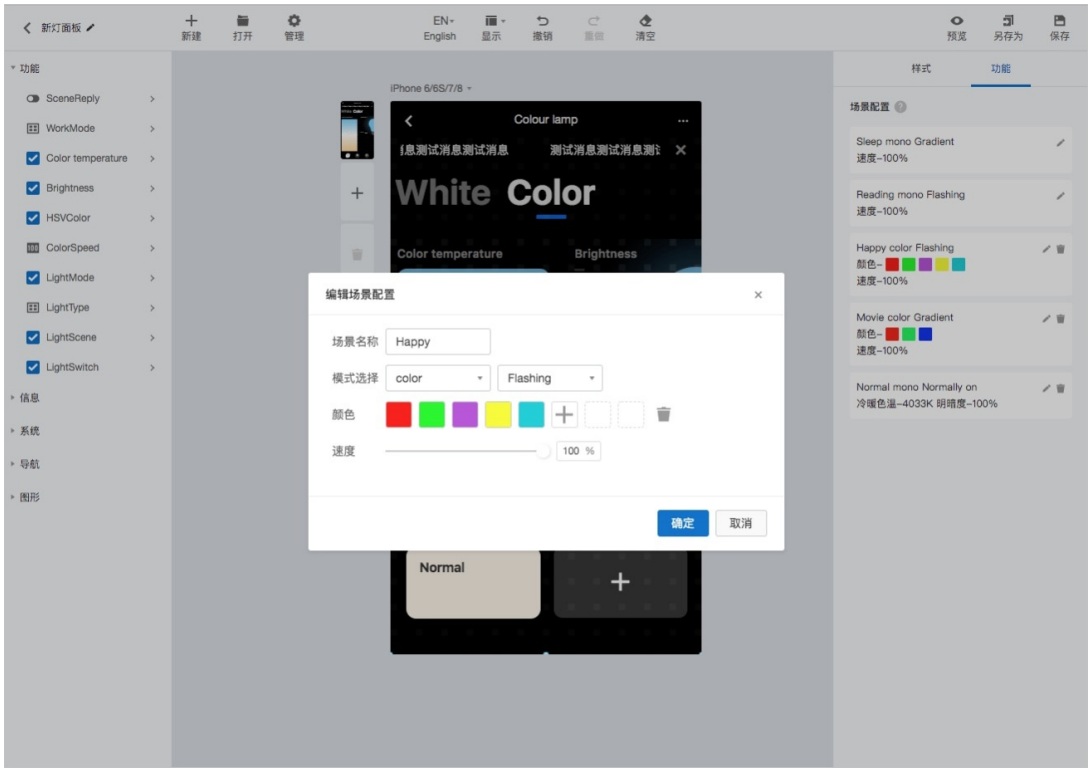
选择灯场景，并单击**打开浮层**。



- 配置冷暖模式



- 配置白光、彩光模式



> **说明** 速度调节为从0%（间隔6s）至100%（间隔1s），速度的步长为50ms。

以彩光场景为例： 可以修改场景名称，6种场景模式，以及场景参数等。不同模式以及1~5路灯所支持的配置不完全相同，支持参数如下。

灯模式	支持的参数情况
白光常亮	支持明暗度调节，如功能定义包含ColorTemperature则支持冷暖调节
白光闪烁	支持速度调节，如功能定义包含ColorTemperature则支持冷暖调节，明亮度Brightness为0和100闪烁
白光渐变	支持速度调节，如功能定义包含ColorTemperature则支持冷暖调节，明亮度Brightness为0~100渐变，渐变步长为1
彩光常亮	仅支持选择一种颜色
彩光闪烁	支持速度调节，最多可选8种颜色，颜色顺序切换
彩光渐变	支持速度调节，最多可选8种颜色，颜色顺序渐变，渐变规则为HSVColor中的Value从0~100渐变，再进入下一个颜色

说明 当您开发同时支持多路灯产品时，如果您在开发产品时还不确定具体支持几路灯，我们允许您添加在功能定义中支持的所有场景。如功能定义中，同时添加了Brightness、ColorTemperature、HSVColor、LightMode，满足5路灯的属性，则可以添加6种场景模式的任意一种。但是最终App用户使用灯时不一定是5路灯，根据设备上报lightType动态决定是几路灯，最终App侧会根据lightType过滤掉设备不支持的场景。

例如，同时预设了白光-闪烁和彩光-闪烁的场景。

- App用户在使用时，如果设备上报 `lightType=1` 即为2路灯，不支持彩光模式，则用户仅能看到白光-闪烁的场景
- App用户在使用时，如果设备上报 `lightType=2` 即为3路灯，不支持白光模式，则用户仅能看到彩光-闪烁的场景

3. 实现灯场景。

   新的灯场景的实现依赖LightScene和HeartbeatStatus属性（属性介绍参见**功能定义**页面）。

i. 用户选择场景时，云端下发数据。

- 白光场景

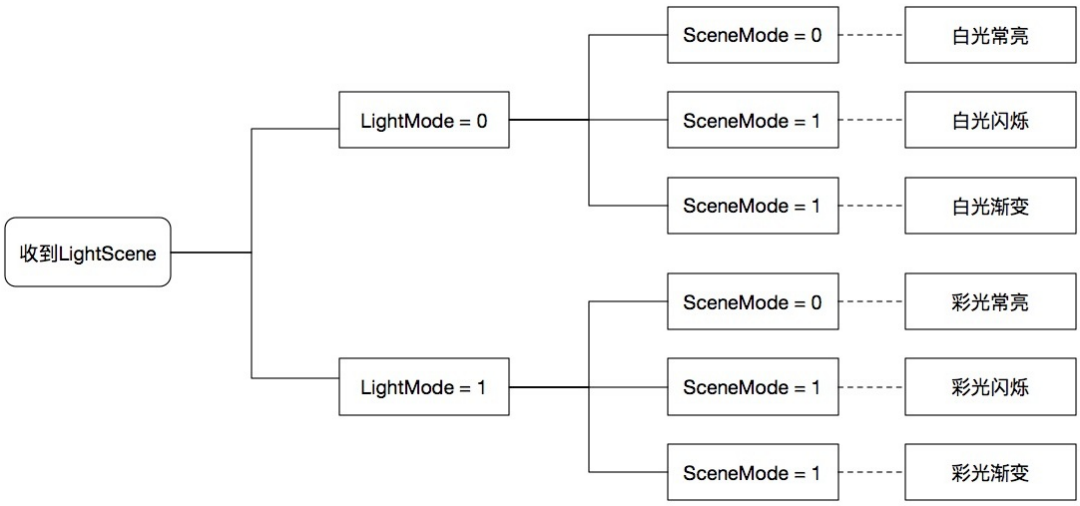
```
{
  "LightScene": {
    "LightMode": 0,
    "ColorSpeed": 100,
    "SceneMode": 1,
    "ColorArr": "[]",
    "Brightness": "{"min":0,"max":100}",
    "Enable": 1,
    "SceneItems": "{"ColorTemperature":4445,"Brightness":100}",
    "SceneId": "B_5"
  }
}
```

- 彩光场景

```
{
  "LightScene": {
    "LightMode": 1,
    "ColorSpeed": 100,
    "SceneMode": 1,
    "ColorArr": "[]",
    "Brightness": "{}",
    "Enable": 1,
    "SceneItems": "{}",
    "SceneId": "B_5"
  }
}
```

其中ColorArr值为JSON字符串，内容如下，注意有些平台双引号需要转义 `{"Hue":6,"Saturation":99,"Value":98},{"Hue":121,"Saturation":96,"Value":97},{"Hue":286,"Saturation":63,"Value":85}`

ii. 设备收到LightScene后，根据LightScene中的参数LightMode和SceneMode来判断当前是哪一种场景模式，执行不同的逻辑。



六种模式对应的设备执行逻辑如下。

灯模式	设备端执行逻辑
白光常亮	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=0，获取SceneItems中的Brightness、ColorTemperature ■ 上报 LightMode、Brightness、ColorTemperature（1路灯和4路灯无ColorTemperature）
白光闪烁	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=0，ColorTemperature为SceneItems中的值 ■ 上报 LightMode、ColorTemperature ■ 根据ColorSpeed定时改变Brightness为0与100（闪烁），并上报Brightness
白光渐变	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=0，ColorTemperature为SceneItems中的值 ■ 上报LightMode、ColorTemperature ■ 根据ColorSpeed 定时改变Brightness为0~100（渐变），步长为1，并上报Brightness
彩光常亮	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=1，HSVColor为ColorArr中的第一个颜色值 ■ 上报LightMode、HSVColor
彩光闪烁	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=1 ■ 上报LightMode ■ 根据ColorSpeed定时改变HSVColor并上报；每次从ColorArr中顺序取下一个颜色值，且循环取值
彩光渐变	■ 上报LightScene，常规属性设置动作，属性上报 ■ 调整LightMode=1 ■ 上报LightMode ■ 根据ColorSpeed 定时改变HSVColor 并上报；每次从ColorArr中顺序取下一个颜色值，取出颜色值后对HSVColor的Value值进行0至100渐变。Value达到100后，取下一个颜色值，Value继续从0至100取值

4. 调试灯场景。

   公版App最多支持10个场景，您最多预设5个场景，剩余的为App用户自定义场景。请您根据业务逻辑来调试预设的场景，并根据以下操作调试App用户自定义的场景（以5路灯为例）。

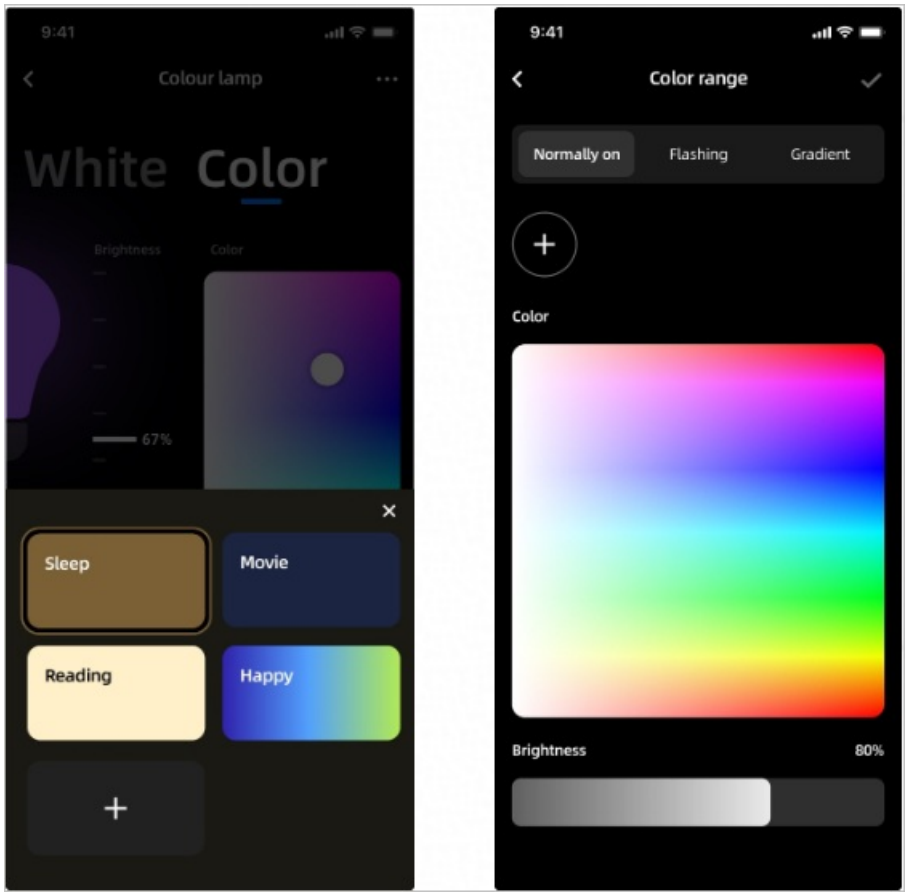
   i. 登录公版App。

   ii. 单击**智能**，并选择**场景**页签。

   此时会弹出浮层，看到支持场景的列表，分为两部分：预设场景和用户自定义场景。预设场景用户不可编辑。

iii. 单击右上角的加号（+）来新增场景。

App用户可以新增常亮、闪烁、渐变等场景，以及最多添加8种颜色。此时设置的参数与生活物联网平台的控制台可设置的参数相同，区别在于自定义用户场景无法切换白光模式与彩光模式。 而是根据1~5路灯适配（1~2路灯仅能设置白光场景；3~5路灯仅设置彩光场景）。App用户可以通过设置白色的场景来实现白光场景的效果。



开发音乐律动场景

音乐律动功能开启后，可在App上使用律动功能，随着手机麦克风接收的音乐节奏，变换灯的颜色和闪烁频率，需配合设备端开发，实现TSL对律动服务的接收。使用音乐律动功能时，务必要保证手机和设备处于同一个Wi-Fi网络下。

请您根据以下操作开发音乐律动的场景。

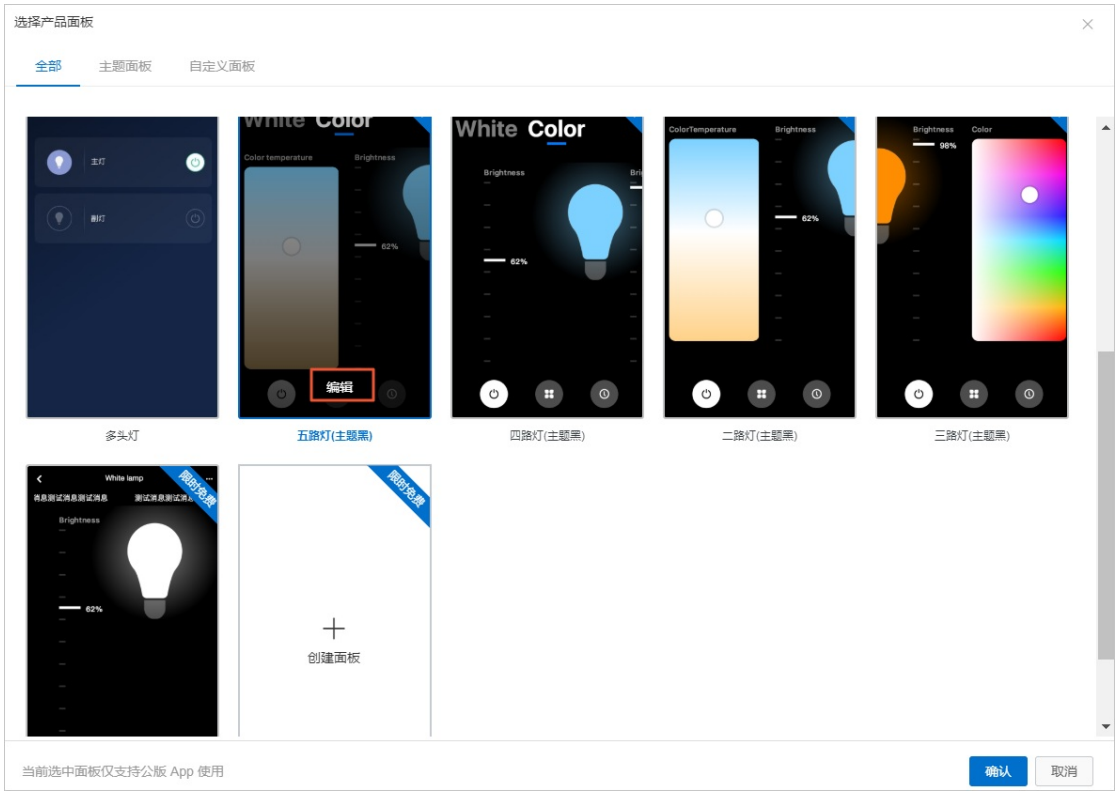
1. 登录生活物联网控制台。
2. 创建产品并定义产品功能。

   您需要新增一个名为Rhythm的服务，详细操作请参见创建产品并定义功能。

3. 预设场景。

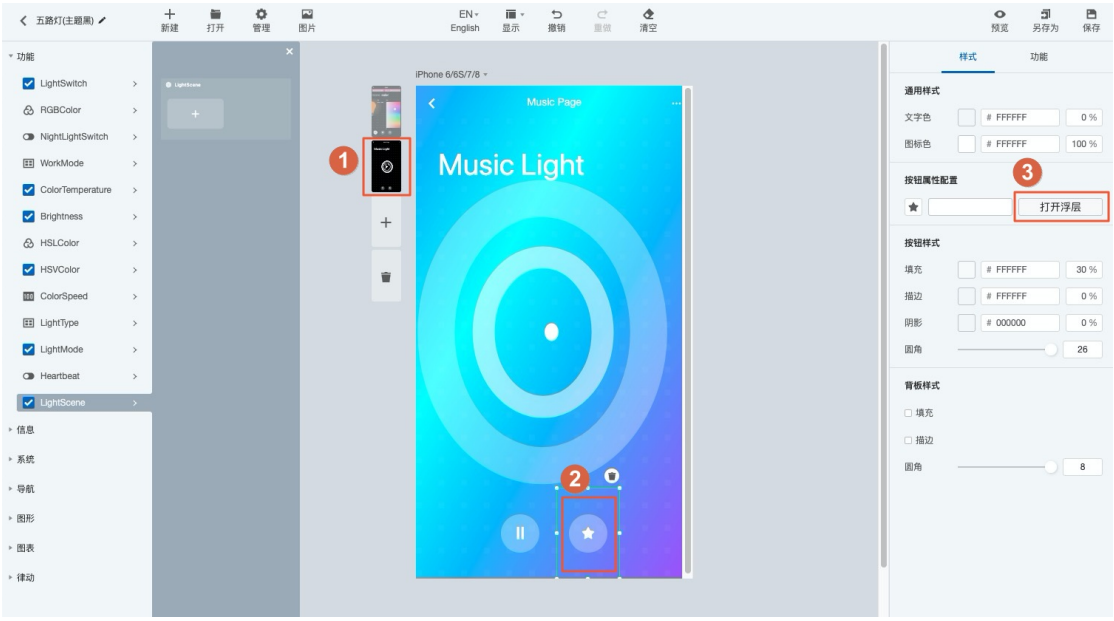
   i. 进入**人机交互**页面。

ii. 选择**X路灯**（根据预置的lightType来选择几路灯），并单击**编辑**。



> **说明** 音乐律动功能仅支持3~5路灯彩灯模式。

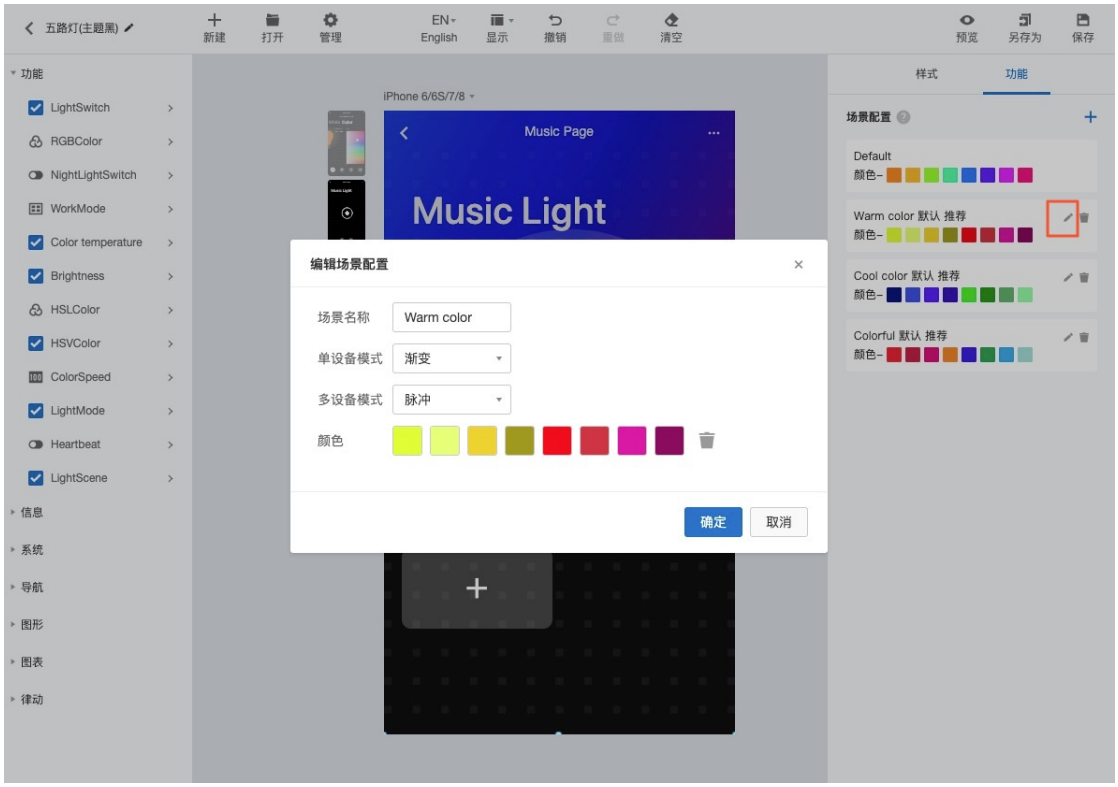
iii. 单击**律动页面 > 律动场景**组件，并打开浮层。



在浮层中您可以看到官方推荐的几种律动场景，即我们在不同音乐模式下反复验证效果较好的场景，建议您可以直接使用。

- 默认场景（不支持编辑和删除）
- 暖色
- 冷色
- 多彩

iv. （可选）选择除默认场景外的一个场景模式，单击铅笔图标编辑律动场景。



参数名称	描述
场景名称	场景名称支持多语言的文案
单设备模式	支持默认（闪烁）、渐变、跳变三种模式
多设备模式	支持推荐、脉冲、同频三种模式，多设备组控时，不同设备可以按不同颜色顺序变化
颜色	最多支持添加8种颜色

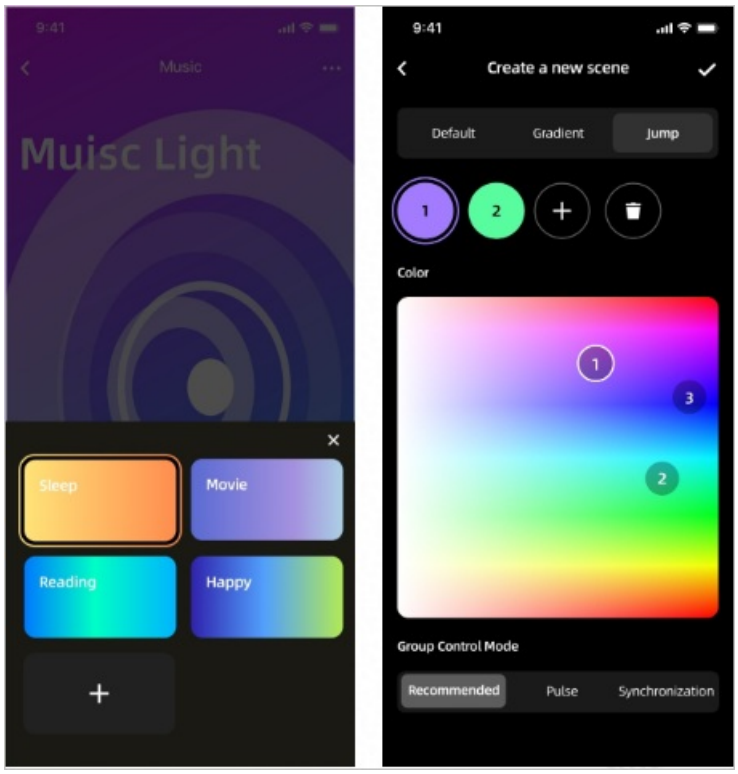
4. 调试音乐律动场景。

   公版App最多支持10个场景，您最多预设5个场景，剩余为App用户自定义场景。请您根据业务逻辑来调试预设的场景，并根据以下操作调试App用户自定义的场景（以5路灯为例）。

   i. 登录公版App。

ii. 单击**智能**，并选择**场景**页签。

此时会弹出浮层，看到支持场景的列表，分为两部分：预设场景和用户自定义场景。预设场景用户不可编辑。



iii. 单击加号（+）新增场景。

App用户可以新增默认（闪烁）、渐变、跳变模式的场景，最多添加8种颜色，和控制台可设置的参数相同。

iv. 预览效果。

开发定时功能

方案2.0默认使用云端定时功能。App用户设置定时任务后，由云端保存该定时任务，并在时间到达时，给设备发送执行指令。

使用云端定时，设备端不需要任何改动便可以支持。

7.2. 蓝牙Mesh智能灯开发实践

7.2.1. 基于TG7100B的Mesh灯应用固件说明

本文以TG7100B芯片为例，基于蓝牙Mesh SDK中的light_ctl应用示例，开发蓝牙Mesh智能灯设备固件。

背景信息

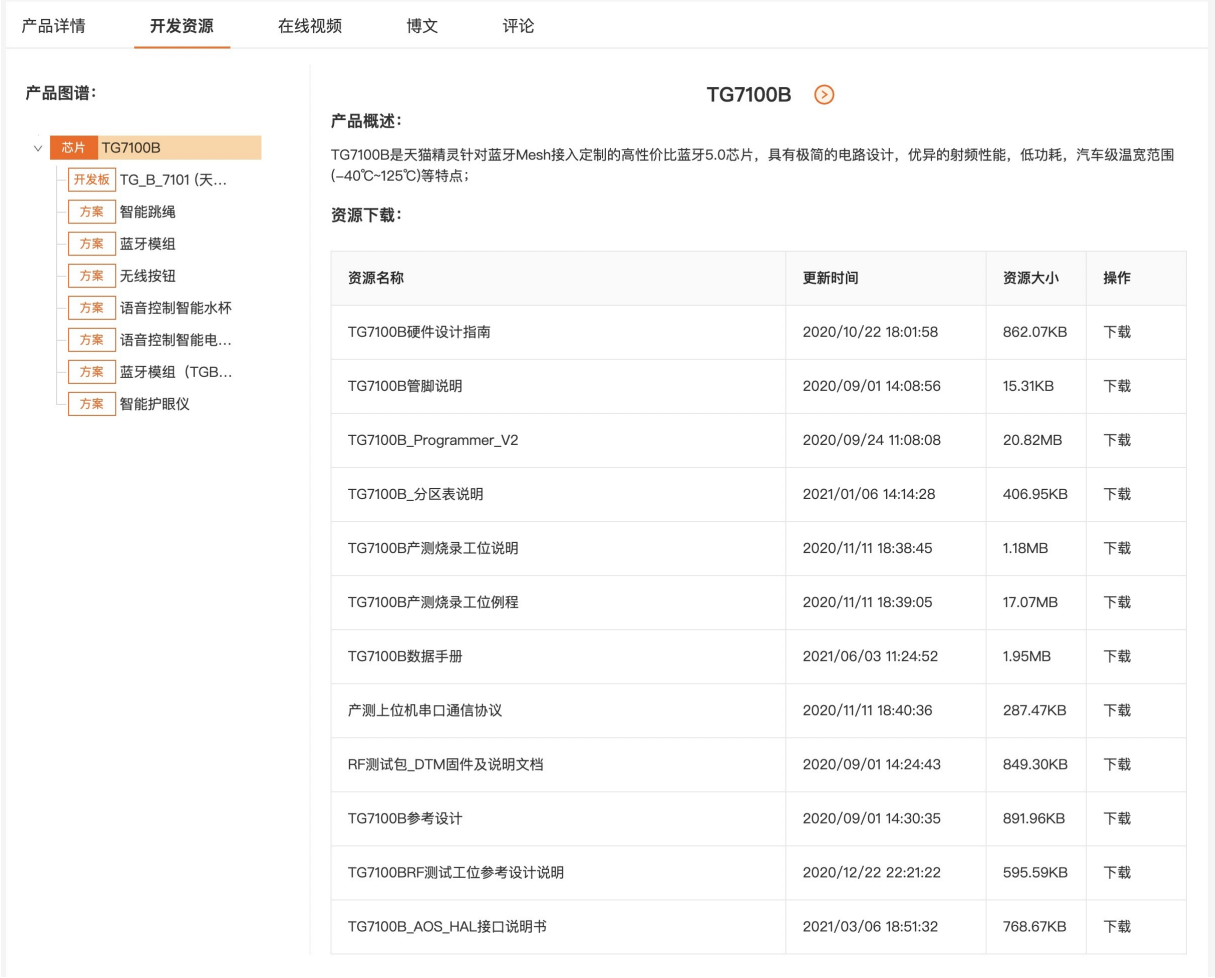
应用示例light_ctl的功能介绍如下。

- 遵循蓝牙Mesh模组软件规范 与蓝牙Mesh设备扩展协议，支持天猫精灵音箱与天猫精灵App配网与控制（创建天猫精灵生态项目产品）与云智能App配网与控制（创建自有品牌项目产品）。

- 支持灯的开关、亮度、色温及场景模式的控制。
- 支持通过生活物联网平台与天猫精灵App进行设备OTA的能力（暂限天猫精灵生态项目）。

TG7100B概述

TG7100B是天猫精灵针对蓝牙Mesh接入定制的高性价比蓝牙芯片，具有极简的电路设计，优异的射频性能，低功耗，汽车级温宽范围(-40℃~125℃)等特点。TG7100B芯片相关文档和软件工具，请参见平头哥开放社区资源介绍



资源名称	更新时间	资源大小	操作
TG7100B硬件设计指南	2020/10/22 18:01:58	862.07KB	下载
TG7100B管脚说明	2020/09/01 14:08:56	15.31KB	下载
TG7100B_Programmer_V2	2020/09/24 11:08:08	20.82MB	下载
TG7100B_分区表说明	2021/01/06 14:14:28	406.95KB	下载
TG7100B产测烧录工位说明	2020/11/11 18:38:45	1.18MB	下载
TG7100B产测烧录工位例程	2020/11/11 18:39:05	17.07MB	下载
TG7100B数据手册	2021/06/03 11:24:52	1.95MB	下载
产测上位机串口通信协议	2020/11/11 18:40:36	287.47KB	下载
RF测试包_DTM固件及说明文档	2020/09/01 14:24:43	849.30KB	下载
TG7100B参考设计	2020/09/01 14:30:35	891.96KB	下载
TG7100BRF测试工位参考设计说明	2020/12/22 22:21:22	595.59KB	下载
TG7100B_AOS_HAL接口说明书	2021/03/06 18:51:32	768.67KB	下载

> **说明**
>
> 目前阿里云和平头哥两个工单系统是各自独立的，请您注意区分。
>
> - 关于TG7100B芯片的驱动、产测、硬件设计、射频等使用问题，请通过平头哥芯片开放社区工单系统反馈。更多介绍，请参见平头哥智能机器人—工单系统使用文档。
> - 基于生活物联网平台蓝牙Mesh SDK的应用开发，例如产品配置、配网、连云、OTA等问题，您可以通过阿里云工单系统反馈。

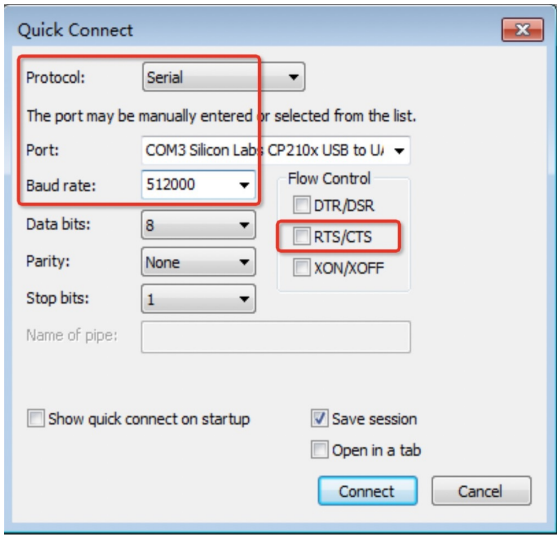
固件编译与烧录

1. 下载SDK。获取下载地址，请参见获取SDK。注意需要下载TG7100B SDK V1.3.4版本。
2. 配置开发环境。详细介绍，请参见准备开发环境。
3. 编译固件。详细介绍，请参见开发设备固件。

> 说明
> - 生成烧录文件：*out/bluetooth.light_ctl@tg7100b/binary/total_image.hexf*
> - 生成OTA文件：*out/bluetooth.light_ctl@tg7100b/binary/fota.bin*

4. 烧录固件。详细介绍，请参见烧录固件。
5. 烧录设备证书。
   - 断开烧写工具TG71XX Programmer.exe，并将开发板的拨码开关置于GND，并按下开发板上的reset键。
   - 打开串口调试工具SecureCRT，选择文件 > 快速连接，并如下图所示设置开发板串口的各参数，默认波特率为512000。

     > 说明 其中端口号与TG71XX Programmer.exe工具自动识别的串口号一致。您也可以右击我的电脑，选择管理 > 系统工具 > 设备管理器 > 端口（COM和LPT）来查看（不同系统下打开设备管理器的操作路径略有差异，请根据您电脑的实际路径操作）。

   - 单击连接。
   - 输入以下命令，烧录设备证书。

     ```
     set_tt <ProductID> <Device Secret> <Device Name>
     ```

     > 说明 该命令中Product id、Device Secret、Device Name（即MAC），三者为添加设备中生成的设备证书。

应用模型配置

Mesh Model的详细说明请参见蓝牙Mesh灯应用Mesh Model说明 。注意蓝牙Mesh SDK中的light_ctl应用示例仅实现了开关、亮度、色温和场景模式控制，固件中默认支持的物模型与Attribute Type如下表所示。

Element	名称	Model	Attribute Type	Attribute Parameter	备注

灯(Primary element)	开关	Generic OnOff Server 0x1000	不适用	不适用	必选
	亮度	Lightness Server 0x1300			可选
	色温	Light CTL Server 0x1303			可选
	模式	Scene Server 0x1203			可选
	错误码	Vendor Model Server 0x01A80000	0x0000	1字节	可选
	开关		0x0100	1字节：0 - 关闭；1 - 打开	必选，状态与Generic OnOff Server Model的开关状态一致，用于设备主动上报开关状态
	亮度		0x0121	2字节：0 ~ 65535	可选，状态与Lightness Server Model的亮度状态一致，用于设备主动上报亮度属性
	色温		0x0122	2字节：800 ~ 20000	可选，状态与Light CTL Server的色温状态一致，用于设备主动上报色温属性
	模式		0xF004	2字节枚举	可选，状态与Scene Server一致，用于设备主动上报模式属性
	事件		0xF009	1字节	• 0x0003 上电事件 • 0x0023 硬件复位事件
	定时开关		0xF010	可变	可选
	时区		0xF01E	1字节：-12 ~ 12	可选
	时间		0xF01F	4字节：标准UNIX时间	可选

模型配置对应的代码在*app/example/bluetooth/light_ctl/light_ctl.c*文件中，如下。

```
/* 灯产品中的SIG通用模型定义 */
static struct bt_mesh_model primary_element[] = {
    BT_MESH_MODEL_CFG_SRV(),     /* 配置 Configuration Server */
    BT_MESH_MODEL_HEALTH_SRV(),  /* Health Server */
    MESH_MODEL_GEN_ONOFF_SRV(&light_elem_stat[0]),    /* 开关 Generic OnOff Server */
    MESH_MODEL_LIGHTNESS_SRV(&light_elem_stat[0]),    /* 亮度 Lightness Server */
    MESH_MODEL_CTL_SRV(&light_elem_stat[0]),          /* 色温 Light CTL Server */
    MESH_MODEL_SCENE_SRV(&light_elem_stat[0]),        /* 场景模式 Scene Server */
};
/* 厂商自定义模型定义 */
static struct bt_mesh_model primary_vendor_element[] = {
    MESH_MODEL_VENDOR_SRV(&light_elem_stat[0]),
};
/* 灯的主Element注册SIG通用模型和厂商自定义模型，其中GENIE_ADDR_LIGHT的定义为灯品类组播地址0xC000*/
struct bt_mesh_elem light_elements[] = {
    BT_MESH_ELEM(0, primary_element, primary_vendor_element, GENIE_ADDR_LIGHT),
};
```

说明 灯的所有模型需要绑定组播地址0xC000（灯品类组播地址，在上示例中指定），0xCFFF（所有产品组播地址，在Mesh SDK Mesh协议组件中默认实现，不需要额外指定）。组播地址定义详情参见设备组播地址

应用层事件处理

应用层事件处理对应的代码在*app/example/bluetooth/light_ctl/light_ctl.c*文件中，如下。

```
static void light_ctl_event_handler(genie_event_e event, void *p_arg)
{
    switch (event)
    {
    case GENIE_EVT_SW_RESET:                                /* 软件复位 */
    {
        light_param_reset();
        light_led_blink(3, 1);
        aos_msleep(3000);
    }
    break;
    case GENIE_EVT_MESH_READY:                              /*Mesh协议栈Ready，可以收发数据*/
    {
        //User can report data to cloud at here
        GENIE_LOG_INFO("User report data");
        light_report_poweron_state(&light_elem_stat[0]);
    }
    break;
    case GENIE_EVT_SDK_MESH_PROV_SUCCESS:                   /* 配网成功 */
    {
        light_led_blink(3, 0);                              /* 配网成功闪灯提示 */
    }
    break;
#ifdef CONFIG_MESH_MODEL_TRANS
    case GENIE_EVT_USER_TRANS_CYCLE:
#endif
```

```
    case GENIE_EVT_USER_ACTION_DONE:                    /* 灯效变化结束 */
    {
        sig_model_element_state_t *p_elem = (sig_model_element_state_t *)p_arg;
        light_update(p_elem);
        if (event == GENIE_EVT_USER_ACTION_DONE)
        {
            light_save_state(p_elem);
        }
    }
    break;
    case GENIE_EVT_SIG_MODEL_MSG:                       /* 收到下行SIG Model 消息 */
    {
        sig_model_msg *p_msg = (sig_model_msg *)p_arg;
        if (p_msg)
        {
            GENIE_LOG_INFO("SIG mesg ElemID(%d)", p_msg->element_id);
        }
    }
    break;
    case GENIE_EVT_VENDOR_MODEL_MSG:                    /* 收到下行Vendor Model 消息 */
    {
        genie_transport_model_param_t *p_msg = (genie_transport_model_param_t *)p_arg;
        if (p_msg && p_msg->p_model && p_msg->p_model->user_data)
        {
            sig_model_element_state_t *p_elem_state = (sig_model_element_state_t *)p_msg->p
_model->user_data;
            GENIE_LOG_INFO("ElemID(%d) TID(%d)", p_elem_state->element_id, p_msg->tid);
        }
    }
    break;
#ifdef CONIFG_GENIE_MESH_USER_CMD                       /* 用于实现用户自定义串口协议 */
    case GENIE_EVT_DOWN_MSG:
    {
        genie_down_msg_t *p_msg = (genie_down_msg_t *)p_arg;
        //User handle this msg,such as send to MCU     /* 在此处可以按自定义串口协议将数据转发给M
CU */
        if (p_msg)
        {
        }
    }
    break;
#endif
#ifdef MESH_MODEL_VENDOR_TIMER
    case GENIE_EVT_TIMEOUT:                             /* TimerOnOff本地定时超时处理入口 */
    {
        vendor_attr_data_t *pdata = (vendor_attr_data_t *)p_arg;
        //User handle vendor timeout event at here
        if (pdata)
        {
            light_ctl_handle_order_msg(pdata);          /* 此处执行灯开关操作 */
        }
    }
    break;
#endif
```

```
    default:
        break;
```

串口命令说明

以下串口命令可以用于开发调试。

命令名称	命令说明	使用参考（示例）
set_tt	设备蓝牙Mesh设备证书	set_tt 5297793 0c51b11c6ec78b52b803b3bbaae64fba486e704a5bf6
get_tt	查看蓝牙Mesh设备证书	无参数
get_info	查看版本和MAC等信息	无参数
reboot	系统重启	无参数
reset	设备复位	无参数
mesg	发送Mesh数据	mesg D4 1 F000 000101

说明 mesg命令参数说明：

- 第一个参数D4就是Vendor Message Attribute Indication消息Opcode的首字节缩写，其他有D3、CE及CF等。
- 第二个参数是发送模式和重发次数参数：
  - 0：表示不重发；
  - 1-252：表示重发次数；
  - 253：表示使用payload的第一个字节作为时间间隔参数，以100ms为单位，例如：mesg D4 253 F000 030201 表示300毫秒发一次0201，mesg D4 253 F000 1e0201是3秒一次0201；
  - 254：表示收到回复或者发送超时就再次发送；
  - 255：表示每秒自动发送一次。
- 第三个参数是接收者地址，必须是四个字符，如果设置为0000会默认使用Mesh网关组播地址F000。
- 第四个参数是发送的内容，例如000101就是发送0x00，0x01，0x01因此必须是偶数个0-f之间的字符。

固件宏定义说明

用户可配置的宏定义在文件*app/example/bluetooth/light_ctl/light_ctl.mk*与*genie_service/genie_service.mk*中。部分重要宏定义说明如下。

宏定义的名称	功能说明
CONFIG_BT_MESH_GATT_PROXY	支持Proxy功能

CONFIG_BT_MESH_PB_GATT	支持手机配网功能
CONFIG_BT_MESH_RELAY	支持中继功能
CONFIG_GENIE_OTA	支持手机OTA功能
CONFIG_GENIE_RESET_BY_REPEAT	支持连续上电五次进入配网状态功能
PROJECT_SW_VERSION	配置版本号，OTA功能使用，int32数据类型
CONFIG_PM_SLEEP	支持低功耗功能
CONFIG_GENIE_MESH_GLP	支持GLP模式的低功耗功能
CONFIG_DEBUG	支持BT_xxx日志输出
CONFIG_BT_DEBUG_LOG	支持BT_DBG日志输出
MESH_DEBUG_PROV	支持配网日志输出
MESH_DEBUG_TX	支持Access层发送Mesh数据日志输出
MESH_DEBUG_RX	支持Access层接收Mesh数据日志输出

7.2.2. 蓝牙Mesh灯应用Mesh Model说明

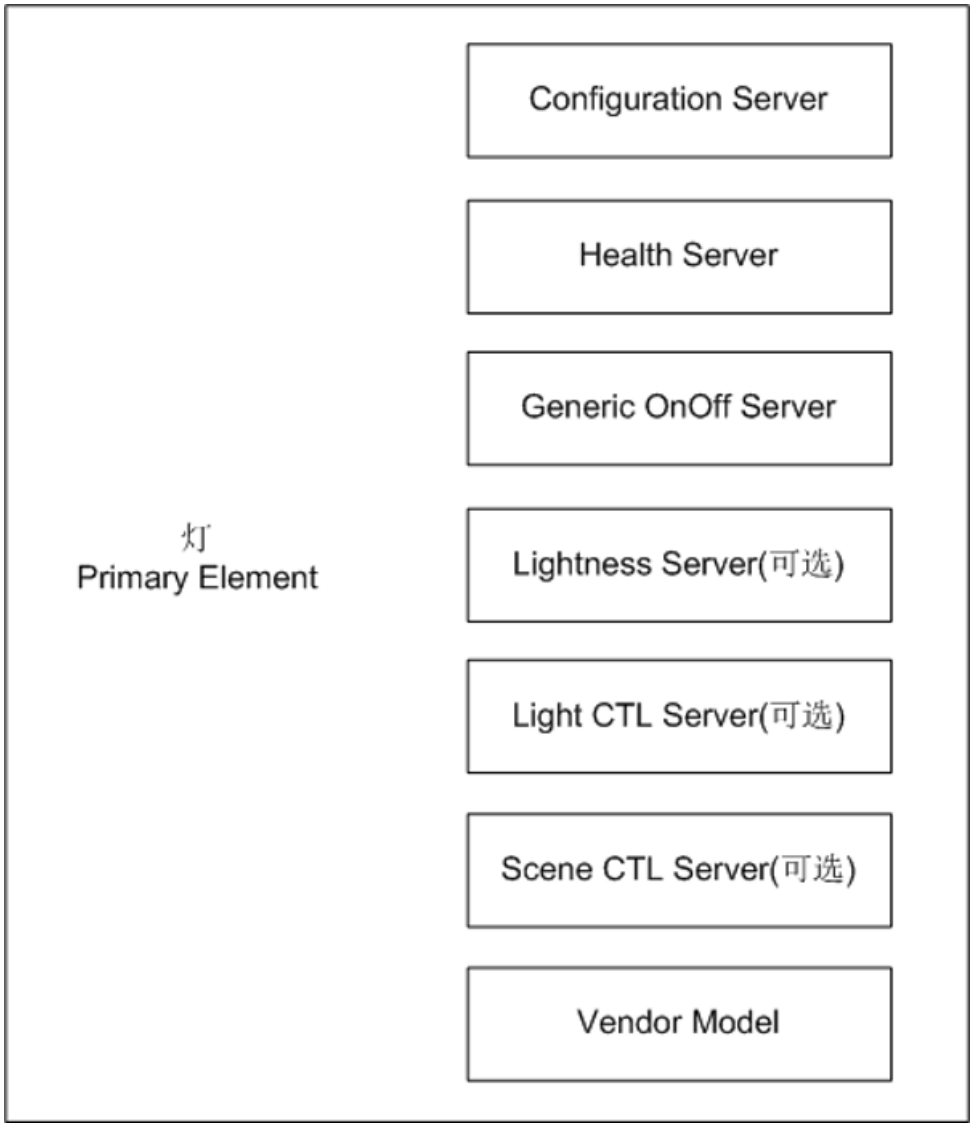
本文在蓝牙技术联盟发布的Mesh Model Specification的基础上，介绍生活物联网平台蓝牙Mesh灯应用的Mesh Model配置。

背景信息

物模型是对设备是什么，能做什么的描述，包括设备的属性（properties）、服务（services）、事件（events）等。生活物联网平台通过物模型定义产品功能。蓝牙Mesh设备通过Mesh Model来实现物模型。

Mesh Model配置

生活物联网平台蓝牙Mesh灯产品的Mesh Model配置如下图所示。当蓝牙mesh智能灯作为一个独立的产品时，一般是一个元素（Element）。配置服务模型（Configuration Server）与健康服务模型（Health Server）是所有设备的主元素（Primary Element）都要求支持的，非主元素（Secondary Elements）无需支持。灯的开关、亮度、色温、场景模式这几个主要属性，通过SIG Model来实现，请参见Mesh Model Specification。除了上述几个主要属性之外的其他属性（如颜色等）、服务和事件等都通过Vendor Model即阿里巴巴厂商自定义模型实现，请参见蓝牙Mesh设备扩展协议。



Element	属性	模型与SIG Model ID
灯 (Primary Element)	开关	Generic OnOff Server（0x0100）
	亮度	Light Lightness Server(0x1300)
	色温	Light CTL Server(0x1303)
	场景模式	Scene Server(0x1203)
	其他属性（如颜色等）、服务和事件	Alibaba Vendor Model Server(0x01A80000)

Mesh Model与Opcode

Model	Message	Opcode	说明
Generic OnOff Server	Generic OnOff Get	0x8201	获取开关状态
	Generic OnOff Set	0x8202	设置开关
	Generic OnOff Set Unacknowledged	0x8203	设置开关（无确认）
	Generic OnOff Status	0x8204	开关状态（设备上报）
Lightness Server	Light Lightness Get	0x824B	获取亮度状态
	Light Lightness Set	0x824C	设置亮度
	Light Lightness Set Unacknowledged	0x824D	设置亮度（无确认）
	Light Lightness Status	0x824E	亮度状态（设备上报）
Light CTL Server	Light CTL Get	0x825D	获取色温状态
	Light CTL Set	0x825E	设置色温
	Light CTL Set Unacknowledged	0x825F	设置色温（无确认）
	Light CTL Status	0x8260	色温状态（设备上报）
Scene Server	Scene Get	0x8241	获取场景模式状态
	Scene Set	0x8242	设置场景模式
	Scene Set Unacknowledged	0x8243	设置模式（无确认）
	Scene Status	0x5E	模式状态（设备上报）
Vendor Model	Vendor Message Attribute Get	0xD001A8	获取一个或多个属性值
	Vendor Message Attribute Set	0xD101A8	设置一个或多个属性值
	Vendor Message Attribute Set Unacknowledged	0xD201A8	设置一个或多个属性值（无确认）
	Vendor Message Attribute Status	0xD301A8	响应获取与设置指令回复的属性状态
	Vendor Message Attribute Indication	0xD401A8	主动属性上报
	Vendor Message Attribute Confirmation	0xD501A8	主动属性上报的确认

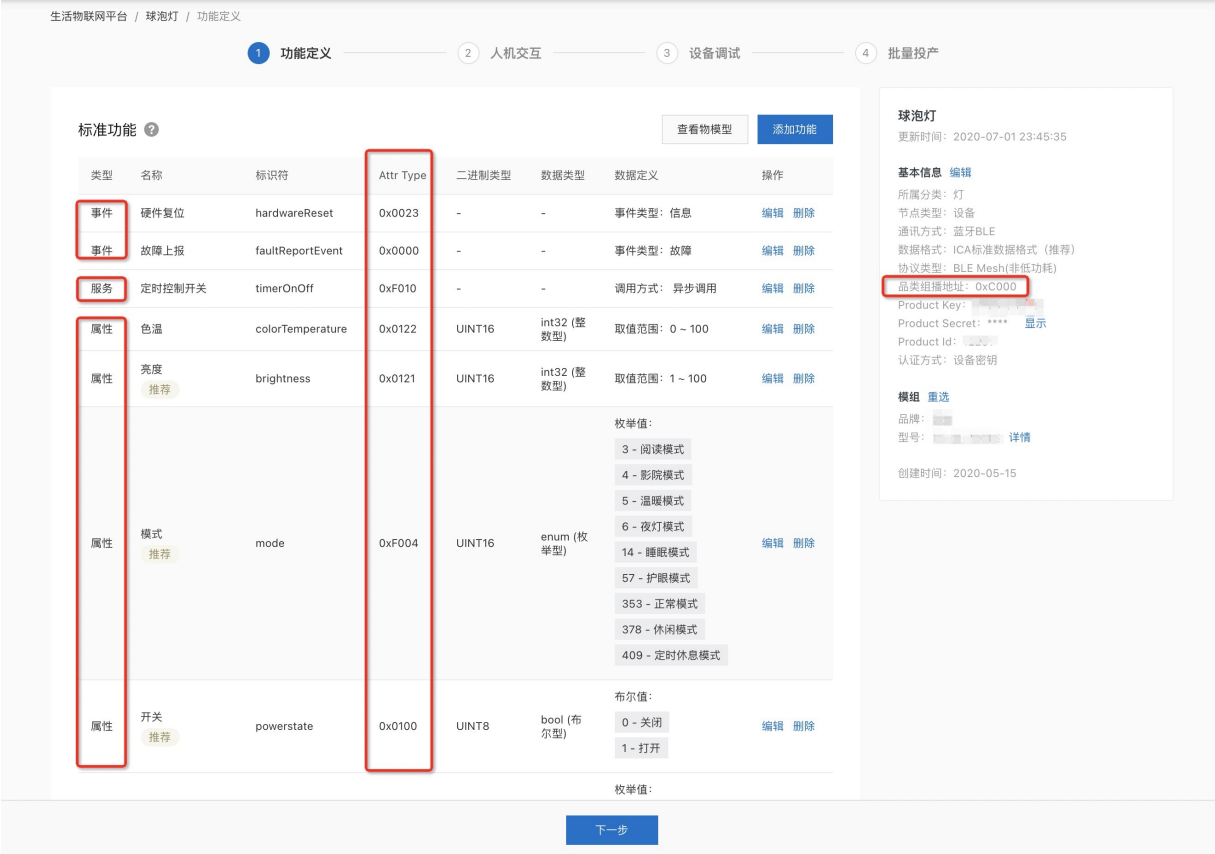
	Vendor Message Attribute Indication To GW	0xDE01A8	设备给网关的上报
	Vendor Message Attribute Confirmation From GW	0xDF01A8	网关给设备回复的确认
	Vendor Message Transparent msg	0xCF01A8	厂商自定义透传数据
	Vendor Message Transparent Indication	0xCE01A8	自定义主动上报
	Vendor Message Transparent ACK	0xCD01A8	自定义上报回复

Mesh Model与物模型

Element	名称	Model	Attribute Type	Attribute Parameter	备注
灯（Primary	开关	Generic OnOff Server 0x1000	不适用	不适用	必选
	亮度	Lightness Server 0x1300			可选
	色温	Light CTL Server 0x1303			可选
	模式	Scene Server 0x1203			可选
	错误码		0x0000	1字节	可选
	开关		0x0100	1字节：0 - 关闭；1 - 打开	必选，状态与Generic OnOff Server Model的开关状态一致，用于设备主动上报开关状态
	电量		0x0104	1字节：0～100 百分比	可选
	亮度		0x0121	2字节：0～65535	可选，状态与Lightness Server Model的亮度状态一致，用于设备主动上报亮度属性
	色温		0x0122	2字节：800～20000	可选，状态与Light CTL Server的色温状态一致，用于设备主动上报色温属性

灯（Primary element）	颜色	Vendor Model Server 0x01A80000	0x0123	6字节	可选
	延时关闭时长		0x0133	2字节：0～65535 秒	可选
	待机亮度		0x01F2	2字节：0～65535	可选
	背光灯		0x0533	1字节：0 - 关闭；1 - 打开	可选
	主灯		0x0534	1字节：0 - 关闭；1 - 打开	可选
	夜灯		0x0572	1字节：0 - 关闭；1 - 打开	可选
	模式		0xF004	2字节枚举	可选，状态与Scene Server一致，用于设备主动上报模式属性
	事件		0xF009	1字节	• 0x0003 上电事件 • 0x0023 硬件复位事件
	定时开关		0xF010	可变	可选
	时区		0xF01E	1字节：-12～12	可选
	时间		0xF01F	4字节：标准UNIX时间	可选

在创建产品并定义产品功能阶段，添加的属性、服务和事件都会显示对应的AttrType和数据定义与取值范围。如果添加的属性、服务和事件没有合法的AttrType值，说明该项不适用于蓝牙Mesh设备，请更换或者联系aligenie.iot@list.alibaba-inc.com添加。



7.2.3. 开发自有品牌项目蓝牙Mesh灯产品

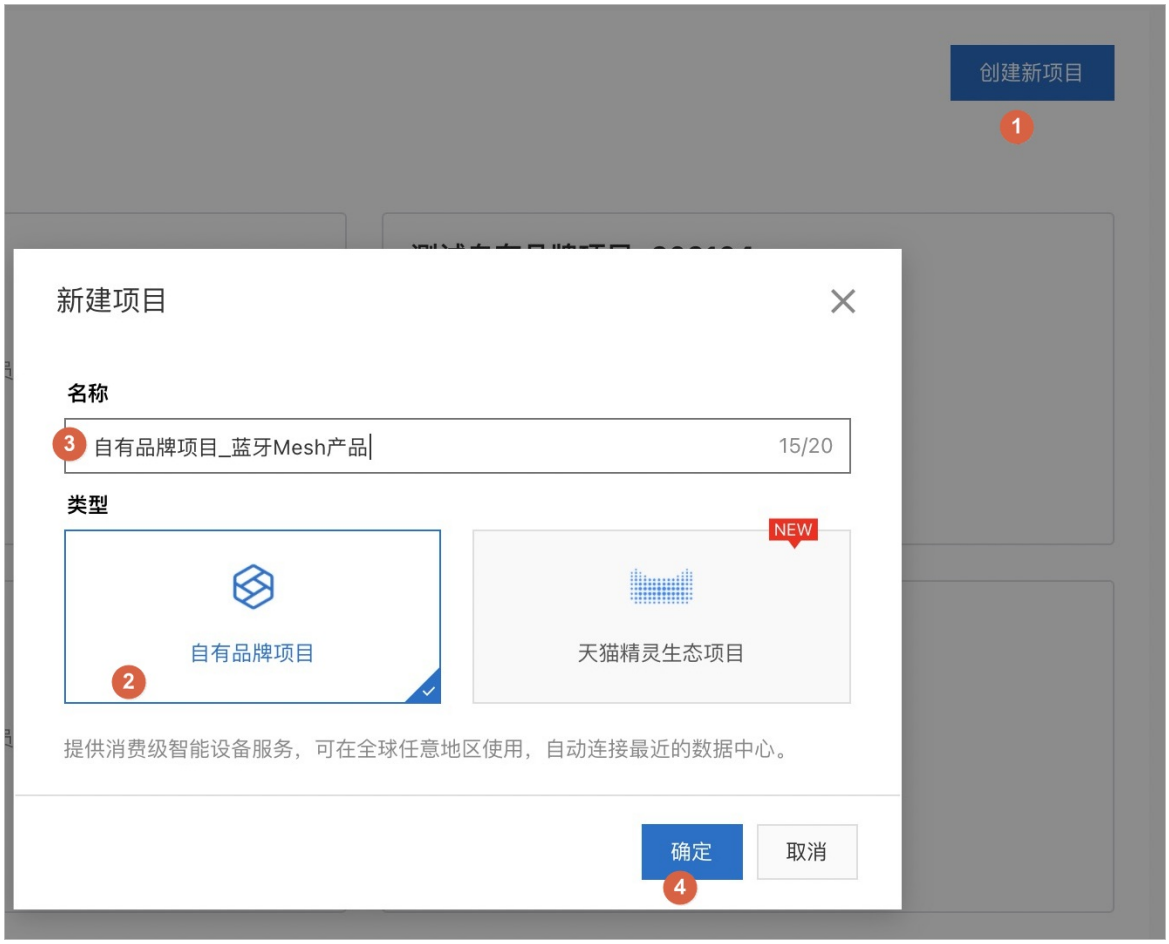
本文介绍如何开发一个自有品牌项目蓝牙Mesh灯产品，包括使用开发板与云智能App调试配网和控制的过程。

背景信息

生活物联网平台自有品牌项目已经支持创建蓝牙Mesh产品，如果基于TG7100B芯片开发，可以按照基于TG7100B的Mesh灯应用固件说明中的介绍，下载V1.3.4版本SDK、编译和运行固件。如果基于TG7120B或者TG7121B芯片开发，注意下载和使用V1.3.5以上版本SDK。在创建与产品后，将设备证书写入设备，通过云智能App（V3.8.0以上版本）可以对蓝牙Mesh产品进行配网和控制。

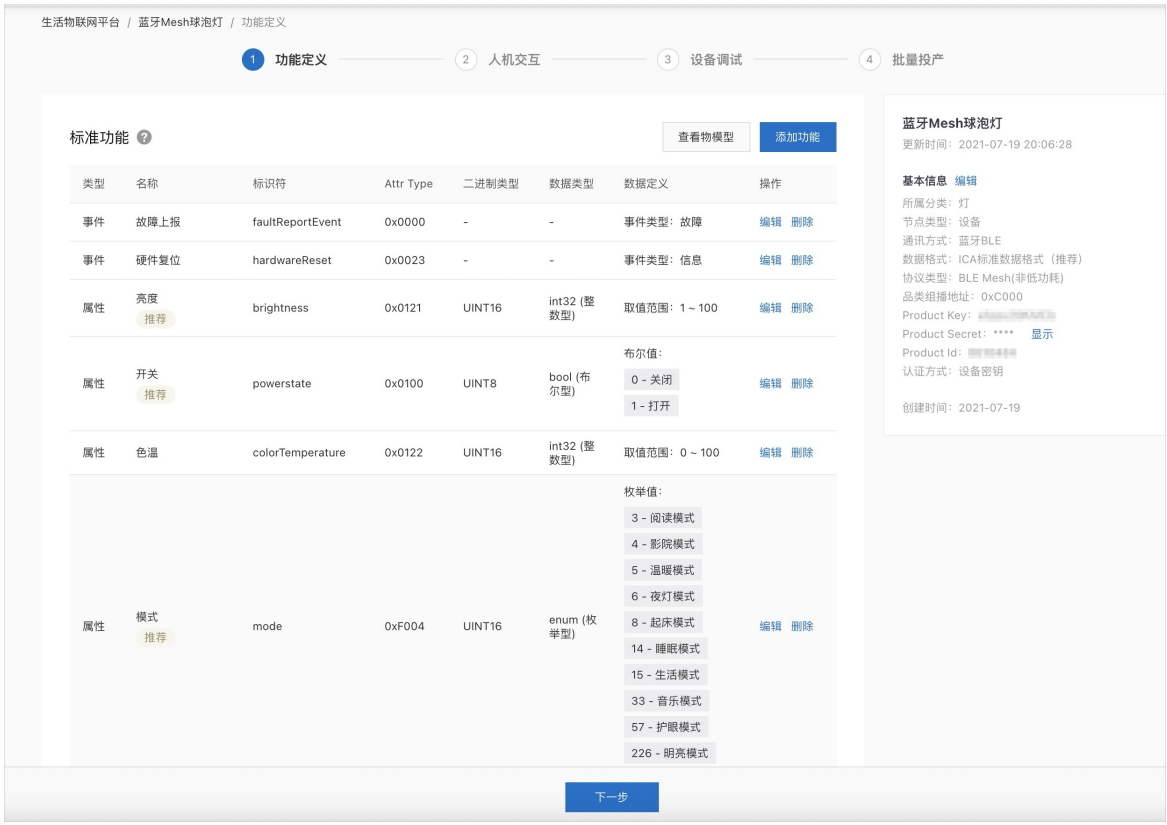
创建与配置产品

1. 创建一个自有品牌项目。



2. 创建产品，在产品中填写产品名称，“所属品类“选择为“电工照明/灯“，“节点类型”选择为“设备“，“是否接入网关”选择“是”；“接入网关协议”选择为“BLE Mesh”。

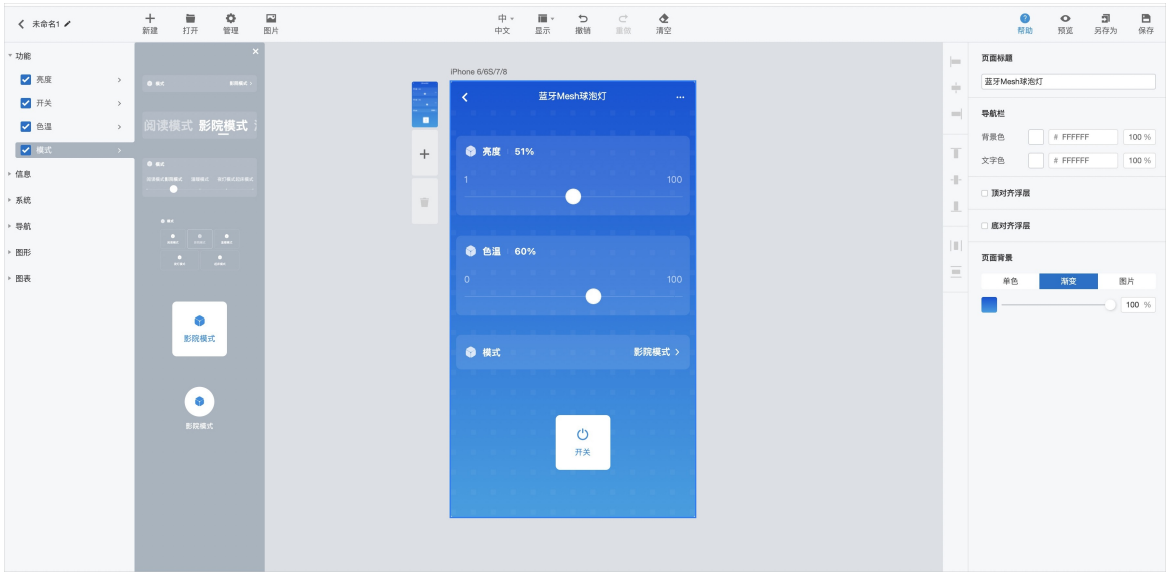
3. 定义产品功能，注意选择硬件复位事件、开关属性、亮度属性、色温属性、模式属性。



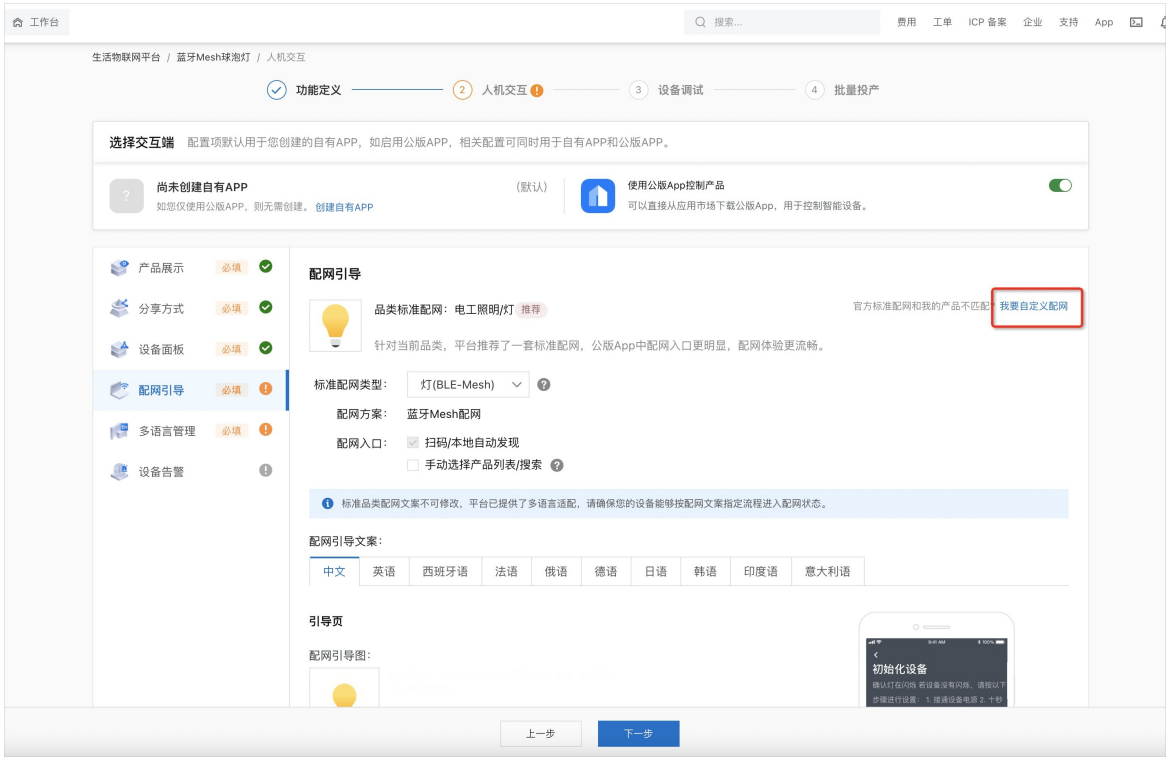
4. 配置产品的人机交互，需要打开“使用公版App控制产品“，后续可以通过云智能App进行验证，需要详细配置和确认完六步。其中“分享方式”无须选择，目前仅支持抢占式。

5. 配置设备面板。

light_ctl应用仅支持开关、亮度、色温、模式控制，可以搭建一个简单的面板，如下所示。

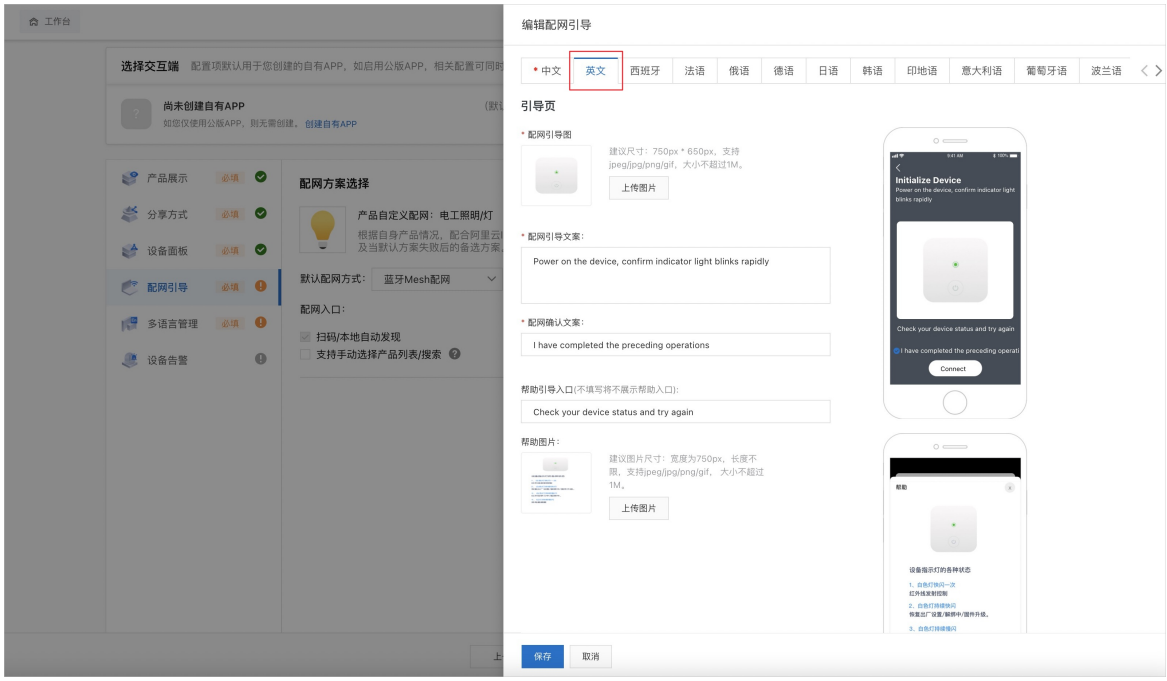


6. 配网引导配置，点击右上角“我要自定义配网”，切换配置界面。

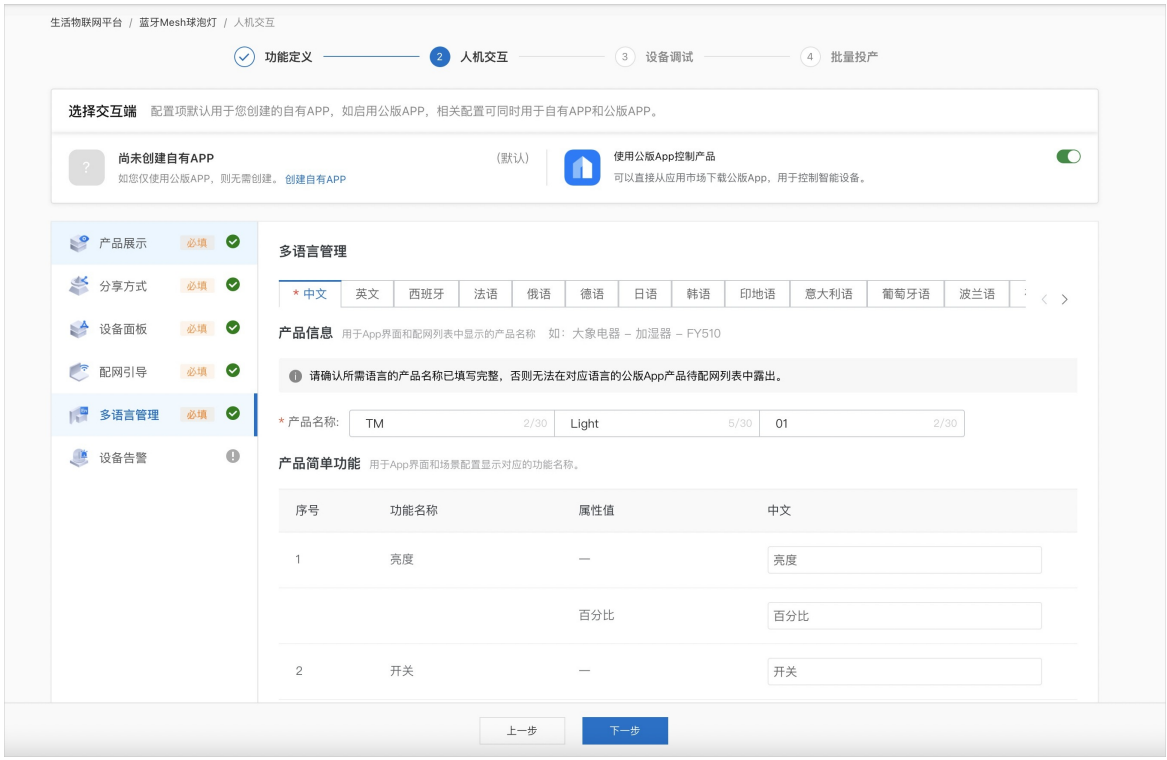


确认配网方式为“蓝牙Mesh配网”，并且点击下述配置，可以配置具体配网引导文案和图片，详见下图。配网引导内容，会有默认的文案，也可以自定义填写。遍历各个语言，会自动生成多语言的默认文案。

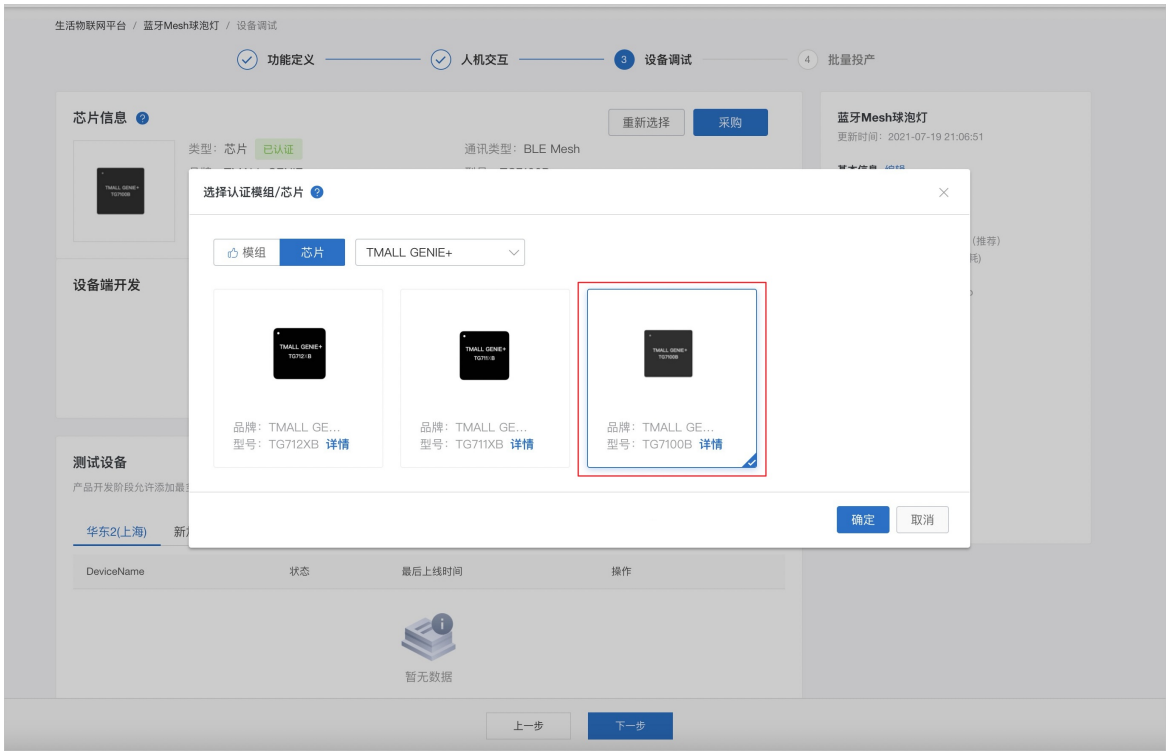

选择交互端
配置项默认用于您创建的自有APP，如启用公版APP，相关配置可同时用于自有APP和公版APP。
尚未创建自有APP
如您仅使用公版APP，则无需创建。创建自有APP
(默认)
使用公版App控制产品
可以直接从应用市场下载公版App，用于控制智能设备。
产品展示 必填
分享方式 必填
设备面板 必填
配网引导 必填
多语言管理 必填
设备告警
配网方案选择
产品自定义配网：电工照明/灯
根据自身产品情况，配合阿里云IoT提供的Wi-Fi配网方案，设置优先进入的默认配网方案，以及当默认方案失败后的备选方案。
自定义配网体验不好？使用品类标准配网
默认配网方式：
蓝牙Mesh配网
备选配网方式：
无
零配方式（需设备支持）：点击配置
配网入口：
扫码/本地自动发现
支持手动选择产品列表/搜索
*蓝牙Mesh配网
未配置
上一步
下一步


编辑配网引导
中文
英文
西班牙
法语
俄语
德语
日语
韩语
印地语
意大利语
葡萄牙语
波兰语
引导页
* 配网引导图
建议尺寸：750px * 650px，支持jpeg/jpg/png/gif，大小不超过1M。
上传图片
* 配网引导文案：
Power on the device, confirm indicator light blinks rapidly
* 配网确认文案：
I have completed the preceding operations
帮助引导入口(不填写将不展示帮助入口)：
Check your device status and try again
帮助图片：
建议图片尺寸：宽度为750px，长度不限，支持jpeg/jpg/png/gif，大小不超过1M。
上传图片
Initialize Device
Power on the device, confirm indicator light blinks rapidly
Check your device status and try again
Connect
保存
取消

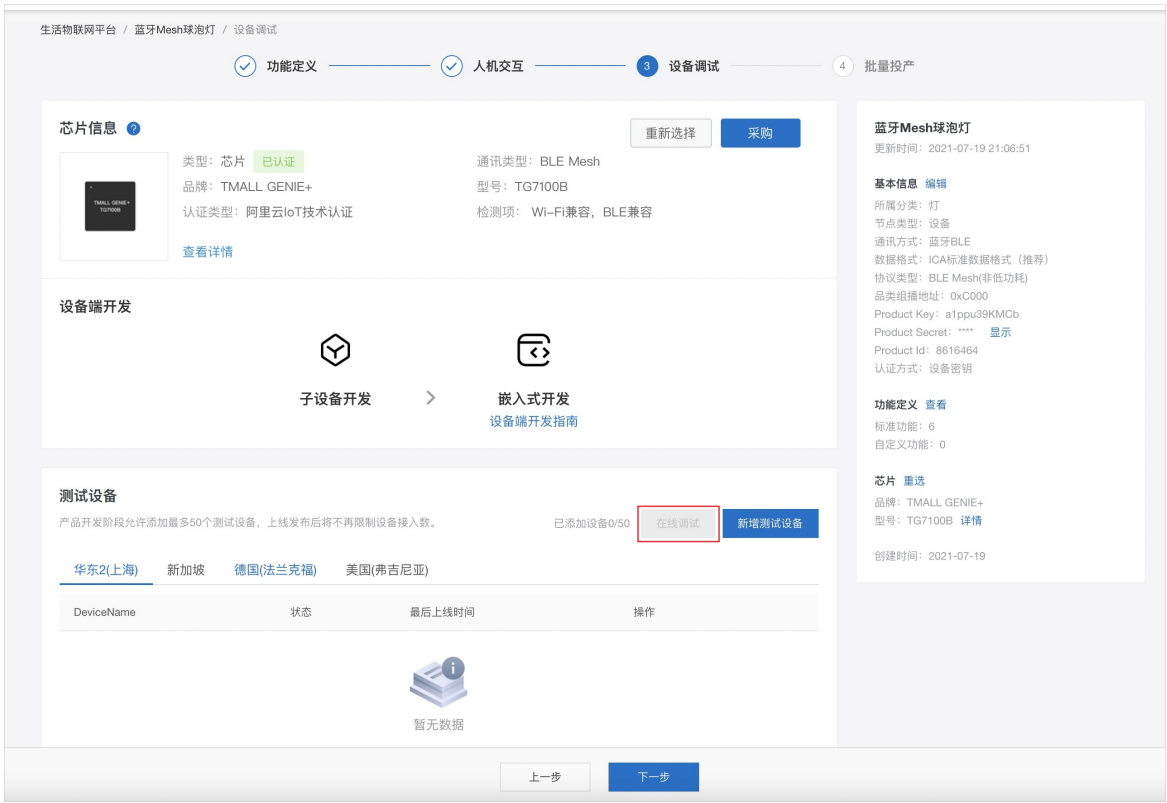
7. 配置多语言，配置完成后保存。



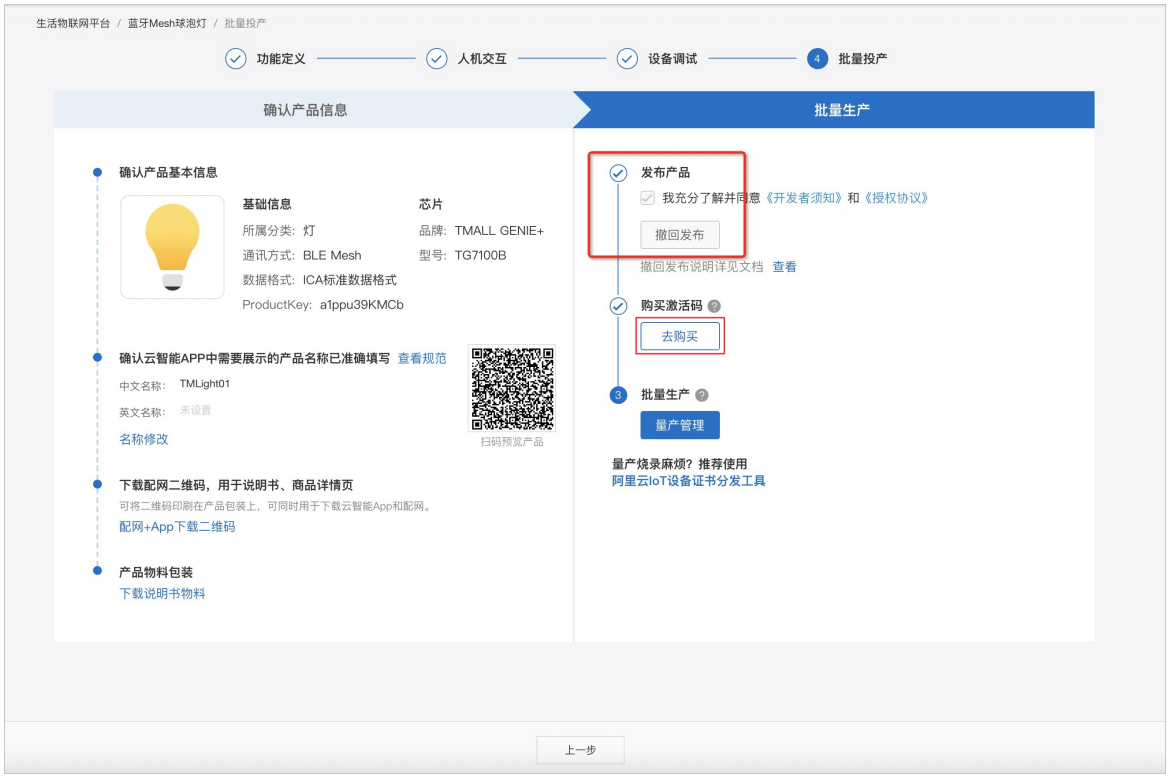
8. 设备调试配置。首先选择认证模组/芯片，可以根据实际使用的芯片选择。



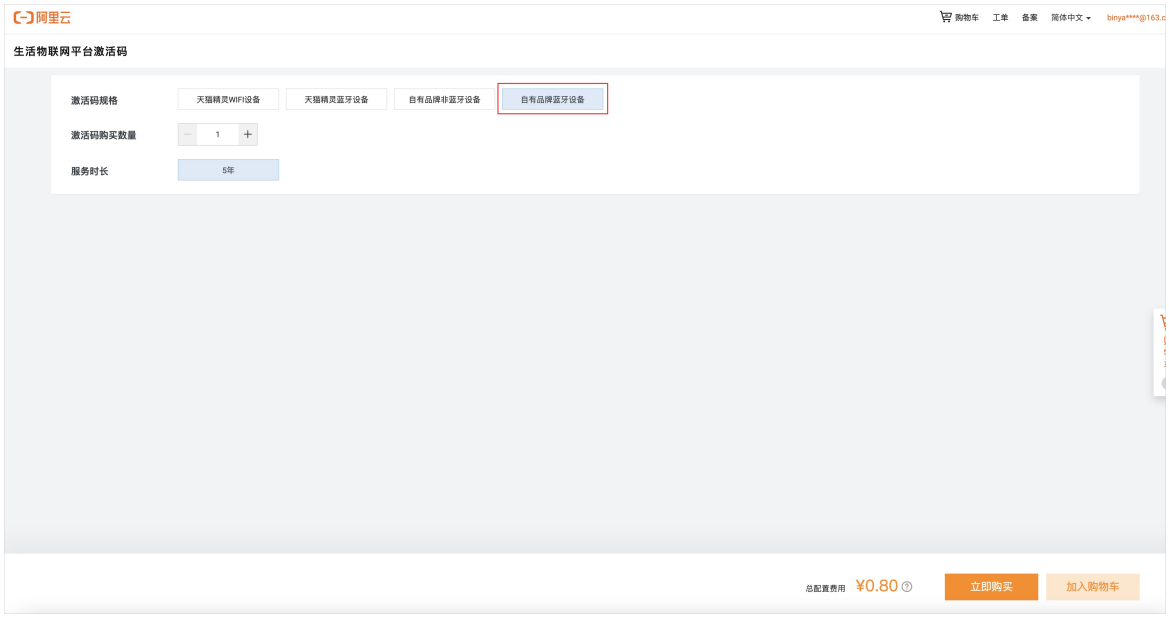
目前自有品牌项目尚无Mesh网关的情况下，暂不支持设备“在线调试”功能。



9. 发布产品与购买激活码。

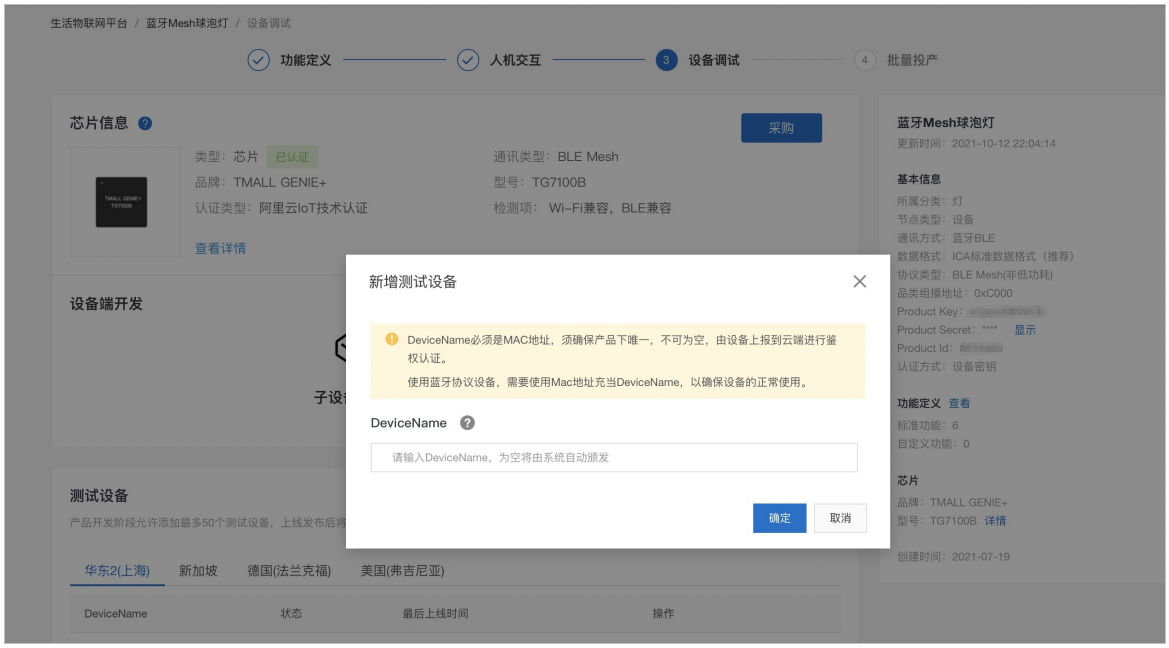


购买激活码时，注意选择“自有品牌蓝牙设备“。



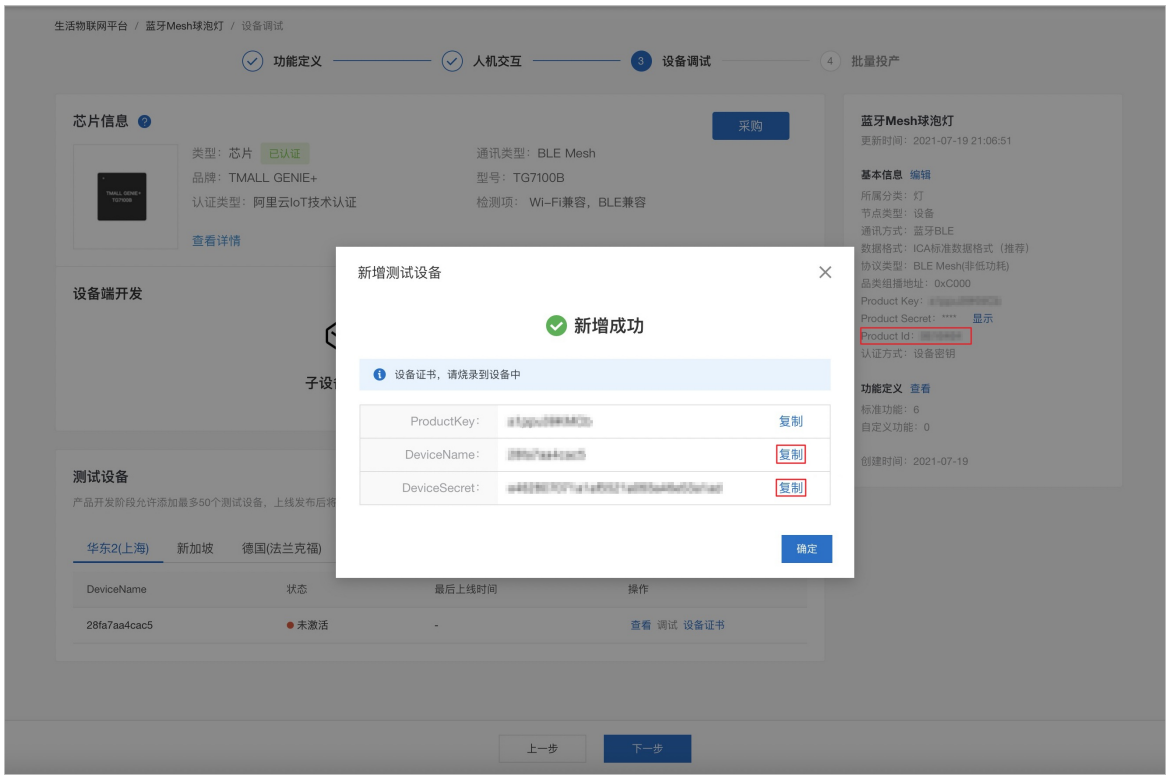
生成与烧录设备证书

1. 在设备调试页面，单击**新增测试设备**生成测试设备证书，可以选择输入合法MAC地址作为DeviceName，或者不输入由平台自动分配。



说明 蓝牙Mesh设备的DeviceName与MAC地址需要保持一致。当输入合法MAC地址作为DeviceName时，调试阶段要避免同一MAC地址在不同产品下作为DeviceName。即要注意保证MAC地址的唯一性，避免造成问题。

2. 如下图拷贝Product ID，Device Secret与Device Name。



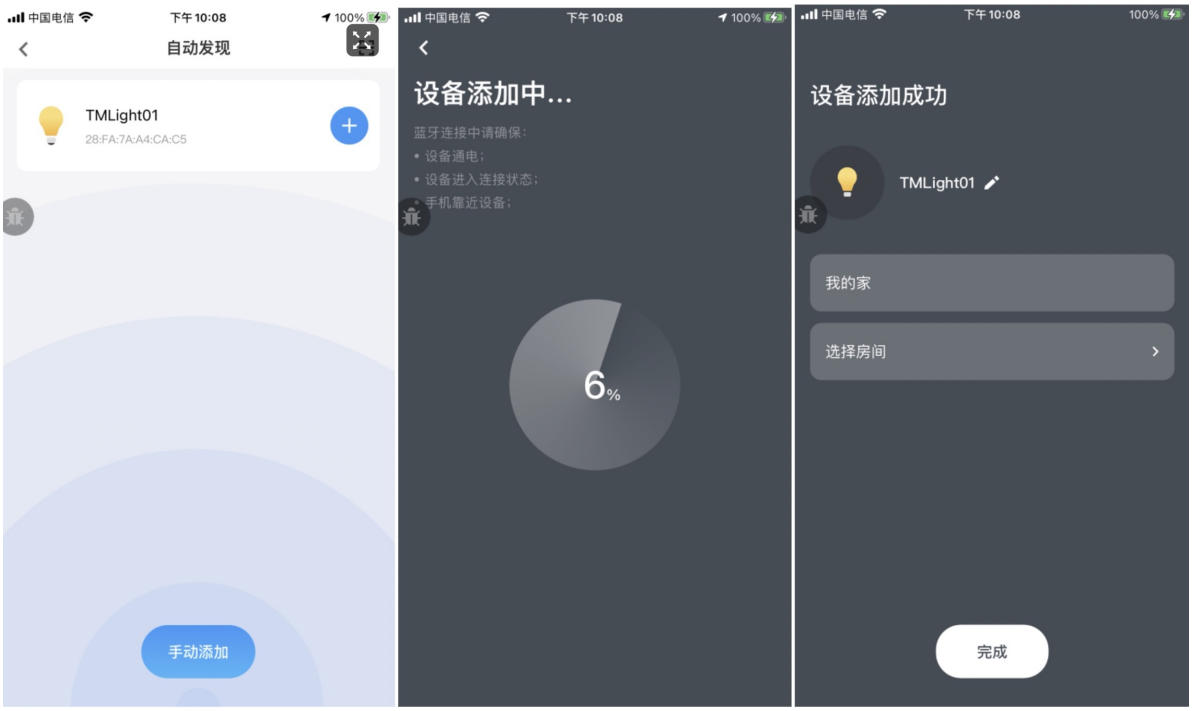
3. 通过set_tt命令，将设备证书烧录到开发板。

```
set_tt 8xxxxx4 exxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxd 28fa7aa4cac5      (输入命令)
8xxxxx4 exxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxd 28fa7aa4cac5            (返回结果)
```

4. 通过reboot命令重启设备，设备进入待配网状态。

调试云智能App添加设备

- 进入云智能App，打开“自动发现“页面，添加设备，整个流程如下图所示。



- App配网过程关键的设备端日志。

- Provision关键日志。

```
DEVICE:light
APP VER:1.3.1
GenieSDK:V1.3.4-tg7100b
PROUDUCT:ALI_AOS_TG7100B
MAC:28:FA:7A:A4:CA:C5
ais pre init
[000609]<I> GenieE:3
[000638]<I> SigE:11
[000648]<I> SigE:12
[000708]<I> GenieE:12
[proxy_connected]conn(0x2000b63c) reason 0x00    /* App与设备之间建立proxy连接*/
[000939]<I> GenieE:11
[001604]<I> SigE:14
[001614]<I> SigE:4
[001624]<I> GenieE:55
[light_save_state]save 1 58982 20000       /*默认开关为1，亮度为58982（90%），色温为20000（
100%）*/
[ALI_PROV][D] prov_invite , 1--->2
[001960]<I> GenieE:20
[ALI_PROV][D] prov_start , 2--->3
[ALI_PROV][D] prov_pub_key , 3--->4
[ALI_PROV][D] prov_confirm , 4--->5
[ALI_PROV][D] prov_random , 5--->6
[ALI_PROV][D] prov_data , 6--->7
[002606]<I> GenieE:21
```

- 设备绑定关键日志。

```
......
[004731]<I> GenieE:30
......
[004965]<I> GenieE:23
......
[005309]<I> GenieE:37
```

- 设备全属性上报日志：开关（Attr Type 0x0100）属性值0x01（开），亮度（Attr Type 0x0121）属性值为0xE666（相比最大值0xFFFF，0xE666相当于90%），色温（Attr Type 0x0122）属性值为0x4E20（最大值20000，相当于100%），模式（Attr Type 0xF004）属性值为0x0000。

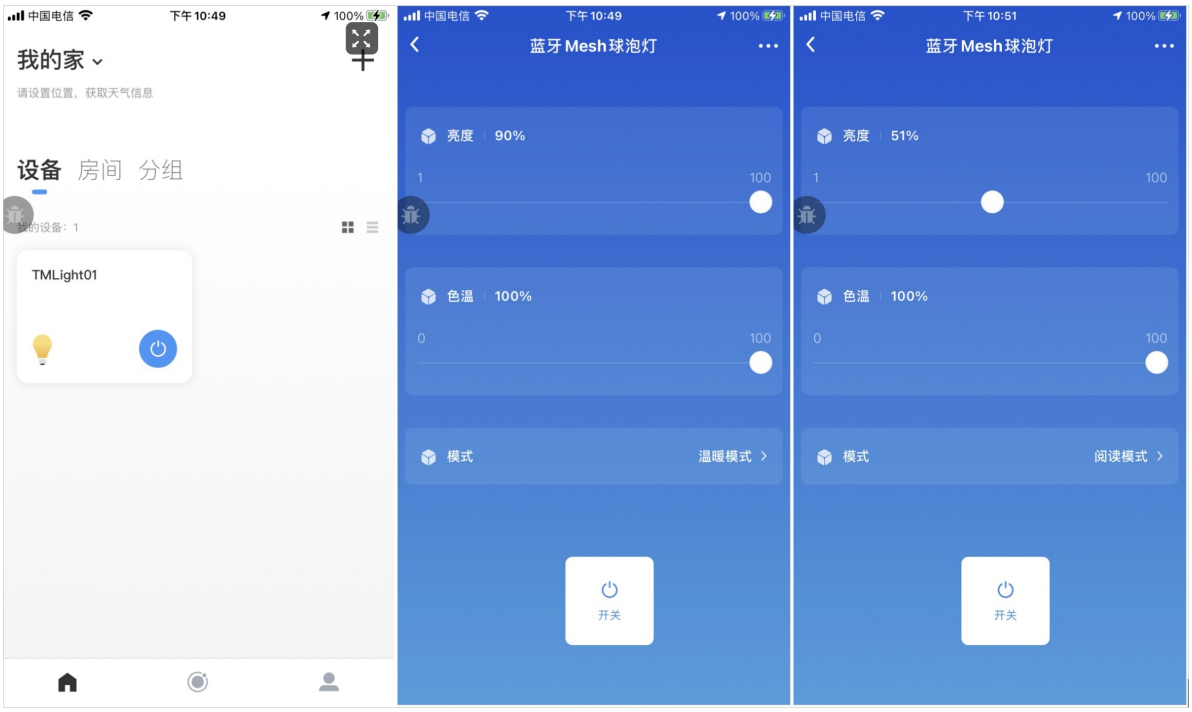
```
[009552]<I> User report data
[009566]<I> SendTID(82)
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 19
[TX] D4 A8 01 82 00 01 01 21 01 66 E6 22 01 20 4E 04
[TX] F0 00 00
```

- 设备上电事件日志：事件上报（Attr Type 0xF009），上电事件（0x03）。

```
[009787]<I> SendTID(84)
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 7
[TX] D4 A8 01 84 09 F0 03
```

调试云智能App控制设备

- 在云智能App控制设备，所下图所示。



- 设备端关键日志。
  - 开灯操作。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x8202
[RX] Payload size: 4
[RX] 01 C0 41 00
[116958]<I> SigE:20
[116968]<I> SIG mesg ElemID(0)
+MESHMSGRX:6,820201C04100    /*App下发开灯操作：Generic OnOff Set消息，Opcode 0x8202, OnOff 0x01*/
[117000]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 5
[TX] 82 04 01 01 41          /*设备回复Generic OnOff Status消息，Opcode 0x8204,Target OnOff 0x01*/
[117065]<I> SigE:1
[117075]<I> SigE:12
[117106]<I> SigE:14
[117116]<I> SigE:4
[117126]<I> SigE:5
[117136]<I> SendTID(85)
[117148]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 11
[TX] D4 A8 01 85 00 01 01 21 01 66 E6   /* 设备上报Indication，TID 0x85，开关0x01，亮度0xE666（即90%）*/
[117223]<I> GenieE:55
[117342]<I> RX Info
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] 85                      /* 设备收到Confirmation，TID 0x85，即上条上报对应的确认 */
```

- 关灯操作。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x8202
[RX] Payload size: 4
[RX] 00 C1 41 00
[126978]<I> SigE:20
[126988]<I> SIG mesg ElemID(0)
+MESHMSGRX:6,820200C14100    /*App下发关灯操作：Generic OnOff Set消息，Opcode 0x8202, OnO
ff 0x00*/
[127021]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 5
[TX] 82 04 01 00 41          /*设备回复Generic OnOff Status消息，Opcode 0x8204,Target OnO
ff 0x00*/
[127086]<I> SigE:1
[127095]<I> SigE:12
[127946]<I> SigE:14
[127956]<I> SigE:4
[127966]<I> SigE:5
[127976]<I> SendTID(86)
[127988]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 11
[TX] D4 A8 01 86 00 01 00 21 01 00 00    /* 设备上报Indication，TID 0x86，开关0，亮度0 */
[128063]<I> GenieE:55
[light_save_state]save 0 58982 20000     /* 保存开关0 */
[128259]<I> RX Info
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] 86                     /* 设备收到Confirmation，TID 0x86，即上条上报对应的确认 */
```

- 设置灯亮度。平台通过下发Light Lightness Set（Opcode 0x824C）消息来设置灯的亮度。Light Lightness Set消息中的字段Lightness（16bit）表示亮度，0xFFFF（65535）表示最大亮度，即100%。示例中的0xE666(58982)为相对于0xFFFF（65535）的90%，0x828F（33423）为相对于0xFFFF（65535）的51%。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x824C
[RX] Payload size: 5
[RX] 8F 82 C3 41 00
[146540]<I> SigE:20
[146550]<I> SIG mesg ElemID(0)
+MESHMSGRX:7,824C8F82C34100  /*下发亮度调至51%：Light Lightness Set消息，Opcode 0x824C,Li
ghtness 0x828F*/
[146584]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 7
[TX] 82 4E 66 E6 8F 82 41    /*设备回复Light Lightness Status消息，Opcode 0x824E,亮度从0x
E666调到0x828F*/
[146652]<I> SigE:1
[146662]<I> SigE:12
[147513]<I> SigE:14
[147523]<I> SigE:4
[147533]<I> SigE:5
[147543]<I> SendTID(88)
[147555]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 8
[TX] D4 A8 01 88 21 01 8F 82     /* 设备上报Indication，TID 0x88，亮度0x828F（33423） */
[147624]<I> GenieE:55
[light_save_state]save 1 33423 20000    /* 保存开关1，亮度33423（51%），色温20000（100%）
*/
[147850]<I> RX Info
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] 88                          /* 设备收到Confirmation，TID 0x88，即上条上报对应的确认 */
```

- 设置灯色温。平台通过下发Light CTL Set（Opcode 0x825E）消息来设置灯的色温。Light CTL Set消息中的CTL Temperature字段（16bit）表示色温，取值范围为800～20000（即0x0320～0x4E20），其中下发0x320（800）代表色温值取最低（对应色温以百分比为单位的0%），0x4E20（20000）代表色温值取最高（对应色温以百分比为单位的100%）。中间的值按比例计算，如0x3020（12320）对应60%。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x825E
[RX] Payload size: 9
[RX] FF FF 20 30 00 00 C5 41 00
[160694]<I> SigE:20
[160704]<I> SIG mesg ElemID(0)
+MESHMSGRX:11,825EFFFF20300000C54100   /*Opcode 0x825F,Lightness 0xFFFF忽略,色温0x3020*/
[160744]<I> uv:0 lightness:65535
[160760]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 11
[TX] 82 60 8F 02 20 4E 8F 02 20 30 41 /*设备回复Light CTL Status消息，Opcode 0x8260,目标色温0x3020*/
[160836]<I> SigE:1
[160846]<I> SigE:12
[161696]<I> SigE:14
[161706]<I> SigE:4
[161716]<I> SigE:5
[161726]<I> SendTID(8A)
[161738]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 8
[TX] D4 A8 01 8A 22 01 20 30          /* 设备上报Indication，TID 0x8A，色温0x3020（12320） */
[161806]<I> GenieE:55                 /* 灯效执行完成 */
[light_save_state]save 1 655 12320    /* 保存开关1，亮度655（1%），色温12320（60%） */
[162070]<I> RX Info
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] 8A                               /* 设备收到Confirmation，TID 0x8A，即上条上报对应的确认 */
```

- 设置灯模式。平台通过下发Scene Set（Opcode 0x8242）消息来设置灯的模式。Scene Set消息中的字段Scene Number（16bit）表示灯的场景模式，是枚举型数据，如0x0003对应阅读模式，0x0004对应影院模式。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x8242
[RX] Payload size: 5
[RX] 03 00 C8 00 00
[182079][gatt_proxy_advertise]Connectable advertising deferred (max connections)  /*设备已有proxy连接*/
[182138]<I> SigE:20
[182148]<I> SIG mesg ElemID(0)
+MESHMSGRX:7,82420300C80000          /*Opcode 0x8242,目标场景号0x0003,TID 0xC8，渐变时间0，delay 0*/
[182181]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 3
[TX] 5E 03 00                        /*设备回复Scene Status消息，Opcode 0x5E，目标场景号0x0003*/
[182243]<I> SigE:1
[182252]<I> SigE:4
[182262]<I> SigE:5
[182272]<I> GenieE:55                /* 灯效执行完成 */
[182360]<I> RX Info
[RX] RSSI: 0
[RX] SRC: 0x128
[182395]<I> ReTID(8C), LR(1)
[182409]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0xF000
[TX] msg size: 8
[TX] D4 A8 01 8C 04 F0 03 00         /* 设备上报Indication，TID 0x8C，模式属性0x0003（阅读模式） */
[RX] DST: 0x1634
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] 8C                              /* 设备收到Confirmation，TID 0x8C，即上条上报对应的确认 */
```

调试云智能App解绑设备

- 在云智能App解绑设备，如下图所示。

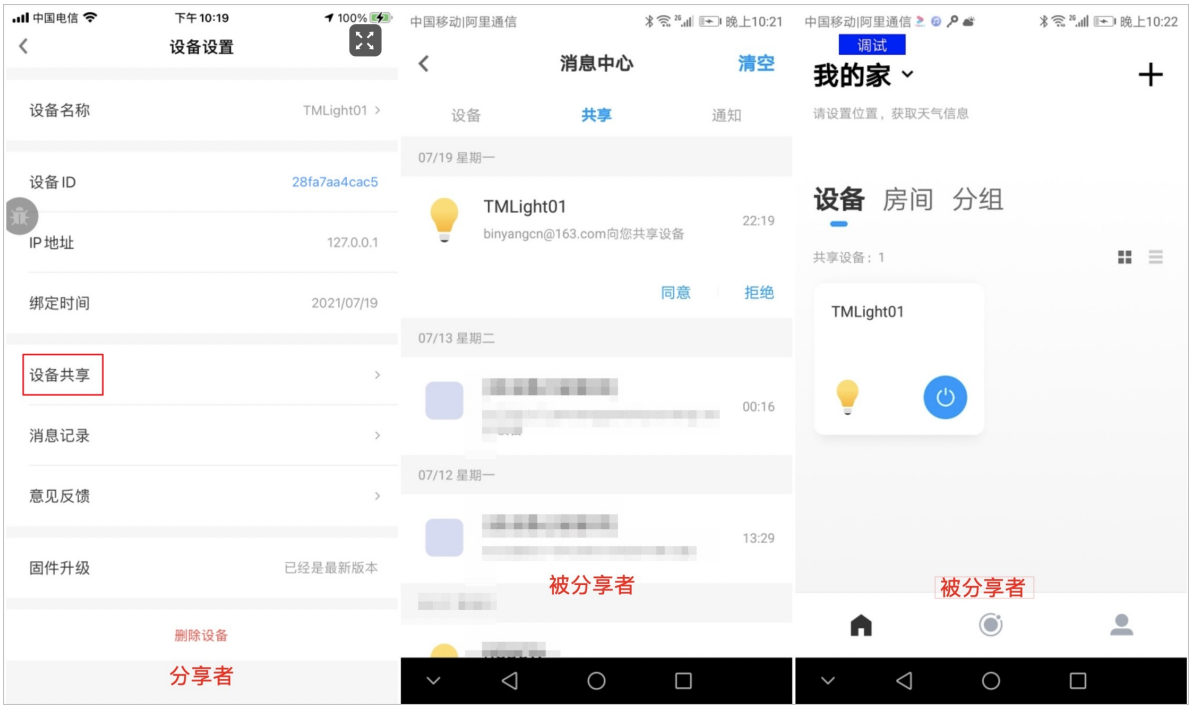
- 云智能App解绑时设备端关键日志（前提：云智能App与网络已建立proxy连接）。

```
[RX] RSSI: 0
[RX] SRC: 0x128
[RX] DST: 0x1634
[RX] OPCODE: 0x8049                /*设备收到Config Node Reset消息，Opcode 0x8049，设备解除
绑定*/
[RX] Payload size: 0
[311956]<I> TX Info
[TX] SRC: 0x1634
[TX] DST: 0x128
[TX] msg size: 2
[TX] 80 4A                         /*设备回复Config Node Reset Status消息，Opcode 0x804A，
确认解除绑定*/
[312022]<I> GenieE:1               /* 1：GENIE_EVT_SW_RESET */
[proxy_disconnected]conn(0x2000b63c) reason 0x16     /*断开proxy连接*/
[315125]<I> GenieE:10              /* 10：AIS连接断开 */
[315143]<I> GenieE:12              /* 12：GENIE_EVT_SDK_MESH_PBADV_START，开始待配网 */
```

> **说明**
>
> - 在没有网关只有App的网络中，当App解绑设备时，只有App已与网络中节点存在proxy连接时，即App与被删除设备之间可通信时，设备才有可能收到解绑指令。
> - 在没有网关只有App的网络中，当设备主动发起硬件复位（恢复出厂设置），上报硬件复位事件时，不能保证此时有App与网络中节点存在proxy连接，所以App不一定能收到设备的硬件复位事件。设备会清除配网信息，进入待配网状态。由于Mesh设备是抢占式的，App直接对其配网即可。

调试云智能App分享与控制设备

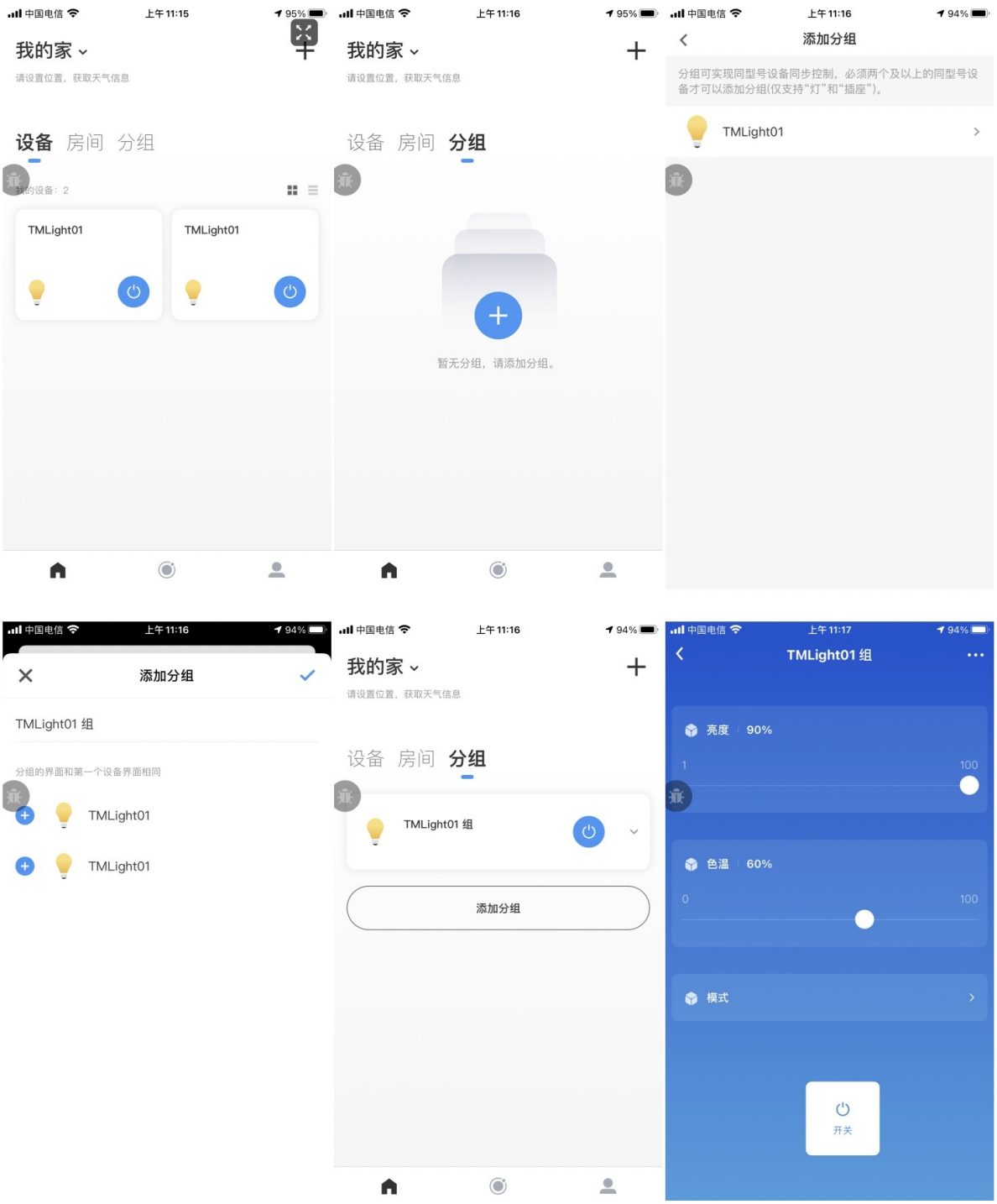
- 在分享者账号下分享设备，在被分享者账号下接受分享，如下图所示。



- 在被分享者账号下控制设备，注意被分享者的App也要先通过proxy连接接入分享者的网络才能实现控制。如果当前分享者网络中只有一个节点，由于一个节点只能同时允许一个proxy接入，则分享者与被分享者无法同时接入该节点的proxy服务。如果当前分享者网络中有N个设备，则理论上允许N个用户的App同时通过N个节点的proxy服务接入该网络。

调试云智能App分组控制

- 分组可实现同型号设备同步控制，必须两个以及以上的同型号设备（相同Product Key）才可以添加分组，且注意当前仅支持灯和插座产品。
- 添加分组与分组控制的示例如下。



- 添加分组时，云智能App会给设备配置添加组播地址，因此需要App与待添加分组设备之间可通信时（通过proxy直连或者别的proxy节点转发），该设备才能被添加成功。
- 添加分组时的设备端关键日志如下。

```
[RX] RSSI: 0
[RX] SRC: 0x133
[RX] DST: 0x2870
[RX] OPCODE: 0x801B              /*设备收到Config Model Sub Add消息，Opcode:0x801B*/
[RX] Payload size: 6
[RX] 70 28 12 E7 00 10           /*Element Address 0x2870，添加组播地址0xE712*/
[116470]<I> TX Info
[TX] SRC: 0x2870
[TX] DST: 0x133
[TX] msg size: 9
[TX] 80 1F 00 70 28 12 E7 00 10  /*设备回复Config Model Sub Status，Opcode:0x801F，Status 0
：添加成功*/
```

- 分组控制时设备端关键日志如下。

```
[RX] RSSI: 0
[RX] SRC: 0x133
[RX] DST: 0xE712                 /* 目标地址为0xE712的组播地址 */
[RX] OPCODE: 0x825E              /* 设备收到Light CTL Set消息，Opcode:0x825E*/
[RX] Payload size: 9
[RX] 66 E6 20 30 00 00 2E 41 00  /* 目标色温值 0x3020（12320），即60%*/
[152135]<I> SigE:20
[152149]<I> SIG mesg ElemID(0)
+MESHMSGRX:11,825E66E6203000002E4100    /* 色温值调整到0x3020，即60% */
[152251]<I> TX Info
[TX] SRC: 0x2870
[TX] DST: 0x133
[TX] msg size: 11
[TX] 82 60 66 E6 20 4E 66 E6 20 30 41   /* 设备通过单播消息回复Light CTL Status */
[152327]<I> SigE:1
[152337]<I> SigE:12
[gatt_proxy_advertise]Connectable advertising deferred (max connections) /* 该节点有proxy
连接，为proxy节点*/
[153229]<I> SigE:14
[153239]<I> SigE:4
[153249]<I> SigE:5
[153260]<I> SendTID(B1)
[153271]<I> TX Info
[TX] SRC: 0x2870
[TX] DST: 0xF000
[TX] msg size: 8
[TX] D4 A8 01 B1 22 01 20 30            /* 设备上报Indication，TID 0xB1，色温属性0x3020（60%
）  */
[153340]<I> GenieE:55
[light_save_state]save 1 58982 12320    /* 保存开关1，亮度90%，色温60% */
[RX] RSSI: 0
[RX] SRC: 0x133
[RX] DST: 0x2870
[RX] OPCODE: 0xD501A8
[RX] Payload size: 1
[RX] B1                                 /* 设备收到Confirmation，TID 0xB1，即上条上报对应的确
认 */
```

- 删除分组时的设备端关键日志如下。

```
[RX] RSSI: 0
[RX] SRC: 0x133
[RX] DST: 0x2870
[RX] OPCODE: 0x801C              /*设备收到Config Model Sub Delete消息，Opcode:0x801C*/
[RX] Payload size: 6
[RX] 70 28 12 E7 00 10           /*Element Address 0x2870，删除组播地址0xE712*/
[116470]<I> TX Info
[TX] SRC: 0x2870
[TX] DST: 0x133
[TX] msg size: 9
[TX] 80 1F 00 70 28 12 E7 00 10  /*设备回复Config Model Sub Status，Opcode:0x801F，Status 0
：删除成功*/
```

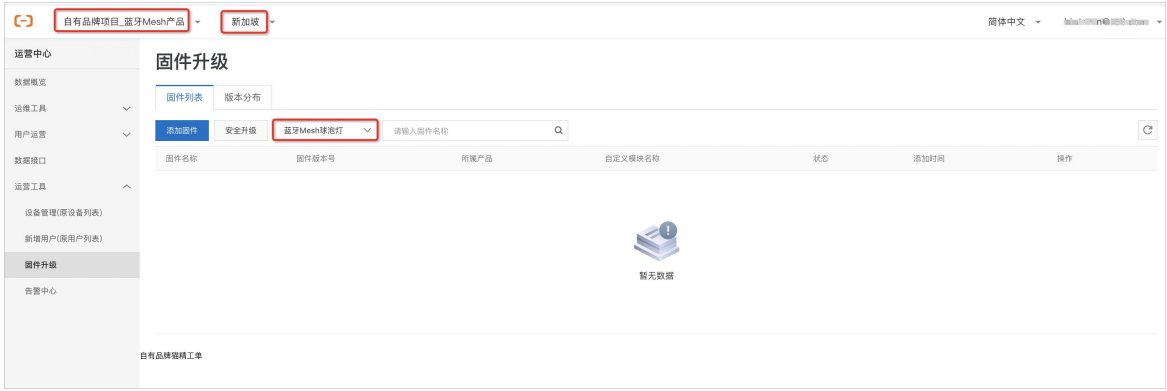
本地定时

自有品牌蓝牙Mesh产品暂不支持本地定时。

固件OTA

云智能App V3.10.0以上版本支持对自有品牌项目蓝牙Mesh设备做固件OTA，因此在开始OTA之前，注意检查和升级云智能App版本。

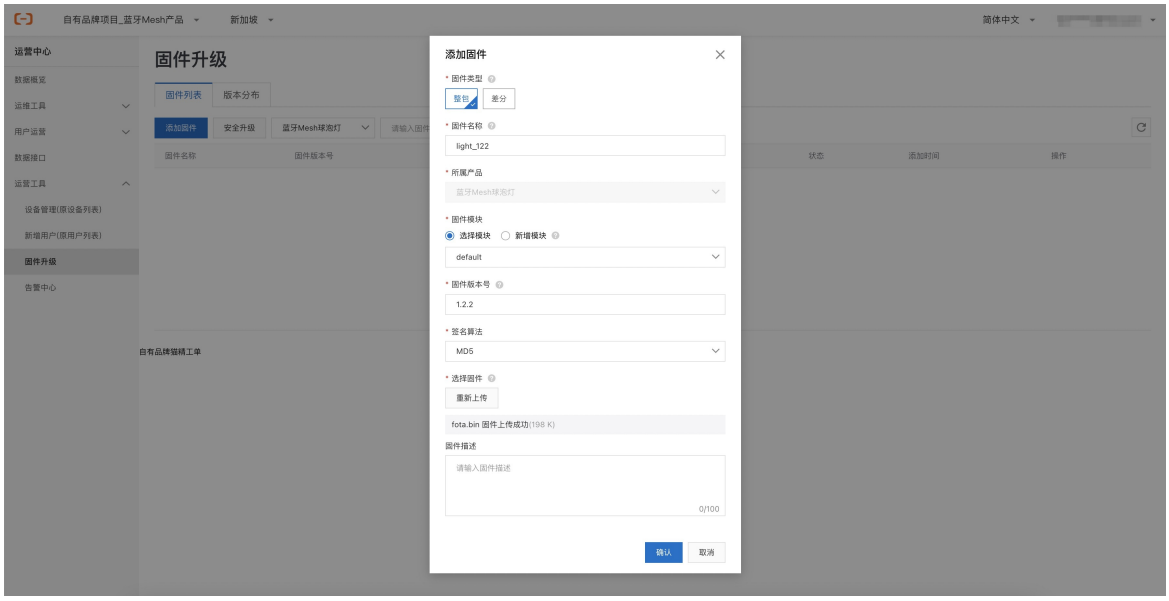
1. 进入生活物联网平台的运营中心。
2. 进入**运营工具 > 固件升级**页面，选择相应项目下的相应产品。



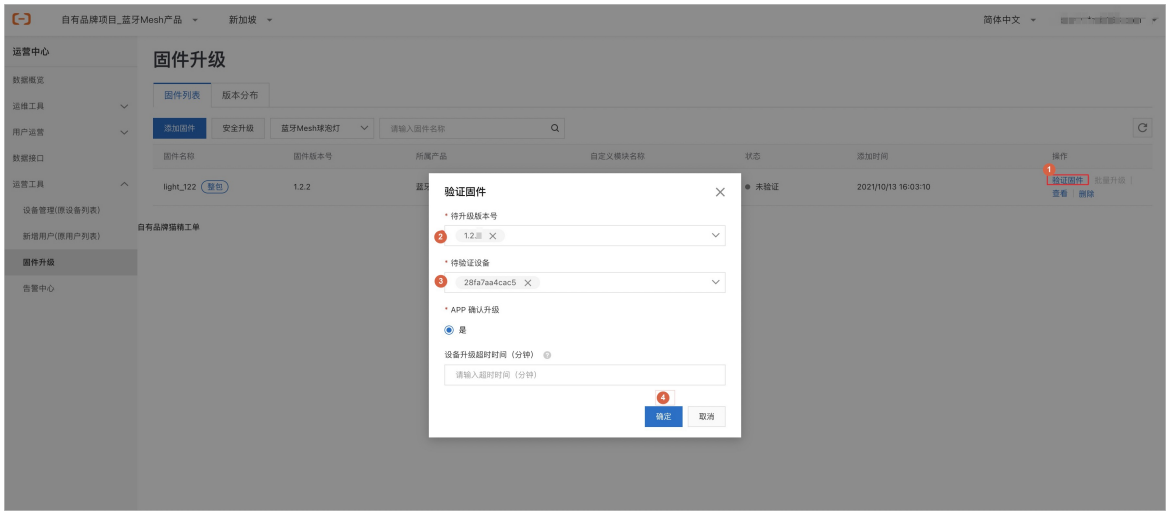
3. 获取待升级固件文件和版本信息。固件编译成功后*fota.bin*文件即为OTA的bin文件。

   版本号即为编译OTA的bin文件时指定的宏PROJECT_SW_VERSION的值。版本号格式遵循版本格式。

4. 进入**固件升级**，在**固件列表**窗口下操作下列步骤。
   i. 单击**添加固件**。
   ii. 在固件版本号栏中并输入**OTA固件版本号**。
   iii. 在签名算法栏中，选择MD5。
   iv. 单击选择固件下的**上传固件**，选择上传待升级的bin文件。
   v. 单击**确定**完成添加固件。



5. 对新增加的固件执行操作**验证固件**。在待验证设备中选择测试设备的DeviceName。



6. 进入云智能App，进入待验证设备的**设备设置**页面，单击**立即升级**开始OTA升级。

7. 升级完成后，查看**运营中心**升级状态。


自有品牌项目_蓝牙Mesh产品
新加坡
简体中文
运营中心
数据概览
运维工具
用户运营
数据接口
运营工具
设备管理(原设备列表)
新增用户(原用户列表)
固件升级
告警中心
light_122
已验证
固件类型
整包
签名算法
Md5
固件签名
b58ee53274dc43499256ed425a7e3f03
下载
自定义模块名称
default
目标设备总数
1
目标成功数
1
目标失败数
0
目标取消数
0
批次管理
设备列表
固件信息
验证固件
批量升级
请输入批次 ID
批次 ID
批次类型
升级策略
状态
创建时间
操作
253870
验证固件
静态升级
已完成
2021/10/13 16:21:21
查看

8. 查看设备端关键日志。

```
GenieE:11        /* 建立AIS连接 */
ais cmd:0x10 state:1 len:20
ais rsp err(0) p_ind(0x1fffd718)
ais cmd:0x12 state:2 len:5
ais rsp err(0) p_ind(0x1fffd718)
ais cmd:0x20 state:3 len:20
ais cmd:0x20 state:3 len:20
ais cmd:0x22 state:3 len:20
[3696.886]<I>AOSBT genie_ota_handle_update_request:ota ver:0x00010202 imgT:0 imgS:20256
8 maxS:217088 /*新固件版本号与大小*/
[3697.192]<I>AOSBT genie_ota_status_report:ota last[15] total[15] size[1984] err[0]
…
ota progress:10
…
ota progress:20
…
ota progress:30
…
ota progress:40
…
ota progress:50
…
ota progress:60
…
ota progress:70
…
ota progress:80
…
ota progress:90
…
ota progress:100
[3732.236]<I>AOSBT genie_ota_status_report:ota last[1] total[1] size[202568] err[0]
ais cmd:0x25 state:4 len:20
ais notify msgid:0 cmd:26 len:16
ais timeout disconnect    /*断开AIS连接*/
proxy disconn:0x1fff94e8 reason:0x16
[   0.004]<I>INIT Build:Oct 12 2021,16:34:29    /*重启*/
[   0.009]<I>INIT find 7 partitions
[   0.217]<I>AOSBT application_start:BTIME:Oct 12 2021,16:34:29
DEVICE:GenieLight
APP VER:1.2.2
GenieSDK:V1.3.6
PROUDUCT:TG7120B
MAC:28:FA:7A:A4:CA:C5
ais pre init
….
GenieE:3
```

量产设备注意事项

- 购买激活码时，注意选择自有品牌蓝牙设备。

- 对于自有品牌项目的产品，智能生活物联网平台可以提供MAC地址段。
  - 如果使用智能生活物联网提供的MAC地址，激活码生成方式可以选择**自动生成**。
  - 如果需要自己上传MAC地址，作为Device Name来生成设备证书，激活码生成方式可以选择**批量上传**。

7.3. Wi-Fi智能插座开发实践

7.3.1. Wi-Fi智能插座设备端开发

本文以TG7100C芯片为例，介绍基于生活物联网平台SDK（V1.6.6）中的smart_outlet应用示例，开发单孔Wi-Fi智能插座设备固件的流程。

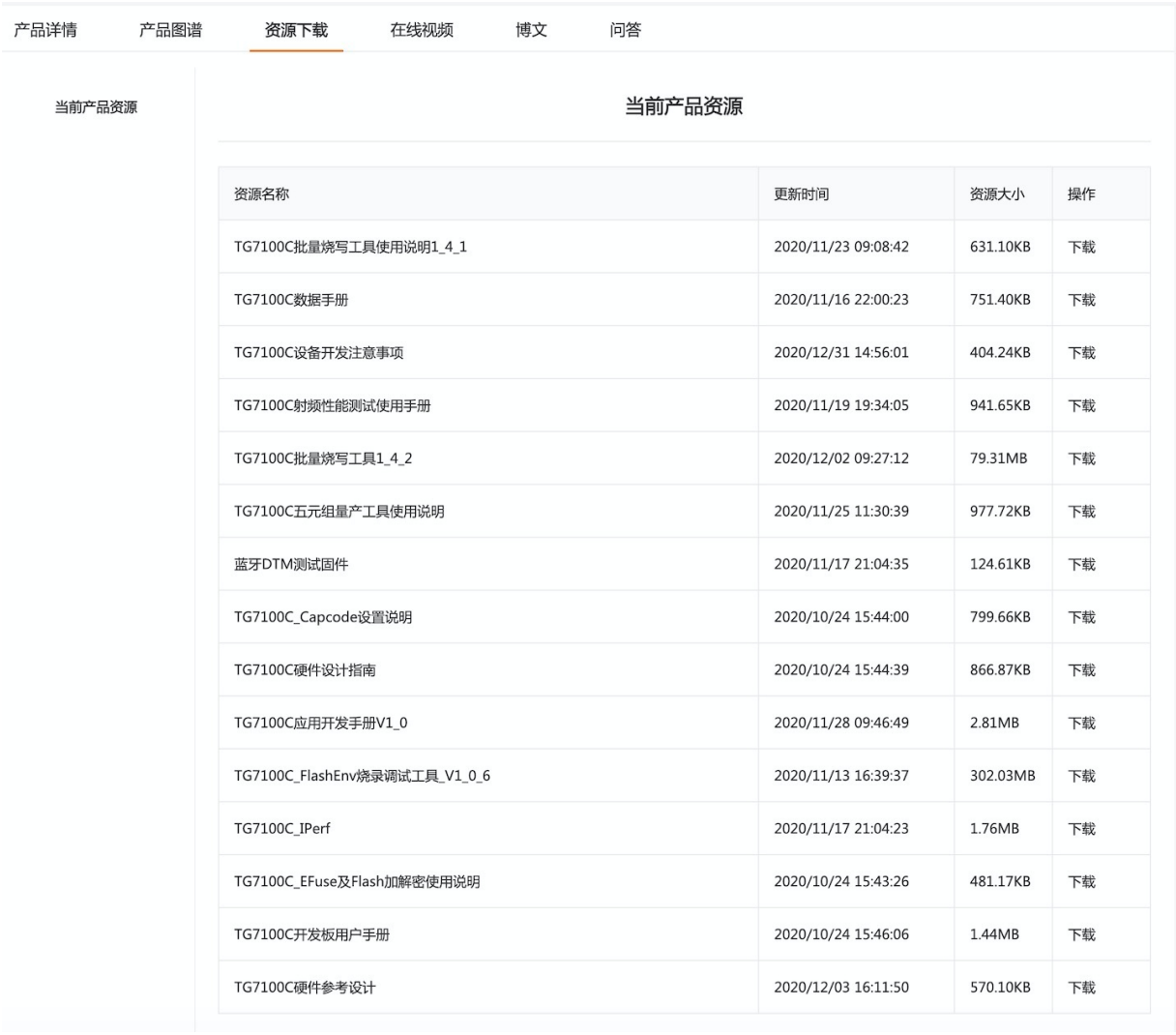
背景信息

应用示例smart_outlet的功能介绍如下：

- 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。

TG7100C概述

TG7100C是天猫精灵推出的Wi-Fi蓝牙Combo芯片。TG7100C芯片相关文档和软件工具介绍，请参见平头哥开放社区资源介绍。



资源名称	更新时间	资源大小	操作
TG7100C批量烧写工具使用说明1_4_1	2020/11/23 09:08:42	631.10KB	下载
TG7100C数据手册	2020/11/16 22:00:23	751.40KB	下载
TG7100C设备开发注意事项	2020/12/31 14:56:01	404.24KB	下载
TG7100C射频性能测试使用手册	2020/11/19 19:34:05	941.65KB	下载
TG7100C批量烧写工具1_4_2	2020/12/02 09:27:12	79.31MB	下载
TG7100C五元组量产工具使用说明	2020/11/25 11:30:39	977.72KB	下载
蓝牙DTM测试固件	2020/11/17 21:04:35	124.61KB	下载
TG7100C_Capcode设置说明	2020/10/24 15:44:00	799.66KB	下载
TG7100C硬件设计指南	2020/10/24 15:44:39	866.87KB	下载
TG7100C应用开发手册V1_0	2020/11/28 09:46:49	2.81MB	下载
TG7100C_FlashEnv烧录调试工具_V1_0_6	2020/11/13 16:39:37	302.03MB	下载
TG7100C_IPerf	2020/11/17 21:04:23	1.76MB	下载
TG7100C_EFuse及Flash加解密使用说明	2020/10/24 15:43:26	481.17KB	下载
TG7100C开发板用户手册	2020/10/24 15:46:06	1.44MB	下载
TG7100C硬件参考设计	2020/12/03 16:11:50	570.10KB	下载

说明

目前阿里云和平头哥两个工单系统是各自独立的，请您注意区分。

- 关于TG7100C芯片的驱动、产测、硬件设计、射频等使用问题，请通过平头哥芯片开放社区工单系统反馈。更多介绍，请参见平头哥智能机器人—工单系统使用文档。
- 基于生活物联网平台SDK的应用开发，例如产品配置、配网、连云、OTA等问题，您可以通过阿里云工单系统反馈。

固件编译

1. 下载SDK。下载地址，请参见获取SDK。
2. 配置开发环境。详细介绍，请参见准备开发环境。
3. 编译代码。

i. （可选）解压SDK压缩包。

如果您通过Git命令的方式下载SDK，则无需解压。

SDK根目录*build.sh*文件说明：根据硬件使用的模组型号和要编译的应用，可以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             //配置debug等级，生产固件建议为0
default_args=""             //配置其他编译参数
//更多介绍，请参见README.md
```

ii. 将开发的业务代码存放到SDK相应的目录下，例如非计量插座标品代码在*Products/example/smart_outlet* 目录下。

iii. 执行以下命令，快速编译smart_outlet应用示例。

```
./build.sh example smart_outlet tg7100cevb MAINLAND ONLINE 0
```

编译完成后，在*out/smart_outlet@tg7100cevb/*目录下会生成*smart_outlet@tg7100cevb.bin*文件。

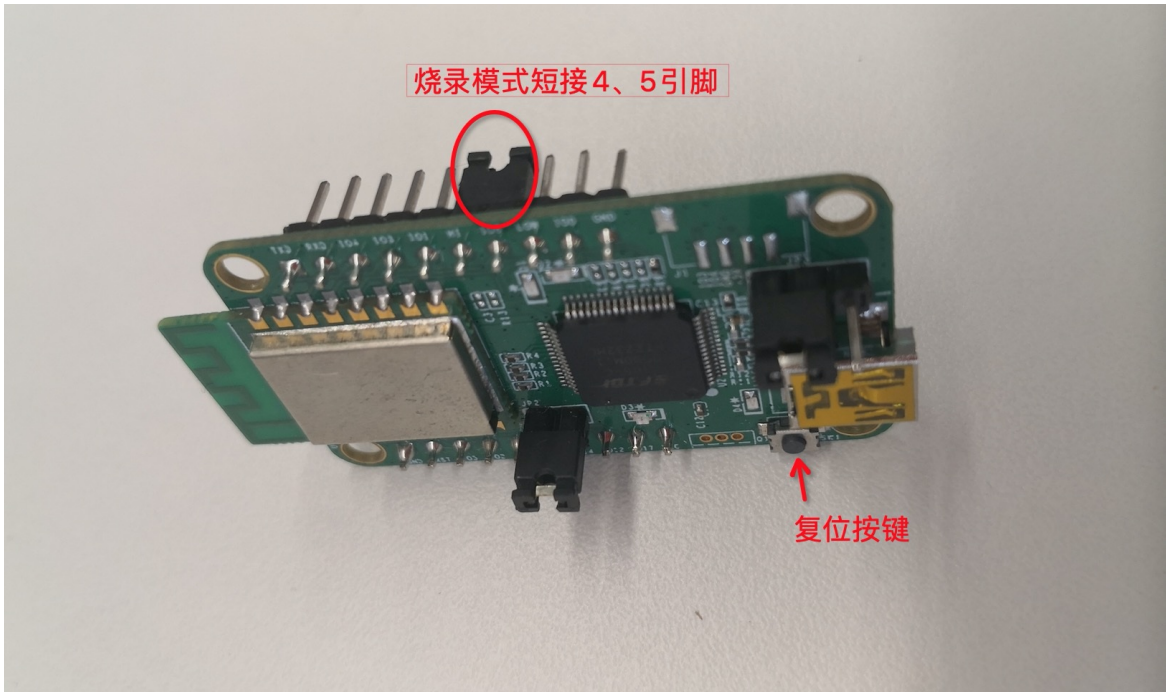
iv. 将*smart_outlet@tg7100cevb.bin*文件烧录到真实设备中。

*tg7100cevb_ota.bin*文件为OTA使用的固件。

固件烧录与运行

1. 在TG7100C开发板上烧录固件。详细操作，请参见TG7100C开发板用户手册。
2. 短路接通开发板的第4个引脚与第5个引脚，并按开发板的复位键。

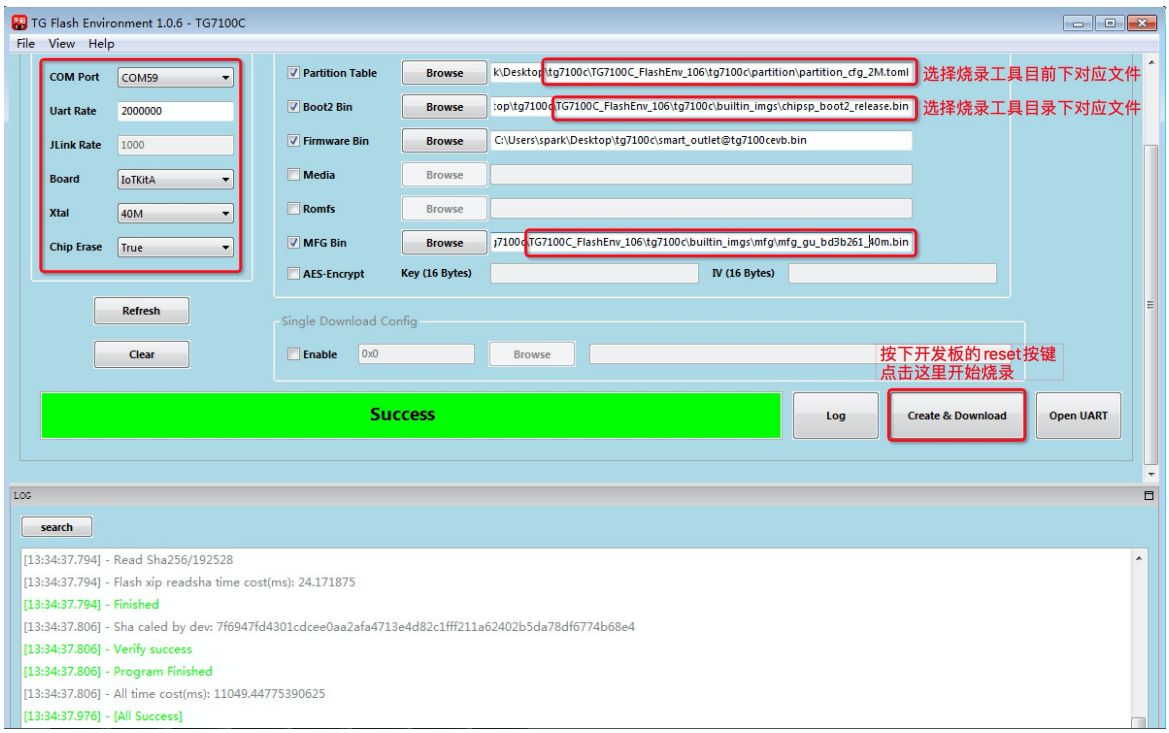
   开发板设置图如下。



3. 打开下载好的烧录工具目录中的TGFlashEnv.exe，单击**Finish**，进入烧录界面。
4. 将Interface选择Uart，并单击**Refresh**按键。
5. 设置串口参数，以及选择好对应的烧录文件。

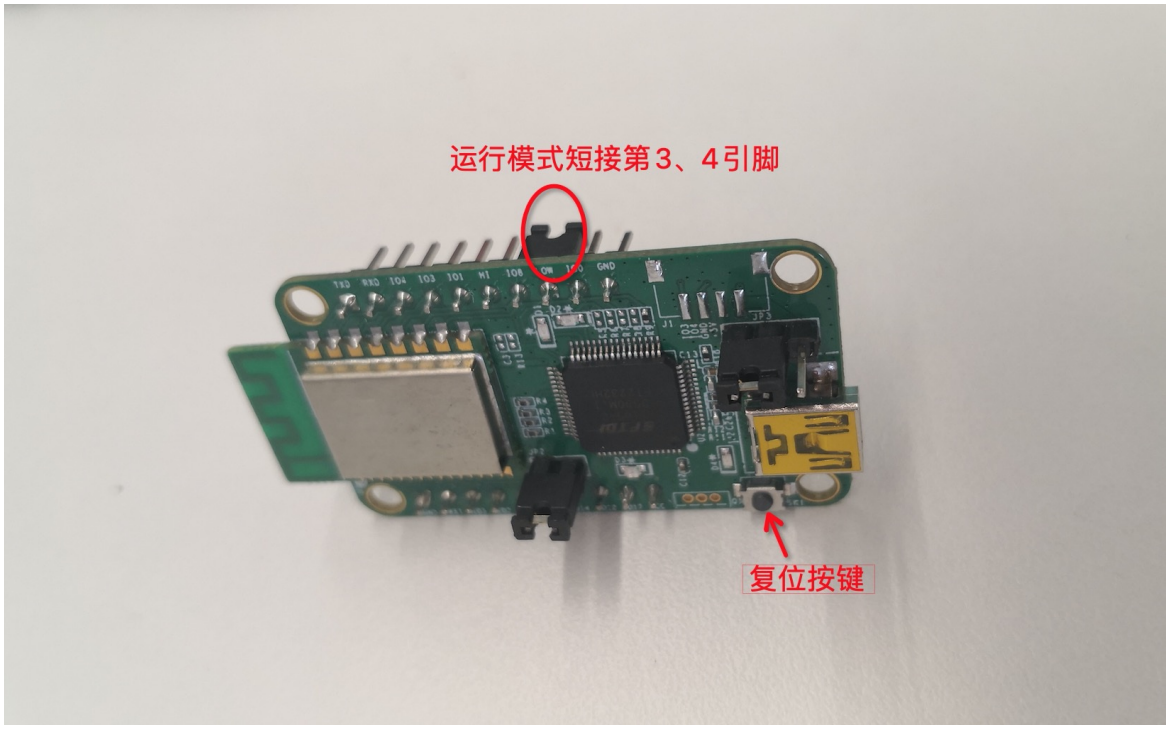
   串口参数的配置如下图所示。

   - Partition Table、Boot 2 Bin与MFG Bin选择烧录工具目录下对应的文件即可，文件名如图。
   - Firmware Bin选择编译出的固件。
   - Chip Erase可以根据是否要擦除整片Flash选择True或者False，如调试中要保留之前写入过的设备证书，可以选择False。



6. 单击**Download**按钮，并同时按下开发板上的复位键，开始烧录固件。
7. 烧录完毕后，查看运行的日志。

   短路连接第3个引脚与第4个引脚（如下图所示），将串口工具波特率设置为2000000，并按下开发板复位键。



   常用的cli指令如下。

   - **reset**：设备重置，清除设备配网信息。
   - **free**：查看内存使用情况。
   - **linkkey**：写入与查看证书。
   - **mac**：查看开发板Wi-Fi MAC地址。

smart_outlet应用代码结构介绍

smart_outlet应用示例中的文件结构如下。

```
├── Products
│   │   ├── example/smart_outlet
│   │   │   ├── app_entry.c
│   │   │   ├── app_entry.h
│   │   │   ├── combo_net.c
│   │   │   ├── device_state_manager.c
│   │   │   ├── device_state_manager.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
```

详细的文件说明如下。

- 厂家需要适配的文件（设备初始化等）：*vendor.c*与*vendor.h*
- 应用程序主入口：*app_entry.c*与*smart_outlet_main.c*
- 配网和连云状态管理：*device_state_manager.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*

固件适配说明

标品固件移植适配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。

- GPIO适配

  单路插座需要两个GPIO分别控制继电器开关、LED亮灭和一个GPIO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。

- 状态LED显示适配

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

```
typedef enum {
    RECONFIGED = 0,             //reconfig with netconfig exist
    UNCONFIGED,                 //配网开始
    AWSS_NOT_START,             //配网超时
    GOT_AP_SSID,                //连接AP成功
    CONNECT_CLOUD_SUCCESS,      //连云成功
    CONNECT_CLOUD_FAILED,       //连云失败
    CONNECT_AP_FAILED,          //连接AP失败
    CONNECT_AP_FAILED_TIMEOUT, //连接AP超时
    APP_BIND_SUCCESS,           //APP绑定成功
    ...
    UNKNOW_STATE
} eNetState;
```

- 状态显示的处理代码在文件*Products/example/smart_outlet/device_state_manager.c*中的`indicate_net_state_task`函数中。可以根据产品的不同需求做调整。

```
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);
    }
    ...
}
```

- 当前代码中实现的默认LED显示如下。

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。

- 按键处理适配

  标品固件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_manager.c*文件中的 `key_detect_event_task` 函数负责按键处理。如下定义了各种行为的时间，如果需要调整各个行为的按键时长，可以自行修改。

```
#define AWSS_REBOOT_TIMEOUT (4 * 1000) //长按4s 进入网络配置模式，开始重新配网
#define AWSS_RESET_TIMEOUT (6 * 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 KEY_PRESSED_VALID_TIME_MIN 100
#define KEY_PRESSED_VALID_TIME_MAX 500 //按键按下超过100ms，小于500ms，表示有按键按下
#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_TI
MEOUT;
    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4s，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6s，设备会进入恢复出厂模式。

设备端通用功能说明

以下功能在smart_outlet应用示例中已有相关实现，仅对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_RAWDATA_ARRIVED, user_down_raw_data_arrived_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);
    ...
}
```

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)
{
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *property_payload = NULL;
    cJSON *response_root = NULL, *item_csr = NULL;
    response_root = cJSON_CreateObject();
    if (response_root == NULL) {
        return;
    }
    if (msg->seq != NULL && strcmp(msg->seq, SPEC_SEQ)) {
        item_csr = cJSON_CreateObject();
        if (item_csr == NULL) {
            cJSON_Delete(response_root);
            return;
        }
        cJSON_AddStringToObject(item_csr, "seq", msg->seq);
        cJSON_AddItemToObject(response_root, "CommonServiceResponse", item_csr);
    }
#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_format
ed,&len);
#ifdef EN_COMBO_NET //对于Wi-Fi&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_PRO
PERTY,
                    (unsigned char *)property_formated, strlen(property_formated), msg->f
lag);
```

```
        } else {
            res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERT
Y,
                    (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_PRO
PERTY,
                    (unsigned char *)property_payload, strlen(property_payload), msg->fla
g);
        } else {
            res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERT
Y,
                    (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 in
t 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(i
tem)) {
        ...
    }
    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_IsNumbe
r(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(i
tem))|| \
        ((item = cJSON_GetObjectItem(root, "CountDownList")) != NULL && cJSON_IsObject(it
em)) || \
        ((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）

  本地通信功能介绍，请参见本地通信开发实践。

  本地通信功能在文件*make.settings*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 in
t request_len, char **response,
        int *response_len)
{
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    device_status_t *device_status = &user_example_ctx->status;
    cJSON *request_root = NULL, *item_propertyid = NULL;
    cJSON *response_root = NULL;
    ...
    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->valuestrin
g);
        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->powerswit
ch);
        }
        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_pow
erstate);
        }
#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->powerswi
tch);
        }
#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。您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。更多介绍，请可以参见示例代码*example/smart_outlet/smart_outlet_main.c*中的 `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，请参见设备端开发。

  > **说明** SDK V1.6.6开始支持新的蓝牙辅助配网方案（配合天猫精灵App V4.13.0以上版本与云智能App V3.5.5以上版本使用），新方案要求设备证书的Device Name与Wi-Fi MAC保持一致。更多介绍，请参见开发自有品牌项目插座产品、开发天猫精灵生态项目插座产品。

- 本地定时功能

  本地定时功能设备端开发，请参见开发设备端本地定时功能。

7.3.2. 开发自有品牌项目插座产品

本文以TG7100C芯片为例，介绍在开发自有品牌项目智能插座产品时，使用开发板调试配网连云的过程，以及开发出海产品、量产设备证书的注意事项。

创建与配置产品

1. 创建一个自有品牌项目。详细操作，请参见创建项目。
2. 创建一个品类为插座的产品。详细操作，请参见创建产品。
3. 定义产品功能。

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*，建议您在创建产品后通过导入物模型复制到新建产品中。

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。

- 拷贝新建产品的ProductKey

  编辑文件*Products/example/smart_outlet/smart_outlet.json*，将productKey字段替换成新的，并保存。

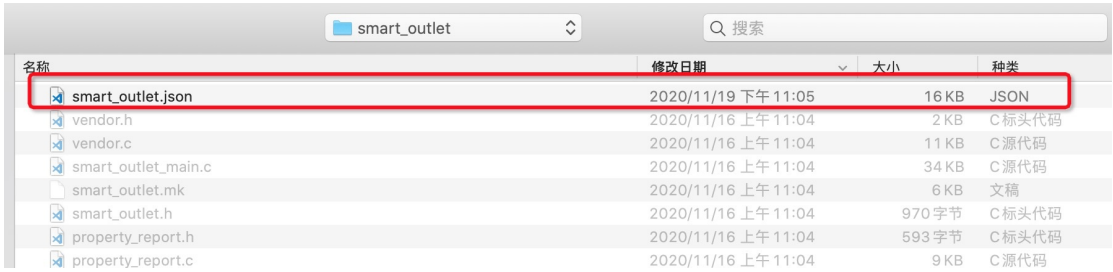

```
{
    "schema":"https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
    "profile":{
        "productKey":"a1LrvE7J2vL"
    },
    "properties":[
        {
            "identifier":"CountDownList",
            "name":"倒计时列表",
            "accessMode":"rw",
            "required":true,
            "dataType":{
                "type":"struct",
                "specs":[
                    {
                        "identifier":"Target",
                        "name":"操作对象",
                        "dataType":{
                            "type":"text",
                            "specs":{
                                "length":"2048"
                            }
                        }
                    },
                    {
                        "identifier":"Contents",
                        "name":"倒计时命令",
                        "dataType":{
                            "type":"text",
                            "specs":{
                                "length":"2048"
                            }
                        }
                    },
                    {
                        "identifier":"allPowerstate",
                        "name":"全控开关",
                        "dataType":{
                            "type":"bool",
                            "specs":{
                                "0":"关闭",
```

- 导入一份json文件

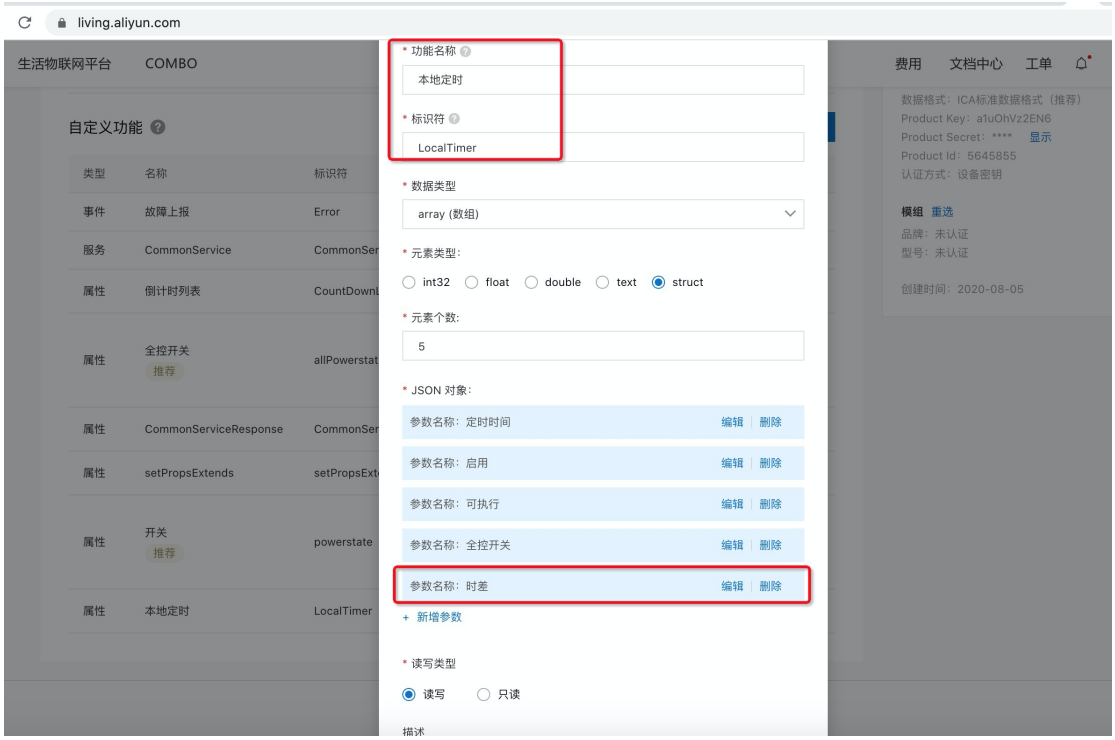
  导入物模型时，建议您选择*Products/example/smart_outlet/smart_outlet.json*路径下的json文件。导入物模型的具体操作，请参见导入物模型。

  导入物模型时，您还需要注意以下内容。

- 建议您选择*Products/example/smart_outlet*路径下的smart_outlet.json文件。



- 当您的产品售往中国内地以外地区时，请确定本地定时属性的JSON对象定义内包括时差（TimezoneOffset），该参数会影响不同时区下的本地定时功能能否正常工作。



4. 配置人机交互。

   如需使用云智能公版App，请打开使用云智能公版App的开关。如需在TG7100C芯片上使用蓝牙辅助配网，注意标准配网类型选择BLE+Wi-Fi（蓝牙辅助配网）。

注意完成所有必填配置项，详细说明，请参见配置人机交互。

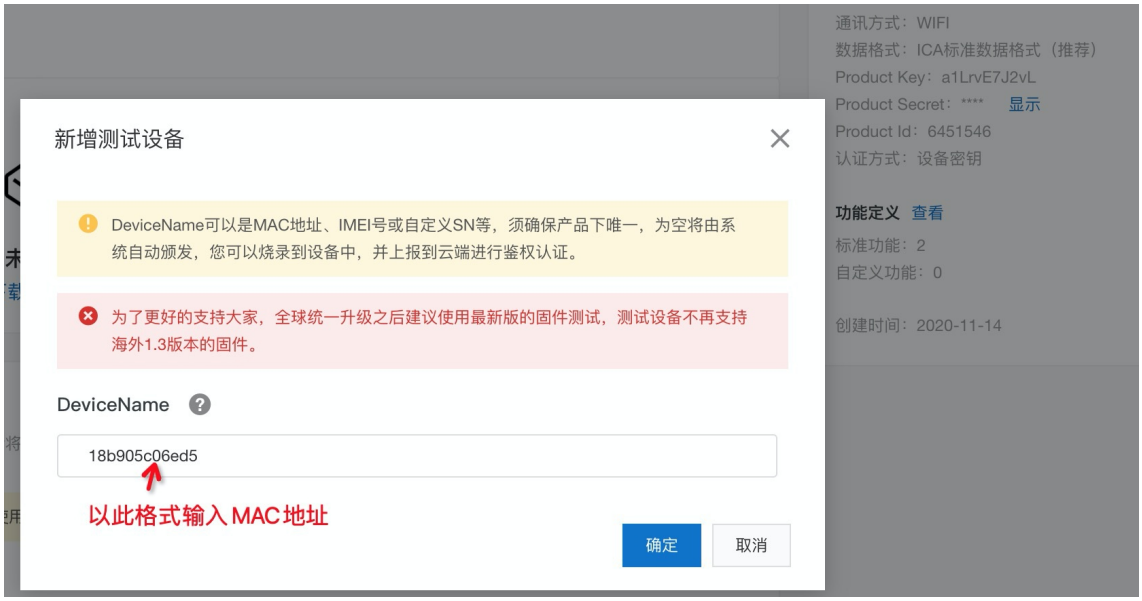
5. 生成测试的设备证书。

   新蓝牙辅助配网要求Wi-Fi的MAC地址与设备证书中的Device Name保持一致。在TG7100C开发板上，可以在固件运行后通过串口输入mac指令，查询芯片当前的Wi-Fi MAC地址，并用此MAC地址作为Device Name在设备调试页面申请测试设备证书。注意**MAC地址中的字母全部为小写格式**。

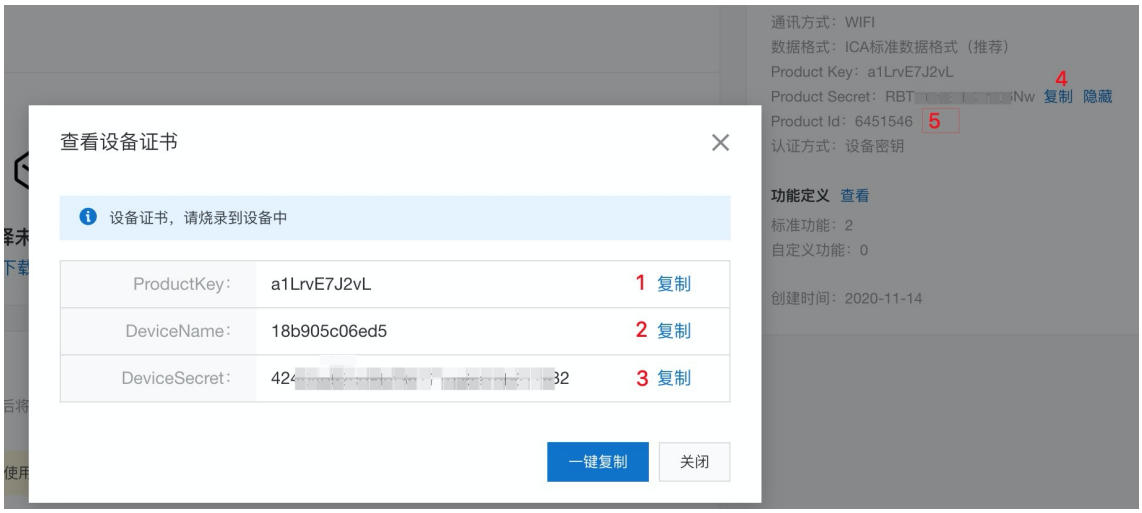
   - 查询Wi-Fi MAC地址示例

     ```
     #mac
     MAC address: 18-b9-05-c0-6e-d5
     ```

   - 输入**小写格式的MAC地址**，如示例中的18b905c06ed5作为DeviceName生成测试设备



- 拷贝测试设备证书，请注意一定要依次完整拷贝ProductKey，DeviceName，DeviceSecret，ProductSecret，ProductID。



6. 批量投产，发布设备

   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。

设备配网连云

- 设置设备证书
  - 将设备固件烧录到开发板之后，可以通过linkkey命令设置设备证书，然后通过reset命令重置设备。
  - 设置设备证书信息，在设备上电后将完整的设备证书信息写入开发板。

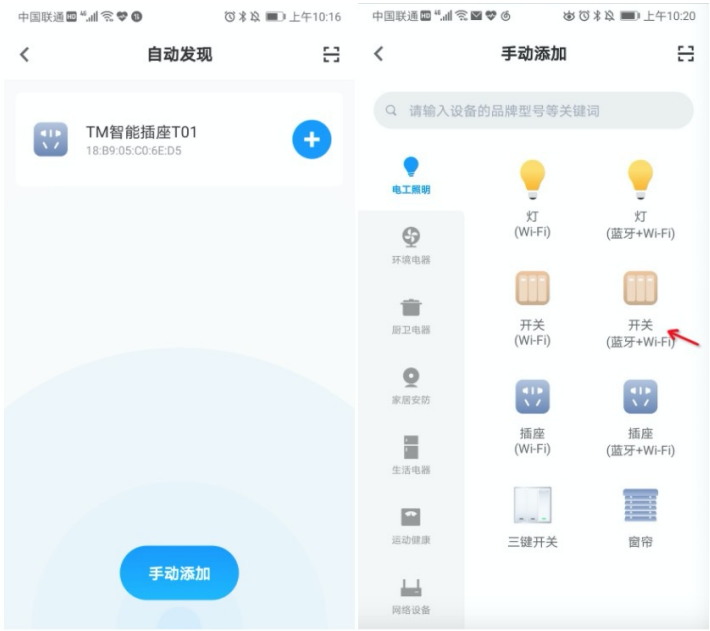
    ```
    linkkey ProductKey DeviceName DeviceSecret ProductSecret ProductID
    ```

  - 设备重置，清除设备配网信息。

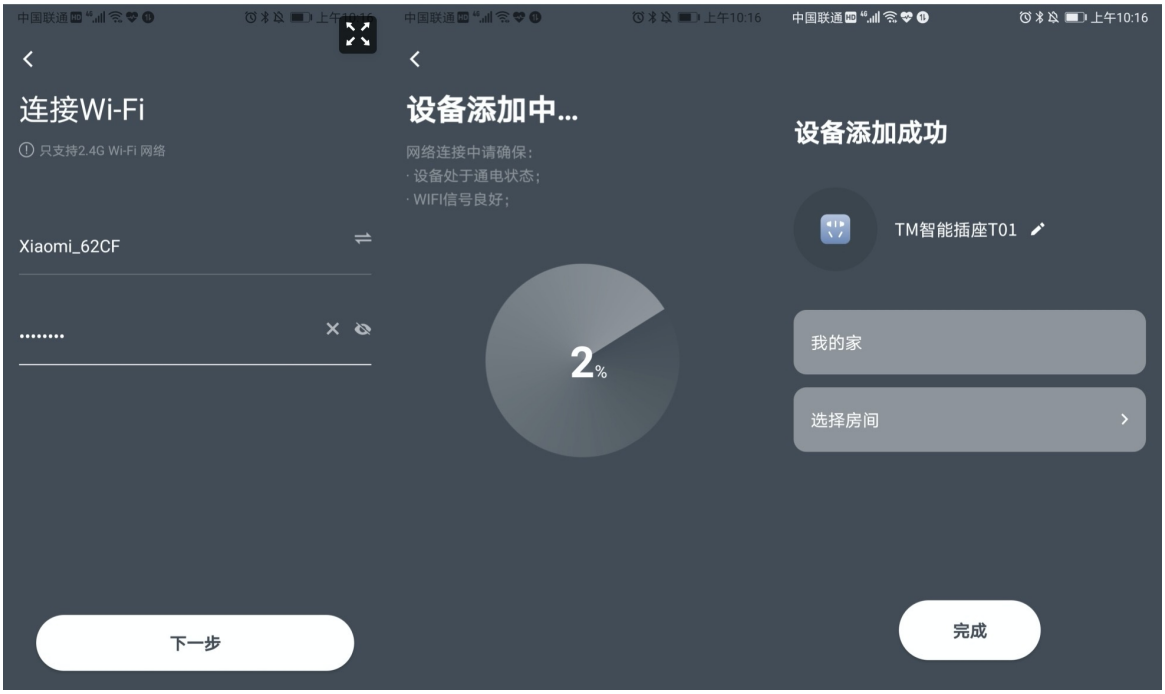
    ```
    reset
    ```

- 云智能App配网（V3.5.5以上版本）
  - 在开发板上设置自有品牌产品的设备证书，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。

- 在**批量投产**页面确认“开发完成”的产品，可以在云智能App**自动发现**页面，发现处于待配网状态的设备。如果产品未确认“开发完成”，可以单击**手动添加**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。



- App配网过程如下图所示。



- 配网连云设备端关键日志示例

- 蓝牙辅助配网

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
开启蓝牙辅助配网	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

- 零配

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
Dev Scan	awss	IOTX_AWSS_START
Awss Process	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

- 一键配网

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
Dev Scan	awss	IOTX_AWSS_START
Awss Process	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

- 设备热点配网

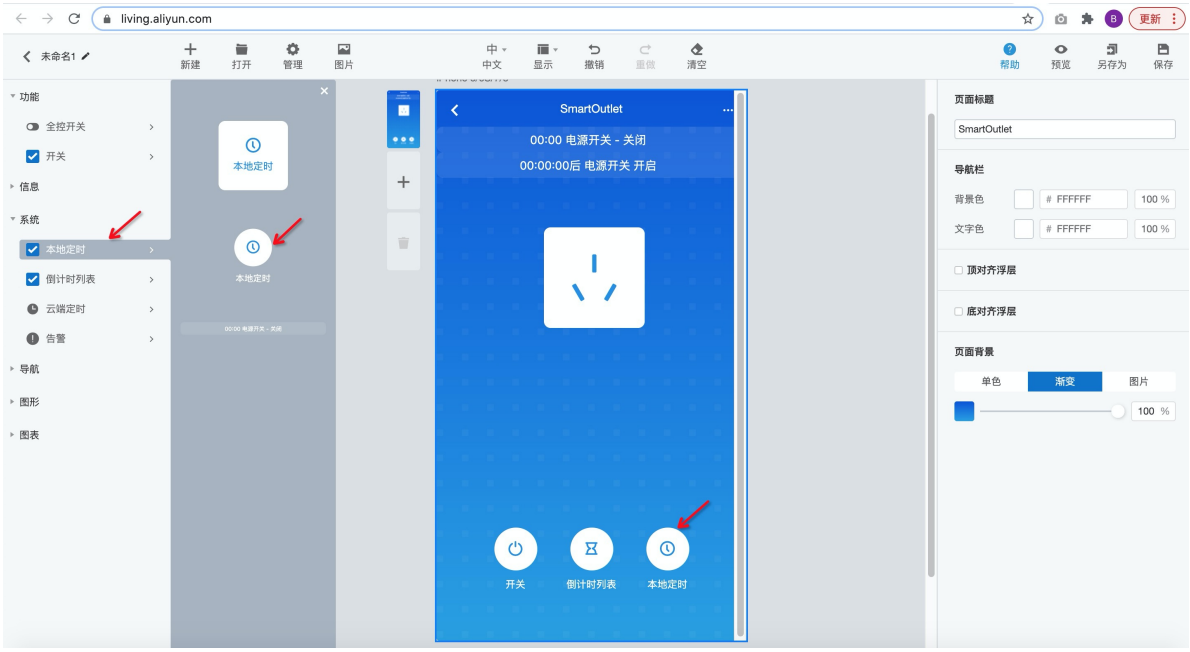
配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
起设备热点	dev_ap	ssid:adh_[pk]_[mac suffix]
Parse SSID/PWD	设备解析出热点信息	IOTX_AWSS_GOT_SSID_PASSWD
AP Connect	设备连接AP	■ switch to ap:[ap ssid] ■ IOTX_AWSS_CONNECT_ROUTER
DHCP Get IP	设备获取IP地址	Let's post GOT_IP event
Cloud Connect	连云成功	Cloud Connected

本地定时功能

本地倒计时功能的详细介绍，请参见开发设备端本地定时功能。

开发过程中，您需要注意以下内容。

- 注意在产品**功能定义**页面里添加LocalTimer本地定时属性。
- 在**人机交互**页面的自动化与定时中，选中本地定时功能复选框。
- 在**人机交互**页面的设备面板中，注意区别本地定时与云端定时。如果产品需要使用本地定时，则明确指定本地定时的组件。



本地倒计时功能

本地倒计时功能的详细介绍，请参见本地倒计时功能开发实践。

开发过程中，您需要注意以下内容。

- 在产品的**功能定义**页面里，添加CountDownList倒计时列表属性。
- 在**人机交互**页面的自动化与定时中，选中本地倒计时功能复选框。
- 在**人机交互**页面的设备面板中，选择的面板中使用了CountDownList倒计时列表的组件。

三方语音平台

三方语音平台Amazon Alexa和Google Home的对接，请参见以下文档。

- 公版App使用Amazon Echo音箱控制设备
- 公版App使用Google Home音箱控制设备
- 自有App定制Amazon Alexa技能
- 自有App定制Google Assistant技能

固件OTA

自有品牌项目的产品除了云智能App与天猫精灵App操作界面上有区别，OTA过程是一致的。详细介绍，请参见固件OTA。

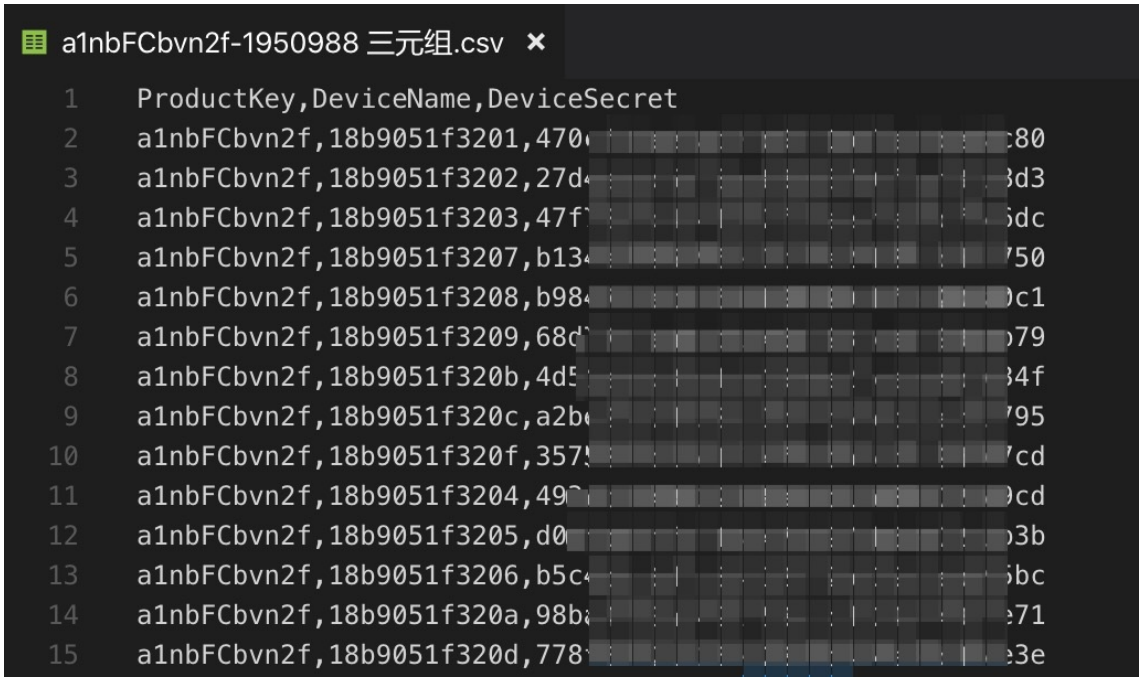
量产设备

前文介绍了新蓝牙辅助配网方案需要在设备中写入设备证书，并且Device Name与Wi-Fi的MAC地址保持一致。测试设备证书可以通过读取开发板的Wi-Fi地址后在**设备调试**页面生成，本节介绍自有品牌项目量产阶段如何生成和获取设备证书。生活物联网平台设备量产介绍，请参见量产流程介绍与量产设备。

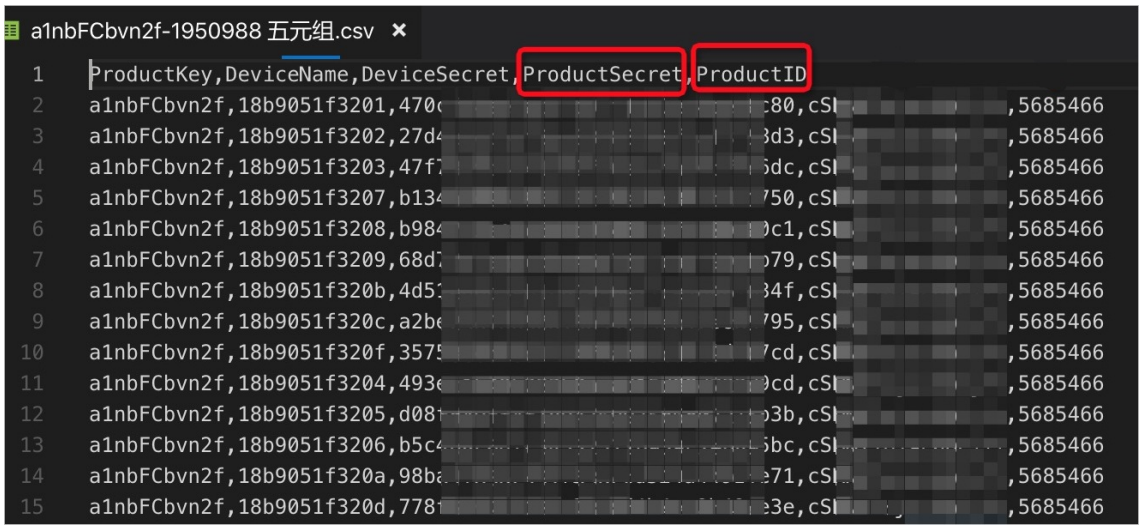
1. 上传MAC地址、生成设备证书。
    i. 对于自有品牌项目的产品，智能生活物联网平台没有MAC地址段提供，需要厂商自己上传MAC地址，作为Device Name来生成设备证书。注意**MAC地址中的字母全部为小写格式**。
    ii. 烧录时通过量产烧录工具写入Device Name同时覆盖芯片Wi-Fi MAC即可实现Device Name与Wi-Fi MAC地址保持一致的要求。
    iii. 自有品牌项目产品在量产设备时，需要准备好符合模版要求的MAC地址段，并选择批量上传。

iv. 下载生成的设备证书清单（csv格式文件）。

2. 在设备证书（默认包括ProductKey，DeviceName， DeviceSecret）中补充 ProductSecret与ProductID。
   - 下载的csv格式的设备证书清单，需要通过文本处理才能生成符合TG7100C批量烧录工具需要的设备证书清单的格式。



- 在设备证书清单中在增加ProductSecret、ProductID的内容。其中第一行的 `ProductSecret,ProductID` 拼写（含大小写）严格保持一致。可以通过文本编辑器或者编写一个文本处理程序来处理。注意不要通过excel编辑，这可能将csv文件转为excel的格式。



3. 烧录设备证书。

   在平头哥芯片开放社区的资源下载页面中，下载五元组量产工具使用说明，按文档将包含设备证书的CSV文件，导入到数据库后进行烧录。

7.3.3. 开发天猫精灵生态项目插座产品

本文以TG7100C芯片为例，介绍在开发天猫精灵生态项目智能插座产品时，使用开发板调试配网连云的过程，以及OTA、量产设备证书等注意事项。

创建与配置产品

1. 创建一个天猫精灵生态项目。具体操作，请参见创建项目。

2. 创建一个品类为插座的产品。

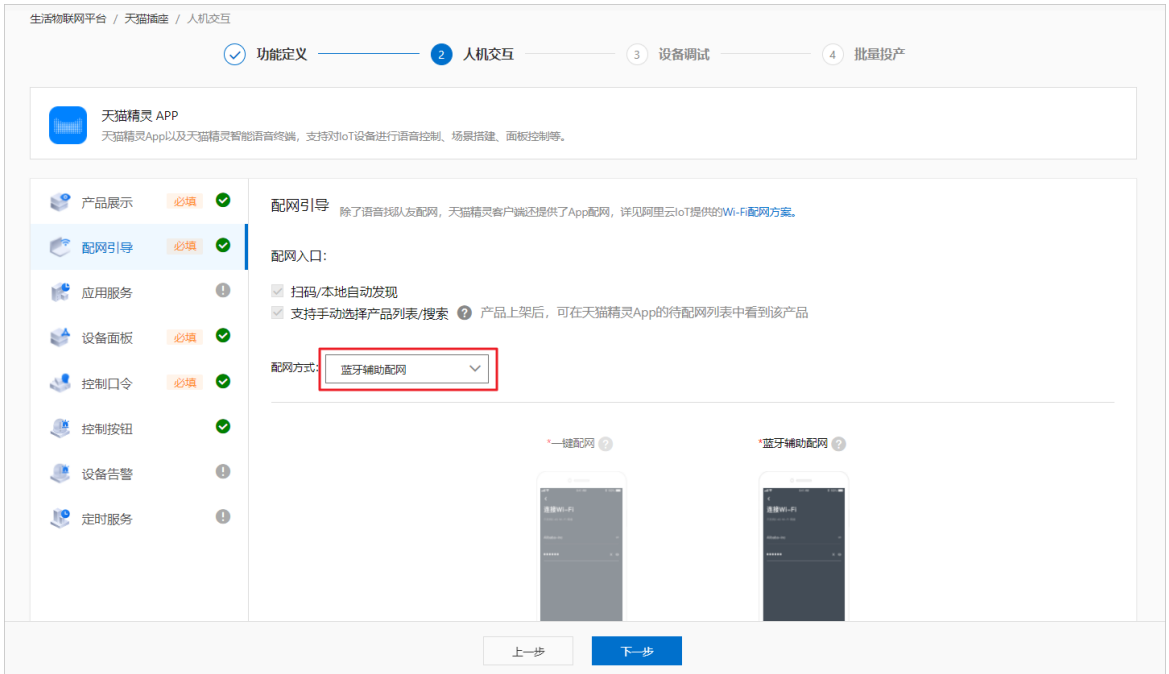
3. 定义产品功能。

   在**功能定义**页面配置产品的功能。

> **说明** 天猫精灵生态项目定时相关的功能需要添加DeviceTimer属性。

4. 配置人机交互。

   如需在TG7100C芯片上使用蓝牙辅助配网，配网方式需选择为蓝牙辅助配网。



   其余配置项的介绍，请参见配置人机交互。

5. 进入**设备调试**界面，单击**新增测试设备**生成测试设备证书。

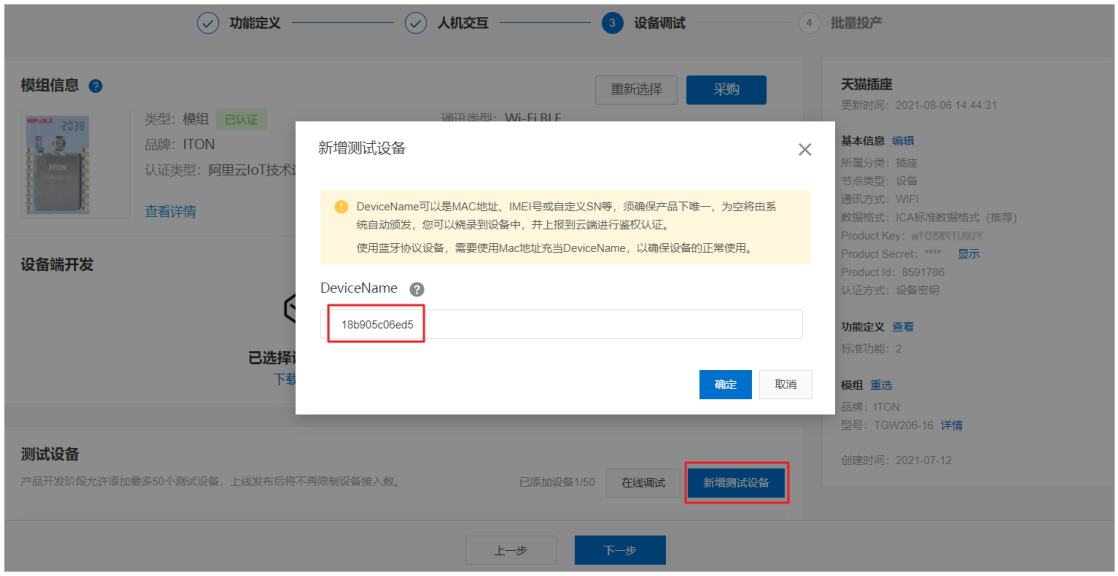
   新蓝牙辅助配网要求Wi-Fi的MAC地址与设备证书中的Device Name保持一致。在TG7100C开发板上，可以在固件运行后通过串口输入mac指令。查询芯片当前的Wi-Fi的MAC地址，并用此MAC地址作为Device Name在设备调试页面申请测试设备证书。

> **说明** MAC地址中的字母全部为小写格式。

i. 查询Wi-Fi的MAC地址示例

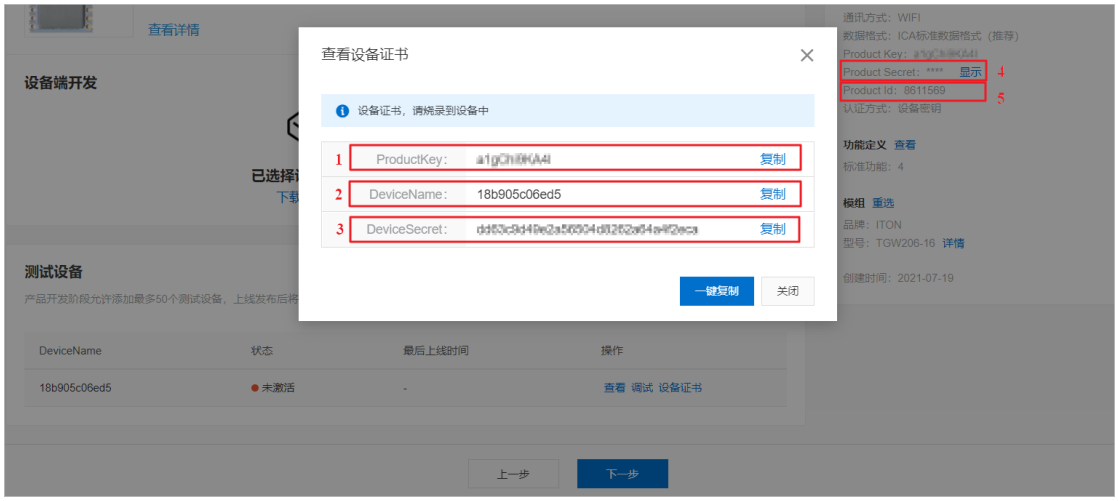
```
#mac
MAC address: 18-b9-05-c0-6e-d5
```

ii. 输入**小写格式的MAC地址**，如示例中的18b905c06ed5作为DeviceName生成测试设备。



iii. 拷贝测试设备证书。

> **说明** 请注意一定要依次完整拷贝1、ProductKey；2、DeviceName；3、DeviceSecret；4、ProductSecret；5、ProductID。



设备配网连云

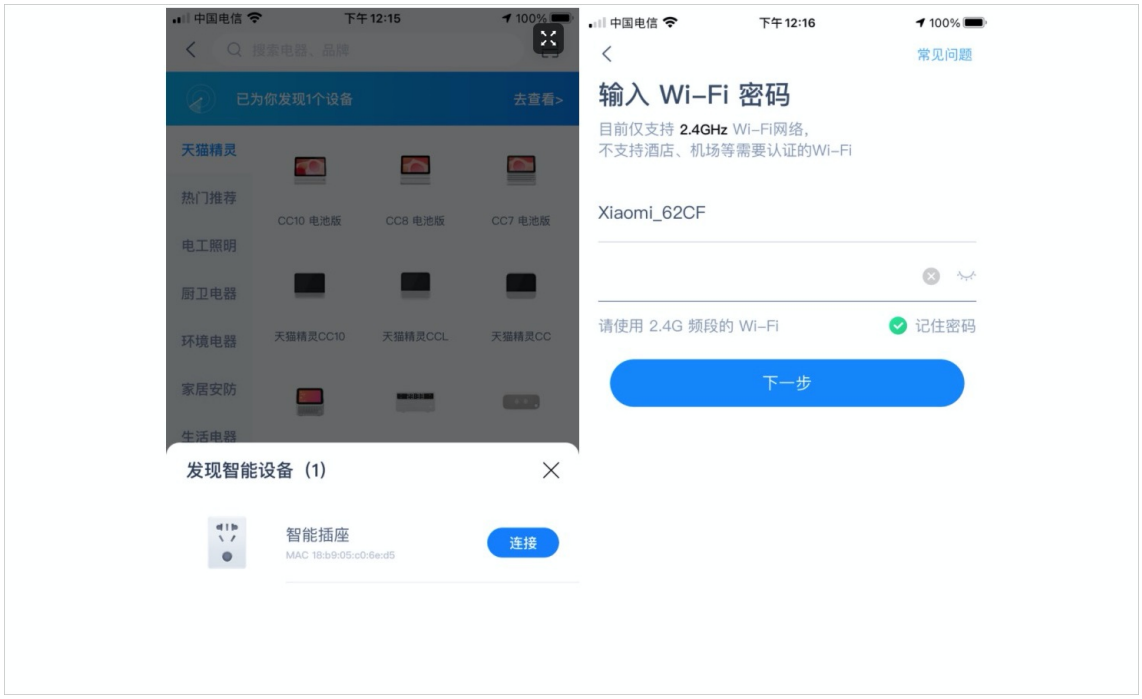
1. 设置设备证书。
   - 将设备固件烧录到开发板之后，可以通过linkkey命令设置设备证书，然后通过reset命令重置设备。
   - 设置设备证书信息，在设备上电后将完整的设备证书信息写入开发板。

```
linkkey ProductKey DeviceName DeviceSecret ProductSecret ProductID
```

- 设备重置，清除设备配网信息。

```
reset
```

2. 设置天猫精灵App配网。
   - 在开发板上设置自有品牌产品的设备证书，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。
   - 天猫精灵App可在其设备中发现页面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。
   - 天猫精灵App配网过程如下图所示。



3. 设置天猫精灵音箱配网。

   通过语音输入天猫精灵找队友即可发起零配。

4. 配网连云设备端关键日志示例。
   - 蓝牙辅助配网

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
开启蓝牙辅助配网	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

配网状态	命令/动作	预期日志
Cloud Connect	连云成功	Cloud Connected

- 零配

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
Dev Scan	awss	IOTX_AWSS_START
Awss Process	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

- 一键配网

配网状态	命令/动作	预期日志
设备重置	reset	start----------hal
Dev Scan	awss	IOTX_AWSS_START
Awss Process	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

本地定时功能

天猫精灵生态项目开发本地定时功能与自有品牌项目有区别，自有品牌项目本地定时功能基于LocalTimer属性，而天猫精灵生态项目本地定时功能基于新的DeviceTimer属性。

因此在控制台定义产品功能的时候，要添加DeviceTimer属性，并确认在面板里添加了预约组件。

控制台配置和设备端开发的详细介绍请参见开发天猫精灵项目Wi-Fi产品本地定时功能。

固件OTA

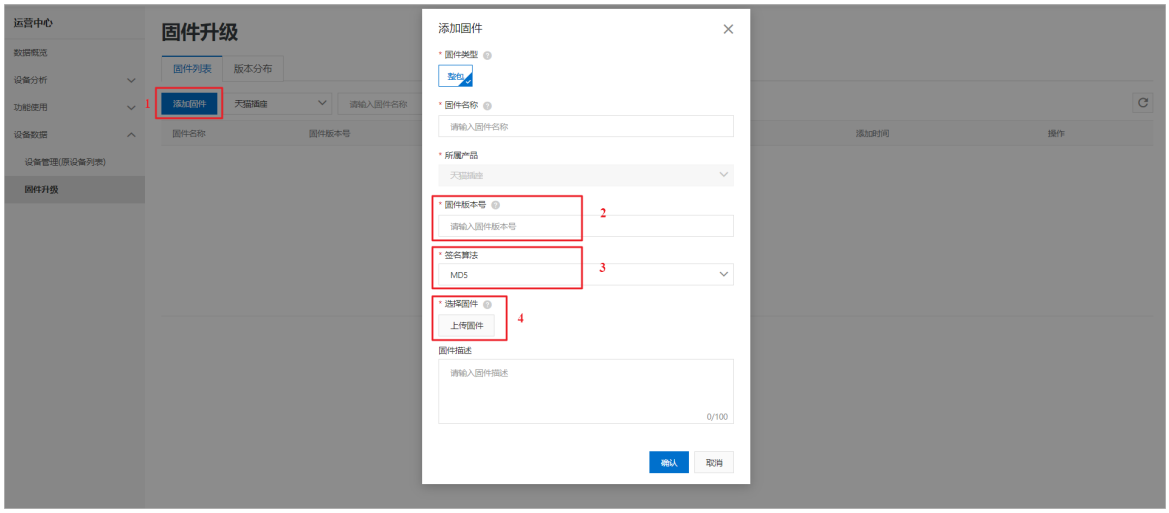
以天猫精灵生态项目产品为例说明OTA的过程。

1. 进入生活物联网平台的运营中心。
2. 进入**设备数据 > 固件升级**页面，选择相应项目下的相应产品。

3. 获取待升级固件文件和版本信息。TG7100C固件编译成功后在*readme.txt*文件中保存固件的版本信息，*tg7100cevb_ota.bin*文件即为OTA的bin文件。

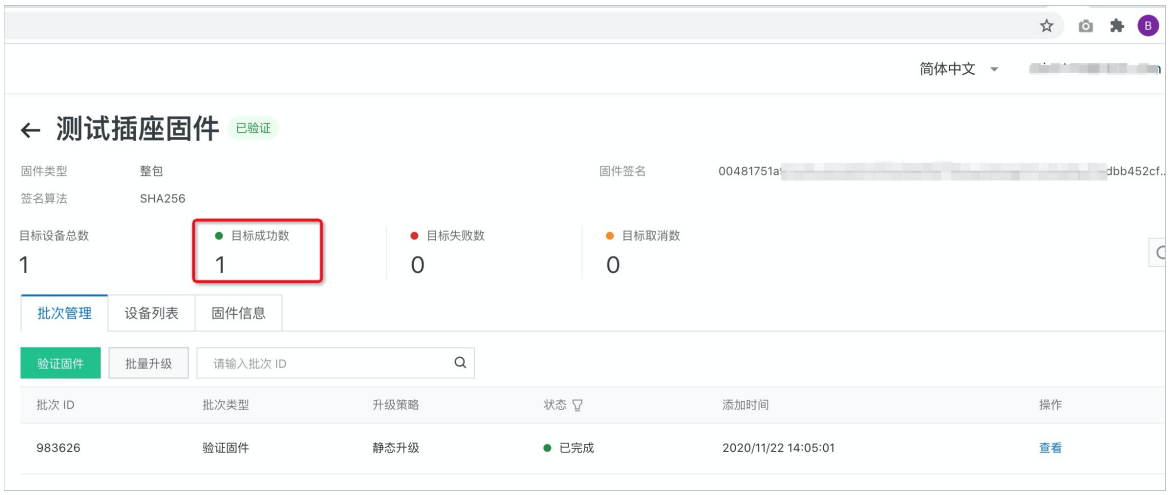
```
spark.yb@[redacted] /home/spark.yb/rel_1.6.6/ali-smartliving-device-alios-things/out/smart_outlet@tg7100cevb
$ls -l
total 16764
-rw-r--r-- 1 spark.yb users       38 Nov 20 16:55 readme.txt
-rwxr-xr-x 1 spark.yb users   853672 Nov 20 16:55 smart_outlet@tg7100cevb.bin
-rwxr-xr-x 1 spark.yb users 10885620 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@[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
```

4. 进入**固件升级**，在**固件列表**窗口下操作下列步骤。
   i. 单击**添加固件**。
   ii. 在固件版本号栏中拷贝并输入**OTA固件版本号**。
   iii. 在签名算法栏中，如TG7100C芯片固件签名算法必须选择SHA256。
   iv. 单击选择固件下的**上传固件**，选择上传待升级的OTA.bin文件。
   v. 单击**确定**完成添加固件。



5. 进入天猫精灵App，在查看待验证设备的设备详情页面单击**立即更新**开始OTA升级。

6. 升级完成后，查看**运营中心**升级状态。



量产设备

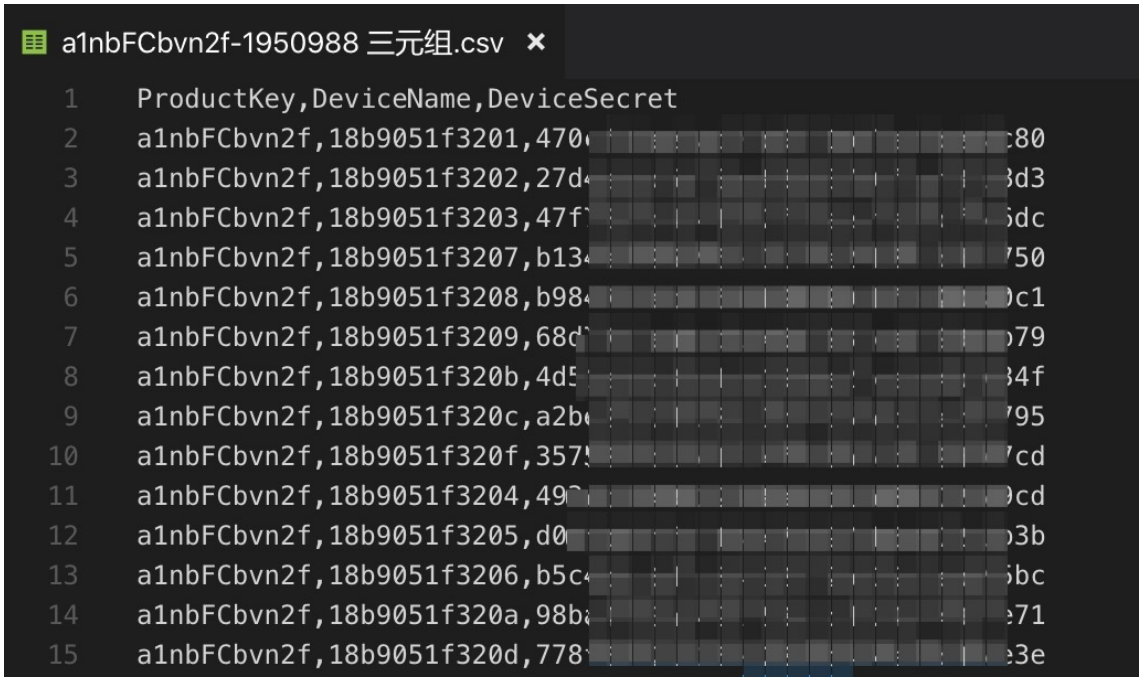
前文介绍了新蓝牙辅助配网方案需要在设备中写入设备证书，并且Device Name与Wi-Fi MAC地址保持一致。测试设备证书可以通过读取开发板的Wi-Fi地址后在**设备调试**页面生成，本节介绍自有品牌项目量产阶段如何生成和获取设备证书。生活物联网平台设备量产详细说明，请参见量产流程介绍与量产设备。

1. 自动生成设备证书。
   - 对于天猫精灵生态项目的产品，生活物联网平台有MAC地址段，在**人机交互**页面的配网引导中，选择蓝牙辅助配网后，生成设备证书时可以分配合法MAC地址作为Device Name。

     > **说明** 注意下载的Device Name（MAC地址）应全部为小写格式，后续的处理流程也要保持小写。

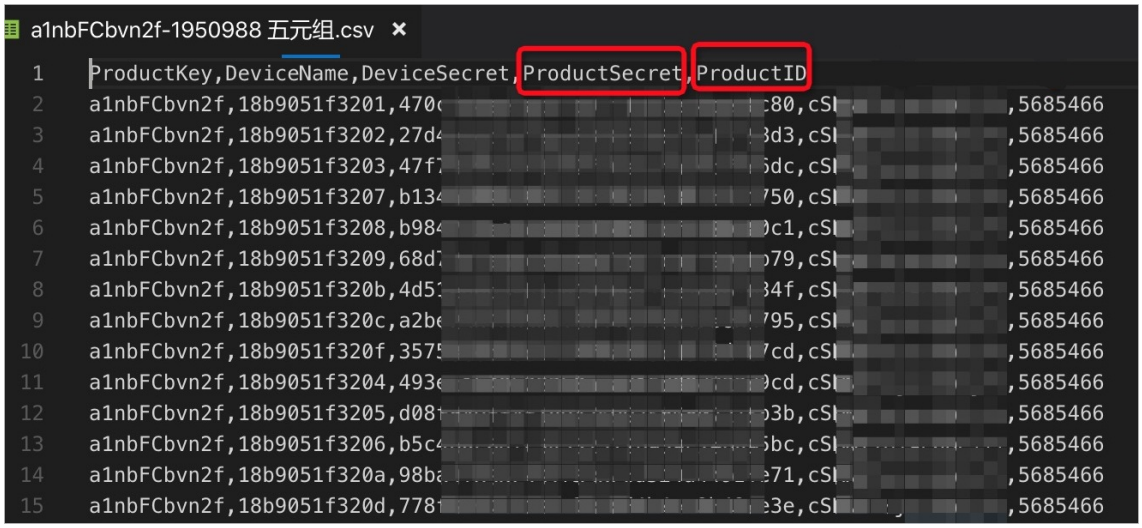
   - 烧录时通过量产烧录工具写入Device Name同时覆盖芯片Wi-Fi MAC即可实现Device Name与Wi-Fi MAC地址保持一致的要求。
   - 天猫精灵生态项目产品在量产设备时，选择自动生成即可。

2. 在设备证书（默认包括ProductKey，DeviceName， DeviceSecret）中补充 ProductSecret与ProductID。
   - 下载的设备证书清单，需要通过文本处理才能生成符合TG7100C批量烧录工具需要的设备证书清单的格式。



- 在设备证书清单中在增加 `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。可以通过文本编辑器或者编写一个文本处理程序来处理。

  > **说明** 注意不要通过Excel编辑，这可能将CSV文件转为Excel的格式。



3. 烧录设备证书。

   在平头哥芯片开放社区的资源下载页面中，下载五元组量产工具使用说明，按文档将包含设备证书的CSV文件，导入到数据库后进行烧录。

7.4. 智能门锁解决方案

阿里云IoT智能门锁解决方案全面赋能合作伙伴，携手打造门锁行业领军解决方案，更好的服务智能门锁品牌厂商。

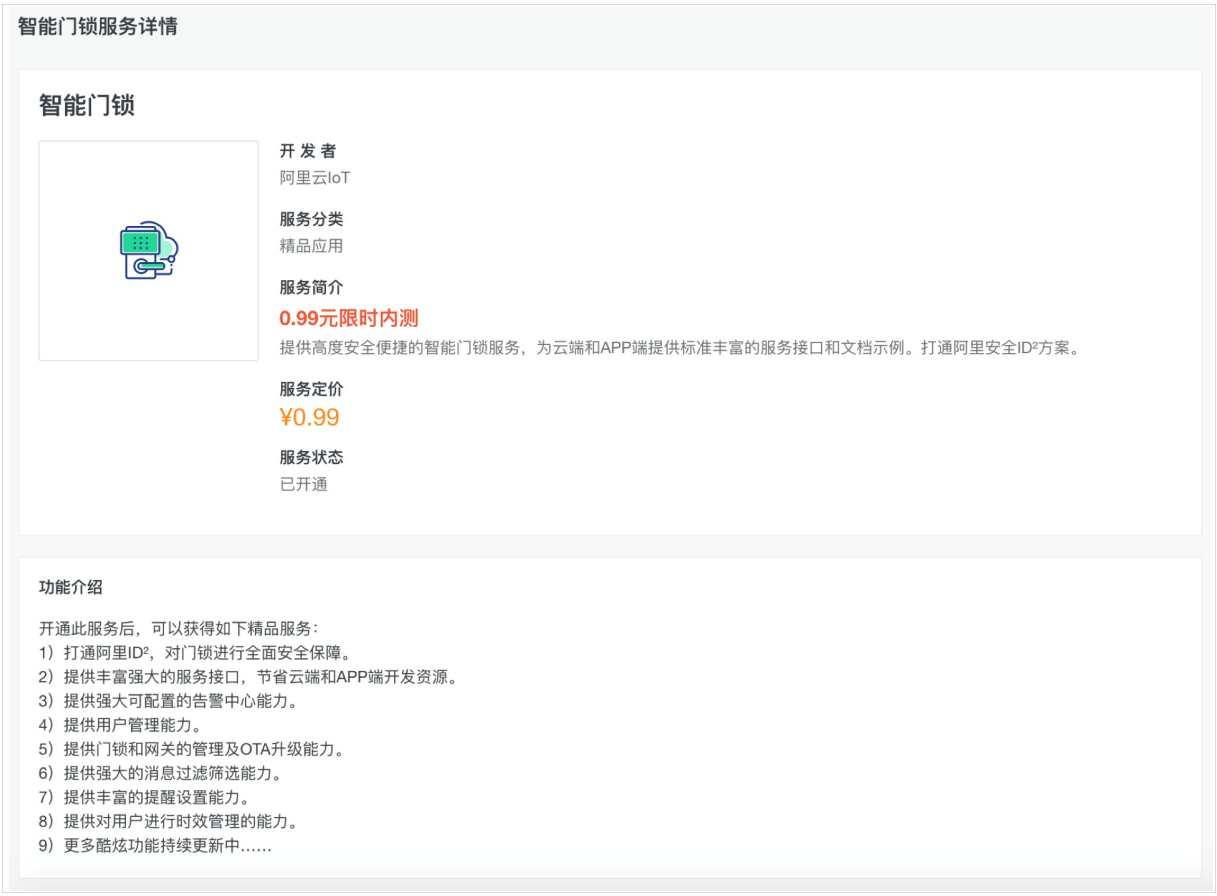
方案简介

阿里云IoT智能门锁解决方案包括两方面核心能力。

- 丰富的云服务接口和告警中心模板
- 阿里ID²的安全保障
  - 软件上有KM安全服务已集成在AliOS中，购买服务即可。
  - 硬件上有SE安全芯片，深度保障门锁安全。

开通智能门锁服务

要想使用智能门锁解决方案，首先要在生活物联网平台的**服务中心**里开通智能门锁服务。详细请参见服务中心。



了解锁接口列表

可以在项目设置中查看包括锁在内的所有接口。

查看锁的接口文档

我们提供了一组智能门锁专用的服务接口，方便对智能门锁相关产品进行快速开发。

安全方案ID²-SE

如果选择ID²-SE硬件级别安全方案，本平台已将ID²安全服务融入在平台接入流程中，客户只需与ID²对接SE芯片即可。查看ID²介绍。

安全方案ID²-KM

如果选择ID²-KM软件级别安全方案，客户无需额外对接，本平台已与ID²做了如下融合打通。

- 已将ID²安全服务融入在平台接入流程中
- 已将ID²-KM软件融入在锁端SDK中（AliOS）

名词解释

虚拟用户：账户下创建的门锁实际的使用用户，例如使用门锁的每个家庭成员都是一个虚拟用户。

7.5. 基于透传协议开发虚拟光照度探测器

为了让开发者在没有硬件的情况下仍能体验平台的透传协议功能，本示例在Linux系统下实现一个虚拟的光照度探测器。

背景信息

虚拟的光照度探测器的主要功能如下。

- 光照度值定期5s上报

  定义属性lux标识当前光照度值，类型为整型，取值范围0～100。

- 当光照值超过设定阈值时触发告警事件

  定义光照过高告警事件OverLuxAlarm，事件参数为当前光照值lux。

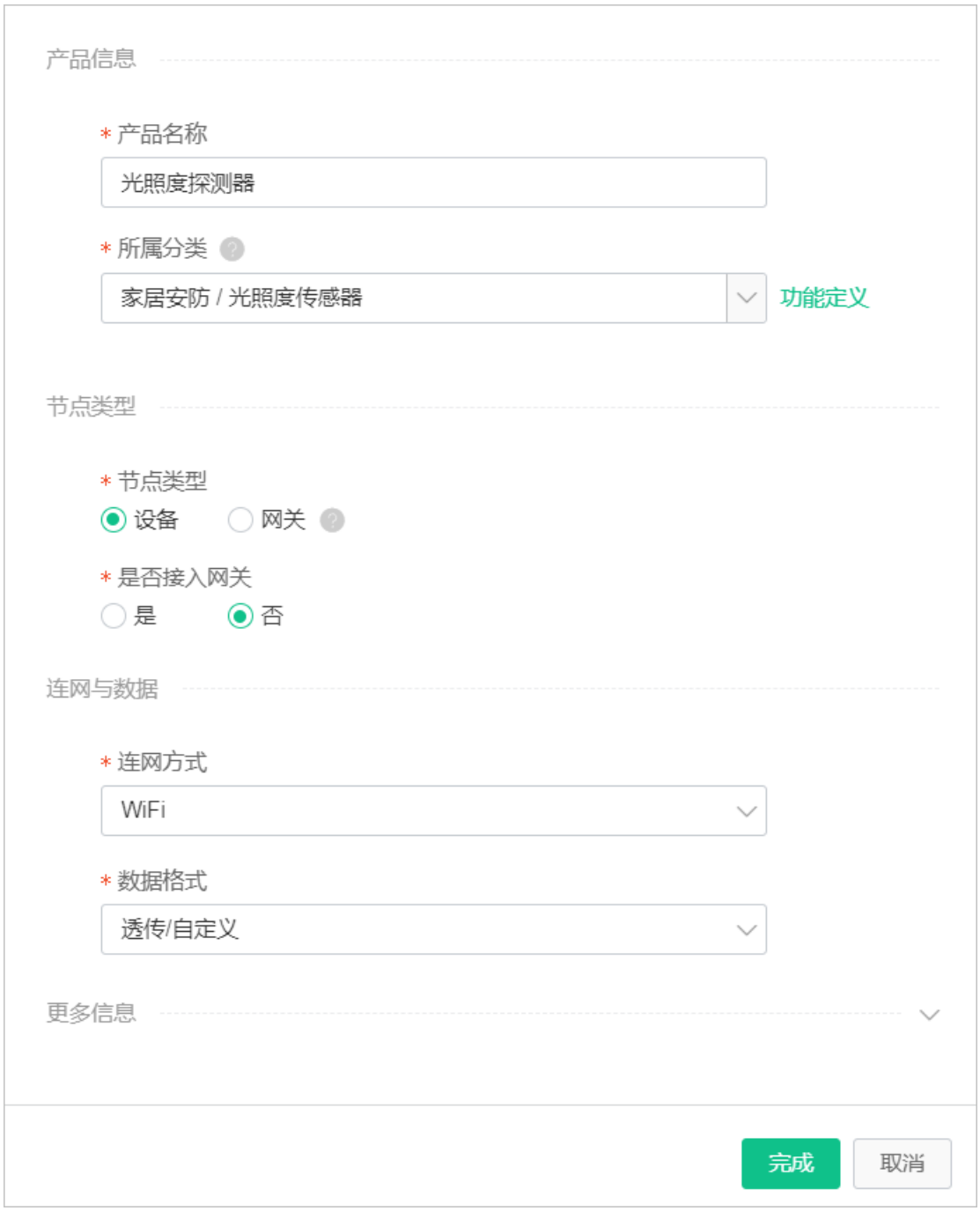
- 通过服务调用设置告警阈值

  定义异步服务SetAlarmThreshold，服务的入参为threshold，整形，表示告警阈值。

操作步骤

1. 创建光照度探测器产品。

   数据格式配置为透传/自定义格式。

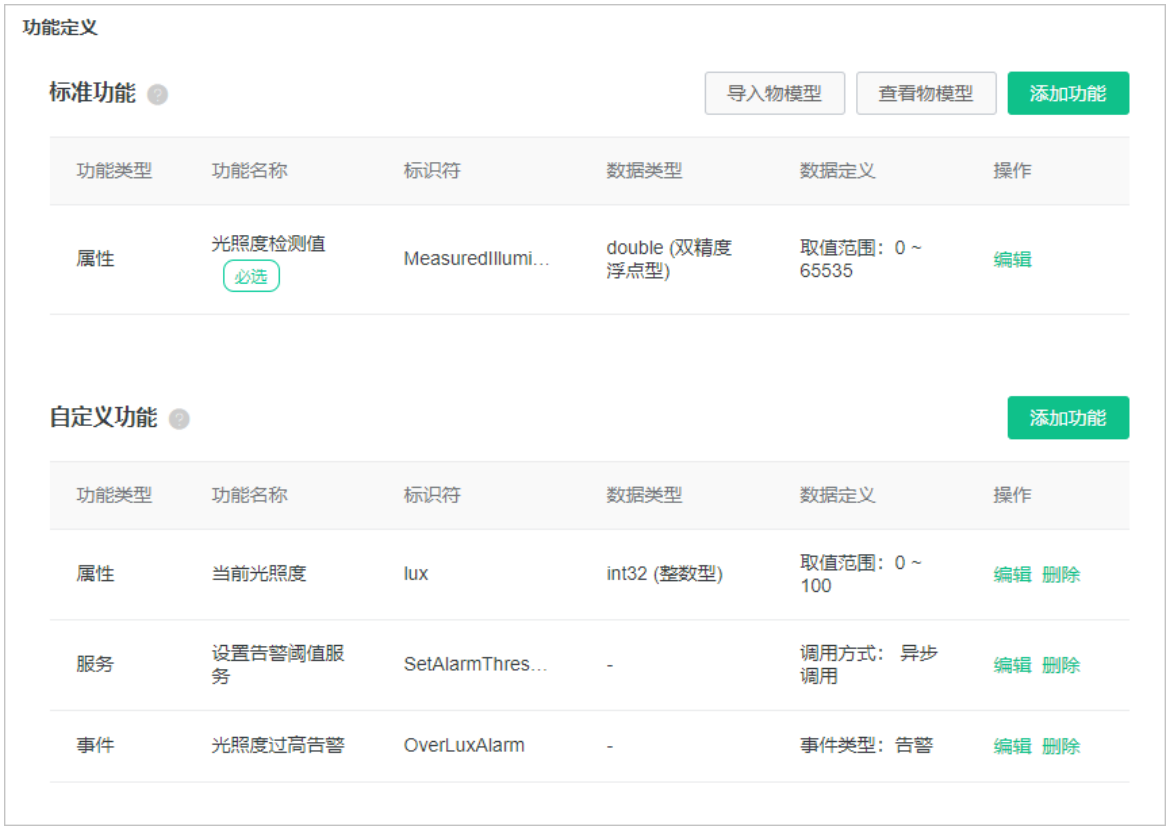


2. 进行产品功能自定义。
   - 定义属性lux标识当前光照度值，类型为整型，取值范围0～100。

- 定义异步服务SetAlarmThreshold，服务入参为threshold，类型为整形，取值范围0 ~ 100。表示告警阈值。

- 定义光照过高告警事件OverLuxAlarm，事件参数为当前光照值lux。

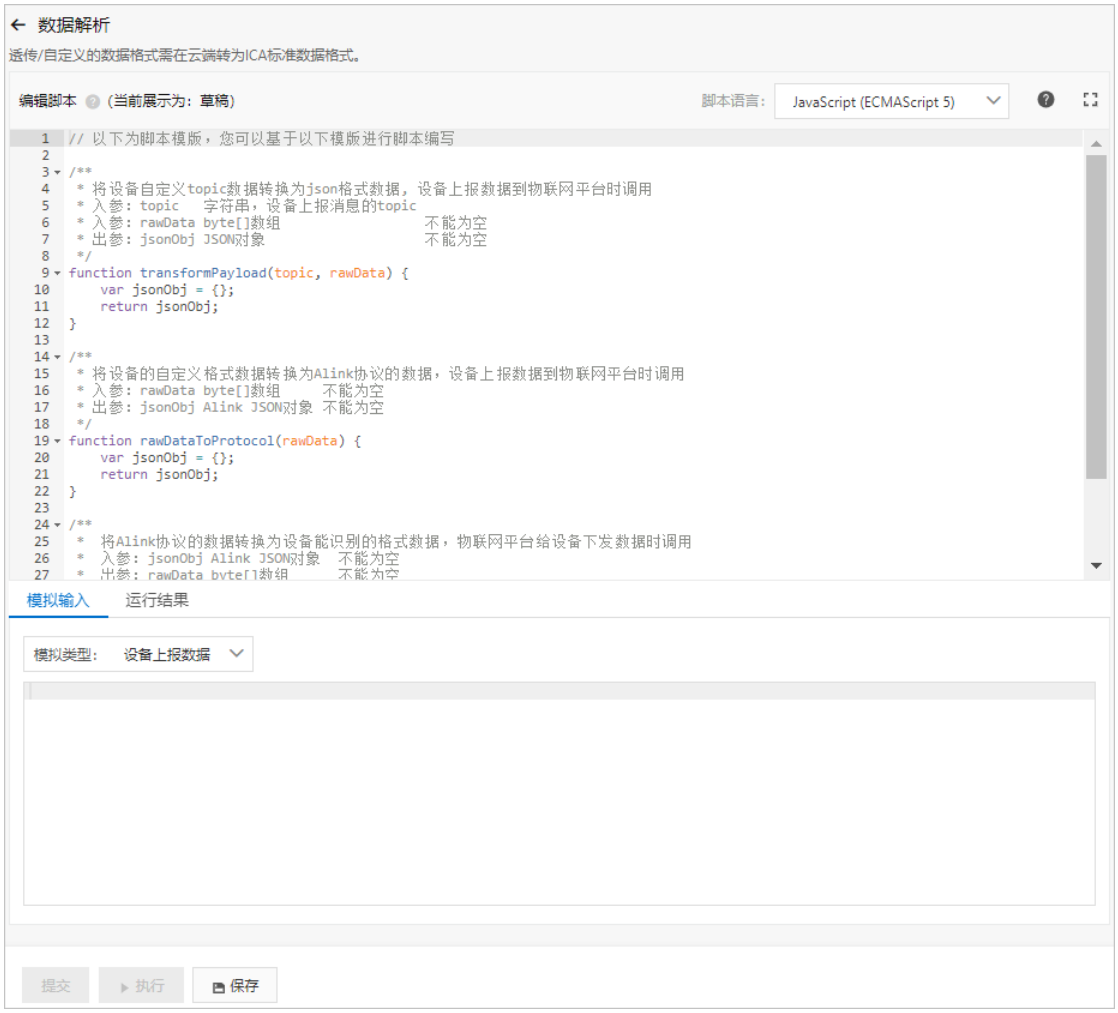
定义完，如下图自定义功能中所示。



3. 添加测试设备。

4. 生成脚本。
   i. 在**设备端开发**页签中，单击数据解析中的**编辑脚本**。

ii. 平台自动生成可编辑脚本。



脚本生成好之后需要进行简单验证之后才能提交生效。

业务逻辑开发

- 属性定时上报

  实现 `userHandler` 函数，放在main函数的 `while(1)` 内调用。

```
void userHandler(void)
{
    Uint8_t lux;
    static Uint32_t last, now;
    sdk_global_t *hsdk = (sdk_global_t *)getSdkObjHandle();
    /* 采集光照度的值，这里随机生成 */
    lux = rand() % 101;  // 取值0~100
    now = hsdk->jiffies;
    if (now - last >= 5000)
    {
        last = now;
        boneSet_lux(lux);   /* 调用设置lux的API */
        bonePostProp();     /* 上报 */
    }
}
Int32_t main(Int32_t argc, char *argv[])
{
    // 此处省略其他代码
    while (1)
    {
        boneSdkRun();
        userHandler();
        usleep(100 * 1000);
    }
```

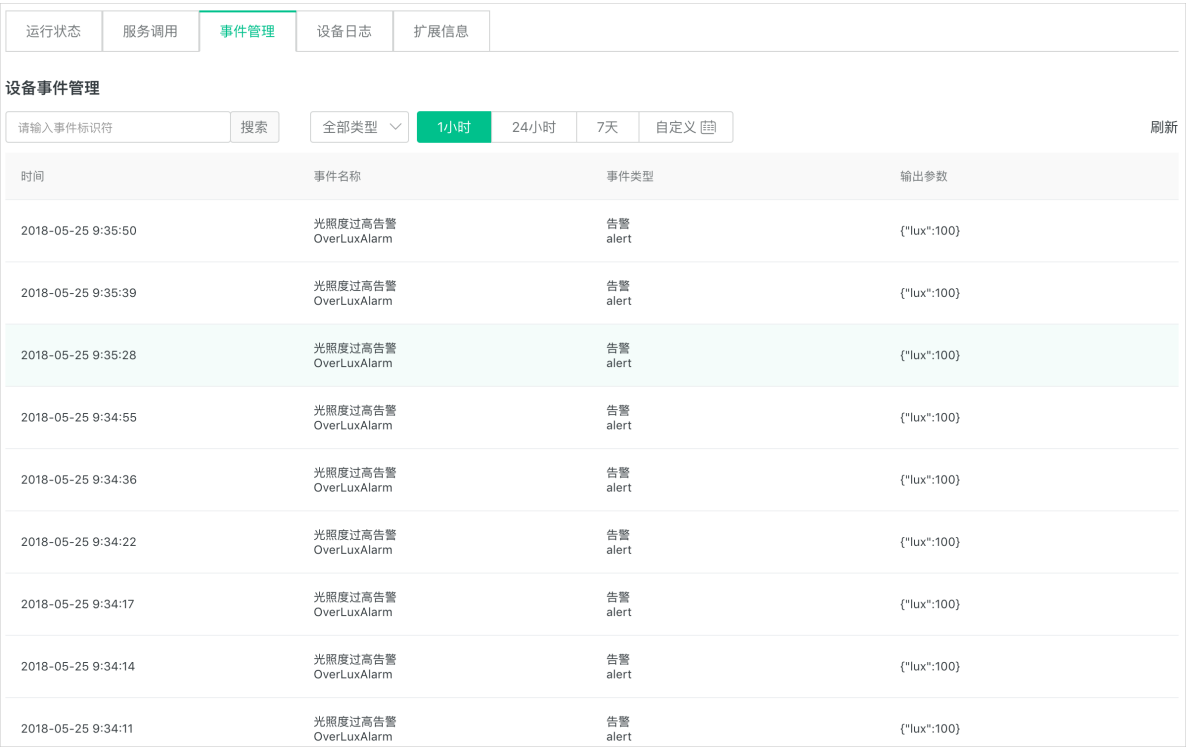
重新将代码编译运行之后可以到云端查看属性的上报日志信息如下。说明：由于使用的软件定时器不是很准，上报的时间间隔并非是严格的5s。

- 光照度过高告警上报

  默认阈值为99，当超过该值时会触发光照度过高告警。

```
Uint8_t threshold = 99;    /* 告警阈值 */
void userHandler(void)
{
    Uint8_t lux;
    static Uint32_t last1, last2, now;
    sdk_global_t *hsdk = (sdk_global_t *)getSdkObjHandle();
    /* 采集光照度的值，这里随机生成 */
    lux = rand() % 101;  // 取值0～100
    now = hsdk->jiffies;
    if ((lux > threshold) && (now - last1) > 2000)
    {
        /* 上报事件，为了避免频繁上报，这里约束事件上报间隔最短2s */
        eo_OverLuxAlarm_t oarg;
        oarg.a_lux = lux;
        boneEvntPost_OverLuxAlarm(&oarg);   /* 调用光照过高告警事件API */
        last1 = now;
    }
    if (now - last2 >= 5000)
    {
        last2 = now;
        boneSet_lux(lux);   /* 调用设置lux的API */
        bonePostProp();     /* 上报 */
    }
}
```

编译执行后，在云端收到当告警信息如下：



- 云端修改告警阈值

  通过云端修改设备上的告警阈值，需要在服务处理接口中添加自己的逻辑代码。

```
extern Uint8_t threshold;
static Int32_t boneServCall_SetAlarmThreshold(Uint32_t id, Uint8_t *data, Uint16_t length
)
{
    Uint8_t len = 0;
    Uint8_t buffer[64] = {0};
    si_SetAlarmThreshold_t *iarg;
    BONE_UNUSED(len);
    BONE_UNUSED(buffer);
    // 1. 解析入参（没有则忽略）
    iarg = (si_SetAlarmThreshold_t*)data;
    // 2. 处理入参（没有则忽略）
    LOG_DBG("threshold = %d\n", iarg->a_threshold);
    threshold = iarg->a_threshold;    // 添加的处理代码
    // 3. 构造出参（没有则忽略）
    // 4. service回复
    return BONE_SUCCESS;
}
```

编译后运行，通过云端调用服务设置告警阈值。

设备端接收日志：

```
[dbg] iotx_mc_handle_recv_PUBLISH(1093):         Packet Ident : 00000000
[dbg] iotx_mc_handle_recv_PUBLISH(1094):         Topic Length : 50
[dbg] iotx_mc_handle_recv_PUBLISH(1098):          Topic Name : /sys/b198prBFE0P/001122334455/thing/model/down_raw
[dbg] iotx_mc_handle_recv_PUBLISH(1101):     Payload Len/Room : 13 / 970
[dbg] iotx_mc_handle_recv_PUBLISH(1102):       Receive Buflen : 1024
[dbg] iotx_mc_handle_recv_PUBLISH(1113): delivering msg ...
[dbg] iotx_mc_deliver_message(868): topic be matched
[mqttRecvMsgArrive-159] Topic: '/sys/b198prBFE0P/001122334455/thing/model/down_raw' (Length: 50)
[getFrameFromRing-193] A protocol frame, 13 Byte
[boneServCall_SetAlarmThreshold-117]> threshold = 89
```

在云端查看事件日志，光照度阈值大于89时均上报了事件。

设备事件管理

请输入事件标识符 | 搜索 | 全部类型 | 1小时 | 24小时 | 7天 | 自定义 | 刷新

时间	事件名称	事件类型	输出参数
2018-05-25 10:01:35	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":96}
2018-05-25 10:01:31	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":100}
2018-05-25 10:01:28	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":99}
2018-05-25 10:01:25	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":93}
2018-05-25 10:01:22	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":92}
2018-05-25 10:01:20	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":92}
2018-05-25 10:01:17	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":100}
2018-05-25 10:01:14	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":93}
2018-05-25 10:01:12	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":95}
2018-05-25 10:01:09	光照度过高告警 OverLuxAlarm	告警 alert	{"lux":99}

至此，基于透传协议的示例已经完成，您可以继续在此平台上的开发之旅。

7.6. 使用已适配IoT SDK的芯片开发智能灯

如您使用已认证过的芯片模组或已经集成IoT SDK的芯片模组，您可以直接快速方便地开发您的产品。

前提条件

- 已从您的芯片模组厂商获取模组的固件代码。
- 已完成产品创建和功能定义。
- 已完成单个调试设备ProductKey、DeviceName、DeviceSecret、ProductSecret的生成。

操作步骤

1. 写入一机一密设备信息。

   按芯片厂商提供的烧录方式烧录，确认您芯片厂商实现的下列函数能正确读取到您的设备ProductKey、DeviceName、DeviceSecret、ProductSecret信息。或者您仅仅为了验证开发流程，可以先将下列HAL函数默认读取您现在生成的设备信息。

   - HAL_GetProductKey
   - HAL_GetDeviceName
   - HAL_GetDeviceSecret
   - HAL_GetProductSecret

2. TSL生成与下载。

   生活物联网平台SDK支持两种获取TSL模型的途径。

   - 直接从云端拉取物的模型

     应用程序调用 `linkkit_start()` 对SDK进行初始化时，需要将get_tsl_from_cloud入参设为1，设备端连云成功后会自动从云端获取。

   - 从本地设置物的模型

     应用程序调用 `linkkit_start()` 对SDK进行初始化时，需要将get_tsl_from_cloud入参设为0且需要调用 `linkit_set_tsl()` 设置物的模型。

     i. 单击创建的产品，进入产品功能定义页面。

ii. 单击**查看物模型**。

iii. TSL字符串转换。

有两种方式可实现TSL字符串转换。

- 将JSON字符串复制到本地，删除所有“ ”（空格）、“\b”、“\n”，将所有的 " 替换为 \" 。
- 使用TSL格式转换工具。
  a. 执行**make reconfig**，选择宿主机类型（ubuntu 、windows）。
  b. 执行**make**，通过编译系统编译TSL格式转换工具。

     编译成功将在*output/release/bin/*目录下生成可执行文件*linkkit_tsl_convert*。
  c. 将控制台中产品TSL模型保存到文件中，使用linkkit_tsl_convert工具对TSL模型格式化。

```
linkkit_tsl_convert (-isrc_path) [-odst_path] [-s]
    -i source file name
    -o destination file name
    -s slim file
//例如从控制台复制的TSL模型保存在文件source.txt，将转换后的文件保存在destination.txt,并对TSL模型进行缩减。其命令为：
linkkit_tsl_convert -isource.txt -odestination.txt -s
```

调用linkkit_set_tsl将转义后的字符作为入参传入，示例如下。

```
const char TSL_STRING[] = "{\"schema\":\"http://aliyun/iot/thing/desc/schema\",\"
profile\":{\"productKey\":\"productKey123\",\"deviceName\":\"deviceName456\"},\"l
ink\":\"/sys/a1AzoSi5TMc/olin_linkkit_test/thing/\",\"services\":[{\"outputData\"
:[],\"identifier\":\"set\",\"inputData\":[{\"identifier\":\"LightSwitch\",\"dataT
ype\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯
开关\"},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Re
d\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\
"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"spec
s\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\
":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unit
Name\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"st
ruct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\
"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},
{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\
",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\
"name\":\"工作模式\"},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\"
:{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type
\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightness\",\"dataType\":{\"
specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"t
ype\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"s
pecs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\
"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"
identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\"
,\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\
"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\"
,\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"
type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HSVColor\",\"dataType\"
:{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"mi
n\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色
调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"mi
n\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱
和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":
\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"
}],\"type\":\"struct\"},\"name\":\"HSV调色\"},{\"identifier\":\"PropertyTime\",\"d
ataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\"},{\"identifier\":\"Pr
opertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\"
:\"double\"},\"name\":\"浮点型\"},{\"identifier\":\"PropertyCharacter\",\"dataType
\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\"}],\"meth
od\":\"thing.service.property.set\",\"name\":\"set\",\"required\":true,\"callType
\":\"sync\",\"desc\":\"属性设置\"},{\"outputData\":[{\"identifier\":\"LightSwitch\
",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"nam
e\":\"主灯开关\"},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identif
ier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":
\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType
```

\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\"},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调色\"},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\"},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\"},{\"identifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\"}],\"identifier\":\"get\",\"inputData\":[\"LightSwitch\",\"RGBColor\",\"NightLightSwitch\",\"WorkMode\",\"ColorTemperature\",\"Brightness\",\"HSLColor\",\"HSVColor\",\"PropertyTime\",\"Propertypoint\",\"PropertyCharacter\"],\"method\":\"thing.service.property.get\",\"name\":\"get\",\"required\":true,\"callType\":\"sync\",\"desc\":\"属性获取\"},{\"outputData\":[{\"identifier\":\"Contrastratio\",\"dataType\":{\"specs\":{\"min\":\"0\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"对比度\"}],\"identifier\":\"Custom\",\"inputData\":[{\"identifier\":\"transparency\",\"dataType\":{\"specs\":{\"min\":\"0\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"透明度\"}],\"method\":\"thing.service.Custom\",\"name\":\"自定义服务\",\"required\":false,\"callType\":\"async\"}],\"properties\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\",\"accessMode\":\"rw\",\"required\":true},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"ColorTemperature\",\"dataTy

pe\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\",\"accessMode\":\"rw\",\"required\":false}],\"events\":[{\"outputData\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\"},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\"},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataT

```
ype\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"1
00\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调
色\"},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date
\"},\"name\":\"时间\"},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{
\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\"},{\"ide
ntifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"ty
pe\":\"text\"},\"name\":\"字符串\"}],\"identifier\":\"post\",\"method\":\"thing.ev
ent.property.post\",\"name\":\"post\",\"type\":\"info\",\"required\":true,\"desc\
":\"属性上报\"},{\"outputData\":[{\"identifier\":\"ErrorCode\",\"dataType\":{\"spe
cs\":{\"0\":\"恢复正常\"},\"type\":\"enum\"},\"name\":\"故障代码\"}],\"identifier\"
:\"Error\",\"method\":\"thing.event.Error.post\",\"name\":\"故障上报\",\"type\":\"
info\",\"required\":true}]}";
int main()
{
    get_tsl_from_cloud  = 0;
    linkkit_start(18, get_tsl_from_cloud, linkkit_loglevel_debug, &alinkops, link
kit_cloud_domain_sh, sample_ctx);
    linkkit_set_tsl(TSL_STRING, strlen(TSL_STRING));
    ...
}
```

3. 编译SDK。

   对于已适配的硬件平台，可以直接编译出适配芯片的SDK lib，以esp32日常环境为例。

   i. 在SDK根目录下执行**make reconfig**，选择2。

```
SELECT A CONFIGURATION:
  1) config.armcc.daily               9) config.h3c.rtk_online
  2) config.esp32.daily              10) config.h3c.rtk_pre
  3) config.esp32.online             11) config.openwrt.c1
  4) config.esp32.pre                12) config.ubuntu.daily
  5) config.esp8266.daily            13) config.ubuntu.online
  6) config.esp8266.online           14) config.ubuntu.online.unittest
  7) config.esp8266.pre              15) config.ubuntu.pre
  8) config.h3c.rtk_daily
  #? 2
  SELECTED CONFIGURATION:
  VENDOR :   esp32
  MODEL  :   daily
  CONFIGURE ............................. [base/log]
  CONFIGURE ............................. [base/tls]
  CONFIGURE ............................. [base/utils]
  CONFIGURE ............................. [connectivity/coap]
  CONFIGURE ............................. [connectivity/examples]
  CONFIGURE ............................. [connectivity/mqtt]
  CONFIGURE ............................. [connectivity/ota]
  CONFIGURE ............................. [connectivity/system]
  CONFIGURE ............................. [external/cut]
  CONFIGURE ............................. [hal-impl]
  CONFIGURE ............................. [layers/cm]
  CONFIGURE ............................. [layers/cm/examples]
  CONFIGURE ............................. [layers/dm]
  CONFIGURE ............................. [layers/linkkit]
  CONFIGURE ............................. [layers/linkkit/samples]
```

```
  CONFIGURE ............................. [modules/alcs]
  CONFIGURE ............................. [modules/awss-ap]
  CONFIGURE ............................. [sdk-tests]
  CONFIGURE ............................. [service/fota]
  BUILDING WITH EXISTING CONFIGURATION:
  VENDOR :   esp32
  MODEL  :   daily
  Components:
  . external/cut
  . connectivity/system
  . connectivity/ota
  . connectivity/mqtt
  . connectivity/coap
  . connectivity/examples
  . layers/dm
  . layers/linkkit
  . layers/linkkit/samples
  . layers/cm
  . layers/cm/examples
  . modules/alcs
  . modules/awss-ap
  . base/log
  . base/utils
  . base/tls
  . hal-impl
  . sdk-tests
  . service/fota
```

ii. 执行**make**，生成*libilop-esp32.a*。

```
make
  BUILDING WITH EXISTING CONFIGURATION:
  VENDOR :   esp32
  MODEL  :   daily
  [AR] libilop-hal.a              <=
  [CC] mqtt-example.o             <= mqtt-example.c

                                     makefile
  [CC] ut_json_parser.o           <= ut_json_parser.c

  [CC] ut_mem_stats.o             <= ut_mem_stats.c

  [CC] ut_json_token.o            <= ut_json_token.c

  [CC] utils_base64.o             <= utils_base64.c
  ...
  [AR] libilop-esp32.a            <=
                                     base/log/lite-log.o
                                     base/utils/json_parser.o
                                     base/utils/json_token.o
                                     base/utils/linked_list.o
                                     base/utils/mem_stats.o
                                     base/utils/string_utils.o
                                     base/utils/utils_base64.o
```

```
base/utils/utils_epoch_time.o
base/utils/utils_hmac.o
base/utils/utils_httpc.o
base/utils/utils_list.o
base/utils/utils_md5.o
base/utils/utils_net.o
base/utils/utils_sha1.o
base/utils/utils_timer.o
base/utils/work_queue.o
connectivity/coap/alcs_api.o
connectivity/coap/alcs_client.o
connectivity/coap/alcs_coap.o
connectivity/coap/alcs_server.o
connectivity/coap/CoAPDeserialize.o
connectivity/coap/CoAPExport.o
connectivity/coap/CoAPMessage.o
connectivity/coap/CoAPNetwork.o
connectivity/coap/CoAPObserve.o
connectivity/coap/CoAPPlatform.o
connectivity/coap/CoAPResource.o
connectivity/coap/CoAPSerialize.o
connectivity/coap/CoAPServer.o
connectivity/mqtt/mqtt_client.o
connectivity/mqtt/MQTTConnectClient.o
connectivity/mqtt/MQTTDeserializePublish.o
connectivity/mqtt/mqtt_instance.o
connectivity/mqtt/MQTTPacket.o
connectivity/mqtt/MQTTSerializePublish.o
connectivity/mqtt/MQTTSubscribeClient.o
connectivity/mqtt/MQTTUnsubscribeClient.o
connectivity/ota/ota.o
connectivity/system/ca.o
connectivity/system/class_interface.o
connectivity/system/device.o
connectivity/system/guider.o
connectivity/system/id2_guider.o
connectivity/system/report.o
connectivity/system/sdk-impl.o
layers/cm/iotx_cloud_conn_coap.o
layers/cm/iotx_cloud_conn_http.o
layers/cm/iotx_cloud_conn_mqtt.o
layers/cm/iotx_cm_api.o
layers/cm/iotx_cm_cloud_conn.o
layers/cm/iotx_cm_common.o
layers/cm/iotx_cm_local_conn.o
layers/cm/iotx_cm_log.o
layers/cm/iotx_cm_ota.o
layers/cm/iotx_local_conn_alcs.o
layers/dm/cJSON.o
layers/dm/dm_cm_impl.o
layers/dm/dm_cm_msg_info.o
layers/dm/dm_impl.o
layers/dm/dm_logger.o
layers/dm/dm_slist.o
```

```
                                        layers/dm/dm_thing_manager.o
                                        layers/dm/dm_thing.o
                                        layers/linkkit/linkkit_export.o
                                        layers/linkkit/lite_queue.o
                                        modules/alcs/alcs_adapter.o
                                        modules/alcs/alcs_mqtt.o
                                        service/fota/service_ota.o
```

iii. 将步骤1的ProductKey、DeviceName、DeviceSecret、ProductSecret分别写入目标平台的HAL_GetProductKey、HAL_GetDeviceName、HAL_GetDeviceSecret、HAL_GetProductSecret函数中。

Link Kit SDK 提供的API位于*layers/linkkit/include/linkkit_export.h*，完成设备功能的编写后，您可以链接libilop-*目标平台*.a编译生成固件运行您的程序。

示例代码

```
/** USER NOTIFICATION
 *  this sample code is only used for evaluation or test of the iLop project.
 *  Users should modify this sample code freely according to the product/device TSL, like
 *  property/event/service identifiers, and the item value type(type, length, etc...).
 *  Create user's own execution logic for specific products.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <stdarg.h>
#include <unistd.h>
#include "linkkit_export.h"
#include "iot_import.h"
/*
 * example for product "灯-Demo"
 */
#define LINKKIT_PRINTF(...)  \
    do {                                                \
        printf("\e[0;32m%s@line%d\t:", __FUNCTION__, __LINE__);  \
        printf(__VA_ARGS__);                            \
        printf("\e[0m");                                \
    } while (0)
/* identifier of property/service/event, users should modify this macros according to your
own product TSL. */
#define EVENT_PROPERTY_POST_IDENTIFIER         "post"
#define EVENT_ERROR_IDENTIFIER                 "Error"
#define EVENT_ERROR_OUTPUT_INFO_IDENTIFIER     "ErrorCode"
#define EVENT_CUSTOM_IDENTIFIER                "Custom"
/* specify ota buffer size for ota service, ota service will use this buffer for bin downlo
ad. */
#define OTA_BUFFER_SIZE                  (512+1)
/* PLEASE set RIGHT tsl string according to your product. */
const char TSL_STRING[] = "{\"schema\":\"http://aliyun/iot/thing/desc/schema\",\"profile\":
{\"productKey\":\"productKey123\",\"deviceName\":\"deviceName456\"},\"link\":\"/sys/a1AzoSi
5TMc/olin_linkkit_test/thing/\",\"services\":[{\"outputData\":[],\"identifier\":\"set\",\"i
nputData\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"
```

开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\"},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\"},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调色\"},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\"},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\"},{\"identifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\"}],\"method\":\"thing.service.property.set\",\"name\":\"set\",\"required\":true,\"callType\":\"sync\",\"desc\":\"属性设置\"},{\"outputData\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\"},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\"},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"

},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调色\"},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\"},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\"},{\"identifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\"}],\"identifier\":\"get\",\"inputData\":[\"LightSwitch\",\"RGBColor\",\"NightLightSwitch\",\"WorkMode\",\"ColorTemperature\",\"Brightness\",\"HSLColor\",\"HSVColor\",\"PropertyTime\",\"Propertypoint\",\"PropertyCharacter\"],\"method\":\"thing.service.property.get\",\"name\":\"get\",\"required\":true,\"callType\":\"sync\",\"desc\":\"属性获取\"},{\"outputData\":[{\"identifier\":\"Contrastratio\",\"dataType\":{\"specs\":{\"min\":\"0\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"对比度\"}],\"identifier\":\"Custom\",\"inputData\":[{\"identifier\":\"transparency\",\"dataType\":{\"specs\":{\"min\":\"0\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"透明度\"}],\"method\":\"thing.service.Custom\",\"name\":\"自定义服务\",\"required\":false,\"callType\":\"async\"}],\"properties\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\",\"accessMode\":\"rw\",\"required\":true},{\"identifier\":\"RGBColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"struct\"},\"name\":\"RGB调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"WorkMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"ColorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"Brightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明暗度\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"HSLColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"unitName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"HSVColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调色\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name\":\"时间\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"max\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\",\"accessMode\":\"rw\",\"required\":false},{\"identifier\":\"PropertyCharacter\",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\",\"accessMode\":\"rw\",\"required\":false}],\"events\":[{\"outputData\":[{\"identifier\":\"LightSwitch\",\"dataType\":{\"specs\":{\"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"主灯开关\"},{\"identifier\":\"RGBColo

r\",\"dataType\":{\"specs\":[{\"identifier\":\"Red\",\"dataType\":{\"specs\":{\"min\":\"0\"
,\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"红色\"},{\"identifier\":\
"Green\",\"dataType\":{\"specs\":{\"min\":\"0\",\"unitName\":\"无\",\"max\":\"255\"},\"type
\":\"int\"},\"name\":\"绿色\"},{\"identifier\":\"Blue\",\"dataType\":{\"specs\":{\"min\":\"0
\",\"unitName\":\"无\",\"max\":\"255\"},\"type\":\"int\"},\"name\":\"蓝色\"}],\"type\":\"str
uct\"},\"name\":\"RGB调色\"},{\"identifier\":\"NightLightSwitch\",\"dataType\":{\"specs\":{\
"0\":\"关闭\",\"1\":\"开启\"},\"type\":\"bool\"},\"name\":\"夜灯开关\"},{\"identifier\":\"Wor
kMode\",\"dataType\":{\"specs\":{\"0\":\"手动\",\"1\":\"阅读\",\"2\":\"影院\",\"3\":\"夜灯\",
\"4\":\"生活\",\"5\":\"柔和\"},\"type\":\"enum\"},\"name\":\"工作模式\"},{\"identifier\":\"Co
lorTemperature\",\"dataType\":{\"specs\":{\"unit\":\"K\",\"min\":\"2700\",\"unitName\":\"开
尔文\",\"max\":\"6500\"},\"type\":\"int\"},\"name\":\"冷暖色温\"},{\"identifier\":\"Brightnes
s\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"
100\"},\"type\":\"int\"},\"name\":\"明暗度\"},{\"identifier\":\"HSLColor\",\"dataType\":{\"s
pecs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\":\"°\",\"min\":\"0\",\"uni
tName\":\"度\",\"max\":\"360\"},\"type\":\"int\"},\"name\":\"色调\"},{\"identifier\":\"Satur
ation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\
":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifier\":\"Lightness\",\"dataType\"
:{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":
\"int\"},\"name\":\"亮度\"}],\"type\":\"struct\"},\"name\":\"HSL调色\"},{\"identifier\":\"HS
VColor\",\"dataType\":{\"specs\":[{\"identifier\":\"Hue\",\"dataType\":{\"specs\":{\"unit\"
:\"%\",\"min\":\"0\",\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"
色调\"},{\"identifier\":\"Saturation\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\"
,\"unitName\":\"百分比\",\"max\":\"100\"},\"type\":\"int\"},\"name\":\"饱和度\"},{\"identifie
r\":\"Value\",\"dataType\":{\"specs\":{\"unit\":\"%\",\"min\":\"0\",\"unitName\":\"百分比\",
\"max\":\"100\"},\"type\":\"int\"},\"name\":\"明度\"}],\"type\":\"struct\"},\"name\":\"HSV调
色\"},{\"identifier\":\"PropertyTime\",\"dataType\":{\"specs\":{},\"type\":\"date\"},\"name
\":\"时间\"},{\"identifier\":\"Propertypoint\",\"dataType\":{\"specs\":{\"min\":\"-100\",\"m
ax\":\"100\"},\"type\":\"double\"},\"name\":\"浮点型\"},{\"identifier\":\"PropertyCharacter\
",\"dataType\":{\"specs\":{\"length\":\"255\"},\"type\":\"text\"},\"name\":\"字符串\"}],\"id
entifier\":\"post\",\"method\":\"thing.event.property.post\",\"name\":\"post\",\"type\":\"i
nfo\",\"required\":true,\"desc\":\"属性上报\"},{\"outputData\":[{\"identifier\":\"ErrorCode\
",\"dataType\":{\"specs\":{\"0\":\"恢复正常\"},\"type\":\"enum\"},\"name\":\"故障代码\"}],\"i
dentifier\":\"Error\",\"method\":\"thing.event.Error.post\",\"name\":\"故障上报\",\"type\":\
"info\",\"required\":true}]}";

```
/* user sample context struct. */
typedef struct _sample_context {
    void* thing;
    int cloud_connected;
    int thing_enabled;
    int service_custom_input_transparency;
    int service_custom_output_contrastratio;
#ifdef SERVICE_OTA_ENABLED
    char ota_buffer[OTA_BUFFER_SIZE];
#endif /* SERVICE_OTA_ENABLED */
} sample_context_t;
sample_context_t g_sample_context;
#ifdef SERVICE_OTA_ENABLED
/* callback function for fota service. */
static void fota_callback(service_fota_callback_type_t callback_type, const char* version)
{
    sample_context_t* sample;
    assert(callback_type < service_fota_callback_type_number);
    sample = &g_sample_context;
    /* temporarily disable thing when ota service invoked */
```

```
    sample->thing_enabled = 0;
    linkkit_invoke_ota_service(sample->ota_buffer, OTA_BUFFER_SIZE);
    sample->thing_enabled = 1;
    /* reboot the device... */
}
#endif /* SERVICE_OTA_ENABLED */
static int on_connect(void* ctx)
{
    sample_context_t* sample = ctx;
    sample->cloud_connected = 1;
    LINKKIT_PRINTF("cloud is connected\n");
    return 0;
}
static int on_disconnect(void* ctx)
{
    sample_context_t* sample = ctx;
    sample->cloud_connected = 0;
    LINKKIT_PRINTF("cloud is disconnect\n");
    return 0;
}
static int raw_data_arrived(void* thing_id, void* data, int len, void* ctx)
{
    char raw_data[128] = {0};
    LINKKIT_PRINTF("raw data arrived,len:%d\n", len);
    /* do user's raw data process logical here. */
    /* ............................. */
    /* user's raw data process logical complete */
    snprintf(raw_data, sizeof(raw_data), "test down raw reply data %lld", HAL_UptimeMs());
    linkkit_invoke_raw_service(thing_id, 0, raw_data, strlen(raw_data));
    return 0;
}
static int thing_create(void* thing_id, void* ctx)
{
    sample_context_t* sample = ctx;
    LINKKIT_PRINTF("new thing@%p created.\n", thing_id);
    sample->thing = thing_id;
    return 0;
}
static int thing_enable(void* thing_id, void* ctx)
{
    sample_context_t* sample = ctx;
    sample->thing_enabled = 1;
    return 0;
}
static int thing_disable(void* thing, void* ctx)
{
    sample_context_t* sample = ctx;
    sample->thing_enabled = 0;
    return 0;
}
#ifdef RRPC_ENABLED
static int handle_service_custom(sample_context_t* sample, void* thing, char* service_ident
ifier, int request_id, int rrpc)
#else
static int handle_service_custom(sample_context_t* sample, void* thing, char* service_ident
```

```
static int handle_service_custom(sample_context_t *sample, void *thing, char *service_iden
ifier, int request_id)
#endif /* RRPC_ENABLED */
{
    char identifier[128] = {0};
    /*
     * get iutput value.
     */
    snprintf(identifier, sizeof(identifier), "%s.%s", service_identifier, "transparency");
    linkkit_get_value(linkkit_method_get_service_input_value, thing, identifier, &sample->s
ervice_custom_input_transparency, NULL);
    /*
     * set output value according to user's process result.
     */
    snprintf(identifier, sizeof(identifier), "%s.%s", service_identifier, "Contrastratio");
    sample->service_custom_output_contrastratio = sample->service_custom_input_transparency
>= 0 ? sample->service_custom_input_transparency : sample->service_custom_input_transparenc
y * -1;
    linkkit_set_value(linkkit_method_set_service_output_value, thing, identifier, &sample->
service_custom_output_contrastratio, NULL);
#ifdef RRPC_ENABLED
    linkkit_answer_service(thing, service_identifier, request_id, 200, rrpc);
#else
    linkkit_answer_service(thing, service_identifier, request_id, 200);
#endif /* RRPC_ENABLED */
    return 0;
}
#ifdef RRPC_ENABLED
static int thing_call_service(void* thing_id, char* service, int request_id, int rrpc, void
* ctx)
#else
static int thing_call_service(void* thing_id, char* service, int request_id, void* ctx)
#endif /* RRPC_ENABLED */
{
    sample_context_t* sample = ctx;
    LINKKIT_PRINTF("service(%s) requested, id: thing@%p, request id:%d\n",
                   service, thing_id, request_id);
    if (strcmp(service, "Custom") == 0) {
#ifdef RRPC_ENABLED
        handle_service_custom(sample, thing_id, service, request_id, rrpc);
#else
        handle_service_custom(sample, thing_id, service, request_id);
#endif /* RRPC_ENABLED */
    }
    return 0;
}
static int thing_prop_changed(void* thing_id, char* property, void* ctx)
{
    char* value_str = NULL;
    char property_buf[64] = {0};
    /* get new property value */
    if (strstr(property, "HSVColor") != 0) {
        int hue, saturation, value;
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Hue");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &hue,
```

```
&value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Saturation");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &satur
ation, &value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Value");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &value
, &value_str);
        LINKKIT_PRINTF("property(%s), Hue:%d, Saturation:%d, Value:%d\n", property, hue, sa
turation, value);
        /* XXX: do user's process logical here. */
    } else if (strstr(property, "HSLColor") != 0) {
        int hue, saturation, lightness;
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Hue");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &hue,
&value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Saturation");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &satur
ation, &value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Lightness");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &light
ness, &value_str);
        LINKKIT_PRINTF("property(%s), Hue:%d, Saturation:%d, Lightness:%d\n", property, hue
, saturation, lightness);
        /* XXX: do user's process logical here. */
    }  else if (strstr(property, "RGBColor") != 0) {
        int red, green, blue;
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Red");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &red,
&value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Green");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &green
, &value_str);
        snprintf(property_buf, sizeof(property_buf), "%s.%s", property, "Blue");
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property_buf, &blue,
&value_str);
        LINKKIT_PRINTF("property(%s), Red:%d, Green:%d, Blue:%d\n", property, red, green, b
lue);
        /* XXX: do user's process logical here. */
    } else {
        linkkit_get_value(linkkit_method_get_property_value, thing_id, property, NULL, &val
ue_str);
        LINKKIT_PRINTF("#### property(%s) new value set: %s ####\n", property, value_str);
    }
    /* do user's process logical here. */
    linkkit_trigger_event(thing_id, EVENT_PROPERTY_POST_IDENTIFIER, property);
    return 0;
}
static linkkit_ops_t alinkops = {
    .on_connect         = on_connect,
    .on_disconnect      = on_disconnect,
    .raw_data_arrived   = raw_data_arrived,
    .thing_create       = thing_create,
    .thing_enable       = thing_enable,
    .thing_disable      = thing_disable,
```

```
    .thing_call_service = thing_call_service,
    .thing_prop_changed = thing_prop_changed,
};
static unsigned long long uptime_sec(void)
{
    static unsigned long long start_time = 0;
    if (start_time == 0) {
        start_time = HAL_UptimeMs();
    }
    return (HAL_UptimeMs() - start_time) / 1000;
}
#if 0
static int post_all_prop(sample_context_t* sample)
{
    return linkkit_trigger_event(sample->thing, EVENT_PROPERTY_POST_IDENTIFIER, NULL);
}
#endif
static int post_event_error(sample_context_t* sample)
{
    char event_output_identifier[64];
    snprintf(event_output_identifier, sizeof(event_output_identifier), "%s.%s", EVENT_ERROR
_IDENTIFIER, EVENT_ERROR_OUTPUT_INFO_IDENTIFIER);
    int errorCode = 0;
    linkkit_set_value(linkkit_method_set_event_output_value,
                      sample->thing,
                      event_output_identifier,
                      &errorCode, NULL);
    return linkkit_trigger_event(sample->thing, EVENT_ERROR_IDENTIFIER, NULL);
}
#if 0
static int post_event_fault_alert(sample_context_t* sample)
{
    char event_output_identifier[64];
    snprintf(event_output_identifier, sizeof(event_output_identifier), "%s.%s", EVENT_ERROR
_IDENTIFIER, EVENT_ERROR_OUTPUT_INFO_IDENTIFIER);
    int errorCode = 0;
    linkkit_set_value(linkkit_method_set_event_output_value,
                      sample->thing,
                      event_output_identifier,
                      &errorCode, NULL);
    return linkkit_trigger_event(sample->thing, EVENT_ERROR_IDENTIFIER, NULL);
}
static int upload_raw_data(sample_context_t* sample)
{
    char raw_data[128] = {0};
    snprintf(raw_data, sizeof(raw_data), "test up raw data %lld", HAL_UptimeMs());
    return linkkit_invoke_raw_service(sample->thing, 1, raw_data, strlen(raw_data));
}
#endif
static int is_active(sample_context_t* sample)
{
    return sample->cloud_connected && sample->thing_enabled;
}
int main(int argc, char* argv[])
```

```
{
    sample_context_t* sample_ctx = &g_sample_context;
    int execution_time = 0;
    int get_tsl_from_cloud = 0;
    int exit = 0;
    int ret;
    unsigned long long now = 0;
    unsigned long long prev_sec = 0;
    int opt;
    while ((opt = getopt(argc, argv, "t:g:h")) != -1) {
        switch (opt) {
        case 't':
            execution_time = atoi(optarg);
            break;
        case 'g':
            get_tsl_from_cloud = atoi(optarg);
            break;
        case 'h':
            LINKKIT_PRINTF("-t to specify sample execution time period(minutes); -g to spec
ify if get tsl from cloud(0: not, !0: yes).\n");
            return 0;
            break;
        default:
            break;
        }
    }
    execution_time = execution_time < 1 ? 1 : execution_time;
    LINKKIT_PRINTF("sample execution time: %d minutes\n", execution_time);
    LINKKIT_PRINTF("%s tsl from cloud\n", get_tsl_from_cloud == 0 ? "Not get" : "get");
    memset(sample_ctx, 0, sizeof(sample_context_t));
    sample_ctx->thing_enabled = 1;
    linkkit_start(18, get_tsl_from_cloud, linkkit_loglevel_debug, &alinkops, linkkit_cloud_
domain_sh, sample_ctx);
    if (!get_tsl_from_cloud) {
        linkkit_set_tsl(TSL_STRING, strlen(TSL_STRING));
    }
#ifdef SERVICE_OTA_ENABLED
    linkkit_ota_init(fota_callback);
#endif /* SERVICE_OTA_ENABLED */
    while (1) {
        linkkit_dispatch();
        now = uptime_sec();
        if (prev_sec == now) {
#ifdef CMP_SUPPORT_MULTI_THREAD
            HAL_SleepMs(100);
#else
            linkkit_yield(100);
#endif
            continue;
        }
#if 0
        /* about 30 seconds, assume trigger post property event about every 30s. */
        if (now % 30 == 0 && is_active(sample_ctx)) {
            post_all_prop(sample_ctx);
```

```
        }
        /* about 31 seconds, assume invoke raw up service about every 31s. */
        if (now % 31 == 0 && is_active(sample_ctx)) {
            upload_raw_data(sample_ctx);
        }
        /* about 60 seconds, assume trigger event about every 60s. */
        if (now % 60 == 0 && is_active(sample_ctx)) {
            post_event_fault_alert(sample_ctx);
        }
#endif
        /* about 60 seconds, assume trigger event about every 60s. */
        if (now % 60 == 0 && is_active(sample_ctx)) {
            post_event_error(sample_ctx);
        }
        if (now % 5 == 0 && is_active(sample_ctx)) {
            ret = linkkit_trigger_deviceinfo_operate(sample_ctx->thing, "[{\"attrKey\":\"Te
mperature\",\"attrValue\":\"36.8\"}]", linkkit_deviceinfo_operate_update);
        }
        if (exit) break;
        /* after all, this is an sample, give a chance to return... */
        /* modify this value for this sample executaion time period */
        if (now > 60 * execution_time) exit = 1;
        prev_sec = now;
    }
    linkkit_end();
    return 0;
}
```

8.三方语音平台

8.1. 公版App使用天猫精灵控制设备

使用公版App的产品，只需定义了生活物联网平台的标准功能属性，并且在天猫精灵支持的品类中，即可快速连接天猫精灵，实现天猫精灵音箱对设备的控制。

背景信息

本示例中使用天猫精灵音箱，详细的官方文档请参见天猫精灵官方推荐品类和天猫精灵设备案例。

限制条件

仅支持在中国境内激活的设备对接天猫精灵。

控制台开通

1. 确认已打开天猫精灵的控制开关。
   i. 选择一个现有的项目，或创建一个新项目。参见创建项目。
   ii. 创建一个新产品，参见创建产品。
   iii. 定义产品功能，参见新增标准功能。
   iv. 添加测试设备，参见新增测试设备。
   v. 配置App功能参数项。详情请参见配置App功能参数。

   生活物联网平台默认打开天猫精灵的对接开关。

2. 单击品类图标，查看功能映射。

   生活物联网平台已经默认完成了部分天猫精灵品类功能的映射。 这些已经完成映射的指令，您无需再自行开发，只需确保设备使用了映射过的标准属性，即可被天猫精灵控制。

3. 调试设备。

   使用公版App配网绑定了您的测试设备后，可以根据下方消费者使用步骤，绑定天猫精灵，即可实现音

箱的控制。 下载公版App请参见云智能App介绍。

> **注意** 首次注册账号时，国家和地区请选择中国内地。

消费者使用

阿里IoT的公版免开发App支持天猫精灵控制，使用方式如下步骤所示。

1. 用户拥有一台支持天猫精灵的设备，下载阿里云IoT提供的公版App**云智能-智能家居助手**，绑定该设备。
2. 用户拥有一台天猫精灵音箱，从手机端应用商店下载**天猫精灵**App并绑定该音箱。
3. 在公版App中，选择**我的 > 第三方平台接入（或更多服务） > 天猫精灵 > 绑定账号**，在弹出的界面中输入淘宝账号，操作步骤如下图所示。

> **说明** 公版App账号和淘宝账号是一一对应的。

4. 登录成功后，您可以通过天猫精灵音箱控制您的设备。

   在天猫精灵App，即可在**我家**页签的设备列表中看到您的设备。

8.2. 公版App使用Amazon Echo音箱控制设备

使用公版App的产品，可以一键开通Amazon Alexa，实现Amazon Echo音箱对设备的控制。

背景信息

本示例中使用Amazon Alexa，详细的官方文档请参见Amazon Alexa官方文档。

限制条件

仅支持在以下地区激活的设备对接Amazon Alexa：

- 海外
- 中国香港
- 中国澳门
- 中国台湾

控制台开通

1. 确保自己的产品在Amazon Alexa可控范围内。

生活物联网平台已支持Amazon Alexa品类	Amazon Alexa官方推荐的品类
○ 扫地机器人 ○ 灯 ○ 插座 ○ 香薰机 ○ 窗帘	○ 灯和开关： Lights, switches and bulbs ○ 门锁：Door locks ○ 智能摄像头：Smart home cameras ○ 恒温调温器、风扇：Thermostats and fans ○ 微波炉：Microwave ovens

> **说明** 平台正在逐渐增加Amazon Alexa支持的品类，如果您的产品不在以上列表中，请在对接前提交工单，与我们的技术支持确认您的产品是否已支持。

2. 购买并开通Amazon Alexa服务。
   i. 进入**服务中心**，单击Amazon Alexa对应的**开通服务**。
   ii. 分配一个项目，并在该项目中创建一个新产品。请参见创建产品。
   iii. 定义产品功能。请参见新增标准功能。
   iv. 在**人机交互**页面，打开使用公版App控制产品的开关，并单击Amazon Alexa对应的**设置**。
   打开Amazon Alexa的开关。

   人机交互页面其余内容详情请参见配置App。
3. 设备调试。
   i. 烧录测试设备的证书信息。请参见开发设备。
   ii. 下载公版App调试设备（请参见云智能App介绍）。配网绑定了您的测试设备后，可以根据下方消费者使用步骤，绑定Amazon Alexa，即可实现音箱的控制。

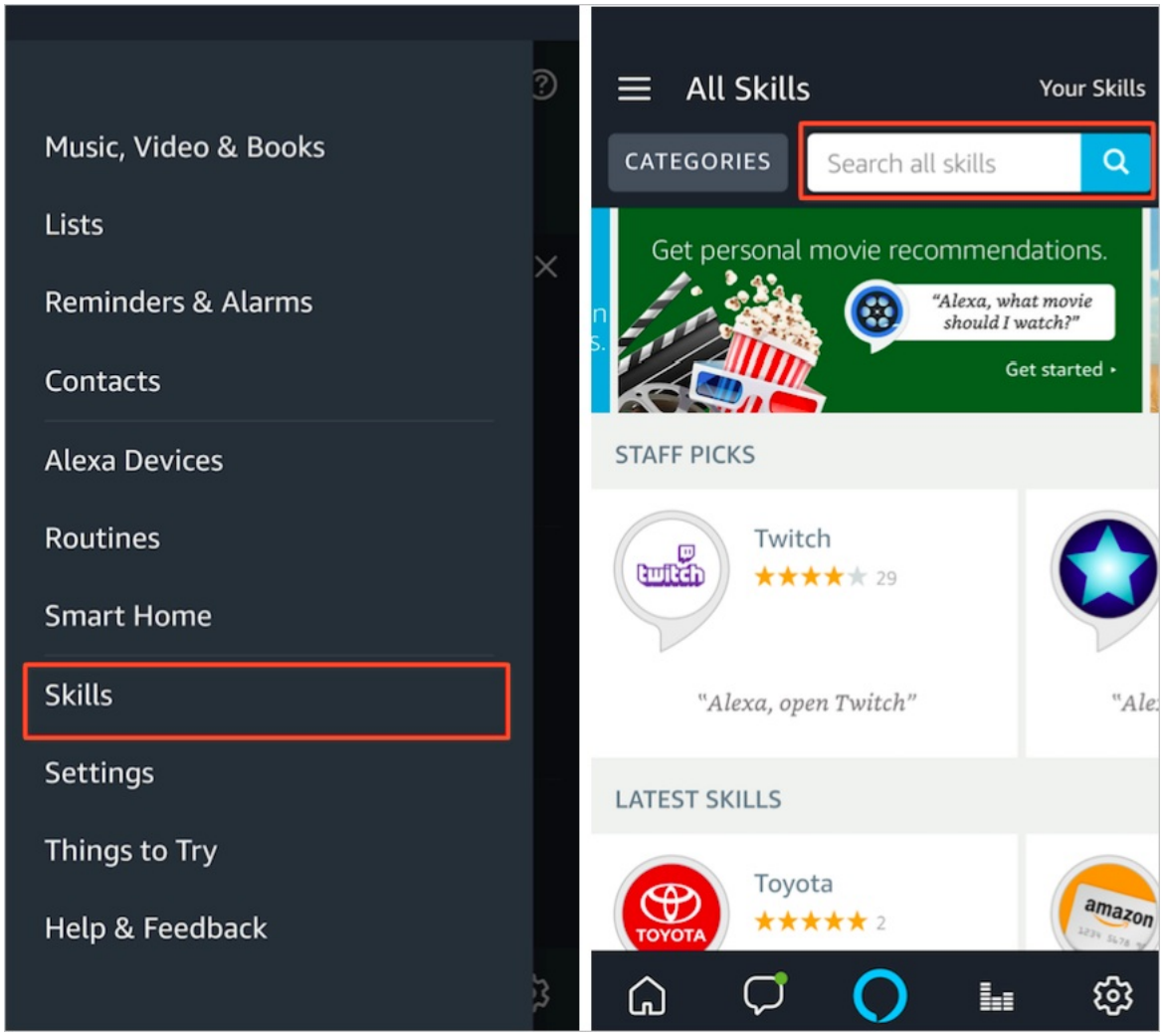
> **注意** 首次注册账号时，国家和地区请选择中国内地以外地区，例如选择美国。

消费者使用

1. 用户拥有一台接入生活物联网平台（国际站）的设备，下载阿里云IoT提供的公版App**云智能-智能家居助手**（中国内地手机端应用商店）或**Cloud Intelligence**（国外手机端应用商店），绑定该设备。
2. 将设备昵称改为英文字母，例如My Light Device。
3. 用户拥有一台Amazon Echo音箱，从手机端应用商店下载**Amazon Alexa** App并绑定该音箱。

> 说明
> - iOS系统用户请至美国区App Store搜索Amazon Alexa下载，Android用户请至Google Play搜索Amazon Alexa下载。
> - 在中国内地地区下载Amazon Alexa需要使用国外的应用商店账号，所以国内测试建议通过网页来配置。

4. 在Amazon Alexa客户端中单击**Skills**，搜索Cloud Intelligence，单击**ENABLE**开启该Skill。



5. 输入您Cloud Intelligence客户端的登录账号和密码，并正确选择您的账号所属国家，然后单击**Link Now**绑定您的智能家居App设备。
6. 控制设备前，Amazon Echo音箱需要先发现设备。您可以对Amazon Echo音箱说：`Alexa, discover devices`。您也可以在Cloud Intelligence页面单击**DISCOVER**来发现设备，绑定过的设备会显示在列表中。

> 说明 如果在公版App Cloud Intelligenc中修改了设备昵称，Amazon Echo音箱必须重新发现设备，否则不能对设备进行控制。

7. 您可以通过下列指令控制您的设备（假设设备昵称为My Light Device）。
   - 打开设备：`Alexa, turn on/off my light device.`

- 调节灯亮度：`Alexa, brighten/dim my light device.`
- 调节灯颜色：`Alexa, set my light device to yellow.`
- 调节灯色温：`Alexa, set my light device to warm white.`

8.3. 公版App使用Google Home音箱控制设备

使用公版App的产品，可以一键开通Google Assistant，实现Google Home音箱对设备的控制。

背景信息

本示例中使用Google Assistant，详细的官方文档请参见Google Assistant官方文档。

限制条件

仅支持在以下地区激活的设备对接Google Home：

- 海外
- 中国香港
- 中国澳门
- 中国台湾

控制台开通

1. 确保自己的产品在Google Assistant可控范围内。

生活物联网平台已支持Google Assistant品类	Google Assistant官方推荐的品类
扫地机器人、灯、插座、香薰机	灯、香薰机、插座、空调、空气净化器、咖啡机、洗碗机、烘干机、风扇、烧水壶、烤箱、冰箱、加湿器、开关、扫地机

> **说明** 平台正在逐渐增加Google Assistant支持的品类，如果您的产品不在以上列表中，请在对接前提交工单，与我们的技术支持确认您的产品是否已支持。

2. 购买并开通Google Assistant服务。
   i. 进入**服务中心**，单击Google Assistant对应的**开通服务**。
   ii. 分配一个项目，并在该项目中创建一个新产品。请参见创建产品。
   iii. 定义产品功能。请参见新增标准功能。

iv. 在**人机交互**页面，打开使用公版App控制产品的开关，并单击Google Assistant对应的**设置**。

打开Google Assistant的开关。

人机交互页面其余内容详情请参见配置App。

3. 设备调试。
   i. 烧录测试设备的证书信息。请参见开发设备。
   ii. 下载公版App调试设备（请参见云智能App介绍）。配网绑定了您的测试设备后，可以根据下方消费者使用步骤，绑定Google Assistant，即可实现设备的控制。

> 注意 首次注册账号时，国家和地区请选择中国内地以外地区，例如选择美国。

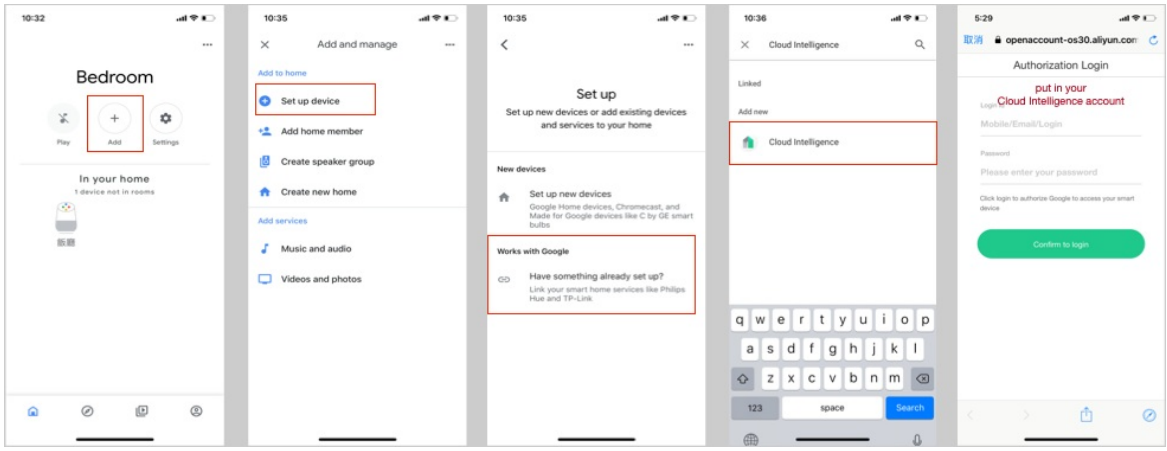
消费者使用

阿里云IoT的公版免开发APP支持Google Home和Google Assistant的控制，使用方式如下步骤所示。

1. 用户拥有一台接入生活物联网平台（国际站）的设备，下载阿里云IoT提供的公版App**云智能**（中国内地手机端应用商店）或**Cloud Intelligence**（国外手机端应用商店），绑定该设备。
2. 用户拥有一台Google Home音箱，从手机端应用商店下载**Google Home**或者 **Google Assistant** App并绑定该音箱。

> 说明 iOS系统用户请至美国区App Store搜索Google Home下载，Android用户请至Google Play搜索Google Home下载。

3. 在Google Home客户端中单击“+”，Set up devices中选择添加已有设备，搜索 Cloud Intelligence。输入您登录公版App的账号和密码，完成添加您的智能设备。



4. 您可以通过下列指令控制您的设备（如Light）。

- 打开或关闭灯：`Hey Google, turn on/off the light.`
- 调节灯强度：`Hey Google, brighten/dim the light.`
- 调节灯颜色：`Hey Google, set the light to yellow.`
- 调节灯色温：`Hey Google, set the light to warm white.`

8.4. 自有App接入天猫精灵教程

生活物联网平台提供了免费的天猫精灵快捷通道。您的产品只需定义了平台的标准功能属性，且在天猫精灵支持的品类中，在完成相应的配置后，即可快速连接天猫精灵，实现天猫精灵音箱对设备的控制。

限制条件

- 生活物联网平台遵循ICA数据标准，为了确保您的设备可以被天猫精灵准确识别和控制，请确保使用标准功能。
- 仅支持在中国境内激活的设备对接天猫精灵。

消费者使用

阿里IoT的公版免开发App支持天猫精灵控制，使用方式如下步骤所示。

1. 用户拥有一台支持天猫精灵的设备，下载厂家自有品牌App，绑定该设备。
2. 用户拥有一台天猫精灵音箱，从手机端应用商店下载**天猫精灵**App并绑定该音箱。
3. 在厂家自有品牌App中，用户授权淘宝账号登录，成功绑定天猫精灵。

   > **说明** App账号和淘宝账号是一一对应的。

4. 用户在天猫精灵App的设备列表中，在我家页签中可以看到绑定的设备。

5. 完成以上步骤后，您可以通过天猫精灵音箱控制您的设备。

控制台开通

1. 确认已打开天猫精灵的控制开关。
   i. 选择一个现有的项目，或创建一个新项目。参见创建项目。
   ii. 创建一个新产品，参见创建产品。
   iii. 定义产品功能，参见新增标准功能。
   iv. 添加测试设备，参见新增测试设备。

v. 配置App功能参数项。详情请参见配置App功能参数。

生活物联网平台默认打开天猫精灵的对接开关。

2. 单击品类图标，查看功能映射。

   生活物联网平台已经默认完成了部分天猫精灵品类功能的映射。 这些已经完成映射的指令，您无需再自行开发，只需确保设备使用了映射过的标准属性，即可被天猫精灵控制。

3. 调试设备。

   使用公版App配网绑定了您的测试设备后，可以根据下方消费者使用步骤，绑定天猫精灵，即可实现音箱的控制。 下载公版App请参见云智能App介绍。

   > 注意 首次注册账号时，国家和地区请选择中国内地。

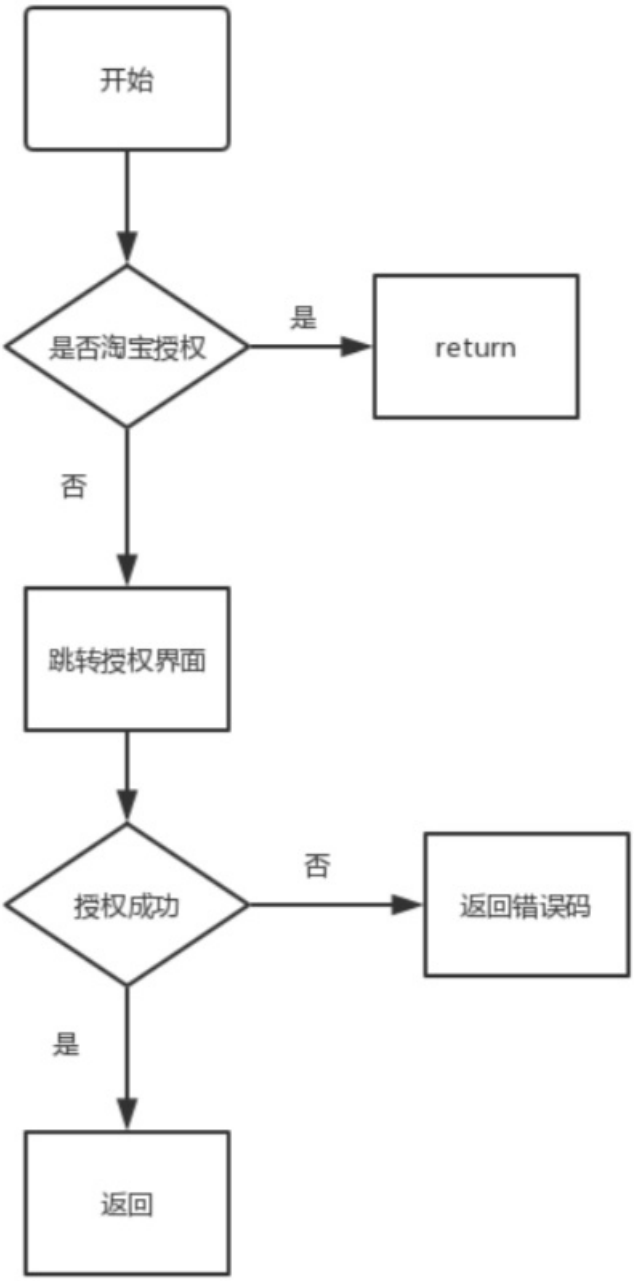
4. 创建一个自有App，参见创建自有App。

开发Android自有App

由于天猫精灵使用淘宝账号体系，需要在您的自有品牌App中，集成淘宝的账号授权，从而实现和天猫精灵的对接。

1. 调用淘宝登录授权页面的唤起接口。

   开发步骤如下图所示。



2. 单击授权按钮，进入授权界面。

   加载WebView，授权网址URL如下（需要拼接AppKey）。

   ```
   String url = "https://oauth.taobao.com/authorize?response_type=code&client_id=<项目的appKey>&redirect_uri=<控制台定义的回调地址>&view=wap";
   ```

3. 在shouldOverrideUrlLoading中判断地址是否包含code，获取code并传到上一步访问URL后的界面。

```
mWebView.setWebViewClient(new WebViewClient() {
            //设置结束加载函数
            @Override
            public void onPageFinished(WebView view, String url) {
                topbar.setTitle(view.getTitle());
            }
            @Override
            public boolean shouldOverrideUrlLoading(WebView view, String url) {
                if (isTokenUrl(url)) {
                    Intent intent = new Intent();
                    intent.putExtra("AuthCode", mAuthCode);
                    setResult(RESULT_CODE, intent);
                    finish();
                    return true;
                }
                view.loadUrl(url);
                return false;
            }
        });
  private boolean isTokenUrl(String url) {
        if (!TextUtils.isEmpty(url)) {
            if ( url.contains("code=")) {
                String[] urlArray = url.split("code=");
                if (urlArray.length > 1) {
                    String[] paramArray = urlArray[1].split("&");
                    if (paramArray.length > 1) {
                        mAuthCode = paramArray[0];
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

4. 根据code绑定淘宝账号。

   示例代码如下，其中 `/account/taobao/bind` 接口的说明，请参见账号服务。

```
public void bindAccount(String authCode) {
        JSONObject params = new JSONObject();
        if (null != authCode) {
            params.put("authCode", authCode);
        }
        Map<String, Object> requestMap = params.getInnerMap();
        IoTRequest ioTRequest = new IoTRequestBuilder()
                .setAuthType("iotAuth")
                .setApiVersion("1.0.5")
                .setPath("/account/taobao/bind")
                .setParams(requestMap)
                .setScheme(Scheme.HTTPS)
                .build();
        new IoTAPIClientFactory().getClient().send(ioTRequest, new IoTCallback() {
            @Override
            public void onFailure(IoTRequest ioTRequest, Exception e) {
            }
            @Override
            public void onResponse(IoTRequest ioTRequest, IoTResponse ioTResponse) {
            }
        });
    }
```

5. （可选）调用接口解除绑定。

```
// 用户解绑淘宝Id
IoTRequestBuilder builder = new IoTRequestBuilder();
builder.setPath("/account/thirdparty/unbind");
builder.setApiVersion("1.0.5");
builder.setAuthType("iotAuth");
builder.addParam("accountType","TAOBAO");
IoTRequest ioTRequest = builder.build();
new IoTAPIClientFactory().getClient().send(ioTRequest, new IoTCallback() {
    @Override
    public void onFailure(IoTRequest ioTRequest, final Exception e) {}
    @Override
    public void onResponse(IoTRequest ioTRequest, final IoTResponse ioTResponse) {}
});
///account/thirdparty/unbind接口的说明，请参见账号服务
//此处para = @{@"accountType":@"TAOBAO"}
```

6. 判断是否已绑定。

`/account/thirdparty/get` 接口的说明，请参见账号服务。

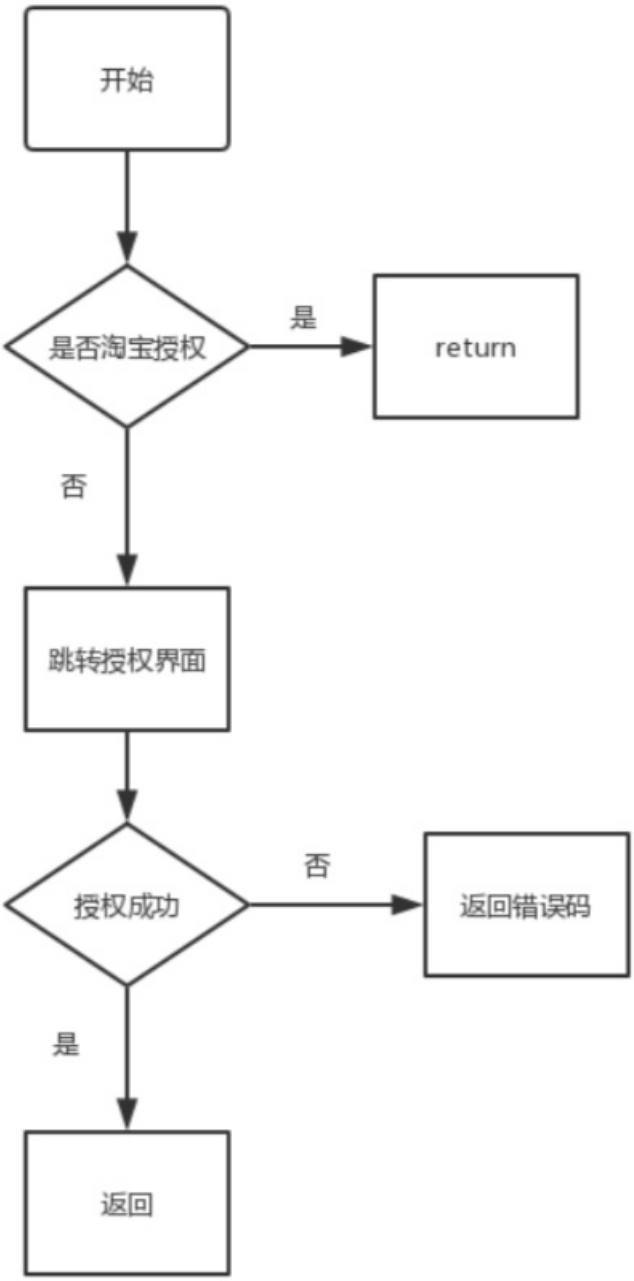
```
// 查询当前用户绑定淘宝Id
IoTRequestBuilder builder = new IoTRequestBuilder();
builder.setPath("/account/thirdparty/get");
builder.setApiVersion("1.0.5");
builder.setAuthType("iotAuth");
builder.addParam("accountType","TAOBAO");
IoTRequest ioTRequest = builder.build();
new IoTAPIClientFactory().getClient().send(ioTRequest, new IoTCallback() {
    @Override
    public void onFailure(IoTRequest ioTRequest, final Exception e) {}
    @Override
    public void onResponse(IoTRequest ioTRequest, final IoTResponse ioTResponse) {}
});
```

开发iOS自有App

由于天猫精灵使用淘宝账号体系，需要在您的自有品牌App中，集成淘宝的账号授权，从而实现和天猫精灵的对接。

1. 调用淘宝登录授权页面的唤起接口。

   开发步骤如下图所示。



2. 单击授权按钮，进入授权界面。

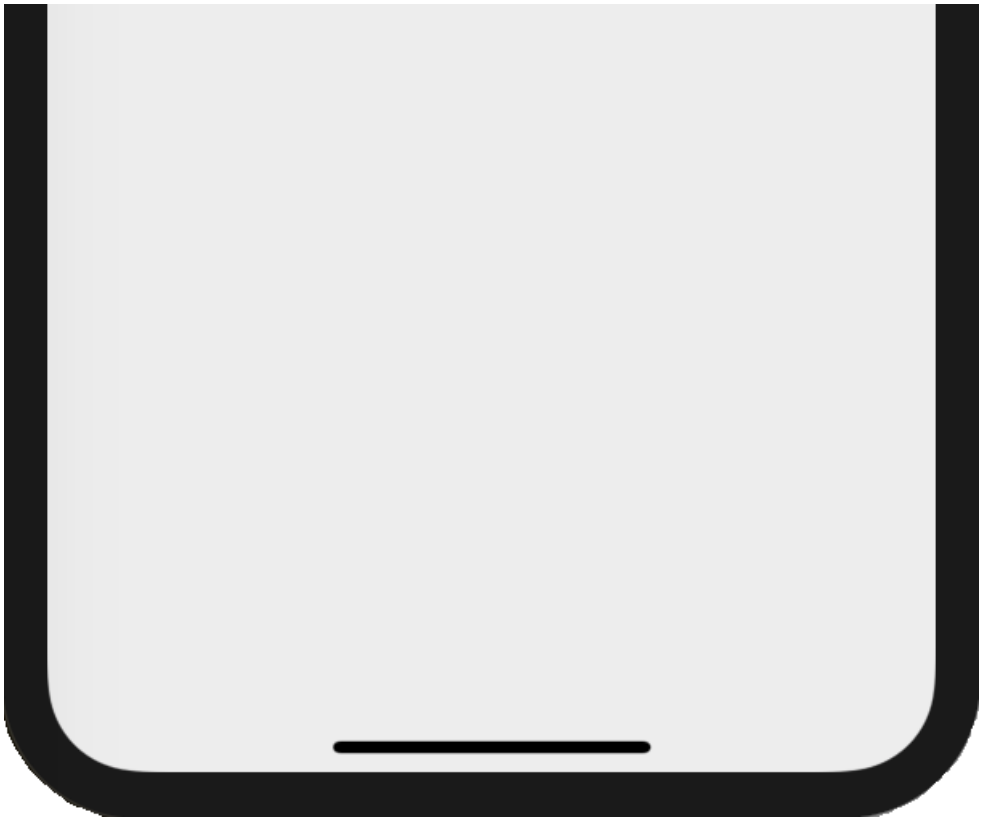
   加载WebView，授权网址URL如下（需要拼接AppKey）。

   ```
   https://oauth.taobao.com/authorize?response_type=code&client_id=<项目的appKey>&redirect_uri=<控制台定义的回调地址>&view=wap
   ```

   初始化一个WebView，并设置代理加载。示例代码如下。

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [super viewDidLoad];
  WKWebView *webView = [[WKWebView alloc]init];
  [self.view addSubview:webView];
  [webView mas_makeConstraints:^(MASConstraintMaker *make) {
      make.left.equalTo(self.view);
      make.right.equalTo(self.view);
      make.top.equalTo(self.view);
      make.bottom.equalTo(self.view);
    }];
  webView.navigationDelegate = self; //代理：WKNavigationDelegate
  [webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString:[NSString stri
ngWithFormat:@"https://oauth.taobao.com/authorize?response_type=code&client_id=%@&redir
ect_uri=控制台定义的回调地址&view=wap", 项目的appkey]]]];
}
```

3. 加载页面成功后，使用真实的淘宝账号登录。



4. 登录成功后，处理您在控制台填写的回调地址。


```
- (void)webView:(WKWebView *)webView decidePolicyForNavigationAction:(WKNavigationAction *)navigationAction decisionHandler:(void (^)(WKNavigationActionPolicy))decisionHandler {
    NSRange range = [navigationAction.request.URL.absoluteString rangeOfString:@"控制台定义的回调地址"];
    if (range.location){
        //允许跳转
        decisionHandler(WKNavigationActionPolicyAllow);
    } else {
        //不允许跳转
        decisionHandler(WKNavigationActionPolicyCancel);
        NSURLComponents *components = [NSURLComponents componentsWithString:navigationAction.request.URL.absoluteString];
        for (NSURLQueryItem *item in components.queryItems){
            if ([item.name isEqualToString:@"code"]){
                //用户绑定淘宝ID请求:此处IMSTmallSpeakerApi 通过下面代码封装一个基础请求类
                [IMSTmallSpeakerApi bindTaobaoIdWithParams:@{@"authCode":item.value} completion:^(NSError *err, NSDictionary *result) {
                    if (self.completion){
                        self.completion(err, result);
                    }
                    [self.navigationController popViewControllerAnimated:YES];
                }];
                break;
```

```
            }
        }
    }
}
// 封装的IMSTmallSpeakerApi请求类，依赖请求类#import <IMSApiClient/IMSApiClient.h>
// .h中
+ (void)requestTmallSpeakerApi:(NSString *)api
                       version:(NSString *)ver
                        params:(NSDictionary *)para
                    completion:(void (^)(NSError *, id))completion;
// .m中
+ (void)requestTmallSpeakerApi:(NSString *)api
                       version:(NSString *)ver
                        params:(NSDictionary *)para
                    completion:(void (^)(NSError *, id))completion {
    IMSIoTRequestBuilder *builder = [[IMSIoTRequestBuilder alloc] initWithPath:api
                                                                    apiVersion:ver
                                                                        params:para];
    [builder setScheme:@"https"];
    IMSRequest *request = [[builder setAuthenticationType:IMSAuthenticationTypeIoT] bui
ld];
    [IMSRequestClient asyncSendRequest:request responseHandler:^(NSError * _Nullable er
ror, IMSResponse * _Nullable response) {
        if (completion) {
            //返回请求过期后，需要重新登录；重新登录后重新初始化主框架，不需要重新请求
            if (response.code == 401) {
                if (NSClassFromString(@"IMSAccountService") != nil) {
                    // 先退出登录
                    if ([[IMSAccountService sharedService] isLogin]) {
                        [[IMSAccountService sharedService] logout];
                    }
                    return;
                }
            }
            if (!error && response.code == 200) {
                completion(error, response.data);
                return ;
            }
            NSError *bError = [NSError errorWithDomain:NSURLErrorDomain
                                                  code:response.code
                                              userInfo:@{NSLocalizedDescriptionKey: res
ponse.localizedMsg ? : @"服务器应答错误"}];
            completion(bError, nil);
            return;
        }
    }];
}
```

```
// 用户绑定淘宝Id
+ (void)bindTaobaoIdWithParams:(NSDictionary *)para
                    completion:(void (^)(NSError *, NSDictionary *))completion{
    [self requestTmallSpeakerApi:@"/account/taobao/bind" version:@"1.0.5" params:para c
ompletion:completion];
///account/taobao/bind接口说明请参见账号服务
}
//此处para = @{@"authCode":@"xxxx"}，其中xxxx为网页回调的code，具体查看登录成功后的回调处理步
骤
```

处理回调后返回code200，表示授权成功。此时请刷新网页。

5. （可选）调用接口解除绑定。

```
// 用户解绑淘宝Id
+ (void)unbindTaobaoIdWithParams:(NSDictionary *)para
                      completion:(void (^)(NSError *, NSDictionary *))completion{
    [self requestTmallSpeakerApi:@"/account/thirdparty/unbind" version:@"1.0.5" params:
para completion:completion];
}
///account/thirdparty/unbind接口的说明，请参见账号服务
//此处para = @{@"accountType":@"TAOBAO"}
```

6. 判断是否已绑定。

`/account/thirdparty/get` 接口的说明，请参见账号服务。

```
// 查询用户绑定的淘宝Id
+ (void)getTaobaoIdWithParams:(NSDictionary *)para
                   completion:(void (^)(NSError *, NSDictionary *))completion{
    [self requestTmallSpeakerApi:@"/account/thirdparty/get" version:@"1.0.5" params:par
a completion:completion];
}
//此处para = @{@"accountType":@"TAOBAO"}
```

返回数据存在以下内容代表已绑定。

```
{
    accountId = 90xxx335;
    accountType = TAOBAO;
    linkIndentityId = 50ebop9xxxxxxxxxxxxxx8dbc58643423092968;
}
```

8.5. 自有App定制Amazon Alexa技能

生活物联网平台支持您的自有App接入Amazon Alexa，并定制上架专属于您自己独特品牌的技能，提升品牌价值和竞争力。

Amazon Alexa技能可以让带有您品牌标识的技能在Amazon Alexa技能列表中展示，终端用户搜索后即可使用。助您树立品牌形象，提升品牌竞争力。

您可以在生活物联网平台控制台的**人机交互**页面Amazon Alexa参数项中，查看您的产品是否在Amazon Alexa支持的品类中，以及哪些功能支持Amazon Alexa的语音控制。

您可以自行接入Amazon Alexa，也可以联系我们定制。定制Amazon Alexa技能是一项收费的增值服务（详细请参见服务计费）。服务提供接入Amazon Alexa的一系列技术协助，包括开发、测试与技能上架等（技能上架的审核结果与所需时间取决于Amazon平台）。

一、准备工作

- 已创建了产品，并完成产品的开发。详细请参见概述。
- 已创建了自有App，且已集成了App SDK，完成了对产品的配网和控制。详细请参见创建自有App和App端开发文档。
- 已在iOS App Store或Google Play上架您的自有App。
- 已完成Amazon Alexa和AWS的账号注册。
- 已完成亚马逊开发者账号的注册。
- 您已开通Amazon Alexa接入功能，并分配到产品所在的项目。
- 在产品的**人机交互**页面中，已打开产品的Amazon Alexa开关。

二、申请Amazon Alexa能力定制

完成准备工作后，您可以通过控制台右上角的工单或您专属的客户代表联系我们，填写Amazon Alexa技能定制申请表。

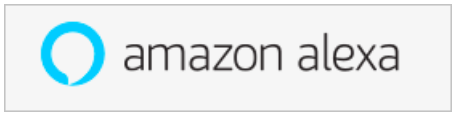
我们收到申请表后会有技术人员跟您联系，协助您完成Amazon Alexa能力定制。

三、体验效果

请您根据以下步骤，使用Amazon Echo音箱控制您自有App绑定的设备，从而体验Amazon Echo定制技能。

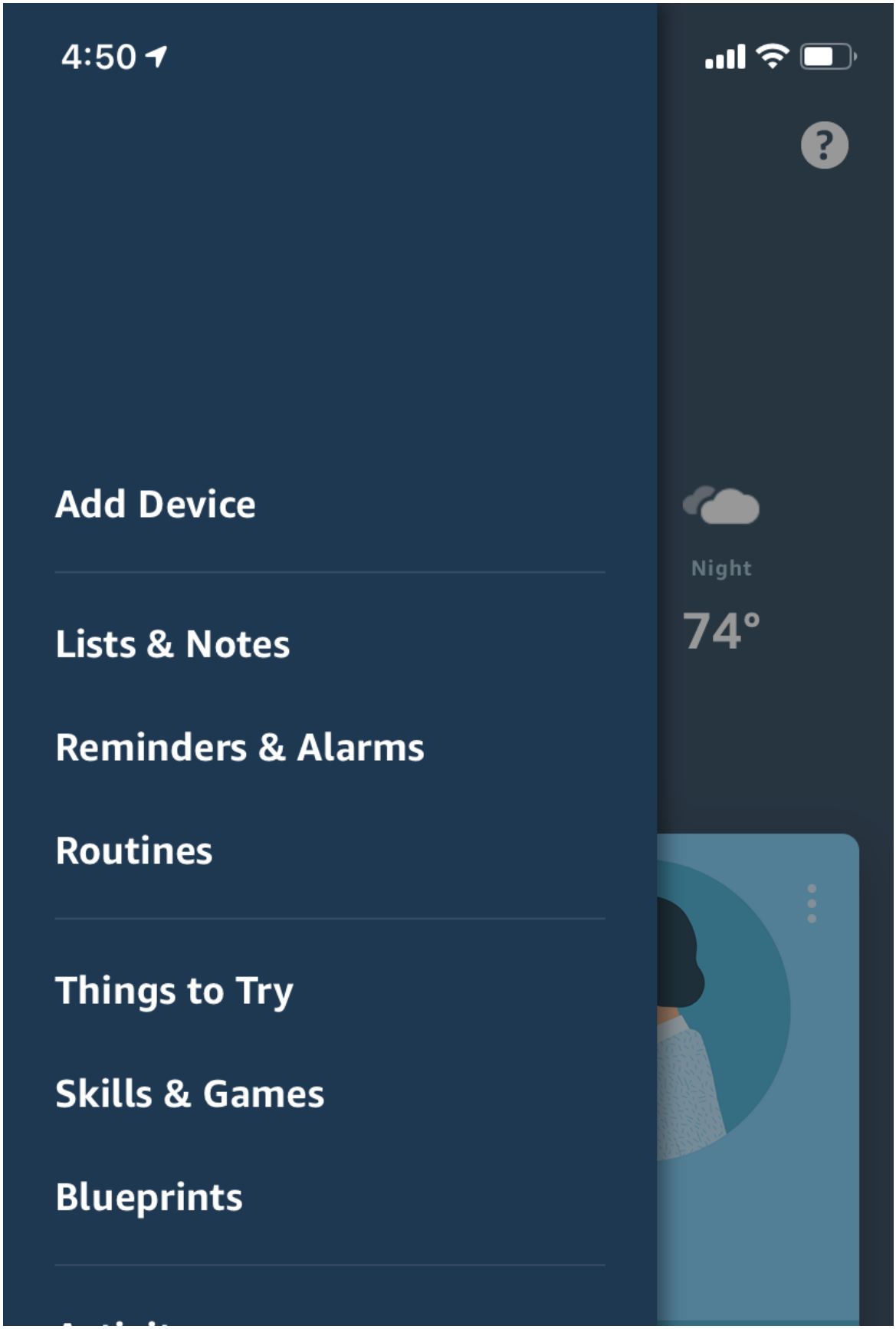
1. 准备一台Amazon Echo音箱，下载Amazon alexa客户端并安装绑定您的Echo音箱。

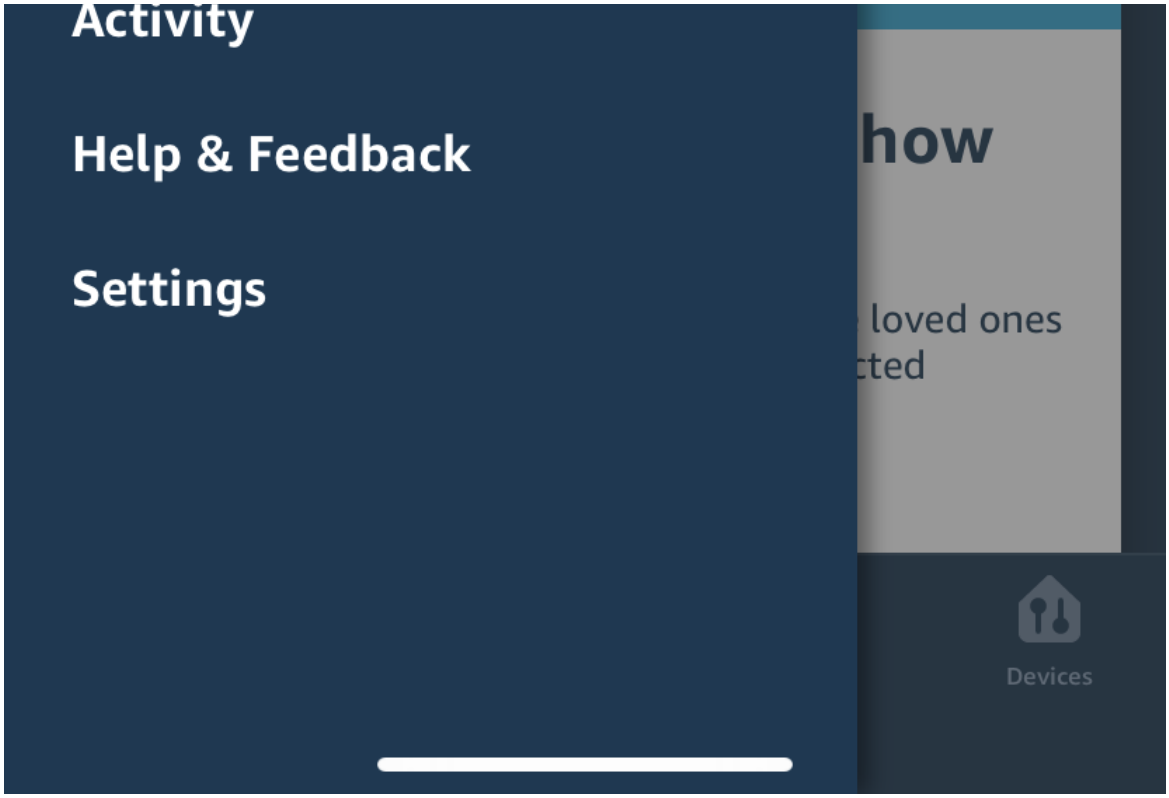
   iOS用户请至美国区App Store搜索“Amazon Alexa”下载，Android客户请至Google Play搜索“Amazon Alexa”下载。

   在大陆地区下载Alexa App需要使用国外的应用市场账号，所以国内测试建议通过网页来配置。网页配置地址为：http://alexa.amazon.com/spa/index.html

2. 准备一台设备，使用您的自有品牌App进行配网绑定。
3. 在Amazon alexa客户端中单击**Skills**，找到自己创建的Skill，单击**ENABLE**启用。





4. 在弹出的界面中输入您自有App登录的账号和密码，并正确选择您的账号所属国家，然后单击**Link Now**来绑定您自有App绑定的设备。
5. 发现设备。

   控制设备前，Echo需要先发现设备。

   - 您可以对Echo说：“Alexa, discover devices”。
   - 您也可以在Skill页面单击**DISCOVER**来发现设备。您绑定过的设备就会出现在列表里。

   > **说明** 如果您在自有App中修改了设备昵称，必须重新发现设备，否则不能对设备进行控制。

6. 通过下列指令控制您的设备（以控制灯为例）。
   - 打开/关闭灯：`Alexa, turn on/off the light.`
   - 调节灯亮度：`Alexa, brighten/dim the light.`
   - 调节灯颜色：`Alexa, set the light to yellow.`
   - 调节灯色温：`Alexa, set the light to warm white.`

8.6. 自有App定制Google Assistant技能

生活物联网平台支持您的自有App接入Google Assistant，并定制上架专属于您自己独特品牌的Action，提升品牌价值和竞争力。

Google Assistant技能可以让带有您品牌标识的技能在Google Assistant的Action列表中展示，终端用户搜索后即可使用。助您树立品牌形象，提升品牌竞争力。

您可以在生活物联网平台控制台的**人机交互**页面Google Assistant参数项中，查看您的产品是否在Google Assistant支持的品类中，以及哪些功能支持Google Assistant的语音控制。

您可以自行接入Google Assistant，也可以联系我们定制。定制Google Assistant技能是一项收费的增值服务（详细请参见服务计费）。服务提供接入Google Assistant Action的一系列技术协助，包括开发、测试、技能上架等（技能上架的审核结果与所需时间取决于Google Assistant平台）。

一、准备工作

- 已创建了产品，并完成产品的开发。详细请参见概述。
- 已创建了自有App，且已集成了App SDK，完成了对产品的配网和控制。详细请参见创建自有App和App端开发文档。
- 已在iOS App Store或Google Play上架您的自有App。
- 已完成Google开发者账号的注册。
- 您已开通Google Assistant，并分配至产品所在的项目。
- 在产品的**人机交互**页面中，已打开产品的Google Assistant开关。

二、申请Google Assistant能力定制

完成准备工作后，您可以通过控制台右上角的工单或您专属的客户代表联系我们，填写Google Assistant技能定制申请表。

我们收到申请表后会有技术人员跟您联系，协助您完成Google Assistant能力定制。

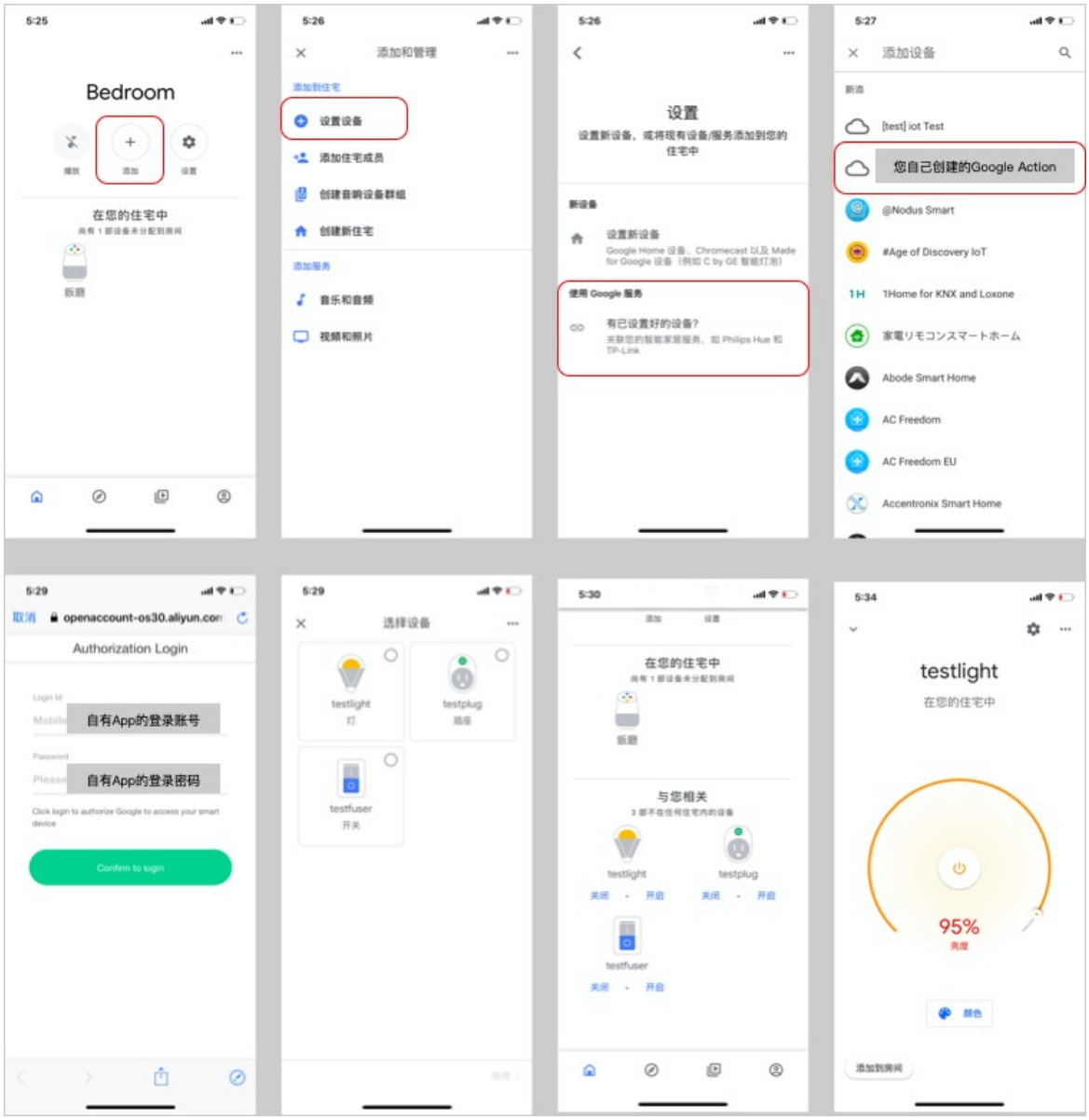
三、体验效果

请您根据以下步骤，使用Google Home音箱控制您自有App绑定的设备，从而体验Google Assistant定制技能。

1. 准备一台Google Home音箱，下载**Google Home**或者**Google Assistant** App，安装并绑定到您的音箱。

> **说明** iOS系统用户请至美国区App Store搜索Google Home下载，Android用户请至Google Play搜索Google Home下载。

2. 准备一台设备，使用您的自有品牌App进行配网绑定。
3. 在Google Home客户端中添加设备，找到您创建的Google Action。输入您自有品牌App登录的账号和密码，完成添加您的智能设备。



4. 您可以通过下列指令控制您的设备（以操控灯为例）。
   - 打开/关闭灯：`Hey Google, turn on/off the light.`
   - 调节灯亮度增强/减弱：`Hey Google, brighten/dim the light.`

- 调节灯颜色：`Hey Google, set the light to yellow.`
- 调节灯色温：`Hey Google, set the light to warm white.`