阿里云
生活物联网平台

开发指南

文档版本：20200428

为了无法计算的价值 | 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. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，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
	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。	**禁止：** 重置操作将丢失用户配置数据。
	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。
	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。
	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。	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

法律声明..... I
通用约定..... I
1 通用参考..... 1
1.1 错误码字典..... 1
1.2 加签算法..... 14
2 设备端开发指南..... 17
2.1 SDK概述与开发环境设置..... 17
2.2 国际站设备开发..... 19
2.3 配置SDK新增功能..... 23
2.4 设备证书分发工具使用指导..... 31
2.5 Wi-Fi设备开发..... 50
2.5.1 Wi-Fi芯片移植..... 50
2.5.2 Wi-Fi模组移植..... 63
2.5.3 Wi-Fi模组认证指导..... 69
2.5.4 基于已认证的模组开发..... 72
2.5.5 基于非认证的模组开发..... 88
2.6 网关及子设备开发..... 91
2.7 以太网设备端开发..... 99
2.8 蜂窝网设备端开发..... 101
2.9 蓝牙设备开发..... 103
2.9.1 蓝牙设备端开发..... 103
2.9.2 蓝牙设备端SDK用户编程接口..... 110
2.9.3 蓝牙设备端SDK移植接口..... 113
2.9.4 蓝牙设备端SDK OTA接口..... 121
2.10 LinkVisual视频设备开发..... 124
2.10.1 LinkVisual视频设备开发概览..... 124
2.10.2 LinkVisual设备端开发-Android SDK..... 126
2.10.3 Link Visual设备端开发-Linux SDK..... 144
2.10.4 Link Visual设备端开发-Linux SDK API..... 148
2.11 配网功能开发..... 166
2.11.1 Wi-Fi设备配网方案介绍..... 166
2.11.2 Wi-Fi设备配网适配开发..... 171
2.11.3 蓝牙辅助配网开发..... 186
3 App端开发指南..... 189
3.1 用户账号开发指南..... 189
3.2 插件使用指南..... 195
3.3 配网开发指南..... 217
3.4 移动应用推送开发指南..... 223
3.5 场景自动化开发指南..... 229
3.6 蓝牙连接开发指南..... 232

3.7 家空间开发指南.....241
3.8 客户端调用API.....243
3.8.1 如何调用API.....243
3.8.2 基础服务.....244
3.8.3 账号服务.....245
3.8.4 长连接服务.....258
3.8.5 产品管理服务.....262
3.8.6 物的模型服务.....273
3.8.7 用户服务.....298
3.8.8 消息服务.....343
3.8.9 家空间服务.....361
3.8.10 配网服务.....418
3.8.11 场景服务2.0.....438
3.8.12 数据聚合查询服务.....488
3.8.13 固件升级服务.....492
3.8.14 意见反馈.....503
3.8.15 锁的服务.....512
3.8.16 Link Visual视频服务.....534
3.9 Android SDK 手册.....567
3.9.1 SDK升级.....567
3.9.2 SDK初始化.....568
3.9.3 通用SDK.....570
3.9.4 账号及用户SDK.....572
3.9.5 身份认证SDK.....575
3.9.6 API通道SDK.....578
3.9.7 BoneMobile容器SDK.....581
3.9.8 长连接通道SDK.....586
3.9.9 配网SDK.....589
3.9.10 移动应用推送SDK.....593
3.9.11 设备模型SDK.....596
3.9.12 蓝牙OTA SDK.....603
3.9.13 Link Visual视频Media SDK.....605
3.9.14 常见问题.....645
3.10 iOS SDK手册.....651
3.10.1 SDK升级.....651
3.10.2 SDK初始化.....652
3.10.3 通用SDK.....654
3.10.4 账号及用户SDK.....656
3.10.5 身份认证SDK.....661
3.10.6 API通道SDK.....664
3.10.7 长连接通道SDK.....665
3.10.8 BoneMobile容器SDK.....669
3.10.9 日志SDK.....671
3.10.10 配网SDK.....672
3.10.11 移动应用推送SDK.....678

3.10.12 设备模型SDK.....679
3.10.13 蓝牙OTA SDK.....684
3.10.14 Link Visual视频Media SDK.....686
3.10.15 常见问题.....698
4 App模板开发指南..... 703
4.1 概述..... 703
4.2 App模板的通用修改..... 704
4.2.1 Android App通用修改项..... 704
4.2.2 iOS App通用修改项.....707
4.3 云智能App2.X系列模板..... 710
4.3.1 概述.....710
4.3.2 iOS 2.X系列模板.....714
4.3.3 Android 2.X系列模板..... 732
4.4 Demo App模板.....750
4.4.1 iOS Demo App模板.....750
4.4.2 Android Demo App模板.....773
5 云端开发指南..... 787
5.1 数据HTTP/2方式推送.....787
5.2 云端调用API..... 795
5.2.1 准备工作..... 795
5.2.2 云端资源服务.....799
5.2.3 产品管理服务..... 802
5.2.4 物的模型服务.....813
5.2.5 用户服务.....832
5.2.6 LinkVisual视频云存储购买服务.....846

1 通用参考

1.1 错误码字典

本文档介绍在生活物联网平台开发过程中遇到的错误码，以及对应的错误信息描述和可尝试解决方法。

常见错误码

错误码	错误信息	描述	可尝试解决方法
200	success	成功	不涉及
400	request error	请求错误	请联系技术支持，或者根据错误信息和日志信息自行定位解决
401	request auth error	请求认证错误	• 尝试检查是否已经使用appKey，appSecret登录 • 检查appKey，appSecret是否配置错误 • 针对配网绑定和设备控制接口，请检查相关设备是否与当前用户绑定
403	request forbidden	请求被禁止	• 检查appKey，appSecret是否配置错误 • 当前使用的appKey是否有接口访问权限
404	service not found	服务未找到	• 检查服务请求Method是否正确 • 检查服务请求路径是否正确
429	too many requests	请求频繁	当前时间段内请求过于频繁，可者降低调用频次或稍后重试
460	request parameter error	请求参数错误	业务参数异常，请检查上传的业务参数是否正确
	batch device amount invalid	非法的量产设备数	检查量产设备数量
	invalid count or offset	无效的count或offset，count须大于0， offset大于等于0	检查count（分页查询数量）和offset（查询偏移量）的值是否正确
500	service error	服务端错误	服务端发生未知异常，联系服务人员解决

错误码	错误信息	描述	可尝试解决方法
503	service not available	服务不可用	服务端发生未知异常，联系服务人员解决
2000	param is invalid	请求参数错误	• 请检查各入参信息是否存在错误 • 检查ProductKey是否填写错误
2062	identityId does not exist	identityId不存在	• 请检查当前是否正确进行账户登录和认证 • iotToken信息是否过期或传输错误
2063	iotId not exist	设备不存在	检查传入的**iotId**是否正确
2065	identityId and iotId not bound	设备和账号未绑定	当前登录的账号和设备之间没有绑定关系，请确定当前的账户与设备确实存在绑定关系
2066	identityId is not manager	该用户不是设备的管理员	检查该用户与设备的分享关系
2073	this product not support share	该设备的分享模式不支持生成二维码，例如：抢占式设备不支持分享	调用前尝试检查设备的分享模式
2074	query msg-center hsf error	调用消息中心错误	服务端发生未知异常，需要联系服务人员
2075	share model not config for this product	分享模式未配置	产品没有设置分享模式，前往产品开发中心配置分享模式
2077	can't get identityId by email and mobile	无法根据手机号或邮箱找到用户identityId	没有该用户，尝试重新输入
2078	not the same initiator when cancel share dev or scene	不是同一个发起者	非该用户发起的分享无法取消该分享，调用前请检查是否为自己发起的分享
2082	receiver has reject share	接收者已拒绝分享	该分享已被操作，无法取消已被操作的分享
2085	receiver has agree share	接收者已同意分享	该分享已被操作，无法取消已被操作的分享
2086	can not share to oneself	不能分享给自己	检查是否将设备的分享人填写为自己

错误码	错误信息	描述	可尝试解决方法
2087	iotId has bind to target user	设备和目标用户已绑定	重复操作，操作之前检查是否已经将设备与用户绑定
2088	forbidden to repeat launch share	不允许重复分享	检查是否已配置分享
2093	forbidden to repeat cancel sharing	不允许重复取消	重复操作，操作之前检查分享状态
2096	sharing time expired	分享已过期	尝试重新分享
2097	sharing record not exist	分享不存在	检查是否存在该分享
20050	xxx required	必填参数缺失	检查接口的必填字段是否填写
28538	event id is invalid or not exist	无效的事件告警 id	检查传入的 eventId 是否正确
5005	product not found	产品没有找到	检查ProductKey填写是否正确
5225	query product failed	查询产品失败	检查传入的ProductKey是否正确
600	SYSTEM_ERROR	系统错误	重试并切换配网方式
603	UNKNOWN_ERROR	未知错误	请联系技术支持人员
604	SDK_ERROR	SDK内部错误	重试并切换配网方式
605	USER_CANCEL	用户取消配网	请检查配网请求是否存在
606	SERVER_FAIL	服务端请求失败	根据日志查看错误类型
607	PROVISION_TIMEOUT	配网超时	稍后重试
60101	NETWORK_ERROR	当前连接的Wi-Fi是5G网络，不支持配网	提示用户切换到2.4G网络
60102	NETWORK_ERROR	Wi-Fi未连接	提示用户开启Wi-Fi连接
60103	NETWORK_ERROR	手机流量未开启或者无SIM卡	提示用户当前配网方式需要SIM卡，且需要开启上网功能

错误码	错误信息	描述	可尝试解决方法
60201	PARAMS_ERROR	productKey为空	检查linkType是否为空，或不支持，或需要的参数未填
60202	PARAMS_ERROR	linkType错误	检查linkType是否为空，或不支持，或需要的参数未填
60203	PARAMS_ERROR	deviceType错误	检查deviceType是否需要配网或者为非Wi-Fi设备
60204	PARAMS_ERROR	配网参数错误	检查配网参数，如配网版本号是否匹配
60205	PARAMS_ERROR	SSID为空	传递正确的SSID
6100	device not found	设备不存在	检查传入的iotId是否正确
6289	device is already active	当前提供的设备信息已在云端存在	检查当前传递的设备信息是否存在重复提交
6608	token not found	设备上报的token信息不存在	token为必填字段，检查是否已填写
9103	update data failed	升级失败	查询设备端上报日志定位错误
9112	there is no device exist	设备不存在	检查传入的iotId是否正确
9119	firmware not exist	固件不存在	检查传入的version版本号是否正确
9131	invalid device	非法的设备	请确认设备状态是否正确
9132	non-existent job record	升级任务记录不存在	确认设备是否进行过固件升级的操作
9143	upgrade info posted is invalid	升级进度上报错误	确认接口参数正确后请重试或联系技术支持人员
9144	device count exceeded	设备数量超限	请参考接口文档中对设备数量的限制说明
9146	upgrade record not exist	升级记录不存在	确认设备进行过该版本的固件升级
9153	device auth failed	设备校验失败	请确认设备状态是否正确
10196	device exist circle	设备存在循环依赖（要求：特定产品的特定属性在TCA中不能重复出现）	请联系技术支持，或者根据错误信息和日志信息自行定位解决

错误码	错误信息	描述	可尝试解决方法
10197	scene exist circle	场景存在循环依赖（要求：action中指定的场景不能形成环闭合）	重新检查场景逻辑内是否存在循环依赖
10198	scene depth illegal	场景嵌套深度非法（要求：场景最多嵌套一层场景）	场景在一层逻辑节点下最多嵌套一层判断节点，检查是否存在逻辑节点嵌套错误
10199	scene width illegal	action中场景个数非法（要求action中场景个数不能超过100）	检查action中场景个数是否超出限制
10200	scene empty	没有这个场景	确认场景是否存在，或场景创建后未检查是否创建成功
20004	gateway. backend service error	请检查当前的请求入参是否存在错误的入参，例如JSON入参传递了JSONString类型	请检查当前的请求入参是否存在错误的入参，例如JSON入参传递了JSONString类型
20017	非法cloudToken或超时	CloudToken非法或请求超时	通过云端资源接口重新获取token，并且在有效期内刷新token
24063	query device failed	查询设备信息失败	检查提供的设备ID信息是否存在错误，或提供了一个非法的设备ID信息
28008	product not exist	产品不存在	检查产品是否存在
28036	no activation code quota	激活码余量不足	购买激活码
28038	product not release	产品未发布	发布该产品
28044	no valid device name	没有合法的设备名	检查设备证书是否正确
28045	device name checking	设备名检查中	检查设备证书是否正确或稍后重试
28046	batch has created	批次已经创建	请检查批次是否已存在
28081	aliyun account not register phone number	阿里云账号未注册手机号码	补充阿里云账号对应手机号码
28082	sms send fail	信息发送失败	补充阿里云账号对应手机号码

错误码	错误信息	描述	可尝试解决方法
28083	The format of the product name is incorrect	产品名称格式不正确	请按照格式要求修改产品名称
28501	project is not exist	项目不存在	操作前检查项目是否已经存在
28502	alarm record is not exist	告警记录不存在	操作前检查告警记录是否已经存在
28504	device is offline	设备不在线	操作前检查设备是否已经上线
28505	product not exist	产品不存在	操作前检查产品是否存在
28517	invalid iot id	无效的iotId	确认传入的iotId是否存在
28532	invalid user info	无效的用户身份信息	确认传入的用户身份ID是否存在
28527	user not bind with lock	钥匙还未和虚拟用户绑定	没有云端创建该虚拟锁用户，检查是否在云端创建了该锁的用户
28528	operation type not support	不支持当前传入的虚拟用户信息更新操作类型	指定的操作类型错误，重新检查代码中相关枚举是否正确。 • 1：添加 • 2：更新 • 3：删除
28529	virtual user attr list is empty	虚拟用户的属性列表为空	传入的用户属性及属性值为空，请检查传入的相关参数值
28530	lock user has been bound to other virtual user	钥匙已经和其他虚拟用户绑定	为新用户重新创建新的钥匙
28551	user no register or router	用户没有绑定已配网设备或路由器	请确认用户是否已绑定成功
28560	device not exist	设备不存在	检查传入的iotid
28563	ble device bind sign error	蓝牙设备验签不通过	• 检查签名的设备证书信息与产品信息是否设置正确 • 检查签名设置是否符合规范
28566	product not exist	产品不存在	检查是否传入了合法存在的productKey
28567	device must be gprs device	必须是GPRS设备	操作前检查设备网络类型

错误码	错误信息	描述	可尝试解决方法
28571	device types that are not supported	不支持的设备类型	检查设备类型是否正确
28573	No permission to bind the device	当前APP无法添加该设备，请联系售后支持	检查该设备对应的产品是否为App的关联产品
28601	user device not bind	用户设备没有绑定	操作前确认用户与设备的绑定关系
28603	Remove all devices in this group before deleting it	请移除分组内的设备后再删除分组	操作前确认分组内已无设备
28604	The group does not exist	该分组不存在	操作前确认分组是否存在
28605	"You are not authorized to delete \"all devices\" or the devices in this group.	“全部设备”及该分组下的设备不可删除	不要删除“全部设备”及该分组下的设备
29003	identityId is null	identityId为空	检查用户是否登录了其他App
29004	device is unbind	设备已解绑	请确认设备是否已解绑

配网错误码

错误码	错误描述	公版App上的提示文案	可尝试解决方法
61701	配网超时，未找到目标设备热点	未发现设备热点，请按引导初始化设备	设备热点配网，检查设备是否进入配网模式
61702	配网超时，未找到目标蓝牙Wi-Fi双模设备	未搜索到设备蓝牙信号，请按引导初始化设备	蓝牙辅助配网，检查设备是否进入配网模式
61703	配网超时，蓝牙设备连接超时	设备蓝牙连接超时，导致添加失败，请重试	重试或切换配网方式
61704	配网超时，获取双模设备信息超时	获取设备蓝牙信息超时，导致添加失败，请重试	重试或切换配网方式
61705	配网超时，无目标设备连接手机热点	未发现设备连接到手机热点，请按引导初始化设备	手机热点模式，检查设备是否进入配网模式

错误码	错误描述	公版App上的提示文案	可尝试解决方法
101600	系统错误	添加失败，请重试或切换另一种配网方式	重试或者切换其他配网方案
101603	未知错误	添加失败，请重试或切换另一种配网方式	重试或者切换其他配网方案
101604	内部错误等其他错误	添加失败，请重试或切换另一种配网方式	重试或者切换其他配网方案
101609	本地获取绑定Token超时	添加失败，请重试或切换另一种配网方式	绑定失败，重试或切换配网方式
101601+60101	当前连接的Wi-Fi是5G网络，不支持配网	不支持当前网络，请切换到2.4G网络	提示用户切换到2.4G网络
101601+60102	Wi-Fi未连接或过程中Wi-Fi断开	获取设备蓝牙信息超时，导致添加失败，请重试	提示用户开启Wi-Fi连接或检查Wi-Fi
101601+60104	手机热点配网，AP热点未打开	手机Wi-Fi连接不稳定，请确保手机Wi-Fi已连接，然后重试	提示用户手动开启aha热点之后再重新开始配网
101616+61601	配网失败，蓝牙连接异常断开或连接失败	家庭网络情况不佳，设备连接路由器失败，请重试	重试配网
101616+61605	配网失败，获取蓝牙Wi-Fi双模设备信息失败	获取设备蓝牙信息失败，请重试。	重试配网
999103	访问云端接口错误异常，超时	手机网络情况不佳，导致添加失败，请重试	检查手机网络连接

家空间错误码

错误码	错误信息	描述	可尝试解决方法
28804	Duplicate name.	家空间名称重复	修改家空间的名称
28805	Invalid name.	家空间名称非法，仅支持字母、数字、空格和汉字	按照要求修家空间的名称
28806	This feature is not supported by the current app version. Please upgrade to the latest version.	当前App版本不支持此功能，请升级到最新版本	请升级App版本
28807	Up to 100 devices can be added.	最多添加100个设备	请确认已添加设备的数量

错误码	错误信息	描述	可尝试解决方法
28808	The maximum number of room that you can create is exceeded.	最多新建20个房间	请确认已创建房间的数量
28809	The maximum number of home that you can create is exceeded.	最多新建10个家庭	请确认已创建家庭的数量
28810	Under the low limit of the number of home.	无法删除，至少保留一个家庭	请确认当前是否仅有一个家庭
28811	One device only can be added to one group.	一个设备只能添加到一个设备组	请确认是否存在设备组
28812	Only letters, numbers and Chinese characters are supported. Must be less than 20 characters.	仅支持字母、数字、空格和汉字，长度最大支持20个字符	请按照规定修改
28813	Only letters, numbers and Chinese characters are supported. Must be less than 40 characters.	仅支持字母、数字、空格和汉字，长度最大支持40个字符	请按照规定修改
28820	Add at least 2 devices.	至少添加2个设备	请确认当前设备的数量
28821	Up to 20 groups can be added.	最多添加20个设备组	请确认当前设备组的数量
28823	Up to 100 devices can be added.	最多添加100个设备	请确认当前设备的数量

公共返回码

结果码	说明
0	成功
100001	AccessToken过期

结果码	说明
100002	用code换取access token值失败
100003	RefreshToken已过期或已失效
100004	用户改密或解除授权导致AccessToken失效
100005	AccessToken非法
100006	无效的OpenId
110000	系统通用错误（无法列出其他所有错误，需联系三方厂商）

设备端错误码

错误码	错误码说明	App展示的提示文案	解决办法
50176（0xC400）	Wi-Fi驱动失败	设备添加失败，请重试	驱动调用失败，重启设备再试或返厂检查
50208（0xC420）	配网开启失败	设备添加失败，请重试	驱动调用失败，重启设备再试或返厂检查
50240（0xC440）	获取AP的SSID与PWD超时	设备连接Wi-Fi超时，请重试	SSID与PWD获取超时，需重试
50241（0xC441）	对端发来的配网协议版本号不符合	设备工作异常，请联系售后支持或提交反馈	配网协议版本不符，检查移动端和设备端版本号确认
50242（0xC442）	配网报文校验出错	设备工作异常，请联系售后支持或提交反馈	配网报文校验出错，需确认设备固件以及设备证书信息是否有误
50243（0xC443）	解码AP的SSID与PWD错误	设备工作异常，请联系售后支持或提交反馈	配网报文解析出错，需确认设备固件以及设备证书信息是否有误
50272（0xC460）	设备热点开启失败	设备添加失败，请重试	驱动调用失败，重启设备再试或返厂检查
50273（0xC461）	设备热点启动后等待App连接超时	设备等待手机连接超时，请重试	设备热点等待连接超时，需确认对端行为并重试
50274（0xC462）	设备热点被App连接后未收到过SSID与PWD	设备连接Wi-Fi超时，请重试	SSID与PWD获取超时，需重试设备状态不对，或回包给App时设备底层出错

错误码	错误码说明	App展示的提示文案	解决办法
50275（0xC463）	对端发来的配网协议版本号不符合	设备工作异常，请联系售后支持或提交反馈	配网协议版本不符，检查移动端和设备端版本号确认App发来的配网包数据不对
50276（0xC464）	配网报文校验出错	设备工作异常，请联系售后支持或提交反馈	配网报文校验出错，需确认设备固件以及设备证书信息是否有误设备校验配网包数据时设备底层出错，导致设备校验出错
50277（0xC465）	解码AP的SSID与PWD错误	设备工作异常，请联系售后支持或提交反馈	配网报文解析出错，需确认设备固件以及设备证书信息是否有误
50278（0xC466）	设备热点切换到station模式失败	设备热点切换失败，请重试	驱动调用失败，重启设备再试或返厂检查
50304（0xC480）	等待零配超时（未收到SSID与PWD）	设备连接Wi-Fi超时，请重试	SSID与PWD获取超时，需重试
50305（0xC481）	对端发来的配网协议版本号不符合	设备工作异常，请联系售后支持或提交反馈	配网协议版本不符，检查移动端和设备端版本号确认
50306（0xC482）	配网报文校验出错	设备工作异常，请联系售后支持或提交反馈	配网报文校验出错，需确认设备固件以及设备证书信息是否有误
50307（0xC483）	解码AP的SSID与PWD错误	设备工作异常，请联系售后支持或提交反馈	配网报文解析出错，需确认设备固件以及设备证书信息是否有误
50336（0xC4A0）	BLE广播开启失败	设备蓝牙初始化失败，请重试	蓝牙驱动调用失败，重启设备再试或返厂检查
50337（0xC4A1）	BLE连接建立失败	设备蓝牙连接失败，请重试	蓝牙连接失败，需确认蓝牙信号以及设备设备证书信息是否正确
50338（0xC4A2）	BLE连接建立后未收到过SSID与PWD	设备连接Wi-Fi超时，请重试	SSID与PWD获取超时，需重试
50339（0xC4A3）	对端发来的配网协议版本号不符合	设备工作异常，请联系售后支持或提交反馈	配网协议版本不符，检查移动端和设备端版本号确认

错误码	错误码说明	App展示的提示文案	解决办法
50340（0xC4A4）	配网报文校验出错	设备工作异常，请联系售后支持或提交反馈	配网报文校验出错，需确认设备固件以及设备证书信息是否有误
50341（0xC4A5）	解码AP的SSID与PWD错误	设备工作异常，请联系售后支持或提交反馈	配网报文解析出错，需确认设备固件以及设备证书信息是否有误
50342（0xC4A6）	BLE连接断开失败	蓝牙断开失败，请重试	蓝牙断开失败，从手机端断开蓝牙连接或重启设备
50343（0xC4A7）	BLE广播关闭失败	蓝牙关闭失败，请重试	蓝牙驱动调用失败，重启设备再试或返厂检查
50400（0xC4E0）	路由器无法发现	未找到可用的Wi-Fi，请检查以下问题后重试： • Wi-Fi信号是否正常 • 设备与路由器距离是否较远 • 路由器Wi-Fi是否正确	需确认路由器信号，SSID输入是否正确，路由器是否处于2.4G工作频段
50401（0xC4E1）	路由器信号太弱连接失败（附上rssi信息）	设备连接Wi-Fi失败，请检查以下问题后重试： • Wi-Fi信号是否正常 • 设备与路由器距离是否较远	需确认路由器信号，以及设备与路由器之间的距离
50402（0xC4E2）	路由器连接认证失败	设备连接Wi-Fi失败，请重新输入正确的Wi-Fi密码	需确认配网阶段输入的SSID/密码是否正确
50403（0xC4E3）	IP地址获取失败	设备连接Wi-Fi失败，请检查路由器设置是否正常	需检查路由器是否工作正常，以及确认是否还能再加入STA路由器连接失败，可能情况有多种
50404（0xC4E4）	连接路由器本地通告失败	设备工作异常，请重试	接口调用失败，需重启设备重试

错误码	错误码说明	App展示的提示文案	解决办法
50432（0xC500）	https初始化失败	设备初始化失败，请重试	驱动调用失败，重启设备再试或返厂检查
50433（0xC501）	https域名解析失败（附上域名信息）	设备解析域名失败，请检查当前Wi-Fi是否正常	https初始化出错表明设备底层系统已经出错，Host等信息都出现了错误
50434（0xC502）	https预认证失败（认证超时）	设备连接Wi-Fi超时，请检查当前Wi-Fi是否正常	https域名解析失败，需检查当前路由器网络是否正常
50435（0xC503）	https预认证失败（设备证书错误）	设备工作异常，请联系售后支持或提交反馈	https预认证失败超时，需检查当前路由器网络是否正常预认证数据中含有非法字段导致
50436（0xC504）	mqtt开启失败	设备初始化失败，请重试	https预认证失败，需确认设备设备证书信息是否正确云端返回明确的code指示错误
50437（0xC505）	mqtt连云超时失败	设备连接Wi-Fi超时，请检查当前Wi-Fi是否正常	驱动调用失败，重启设备再试或返厂检查
50438（0xC506）	mqtt认证失败	设备工作异常，请联系售后支持或提交反馈	mqtt连云超时，需检查当前路由器网络是否正常分证书失效、认证被拒绝、服务端无法进行认证等情况
50439（0xC507）	设备coap服务开启失败	设备初始化失败，请重试	mqtt认证失败，需确认设备设备证书信息是否正确失败可能情况有许多种，需要根据具体错误消息确认
50440（0xC508）	设备绑定token上报云端超时	设备连接Wi-Fi超时，请检查当前Wi-Fi是否正常	驱动调用失败，重启设备再试或返厂检查
50441（0xC509）	设备绑定token上报云端回应报文错误	设备工作异常，请联系售后支持或提交反馈	token上报云端超时，需检查当前路由器网络是否正常
			token上报云端回应报文错误，需云端查询具体原因

错误码	错误码说明	App展示的提示文案	解决办法
50442（0xC50A）	接收到同一局域网内其他设备的token查询请求的报文不合法	设备工作异常，请联系售后支持或提交反馈	token查询请求的报文不合法，需查询端确认具体原因
50443（0xC50B）	对同一局域网内其他设备的token查询回复失败	设备工作异常，请联系售后支持或提交反馈	接口调用失败，需重启设备重试

1.2 加签算法

介绍生活物联网平台在开发中用到的加签算法。

系统级Header

- 【必选】X-Ca-Key：AppKey。
- 【必选】X-Ca-Signature：签名字符串。
- 【可选】X-Ca-Timestamp：API 调用者传递时间戳，值为当前时间的毫秒数，也就是从1970年1月1日起至今的时间转换为毫秒，时间戳有效时间为15分钟。
- 【可选】X-Ca-Nonce：API 调用者生成的 UUID，结合时间戳防重放。
- 【可选】X-Ca-Stage请求 API 所属 Stage，目前仅支持 TEST 、PRE 和 RELEASE，默认 RELEASE，若您调用的 API 不在线上环境，请一定要指定该参数的值，否则会报 URL 错误。

签名校验

1. 构建加密字符串。

```
String stringToSign=
HTTPMethod + "\n" +
Accept + "\n" + //建议显示设置 Accept Header。当 Accept 为空时，部分 Http 客户端会给 Accept 设置默认值为 */*，导致签名校验失败。
Content-MD5 + "\n"
Content-Type + "\n" +
Date + "\n" +
Headers +
```

```
Url
```

其中，**HTTPMethod**为全大写，如 POST。**Accept**、**Content-MD5**、**Content-Type**、**Date**如果为空也需要添加换行符 ”\n” ；**Headers**如果为空不需要添加 ”\n” 。

- Content-MD5

  Content-MD5是指Body的MD5值，只有当Body非Form表单时才计算 MD5，计算方式如下。

  ```
  String content-MD5 = Base64.encodeBase64(MD5(bodyStream.getbytes("UTF-8"))); // bodyStream为字节数组
  ```

- Headers

  Headers是指参与签名计算的Header的Key、Value拼接的字符串，建议对 X-Ca 开头的Header和自定义Header计算签名。

  > **说明:**
  > 以下参数不参与Headers签名计算：X-Ca-Signature、X-Ca-Signature-Headers、Accept、Content-MD5、Content-Type、Date。

  Headers组织方法如下。

  对参与签名计算的Header的Key按照字典排序后使用。如果某个Header的Value为空，则使用HeaderKey + “:” + “\n” 参与签名，需要保留Key和英文冒号。示例如下所示。

  ```
  String headers =
  HeaderKey1 + ":" + HeaderValue1 + "\n"\+
  HeaderKey2 + ":" + HeaderValue2 + "\n"\+
  ...
  HeaderKeyN + ":" + HeaderValueN + "\n"
  ```

  此外，还需要将所有参与签名的Header的key拼接到一起（用英文冒号分隔，假设此处拼接得到的字符串为Key1:Key2:Key3），且在调用API的HTTP Request中添加对应的Header项，即X-Ca-Signature-Headers：Key1:Key2:Key3。

- URL

  URL指Path+Query+Body中Form参数，组织方法如下。

  对Query+Form参数按照字典对Key进行排序后，按照如下方法拼接，如果Query或Form参数为空，则URL=Path，不需要添加，如果某个参数的Value为空只保留Key参与签名，等号则不需要再加入签名。

  ```
  String url =
  Path +
  "?" +
  Key1 + "=" + Value1 +
  "&" + Key2 + "=" + Value2 +
  ...
  "&" + KeyN + "=" + ValueN
  ```

2. 计算签名。

```
Mac hmacSha256 = Mac.getInstance("HmacSHA256");
byte[] keyBytes = secret.getBytes("UTF-8"); //如果是App端调用则为App的appsecret；如果是云端API调用则为云云对接的appSecret
hmacSha256.init(new SecretKeySpec(keyBytes, 0, keyBytes.length, "HmacSHA256"));
String sign = new String(Base64.encodeBase64(hmacSha256.doFinal(stringToSign.
getBytes("UTF-8")),"UTF-8"));
```

3. 传递签名。

   将计算的签名结果放到 Request 的 Header 中，Key为：X-Ca-Signature。

4. 排查签名错误。

   当签名校验失败时，API网关会将服务端的StringToSign放到HTTP Response的Header中返回到客户端，Key为：X-Ca-Error-Message，只需要将本地计算的StringToSign与服务端返回的StringToSign进行对比即可找到问题。

   如果服务端与客户端的StringToSign一致请检查用于签名计算的密钥是否正确。因为HTTP Header中无法表示换行，因此StringToSign中的换行符都被过滤掉了，对比时请忽略换行符。

5. 签名demo。

   签名计算的详细 demo（JAVA）请参照链接：https://github.com/aliyun/api-gateway-demo-sign-java

2 设备端开发指南

2.1 SDK概述与开发环境设置

介绍生活物联网平台的SDK与开发环境的安装。

生活物联网平台SDK

生活物联网平台SDK是阿里云IoT针对生活物联网平台所提供的设备端SDK。SDK基于C语言开发，代码开源，提供了API供您调用，用于实现与阿里云IoT平台通信以及其它的辅助功能（例如WiFi配网、本地控制等）。

生活物联网平台SDK分两种，您可以通过阿里云账号（普通账号即可）下载对应版本的生活物联网平台SDK。

- 包含AliOS Things的SDK（基于AliOS Things V1.3.4）

  如果您基于AliOS Things操作系统开发，请选择该版本SDK。

版本号	下载方式	SDK
V1.3.0	zip压缩包	生活物联网平台SDK（V1.3.0）
	git链接	git clone git@code.aliyun.com:living_platform/ali-smartliving-device-alios-things.git -b rel_1.3.0
V1.1.0	zip压缩包	生活物联网平台SDK（V1.1.0）
	git链接	git clone git@code.aliyun.com:living_platform/ali-smartliving-device-alios-things.git -b rel_1.1.0

  说明:
  如果您是新用户，建议您下载最新版本V1.3.0 SDK；如果您无需使用新版本的功能（功能更新请参见#unique_8），也可以继续使用V1.1.0版本SDK。

- 无AliOS Things的SDK（基于Link Kit V2.3.0）

  如果您基于其他操作系统开发（例如Linux、FreeRTOS、Windows等），请选择该版本SDK。

版本号	下载方式	SDK（无AliOS Things）
V1.3.0	zip压缩包	生活物联网平台SDK（V1.3.0）
	git链接	git clone git@code.aliyun.com:living_platform/ali-smartliving-device-sdk-c.git -b rel_1.3.0
V1.1.0	zip压缩包	生活物联网平台SDK（V1.1.0）
	git链接	git clone git@code.aliyun.com:living_platform/ali-smartliving-device-sdk-c.git -b rel_1.1.0

说明：

如果您是新用户，建议您下载最新版本V1.3.0 SDK；如果您无需使用新版本的功能（功能更新请参见#unique_8），也可以继续使用V1.1.0版本SDK。

设置开发环境

建议在64位Ubuntu下搭建开发环境，并使用vim编辑代码。

- 安装程序运行时库

  Ubuntu（版本16.04 X64）

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

- 安装依赖软件包

  Ubuntu（版本16.04 X64）

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

- 安装Python packages

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

```
python -m pip install scons
```

2.2 国际站设备开发

国内站产品与国际站产品的开发有些区别，本文主要介绍国际站设备端开发需要额外修改的内容。

背景介绍

目前在中国内地之外的国家和地区（包括港澳台地区）支持3个region：新加坡、美国和德国。中国内地之外的设备激活联网时，将统一连接到新加坡激活中心。在设备绑定时，平台将根据App用户所在区域，自动将设备切换到相应的region（示例如下）。

- 示例一：App用户注册时选择的国家为美国，平台会将该用户待绑定的设备切换到美国的服务器。
- 示例二：App用户注册时选择的国家为欧洲国家，平台会将该用户待绑定的设备切换到德国的服务器。
- 示例三：App用户注册时选择的国家为东南亚国家，设备连接新加坡服务器，此时无需切换region。

请确认您使用的生活物联网平台SDK版本，并确保版本为V1.1.0及以上版本。该版本SDK集成了设备自动切换region的功能，无需您额外修改代码。获取SDK请参见SDK概述与开发环境设置。

基于V1.3.0及以上SDK编译

您基于V1.3.0以上版本开发时，请根据以下步骤编译国际站的固件。

1. 修改build.sh默认变量参数。

   更多build.sh的参数说明请参见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"
   连云区域：default_region=SINGAPORE //连接国内为MAINLAND
   连云环境：default_env=ONLINE
   Debug log：default_debug=0
   其他参数：default_args=""    //可配置其他编译参数
   ```

   说明：

   将**default_region**指定为**SINGAPORE**即可编译国际站固件，脚本会自动做好其他所需的配置。

   反之该参数指定为**MAINLAND**即可编译国内站固件。

2. 执行./build.sh 编译固件。编译输出的固件存放在out/smart_outlet@uno-91h/目录下。

基于V1.1.0 SDK编译（linkkitapp）

如果基于AliOS Things中的linkkitapp开发应用，设备端固件需要修改以下两个配置来支持统一连接到新加坡激活中心。请您根据以下步骤操作。

1. 更改example下应用代码的domain type设定。

   在example/linkkitapp/linkkit_example_solo.c文件的linkkit_main()函数中，如下图所示将**domain_type**的赋值从**IOTX_CLOUD_REGION_SHANGHAI**改为**IOTX_CLOUD_REGION_SINGAPORE**。

```
    /* Choose Login Server, domain should be configured before IOT_Linkkit_Open() */
#if USE_CUSTOME_DOMAIN
    IOT_Ioctl(IOTX_IOCTL_SET_MQTT_DOMAIN, (void *)CUSTOME_DOMAIN_MQTT);
    IOT_Ioctl(IOTX_IOCTL_SET_HTTP_DOMAIN, (void *)CUSTOME_DOMAIN_HTTP);
#else
    int domain_type = IOTX_CLOUD_REGION_SINGAPORE;
    IOT_Ioctl(IOTX_IOCTL_SET_DOMAIN, (void *)&domain_type);
#endif
```

2. 在example/linkkitapp/make.settings文件中，将**FEATURE_MQTT_DIRECT**的状态设置为关闭（即如下图所示配置为**n**）。

```
#
# MQTT Configurations
#
FEATURE_MQTT_DIRECT=n
# FEATURE_MQTT_SHADOW is not set
# FEATURE_MQTT_LOGPOST is not set
FEATURE_DEVICE_MODEL_ENABLED=y
```

3. 在example/linkkitapp/app_entry.c文件的函数application_start中，确认set_iotx_info()放在netmgr_init()前面，如下图所示。

```
int application_start(int argc, char **argv)
{
#ifdef CONFIG_PRINT_HEAP
    print_heap();
    aos_post_delayed_action(5000, duration_work, NULL);
#endif

#ifdef CSP_LINUXHOST
    signal(SIGPIPE, SIG_IGN);
#endif

#ifdef WITH_SAL
    sal_init();
#endif

#ifdef MDAL_MAL_ICA_TEST
    HAL_MDAL_MAL_Init();
#endif
    aos_set_log_level(AOS_LL_DEBUG);

    set_iotx_info();
    netmgr_init();
    aos_register_event_filter(EV_KEY, linkkit_key_process, NULL);
    aos_register_event_filter(EV_WIFI, wifi_service_event, NULL);
    aos_register_event_filter(EV_YUNIO, cloud_service_event, NULL);
    IOT_RegisterCallback(ITE_MQTT_CONNECT_SUCC, mqtt_connected_event_handler);

#ifdef CONFIG_AOS_CLI
    aos_cli_register_command(&resetcmd);
    aos_cli_register_command(&awss_enable_cmd);
    aos_cli_register_command(&awss_dev_ap_cmd);
    aos_cli_register_command(&awss_cmd);
#endif

    extern void LITE_set_loglevel(int);
    LITE_set_loglevel(5);
```

4. 编译linkkitapp程序。

   以mk3080模组为例，执行以下代码编译。

   ```
   aos make linkkitapp@<模组名>
   ```

   编译输出的固件存放在out/linkkitapp@mk3080/binary/ 目录下。

基于V1.1.0 SDK编译（living_platform）

如果基于含AliOS Things中的living_platform开发应用，example/living_platform/gen_firmware.sh编译脚本支持通过不同选项编译中国内地和中国内地以外地区的固件。编译时根据该选项，**domain type**变量和宏**FEATURE_MQTT_DIRECT**会被自动配置，无需再手动修改。

请根据以下步骤基于gen_firmware.sh脚本编译固件。

1. 打开gen_firmware.sh文件，按红色标识指定board。

```
~/code/ali-smartliving-device-alios-things/example/living_platform
#!/bin/sh
#gen_firmware.sh

#Firmware image path
project_example=living_platform
project_board=hf-lpt230
product_vendor=ali
product_name=smart_living
```

2. 执行以下命令编译脚本。

```
bash gen_firmware.sh
```

```
                ali_smart_living firmware compile menu
************************************************************************
@@@ Firmware type: release
@@@ Board name:    hf-lpt230
@@@ Application:   living_platform
************************************************************************

(1) gen mainland daily
(2) gen mainland pre
(3) gen mainland online
(4) gen overseas(singapore) pre
(5) gen overseas(singapore) online
(a) gen all firmwares
(f) gen factory firmwares
(c) clean all
(t) toggle firmware type
(q) Quit
Please enter your selection: 5
```

3. 输入编译固件的编码。

   固件的编码解释如下。

   - c：每次编译前需进行clean操作
   - 3：编译国内站固件
   - 5：编译国际站固件

4. 查看编译结果。

   该示例编译输出的固件存放在out/living_platform@boardname/binary目录下。

常见问题

Q：国际站设备是否可以不关闭**FEATURE_MQTT_DIRECT**?

A：国际站设备必须关闭**FEATURE_MQTT_DIRECT**，否则会存在GDPR合规风险，以及设备通过公版App配网绑定后在部分区域无法正常激活的问题。

Q：如果已有**FEATURE_MQTT_DIRECT**未关闭的设备送样、出货应该怎么处理?

A：请尽快按本文档中描述的步骤修改，并进行设备固件OTA升级，设备升级时请注意选择静默升级（即App确认升级选项选否），并且升级策略选择动态升级（动态升级将持续维护需升级的设备范围，包括当前已经上报版本号的设备和新激活的设备），需要先做固件验证，再做批量升级，详细的OTA步骤请参见#unique_10。

Q：国内站设备打开和关闭**FEATURE_MQTT_DIRECT**有什么区别?

A：打开**FEATURE_MQTT_DIRECT**为直连模式，关闭**FEATURE_MQTT_DIRECT**为预认证模式，国内设备在两种模式下都能建立连接，区别是后者连接速度慢一点。因此，建议国内站设备用直连模式，国际站设备用预认证模式。

Q：国内站设备和国际站设备需要维护两个固件吗?

A：是的，国内站设备和国际站设备的ProductKey是不一样的，生产烧录的时候对设备证书和固件要做相应的管理。

Q：国际站设备一定要用生活物联网平台SDK v1.1.0及以上的版本吗?

A：是的，中国内地以外国家和地区（包括港澳台地区）统一激活中心需要设备端支持region切换。

后续步骤

完成国际站设备端开发后，您还需要开发App端，请参见#unique_11。

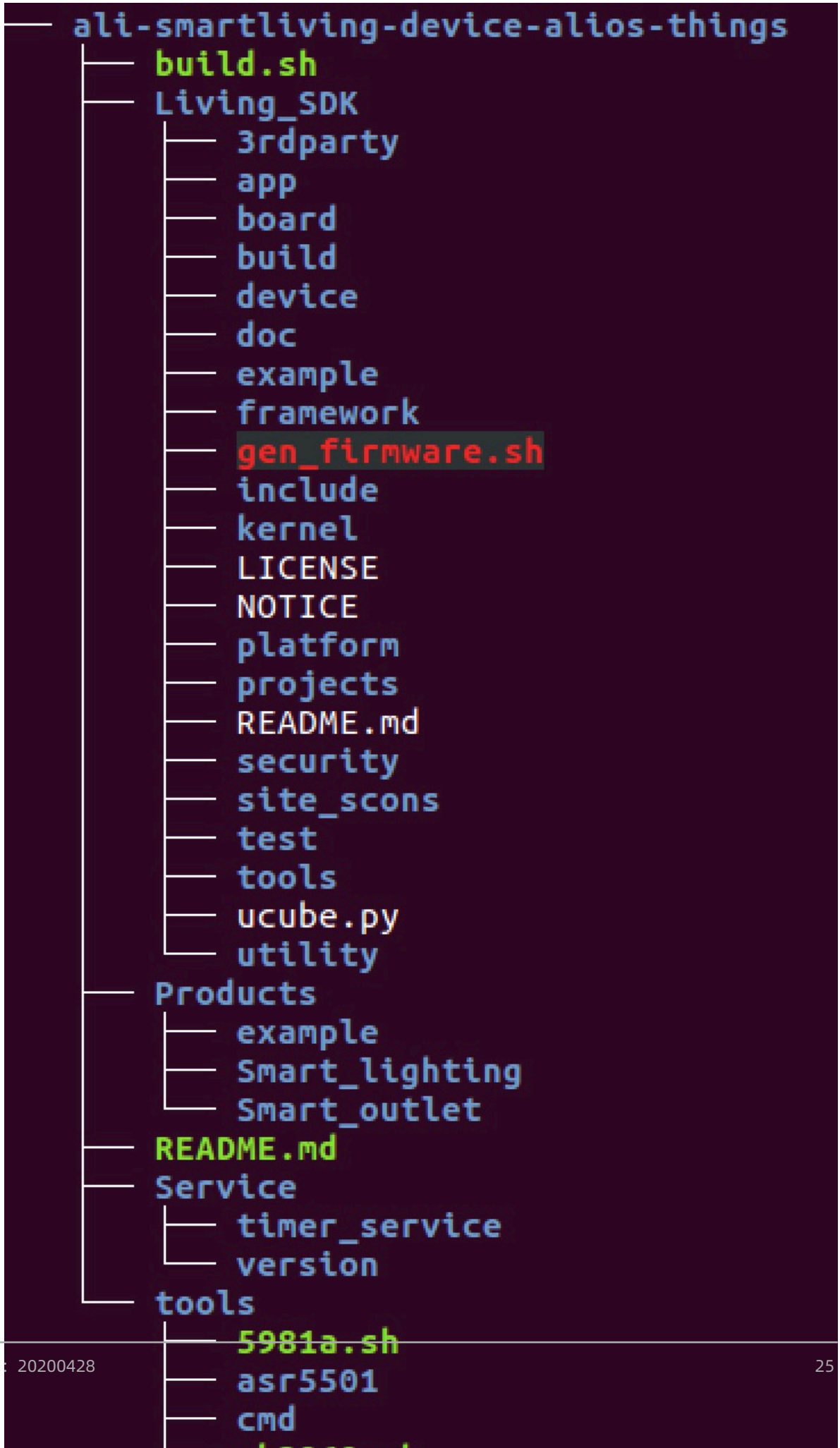
2.3 配置SDK新增功能

本文介绍生活物联网SDK V1.3.0版本相对于SDK V1.1.0新增的功能，以及开启相应功能的方法。

生活物联网平台SDK V1.3.0是基于上一个版本（V1.1.0）的功能递增，主要优化了目录结构和新增了功能。

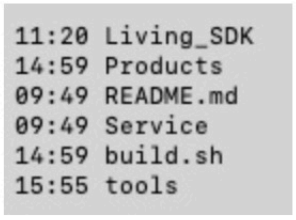
- 优化目录结构

  将开发的代码存放到Products/品类目录/应用名/...下（例如Products/example/smart_outlet为单路智能插座参考实现）。SDK V1.3.0的详细目录结构如下图所示。


ali-smartliving-device-alios-things
build.sh
Living_SDK
3rdparty
app
board
build
device
doc
example
framework
gen_firmware.sh
include
kernel
LICENSE
NOTICE
platform
projects
README.md
security
site_scons
test
tools
ucube.py
utility
Products
example
Smart_lighting
Smart_outlet
README.md
Service
timer_service
version
tools
5981a.sh
asr5501
cmd

SDK V1.3.0版本与SDK V1.1.0的目录结构对比，如下图所示。

rel_1.1.0 及之前的版本目录结构



rel_1.3.0 及之后的目录结构

- 新增功能

  除了优化了目录结构，生活物联网平台SDK V1.3.0相比SDK V1.1.0主要新增以下功能。

  - 设备厂商测试批量配网
  - 终端用户批量配网
  - 设备热点配网错误码诊断
  - 本地组控/组律动
  - 离线OTA
  - WiFi设备零配token优化

设备厂商测试批量配网

如果您是设备厂商，您可以在产线上将设备连到指定的产测路由器，开启此功能时设备在上电3s内自动连接路由器。

该功能默认关闭，您如需开启此功能，请按以下步骤操作。

1. 在应用mk文件中增加配置项GLOBAL_CFLAGS += -DMANUFACT_AP_FIND_ENABLE。

2. 在app_entry.c文件中更改产测路由器SSID和密码。

   - 默认SSID为ali_mprov_TEST_AP，其中ali_mprov_是模块自动加入的SSID前缀，TEST_AP是开发者可自定义设定的字段
   - 默认密码为TEST_PASSWORD

```
app_entry.c ×
795     ota_init();
796 #endif
797     netmgr_init();
798 #ifdef MANUFACT_AP_FIND_ENABLE
799     {
800         uint8_t m_manu_flag = 0;
801         int m_manu_flag_len = sizeof(m_manu_flag);
802         aos_kv_get(MANUAP_CONTROL_KEY, &m_manu_flag, &m_manu_flag_len);
803         if (m_manu_flag) {
804             netmgr_manuap_info_set("TEST_AP", "TEST_PASSWORD", manufact_ap_find_process);
805         }
806     }
807 #endif
```

SSID后缀　Password

说明：

如果从应用层设定的部分SSID字段，或路由器密码字段为空，该模块功能不会启用。如果设定了正确的SSID和密码，设备在未配网状态下开机时会搜索该路由器，如果搜索到就会连接，搜索不到则会进入正常的待配网状态。

终端用户批量配网

支持App终端用户对多个同一型号的设备进行快速的批量配网，设备需支持设备热点配网或零配，每次最多可批量配网20个设备。

您如需开启此功能，需在应用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）。

```
living_platform.mk ×
57  #Create thread of cm_yield in mqtt_client.c
58  GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1
59
60  GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
61                   -DDEV_OFFLINE_OTA_ENABLE \
62                   -DAWSS_BATCH_DEVAP_ENABLE \
63                   -DAWSS_ZCONFIG_APPTOKEN
```

设备热点配网错误码诊断

此功能支持设备通过热点配网连路由器或连云过程中产生的异常，通过App端的“错误诊断”功能展示出来。

您如需开启此功能，需在应用mk文件中增加配置项GLOBAL_CFLAGS += -DDEV_ERRCODE_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）。

```
living_platform.mk
57  #Create thread of cm_yield in mqtt_client.c
58  GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1
59
60  GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
61                   -DDEV_OFFLINE_OTA_ENABLE \
62                   -DAWSS_BATCH_DEVAP_ENABLE \
63                   -DAWSS_ZCONFIG_APPTOKEN
```

本地组控/组律动

此功能提供了灯品类组内设备同步律动能力。

您如需开启此功能，请按以下步骤操作。

1. 确认本地通信功能已经打开，即make.settings文件中**FEATURE_ALCS_ENABLED**设置为**y**，如下图所示。

```
make.settings
7   # Configure Link Kit SDK for IoT Embedded Devices
8   #
9   FEATURE_SRCPATH="."
10  FEATURE_MQTT_COMM_ENABLED=y
11  FEATURE_ALCS_ENABLED=y
12  #
13  # MQTT Configurations
14  #
```

2. 在应用mk文件中增加配置项。如您基于living_platform开发，在living_platform.mk文件中增加以下配置项。

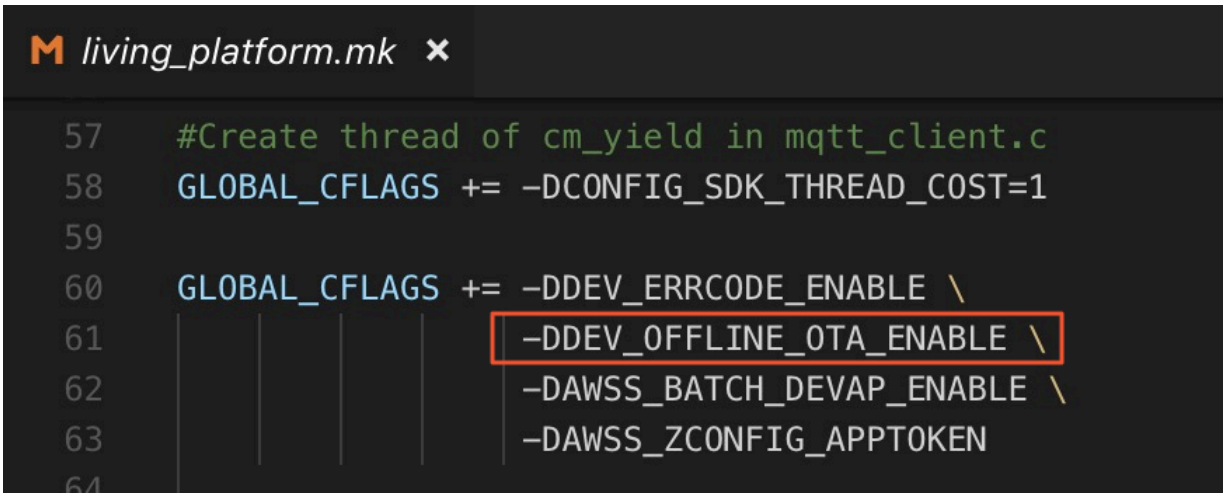
```
GLOBAL_CFLAGS += -DALCS_GROUP_COMM_ENABLE
```

```
GLOBAL_CFLAGS += -DDM_UNIFIED_SERVICE_POST
```

离线OTA

此功能提供了设备在设备热点下配网或连云失败后通过固件升级进行固件修复的手段，支持普通和安全离线升级。

您如需开启此功能，需在应用mk文件中增加配置项GLOBAL_CFLAGS += -DDEV_OFFLINE_OTA_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）。



您还可以按照以下步骤启用离线OTA安全签名校验。

1. 在应用mk文件中增加配置项：GLOBAL_CFLAGS += -DDEV_OFFLINE_SECURE_OTA_ENABLE。
2. 获取产品的公钥。在控制台的**运营中心** > **设备运维** > **固件升级** > **安全升级**页面中， 打开对应产品的安全升级开关，并单击**复制**。

3. 在Living_SDK/framework/uOTA/src/verify/ota_public_key_config.h文件中，将下图的两行数据（红框所示），替换为获取的公钥信息。

```
ota_public_key_config.h
#ifndef OTA_PUBLIC_KEY_CONFIG_H_
#define OTA_PUBLIC_KEY_CONFIG_H_

/*below two bufs are about the rsa public key, user needs sign in alibaba cloud to get them*/
//static const unsigned char ota_pubn_buf[256] = {0x00};
//static const unsigned char ota_pube_buf[3] = {0x00};


    static const unsigned char ota_pubn_buf[256] = {0xA9, 0x82, 0x89, 0x44, 0x46, 0xC3, 0x1A, 0xA3, 0x50, 0x8C, 0x46, 0x
    static const unsigned char ota_pube_buf[3] = {0x01, 0x00, 0x01};

#endif
```

WiFi设备零配token优化

此功能支持在零配过程中，由App端生成绑定用的token并传输给设备，从而解决零配方案在路由器设置AP隔离模式下绑定失败的问题。

您如需开启此功能，需在应用mk文件中增加配置项GLOBAL_CFLAGS += -DAWSS_ZCONFIG_APPTOKEN（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）。

```
living_platform.mk
#Create thread of cm_yield in mqtt_client.c
GLOBAL_CFLAGS += -DCONFIG_SDK_THREAD_COST=1

GLOBAL_CFLAGS += -DDEV_ERRCODE_ENABLE \
                 -DDEV_OFFLINE_OTA_ENABLE \
                 -DAWSS_BATCH_DEVAP_ENABLE \
                 -DAWSS_ZCONFIG_APPTOKEN
```

编译SDK V1.3.0

您可通过更改build.sh默认变量参数，并执行./build.sh 来编译SDK，从而生成烧录所需的固件。build.sh的参数说明如下（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"
连云区域：default_region=SINGAPORE //连接国内为MAINLAND
连云环境：default_env=ONLINE
Debug log：default_debug=0
```

其他参数：default_args=""　//可配置其他编译参数

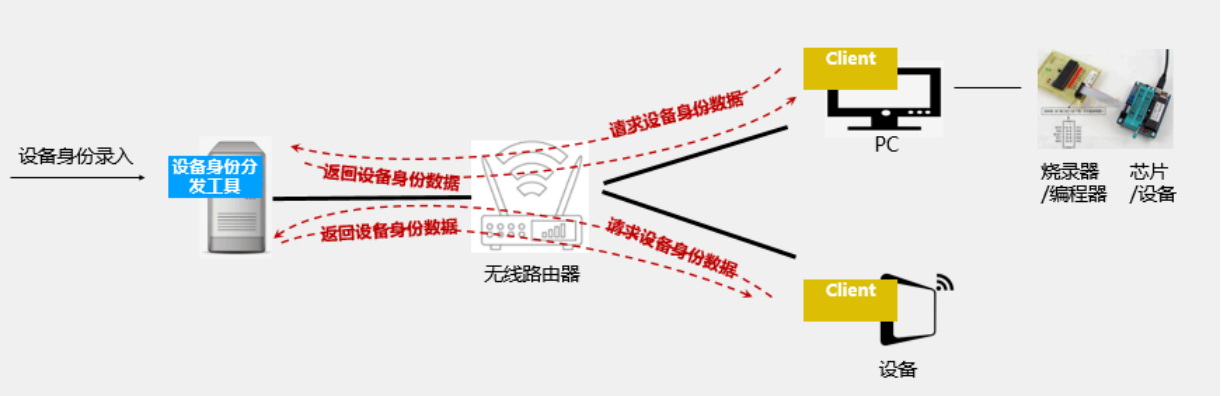
```
#!/bin/bash

##------default config-------------------

default_type="example"
default_app="smart_outlet"
default_board="uno-91h"
default_region=SINGAPORE
default_env=ONLINE
default_debug=0
default_args=""
```

2.4 设备证书分发工具使用指导

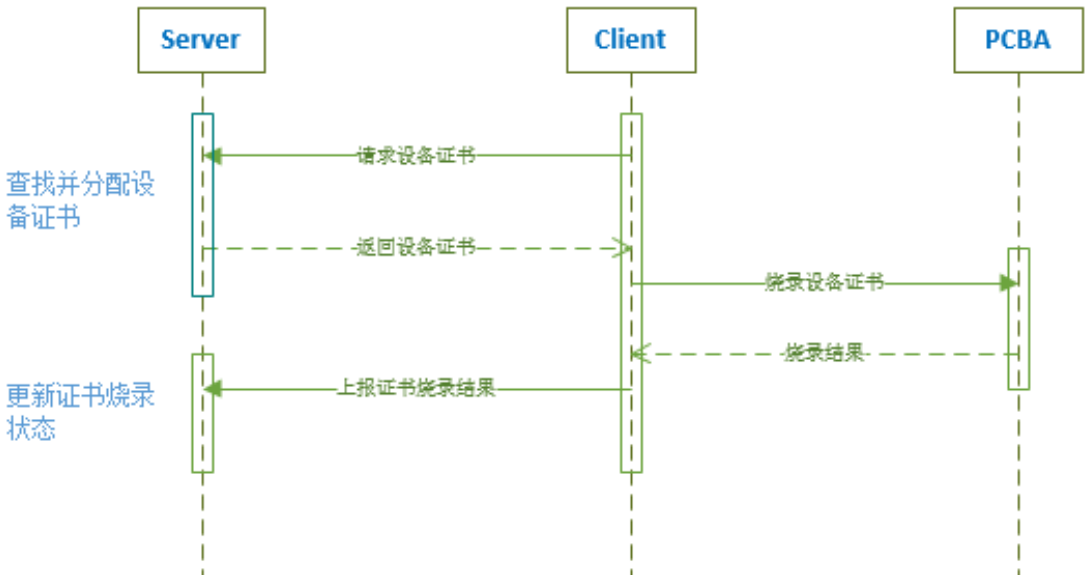
为了帮助您提高量产效率，快速从生活物联网平台下载的设备证书文件（csv格式）中读取设备证书信息，并烧写到产线上的设备中，生活物联网平台提供设备证书分发工具来帮助您完成csv文件读取及分发工作。您可以根据本文档改造您的产线。

方案介绍

IoT设备连接到生活物联网平台时需要认证设备身份（认证原理如下图所示），认证凭证为阿里云物联网为设备颁发的唯一的设备证书信息。



基于以上身份认证原理，生活物联网的设备身份分发工具采用Server-Client架构，Server与Client交互流程图如下所示。



- Server：由生活物联网平台提供，您下载后可直接使用（工具下载地址）。

  您可以使用生活物联网平台提供的NodeJS SDK（参见本文档下方相关内容），来验证Server端链路是否正常。

- Client：部署在产线烧录仪器或集成到产品固件中（两者对比如下表所示），需要您自行开发。

  为了方便您快速集成Client，生活物联网平台提供了C SDK供您参考（参见本文档下方相关内容）。 您集成C SDK后，再自行集成烧录设备证书的方式到Client中。

Client部署方式	适用场景	描述	特点
集成到设备固件中	产线上没有用于将设备证书烧写到芯片/设备的仪器。	该方案设备上电后，Client向Server申请身份信息，Client获取到设备证书后将这些信息写入设备的NVRAM/Flash。 该方案您需要实现以下内容： - 设计如何触发Client去获取设备证书的功能。 - Client获取设备证书后，需要实现将信息写入NVRAM/Flash的操作。	- 产线无需部署专门用于烧写设备证书的仪器。 - 设备需要实现Client的功能，因此会占用设备的部分存储空间。 - 可以实现批量进行身份信息获取。

Client部署方式	适用场景	描述	特点
部署在产线烧录仪器上	产线上存在将设备证书烧写到芯片/设备的仪器，如烧录器或编程器等。	该方案由烧录软件实现Client的功能向Server申请身份信息，取得设备证书后，通过烧录器/编程器写入设备的NVRAM/Flash。 阿里云物联网提供了产线分发App的通信协议，该方案您需要修改产线烧写工具按照该通信协议从产线分发App获取设备证书。	- 设备无需实现Client的功能，不会为了获取身份信息功能消耗存储空间。 - 您需要对烧录器/编程器进行软件改造。 - 可以根据产线上的生产设备数量、规模，配置烧录器/编程器的数量。

Server-Client协议说明

Client与Server采用HTTP接口通讯，Client通过POST方法向Server发送请求，请求协议以json方式放在body里。

表 2-1: 请求协议格式与参数说明

字段	类型	是否必选	描述
ProductKey	string	是	产品标识（生活物联网后台新建产品是生成）
DeviceID	string	否	设备ID，唯一标识
Action	Integer	是	操作 • 0：请求分配设备证书 • 1：上传烧录成功 • 2：上传烧录失败

请求与响应数据示例如下。

- Client向Server申请设备证书请求数据

```
{
  "ProductKey":"a1xxxxNW",
  "DeviceID":"AAxxxx22",
  "Action":0
}
```

- Server正常分发设备证书给Client数据

```
{
  "ProductKey":"a1xxxxNW",
```

```
   "ProductSecret":"XmDxxxx9R",
   "DeviceSecret":"NmMExxxxjE5",
   "CRC32":"B5488744",    //CRC32介绍参见下方文档
   "DeviceName":"ZTMxxxxxkUy",
    "DeviceID":"AABxxxx22",
    "Status":"OK"
}
```

- 烧录成功状态更新请求

```
{
   "ProductKey":"a1xxxxW",
   "DeviceID":"AAxxxx22",
   "Action":1
}{
   "ProductKey": "a1xxxxNW",
   "DeviceID": "AAxxxx22",
   "Status": "OK"
}
```

- 烧录成功状态更新返回数据

```
{
   "ProductKey": "a1xxxxNW",
   "DeviceID": "AAxxxx22",
   "Status": "OK"
}
```

Server分发设备证书给Client的数据中使用的校验算法采用标准CRC32（CRC32计算工具网址），多项式为04C11DB7，初始值与结果异或值均为FFFFFFFF，输入与输出均取反。校验payload为设备凭证（WiFi类设备为ProductKey、ProductSecret、DeviceName、DeviceSecret；蓝牙类设备为ProductKey、ProductSecret、DeviceName、DeviceSecret、ProductID）。校验结果为4字节十六进制。

CRC32计算结果校验工具使用方法的示例如下。

- java代码参考实现

```
import java.nio.ByteBuffer;
import java.util.zip.CRC32;

public class example {

public static void main(String[] args){

   ///this is user supplied string
   String pk  = "a1xxxxH";
   String dn  = "8jQxxxx0WW6";
   String ds  = "aW60ImxxxxIj61ZbV";
   String ps  = "EBcjxxxxWxdF";

   if ( args.length >= 4) {
      pk = args[0];
      dn = args[1];
      ds = args[2];
      ps = args[3];
   }

   String payload = pk + dn + ds + ps;
   ByteBuffer bbuffer = ByteBuffer.allocate(payload.length());
   bbuffer.put(payload.getBytes());

   //your crc class
   CRC32 crc = new CRC32();
   crc.update(bbuffer.array());
```

```
    String enc = Long.toHexString(crc.getValue()).toUpperCase();
    System.out.println("payload: " + payload);
    System.out.println("length: " + Integer.toString(payload.length()));
    System.out.println("crc32: " + enc);
    }
}
```

- python代码参考实现

```
import zlib

pk  = "a2xxxx8HZ"
dn  = "ouxxxxemo"
ds  = "GD4JauYxxxxsICazhIzb"
ps  = "4I3xxxxyF96"

def Crc32Hash(input_data):

    crc32 = 0
    crc32 = zlib.crc32(input_data, crc32)

    return format(crc32 & 0xFFFFFFFF, '08X')

payload = pk + dn + ds + ps
print "payload: ", payload
print "length: ", len(payload)
print "crc32: ", Crc32Hash(payload)
```

- 设备端参考代码

```
#define CHIP_CPU_BE

#define dwPolynomial_BE          0xEDB88320UL      //CRC32 Polynomial
#define dwPolynomial             0x04C11DB7UL      //CRC32 Polynomial
#if defined(CHIP_CPU_BE)
#define dwPolynomial             dwPolynomial_BE
#else
#define dwPolynomial             dwPolynomial_LE
#endif
#define CRC_INIT_VALUE           0xFFFFFFFFUL

unsigned int calc_crc32(unsigned char *message) {
  int i, j;
  unsigned int byte, crc, mask;

  i = 0;
  crc = CRC_INIT_VALUE;
  while (message[i] != 0) {
    byte = message[i];            // Get next byte.
    crc = crc ^ byte;
    for (j = 7; j >= 0; j--) {    // Do eight times.
      mask = -(crc & 1);
      crc = (crc >> 1) ^ (dwPolynomial_BE & mask);
    }
    i = i + 1;
  }
  return ~crc;
```

```
}
```

Sever端工具使用

Server端由设备证书分发和设备证书管理两部分组成。

1. 下载设备证书相关的工具（工具下载地址）。
2. 启动设备证书分发服务。

   a) 解压下载的zip，并双击server目录下的http_server.exe，来启动分发服务。

   出现以下消息时，表示分发服务启动成功。

   ```
   Waiting for client request
   ```

   说明:

当前版本还未做成系统服务，启动后请您不要关闭窗口。

b) （可选）修改端口号。

分发服务默认采用的端口号为**8000**，若需要更改端口请执行以下命令修改（以修改成**50000**为例）。

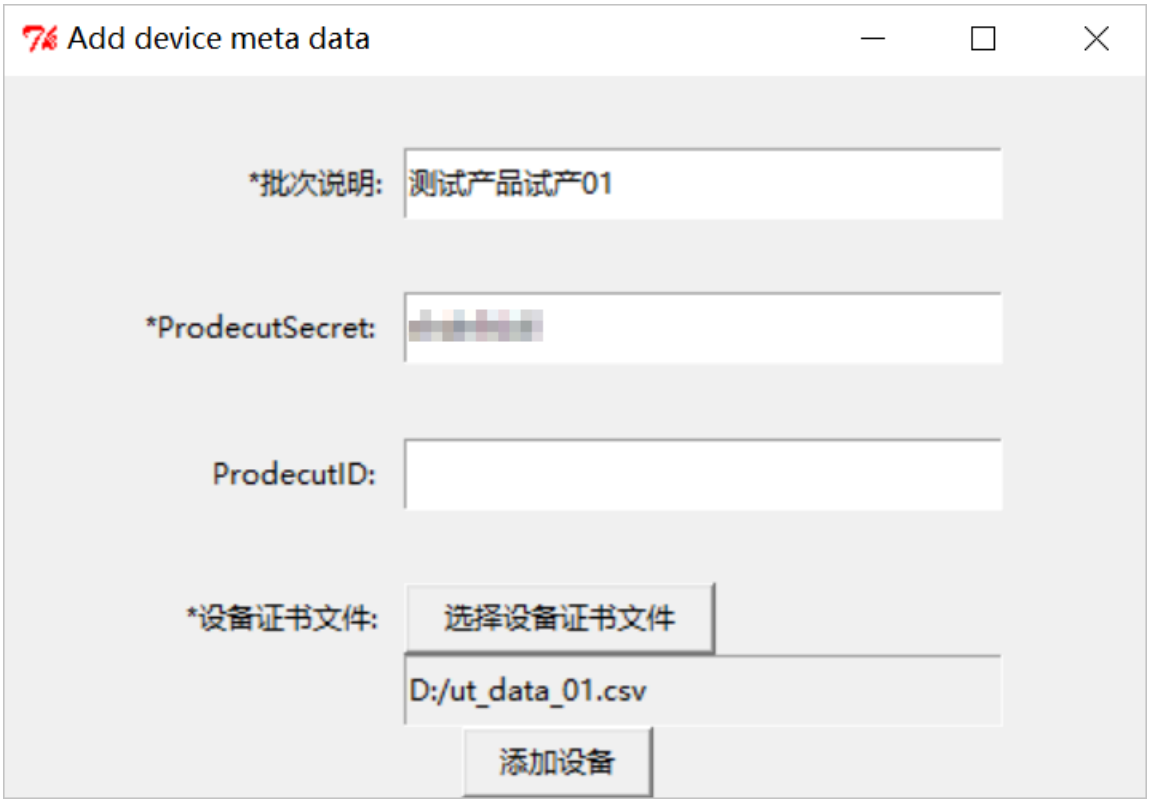
```
http_server.exe 50000
```

3. 管理产品批次，并导入设备证书。

a) 双击server目录下的factory.exe，来启动管理工具。

b) 单击**添加设备**。

c) 配置产品批次及产品信息，并单击**选择设备身份文件**，选择从生活物联网后台下载的设备身份文件。

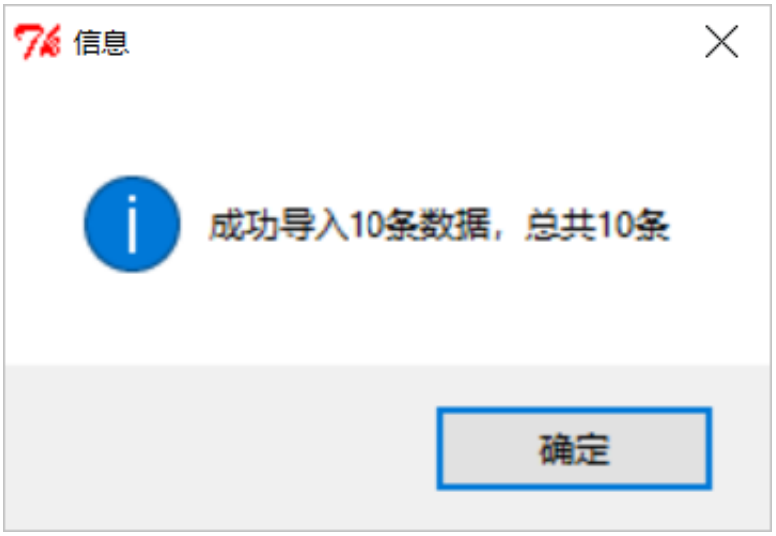


说明:

若身份文件带**ProductSecret**字段，程序会自动读取并以身份文件中的内容为准。

d) 单击**添加设备**按钮。

导入完成之后会弹出消息提示导入的详细信息。

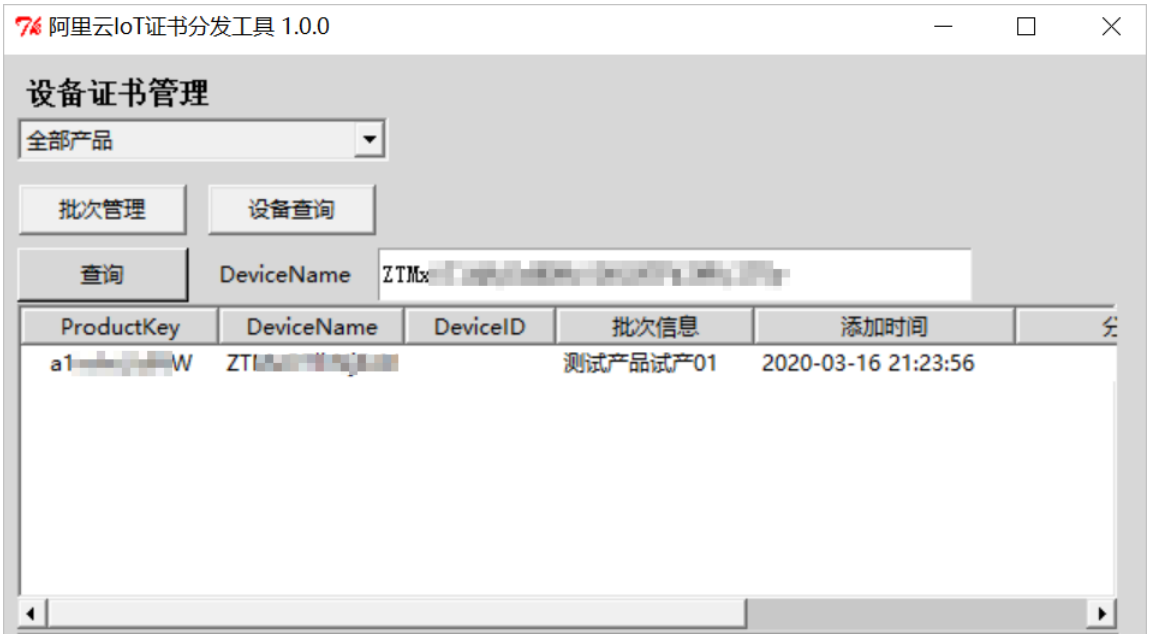


4. 查询产品信息。

   a) 单击**设备查询**，并在**DeviceName**输入框中输入待查询的产品名称。

   b) 单击**查询**。

   此时显示设备的详细信息。



（可选）使用NodeJS SDK验证Server

在设备证书分发工具（工具下载地址）中提供了NodeJS SDK参考实现，您可以使用NodeJS SDK来验证Server获取设备证书的链路是否正常。

您下载并解压的分发工具后，在client\NodeJS目录下查看NodeJS SDK的相关文件，文件说明如下。

- test.js：是一个示例文件，该文件调用devReqCert.js获取设备证书。
- devReqCert.js：提供req_dev_cert()和send_burn_result_by_did()两个函数（说明如下）。
  - req_dev_cert('ip',"pk",id,callback)

    向指定的Server申请设备证书。函数中参数说明如下。

参数	类型	描述
ip	String	Server的IP地址或者域名

参数	类型	描述
pk	String	设备希望申请的身份信息对应的产品型号，该参数对应阿里云IoT平台上产品的ProductKey
id	String	设备的唯一标识，可以是MAC地址、SN，只要是设备唯一的即可 ■ 如果使用同样的id申请，Server将会返回相同的设备证书 ■ 若id值为null，表示设备希望分配一个新的身份信息

参数	类型	描述
callback	function	当Server返回设备证书后调用的回调函数，用于保存设备证书。设备证书该函数需要由您自行实现

- send_burn_result_by_did('ip',"pk","id")

  向指定的Server发送设备身份信息烧写结果，Server收到Client发送的烧写结果后，将会把对应设备的身份信息状态从**已分配**修改为**已烧写**。函数中参数说明如下。

参数	类型	描述
ip	String	Server的IP地址或者域名
pk	String	设备希望申请的身份信息对应的产品型号，该参数对应阿里云IoT平台上产品的ProductKey

参数	类型	描述
id	String	设备的唯一标识，可以是MAC地址、SN，只要是设备唯一的即可 ■ 如果使用同样的id申请，Server将会返回相同的设备证书 ■ 若id值为null，表示设备希望分配一个新的身份信息

请您根据以下步骤开发并运行示例代码。

1. 开发示例代码。

基于文件“test.js”进行示例代码说明。

```
var http = require('http'); // 引入对http库的依赖
var devReq = require('./devReqCert'); //引入对devReqCert的依赖

var did = null;

//下面的函数handler是Client收到Server分配的设备证书后的处理函数，
//在本示例代码中仅仅将设备的身份信息进行打印输出，
//实际产品生产时需要将设备的身份信息进行持久化保存
function handler(devInfo)
{
   var jsonObj = JSON.parse(devInfo);
   console.log("DevInfo:" + devInfo);
}

//下面的语句调用req_dev_cert获取设备的身份信息，其中
//“127.0.0.1”是Server的IP地址，本测试中Server和Client在同一台PC上，所以使用了127.0.0.1地址
//参数"a1R3WBv1Bm9"是设备希望分配身份信息对应的ProductKey
//null表示设备没有唯一标识，希望Server分配一个新的设备身份
devReq.req_dev_cert('127.0.0.1',"a1R3WBv1Bm9",null, handler);

//下面的代码设置了设备的唯一标识，并再次申请设备的唯一标识
did = "123456";
devReq.req_dev_cert('127.0.0.1',"a1R3WBv1Bm9",did,handler);
```

2. 在运行Client的设备/PC上部署NodeJS环境。
3. 进入产线分发工具的NodeJS目录，并运行**node test.js**。

   输出的内容如下所示则Server获取设备证书链路正常。其中DeviceName、ProductKey、ProductSecret、DeviceSecret是阿里云IoT平台为设备分配的设备证书，DeviceID是设备申请设备身份时传入的设备标识。

使用C SDK开发Client

设备获取身份信息的功能，可采用以下两种实现方式。

- 产线上将设备的某个GPIO作为输入接口，将其拉高或者拉低，当设备上电运行时发现该GPIO为高/低时，自动连接某个固定的WiFi AP，然后从Server（Server的IP地址需要固定）去申请设备证书。
- 设备上电后检查是否有设备身份，如没有设备身份信息，则自动连接到一个固定的产线AP（AP的SSID和密码由您自行定义）。此时请参照设备证书分发工具（工具下载地址）提供的C-SDK（调用get_triple）去申请设备身份。

您下载并解压的分发工具后，在client\C-SDK目录下查看C SDK的相关文件，文件说明如下。

- triple_burn.c：用来存放SDK代码。
- triple_burn.h：头文件，包含send_req()和parse_data()两个函数（说明如下）。
  - char* send_req(act, pk, did, const server_ip)

    向指定的Server申请设备证书，或发送设备证书烧写结果设备证书。其中参数说明如下。

参数	类型	描述
act	Int	■ 0：Client向Server请求设备身份 ■ 1：Client告知Server设备身份写入成功

参数	类型	描述
pk	char *	设备希望申请的身份信息对应的产品型号，该参数对应阿里云IoT平台上产品的ProductKey
did	char *	设备的唯一标识，可以是MAC地址、SN等唯一的设备标识 特殊情况说明： ■ 使用同样的id申请，Server将返回相同的设备证书 ■ 设备不具备唯一的标识，可传入空字符串，但请谨慎操作，可能会造成设备证书浪费

参数	类型	描述
server_ip	char *	Server服务所在设备的IP地址

Server返回的设备身份，格式为JSON。若为NULL表示申请身份失败。

- int parse_data(const data, pk, dn, ps, ds, did)

该函数用于解析从Server返回的JSON字符串，并获取其中的ProductKey、ProductSecret、DeviceName、DeviceSecret、DeviceID（如果您还需在Server处为每个设备增加其它的配置信息，可自行修改本函数）。

参数	类型	描述
data	char *	函数 `send_req()`从Server获取到的设备证书
pk	char *	设备的产品型号，该参数对应阿里云IoT平台上产品的ProductKey
dn	char *	设备的名称，该参数对应阿里云IoT平台上设备的DeviceName
ps	char *	该参数对应阿里云IoT平台上设备的DeviceSecret
ds	char *	该参数对应阿里云IoT平台上设备的DeviceSecret

参数	类型	描述
did	char *	设备的唯一标识，可以是MAC地址、SN等唯一的设备标识。 特殊情况说明： ■ 使用同样的id申请，Server将返回相同的设备证书 ■ 设备不具备唯一的标识，可传入空字符串，但请谨慎操作，可能会造成设备证书浪费
server_ip	char *	Server服务所在设备的IP地址

调用该函数的返回值说明如下。

■ 0：解析成功

■ 非0：无效数据

请您根据以下步骤基于示例代码开发Client。

1. 开发get_triple()函数。

   通过C SDK去获取某个已集成生活物联网SDK的设备证书为示例，请您根据以下步骤开发示例代码。

   a) 确保设备已经通过DHCP正确的获取了IP地址。

   b) 调用HAL_Wifi_Get_IP()获取设备的IP地址。

   c) 调用send_req()获取设备证书。

      若未获取到数据，会自动再请求一次（若仍然未获取到数据，后续业务逻辑您需根据业务自行实现）。

   d) 调用parse_data()解析数据。

      数据解析成功，将获得设备的ProductKey、ProductSecret、DeviceName，DeviceSecret。

      示例中仅对设备证书进行打印输出（以下代码中LOG中所示），您开发时需要将这些信息存储到Flash/NVRAM中。

```
static void get_triple(char* server_ip, char* productKey, char* deviceid){
    char ip_addr[16] = {0};
    char pk[PRODUCT_KEY_LEN] = {0};
    char dn[DEVICE_NAME_LEN] = {0};
    char ds[DEVICE_SECRET_LEN] = {0};
    char ps[PRODUCT_SECRET_LEN] = {0};
    char did[PRODUCT_SECRET_LEN] = {0};

    HAL_Wifi_Get_IP(ip_addr, 0);
    if (strlen(ip_addr) > 5){
        LOG("wifi_service_event ip=%s", ip_addr);
        char *response_payload = send_req(0, productKey, deviceid, server_ip);
        if (strlen(response_payload) == 0){
            if (response_payload != NULL)
                HAL_Free(response_payload);
            response_payload = send_req(0, productKey, deviceid, server_ip);
        }
        if( response_payload != NULL && 0 == parse_data(response_payload, pk, dn, ps
, ds, did)){
            HAL_Free(response_payload);
            LOG("triple:pk=%s dn=%s ds=%s ps=%s did=%s", pk,dn,ds,ps,did);
            // write pk,dn,ds,ps to falsh

            // send write success to server
            HAL_Free(send_req(1, productKey, did, server_ip));
        } else {
            if (response_payload != NULL)
                HAL_Free(response_payload);
        }
    }
```

   e) 再次调用send_req()，告知Server已成功保存设备证书。

2. 集成Client SDK示例代码。

以下集成基于已发布的生活物联网平台SDK。获取生活物联网平台SDK请参见SDK概述与开发环境设置。

以下基于living_platform为例开发。若您基于其他应用开发，需调整相应的路径。

a) 在Product/example/living_platfrom/living_platfrom.mk中，新增triple_burn.c文件。

b) 在app_entry.c文件中增加include triple_burn.h。

```
#if defined(OTA_ENABLED) && defined(BUILD_AOS)
#include "ota_service.h"
#endif

#include "triple_burn.h"  //triple_burn.h为头文件

#include <aos/network.h>
```

c) 将**get_triple**放在living_platform应用的app_entry.c文件中。

d) 参照以下示例代码，实现在handle_linkkey_cmd()中调用get_triple()来获取设备证书。

更多代码可以参见client\C-SDK\app_entry.c文件。

```
static void handle_linkkey_cmd(char *pwbuf, int blen, int argc, char **argv)
{
    if (argc == 1)
    {
        int len = 0;
        char product_key[PRODUCT_KEY_LEN + 1] = {0};
        char product_secret[PRODUCT_SECRET_LEN + 1] = {0};
        char device_name[DEVICE_NAME_LEN + 1] = {0};
        char device_secret[DEVICE_SECRET_LEN + 1] = { 0 };

        len = PRODUCT_KEY_LEN+1;
        aos_kv_get("linkkit_product_key", product_key, &len);

        len = PRODUCT_SECRET_LEN+1;
        aos_kv_get("linkkit_product_secret", product_secret, &len);

        len = DEVICE_NAME_LEN+1;
        aos_kv_get("linkkit_device_name", device_name, &len);

        len = DEVICE_SECRET_LEN+1;
        aos_kv_get("linkkit_device_secret", device_secret, &len);

        aos_cli_printf("Product Key=%s.\r\n", product_key);
        aos_cli_printf("Device Name=%s.\r\n", device_name);
        aos_cli_printf("Device Secret=%s.\r\n", device_secret);
        aos_cli_printf("Product Secret=%s.\r\n", product_secret);

    }
    else if (argc == 5)
    {
        aos_kv_set("linkkit_product_key", argv[1], strlen(argv[1]) + 1, 1);
        aos_kv_set("linkkit_device_name", argv[2], strlen(argv[2]) + 1, 1);
        aos_kv_set("linkkit_device_secret", argv[3], strlen(argv[3]) + 1, 1);
        aos_kv_set("linkkit_product_secret", argv[4], strlen(argv[4]) + 1, 1);

        aos_cli_printf("Done");
```

```
    }
    else if (argc == 4){
        get_triple(argv[1], argv[2], argv[3]);
    }
    else if (argc == 3){
        get_triple(argv[1], argv[2], "");
    }
    else
    {
        aos_cli_printf("Error: %d\r\n", __LINE__);
        return;
    }
}
```

e) 配置设备批量配网能力。

该能力可以让没有WiFi热点信息的WiFi设备，自动连接到厂测路由器，详细操作请参见配置SDK新增功能。

f) 编译固件并烧录到设备。

3. 调试示例代码。

设备烧写包含以上代码的固件后，在串口终端输入以下命令进行测试。

- 携带DeviceID申请设备证书

```
linkkey ServerIP productKey DID
//ServerIP是实际环境中Server的IP地址，productKey是产品的ProductKey的值
//DID是设备的唯一标识，唯一标识可以是设备MAC地址、序列号等可以唯一标识设备的值
```

- 不携带DeviceID申请设备证书（该情况请谨慎操作，可能会造成设备证书浪费）

```
linkkey ServerIP productKey
//ServerIP在运行时需要输入Server在实际环境中的IP地址，productKey是您产品的
ProductKey的值
```

若设备成功从Server取到数据，会得到类似以下的日志。

```
resp={"CRC32":3041429316,"ProductKey":"a1xxxxNW",
"DeviceName":"ZTMxxxxxTUy",
"DeviceSecret":"NmMzxxxxjE5",
"ProductSecret":"Xmxxxxb9R","Status":"OK"}.
```

2.5 Wi-Fi设备开发

2.5.1 Wi-Fi芯片移植

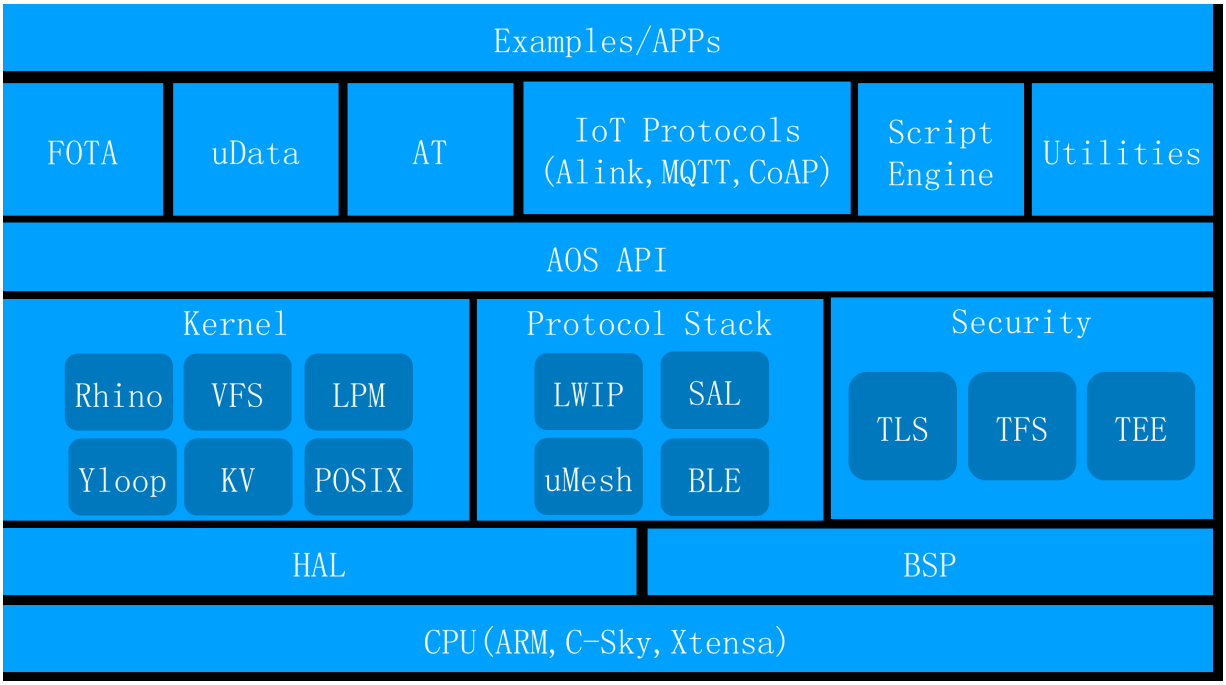
生活物联网平台SDK（含AliOS Things）是阿里云IoT针对生活物联网平台所提供的设备端SDK，其中包含AliOS Things物联网操作系统（基于AliOS Things V1.3.4）和Link Kit V2.3.0。生活物联网平台SDK含AliOS Things版本包含了无AliOS Things的版本，本文档以移植含AliOS Things版本为例，介绍Wi-Fi芯片如何适配生活物联网平台SDK。

架构和移植概要

如果Wi-Fi芯片还没有移植AliOS Things，请单击生活物联网平台SDK（含AliOS Things）下载代码。

AliOS Things的架构可适用于分层架构和组件化架构。从底部到顶部的内容如下。

- 板级支持包（BSP）：主要由SoC供应商开发和维护。
- 硬件抽象层（HAL）：例如Wi-Fi和UART。
- 内核：包括Rhino实时操作系统内核、Yloop、VFS、KV 存储。
- 协议栈：包括TCP/IP协议栈（LwIP）、uMesh网络协议栈。
- 安全：安全传输层协议（TLS）、可信服务框架（TFS）、可信运行环境（TEE）。
- AOS API：提供可供应用软件和中间件使用的API。
- 中间件：包括常见的物联网组件和阿里巴巴增值服务中间件。
- 示例应用：阿里云自主开发的示例代码，以及通过了完备测试的应用程序（例如Link Kit App）。



如上图所示，AliOS Things是一个分层加组件的架构，可以较简单的移植到Wi-Fi芯片上。在Wi-Fi芯片上移植AliOS Things的主要工作包括：

- 内核移植
- 硬件抽象层HAL移植
- Wi-Fi HAL和配网移植
- LwIP协议栈移植
- OTA移植

内核移植

- 概要

  AliOS Things中使用的内核为Rhino。关于Rhino内核移植的介绍，请参见Rhino内核移植。

- 系统移植

  关于Rhino内核移植的具体内容，请参见Rhino系统移植。

- 移植示例

  本章节介绍如何将Rhino最小系统移植到STM32平台。

  - 目标开发板：STM32L496G-Discovery，Cotex-m4架构。
  - 移植目标：基本任务运行，tick时钟实现krhino_task_sleep，基本串口打印。

  具体移植步骤，请参见Rhino基于Keil最小内核移植示例。

- 验收测试

  系统移植完成后，验收方法如下。

  - 系统能启动一个任务并调用krhino_task_sleep函数实现定时打印，例如每秒打印一次日志。
  - 运行Helloworld程序。

HAL移植

- 概要

  硬件抽象层HAL抽象层普遍存在于各个操作系统之中，最主要的目的是为了屏蔽不同芯片平台的差异，从而使上面的应用软件不会随芯片而改变。目前AliOS Things定义了全面的HAL抽象层，只要对接相应的HAL接口就能控制芯片的控制器，从而达到控制硬件外设的目的。

- 硬件抽象层移植

  AliOS Things中定义的硬件HAL包括：

  - ADC
  - GPIO
  - I2C
  - NAND/NOR
  - PWM
  - RNG
  - RTC
  - SD
  - SPI
  - Timer
  - UART
  - Watchdog
  - DAC

  关于AliOS Things硬件HAL移植的具体内容，请参见硬件抽象层移植。

- Flash HAL及KV移植

  AliOS Things中设计了一套简单使用的永久存储机制KV（Key-Value），该机制依赖Flash HAL。因此平台移植时，需要对Flash HAL进行移植，以实现KV功能。

  关于Flash HAL和KV移植的具体细节，请参见Flash和KV移植。

- 参考实现

  以RDA5981平台为例，硬件抽象相关的移植代码请参见RDA5981 HAL参考实现。

Wi-Fi和配网移植

- 概要

  对于Wi-Fi芯片，AliOS Things的Wi-Fi配网功能需要Wi-Fi HAL的支持。平台对接过程中，需要对这些Wi-Fi HAL接口进行移植。

  当Wi-Fi和BLE Combo芯片需要蓝牙辅助配网能力的时候，除了对接Wi-Fi HAL之外，请参见蓝牙对接。

- Wi-Fi HAL移植

  Wi-Fi HAL相关的数据结构和接口定义请参见wifi.h。

- Wi-Fi模块结构体

  在AliOS Things中与Wi-Fi HAL相关联的结构体为 - hal_wifi_module_t。Wi-Fi相关的操作和接口都封装在hal_wifi_module_t结构体中，Wi-Fi的相关定义请参见wifi.h。

```
struct hal_wifi_module_s {
    hal_module_base_t    base;
    const hal_wifi_event_cb_t *ev_cb;

    int  (*init)(hal_wifi_module_t *m);
    void (*get_mac_addr)(hal_wifi_module_t *m, uint8_t *mac);
    void (*set_mac_addr)(hal_wifi_module_t *m, const uint8_t *mac);
    int  (*start)(hal_wifi_module_t *m, hal_wifi_init_type_t *init_para);
    int  (*start_adv)(hal_wifi_module_t *m, hal_wifi_init_type_adv_t *init_para_adv);
    int  (*get_ip_stat)(hal_wifi_module_t *m, hal_wifi_ip_stat_t *out_net_para, hal_wifi_t
ype_t wifi_type);
    int  (*get_link_stat)(hal_wifi_module_t *m, hal_wifi_link_stat_t *out_stat);
    void (*start_scan)(hal_wifi_module_t *m);
    void (*start_scan_adv)(hal_wifi_module_t *m);
    int  (*power_off)(hal_wifi_module_t *m);
    int  (*power_on)(hal_wifi_module_t *m);
    int  (*suspend)(hal_wifi_module_t *m);
    int  (*suspend_station)(hal_wifi_module_t *m);
    int  (*suspend_soft_ap)(hal_wifi_module_t *m);
    int  (*set_channel)(hal_wifi_module_t *m, int ch);
    void (*start_monitor)(hal_wifi_module_t *m);
    void (*stop_monitor)(hal_wifi_module_t *m);
    void (*register_monitor_cb)(hal_wifi_module_t *m, monitor_data_cb_t fn);
    void (*register_wlan_mgnt_monitor_cb)(hal_wifi_module_t *m, monitor_data_cb_t
fn);
    int  (*wlan_send_80211_raw_frame)(hal_wifi_module_t *m, uint8_t *buf, int len);
};
```

  在具体的平台移植过程中，用户需要分别实现Wi-Fi模块结构体中对应的接口函数。

  AliOS Things对Wi-Fi HAL层接口有一层封装，参见kernel/hal/wifi.c文件。具体的接口实现，一般在platform/mcu/xxx/hal/wifi_port.c中。参考实现请查看platform/mcu/rda5981x/hal/wifi_port.c文件。Wi-Fi HAL接口的说明，请参见Wi-Fi HAL。

- Wi-Fi接口说明

  - init

    该接口需要对Wi-Fi进行初始化，使Wi-Fi达到可以准备进行连接工作的状态，如分配Wi-Fi资源、初始化硬件模块等操作。

  - get_mac_addr

    通过该接口可以获取到Wi-Fi的物理硬件地址。

    说明:

回传的MAC地址格式为6个字节（不含英文逗号）二进制值，例如 uint8_t mac[6] = {0xd8, 0x96,0xe0,0x03,0x04,0x01};。

- set_mac_addr

  通过该接口可以设置Wi-Fi的物理硬件地址。

- start

  通过该接口可以启动Wi-Fi，根据启动参数不同来区分启动station模式还是AP模式。

  ■ station模式：连接AP

  在station模式下，该函数触发AP连接操作后即返回。后续底层处理过程中，拿到IP信息后，需要调用ip_got回调函数来通知上层获取IP事件。

  ■ AP模式：根据参数启动AP功能

  注意：

  ■ station模式下启动Wi-Fi连接时，传入的SSID长度不超过32位。

  ■ 在连接AP后，Wi-Fi底层需要维护自动重连操作。

- start_adv

  该接口类似start，但启动的参数更丰富，为可选接口。

- get_ip_stat

  通过该接口可以获取Wi-Fi工作状态下的IP信息，如IP、网关、子网掩码、MAC地址等信息。

- get_link_stat

  通过该接口可以获取Wi-Fi工作状态下的链路层信息，如连接信号强度、信道、SSID等信息。

- start_scan

  该接口启动station模式下的信道扫描。扫描结束后，调用scan_compeleted回调函数，将各个信道上扫描到的AP信息通知给上层。需要得到的扫描信息在hal_wifi_scan_result_t中定义。

  说明：

扫描结果存储所需要的内存在底层实现中分配，回调函数返回后再将该内存释放。

- start_scan_adv

  该接口与hal_wifi_start_scan类似，但扫描的信息更多，例如**bssid**、**channel**信息等，需要扫描得到的信息在hal_wifi_scan_result_adv_t中定义。扫描结束后，通过调用scan_adv_compeleted回调函数通知上层。

  说明:
  扫描结果存储所需要的内存在底层实现中分配，回调函数返回后再将该内存释放。

- power_off

  该接口对Wi-Fi硬件进行断电操作。

- power_on

  该接口对Wi-Fi硬件进行上电操作。

- suspend

  该接口断开Wi-Fi所有连接，同时支持station模式和soft ap模式。

- suspend_station

  该接口断开Wi-Fi所有连接，支持station模式。

- suspend_soft_ap

  该接口断开Wi-Fi所有连接，支持soft ap模式。

- set_channel

  通过该接口可以设置信道。

- wifi_monitor

  该接口启动监听模式，并且在收到任何数据帧（包括beacon、probe request等）时，调用monitor_cb回调函数进行处理。

  说明:

回调函数是上层通过register_monitor_cb注册的。监听模式下，上层cb函数期望处理的包不带FCS域，所以底层的数据包如果带FCS应当先剥离再往上层传递。

- stop_wifi_monitor

  该接口关闭侦听模式。

- register_monitor_cb

  该接口注册侦听模式回调函数，回调函数在底层接收到任何数据帧时被调用。

- register_wlan_mgnt_monitor_cb

  该接口注册管理帧回调函数（非监听模式下），该回调函数在底层接收到管理帧时被调用。

- start_debug_mode

  该接口进入调试模式，可选（若模块支持）。

- stop_debug_mode

  该接口退出调试模式，可选。

- wlan_send_80211_raw_frame

  该接口可以用于发送802.11格式的数据帧。

  参考RDA5981平台实现：platform/mcu/rda5981x/hal/wifi_port.c。

- Wi-Fi事件回调

  Wi-Fi移植过程中，Wi-Fi事件及回调函数也是很重要的一个内容。AliOS Things中Wi-Fi事件的回调函数在netmgr模块中定义，请参见network/netmgr/目录。在配网过程中，netmgr负责定义和注册Wi-Fi事件回调函数。

  而在Wi-Fi启动和运行的过程中，通过调用回调函数来通知上层应用，以执行相应的动作。这些Wi-Fi事件的回调函数，应该在Wi-Fi网络驱动（通常是HAL层实现或更底层的代码）的任务上下文中被触发，示例如下。

  - 在Wi-Fi底层拿到IP后，应执行ip_got回调，并将IP信息传递给上。
  - 在Wi-Fi完成信道扫描后，应调用scan_compeleted或scan_adv_compeleted回调，将扫描结果传递给上层。
  - 在Wi-Fi状态改变时，应调用stat_chg回调。

  说明：
  这些事件回调函数由netmgr配网模块定义并注册，在Wi-Fi底层（如HAL）里面触发调用。

  下面是AliOS Things中定义的Wi-Fi事件回调函数和接口，相关定义请参见wifi.h。

```
typedef struct {
    void (*connect_fail)(hal_wifi_module_t *m, int err, void *arg);
```

```
    void (*ip_got)(hal_wifi_module_t *m, hal_wifi_ip_stat_t *pnet, void *arg);
    void (*stat_chg)(hal_wifi_module_t *m, hal_wifi_event_t stat, void *arg);
    void (*scan_compeleted)(hal_wifi_module_t *m, hal_wifi_scan_result_t *result,
                            void *arg);
    void (*scan_adv_compeleted)(hal_wifi_module_t *m,
                                hal_wifi_scan_result_adv_t *result, void *arg);
    void (*para_chg)(hal_wifi_module_t *m, hal_wifi_ap_info_adv_t *ap_info,
                     char *key, int key_len, void *arg);
    void (*fatal_err)(hal_wifi_module_t *m, void *arg);
} hal_wifi_event_cb_t;
```

回调函数的定义，请参照**netmgr**中的g_wifi_hal_event。

- Wi-Fi模块注册和初始化

  在完成Wi-Fi接口和事件回调的实现后，定义一个hal_wifi_module_t的结构体，将各个接口和回调的实现地址赋值给结构体中对应的域，示例如下。

```
hal_wifi_module_t sim_aos_wifi_vendor = {
    .base.name          = "alios_wifi_vender_name",
    .init               =  wifi_init,
    .get_mac_addr       =  wifi_get_mac_addr,
    .start              =  wifi_start,
    .start_adv          =  wifi_start_adv,
    .get_ip_stat        =  get_ip_stat,
    .get_link_stat      =  get_link_stat,
    .start_scan         =  start_scan,
    .start_scan_adv     =  start_scan_adv,
    .power_off          =  power_off,
    .power_on           =  power_on,
    .suspend            =  suspend,
    .suspend_station    =  suspend_station,
    .suspend_soft_ap    =  suspend_soft_ap,
    .set_channel        =  set_channel,
    .start_monitor      =  start_monitor,
    .stop_monitor       =  stop_monitor,
    .register_monitor_cb =  register_monitor_cb,
    .register_wlan_mgnt_monitor_cb = register_wlan_mgnt_monitor_cb,
    .wlan_send_80211_raw_frame = wlan_send_80211_raw_frame,
};
```

一般在板级初始化的过程中，先通过hal_wifi_register_module接口对Wi-Fi模块进行注册，再调用hal_wifi_init接口对Wi-Fi硬件模块进行初始化。Wi-Fi HAL模块完成初始化后才可以使用。

```
void hal_wifi_register_module(hal_wifi_module_t *m);
int hal_wifi_init(void);
```

参考实现：platform/mcu/rda5981x/bsp/entry.c。

- 参考实现

  以RDA5981平台为例：

  - Wi-Fi HAL接口实现代码位于platform/mcu/rda5981x/hal/wifi_port.c。
  - Wi-Fi芯片注册和初始化代码位于platform/mcu/rda5981x/bsp/entry.c。

- 验收测试

  在完成W-iFi的移植后，可以通过Wi-Fi APP来验证移植工作的有效性。Wi-Fi App位于app/example/wifihalapp/目录下，验证步骤如下。

  1. 编译和运行Wi-Fi HAL App。
  2. 在CLI输入testwifihal all <AP SSID> <AP password>命令。

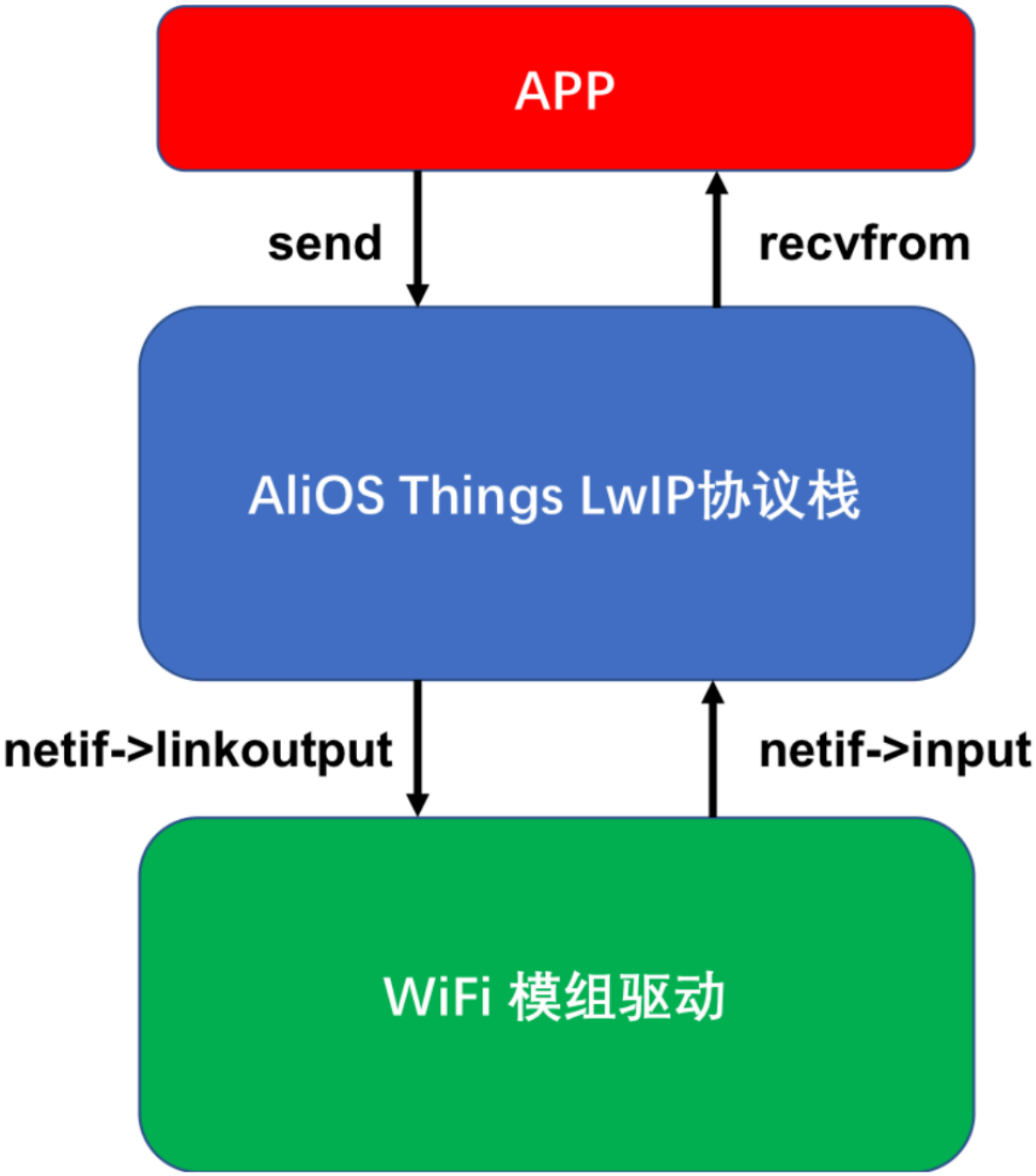
     正常情况下所有测试子项都通过。

     说明:
     若失败，可在Log中搜索failed关键字，查看具体失败的测试子项。

LwIP移植

Wi-Fi芯片上使用提供的LwIP协议栈需要如下五步操作。其中，与OS对接默认是已经完成的，无须额外操作。

1. 对接网卡与驱动。



对接网卡与驱动需要完成底层初始化、输出对接、输入对接。代码示例请参见network/lwip/netif/ethernet.c。主要涉及到以下函数的修改。

```
static void low_level_init(struct netif *netif);
static err_t low_level_output(struct netif *netif, struct pbuf *p);
static struct pbuf *low_level_input(struct netif *netif);
```

函数名	作用	描述
low_level_init	底层初始化	网卡（netif）信息填充，例如MAC地址、MTU大小、flag标示设置等

函数名	作用	描述
low_level_output	输出对接	调用驱动层发送函数；low_level_output函数指针将赋给netif->linkoutput
low_level_input	输入对接	该函数调用将被Wi-Fi芯片驱动函数调用，用于向上交付数据包。

修改完成后，源代码需要存放在对应的平台目录platform/mcu/xxxx下。

以某一设备为例，若其相关修改项在platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/port/ethernetif.c目录中，low_level_init主要工作是填充网卡（netif）信息，例如设置MAC地址、MTU大小、flag标识等。

```
static void low_level_init(struct netif *netif)
{
   u8_t wireless_mac[NETIF_MAX_HWADDR_LEN];

   wifi_get_mac_address((char *)wireless_mac);
   /* set MAC hardware address length */
   netif->hwaddr_len = ETHARP_HWADDR_LEN;
   os_memcpy(netif->hwaddr, wireless_mac, ETHARP_HWADDR_LEN);
   /* maximum transfer unit */
   netif->mtu = 1500;
   /* device capabilities */
   /* don't set NETIF_FLAG_ETHARP if this device is not an ethernet one */
   netif->flags = NETIF_FLAG_BROADCAST | NETIF_FLAG_ETHARP | NETIF_FLAG_LINK_UP |
NETIF_FLAG_IGMP;
   ETH_INTF_PRT("leave low level!\r\n");
}
```

low_level_output调用驱动层发送函数。

```
static err_t low_level_output(struct netif *netif, struct pbuf *p)
{
   int ret;

   ret = bmsg_tx_sender(p);

   if (ret == 0)
      return ERR_OK;
   else
      return  ERR_TIMEOUT;
}
```

未实现low_level_input，而是直接在驱动函数里调用netif->linkinput。

```
void ethernetif_input(int iface, struct pbuf *p)
{
   ...
netif = (struct netif *)net_get_netif_handle();
   ...

switch (htons(ethhdr->type))
   {
   ...
   case ETHTYPE_IP:
```

```
        /* full packet send to tcpip_thread to process */
        if (netif->input(p, netif) != ERR_OK)    // ethernet_input
        {
          LWIP_DEBUGF(NETIF_DEBUG, ("ethernetif_input: IP input error\r\n"));
          pbuf_free(p);
          p = NULL;
        }
      break;
    ...
}
```

2. 移植平台相关内容。

   平台相关的移植工作，主要定义包括类型定义，大小端设置，内存对齐等，如表所示。

平台相关定义	示例
类型定义	typedef unsigned char u8_t;
大小端设置	#define BYTE_ORDER LITTLE_ENDIAN
内存对齐	#define PACKSTRUCTSTRUCT__attribute((packed))

   如果参考实现与开发者实现一致，可以直接拷贝存放在对应的平台platform下面。

   以某一设备为例，若其相关修改项在platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/port/cc.h目录中。

```
/*
 *  Typedefs for the types used by lwip -
 *  u8_t, s8_t, u16_t, s16_t, u32_t, s32_t, mem_ptr_t
 */
typedef unsigned   char    u8_t;     /* Unsigned 8 bit quantity       */
typedef signed     char    s8_t;     /* Signed   8 bit quantity       */
typedef unsigned   short   u16_t;    /* Unsigned 16 bit quantity      */
typedef signed     short   s16_t;    /* Signed   16 bit quantity      */
typedef unsigned   long    u32_t;    /* Unsigned 32 bit quantity      */
typedef signed     long    s32_t;    /* Signed   32 bit quantity      */
...
```

3. 定制协议栈配置。

   LwIP配置一般放在lwipopts.h，该头文件放置在平台相关目录platform/mcu/xxxx下。

   以某一设备为例，若相关修改项在platform/mcu/moc108/mx108/mx378/func/mxchip/lwip-2.0.2/lwipopts.h目录中。

```
#define IP_DEBUG                LWIP_DBG_OFF
#define ETHARP_DEBUG              LWIP_DBG_OFF
#define NETIF_DEBUG             LWIP_DBG_OFF
#define PBUF_DEBUG              LWIP_DBG_OFF
#define MEMP_DEBUG               LWIP_DBG_OFF
#define API_LIB_DEBUG           LWIP_DBG_OFF
#define API_MSG_DEBUG             LWIP_DBG_OFF
#define ICMP_DEBUG              LWIP_DBG_OFF
#define IGMP_DEBUG              LWIP_DBG_OFF
```

```
#define INET_DEBUG                LWIP_DBG_OFF
```

4. 与OS对接。

   LwIP与OS的对接AliOS Things已经默认完成，开发者可以直接使用。

5. 编译配置。

   以某一设备为例，若其相关修改项在platform/mcu/moc108/aos.mk目录中。

```
GLOBAL_INCLUDES += mx108/mx378/func/mxchip/lwip-2.0.2/port

$(NAME)_SOURCES += mx108/mx378/func/mxchip/lwip-2.0.2/port/ethernetif.c  \
                   mx108/mx378/func/mxchip/lwip-2.0.2/port/net.c
```

OTA移植

- 概要

  OTA升级作为物联网设备的一项基础功能，可以快速修复软件漏洞、更新系统，对于快速迭代的物联网产品是刚性需求。目前IOT设备种类繁多，但并未提供统一的OTA升级方案，针对日益发展的IOT设备，开发者迫切需要一套云端一体化的OTA升级方案来满足快速迭代的产品开发周期，同时降低产品开发和部署的成本。

  对于接入生活物联网平台的物联网设备，阿里巴巴云端和设备端OTA组件为用户提供云端一体化的OTA升级服务。对于使用OTA升级服务的用户只需按照此文档实现移植层接口后就可以轻松使用阿里巴巴物联网OTA升级服务。

- 平台移植

  OTA相关的移植详情，请参见OTA移植文档。

- 参考实现

  以RTL8710平台为例，OTA相关的平台移植实现代码位于platform/mcu/rtl8710bn/hal/ota_port.c目录下。

- 移植验证

  请参见OTA移植demo。

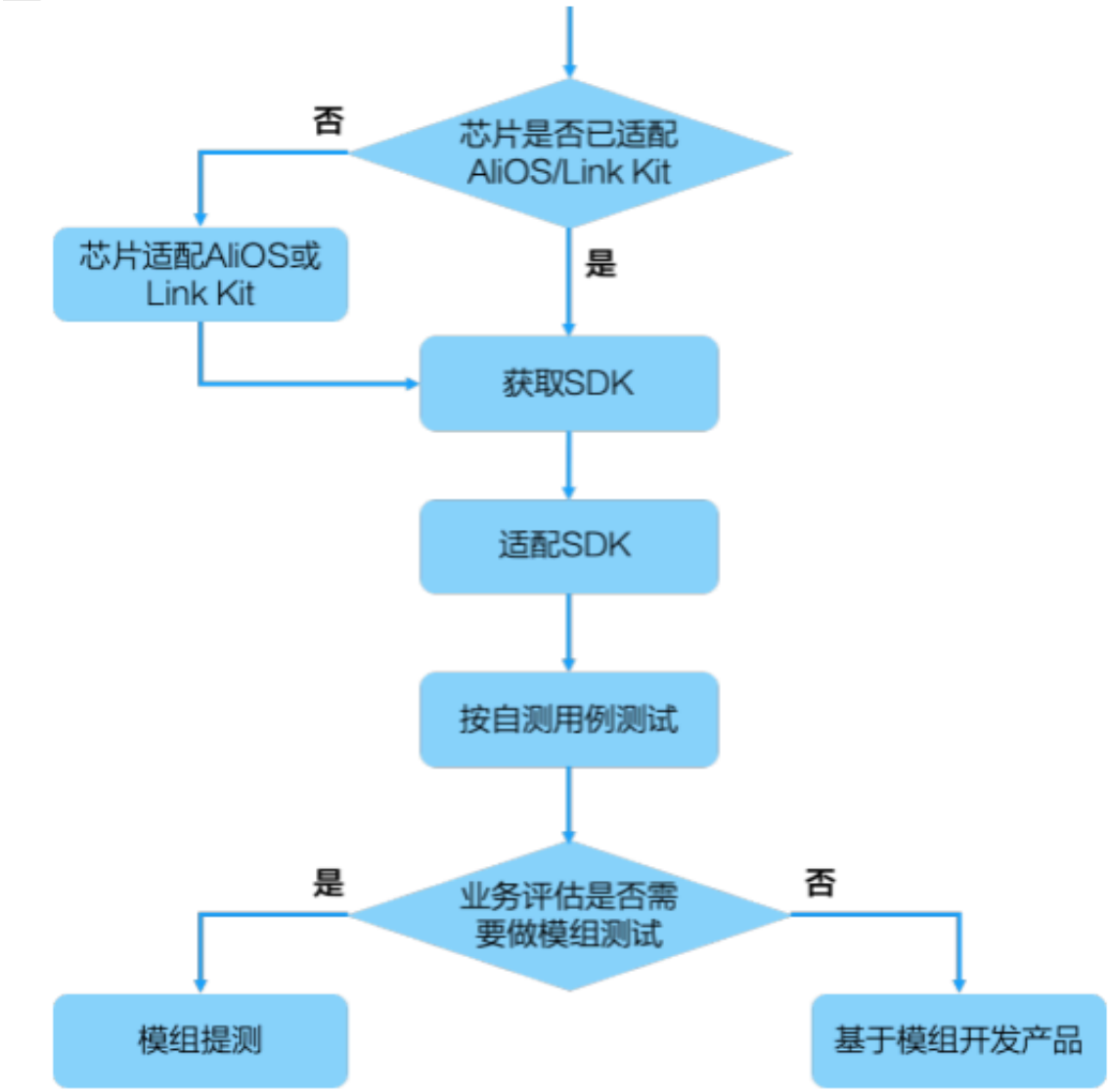
2.5.2 Wi-Fi模组移植

介绍Wi-Fi模组如何适配生活物联网平台SDK。

概述

生活物联网平台SDK是阿里云IoT针对生活物联网平台所提供的设备端SDK，基于C语言，分为包含AliOS Things物联网操作系统和无AliOS Things物联网操作系统两个版本。

基本流程如下图所示。



芯片适配

在适配生活物联网平台SDK前，请确认模组使用的Wi-Fi芯片是否已经支持AliOS或者Link Kit。

生活物联网平台SDK已支持的Wi-Fi芯片如下表所示。

芯片厂商	芯片型号
展锐	RDA5981X
翱捷科技	ASR5501
博芯科技	BK7231
	BK7231S
	BK7231U
南方硅谷	SV6166F

芯片厂商	芯片型号
	SV6166
	SV6266
Realtek	RTL8710BN
庆科	MOC108
	MX1101

如果模组使用的Wi-Fi芯片尚未支持AliOS或者Link Kit，可以推动芯片原厂移植生活物联网平台SDK。请联系：feiyan_fw_group@alibabacloud.com

模组移植

- SDK获取与开发环境设置

  请参见SDK概述与开发环境设置。
- 切换生活物联网平台SDK
  - 模组已适配AliOS

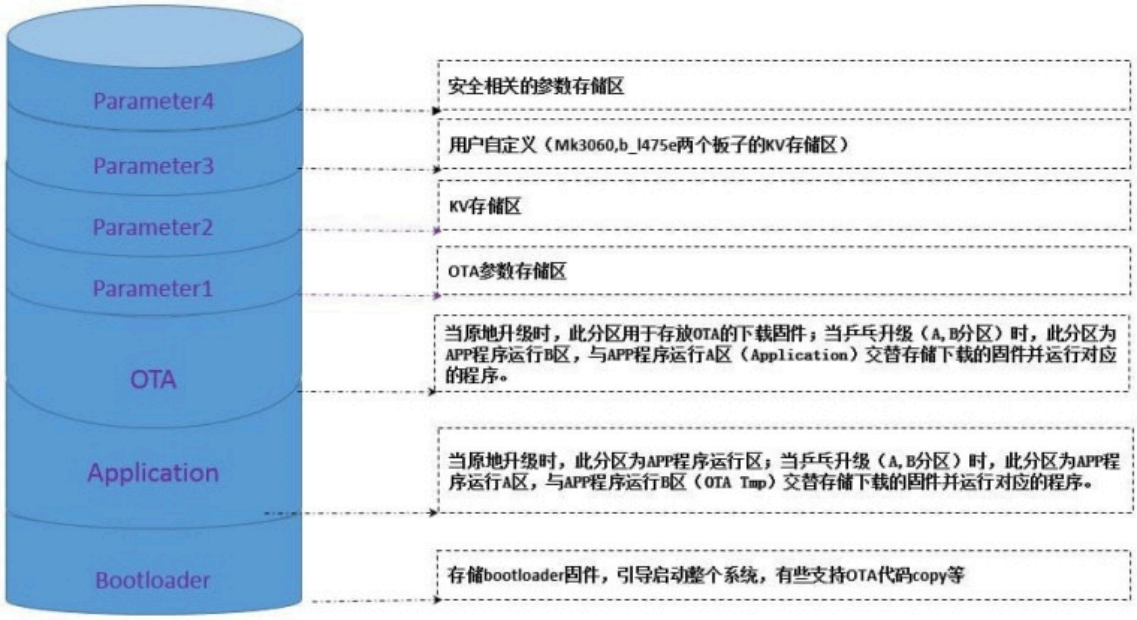
    此情况建议以包含AliOS Things的生活物联网平台SDK为基础，将该模组对应的AliOS Things代码中platform目录下对应芯片型号的代码和board目录下对应模组的代码合入生活物联网平台SDK。
  - 模组基于其他操作系统

    如模组基于FreeRTOS或者其他操作系统，建议将无AliOS Things的生活物联网平台SDK合入模组现有工程中。

- Bootloader/Flash分区

  Bootloader和Flash分区是board目录下对应模组文件夹的重要部分。

  AliOS Things 由于功能需要，维护了一张flash分区表，这张表包括bootloader区、Application区、OTA TMP区以及parameters区，如下图所示。



  分区大小划分原则如下。

  - 获取芯片平台的Flash大小。
  - 获取bootloader信息，包括：bootloader支持的升级类型（原地还是乒乓）、bootloader跳转地址（如果是乒乓会有两个跳转地址）。
  - 跟据bootloader获取的信息，划分整个Flash。

  根据以上的划分原则，以HF-LPT230模组为例说明其Flash分区。其芯片RDA5981A，Flash size为1Mbytes。分区表数组hal_logic_partition_t hal_partitions[] 定义在board/hf-lpt230/board.c 文件中。

分区	地址范围	大小	备注
Bootloader	0x18001000 ~ 0x18004000	12Kbytes	存储bootloader固件
Application	0x18004000 ~ 0x18095000	580Kbytes	应用代码分区
OTA Storage	0x18095000 ~ 0x180F7000	372Kbytes	OTA代码分区

分区	地址范围	大小	备注
PARAMETER1	0x180F7000 ~ 0x180F8000	4Kbytes	OTA参数存储区
PARAMETER2	0x180F8000 ~ 0x180FA000	8Kbytes	KV存储区
PARAMETER3	0x180FA000 ~ 0x180FB000	4Kbytes	用户自定义
PARAMETER4	0x180FB000 ~ 0x180FC000	4Kbytes	安全相关的参数存储区（如ID2秘钥）
SYS RF Data	0x180FC000 ~ 0x180FD000	4Kbytes	Wi-Fi模组RF参数
HFILOP	0x180FD000 ~ 0x180FF000	8Kbytes	模组商用于存放三元组和MAC地址

模组可以结合自身Flash大小、Bootloader实现等调整Flash分区。

模组自测

模组完成SDK移植后，请参见生活物联网平台平台模组厂家自测用例集 文档规定的自测用例完成模组自测。

自测主要从以下几个方面验证模组功能和稳定性。

- 配网
- 设备控制（云端控制和本地控制）
- 通道稳定性
- 固件升级

自测用例中通道稳定性测试高频压测部分，目前需要联系阿里小二，使用通道压测工具向设备发送下行业务指令（5pps）。联系方式为feiyan_fw_group@alibabacloud.com。

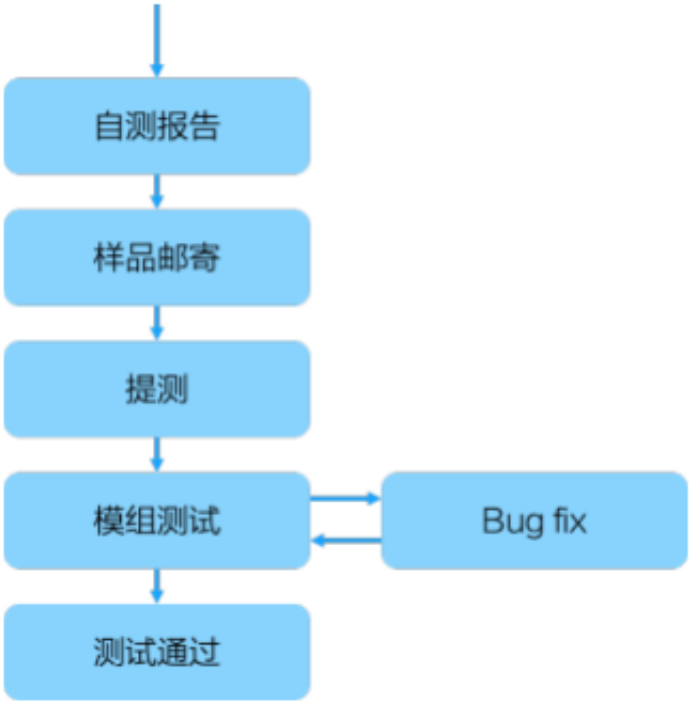
模组提测

模组测试并非强制，完成模组移植生活物联网平台SDK后，即可基于模组开发产品。是否需要做模组测试，请根据实际业务需要，先与阿里云IoT智能生活业务团队或者阿里天猫精灵业务团队确认。

如需要做模组测试（提测窗口为：feiyan_certification@alibabacloud.com），需要提供以下内容。

- 代码库位置（提供board文件夹代码位置）
- 自测报告
- 模组开发板 8块 + 配套供电线 + 串口线（用于获取设备端日志）+ 天线等配件

测试流程如下。



常见问题

Q：生活物联网平台SDK与AliOS和Link Kit是什么关系?

A：生活物联网平台SDK是基于AliOS v1.3.4和Link Kit v2.3.0的适合量产的稳定版本，优化了海外连接，并针对生活物联网平台业务进行了定制。

Q：我的模组已经适配过AliOS或者Link Kit别的版本，是否还需要切换生活物联网平台SDK?

A：生活物联网平台SDK针对配网成功率、稳定性、海外连接、生活物联网平台业务定制等方面进行了优化，对于量产产品，特别是要在海外大量出货的产品，需要切换到生活物联网平台SDK，否则可能会出现配网、连云、稳定性、兼容性等问题。

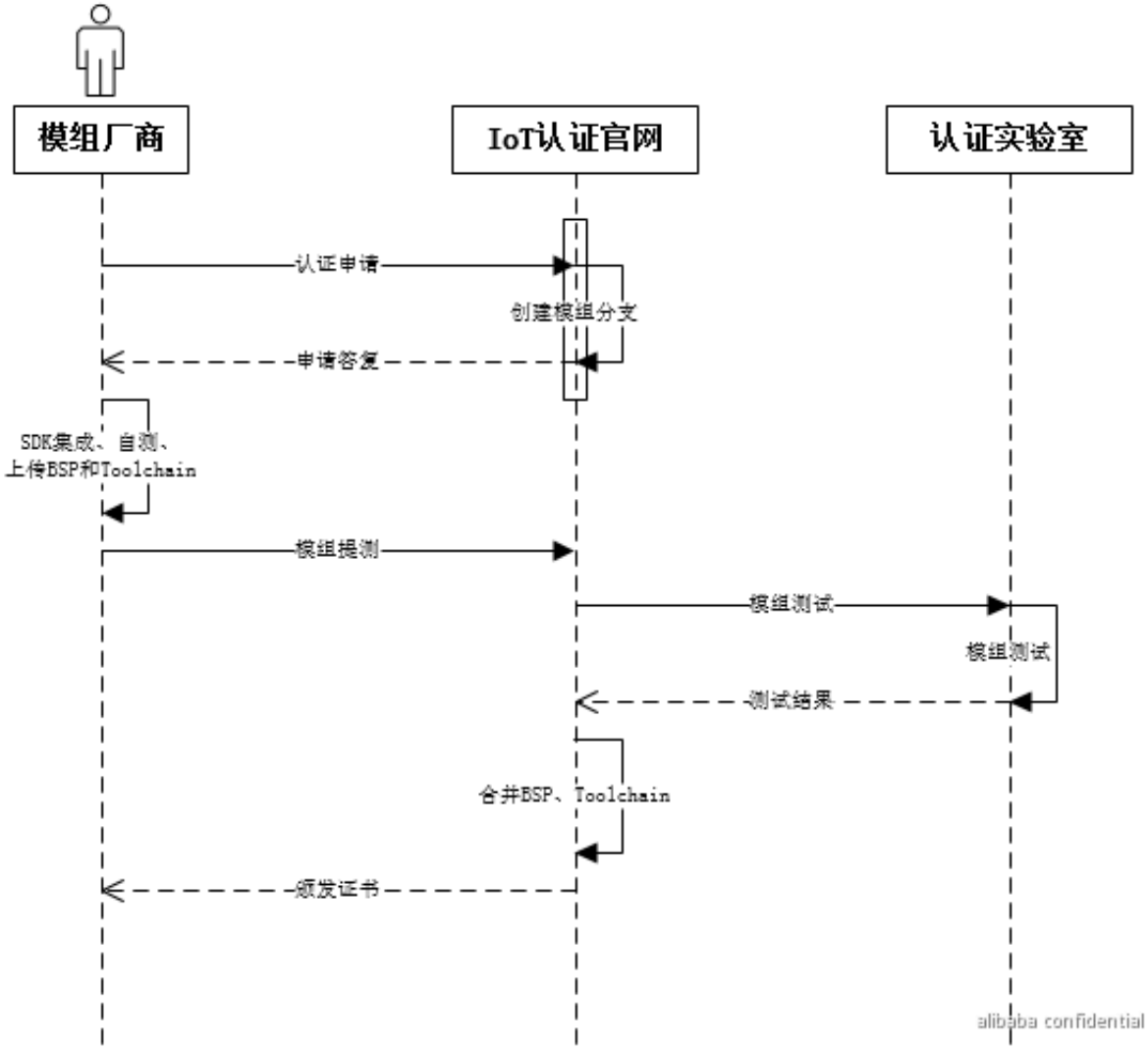
Q：模组如何开始配网绑定?

A：可以编译linkkitapp（编译命令`aos make clean; aos make linkkitapp@yourboardname`），启动后依次执行**awss**命令和**active_awss**命令，设备会进入配网绑定流程。

2.5.3 Wi-Fi模组认证指导

模组认证非强制，目前主要由阿里云IoT智能生活业务团队或者阿里天猫精灵业务团队，根据项目需要邀请进行。一般情况下，模组移植生活物联网平台SDK完成后，即可基于模组开发产品。如您收到明确的模组认证的需求，可以根据本文档完成Wi-Fi模组认证。

模组认证的流程图如下。



- 模组厂商：申请进行Wi-Fi模组认证，希望模组具备接入生活物联网平台能力的厂商。
- IoT认证官网：接受模组厂商认证申请的阿里云IoT认证服务。
- 认证实验室：实际执行模组认证的第三方认证机构。

下面从三种角色的角度，分别描述各自在模组认证流程中的操作。

模组厂商

当模组厂商希望其模组成为生活物联网平台的认证模组时，向阿里云IoT认证服务官网提出模组认证申请。

1. 在阿里云IoT认证服务官网，提交模组认证申请。

   提交申请时需要包含以下内容。

申请包含内容	描述
厂商名称	厂商的完整公司名称
厂商地址	厂商公司所在的地址
厂商简介	厂商公司的简介，如公司主营业务、规模以及在行业内的位置等信息
联系人	认证时与阿里云IoT沟通的联系人
联系方式	包含联系人的电话、邮件地址
模组型号	模组的正式型号
模组说明	模组用途、功能说明
Wi-Fi芯片型号	Wi-Fi芯片的生产厂商及芯片
模组图片	模组的外观图片
售卖链接	模组的客户可以购买该模组的链接
认证实验室列表	与阿里云IoT合作的认证实验室的列表、说明以及联系方式
Username的参数值	访问 https://code.aliyun.com/profile查看**Username**的参数值，主要用于登录代码仓库，提交BSP代码等

2. 等待认证申请回复。

   认证申请的回复中包含以下内容：

   - 模组认证流程说明
   - 模组厂商用于上传BSP、Toolchain的代码仓库的地址，以及模组厂商用于上传代码的身份信息
   - 模组移植开发指南（含自测说明）
   - 认证实验室列表说明，含认证实验室的介绍及联系方式

3. 按照“认证申请回复”中的模组移植开发指南，完成模组集成生活物联网SDK移植工作。

   如果在模组移植中碰到问题，可以按照“认证申请回复”中提供的技术支持方式进行问题定位和处理。

4. 按照“认证申请回复”中的模组自测内容，测试模组。

5. 模组自测成功后，将模组的BSP、Toolchain上传到“认证申请回复”中指定的位置。

6. 在“认证申请回复”中，选择任一家阿里云IoT提供的认证实验室，并提交模组测试。

   模组提交测试时需包含以下内容：

   - 模组硬件
   - BSP、Toolchain所在位置
   - 模组固件编译、烧写方法
   - 模组串口连接、控制按钮、LED指示说明

7. （可选）在阿里云IoT认证服务官网查询认证进展。
8. 待认证试验室测试通过后，在阿里云IoT认证服务官网获取认证证书。

IoT认证官网

阿里云IoT认证服务是阿里云IoT提供的面向Wi-Fi模组商的一套流程管理服务，用于向模组厂商展示相关的流程、文档、并提供操作页面等。IoT认证服务主要提供以下服务项。

- 处理模组厂商发送的认证申请
  - 记录模组厂商提供的信息。
  - 为厂商的模组创建一个单独的代码上传分支，供模组厂商上传BSP、Toolchain。
  - 回复模组厂商递交的认证申请。
- 处理模组厂商发送的模组提测
  - 验证厂商是否已经按照要求上传BSP、Toolchain、自测结果说明。
  - 模组厂商是否已经选定某个认证实验室进行测试。
  - 向指定认证实验室转交认证申请、客户提供的各种信息和硬件。
- 处理认证实验室发送的模组测试结果
  - 留存模组厂商的BSP、Toolchain
  - 为模组厂商生成认证证书

认证实验室

认证实验室是执行模组认证的第三方认证机构，当收到阿里云IoT的模组测试通知后，按以下流程执行。

1. 根据模组模组厂商的BSP、Toolchain，编译模组固件。
2. 依据阿里云IoT提供的测试案例，测试模组。
3. （可选）测试过程中发现Bug时，向模组厂商反馈。当模组厂商将修复Bug的代码提交到代码仓库后，重新编译固件并回归测试模组。
4. 将测试结果反馈给阿里云IoT认证服务，包含实际测试中的实际测试结果和数据。

2.5.4 基于已认证的模组开发

本文讲解设备开发者如何通过阿里云IoT已认证的Wi-Fi模组上开发产品功能，并将设备连接到生活物联网平台。

前提条件

已完成开发环境的安装，请参见SDK概述与开发环境设置。

获取SDK代码及编译

1. 获取SDK代码，请参见生活物联网平台SDK。
2. 解压下载的zip包。
3. 验证aos是否可以正常编译。

   以在某一模组上编译一个living_platform程序为例。

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

   以模组名为mk3080为例，完成编译后，会在out\living_platform@mk3080\binary\目录下生成living_platform@mk3080.all.bin文件，该文件即为需要烧写的文件，用户可以将其烧写到模组上查看程序是否可以正常运行。

   > **说明:**
   > 可以在board目录，查看已支持的模组。

烧写固件到模组

在编译得到固件后，即可以将固件烧写到模组中执行。不同的模组烧写过程不一样，请联系模组厂商获取烧写工具以及烧写说明。

在生活物联网平台定义产品

1. 创建产品。

   参见#unique_19，并按如下要求设置参数。

   - 选择**节点类型**为**设备**
   - 选择**是否接入网关**为**否**
   - 选择**连网方式**为**WiFi**
   - 选择**数据格式**为**ICA标准数据格式（Alink JSON）**

2. 产品功能定义。

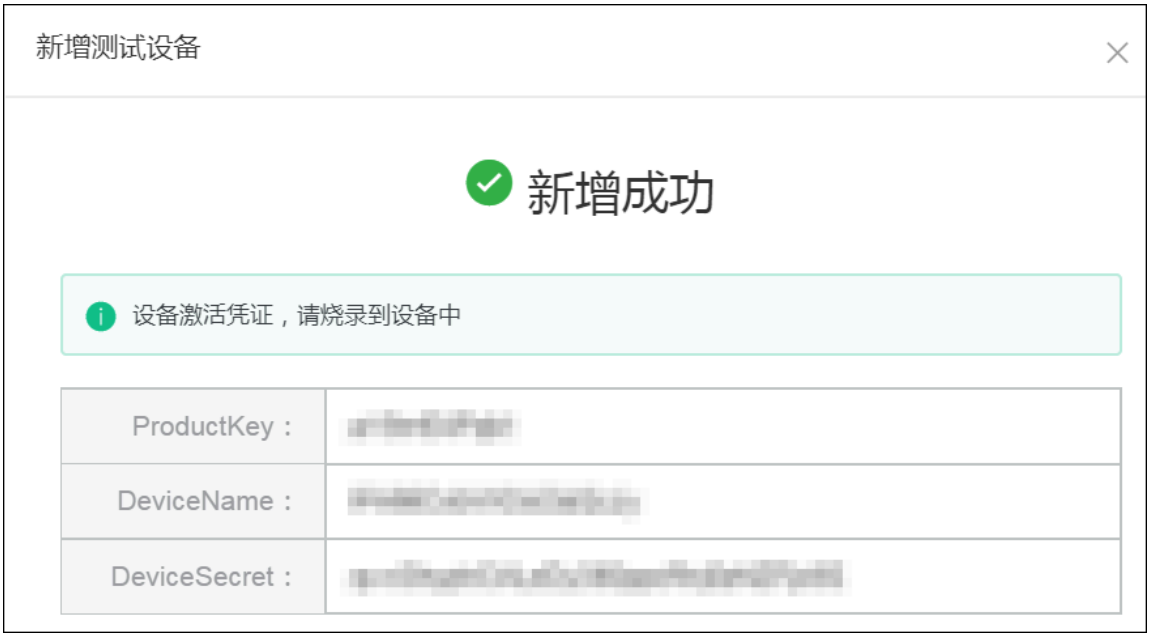
   参见#unique_20来定义产品的功能。

3. 调试设备。

a) 选择已认证的模组。

b) 新增测试设备，并获取设备激活凭证。

详细操作参见#unique_21。



4. 在产品的**人机交互**页面，打开**使用公版App控制产品**的开关，使用生活物联网平台提供的公版App对设备进行控制。App参数的配置请参见#unique_22。

5. 在**批量投产**页面确认设备信息。

> **说明:**
> 设备开发结束之后才会投产，开发阶段请不要单击**完成开发**。

产品功能开发

1. 开发产品模型。

   对于Wi-Fi设备来说，首先需要通过Wi-Fi配网来获得Wi-Fi热点的SSID/密码。而Wi-Fi配网的移植和调试比较耗费时间，提供以下两种方式，可以实现在产品开发阶段，同时进行产品功能开发和Wi-Fi配网功能移植调试。

   - 让设备直接连接一个指定的热点，然后连接到生活物联网平台，从而可以开始开发与调试产品物模型功能。如下示例代码中注释掉配网的代码。

     > **说明:**
     > 被修改的start_netmgr()函数位于AliOS-Things/example/linkkitapp/app_entry.c文件中。

     ```
     int application_start(int argc, char **argv)
     {
       ...
     #if 0
     #ifdef SUPPORT_DEV_AP
         aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
     #else
        aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
     #endif
     #endif
        //下面的代码让设备直接去连接一个指定SSID
        netmgr_ap_config_t config;
        strncpy(config.ssid, "your_ssid", sizeof(config.ssid) - 1);
        strncpy(config.pwd, "your_ssid_password", sizeof(config.pwd) - 1);
        netmgr_set_ap_config(&config);
        netmgr_start(false);
     ...
      }
     ```

   - 在调试阶段可以用**cli**命令进行配网。当设备连接到Wi-Fi热点，获得一个IP地址之后会自动去连接生活物联网平台。

     ```
     netmgr connect ssid password
     ```

2. 配置设备身份信息。

   在linkkit_example_solo.c的代码中设置了设备的身份信息，设备开发者需要将测试设备的信息对其进行替换。

   ```
   // for demo only
   #define PRODUCT_KEY     "a15****PqM"
   ```

```
#define PRODUCT_SECRET  "4uZsr*****zhjPM"
#define DEVICE_NAME     "IFn6******OaI2cJy"
#define DEVICE_SECRET   "qwvShyphC*******NZFjc8S"
```

设备开发者将设备身份信息设置到程序之后，可以将代码进行编译并遵循模组商提供的烧写工具将固件写入模组，确保设备可以连接到阿里云物联网平台。如果模组提供串口打印输出，当模组连接到阿里云物联网平台后将会输出类似如下的提示信息。

```
[inf] iotx_mc_init(2183): MQTT init success!

[023265]<I> Loading the CA root certificate ...

[023274]<I>  ok (0 skipped)

[023279]<I> Connecting to /a15trrE4PqM.iot-as-mqtt.cn-shanghai.aliyuncs.com/1883...

[023412]<I>  ok

[023414]<I>   . Setting up the SSL/TLS structure...

[023419]<I>  ok

[023423]<I> Performing the SSL/TLS handshake...

[023761]<I>  ok

[023764]<I>   . Verifying peer X.509 certificate..


[023769]<I> certificate verification result: 0x00

[inf] iotx_mc_connect(2533): mqtt connect success!
```

如果模组可以正常连接阿里云物联网平台，在生活物联网平台的商家后台可以看到该设备已激活，以及设备连接到物联网平台的时间信息。

3. 上报产品属性。

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。属性变化的检测以及上报是由设备开发者定义和实现的。

示例代码中对应的产品具有一个LightSwitch的属性，类型为bool，还具有一个RGBColor的属性，类型为复合型。在linkkit_example()函数中调用了user_post_property()函数，用于上报相关属性，用户可以参照该代码上报产品的属性变化。

```
void user_post_property(void)
{
    static int example_index = 0;
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *property_payload = "NULL";

    if (example_index == 0) {
        /* Normal Example */
        property_payload = "{\"LightSwitch\":1}";
        example_index++;
    } else if (example_index == 1) {
        /* Wrong Property ID */
        property_payload = "{\"LightSwitchxxxx\":1}";
        example_index++;
    } else if (example_index == 2) {
        /* Wrong Value Format */
        property_payload = "{\"LightSwitch\":\"test\"}";
        example_index++;
    } else if (example_index == 3) {
        /* Wrong Value Range */
        property_payload = "{\"LightSwitch\":10}";
        example_index++;
    } else if (example_index == 4) {
        /* Missing Property Item */
        property_payload = "{\"RGBColor\":{\"Red\":45,\"Green\":30}}";
        example_index++;
    } else if (example_index == 5) {
        /* Wrong Params Format */
        property_payload = "\"hello world\"";
        example_index++;
    } else if (example_index == 6) {
        /* Wrong Json Format */
        property_payload = "hello world";
        example_index = 0;
    }

    res = IOT_Linkkit_Report(user_example_ctx->master_devid, ITM_MSG_POST_PROPERT
Y,
                             (unsigned char *)property_payload, strlen(property_payload));

    EXAMPLE_TRACE("Post Property Message ID: %d", res);
```

```
}
```

4. 上报产品事件。

   如果产品定义了事件，当事件发生时也需要向云端发送事件。事件的检测以及上报由设备开发者实现。

   例如产品定义了一个标识符为Error的事件，该事件还有一个标识符为ErrorCode的输出参数。如下示例代码描述了如何向物联网平台发送一个事件。

```
void user_post_event(void)
{
    static int example_index = 0;
    int res = 0;
    user_example_ctx_t *user_example_ctx = user_example_get_ctx();
    char *event_id = "Error";
    char *event_payload = "NULL";

    if (example_index == 0) {
        /* Normal Example */
        event_payload = "{\"ErrorCode\":0}";
        example_index++;
    } else if (example_index == 1) {
        /* Wrong Property ID */
        event_payload = "{\"ErrorCodexxx\":0}";
        example_index++;
    } else if (example_index == 2) {
        /* Wrong Value Format */
        event_payload = "{\"ErrorCode\":\"test\"}";
        example_index++;
    } else if (example_index == 3) {
        /* Wrong Value Range */
        event_payload = "{\"ErrorCode\":10}";
        example_index++;
    } else if (example_index == 4) {
        /* Wrong Value Range */
        event_payload = "\"hello world\"";
        example_index++;
    } else if (example_index == 5) {
        /* Wrong Json Format */
        event_payload = "hello world";
        example_index = 0;
    }

    res = IOT_Linkkit_TriggerEvent(user_example_ctx->master_devid, event_id, strlen(
event_id),
                                   event_payload, strlen(event_payload));
    EXAMPLE_TRACE("Post Event Message ID: %d", res);
}
```

   linkkit_example()函数中的linkkit_ops定义了系统的各种事件处理函数，处理产品回调函数。示例代码如下所示。

```
int linkkit_example()
{
...
     /* 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_eve
nt_handler);
    IOT_RegisterCallback(ITE_SERVICE_REQUST, user_service_request_event_handler);
    IOT_RegisterCallback(ITE_PROPERTY_SET, user_property_set_event_handler);
    IOT_RegisterCallback(ITE_PROPERTY_GET, user_property_get_event_handler);
    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_TIMESTAMP_REPLY, user_timestamp_reply_event_handler);
    IOT_RegisterCallback(ITE_INITIALIZE_COMPLETED, user_initialized);
    IOT_RegisterCallback(ITE_FOTA, user_fota_event_handler);
    IOT_RegisterCallback(ITE_COTA, user_cota_event_handler);
```

```
};
```

关于回调函数的说明如下。

事件	回调函数原型	事件触发条件说明
ITE_CONNECT_SUCC	int callback(void);	与云端连接成功时
ITE_DISCONNECTED	int callback(void);	与云端连接断开时
ITE_RAWDATA_ARRIVED	int callback(const int devid, const unsigned char *payload, const int payload_len);	Linkkit收到raw data数据时
ITE_SERVICE_REQUEST	int callback(const int devid, const char _serviceid, const int serviceid_len, const char request, const int request_len, char _response, int *response_len);	Linkkit收到服务（同步/异步）调用请求时
ITE_PROPERTY_SET	int callback(const int devid, const char *request, const int request_len);	Linkkit收到属性设置请求时
ITE_PROPERTY_GET	int callback(const int devid, const char _request, const int request_len, char _response, int response_len);	Linkkit收到属性获取的请求时
ITE_REPORT_REPLY	int callback(const int devid, const int msgid, const int code, const char *reply, const int reply_len);	Linkkit收到上报消息的应答时
ITE_TRIGGER_EVENT_REPLY	int callback(const int devid, const int msgid, const int code, const char eventid, const int eventid_len, const char message, const int message_len);	Linkkit收到事件上报消息的应答时
ITE_TIMESTAMP_REPLY	int callback(const char * timestamp);	当Linkkit收到查询时间戳请求的应答时
ITE_TOPOLIST_REPLY	int callback(const int devid, const int msgid, const int code, const char * payload, const int payload_len);	Linkkit收到查询拓扑关系请求的应答时
ITE_PERMIT_JOIN	int callback(const char * product_key, const int time);	Linkkit收到允许子设备入网的请求时
ITE_INITIA		设备初始化完成时

在linkkit_example()中存在一个循环，其中IOT_Linkkit_Yield必须周期调用，用于linkkit业务处理。代码如下所示。

```
time_begin_sec = user_update_sec();
    while (1) {
        IOT_Linkkit_Yield(USER_EXAMPLE_YIELD_TIMEOUT_MS);

        time_now_sec = user_update_sec();
        if (time_prev_sec == time_now_sec) {
            continue;
        }
        if (max_running_seconds && (time_now_sec - time_begin_sec > max_runnin
g_seconds)) {
            EXAMPLE_TRACE("Example Run for Over %d Seconds, Break Loop!\n",
max_running_seconds);
            break;
        }

        /* Post Property Example */
        if (time_now_sec % 11 == 0 && user_master_dev_available()) {
            user_post_property();
        }
        /* Post Event Example */
        if (time_now_sec % 17 == 0 && user_master_dev_available()) {
            user_post_event();
        }

        /* Device Info Update Example */
        if (time_now_sec % 23 == 0 && user_master_dev_available()) {
            user_deviceinfo_update();
        }

        /* Device Info Delete Example */
        if (time_now_sec % 29 == 0 && user_master_dev_available()) {
            user_deviceinfo_delete();
        }

        /* Post Raw Example */
        if (time_now_sec % 37 == 0 && user_master_dev_available()) {
            user_post_raw_data();
        }

        time_prev_sec = time_now_sec;
    }
```

linkkit_example()中还包含了如下所示一段演示代码，用于上报所有的属性、事件等，在实际产品开发时需要将其修改为设备商自己的逻辑。

```
while (1) {
        IOT_Linkkit_Yield(USER_EXAMPLE_YIELD_TIMEOUT_MS);


        /* Post Property Example */
        if (user_master_dev_available()) {
            user_post_property();
        }
        /* Post Event Example */
        if (user_master_dev_available()) {
            user_post_event();
        }
```

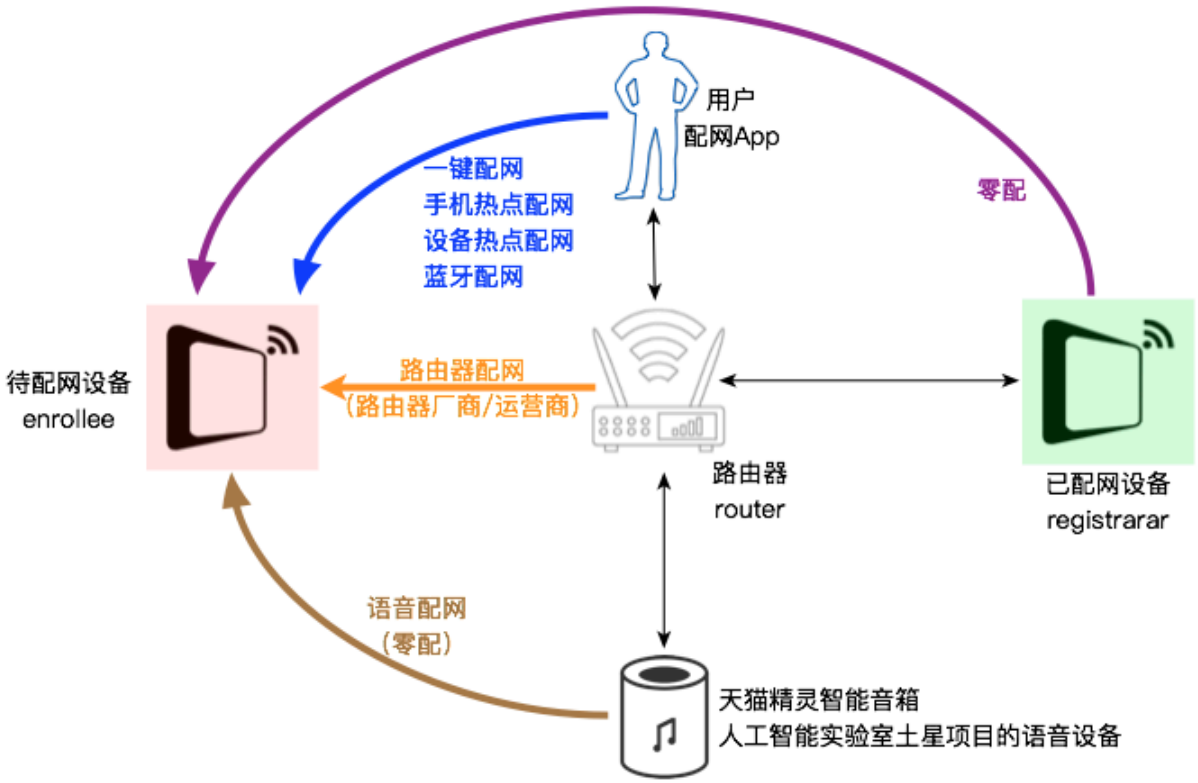
```
}
```

设备商完成自己物模型功能的代码编写之后，可以将固件编译出来并根据相关模组的烧写方法把固件烧写到模组上进行功能验证和调试。

Wi-Fi配网

配网支持如下所示。

- 一键配网（Smartconfig）：App直接给设备配网
- 手机热点配网（phone-config）：App直接给设备配网
- 路由器热点配网（router-config）：输出到路由器厂商/运营商
- 零配（zero-config）：已配网设备为待配网设备配网
- 设备热点配网（dev-ap）：设备开热点，手机连接设备热点完成为设备配网
- 蓝牙配网（ble-config）：借助BT/BLE为设备配网



注意：

在开发Wi-Fi配网时，请确保去除前面章节开发产品物模型时对start_netmgr()的修改。

```
static void start_netmgr(void *p)
{

    /*
     * register event callback to detect event of AWSS
```

```
    */
    iotx_event_regist_cb(linkkit_event_monitor);
    netmgr_start(true);
    aos_task_exit(0);
}
```

详细的开发过程如下步骤所示。

1. 配网对外披露的API列表。

```
/*
 * Copyright (C) 2015-2018 Alibaba Group Holding Limited
 */

#ifndef __IOT_EXPORT_AWSS_H__
#define __IOT_EXPORT_AWSS_H__

#if defined(__cplusplus)  /* If this is a C++ compiler, use C linkage */
extern "C" {
#endif

/**
 * @brief   start Wi-Fi setup service
 *
 * @retval  -1 : Wi-Fi setup fail
 * @retval  0 : sucess
 * @note: awss_config_press must been called to enable Wi-Fi setup service
 */
int awss_start();

/**
 * @brief   stop wifi setup service
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *     if awss_stop is called before exit of awss_start, awss and notify will stop.
 *     it may cause failutre of awss and device bind.
 */
int awss_stop();

/**
 * @brief   make sure user touches device belong to themselves
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note: AWSS dosen't parse awss packet until user touch device using this api.
 */
int awss_config_press();

/**
 * @brief   start Wi-Fi setup service with device ap
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *     1. if awss_stop or awss_dev_ap_stop is called before exit of awss_dev_ap_start
 *        awss with device ap and notify will stop, it may cause failutre of device ap
 *        and device bind.
 *     2. awss_dev_ap_start doesn't need to call awss_config_press to been enabled.
 */
int awss_dev_ap_start();
```

```
/**
 * @brief   stop Wi-Fi setup service with device ap
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      if awss_dev_ap_stop is called before exit of awss_dev_ap_start
 *      awss with device ap and notify will stop, it may cause failutre of device ap
 */
int awss_dev_ap_stop();

/**
 * @brief   report token to cloud after Wi-Fi setup success
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 */
int awss_report_cloud();

/**
 * @brief   report reset to cloud.
 *
 * @retval  -1 : failure
 * @retval  0 : sucess
 * @note
 *      device will save reset flag if device dosen't connect cloud, device will fails to send
 reset to cloud.
 *      when connection between device and cloud is ready, device will retry to report
reset to cloud.
 */
int awss_report_reset();

enum awss_event_t {
    AWSS_START = 0x1000,       // AWSS start without enbale, just supports device
discover
    AWSS_ENABLE,               // AWSS enable
    AWSS_LOCK_CHAN,            // AWSS lock channel(Got AWSS sync packet)
    AWSS_CS_ERR,               // AWSS AWSS checksum is error
    AWSS_PASSWD_ERR,           // AWSS decrypt passwd error
    AWSS_GOT_SSID_PASSWD,      // AWSS parse ssid and passwd successfully
    AWSS_CONNECT_ADHA,         // AWSS try to connnect adha (device discover, router
solution)
    AWSS_CONNECT_ADHA_FAIL,    // AWSS fails to connect adha
    AWSS_CONNECT_AHA,          // AWSS try to connect aha (AP solution)
    AWSS_CONNECT_AHA_FAIL,     // AWSS fails to connect aha
    AWSS_SETUP_NOTIFY,         // AWSS sends out device setup information (AP and
router solution)
    AWSS_CONNECT_ROUTER,       // AWSS try to connect destination router
    AWSS_CONNECT_ROUTER_FAIL,  // AWSS fails to connect destination router.
    AWSS_GOT_IP,               // AWSS connects destination successfully and got ip address
    AWSS_SUC_NOTIFY,           // AWSS sends out success notify (AWSS sucess)
    AWSS_BIND_NOTIFY,          // AWSS sends out bind notify information to support bind
 between user and device
    AWSS_ENABLE_TIMEOUT,       // AWSS enable timeout(user needs to call awss_confi
g_press again to enable awss)
    AWSS_RESET = 0x3000,       // Linkkit reset success (just got reset response from cloud
 without any other operation)
};

#if defined(__cplusplus)  /* If this is a C++ compiler, use C linkage */
}
#endif
```

```
#endif
```

2. App中调用配网。

```
/*
 * application_start is application entrance based on sdk.
 */
int application_start(int argc, char **argv)
{
    ......

    /*
     * set device triple ID information before AWSS, otherwise AWSS will fail.
     * HAL_SetProductKey(product_key);
     * HAL_SetProductSecret(product_secret);
     * HAL_SetDeviceName(dev_name);
     * HAL_SetDeviceSecret(dev_secret);
     */
    set_iotx_info();

    /*
     * set log level to print debug information about AWSS
     */
    LITE_set_loglevel(5);  // 5 for debug level

    /*
     * Start netmgr task for AWSS
     * default stack size of netmgr task is 4096B,
     * if the module or die takes more stack size,
     * please set larger stack size (maybe, 6KB)
     */
#ifdef SUPPORT_DEV_AP
    aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
#else
    aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
#endif

    aos_loop_run();

    return 0;
}
#ifdef SUPPORT_DEV_AP
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);
        awss_dev_ap_start();
    }
    aos_task_exit(0);
}
#endif
static void start_netmgr(void *p)
{
    /*
     * register event callback to detect event of AWSS
     */
    iotx_event_regist_cb(linkkit_event_monitor);
    netmgr_start(true);
    aos_task_exit(0);
```

```
}
```

说明：

- set_iotx_info一定要在**awss_start**或**netmgr_start**之前调用设备证书信息，否则设备无法解析路由器的PASSWORD，PASSWORD采用加密措施保证安全性，而密钥需要借助于设备证书信息计算。
- LIET_set_loglevel函数如果需要调试打开log，该函数需要在**awss_start**前调用。
- **awss_start**只是开始AWSS服务（发现周围的AP列表），并未使能AWSS服务。如果需要设备解析配网包，考虑安全问题，还需要调用awss_press_config来使能AWSS（设备热点配网除外）。
- 关于调用awss_press_config的时机和策略，虽然Demo中采用按键触发，但产品厂商可以根据自己的产品的特点来自行设计。

目前awss_start已经被AliOS封装在netmgr模块中（netmgr_start），具体可以阅读netmgr_start的代码。

```
int netmgr_start(bool autoconfig)
{
    ......

    /*
     * if the last AP information exists
     * try to connect the last AP.
     */
    if (has_valid_ap() == 1) {
        aos_post_event(EV_WIFI, CODE_WIFI_CMD_RECONNECT, 0);
        return 0;
    }

    ......

     /*
     * if the last AP information doesn't exist
     * start AWSS (Alibaba Wireless Setup Service)
     */
    if (autoconfig) {
        netmgr_wifi_config_start();  // call awss_start()
        return 0;
    }

    ......

    return -1;
}
```

3. 特定Demo说明。

   - 从设备热点配网切换到其他配网模式，调用do_awss。

     ```
     void do_awss()
     {
     ```

```
    aos_task_new("dap_close", awss_close_dev_ap, NULL, 2048);
    aos_task_new("netmgr_start", start_netmgr, NULL, 4096);
}

static void awss_close_dev_ap(void *p)
{
    awss_dev_ap_stop();
    LOG("%s exit\n", __func__);
    aos_task_exit(0);
}

static void start_netmgr(void *p)
{
    iotx_event_regist_cb(linkkit_event_monitor);
    LOG("%s\n", __func__);
    aos_msleep(2000);
    netmgr_start(true);
    aos_task_exit(0);
}
```

- 从其他配网模式切换到设备热点配网，调用**do_awss_dev_ap**。

```
void do_awss_dev_ap()
{
    aos_task_new("netmgr_stop", stop_netmgr, NULL, 4096);
    aos_task_new("dap_open", awss_open_dev_ap, NULL, 4096);
}

static void stop_netmgr(void *p)
{
    awss_stop();
    LOG("%s\n", __func__);
    aos_task_exit(0);
}

static 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);
        awss_dev_ap_start();
    }
    aos_task_exit(0);
}
```

更多的Wi-Fi配网的描述可参见配网开发文档。

云端解绑与恢复出厂默认设置通知

设备被解绑后，云端会下发一个解绑事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。

如果通过App将设备恢复出厂默认设置，云端会下发一个Reset事件通知：`{"identifier":"awss.BindNotify","value":{"Operation":"Reset"}}` 设备收到此消息可以做重置配网、清空本地数据等处理。设备开发者可以结合具体产品类型，决定收到解绑和恢复出厂默认设置通知后做哪些清空操作。

可以参考示例代码example/smart_outlet/smart_outlet_main.c中notify_msg_handle函数，做如下改动。

```
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 (item == 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")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }

    ....
}
```

设备重置

对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用awss_report_reset()函数告知云端清除设备与用户的绑定关系。

因此，设备商需要在处理reset按键的逻辑中增加对awss_report_reset()的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

开发OTA

若使能了OTA功能，请参见OTA编程。

2.5.5 基于非认证的模组开发

如果用户不使用已认证的模组，本文讲解用户如何直接通过生活物联网SDK（不含AliOS Things）开发Wi-Fi单品设备，用以连接生活物联网平台。

在生活物联网平台定义产品

1. 创建项目。

   参见#unique_24。

2. 创建产品。

   参见#unique_19，**连网方式**选择为**WiFi**。生活物联网的公版App可以通过本地发现方式来添加设备。

   > **说明:**
   >
   > 如果手机App使用生活物联网的公版App对设备进行管理，那么添加设备时如果发现设备是Wi-Fi设备，那么就会进入阿里的Wi-Fi配网过程对设备进行配网，因此设备需要集成阿里的Wi-Fi配网功能。
   >
   > 如果设备并没有集成阿里的Wi-Fi配网功能，而是通过触摸屏输入Wi-Fi热点的SSID/密码或者其它配网方式，联网方式可以选择**以太网**，以避免生活物联网的公版App使用阿里配网流程来添加设备，但是同时可以使用本地发现方式来添加设备。

3. 产品功能定义。

   参见#unique_20来定义产品的功能。

4. 添加测试设备。

   参见#unique_25来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见SDK概述与开发环境设置下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行`make menuconfig` 来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台

功能	说明
FEATURE_MQTT_DIRECT	- y：连国内服务器 - n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	y：使能本地控制功能
FEATURE_ALCS_SERVER_ENABLED	y：使能本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	y：使能用户绑定相关功能
FEATURE_SUPPORT_TLS	y：使能TLS加密
FEATURE_OTA_ENABLED	y：使能OTA
FEATURE_WIFI_PROVISION_ENABLED	y：使能Wi-Fi配网
FEATURE_AWSS_SUPPORT_DEV_AP	y/n：根据需要打开或关闭设备热点配网
FEATURE_AWSS_SUPPORT_SMARTCONFIG	y/n：根据需要打开或关闭一键配网
FEATURE_AWSS_SUPPORT_PHONEASAP	y/n：根据需要打开或关闭手机热点配网
FEATURE_AWSS_SUPPORT_ROUTER	y/n：根据需要打开或关闭路由器配网
FEATURE_AWSS_SUPPORT_ZEROCONFIG	y/n：根据需要打开或关闭零配配网

设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配
  - 线程相关HAL的适配
  - Wi-Fi配网HAL的适配注：若未选择集成阿里提供的Wi-Fi配网功能，可以不用适配这些HAL
  - OTA HAL的适配
  - 本地通信HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- Wi-Fi配网开发

  请访问生活物联网SDK的Wi-Fi配网开发指南了解如何调用配网相关的API实现Wi-Fi配网。

- 云端解绑与恢复出厂默认设置

  设备被解绑后，云端会下发一个解绑事件通知：{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}} 设备收到此消息可以做重置配网、清空本地数据等处理。

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

  可以参考示例代码example/smart_outlet/smart_outlet_main.c中notify_msg_handle函数，做如下改动。

```
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 (item == 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")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }

    ....
```

```
}
```

- 设备重置

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用awss_report_reset()函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对awss_report_reset()的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

- OTA开发

  若使能了OTA功能，请参见OTA编程。

2.6 网关及子设备开发

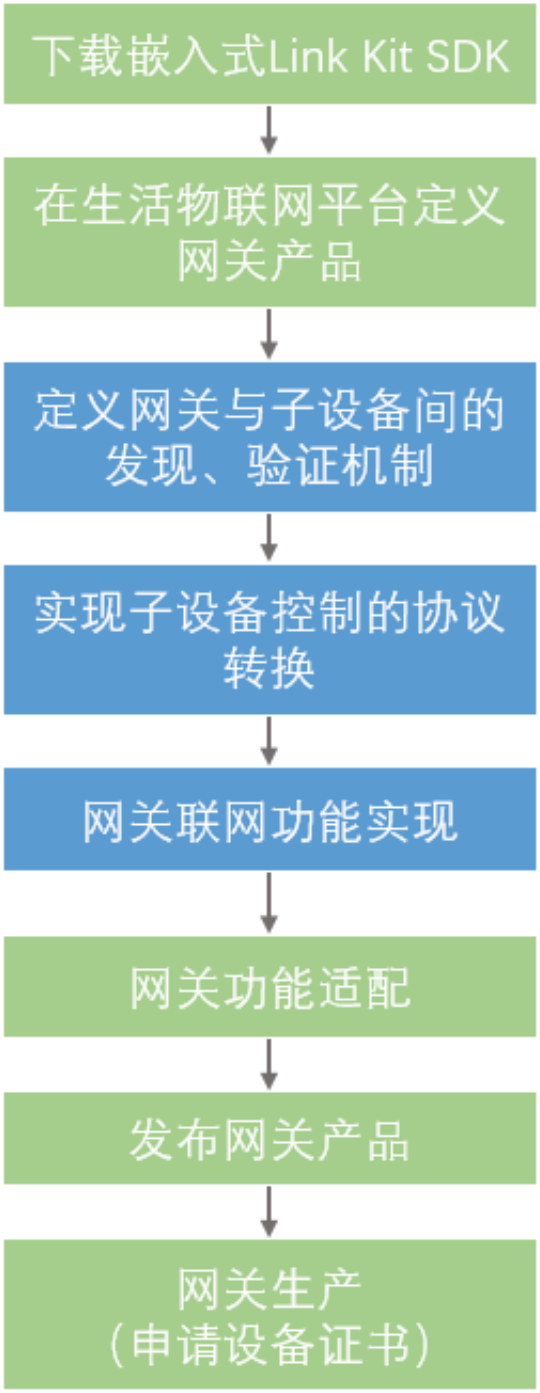
本文介绍用户如何通过生活物联网SDK开发接入生活物联网平台的网关及子设备。

术语解释

- 网关设备：也叫主设备，是指可以挂载子设备的直连设备，网关具有子设备管理模块，维持子设备的拓扑关系，并且可以将拓扑关系同步到云端。
- 子设备：本质上也是设备，但子设备不能直接连接到云端，只能通过挂接到网关上，通过网关间接连云，例如使用zigbee协议联网的设备。
- 设备ID：设备句柄，在网关SDK中用于标识一个具体的设备。
- 设备证书：指设备的productKey、deviceName和deviceSecret，用于在阿里IoT云中唯一标识设备。
- 拓扑关系：子设备和网关的关联关系为拓扑关系，子设备与网关建立拓扑关系后，便可以复用网关的物理通道进行数据通信。
- 子设备动态注册：子设备无需烧录一机一密（设备证书），只需要烧录ProductKey、ProductSecret，然后基于子设备的唯一标识作为设备证书中的deviceName（例如SN），再使用网关SDK到云端动态注册获取deviceSecret，从而得到完整的设备证书。子设备就可以使用该设备证书到云端进行设备的身份认证，并进而被云端统一管理。
- TSL：Thing Specification Language，基于JSON格式，用于描述设备所具备的功能和能力，详细说明参见物模型介绍。

网关开发流程

网关产品的开发流程如下图所示。



请参见SDK概述与开发环境设置下载无AliOS的SDK。

- 底色为蓝色的步骤表示该步骤中的功能完全由网关厂商自己定义与实现。

- 网关如何去发现以及连接子设备是由网关厂商与子设备厂商去协商和定义的，平台并不参与网关与子设备之间的通信过程和数据格式定义。但是当网关添加一个子设备后，需要使用平台提供的子设备验证机制 来验证子设备的合法性。
- 当手机App对子设备进行控制时，命令将会通过手机发送到云端、再由云端发送到网关设备后转发给子设备。网关与子设备之间的数据格式由网关厂商和子设备厂商定义，因此网关需要进行数据格式从平台格式到厂商数据格式的转换。
- 平台提供的网关SDK并不包含网关如何获取IP地址的功能。网关可能使用以太网、WiFi、或者蜂窝网技术（GRPS、3G、4G等）方式连网，网关上如何通过DHCP获取IP地址、域名服务器地址、下一跳路由器的IP地址等功能由网关厂商实现。
- 如果网关使用WiFi接入网络，并且网关具有串口、或者web server这样的功能可以让用户输入WiFi热点的SSID/密码，那么网关厂商可以不用适配与集成阿里的WiFi配网功能。

在生活物联网平台定义网关产品

1. 创建项目。

   参见#unique_24。

2. 创建产品。

   参见#unique_19，并按照以下要求设置参数。

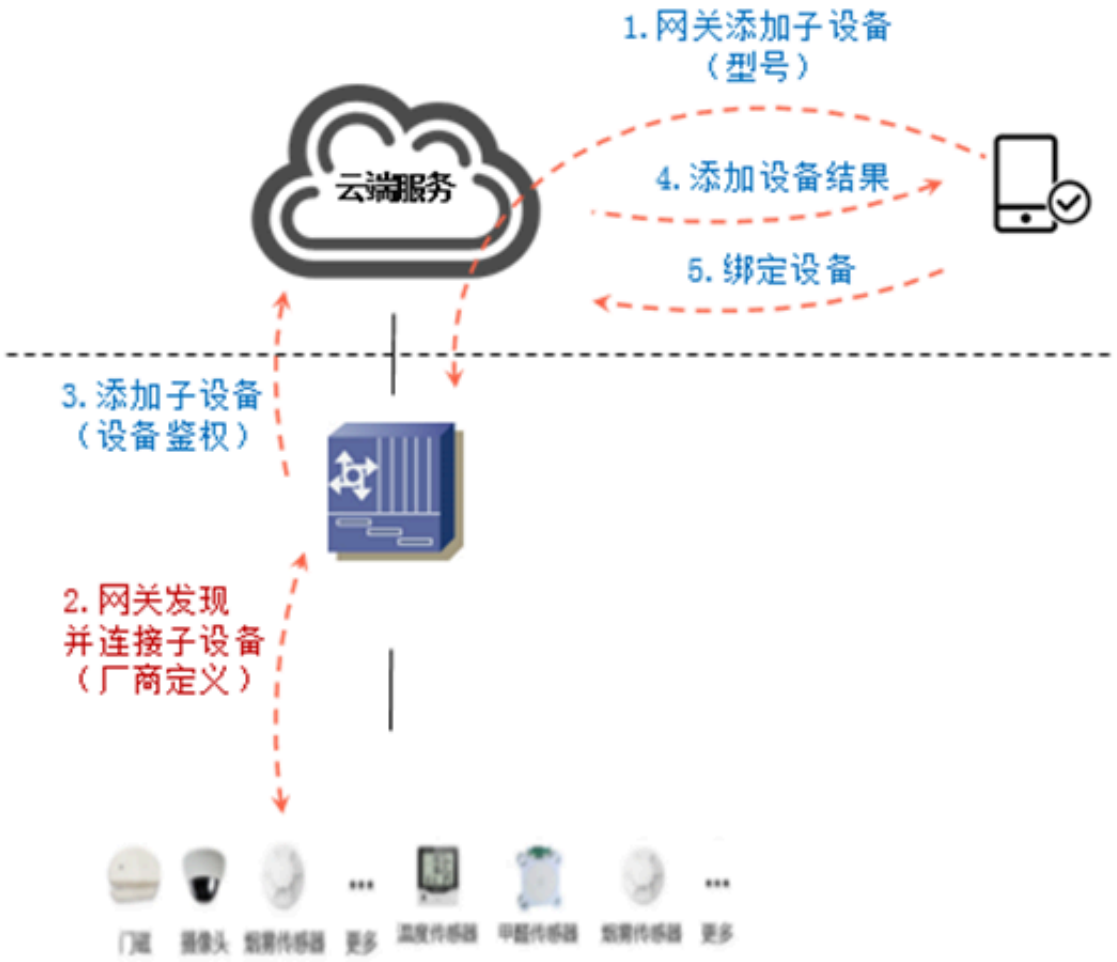
   - 选择**所属分类**为**网络设备**中的**网关**
   - 选择**节点类型**为**网关**
   - 根据网关实际连网方式选择**连网方式**

3. 添加测试设备。

   参见#unique_25来添加测试设备，并设置设备的绑定策略等。

终端用户添加子设备的交互示意图

本节介绍用户添加子设备的整个交互示意，以让网关厂商了解子设备是添加到云端的整体过程。设备用户购买了网关和子设备之后，需要先添加网关，再将子设备添加到网关。完成后才能对子设备进行控制。下面是已添加网关后，添加子设备的多端交互示意图。



当用户发起子设备添加时，手机将会向云端发送PermitJoin命令，之后云端会将该命令转发给网关，此时网关需要去发现与连接子设备，网关如何去发现与连接子设备是由网关厂商与子设备厂商来定义和实现的。当网关收到PermitJoin后，将发现的子设备上报到云端。

说明：

如果网关没有收到来自云端的permitJoin消息，即使子设备已连接到网关，也不要向云端添加子设备；可以等到收到permitJoin之后再将已连接的子设备通知云端。

云端与网关之间数据格式由阿里云IoT定义，网关与子设备之间的数据格式是网关厂商与子设备厂商定义，因此网关需要实现阿里云IoT数据格式与子设备之间数据格式的转换工作。

当云端添加子设备到网关之后，将会发送消息到手机端，从而让用户可以看见添加的子设备，继而对子设备进行控制。

SDK应用于网关时的推荐配置

解压获取到的SDK包后，在主目录下的make.settings文件，定义功能模块的需求。对网关而言，除了不同联网方式（WiFi，以太网、蜂窝网）的不同配置之外，还需要配置以下参数的使能。

```
FEATURE_DEVICE_MODEL_GATEWAY=y
FEATURE_DEPRECATED_LINKKIT=n
```

HAL适配

请参照下面的文档进行HAL的实现。

- 基础HAL适配
- MQTT连云相关的HAL适配
- OTA HAL的适配
- WiFi配网HAL的适配

  若未选择集成阿里提供的WiFi配网功能，可以不用适配这些HAL
- 本地通信HAL的适配

  如果网关使用蜂窝网接口上行，不需要适配本地通信的HAL
- 线程相关HAL的适配

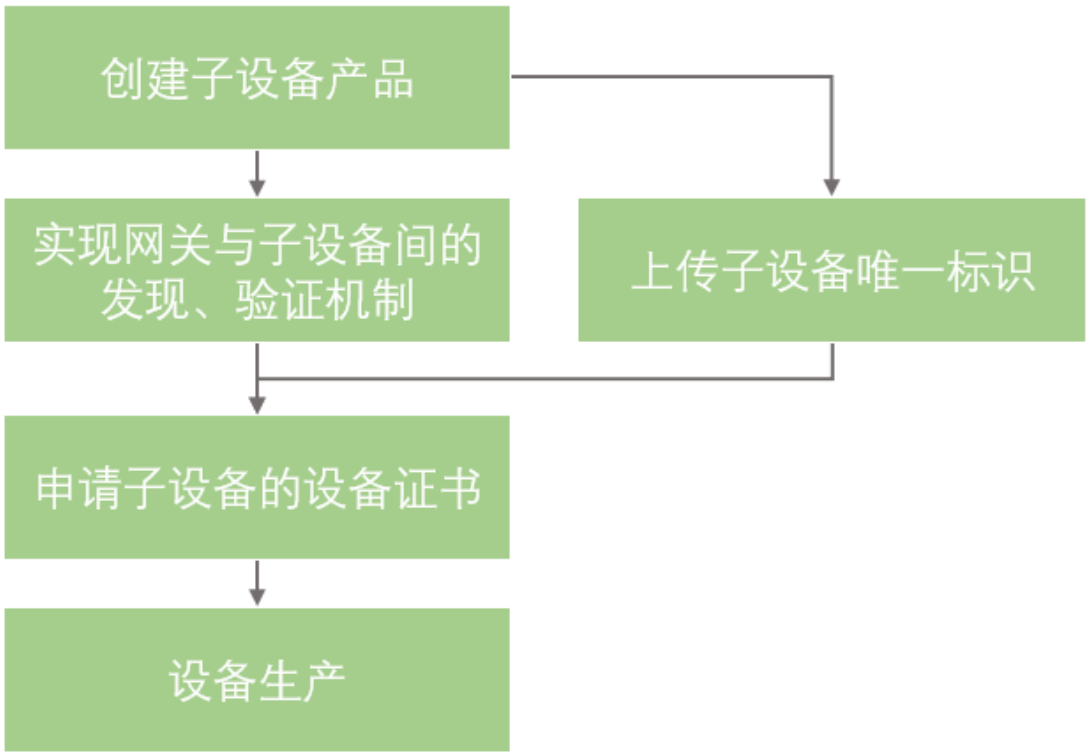
  如果未选择WiFi配网、也未选择本地通信功能，可以不用适配线程相关的HAL

网关编程

请参见子设备管理，了解如何进行网关的初始化、如何添加子设备、如何上报子设备的在线状态、以及对子设备进行管理。

子设备产品开发

子设备产品开发流程如下图所示。



在生活物联网平台为子设备创建产品，并按照以下要求设置参数。

- 选择**节点类型**为**设备**
- 选择**是否接入网关**为**是**
- 根据子设备实际连网方式选择**连网方式**

说明：

每个子设备也需要在智能生活平台上创建产品，并定义该产品支持的属性、事件、服务。子设备产品的注册方式为“动态注册”，每个子设备都需要一个有效且唯一的阿里IoT激活码。目前该激活码不需要烧写到子设备上，但是子设备厂商需要为每个产品申请激活码。子设备的激活码的DeviceName需要由子设备厂商指定，并上传到生活物联网平台，请参见#unique_27了解步骤。

2.7 以太网设备端开发

本文介绍用户如何通过生活物联网SDK开发接入生活物联网的以太网设备。

在生活物联网平台定义产品

1. 创建项目。

   参见#unique_24。

2. 创建产品。

   参见#unique_19，**连网方式**选择为**以太网**。生活物联网的公版App可以通过本地发现方式来添加设备。

3. 产品功能定义。

   参见#unique_20来定义产品的功能。

4. 添加测试设备。

   参见#unique_25来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见SDK概述与开发环境设置下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行`make menuconfig` 来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台
FEATURE_MQTT_DIRECT	- y：连国内服务器 - n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	y：使能本地控制功能
FEATURE_ALCS_SERVER_ENABLED	y：使能本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	y：使能用户绑定相关功能
FEATURE_SUPPORT_TLS	y：使能TLS加密

功能	说明	
FEATURE_OTA_ENABLED	y：使能OTA	

设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配
  - 线程相关HAL的适配
  - OTA HAL的适配
  - 本地通信HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- OTA开发

  若使能了OTA功能，请参见OTA编程。

- 云端解绑与恢复出厂默认设置

  设备被解绑后，云端会下发一个解绑事件通知：{"identifier":"awss.BindNotify","value":{"Operation":"Unbind"}} 设备收到此消息可以做重置配网、清空本地数据等处理。

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

  可以参考示例代码example/smart_outlet/smart_outlet_main.c中notify_msg_handle函数，做如下改动。

```
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 (item == 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")) {
                EXAMPLE_TRACE("Device Bind");
                vendor_device_bind();
            }
            if (!strcmp(op->valuestring, "Unbind")) {
                EXAMPLE_TRACE("Device unBind");
                vendor_device_unbind();
            }
            if (!strcmp(op->valuestring, "Reset")) {
                EXAMPLE_TRACE("Device Reset");
                vendor_device_reset();
            }
        }
    }

    ....
}
```

- 设备重置开发

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用awss_report_reset()函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对awss_report_reset()的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

2.8 蜂窝网设备端开发

本文介绍用户如何通过生活物联网SDK开发接入生活物联网的蜂窝网（2/3/4/5G移动通信协议、NB-IoT）设备。

在生活物联网平台定义产品

1. 创建项目。

   参见#unique_24。

2. 创建产品。

   参见#unique_19，**连网方式**选择**蜂窝（2/3/4G）**。

3. 添加测试设备。

   参见#unique_25来添加测试设备，并设置设备的绑定策略等。

设备端开发

请参见SDK概述与开发环境设置下载无AliOS的SDK（基于Link Kit v2.3.0）。

- SDK推荐配置

  建议开发者阅读编译说明中的“SDK裁剪”，了解SDK配置以及各选项的意义。

  可以通过修改make.settings或者Linux下执行make menuconfig来配置需要的功能。

功能	说明
FEATURE_MQTT_COMM_ENABLED	y：使用MQTT连接阿里云物联网平台
FEATURE_MQTT_DIRECT	y：连国内服务器 n：连海外服务器
FEATURE_Device_MODEL_ENABLED	y：使能物模型
FEATURE_ALCS_ENABLED	n：关闭本地控制功能
FEATURE_ALCS_SERVER_ENABLED	n：关闭本地控制被控端功能
FEATURE_DEV_BIND_ENABLED	n：关闭用户绑定相关功能
FEATURE_SUPPORT_TLS	y：使能TLS加密
FEATURE_OTA_ENABLED	y：使能OTA

  设备需要连接海外服务器请参见国际站设备开发。

- HAL适配

  请参照下面的文档进行HAL的实现：

  - 基础HAL适配
  - MQTT连云相关的HAL适配
  - OTA HAL的适配

- 设备身份认证模式

  设备连接阿里云物联网平台时，可以使用预置设备证书的方式进行设备的身份认证，也可以采用动态注册方式得到完整的设备证书再进行身份认证，请参见设备认证。

- 产品功能实现

  在设备上根据云端定义的产品功能进行相应功能的实现，请参见物模型编程。

- OTA开发

  若使能了OTA功能，请参见OTA编程。

- 设备重置开发

  对于生活物联网平台来说，建议产品设计一个reset按键用于清除设备上的配置，将设备恢复到出厂状态，同时调用awss_report_reset()函数告知云端清除设备与用户的绑定关系。

  因此，设备商需要在处理reset按键的逻辑中增加对awss_report_reset()的调用。

```
/*
 * 应用程序调用该API后，Linkkit首先往Flash里存储恢复出厂设置的标志，并向云端上报reset操作，
 * 在规定的时间内（3秒）如果没有收到云端的回复，设备会重新上传reset，直至收到云端的回复位置；
 * 有些产品希望发生reset时设备可以重新启动，如果重新启动之前reset没有上报成功，下一次连接云后，
 * 设备会首先检查Flash中恢复出厂标志是否设置，如果设置了则首先向云端上报reset，直至成功；
 */
int awss_report_reset();
```

2.9 蓝牙设备开发

2.9.1 蓝牙设备端开发

阿里云IoT生活物联网平台对低功耗蓝牙通信类型（BLE）的设备，提供了Breeze协议接入方案，可以便捷实现APP-设备-云的完整链路。

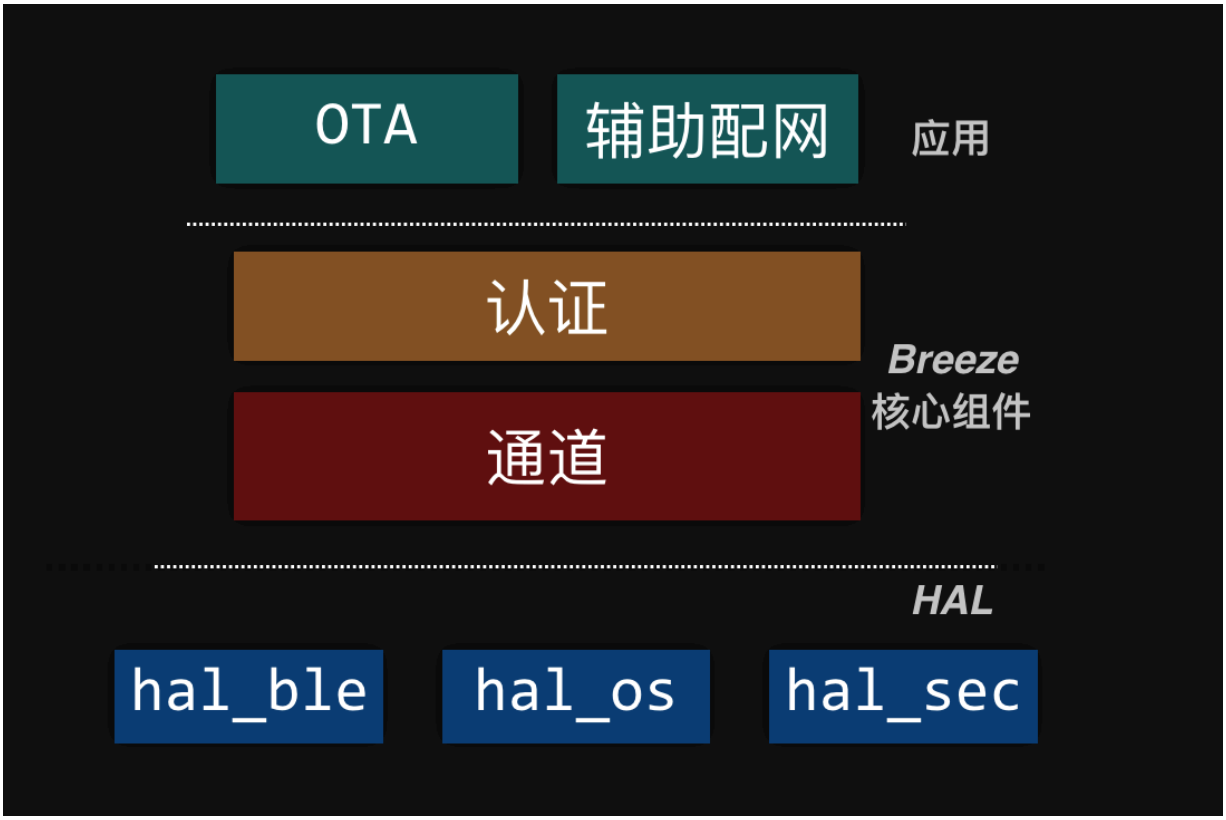
背景介绍

本文主要介绍在具备低功耗蓝牙（BLE）能力的设备端上，如何集成Breeze协议，并与手机端App建立连接、对BLE设备的身份进行认证、建立BLE设备与手机之间的安全通信通道，以及将BLE设备端数据通过手机App通道推送至云端等。手机端客户端和云端开发请参见蓝牙连接开发指南。

Breeze协议类似于BLE协议栈中的Profile（配置文件），需要适配并运行于BLE协议栈之上。

设备端Breeze SDK

Breeze协议规定了接入阿里云IoT生活物联网平台的BLE设备与移动端App之间的通信协议，为了方便用户实现更快速的接入，生活物联网设备端SDK提供了Breeze协议的代码实现：Breeze SDK，其结构框图如下所示。



- HAL层：Breeze SDK需要适配BLE Stack，HAL层就是用来适配BLE Stack之用，不同的BLE Stack需要适当修改进行移植和适配。
- 核心组件层：实现了设备端-移动端-云端的认证、通道等逻辑的源码，用户一般不用做修改即可使用。
- 应用层：Breeze提供了设备OTA通道、辅助WIFI配网通道等，用户需要根据实际场景进行使用。

设备端Breeze SDK代码结构

设备端Breeze SDK随生活物联网设备端SDK对外开源，最新版本请获取最近发布的生活物联网设备端SDK。

代码位于$(SDK Src)/framework/bluetooth/breeze，目录如下。

```
.
├──── Config.in  //配合meunconfig组件配置
├──── breeze.mk  //makefile文档
├──── README.md  //readme文档
├──── api    //目录，展示给应用的接口为breeze_export.*是Breeze SDK提供的接口;
breeze_awss_export.*是蓝牙辅助配网的接口。依照应用场景选择其一使用
├──── core  //目录，源码核心实现，包含安全，通道以及扩展指令等的实现
├──── hal  //参考移植实现，使用AliOS Things OS, BLE协议栈，以及内部mebdtls和安全部分
参考
├──── include   //Breeze SDK的头文件
```

使用开发指南

开发者可以通过SDK提供的HAL接口进行对接，在自己的运行了BLE协议栈的设备上支持Breeze协议。

移植SDK到第三方平台

breeze_hal_sec对接安全部分，包含AES128 CBC加解密算法和SHA256接口。其余部分代码和应用维持不变。参见源代码目录下的hal，将第三方的实现替换此文件夹下的HAL实现：OS、BLE协议栈、Security，定义在$(SDK Src)/framework/bluetooth/breeze/hal下，其余部分代码和应用维持不变。

- breeze_hal_ble：对接BLE蓝牙协议栈部分接口，包含蓝牙广播，注册Breeze蓝牙服务等
- breeze_hal_sec：对接安全部分，包含AES128 CBC加解密算法
- breeze_hal_os：对接不同OS的接口，包含OS timer的资源，KV（存储和读取接口）操作等

API接口列表

详细定义和说明参见蓝牙设备端SDK用户编程接口。

HAL接口列表

详细定义和说明参见蓝牙设备端SDK移植接口。

在生活物联网平台定义产品

由于设备最终要通过生活物联网平台认证，所以需要在生活物联网平台申请对应的产品认证信息。

1. 创建项目。

   参见#unique_24。

2. 新建产品。

   参见#unique_19，并按照以下要求配置。

   - 选择**节点类型**为**设备**
   - 选择**是否接入网关**为**是**
   - 选择**接入网关协议**为**BLE**

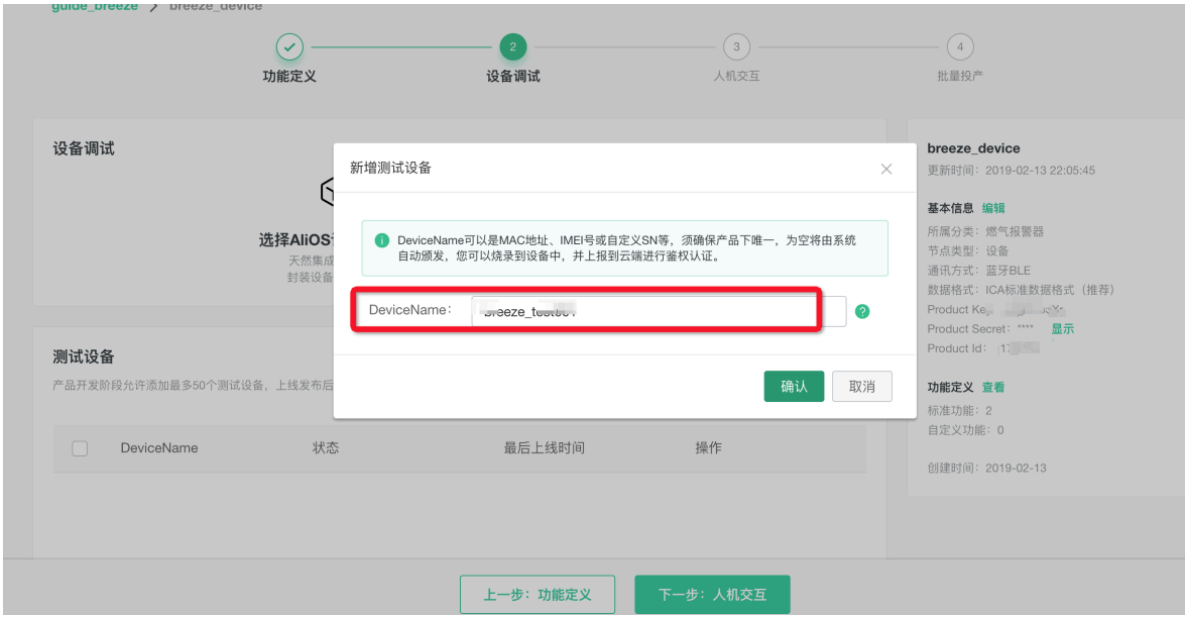
3. 定义产品功能。

   参见#unique_20来定义产品的功能。

4. （可选）在**产品-设备调试**界面，单击**新增测试设备**。

   在产品定义界面右边可以看到产品的三个信息：Product Key、Product Secret和Product ID。这对于一型一密设备来说已经足够，但一机一密还需要在**产品-设备调试**中生成对应同一类型产品的

多个设备，根据**DeviceName**进行区分**Device Secret**。此时，一机一密需要的五个元素信息都已经获得。



设备端高级应用场景开发

在SDK基础建立安全通道基础上，主要有以下两应用场景：低功耗蓝牙辅助WiFi配网和低功耗蓝牙设备OTA。

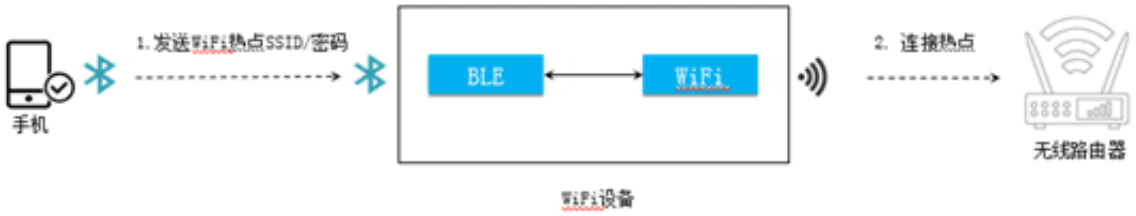
蓝牙辅助配网

- 使用场景

  WiFi设备需要连接WiFi热点（WiFi AP）之后才能与其它设备进行IP通信。对于没有键盘、没有触摸屏、没有Web Server的WiFi IoT设备而言，如何获取WiFi热点的SSID/密码是实现设备远程管理的第一个关键步骤。

  我们将WiFi设备获取到WiFi热点的SSID/密码的步骤称为WiFi配网，蓝牙辅助WiFi配网方案适用于同时支持蓝牙BLE以及WiFi的设备，该方案通过蓝牙BLE通道将WiFi热点的SSID/密码发送给WiFi

设备，从而让WiFi设备可以连接到WiFi AP（通常为WiFi无线路由器）。蓝牙辅助WiFi配网的工作原理示意图如下。

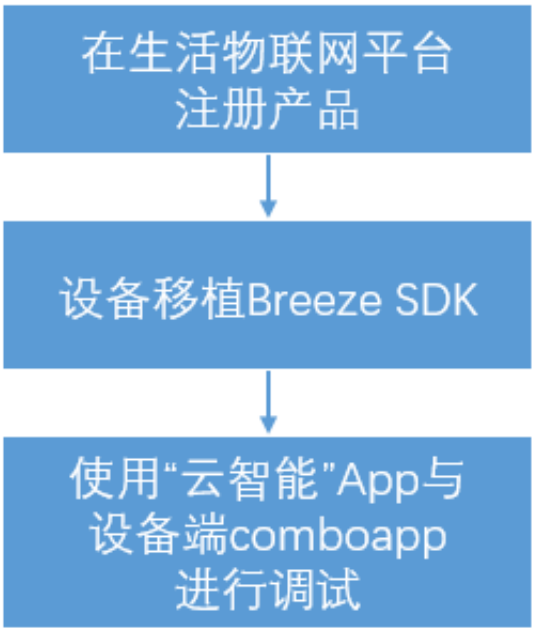


可以很容易得出蓝牙辅助WiFi配网构成。

- 设备端SDK：设备端硬件支持WiFi/BLE combo芯片，并运行基于Breeze SDK的蓝牙配网例程。
- 配网手机端SDK：手机集成配套的蓝牙配网SDK或者“云智能”App。

- 开发流程

  如果您是一个WiFi的模组商或者设备商，您的模组/设备支持蓝牙BLE以及WiFi并且希望集成阿里云IoT生活物联网平台提供的蓝牙辅助WiFi配网功能，可以按照下面的过程进行模组/设备开发。



  - 产品注册，参见设备开发流程，并在产品的**产品-人机交互**的**配网引导**中增加**蓝牙辅助配网**的配网方式。
  - 使用“云智能”App与设备端的comboapp开发。如果集成移动端SDK开发自有App的方式，可参见蓝牙连接开发指南。

- 设备端comboapp开发

  在完成设备移植Breeze SDK步骤后，可参考example/comboapp实现低功耗蓝牙辅助WIFI配网的功能，并将“产品注册”阶段的设备信息设置到comboapp中。

Breeze设备OTA

- OTA编译配置

  可以参考breezeapp示例，默认已经使能OTA功能。

- OTA对接移植

  OTA模块代码实现在framework/uOTA/src/device/ble下，包含以下目录。

```
.
├──── README.md
├──── ble.mk
```

```
├──── inc  //包含了对外的接口和对接API
│    ├──── ais_ota.h
│    ├──── ota_breeze.h
│    ├──── ota_breeze_export.h
│    ├──── ota_breeze_plat.h
│    └──── ota_breeze_transport.h
└──── src  // OTA内部逻辑的实现
   ├──── ota_breeze.c
   ├──── ota_breeze_plat.c
   ├──── ota_breeze_service.c
   └──── ota_breeze_transport.c
```

具体对接函数请参见蓝牙设备端SDK OTA接口。

- OTA调试和使用

  OTA的具体用法请参见蓝牙连接开发指南中“运营中OTA配置”章节。

2.9.2 蓝牙设备端SDK用户编程接口

Breeze协议提供基于蓝牙链路连接阿里云IoT的安全通道和服务，并且提供蓝牙辅助配网功能（通过BLE链路获取AP的SSID和password）。基本SDK API在breeze_export.h中，蓝牙配网的接口API在breeze_awss_export.h中。

breeze_start

启动breeze SDK服务。用户使用此接口初始化和启动breeze服务。

参数

名称	类型	描述
dev_conf	device_config	初始化Breeze SDK的信息，包含设备信息，回调函数等。

返回值

0-成功；-1-失败

breeze_end

停止breeze服务，用户调用此接口停止breeze服务。

参数

无

返回值

0-成功；-1-失败

breeze_post

推送设备端状态数据至移动端，使用BLE indicate方式。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

0-成功；其他错误值-失败

breeze_post_fast

和breeze_post类似，推送设备端状态数据至移动端，区别在于使用BLE notify方式。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

0-成功；其他错误值-失败

breeze_post_ext

设备端上报带有cmd字段的数据至移动端。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。
cmd	uint8_t	推送给移动端的cmd类型。

返回值

0-成功；其他错误值-失败

breeze_append_adv_data

广播内容增加用户自定义数据。

参数

名称	类型	描述
buffer	uint8_t*	数据指针。
length	uint32_t	数据长度，byte数。

返回值

无

breeze_restart_advertising

SDK重启蓝牙广播。

参数

无

返回值

无

breeze_awss_init

该接口对蓝牙配网SDK进行初始化。在用户业务逻辑初始化阶段调用。

参数

名称	类型	描述
cb	apinfo_ready_cb	为设备完成WiFi信息（SSID、密码）获取后的回调函数，由用户定义/提供，并由SDK完成调用。
info	breeze_dev_info_t	为设备信息，包括ProductID，Product Key，Product Secret，Device Name，Device Secret等字段，由用户提供。

返回值

无

breeze_awss_end

停止蓝牙配网服务。

参数

无

返回值

无

2.9.3 蓝牙设备端SDK移植接口

Breeze SDK是阿里巴巴IoT提供的基于低功耗蓝牙协议栈上层提供的一套安全接入的解决方案，并可通过蓝牙通道支持WiFi辅助配网功能，本文对Breeze SDK的HAL接口进行说明，厂商对接时可以参考本文档。

HAL接口说明

Breeze SDK的HAL接口分为协议栈、OS、安全三个部分，这些接口的定义在$(SDK Src)/framework/bluetooth/breeze/hal目录下。

- breeze_hal_ble.h

  定义了蓝牙配网SDK对接到不同厂商蓝牙协议栈的接口，参考实现见breeze_hal_ble.c文件。
- breeze_hal_os.h

  定义了蓝牙配网SDK对接到不同OS系统的接口，基于AliOS Things的参考实现见breeze_hal_os.c，用户如果使用AliOS Things时，该部分接口不需要对接。
- breeze_hal_sec.h

  定义了蓝牙配网SDK对接到不同安全算法实现的接口，基于mbedtls的参考实现见breeze_hal_security.c文件，用户使用AliOS Things时，该部分接口不需要对接。

蓝牙协议栈HAL列表

- ble_stack_init

  该接口完成对BLE协议栈的初始化。

  参数

名称	类型	描述
ais_init	ais_bt_init_t	指定了蓝牙service和characteristics，以及其权限属性等相关设置，包括以下内容。 - uuid_svc：AIS服务的uuid（type、value）信息 - rc/wc/ic/nc/ wwnrc：AIS服务的characteristic的属性信息 - on_connected：蓝牙连接时间的回调函数 - on_disconnected：蓝牙断开连接事件的回调函数

名称	类型	描述
init_done	stack_init_done_t	协议栈初始化完成回调函数，需要在蓝牙协议栈完成之后主动调用传入成功参数AIS_ERR_SUCCESS(0)

返回值

该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_stack_deinit

  该接口负责协议栈停止和资源销毁等工作。

  参数

  无

  返回值

  该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_send_notification

  该接口通过notify方式发送数据。

  参数

名称	类型	描述
p_data	uint8_t*	发送数据的buffer地址
length	uint16_t	数据长度

  返回值

  该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_send_indication

  该接口通过indicate方式发送数据。

  参数

名称	类型	描述
p_data	uint8_t*	发送数据的buffer地址
length	uint16_t	数据长度

名称	类型	描述
txdone	callback	回调处理函数，由HAL实现代码在发送indication数据完毕后调用

返回值

该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_disconnect

  该接口断开已有的蓝牙连接。与ble_stack_deinit的区别是，后续还可以进行连接。

  参数

名称	类型	描述
reason	uint8_t	SDK断开蓝牙连接原因，注意非蓝牙spec规定的断开原因，如Remote User Terminated Connect（0x13）、Connection Accept Timeout Exceeded（0x10）等，需要在实现中对接做一次映射。例如SDK传入AIS_BT_REASON_REMOTE_USER_TERM_CONN(0x00)，在实现的时候映射成BT_HCI_ERR_REMOTE_USER_TERM_CONN(0x13)

  返回值

  无

- ble_advertising_start

  该接口开启蓝牙广播。

  参数

名称	类型	描述
adv	ais_adv_init_t*	输入参数adv指定了所需的广播信息，包括flag、设备名、厂商数据段内容

  返回值

  该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_advertising_stop

  该接口停止蓝牙广播。

  参数

  无

  返回值

  该函数成功时返回AIS_ERR_SUCCESS，错误发生时在breeze_hal_ble.h中返回对应的错误编码。

- ble_get_mac

  该接口用于获取蓝牙设备的MAC地址。

  参数

名称	类型	描述
mac	uint8_t*	用于存储获取到的蓝牙MAC地址，6字节二进制格式：0xAA，0xBB，0xCC，0xDD，0xEE，0xFF（对应MAC地址“AA:BB:CC:DD:EE:FF”）

  返回值

  该函数成功时返回AIS_ERR_SUCCESS，错误发生时返回Breeze错误编码。

OS HAL列表

OS HAL接口提供以下功能：

- 定时器支持
- 系统重启接口
- 系统时间戳获取和睡眠接口
- Key-Value键值读取接口

OS HAL接口如下。

- os_timer_new

  该接口用于创建定时器。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针

名称	类型	描述
cb	os_timer_cb_t	定时器超时回调处理函数
arg	void**	回调函数函数参数
ms	int*	超时时间

返回值

成功返回0，否则返回非0值。

- os_timer_start

  该接口用于启动定时器。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针

  返回值

  成功返回0，否则返回非0值

- os_timer_stop

  该接口用于停止定时器。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针。

  返回值

  成功返回0，否则返回非0值

- os_timer_free

  该接口用于销毁定时器资源。

  参数

名称	类型	描述
timer	os_timer_t*	定时器指针

  返回值

  成功返回0，否则返回非0值

- os_msleep

  该接口触发系统/进程进行睡眠。

  参数

名称	类型	描述
ms	int	休眠时长，单位为ms。

  返回值

  无

- os_reboot

  该接口用于重启OS。

  参数

  无

  返回值

  无

- os_now_ms

  该接口用于获取系统当前时间戳（从系统启动开始计数）。

  参数

  无

  返回值

  系统时间戳，单位为ms。

- os_kv_get

  该接口用于读取永久存储Key-Value键值。

  参数

名称	类型	描述
key	const char*	键名（字符串）
buffer	void*	键值（任意值）

名称	类型	描述
buffer_len	int*	键值长度指针

返回值

成功返回0，否则返回非0值

- os_kv_del

  该接口用于删除永久存储Key-Value键值。

  参数

名称	类型	描述
key	const char*	键名（字符串）。

  返回值

  成功返回0，否则返回非0值

- os_rand

  该接口用于产生一个整型的随机值。

  参数

  无

  返回值

  随机有符号整型值。

安全相关的HAL

- ais_aes128_init

  该接口对AES128加解密算法context进行初始化。

  参数

名称	类型	描述
key	const uint8_t*	AES128算法要求的key输入。
iv	const uint8_t*	AES128 CBC算法要求输入的iv值。

  返回值

  void指针，指向初始化后的context，后续加解密流程中使用该返回值

- ais_aes128_destroy

  销毁AES128加解密算法context，释放相关资源。

  参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。

  返回值

  成功返回0，否则返回-1

- ais_aes128_cbc_encrypt

  调用该接口进行AES128 CBC加密计算。

  参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。
src	const void*	指针指向需要加密的数据。
block_num	size_t	需要加密的数据块数（16字节为一块，不足16字节按一块计算）。
dst	void*	解密后数据的存储地址。

  返回值

  成功返回0，否则返回-1

- ais_aes128_cbc_decrypt

  调用该接口进行AES128 CBC解密计算。

  参数

名称	类型	描述
aes	void*	接口初始化之后返回的context。
src	const void*	指针指向需要加密的数据。
block_num	size_t	需要加密的数据块数（16字节为一块，不足16字节按一块计算）。

名称	类型	描述
dst	void*	解密后数据的存储地址。

返回值

成功返回0，否则返回-1

2.9.4 蓝牙设备端SDK OTA接口

Breeze协议提供基于蓝牙链路连接阿里云的安全通道和服务，Breeze OTA是在其基础上实现的BLE固件升级功能。用户移植需要对接的接口和数据结构有：上层应用接口，底层驱动对接接口和bootloader对接数据结构。

上层应用对接API

ota_breeze_service_init

```
int ota_breeze_service_init(ota_breeze_service_manage_t* ota_manage);
```

参数

名称	类型	描述
ota_manage	ota_breeze_service_manage_t*	初始化ota信息，包括版本号，消息接收回调函数等

类型介绍

```
typedef struct _ota_ble_service_dat{
    unsigned char is_ota_enable; /*0:disable ota; 1:enable ota*/
    ota_breeze_version_t verison;/*os version:used to version verify*/
    ota_breeze_get_data_t get_dat_cb;/* breeze to ota send message callback function*/
}ota_breeze_service_manage_t;
```

返回值

成功：0；失败：-1

详细内容查看ota_breeze_export.h。

底层驱动对接API

- hal_flash_erase_sector_size

```
uint32_t hal_flash_erase_sector_size(void);
```

参数

无

返回值

返回芯片flash最小擦写单元大小

- hal_flash_erase

```
int32_t hal_flash_erase(hal_partition_t pno, uint32_t off_set, uint32_t size);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
off_set	uint32_t	相对分区起始地址偏移量
size	uint32_t	擦写大小

返回值

成功：0；失败：-1

- hal_flash_read

```
int32_t hal_flash_read(hal_partition_t pno, uint32_t* poff, void* buf, uint32_t buf_size
);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
poff	uint32_t*	相对分区起始地址偏移量
buf	void*	读取的数据数组
buf_size	uint32_t	读取的数据长度

返回值

成功：0；失败：-1

- hal_flash_write

```
int32_t hal_flash_write(hal_partition_t pno, uint32_t* poff, const void* buf, uint32_t
buf_size);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID
poff	uint32_t*	相对分区起始地址偏移量
buf	const void*	写入数据数组
buf_size	uint32_t	写入数据长度

返回值

成功：0；失败：-1

- hal_flash_get_info

```
hal_logic_partition_t *hal_flash_get_info(hal_partition_t pno);
```

参数

名称	类型	描述
pno	hal_partition_t	flash 分区ID

返回值

成功：flash分区信息：分区起始地址，长度等；失败：NULL

详细内容查看flash.h。

Bootloader对接数据结构

升级完成后，系统会将固件信息等保存flash参数区，AliOS Things 保存在参数分区1；Bootloader 需要读取保存的参数，完成固件的copy和替代，实现升级。

```
typedef struct
{
    unsigned int settings_crc32;   /*settings crc32 value*/
    unsigned int ota_flag;         /*flag ota status*/
    unsigned int src_addr;         /*image storage address*/
    unsigned int dest_addr;        /*iamge run address*/
    unsigned int image_size;       /*image size*/
    unsigned int image_crc32;      /*image crc32*/
    unsigned int image_has_copy_len;   /*remember the copy len for bootloader off line*/
    unsigned char reserved_buf1[RESERVED_SIZE1]; /*reserved*/
    /*backup image include store addr, recovery bank addr, image size and image crc32*/
    unsigned int  backup_store_addr;
    unsigned int  backup_dest_addr;
```

```
    unsigned int  backup_image_size;
    unsigned int  backup_image_crc32;
    unsigned int  backup_has_copy_len;          /*remember the copy len for off line*/
    unsigned char reserved_buf2[RESERVED_SIZE2]; /*reserved*/
    /*down_loader image parameters bootloader don't need to care*/
    unsigned int break_point_offset;                /*breakpoint save when ota processing.*/
    unsigned char version_store_buf[VERSION_BUF_SIZE]; /*version save*/
    unsigned short image_info_crc16;                /*image info crc value*/
    unsigned char bin_type;                         /*Image type*/
    unsigned char reserved_buf3[RESERVED_SIZE3];
} ota_settings_t;
```

2.10 LinkVisual视频设备开发

2.10.1 LinkVisual视频设备开发概览

本文介绍了摄像头类设备如何对接LinkVisual视频服务。

LinkVisual设备端SDK依赖生活物联网平台SDK提供的基础物联网能力。您需要同时集成生活物联网平台SDK和LinkVisual设备端SDK才可完成IPC设备端的功能。设备端在对接好生活物联网平台SDK成为物联网设备后，通过集成LinkVisual SDK为设备增加视频能力。

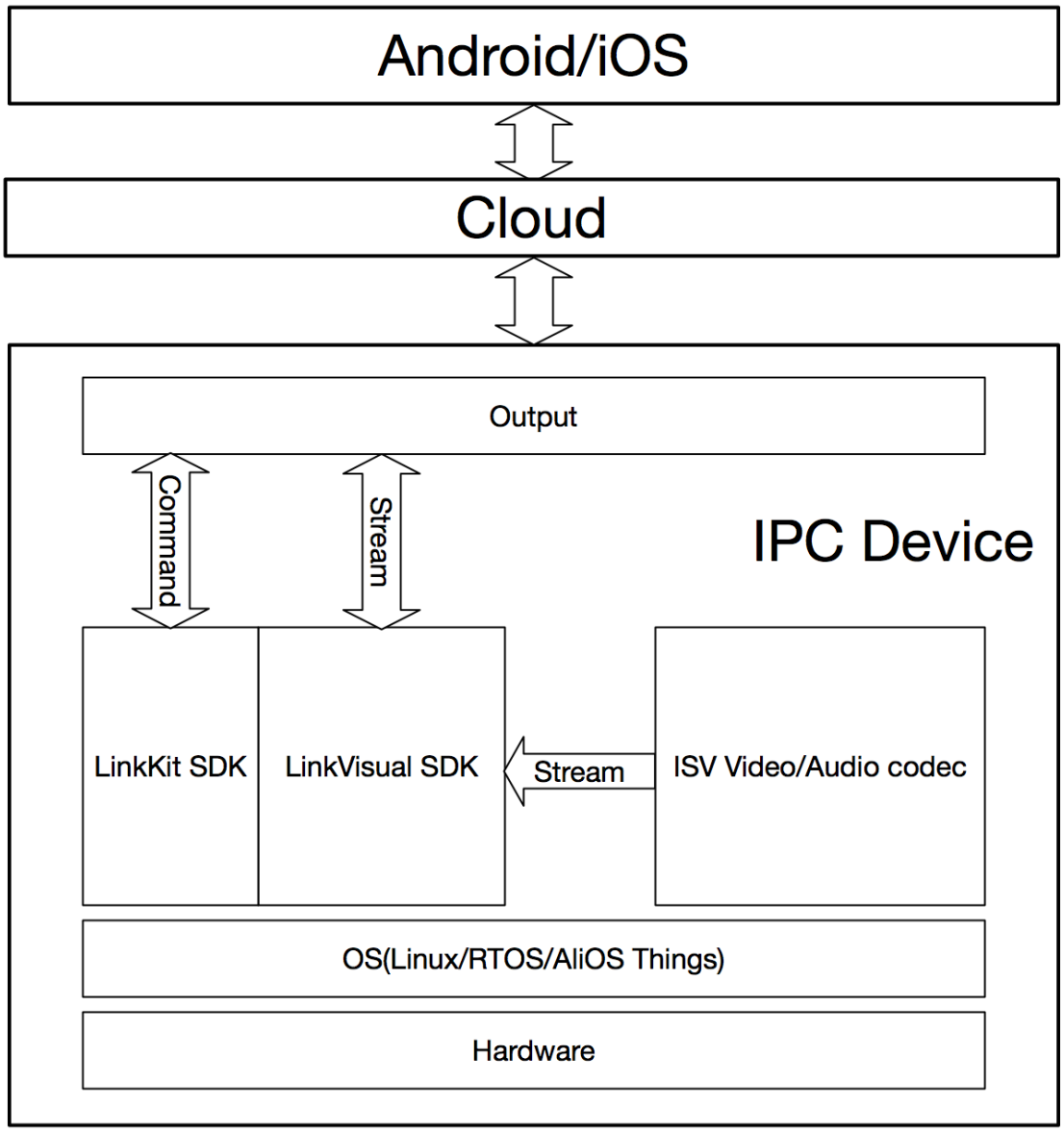
关于如何对接生活物联网平台SDK成为物联网设备的详细内容，请参见基于已认证的模组开发。

功能说明

LinkVisual设备端SDK（当前支持Linux和Android版本）提供了网络摄像机（IP Camera，简称IPC）的音视频流接入阿里云IoT平台的基础API，目前主要功能如下。

- 音视频直播服务
- 设备录像的音视频点播服务
- 图片上传服务
- 语音对讲服务
- 内置了可开关的音视频流、图片数据加密功能

生活物联网平台SDK提供了设备、云、App的物联通道，LinkVisual SDK提供了设备、云、App的音视频传输能力，设备厂商提供音视频流数据及运行环境。产品架构示意图如下图所示。



SDK获取方式相关

- LinkVisual设备端SDK（当前支持Linux和Android版本）：

  参见#unique_39获取。
- Linux版本设备端SDK：请参见基于Linux版本设备端SDK开发LinkVisual视频设备。
- Android版本设备端SDK：请参见基于Android版本设备端SDK开发LinkVisual视频设备。

版权信息

LinkVisual使用了三方**librtmp**库，相关版权信息声明如下。

```
Copyright (C) 2005-2008 Team XBMC
    http://www.xbmc.org
    Copyright (C) 2008-2009 Andrej Stepanchuk
    Copyright (C) 2009-2010 Howard Chu
 librtmp is free software; you can redistribute it and/or modify
```

```
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation; either version 2.1,
or (at your option) any later version.
librtmp is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
You should have received a copy of the GNU Lesser General Public License
along with librtmp see the file COPYING.  If not, write to
the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor,
Boston, MA  02110-1301, USA.
http://www.gnu.org/copyleft/lgpl.html
```

2.10.2 LinkVisual设备端开发-Android SDK

本章主要描述基于Android版本设备端SDK开发LinkVisual视频设备的SDK和操作流程。

设备端SDK依赖如下。

依赖SDK	概述
Link Kit Android SDK	提供设备与云端的双向数据通道能力。

初始化SDK

请在Link Kit SDK初始化完毕后对IPC SDK的初始化，需要传入设备证书信息。

```
LinkKit.getInstance().init(this, params, new ILinkKitConnectListener() {
        @Override
        public void onError(AError error) {
            Log.d(TAG,
                "onError() called with: error = [" + (error == null ? "null" : (error.getCode() +
error.getMsg()))
                    + "]");

        }

        @Override
        public void onInitDone(Object data) {
            Log.d(TAG, "onInitDone() called with: data = [" + JSON.toJSONString(data) + "]");
            // 初始化IPC SDK
            IPCDev.getInstance().init(context, your_productKey, your_deviceName,
your_deviceSecret);
        }
```

其中your_productname、your_devicename、your_devicesecret需要替换为您自己的设备证书信息。

IPC SDK需借助Linkkit的能力来完成消息监听和处理。注册监听需要两步，首先在设备服务。

```
//注册异步服务调用监听器
LinkKit.getInstance().getDeviceThing().setServiceHandler(service.getIdentifier(),
            itResRequestHandler);
//异步服务调用监听器
private ITResRequestHandler itResRequestHandler = new ITResRequestHandler() {
    @Override
    public void onProcess(String identify, Object result, ITResResponseCallback
        itResResponseCallback) {
```

```
        Log.d(TAG,
            "ITResRequestHandler  onProcess() called with: identify = [" + identify + "], result
 = ["
                + JSON.toJSONString(result) + "], itResResponseCallback = ["
                + itResResponseCallback + "]");
        /**
         * 添加IPC SDK对异步服务调用的监听
         */
        IPCDev.getInstance().notifyAsyncTopicReceived(identify, result, itResRespo
nseCallback);
    }

    @Override
    public void onSuccess(Object o, OutputParams outputParams) {
        Log.d(TAG,
            "onSuccess() called with: o = [" + JSON.toJSONString(o) + "], outputParams = [" +
JSON
                .toJSONString(outputParams) + "]");
    }

    @Override
    public void onFail(Object o, ErrorInfo errorInfo) {
        Log.d(TAG, "onFail() called with: o = [" + JSON.toJSONString(o) + "], errorInfo = [" +
JSON
            .toJSONString(errorInfo) + "]");
    }
  };
```

```
/**
 * 注册同步服务调用的监听器
 */
LinkKit.getInstance().registerOnPushListener(connectNotifyListener);
//同步服务调用监听器
private IConnectNotifyListener connectNotifyListener = new IConnectNotifyListener() {
    @Override
    public void onNotify(String connectId, String topic, AMessage aMessage) {
        /**
         * 添加IPC SDK对同步服务调用的监听
         */
        IPCDev.getInstance().notifySyncTopicReceived(connectId, topic, aMessage);

        if (CONNECT_ID.equals(connectId) && !TextUtils.isEmpty(topic) &&
            topic.startsWith("/sys/" + productKey + "/" + deviceName + "/rrpc/request")) {
            Log.d(TAG, "IConnectNotifyListener   onNotify() called with: connectId = [" +
connectId + "], topic = ["
                + topic + "], aMessage = ["
                + new String((byte[])aMessage.data) + "]");
        }
    }

    @Override
    public boolean shouldHandle(String connectId, String topic) {
        return true;
    }

    @Override
    public void onConnectStateChange(String connectId, ConnectState connectState) {

    }
```

```
};
```

混淆配置

```
# keep linkvisual

-keep class com.aliyun.iotx.linkvisualipc.** { *; }
```

依赖引入

```
// 1. 根build.gradle添加对aliyun maven仓库的引用
allprojects {
    repositories {
        maven {
            url "http://maven.aliyun.com/nexus/content/repositories/releases"
        }
    }
}

// 2. app build.gradle中添加依赖
implementation 'com.aliyun.iotx:linkvisual-ipc:1.4.0'
```

功能描述

- 存储卡录像查看：将存储在SD卡等外存中的录像文件推到服务端，支持seek到指定位置操作。
- 语音对讲：与App端建立双向语音通道，设备端采集录音并实时发送至App端，同时接收到App端发送的语音进行播放。
- 拍照：抓拍当前摄像头画面并上传至云端。
- 上报事件：上报带图片的事件给云端。

直播推流

支持RTMP推流，视频支持H264/H265，音频支持G711a以及AAC_LC格式。

您可以按以下步骤开发直播推流功能。

1. 注册直播事件监听器和流错误监听器。

   PC SDK收到服务端下发的开始推流指令后，会通过事先注册的直播流事件监听器（OnLiveStreamListener）来通知何时开始或结束推流/强制I帧等。

   推流中发生的错误也将通过流错误监听器来通知。请不要在回调接口中执行阻塞任务。

   ```
   // 设置直播流事件监听
   IPCDev.getInstance().getIpcStreamManager().setOnLiveStreamListener(MainActivity.this);
   // 设置流错误监听
   IPCDev.getInstance().getIpcStreamManager().setOnStreamErrorListener(MainActivity.this);
   ```

   ```
   public interface OnLiveStreamListener {
       /**
        * 收到开始推直播流请求
   ```

```
     *
     * @param streamId   流ID
     * @param streamType 码流类型: 0为主码流, 1为辅码流
     * @param preTimeInS 预先录制时间，单位S
     */
    void onStartPushLiveStreaming(final int streamId, final int streamType, final int
preTimeInS);

    /**
     * 收到停止推流请求
     *
     * @param streamId 流ID
     */
    void onStopPushStreaming(final int streamId);

    /**
     * 收到强制I帧请求
     * 需立即构造一个I帧并发送
     * @param streamId 流ID
     */
    void onForceIFrame(int streamId);
}
```

```
public interface OnStreamErrorListener {
    /**
     * 流异常时回调
     * @param streamId
     * @param error 参考StreamError定义
     */
    void onError(int streamId, StreamError error);
}
```

流错误码如下所示。

错误码	标志符	错误描述
1	StreamError.ERROR_STREAM_CREATE_FAILED	创建流实例失败。
2	StreamError.ERROR_STREAM_START_FAILED	开流失败。
3	StreamError.ERROR_STREAM_STOP_FAILED	停止流失败。
4	StreamError.ERROR_STREAM_SEND_VIDEO_FAILED	发送视频数据失败。
5	StreamError.ERROR_STREAM_SEND_AUDIO_FAILED	发送音频数据失败。

错误码	标志符	错误描述
6	StreamError.ERROR_STREAM_INVALID_PARAMS	无效的流参数。

RTMP错误码如下所示。

错误码	标志符	描述
-1	RTMP_ILLEGAL_INPUT	输入不合法，请检查输入参数。
-2	RTMP_MALLOC_FAILED	内存分配失败。
-3	RTMP_CONNECT_FAILED	RTMP建连失败。
-4	RTMP_IS_DISCONNECTED	RTMP连接未建立。
-5	RTMP_UNSUPPORT_FORMAT	不支持的音视频格式。
-6	RTMP_SEND_FAILED	RTMP数据包发送失败。
-7	RTMP_READ_MESSAGE_FAILED	RTMP消息读取失败。
-8	RTMP_READ_TIMESTAMP_ERROR	输入时间戳错误。

2. 处理开始直播推流请求。

当服务端下发推流请求时，回调OnLiveStreamListener.onStartPushLiveStreaming(int streamId, int streamType, int preTimeInS)方法来通知设备端需要开始采流并推流。

一般需要开启摄像头和录音机进行采流，对摄像头采集的数据调用MediaCodec进行H264编码，对录音机采集的数据进行G711a编码，提前设置对应格式的音视频参数，分别调用发送音视频的接口来持续发送采集到编码后的数据。

```
@Override
    public void onStartPushLiveStreaming(int streamId, int streamType, int preTimeInS) {
        this.streamId = streamId;
        try {

            // 构造视频参数
            VideoStreamParams videoStreamParams = new VideoStreamParams();
            // 直播流该参数始终为0
            videoStreamParams.setDurationInS(0);
            videoStreamParams.setVideoFormat(VideoStreamParams.VIDEO_FORMAT_H264
);

            // 构造音频参数
            AudioStreamParams audioStreamParams = new AudioStreamParams();
            audioStreamParams.setAudioChannel(AudioStreamParams.AUDIO_CHAN
NEL_MONO);
            audioStreamParams.setAudioFormat(AudioStreamParams.AUDIO_FORM
AT_G711A);
            audioStreamParams.setAudioEncoding(AudioStreamParams.AUDIO_ENCO
DING_16BIT);
```

```
        audioStreamParams.setAudioSampleRate(AudioStreamParams.AUDIO_SAMP
LE_RATE_8000);

        // 设置推流参数
        IPCDev.getInstance().getIpcStreamManager().setStreamParams(streamId,
videoStreamParams, audioStreamParams);

        // TODO 开始采流、编码并发送音视频数据
      } catch (NoSuchStreamException e) {
        e.printStackTrace();
      }
  }
```

发送音视频数据接口（IPCStreamManager）。

```
    /**
    * 发送音频帧数据
    *
    * @param streamId        流ID
    * @param directByteBuffer 源数据
    * @param length          数据长度
    * @param timeStampInMs    音频帧时间戳，单位ms
    */
    void sendAudioData(int streamId, ByteBuffer directByteBuffer, int length, long
timeStampInMs) throws NoSuchStreamException
   /**
    * 发送视频帧数据
    *
    * @param streamId        流ID
    * @param directByteBuffer 源数据
    * @param length          数据长度
    * @param isIFrame        是否为I帧
    * @param timeStampInMs    视频帧时间戳，单位ms
    */
   void sendVideoData(int streamId, ByteBuffer directByteBuffer, int length, boolean
isIFrame, long timeStampInMs) throws NoSuchStreamException

    /**
    * 发送音频帧数据
    *
    * @param streamId     流ID
    * @param data         源数据
    * @param offset       偏移量
    * @param length       数据长度
    * @param timeStampInMs 音频帧时间戳，单位ms
    * @deprecated 使用 {@link #sendAudioData(int, ByteBuffer, int, long)}来替换
    */
   @Deprecated
   void sendAudioData(int streamId, byte[] data, int offset, int length, long timeStampI
nMs) throws NoSuchStreamException

   /**
    * 发送视频帧数据
    *
    * @param streamId     流ID
    * @param data         源数据
    * @param offset       偏移量
    * @param length       数据长度
    * @param isIFrame     是否为I帧
    * @param timeStampInMs 视频帧时间戳，单位ms
    * @deprecated 使用 {@link #sendVideoData(int, ByteBuffer, int, boolean, long)}来替
换
    */
```

```
    @Deprecated
    public void sendVideoData(int streamId, byte[] data, int offset, int length, boolean
isIFrame, long timeStampInMs) throws NoSuchStreamException
```

H264/H265的帧结构会有一定的要求，可以以打印I帧的前256个字节进行查看，打印代码如下。

```
for (int i = 0; i < ((buffer_size > 256)?256:buffer_size); i++) {
    printf("%02x ", buffer[i]);
    if ((i + 1) % 30 == 0) {
        printf("\n");
    }
}
printf("\n");
```

H264要求I帧为：帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR。如下图所示。

```
00 00 01 67 42 c0 1e d9 00 a0 2f f9 3f f2 57 92 58 01 00 00 03 00 01 00 00 03 00 32 0f 16
2e 48 00 00 00 01 68 cb 8c b2 00 00 01 65 88 84 1f c9 81 0b 40 00 24 c7 19 c5 87 40 65 89
c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 ff ff
77 c2 a1 6e 00 16 c4 d5 6f e8 1b 0d 2c af b8 00 d1 72 9b 8d 81 28 7d 2c 8c 18 5d 50 9d 09
8e 79 94 b4 d8 9d e9 7e 0b ad 99 ca 06 a7 ff c7 84 56 15 91 7b 50 01 9f a0 8a f2 75 64 67
ab c2 b2 11 61 6f 23 00 03 4a d5 93 e4 1c 11 8b 2f f8 23 0d 26 cf d4 01 5a 2c c2 67 23 bd
1d 0e ff bf ff f0 42 5e 00 09 9a 88 33 36 eb 90 00 0f e0 33 ff 80 60 b0 42 4e 00 13 59 0a
ef 4a fc 89 bf ff 80 42 bc 30 00 4c e9 0c 89 ba e4 00 1f 40 8c 1a ff f0 f5 85 78 00 73 93
75 f4 bc fc 00 0f 17 21 9c e2 40 7f cc 3c c0
```

0x000001或者0x00000001是帧分隔符，0x67是SPS的开始，0x68是PPS的开始，0x65是IDR的开始H265要求I帧为：帧分隔符+VPS+帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR。如下图所示。

```
00 00 00 01 40 01 0c 01 ff ff 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b ac 09 00 00 00 00 01
42 01 01 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b a0 03 c0 80 10 e5 96 b9 2c ad 9a e5 51 36
00 80 00 00 00 00 00 01 44 01 c0 e3 0f 03 32 40 00 00 00 00 01 26 01 af 13 80 5b ff ff ff a9 59
16 ca fe 88 26 26 11 cc 9a 40 61 2f 2d 47 37 39 09 25 e4 6d a2 96 33 cb 99 ff c6 74 73 cd 15 0b
98 45 db fe 98 68 e2 80 bd f6 99 b9 37 c0 a3 5b 9f 38 70 82 8a 20 b8 44 a6 d7 35 35 5c 6e ed ce
25 28 10 00 88 21 0c e0 4b fd aa 47 9b a4 e7 ad 27 f0 f5 83 e3 a6 6e 89 2b 2a 7b 18 65 1d 4a 05
ff a1 fd d3 44 fe c3 34 ad ae a5 1a b5 a2 61 83 c3 ca 6a 1a c0 7b 08 44 42 f9 bf b7 ec 88 11 01
39 33 1e 96 b9 2d 91 72 63 e0 d5 e2 af a0 82 c0 46 b7 bd 43 cf f5 36 b5 f1 09 4f b6 2e e9 e3 00
```

0x000001或者0x00000001是帧分隔符，0x40是VPS的开始，0x42是SPS的开始，0x44是PPS的开始，0x26是IDR的开始。

3. 处理结束推流请求。

   当服务端下发停止推流请求时，回调OnLiveStreamListener.onStopPushLiveStreaming()方法来通知设备端停止推流。

   一般需要停止摄像头数据和录音机数据的采集，并调用IPCStreamManager的stopStreaming(int streamId)方法。

```
/**
    * 收到停止推流请求
    *
    * @param streamId 流ID
    */
```

```
    @Override
    public void onStopPushStreaming(int streamId) {
        // TODO 停止音视频数据的发送
        try {
            // 调用停止推流接口
            IPCDev.getInstance().getIpcStreamManager().stopStreaming(streamId);
        } catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

4. 处理流错误。

   推流过程中需要处理流错误，通过OnStreamErrorListener.onError(int streamId, StreamError error)来接收和处理。

点播推流

支持RTMP推流，视频支持H264/H265，音频支持G711a以及AAC_LC格式。

1. 注册点播事件监听器和流错误监听器。

   IPC SDK收到服务端下发的开始推流指令后，会通过事先注册的点播流事件监听器（OnVodStreamListener）来通知何时开始或结束推流/暂停/恢复/seek等。

   推流中发生的错误也将通过流错误监听器来通知。请不要在回调接口中执行阻塞任务。

2. 处理查询设备端录像列表请求。

   App端发起查询设备端录像列表的请求，设备端会收到同步服务调用（rrpc/request），收到查询设备录像列表请求，响应该请求将当查询范围内的文件列表返回给App端。

```
@Override
      public void onNotify(String connectId, String topic, AMessage aMessage) {
          Log.d(TAG, "onNotify() called with: connectId = [" + connectId + "], topic = [" +
topic + "], aMessage = ["
                  + new String((byte[]) aMessage.data) + "]");
          /**
           * 添加SDK的监听。
           */
          IPCDev.getInstance().notifySyncTopicReceived(connectId, topic, aMessage);

          // 处理同步服务调用
          if (CONNECT_ID.equals(connectId) && !TextUtils.isEmpty(topic) &&
                  topic.contains("rrpc")) {
              Log.d(TAG, "IConnectNotifyListener   onNotify() called with: connectId = [" +
connectId + "], topic = ["
                      + topic + "], aMessage = ["
                      + new String((byte[]) aMessage.data) + "]");

              int code = 200;
              String data = "{}";

              JSONObject json = JSON.parseObject(new String((byte[]) aMessage.data));
              if (json != null) {
                  String method = json.getString("method");
                  JSONObject params = json.getJSONObject("params");
                  switch (method) {
                      // 查询设备录像列表请求
```

```
                case "thing.service.QueryRecordList":
                    int beginTime = params.getIntValue("BeginTime");
                    int endTime = params.getIntValue("EndTime");
                    int querySize = params.getIntValue("QuerySize");
                    int type = params.getIntValue("Type");
                    appendLog("收到查询设备录像列表的请求: beginTime=" + beginTime +
                            "\tendTime=" + endTime + "\tquerySize=" + querySize + "\ttype=" +
type);

                    JSONArray resultArray = new JSONArray();
                    JSONObject item1 = new JSONObject();
                    item1.put("FileName", Base64.encode("file1".getBytes(), Base64.
DEFAULT));
                    item1.put("BeginTime", System.currentTimeMillis() / 1000 - 200);
                    item1.put("EndTime", System.currentTimeMillis() / 1000 - 100);
                    item1.put("Size", 1024000);
                    item1.put("Type", 0);
                    resultArray.add(item1);

                    JSONObject item2 = new JSONObject();
                    item2.put("FileName", Base64.encode("file2".getBytes(), Base64.
DEFAULT));
                    item2.put("BeginTime", System.currentTimeMillis() / 1000 - 100);
                    item2.put("EndTime", System.currentTimeMillis() / 1000);
                    item2.put("Size", 1024000);
                    item2.put("Type", 0);
                    resultArray.add(item2);

                    JSONObject result = new JSONObject();
                    result.put("RecordList", resultArray);

                    code = 200;
                    data = result.toJSONString();
                    break;
                default:
                    break;
            }
        }

        MqttPublishRequest request = new MqttPublishRequest();
        request.isRPC = false;
        request.topic = topic.replace("request", "response");
        String resId = topic.substring(topic.indexOf("rrpc/request/") + 13);
        request.msgId = resId;
        request.payloadObj = "{\"id\":\"" + resId + "\", \"code\":" + code + ",\"data\":"
 + data + "}";
        LinkKit.getInstance().publish(request, new IConnectSendListener() {
            @Override
            public void onResponse(ARequest aRequest, AResponse aResponse) {
                appendLog("上报成功");
            }

            @Override
            public void onFailure(ARequest aRequest, AError aError) {
                appendLog("上报失败:" + aError.toString());
            }
        });
    }
}
```

注意:

文件名需使用Base64进行编码。

3. 处理开始开始推流指令。

App端请求上一步返回的列表中某个设备录像文件后，服务端会下发推流指令，回调OnVodStreamListener.onStartPushVodStreaming(int streamId, String fileName)或OnVodStreamListener.onStartPushVodStreaming(int streamId, int beginTimeUtc, int endTimeUtc)方法来通知设备端需要将音视频数据推流。

- 按文件方式播放设备端录像的响应

```
    @Override
    public void onStartPushVodStreaming(int streamId, String fileName) {
        appendLog("开始推点播流 " + streamId + " 文件名:" + new String(Base64.decode
(fileName, Base64.NO_WRAP)));

        try {
            // 构造视频参数
            VideoStreamParams videoStreamParams = new VideoStreamParams();
            // 该视频文件的时长，单位S
            videoStreamParams.setDurationInS(H264_DURATION_IN_S);
            videoStreamParams.setVideoFormat(VideoStreamParams.VIDEO_FORM
AT_H264);

            // 构造音频参数
            AudioStreamParams audioStreamParams = new AudioStreamParams();
            audioStreamParams.setAudioChannel(AudioStreamParams.AUDIO_CHAN
NEL_MONO);
            audioStreamParams.setAudioFormat(AudioStreamParams.AUDIO_FORM
AT_G711A);
            audioStreamParams.setAudioEncoding(AudioStreamParams.AUDIO_ENCO
DING_16BIT);
            audioStreamParams.setAudioSampleRate(AudioStreamParams.AUDIO_SAMP
LE_RATE_8000);

            // 设置推流参数
            IPCDev.getInstance().getIpcStreamManager().setStreamParams(streamId,
videoStreamParams, audioStreamParams);

            // TODO 读取fileName文件，调用发送音视频数据接口进行推流
            // 文件推流完毕后应调用 IPCDev.getInstance().getIpcStreamManager().
notifyVodComplete(streamId) 通知推流完成

        } catch (NoSuchStreamException e) {
            e.printStackTrace();
        }
    }
```

- 按时间方式播放设备端录像的响应

```
    @Override
    public void onStartPushVodStreaming(int streamId, int beginTimeUtc, int
endTimeUtc) {
        appendLog("开始推点播流 " + streamId + " beginTimeUtc: "+beginTimeUtc + "
endTimeUtc:"+endTimeUtc);

        //TODO 推流逻辑需要添加:
        // 1. beginTimeUtc和endTimeUtc一般是一天的开始和结束时间
```

```
        // 2. 当收到onStartPushVodStreaming回调后，应从beginTimeUtc开始向后最近的I帧开始推流，时间戳应使用对应帧的UTC时间
        // 3. 若beginTimeUtc到endTimeUtc范围内没有录像或范围内推流已经完成了，则应调用 IPCDev.getInstance().getIpcStreamManager().notifyVodComplete(streamId) 通知推流完成
        // 4. 只要是beginTimeUtc到endTimeUtc范围内有数据, 即使跨文件，推流应该持续不断
    }
```

4. 处理暂停/恢复指令。

需响应暂停/恢复推流指令（OnVodStreamListener），相应的暂停/恢复发送音视频数据。

```
/**
 * 收到暂停推流请求
 *
 * @param streamId 流ID
 */
void onPausePushVodStreaming(int streamId);

/**
 * 收到恢复推流的请求
 *
 * @param streamId 流ID
 */
void onResumePushVodStreaming(int streamId);
```

5. 处理Seek指令。

需响应Seek指令（OnVodStreamListener），例如App端播放器进度条Seek到80秒时，对应会回调onSeekTo方法，请从该timeStampInS时间点最近的I帧开始继续推流。

```
/**
 * 收到重新定位请求
 *
 * @param streamId     流ID
 * @param timeStampInS 时间偏移量，相对于视频开始时间，单位为S
 */
void onSeekTo(int streamId, long timeStampInS);
```

6. 处理停止推流指令。

当服务端下发停止推流请求时，回调OnVodStreamListener.onStopPushLiveStreaming()方法来通知设备端停止推流。

一般要停止调用发送音视频接口、关闭视频文件，并调用IPCStreamManager的stopStreaming(int streamId)方法。

```
/**
 * 收到停止推流请求
 *
 * @param streamId 流ID
 */
@Override
public void onStopPushStreaming(int streamId) {
    // TODO 停止音视频数据的发送
    try {
        // 调用停止推流接口
        IPCDev.getInstance().getIpcStreamManager().stopStreaming(streamId);
```

```
        catch (NoSuchStreamException e) {
          e.printStackTrace();
        }
    }
```

7. 处理流错误。

   推流过程中需要处理流错误，通过OnStreamErrorListener.onError(int streamId, StreamError error)来接收和处理。

语音对讲

语音对讲支持单/双讲模式。

- 单讲：App端采集并发送音频数据到设备端进行播放。
- 双讲：App端和设备端都需要同时做采音和放音，设备端必须支持AEC，否则不建议使用该方案。

音频格式支持如下。

格式	采样率	编码	解码
G711A	8Khz/16Khz	#	#
G711U	8Khz/16Khz	#	#

说明：

对讲提供V1/V2两套接口，除特殊说明外，请使用V2版本。

- V1版本：需要自行实现录音采集、播放以及回声消除。
- V2版本：支持单/双讲模式，内部已经实现录音采集、播放以及回声消除，支持设置采集音频增益大小。

1. 注册语音对讲事件监听器和错误监听器。

   IPC SDK收到服务端下发的开始推流指令后，会通过事先注册的语音对讲事件监听器（OnLiveIntercomListener）来来通知开始/结束语音对讲、接收对端的音频参数和语音数据。

   推流中发生的错误也将通过流错误监听器来通知。请不要在回调接口中执行阻塞任务。

   选择使用的对讲版本。

```
// 设置使用对讲第二版本, 默认为此版本
IPCDev.getInstance().setLiveIntercomVersionBeforeInit(IPCDev.LiveIntercomVersion.VERSION_2);
```

   选择对讲模式。

```
// 设置双向对讲模式，只对V2版本接口有效
```

```
IPCDev.getInstance().setLiveIntercomModeBeforeInit(IPCLiveIntercomV2.LiveIntercomMode.DoubleTalk);
```

设置如下监听。

```
// 设置语音对讲事件监听
IPCDev.getInstance().getIpcLiveIntercom().setOnLiveIntercomListener(MainActivity.this);
// 设置语音对讲错误回调
IPCDev.getInstance().getIpcLiveIntercom().setOnLiveIntercomErrorListener(MainActivity.this);
```

```
public interface OnLiveIntercomListener {

    /**
     * 收到App端发起的开始语音对讲请求
     *
     * @return 返回当前设备端上行音频参数格式，如采样率、通道数、采样位宽、音频格式, 请确保对App端能支持该音频参数配置
     */
    AudioParams onStartVoiceIntercom();

    /**
     * 收到结束语音对讲请求
     */
    void onStopVoiceIntercom();

    /**
     * 收到App端音频参数，表示与App端的通道已建立，可以开始对讲
     * @param audioParams  App端的音频参数
     */
    void onAudioParamsChange(AudioParams audioParams);

    /**
     * 收到App端发送的PCM数据，一般用来做UI展示，比如绘制音量大小
     * @param buffer
     * @param size
     */
    void onAudioBufferReceive(byte[] buffer, int size);
}
```

```
public interface OnLiveIntercomErrorListener {

    /**
     * 语音对讲发生错误
     * @param error 见{@link LiveIntercomError}
     */
    void onError(LiveIntercomError error);
}
```

2. 响应开始语音对讲请求（仅V1版本接口需要处理）。

收到开始语音对讲请求后，需要启动录音机开始音频采集并将音频数据发送给对端 。

```
@Override
    public AudioParams onStartVoiceIntercom() {
        appendLog("收到开始语音对讲指令");

        // 收到开始语音对讲请求, 启动录音机
        simpleAudioRecord.setAudioRecordListener(new AudioRecordListener() {
            @Override
```

```
        public void onRecordStart() {
            // 录音开始
            appendLog("录音开始");
        }

        @Override
        public void onRecordEnd() {
            // 录音结束
            appendLog("录音结束");
        }

        @Override
        public void onBufferReceived(byte[] buffer, int offset, int size) {
            // 收到录音机抛出的PCM数据, 调用发送接口发送给对端
            IPCDev.getInstance().getIpcLiveIntercom().sendAudioBuffer(buffer, offset,
size);
        }

        @Override
        public void onError(int error, String message) {
            appendLog("录音机错误:" + error + " " + message);
        }
    });

    simpleAudioRecord.start();

    // 通知使用G711A作为音频发送格式，内部会对PCM数据重新做编码
    return AudioParams.AUDIOPARAM_MONO_8K_G711A;
}
```

发送音频数据接口（IPCLiveIntercom，仅V1版本接口需要处理）。

```
/**
 * 发送音频数据<br>
 * 等价{@link #sendAudioBuffer(byte[] data, int offset, int length, boolean
enableEncode)} enableEncode=true
 */
void sendAudioBuffer(byte[] data, int offset, int length);

/**
 * 发送音频数据<br>
 *
 * @param data         数据buffer
 * @param offset       偏移量
 * @param length       长度
 * @param enableEncode true: 内部会依据{@link OnLiveIntercomListener#onStartVoi
ceIntercom()}返回的音频格式对送入的数据（必须为PCM）重新编码并发送
 *                     false: 内部会直接发送音频数据
 */
void sendAudioBuffer(byte[] data, int offset, int length, boolean enableEncode);
```

3. 处理声音播放（仅V1版本接口需要处理）。

   当App端和设备端语音对讲通道建立后，设备端通过onAudioParamsChange(AudioParams audioParams)收到App端发过来的音频参数。依据该音频参数新建音频播放器，供后续接受的音频数据播放。

```
@Override
public void onAudioParamsChange(AudioParams audioParams) {
    // 收到对端发送的音频参数
```

```
        appendLog("收到客户端的音频参数: " + audioParams.toString());

        // 初始化播放器
        if (simpleStreamAudioTrack != null) {
            simpleStreamAudioTrack.release();
            audioTrackQueue.clear();
        }
        if (acousticEchoCanceler != null) {
            acousticEchoCanceler.release();
        }
        if (noiseSuppressor != null) {
            noiseSuppressor.release();
        }
        noiseSuppressor = NoiseSuppressor.create(simpleAudioRecord.getAudioSe
ssionId());
        if (noiseSuppressor != null) {
            noiseSuppressor.setEnabled(true);
        }
        simpleStreamAudioTrack = new SimpleStreamAudioTrack(audioParams,
AudioManager.STREAM_MUSIC, audioTrackQueue,
                simpleAudioRecord.getAudioSessionId());
        if (AcousticEchoCanceler.isAvailable()) {
            acousticEchoCanceler = AcousticEchoCanceler.create(simpleAudioRecord.
getAudioSessionId());
            if (acousticEchoCanceler != null) {
                appendLog("已开启回声消除");
                acousticEchoCanceler.setEnabled(true);
            }
        }

        simpleStreamAudioTrack.start();
    }


    @Override
    public void onAudioBufferReceive(byte[] buffer, int size) {
        // 收到对端发送的PCM数据
        audioTrackQueue.add(buffer);
    }
```

4. 处理结束语音对讲指令（仅V1版本接口需要处理）。

```
@Override
public void onStopVoiceIntercom() {
 appendLog("收到停止语音对讲指令");
 // 收到停止语音对讲请求，停止录音
 simpleAudioRecord.stop();
}
```

5. 语音对讲错误处理。

```
@Override
public void onError(LiveIntercomError error) {
 // 语音对讲发生错误
 appendLog("语音对讲错误:" + error.getCode() + " msg:" + error.getMessage());

 // 停止录音（仅V1版本接口需要处理）
 simpleAudioRecord.stop();

 // 停止播放（仅V1版本接口需要处理）
 simpleStreamAudioTrack.stop();
```

```
}
```

语音对讲错误列表如下。

错误码	错误描述
LiveIntercomError.INVALID_AUDIO_PARAMS	无效的设备端音频参数
LiveIntercomError.START_LIVE_INTERCOM_REQUEST_FAILED	无效的语音对讲请求
LiveIntercomError.CONNECTION_STREAM_FAILED	建立语音对讲流通道失败
LiveIntercomError.SEND_STREAM_DATA_FAILED	发送音频数据失败
LiveIntercomError.RECEIVE_STREAM_DATA_FAILED	接收音频数据失败
LiveIntercomError.INIT_RECORD_FAILED	录音机初始化错误
LiveIntercomError.START_RECORD_FAILED	录音机启动错误
LiveIntercomError.READ_RECORD_BUFFER_FAILED	录音机数据读取错误
LiveIntercomError.INIT_AUDIO_PLAYER_FAILED	音频播放器创建失败

图片上传

当前图片上传主要有以下两种方式。

- App端触发设备拍照，设备拍照后上传。
- 设备端触发事件主动上报事件而后进行的图片上传。

当前图片上传和事件上报中的图片上传都是由一个统一的回调IUploadPicListener通知用户进行图片上传。回调中会为用户提供上传地址，此次上传的图片ID（当此次上传图片为事件上传图片，则此ID为上报事件时上报的图片ID）以及上传类型，用户根据这些信息自行进行图片上传，待上传完成后用户通过IUploadPicProcessCallback回调通知SDK上传任务完成。

说明：

需要确认一下拍照的物模型**TriggerPicCapture**里的UploadUrl字符串长度，如果小于512字符，请增加长度到512字符以上，否则图片上传会受到影响。

1. 拍照上传。

   拍照上传任务大体流程依次是SDK通知设备拍照、设备拍照、设备上传图片、通知SDK上传结果。

   具体的流程如下。

   a. 设备端注册上传图片监听。

   b. 当App触发拍照，SDK通过IUploadPicListener的onUpload通知用户进行拍照以及上传。

   c. 设备端通过onUpload中的UploadInfo获取此次拍照上传的URL。

   d. 设备端自行拍照，并通过URL上传图片。

   e. 设备端通过IUploadPicProcessCallback回调通知SDK上传结果。

```
/**
 * 上传图片监听
 */
public interface IUploadPicListener {
    /**
     * 触发上传
     * @param uploadInfo  上传信息
     * @param callback    上传传结果回调
     */
    void onUpload(UploadInfo uploadInfo, IUploadPicProcessCallback callback);
}
```

```
/**
 * 上传图片
 */
public class UploadInfo {
    /**
     * 上传url
     */
    private String uploadUrl;
    /**
     * 上传图片ID（用户生成的，只有事件上报时才有）
     */
    private String picId;
    /**
     * 上传类型，0为拍照，1为事件图片上传。
     */
    private int type;
}
```

```
/**
 * 上传图片结果回调
 */
public interface IUploadPicProcessCallback {
    /**
     * 拍照任务成功
     */
    void onSucceed();

    /**
     * 上传任务失败
     *
     * @param errorMsg 错误信息
     */
    void onFailed(String errorMsg);
```

```
}
```

用户通过如下代码进行注册监听，LinkKit SDK初始化成功后进行注册。注册代码如下。

```
//注册上传图片监听，当App端触发拍照或者设备上报了事件以后，会在注册的回调中收到上传的回调。
IPCDev.getInstance().registerUploadPicListener(uploadPicListener);

/**
 * 上传图片回调，回调中只有是事件照片上传的时候picId不为空且为上报事件时用户上传的ID。
 * type为上传类型，0为App触发设备拍照，1为设备上传事件照片。
 */
private IUploadPicListener uploadPicListener = new IUploadPicListener() {
    @Override
    public void onUpload(final UploadInfo uploadInfo, final IUploadPicProcessCallback callback) {
        final String url = uploadInfo.getUploadUrl();
        final String picId = uploadInfo.getPicId();
        final int type = uploadInfo.getType();
        switch (type) {
            //拍照上报
            case 0:
                //拍照并且上传
                upLoadFile(url, pic, new CallBack(){
                    @Override
                    public void onSucceed() {
                        Log.d(TAG, "uploadImage onSucceed");
                        if (callback != null) {
                            callback.onSucceed();
                        }
                    }

                    @Override
                    public void onFailed(String error) {
                        Log.e(TAG, "uploadImage onFailed:" + error);
                        if (callback != null) {
                            callback.onFailed(error);
                        }
                    }
                });
                break;
            //事件图片上报
            case 1:
                //上传图片
                upLoadFile(url, pic, new CallBack(){
                    @Override
                    public void onSucceed() {
                        Log.d(TAG, "uploadImage onSucceed");
                        if (callback != null) {
                            callback.onSucceed();
                        }
                    }

                    @Override
                    public void onFailed(String error) {
                        Log.e(TAG, "uploadImage onFailed:" + error);
                        if (callback != null) {
                            callback.onFailed(error);
                        }
                    }
                });
```

```
                break;
            default:
                break;
        }
    }};
```

2. 事件上报。

   事件上报大体流程依次是设备触发事件、事件上报、SDK通知设备上传图片、设备上传图片、通知SDK上传结果。

   具体的流程：设备端注册上传图片监听，当设备端触发事件，设备通过SDK提供的接口进行事件上报（需要上传图片ID），SDK会通过IUploadPicListener的onUpload通知用户进行图片上传，设备端能够通过onUpload中的UploadInfo获取此次上传的URL以及图片ID，设备端自行通过URL上传图片，然后通过IUploadPicProcessCallback回调通知SDK上传结果。

   上报事件接口实例如下。

```
//上报报警事件分成两步：1.上传报警，2.上传图片
//上报报警事件需要用户生成一个报警图片ID，这个报警图片ID要求是设备唯一的图片ID，建议用Unix时间戳。
// 不过这边需要确认一下拍照的物模型"TriggerPicCapture"里的"UploadUrl"的字符串长度，如果小于512请增加长度到512以上。
// 上报事件以后会收到IUploadPicListener（需要注册监听）内的onUpload回调，然后请将对应的图片上传到回调中url指定地址。
final String alarmPicId = String.valueOf(System.currentTimeMillis() / 1000);
IPCDev.getInstance().reportAlarmEvent(alarmPicId, 1, new ReportAlarmEventListener() {
    @Override
    public void onSucceed() {
        Log.d(TAG, " report onSucceed  alarmPicId:" + alarmPicId);
        appendLog("上报事件成功");
    }

    @Override
    public void onFailed(String msg) {
        Log.e(TAG, " report onFailed  e:" + msg);
        appendLog("上报事件失败  e:" + msg);
    }
});
```

2.10.3 Link Visual设备端开发-Linux SDK

Link Visual设备端SDK提供了网络摄像机（IPC）接入阿里云视频边缘智能服务的基础API，主要包括实时视频直播相关的接口、设备本地文件点播相关的接口、报警事件上报和图片上传的接口以及其他IPC控制相关的接口等。通过这些接口，可以实现物联网设备视频直播、视频点播、图片上传等功能。

限制条件

Link Visual设备端SDK的资源占用、平台支持和运行依赖如下所示。

- 资源占用情况
  - RAM：1 MB的码流，预计占用500 KB的RAM内存。
  - ROM：占用2 MB的ROM存储。
- 平台支持

  目前只能在支持C++11标准（GCC版本4.8.1以上）的Linux平台中使用。
- 运行依赖

  依赖阿里云生活物联网SDK。生活物联网平台SDK提供了物联网通道能力，开发者需要同时获取Link Visual SDK和生活物联网平台SDK来完成设备的接入。

  生活物联网平台SDK主要提供物联网控制通道能力，包括长连接、消息通知、事件上报等，Link Visual SDK主要提供音视频流通道能力。Link Visual响应生活物联网平台SDK的控制消息，进行流媒体业务处理。

API列表

- SDK生命周期

功能说明	对应接口
SDK初始化	lv_init
SDK停止	lv_destroy

- 直播及点播服务

功能说明	对应接口
通知直播、点播服务已开启	lv_start_push_streaming_cb
通知直播、点播服务已结束	lv_stop_push_streaming_cb
推送视音频配置参数	lv_stream_send_config
推送视频数据	lv_stream_send_video
推送音频数据	lv_stream_send_audio
推流过程中命令控制（暂停等）	lv_on_push_streaming_cmd_cb
点播的文件列表查询	lv_query_storage_record_cb
点播的文件列表按月查询	lv_query_storage_record_by_month_cb
录像文件播放结束	lv_post_file_close
预录数据结束	lv_post_pre_complete

- 图片服务

功能说明	对应接口
图片上传	lv_post_alarm_image
通知上传图片	lv_trigger_pic_capture_cb

- 语音对讲服务

功能说明	对应接口
通知开启服务	lv_start_voice_intercom_cb
通知结束服务	lv_stop_voice_intercom_cb
开启服务	lv_voice_intercom_start_service
停止服务	lv_voice_intercom_stop_service
发送音频	lv_voice_intercom_send_audio
接收音频	lv_voice_intercom_receive_data_cb
接收音频参数配置	lv_voice_intercom_receive_metadata_cb

- 其他功能

功能说明	对应接口
生活物联网平台SDK消息注入	lv_linkkit_adapter
SDK功能动态控制	lv_control

- 更多Linux SDK API介绍，参见Link Visual设备端开发-Linux SDK API。

体验Link Visual Demo

体验Link Visual Demo前，请查阅以下Demo说明，且已在生活物联网的控制台上创建产品，并完成产品的开发工作，请参见#unique_43。

- 生活物联网平台提供了在x86 64位Ubuntu 16.04系统上编译的Link Visual Demo供您快速体验。在其他Linux版本上尚未验证过，推荐您安装与阿里云一致的发行版以免出现兼容性问题。
- 生活物联网平台也提供在内置了Link Visual Demo的Docker镜像供您快速体验。Docker镜像提供更好的跨平台能力，您可以在Windows/Mac/Linux等已安装有Docker软件的系统上运行Demo。

1. 下载Link Visual的Demo。

   - 基于Ubuntu的Link Visual Demo
   - 基于Docker的Link Visual Demo

   说明:

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

2. 运行Demo程序。

   - Ubuntu 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_devic
     e_secret
     ```

   - Docker 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_devic
     e_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，查看Demo的日志信息。

获取SDK

Link Visual设备端SDK以静态库的形式提供，支持通过编译SDK接入不同芯片的设备，需要您发送邮件申请。

请按以下模板发送邮件至ray.wl@antfin.com联系我们获取Link Visual设备端SDK。收到邮件后，我们会及时联系您。

- 邮件主题：获取固件升级SDK和操作说明文档
- 邮件内容：

```
公司名称：
公司地址：
联系人：
联系电话：
应用场景描述：
```

2.10.4 Link Visual设备端开发-Linux SDK API

本章详细描述Linux版本Link Visual设备端SDK相关API。

SDK生命周期

- **lv_init**

  接口详情：int lv_init(const lv_config_s *config)

  SDK初始化。

  请求参数说明如下。

参数	类型	说明
config	lv_config_s*	配置参数结构体。

  示例代码如下。

```
//新建一个配置结构体，并置空
lv_config_s config;
memset(&config, 0, sizeof(lv_config_s));

//以下是设置具体的配置属性
//设备三元组
memcpy(config.product_key, product_key.c_str(), product_key.length());
```

```
memcpy(config.device_name, device_name.c_str(), device_name.length());
memcpy(config.device_secret, device_secret.c_str(), device_secret.length());

//SDK的日志配置
config.log_level = LV_LOG_DEBUG;
config.log_dest = LV_LOG_DESTINATION_STDOUT;
//config.log_dest_file;

//音视频推流服务
config.start_push_streaming_cb = startPushStreamingCallback;
config.stop_push_streaming_cb = stopPushStreamingCallback;
config.on_push_streaming_cmd_cb = onPushStreamingCmdCb;

//语音对讲服务
config.start_voice_intercom_cb = startVoiceIntercomCallback;
config.stop_voice_intercom_cb = stopVoiceIntercomCallback;
config.voice_intercom_receive_metadata_cb = voice_intercom_receive_metadata_cb;
config.voice_intercom_receive_data_cb = VoiceIntercomReceiveDataCallback;

//获取存储录像录像列表
config.query_storage_record_cb = queryStorageRecordCallback;

//触发设备抓图
config.trigger_pic_capture_cb = triggerPicCaptureCallback;

//触发设备录像
config.trigger_record_cb = triggerRecordCallback;

//初始化SDK，失败则退出
int ret = lv_init(&config);
if (ret < 0) {
    return -1;
}
```

- **lv_destroy**

  接口详情：`int lv_destroy(void);`

  SDK销毁。

  无请求参数。

  示例代码如下。

  ```
  //运行结束后
  ```

```
lv_destroy();
```

- 关于接口调用的其他说明如下。
  - lv_init会打印相关版本号信息，获取技术支持时请提供该版本信息，方便追溯问题。
  - lv_init设备初始化不会因无网络而失败，有可能是入参不正确或者资源分配失败，请您确认入参和资源分配。
  - lv_init注册了大量的回调函数，这些回调函数共用一个消息队列线程，因此一个回调消息的阻塞将导致后续消息的延迟，一个回调的永久阻塞将导致所有回调无法抛出。建议您不要在回调中做过于耗时的操作。
  - lv_init定义了日志类型log_level，建议对接初期将日志类型设置为LV_LOG_DEBUG。
  - 推荐lv_init在IOT_Linkkit_Connect成功后调用。
  - 推荐lv_destroy在IOT_Linkkit_Close调用之前调用。

Link kit消息注入

Link Visual需要接受Link Kit下发的部分消息并代为处理，处理用接口为**lv_linkkit_adapter**，已被Link Visual处理过的消息不再需要您处理。

- **lv_linkkit_adapter**

  接口详情：`int lvlinkkit_adapter(lv_linkkit_adapter_type_s type, const lv_linkkit_adapter_property_s *in);`

  SDK初始化。

  请求参数说明如下。

参数	类型	说明
type	lv_linkkit_adapter_type_s	Link Kit的消息适配类型，目前有Link Kit物模型服务类消息和Link Visual自定义消息。
in	const lv_linkkit_adapter_property_s*	Link Kit的消息结构体。

  示例代码如下。

```
IOT_RegisterCallback(ITE_LINK_VISUAL, user_link_visual_handler);//订阅了Link Visual自定义消息回调
```

```
IOT_RegisterCallback(ITE_SERVICE_REQUST, user_service_request_event_handler);//订阅了Linkkit物模型服务类消息回调
```

- **ITE_LINK_VISUAL**

  ITE_LINK_VISUAL是Link Visual自定义的消息。由于Link Kit v2.3.0接口暂不支持自定义消息，因此Link Visual修改了Link Kit部分代码，非Link Visual专用的Link Kit版本将无法使用该功能。

  自定义消息全部由Link Visual处理业务逻辑，示例如下。

```
//user_link_visual_handler为IOT_RegisterCallback使用ITE_LINK_VISUAL订阅的回调，
//在回调中，所有消息都使用lv_linkkit_adapter注入
static int user_link_visual_handler(const int devid, const char *payload,
                                    const int payload_len)
{
    //Link Visual自定义的消息，直接全交由Link Visual来处理
    if (payload == nullptr || payload_len == 0) {
        return 0;
    }

    lv_linkkit_adapter_property_s in = {0};
    in.request = payload;
    in.request_len = payload_len;
    int ret = lv_linkkit_adapter(LV_LINKKIT_ADAPTER_TYPE_LINK_VISUAL, &in);
    if (ret < 0) {
        EXAMPLE_TRACE("LinkVisual process service request failed, ret = %d", ret);
        return -1;
    }

    return 0;
}
```

- **ITE_SERVICE_REQUST**

  ITE_SERVICE_REQUST是Link Kit定义的物模型服务类消息，部分物模型消息需要由Link Visual处理业务逻辑，此类消息需要使用**lv_linkkit_adapter**处理。

```
//Link Visual代为处理的服务类消息
static std::vector<std::string> link_visual_service = {
    "TriggerPicCapture",//触发设备抓图
    "StartVoiceIntercom",//开始语音对讲
    "StopVoiceIntercom",//停止语音对讲
    "StartVod",//开始录像观看
    "StartVodByTime",//开始录像按时间观看
    "StopVod",//停止录像观看
    "QueryRecordList",//查询录像列表
    "StartP2PStreaming",//开始P2P直播
    "StartPushStreaming",//开始直播
    "StopPushStreaming",//停止直播
};

//user_service_request_event_handler为IOT_RegisterCallback使用ITE_SERVICE_REQUST订阅的回调，
//在回调中，判断当前消息是否需要Link Visual处理，如需要，则使用lv_linkkit_adapter注入
static int user_service_request_event_handler(const int devid, const char *id, const int id_len,
                                              const char *serviceid, const int serviceid_len,
                                              const char *request, const int request_len,
```

```
                                char **response, int *response_len)
{
    /* 打印方式1 */
   printf("Service Request Received, Devid: %d, ID %.*s, Service ID: %.*s, Payload: %s \
n",
                devid, id_len, id, serviceid_len, serviceid, request);
   /* 打印方式2 */
   printf("Service Request Received, Devid: %d, ID %s, Service ID: %s, Payload: %s \n",
                devid, id, serviceid, request);
    bool link_visual_process = false;
    //判断当前serviceid是否是Link Visual需要处理的消息
   for (int i = 0; i < link_visual_service.size(); i++) {
      //这里需要根据字符串的长度来判断
      if (!strncmp(serviceid, link_visual_service[i].c_str(), link_visual_service[i].length()))
{
         link_visual_process = true;
         break;
      }
   }

     //根据前文的判断，将需要Link Visual处理的消息使用lv_linkkit_adapter注入
   if (link_visual_process) {
      //ISV将某些服务类消息交由Link Visual来处理，不需要处理response
      lv_linkkit_adapter_property_s in = {0};
      in.dev_id = devid;
      in.id = id;
      in.id_len = id_len;
      in.service_id = serviceid;
      in.service_id_len = serviceid_len;
      in.request = request;
      in.request_len = request_len;
      int ret = lv_linkkit_adapter(LV_LINKKIT_ADAPTER_TYPE_TSL_SERVICE, &in);
      if (ret < 0) {
         EXAMPLE_TRACE("LinkVisual process service request failed, ret = %d", ret);
         return -1;
      }
   } else {
      //这里给一个非Link Visual处理的消息示例
      if (!strncmp(serviceid, "PTZActionControl", (serviceid_len > 0)?serviceid_len:0)) {
         cJSON *root = cJSON_Parse(request);
         if (root == NULL) {
            EXAMPLE_TRACE("JSON Parse Error");
            return -1;
         }

         int action_type = 0;
         cJSON *child = cJSON_GetObjectItem(root, "ActionType");
         if (!child) {
            EXAMPLE_TRACE("JSON Parse Error");
            cJSON_Delete(root);
            return -1;
         }
         action_type = child->valueint;

         int step = 0;
         child = cJSON_GetObjectItem(root, "Step");
         if (!child) {
            EXAMPLE_TRACE("JSON Parse Error");
            cJSON_Delete(root);
            return -1;
         }
         step = child->valueint;

         cJSON_Delete(root);
```

```
            EXAMPLE_TRACE("PTZActionControl %d %d", action_type, step);
        }
    }

    return 0;
}
```

- PTZ功能

  Link Visual定义了PTZ属性，但是这些PTZ消息，需要您自行处理。

  ITE_SERVICE_REQUST的示例代码中已经给出了PTZ的解析示例代码，此处补充PTZ的定义。

PTZActionControl	物模型服务类标识符	说明
ActionType	动作类型	- 0：左 - 1：右 - 2：上 - 3：下 - 4：上左 - 5：上右 - 6：下左 - 7：下右 - 8： 放大 - 9：缩小
Step	步长量	- 取值范围：0 ~ 10 - 步长：1

- 关于接口调用的其他说明如下。
  - 任何一个自定义消息或者物模型消息缺失，都将影响SDK的功能。若您不使用Demo中的示例代码，请确保自写代码中的消息成功注入了lv_linkkit_adapter。
  - 通过控制台的设备调试功能，云端可下发测试数据至设备，您可通过此方式测试注册消息。
  - Link Kit的IOT_RegisterCallback注册的回调，共用一个消息队列线程，因此一个回调消息的阻塞将导致后续消息的延迟，一个回调的永久阻塞将导致所有回调无法抛出。建议您不要在回调中做过于耗时的操作。
  - 在Link Kit注册IOT_RegisterCallback的ITE_SERVICE_REQUST回调中，id、serviceid、request指向同一个字符串的不同位置（对比示例代码中打印方式1和打印方式2），因此对serviceid的比较时，需要带 serviceid_len。
  - PTZ功能中，单位步长量在设备的具体步长大小可由您自行定义。实现了画面翻转功能后，您需要翻转PTZ方向的调整。

直播和卡录像点播

- **lv_start_push_streaming_cb**

  接口详情：typedef int (lv_start_push_streaming_cb)(int service_id, lv_stream_type_e type , const lv_stream_param_s *param)

  回调函数，通知直播或者卡录像点播链路已经建立成功，并附带一些配置信息。在此回调收到后，开发者应该根据配置信息初始化编码器，并开始推送音视频数据。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
type	lv_stream_type_e	链路的类型，分为直播和卡录像点播。
param	const lv_stream_param_s*	链路的参数，例如卡录像要点播的文件名称。

  示例代码如下。

```
//demo，定义了回调函数startPushStreamingCallback作为lv_start_push_streaming_cb的实现
lv_start_push_streaming_cb = startPushStreamingCallback;

//demo,回调函数startPushStreamingCallback
int startPushStreamingCallback(int service_id, lv_stream_type_e cmd_type, const lv_stream_param_s *param)
{
      //忽略不合法的service_id
      if (service_id < 0) {
      return -1;
   }

   if (cmd_type == LV_STREAM_CMD_LIVE) {
        //使用lv_stream_send_video/lv_stream_send_audio推送音视频数据；
        //实际使用中建议新建线程进行数据发送
       ......
       return 0;
   } else if (cmd_type == LV_STREAM_CMD_STORAGE_RECORD_BY_FILE) {
     //使用lv_stream_send_video/lv_stream_send_audio推送音视频数据
     //实际使用中建议新建线程进行数据发送
     ......
     return 0;
   }

   return 0;
```

```
}
```

- **lv_stop_push_streaming_cb**

  接口详情：`typedef int (*lv_stop_push_streaming_cb)(int service_id)`

  回调函数，通知直播或者卡录像点播链路已经断开。在此回调收到后，您应该停止推送音视频数据。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

  示例代码如下。

```
//demo，定义了回调函数stopPushStreamingCallback作为lv_start_push_streaming_cb的实现
lv_stop_push_streaming_cb = stopPushStreamingCallback;

//demo,回调函数stopPushStreamingCallback
int stopPushStreamingCallback(int service_id)
{
      //忽略不合法的service_id
   if (service_id < 0) {
      return -1;
   }

   if (service_id == g_live_service_id) {
      //您停止推流
   } else if (service_id == g_storage_record_service_id) {
     //您停止推流
   }

   return 0;
}
```

- **lv_on_push_streaming_cmd_cb**

  接口详情：`typedef int (*lv_on_push_streaming_cmd_cb)(int service_id, lv_on_push_streaming_cmd_type_e cmd, unsigned int timestamp_ms)`

  回调函数，直播、卡录像点播链接建立期间，进行一些命令控制，例如要求发送I帧命令。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
cmd	lv_on_push_streaming_cmd_type_e	控制命令，目前有开始播放、停止播放、暂停播放、恢复播放、定位、强制I帧。

参数	类型	说明
timestamp_ms	unsigned int	卡录像点播的定位的时间戳。

示例代码如下。

```
//demo，定义了回调函数stopPushStreamingCallback作为lv_start_push_streaming_cb的实现
on_push_streaming_cmd_cb = onPushStreamingCmdCb;

int onPushStreamingCmdCb(int service_id, lv_on_push_streaming_cmd_type_e cmd, unsigned int timestamp_ms)
{
    if (cmd == LV_STORAGE_RECORD_START) {
        //卡录像开始推流
    } else if (cmd == LV_STORAGE_RECORD_STOP) {
        //卡录像停止推流
    } else if (cmd == LV_STORAGE_RECORD_PAUSE) {
        //卡录像暂停推流
    } else if (cmd == LV_STORAGE_RECORD_UNPAUSE) {
        //卡录像恢复推流
    } else if (cmd == LV_STORAGE_RECORD_SEEK) {
        //卡录像定位到某一段
    } else if (cmd == LV_LIVE_REQUEST_I_FRAME) {
        //对于直播，需要强制生成一个I帧
    }

    return 0;
}
```

- **lv_stream_send_config**

  接口详情：`int lv_stream_send_config(int service_id, unsigned int bitrate_kbps, double duration, const lv_video_param_s *video_param, const lv_audio_param_s *audio_param )`

  发送音视频的配置。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
bitrate_kbps	unsigned int	链路的码流，设置的码流越大，内部音视频数据缓冲区越大。
duration	double	文件时长，卡录像点播有效。
video_param	const lv_video_param_s*	视频的参数集。

参数	类型	说明
audio_param	const lv_audio_param_s*	音频的参数集。

示例代码如下。

```
lv_video_param_s video_param;
memset(&video_param, 0, sizeof(lv_video_param_s));
video_param.format = LV_VIDEO_FORMAT_H264;
video_param.fps = 25;

lv_audio_param_s audio_param;
memset(&audio_param, 0, sizeof(lv_audio_param_s));
audio_param.format = LV_AUDIO_FORMAT_G711A;
audio_param.channel = LV_AUDIO_CHANNEL_MONO;
audio_param.sample_bits = LV_AUDIO_SAMPLE_BITS_16BIT;
audio_param.sample_rate = LV_AUDIO_SAMPLE_RATE_8000;

lv_stream_send_config(service_id, 1000, 0, &video_param, &audio_param);
```

- **lv_stream_send_video**

  接口详情：int lv_stream_send_video(int service_id, unsigned char* buffer, unsigned int buffer_len, bool key_frame, unsigned int timestamp_ms)

  视频数据发送。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
buffer	unsigned char*	视频的数据。
buffer_len	unsigned int	视频的数据长度。
key_frame	bool	是否为关键帧。I帧为关键帧，P帧为非关键帧，不接受B帧。
timestamp_ms	unsigned int	视频的时间戳。

  示例代码如下。

```
//demo，这个函数回调输入音频帧数据
void handleVideoBuffer(unsigned char *buffer, unsigned int buffer_size,
   unsigned int present_time, int nal_type, int service_id)
{
    if (nal_type == 5) {
      lv_stream_send_video(service_id, buffer, buffer_size, 1, present_time);
   } else if (nal_type == 1) {
      lv_stream_send_video(service_id, buffer, buffer_size, 0, present_time);
   }
```

```
}
```

- **lv_stream_send_audio**

  接口详情：int lv_stream_send_audio(int service_id, unsigned char* buffer, unsigned int buffer_len, unsigned int timestamp_ms)

  音频数据发送。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
buffer	unsigned char*	音频的数据。
buffer_len	unsigned int	音频的数据长度。
timestamp_ms	unsigned int	音频的时间戳。

  示例代码如下。

```
//demo，这个函数回调输入音频帧数据
void handleAudioBuffer(unsigned char *buffer, unsigned int buffer_size,
      unsigned int present_time, int service_id)
{
   lv_stream_send_audio(service_id, buffer, buffer_size, present_time);
}
```

- **lv_query_storage_record_cb**

  接口详情：typedef void (lv_query_storage_record_cb)(unsigned int start_time, unsigned int stop_time, int num, const char *id, int (on_complete)(int num, const char *id, const lv_query_storage_record_response_s *response))

  回调函数，查询卡录像的文件信息。

  请求参数说明如下。

参数	类型	说明
start_time	unsigned int	查询录像的开始时间。
stop_time	unsigned int	查询录像的结束时间。
num	int	录像数量小于等于0的时候表示已查询全部的录像。
id	const char *	查询的会话ID，需要回传。

参数	类型	说明
on_complete	函数指针	将查询结果通过该函数同步返回。

示例代码如下。

```
//demo，该函数是lv_query_storage_record_cb的实际实现
void queryStorageRecordCallback(unsigned int start_time,
                                unsigned int stop_time,
                                int num,
                                const char *id,
                                int (*on_complete)(int num, const char *id, const lv_query_s
torage_record_response_s *response))
{
    DEMO_TRACE("start_time:%d stop_time:%d num:%d", start_time, stop_time, num);

    if (num <= 0) {
        return ;
    }
    int answer_num = (num > 1)?(num - 1):1;

    static int index = 0;
    auto *response =  new lv_query_storage_record_response_s[num];
    memset(response, 0, sizeof(lv_query_storage_record_response_s) * answer_num);
    for (int i = 0; i < answer_num; i ++) {
        response[i].file_size = 15320990;
        response[i].record_type = (lv_storage_record_type_e)(i % 3);
        response[i].start_time = (int)start_time + index * 60;
        response[i].stop_time = (int)start_time + index * 60 + 59;
        snprintf(response[i].file_name, 64, "%d.mp4", index);//注意不要溢出
        index ++;
    }
    int result = on_complete(answer_num, id, response);
    DEMO_TRACE("result:%d", result);
    delete [] response;
}
```

- **lv_stream_force_stop**

  接口详情：`int lv_stream_force_stop(int service_id)`

  强制断开一路直播、卡录像点播。您在自身的功能出错等无法继续正常往一路流发送数据时，自行断开连接。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

  示例代码如下。

```
//demo，在收到推流通知的回调中，直接强制关闭该路链接
int startPushStreamingCallback(int service_id, lv_stream_type_e cmd_type, const
lv_stream_param_s *param)
{
    DEMO_TRACE("startPushStreamingCallback:%d %d", service_id, cmd_type);
```

```
    if (service_id < 0) {
        return -1;
    }
     lv_stream_force_stop(service_id);
}
```

- 关于接口调用的其他说明如下。
  - 强制I帧命令发出时，为了保证尽可能快速的发出I帧，SDK内部将清空缓冲区，并不再接受音频数据和视频的P帧数据，直到I帧到达。
  - 强制I帧命令发出时，为了保证尽可能快速的发出I帧，您应尽可能快的产生I帧。若300毫秒（ms）内未收到I帧，SDK会重发若干次强制I帧请求。
  - 强制I帧命令发出时，您需要重新调用**lv_stream_send_config**发送流配置。
  - 进行卡录像的定位操作时，为了尽可能快速的显示定位后的数据，您在定位操作后，应该尽可能快速的发出I帧，同时SDK在定位操作后也不在接收音频数据和视频的P帧数据，直到I帧到达。
  - H264/H265的帧结构会有一定的要求，可以打印I帧的前256个字节进行查看，打印代码如下。

    ```
    for (int i = 0; i < ((buffer_size > 256)?256:buffer_size); i++) {
        printf("%02x ", buffer[i]);
        if ((i + 1) % 30 == 0) {
            printf("\n");
        }
    }
    ```

```
printf("\n");
```

H264要求I帧为：帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR。如下图所示。

```
00 00 01 67 42 c0 1e d9 00 a0 2f f9 3f f2 57 92 58 01 00 00 03 00 01 00 00 03 00 32 0f 16
2e 48 00 00 00 01 68 cb 8c b2 00 00 01 65 88 84 1f c9 81 0b 40 00 24 c7 19 c5 87 40 65 89
c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 c9 ff ff
77 c2 a1 6e 00 16 c4 d5 6f e8 1b 0d 2c af b8 00 d1 72 9b 8d 81 28 7d 2c 8c 18 5d 50 9d 09
8e 79 94 b4 d8 9d e9 7e 0b ad 99 ca 06 a7 ff c7 84 56 15 91 7b 50 01 9f a0 8a f2 75 64 67
ab c2 b2 11 61 6f 23 00 03 4a d5 93 e4 1c 11 8b 2f f8 23 0d 26 cf d4 01 5a 2c c2 67 23 bd
1d 0e ff bf ff f0 42 5e 00 09 9a 88 33 36 eb 90 00 0f e0 33 ff 80 60 b0 42 4e 00 13 59 0a
ef 4a fc 89 bf ff 80 42 bc 30 00 4c e9 0c 89 ba e4 00 1f 40 8c 1a ff f0 f5 85 78 00 73 93
75 f4 bc fc 00 0f 17 21 9c e2 40 7f cc 3c c0
```

0x000001或者0x00000001是帧分隔符，0x67是SPS的开始，0x68是PPS的开始，0x65是IDR的开始。

H265要求I帧为：帧分隔符+VPS+帧分隔符+SPS+帧分隔符+PPS+帧分隔符+IDR，如下图所示。

```
00 00 00 01 40 01 0c 01 ff ff 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b ac 09 00 00 00 00 01
42 01 01 01 40 00 00 03 00 80 00 00 03 00 00 03 00 7b a0 03 c0 80 10 e5 96 b9 2c ad 9a e5 51 36
00 80 00 00 00 00 00 01 44 01 c0 e3 0f 03 32 40 00 00 00 00 01 26 01 af 13 80 5b ff ff ff a9 59
16 ca fe 88 26 26 11 cc 9a 40 61 2f 2d 47 37 39 09 25 e4 6d a2 96 33 cb 99 ff c6 74 73 cd 15 0b
98 45 db fe 98 68 e2 80 bd f6 99 b9 37 c0 a3 5b 9f 38 70 82 8a 20 b8 44 a6 d7 35 35 5c 6e ed ce
25 28 10 00 88 21 0c e0 4b fd aa 47 9b a4 e7 ad 27 f0 f5 83 e3 a6 6e 89 2b 2a 7b 18 65 1d 4a 05
ff a1 fd d3 44 fe c3 34 ad ae a5 1a b5 a2 61 83 c3 ca 6a 1a c0 7b 08 44 42 f9 bf b7 ec 88 11 01
39 33 1e 96 b9 2d 91 72 63 e0 d5 e2 af a0 82 c0 46 b7 bd 43 cf f5 36 b5 f1 09 4f b6 2e e9 e3 00
```

说明：

音频目前支持G711A/G711U/LC-AAC编码方式，编码参数的支持需要参考头文件link_visual_def.h中的宏定义值。

同一路流切换视频码流时（如主码流切为子码流、修改码流分辨率等），请保证切换后的第一帧为I帧，否则会引发花屏等现象。

图片服务

- **lv_trigger_pic_capture_cb**

  接口详情：typedef int (lv_trigger_pic_capture_cb)(const char *id)

  通知设备开始抓取图片。

  请求参数说明如下。

参数	类型	说明
id	const char *	本次请求的ID，需要回传。

- **lv_post_alarm_image**

  接口详情：int lv_post_alarm_image(const char *buffer, int buffer_len, lv_event_type_e type, const char *id)

  发送图片和报警事件，适用于抓图回调后进行上报，或者设备主动发起上报。

  请求参数说明如下。

参数	类型	说明
buffer	const char *	图片数据。
buffer_len	int	图片数据长度。
type	lv_event_type_e	上报类型，分为抓图回调后进行上报和设备主动发起上报。
id	const char *	抓图回调的ID回传，主动上报时传空字符串或者NULL。

  示例代码如下。

```
//demo，读取一个图片文件并上传
void testPicUpload(lv_event_type_e type, const char *id)
{
    char *buf = new char[1024*1024];
    FILE *fp = nullptr;
    int ret = 0;
    if((fp = fopen("1.jpg", "r")) == nullptr) {
        DEMO_TRACE("Failed to open file.");
        return;
    }
    ret = fseek(fp, 0, SEEK_SET);
    if(ret != 0) {
        return ;
    }

    ret = fread(buf, 1, 1024*1024, fp);
    fclose(fp);

    lv_post_alarm_image(buf, ret, type, id);
    delete [] buf;
}
```

- 关于接口调用的其他说明如下。
  - 图片需要有云储存权限才可上传。
  - 上传的最大图片大小为1MB。
  - 图片上传的最小间隔为1秒，频繁触发的图片将会被丢弃。
  - 图片会由SDK内部保存一份拷贝，并形成待发送的图片队列。图片队列的长度为5张图片，在网络差的情况下，图片队列可能会满，满队列后新生成的图片将会被丢弃。
  - SDK不限制图片的格式，只要云端或者从云端拉取图片的设备能够支持解析即可。推荐使用常见的图片格式，如,jpg格式。

语音对讲服务

- **lv_start_voice_intercom_cb**

  接口详情：typedef int (*lv_start_voice_intercom_cb)(int service_id)

  回调函数，通知一路语音对讲服务已可以建立。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

- **lv_voice_intercom_start_service**

  接口详情：int lv_voice_intercom_start_service(int service_id, const lv_audio_param_s * audio_param)

  建立一路语音对讲链路，在接收到语音对讲开始回调后调用。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
audio_param	const lv_audio_param_s *	设备的音频参数结构体。

  示例代码如下。

```
//demo，startVoiceIntercomCallback是lv_start_voice_intercom_cb的实际实现
lv_start_voice_intercom_cb = startVoiceIntercomCallback;

int startVoiceIntercomCallback(int service_id)
{
    //收到开始语音对讲请求时，主动开启对讲服务
    lv_audio_param_s audio_param;
    memset(&audio_param, 0, sizeof(lv_audio_param_s));
    audio_param.format = LV_AUDIO_FORMAT_G711A;
    audio_param.channel = LV_AUDIO_CHANNEL_MONO;
    audio_param.sample_bits = LV_AUDIO_SAMPLE_BITS_16BIT;
    audio_param.sample_rate = LV_AUDIO_SAMPLE_RATE_8000;
    int ret = lv_voice_intercom_start_service(service_id, &audio_param);

    return 0;
```

```
}
```

- **lv_stop_voice_intercom_cb**

  接口详情：typedef int (*lv_stop_voice_intercom_cb)(int service_id)

  回调函数，通知一路语音对讲服务已可以断开。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

- **lv_voice_intercom_stop_service**

  接口详情：int lv_voice_intercom_stop_service(int service_id)

  断开一路语音对讲链路，在接收到语音对讲结束回调后调用。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。

  示例代码如下。

```
//demo，stopVoiceIntercomCallback是lv_start_voice_intercom_cb的实际实现
lv_stop_voice_intercom_cb = stopVoiceIntercomCallback;

int stopVoiceIntercomCallback(int service_id)
{
   lv_voice_intercom_stop_service(service_id);
}
```

- **lv_voice_intercom_receive_metadata_cb**

  接口详情：typedef int (lv_voice_intercom_receive_metadata_cb)(int service_id, const lv_audio_param_s *audio_param)

  回调函数，通知语音对讲时，App发送的音频参数格式。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
audio_param	const lv_audio_param_s *	APP发送的音频格式结构体。

- **lv_voice_intercom_receive_data_cb**

  接口详情：typedef void (lv_voice_intercom_receive_data_cb)(const char *buffer, unsigned int buffer_len)

  回调函数，语音对讲时，App发送的音频数据。

  请求参数说明如下。

参数	类型	说明
buffer	const char *	APP发送的音频数据。
buffer_len	int	APP发送的音频数据长度。

- **lv_voice_intercom_send_audio**

  接口详情：int lv_voice_intercom_send_audio(int service_id, const char *buffer, int buffer_len , unsigned int timestamp_ms)

  向其中一路语音对讲链路发送音频数据。

  请求参数说明如下。

参数	类型	说明
service_id	int	链路的ID。
buffer	const char *	设备音频数据。
buffer_len	int	设备音频数据长度。
timestamp_ms	unsigned int	设备音频时间戳。

  示例代码如下。

```
//这个回调输入音频参数
void handlerVoiceIntercomBuffer(unsigned char *buffer, unsigned int buffer_size,
      unsigned int present_time, int service_id)
{
   lv_voice_intercom_send_audio(service_id, (const char *)buffer, buffer_size,
present_time);
}
```

自验收标准

- 首帧加载95%的请求，请控制在1.2秒以内。
- 推流延迟控制在1秒以内，音视频同步应控制在300毫秒以内。
- 设备端支持强制I帧指令，且两次I帧的最小间隔控制在100毫秒以内。
- TF卡列表查询一天的录像文件，返回索引时间控制在1秒以内。
- TF卡视频播放，Seek指令的响应延迟在500毫秒以内，且持续Seek 100次稳定可用。

- 语音对讲在收到G711A编码之后的解码耗时要小于音频文件的实际时间。
- PTZ在云台转到底之后，收到该方向的指令需忽略指令，避免指令堆积，保持对后续指令的快速响应。
- 产生图片的延迟在300毫秒以内，直播时抓图，直播画面帧率无明显受影响。
- 7*24打开设备，正常使用情况下，设备不会崩溃。
- 设备支持静默定期重启机制，建议每五天自动静默重启一次，可提供用户设置。
- 设备需支持OTA。
- 若采用二维码配网，则二维码扫描成功时间应控制在1.5秒以内。
- 关于语音对讲，设备端必须支持回声消除器（Acoustic Echo Canceller），确保设备端采集的音频数据没有回声。

2.11 配网功能开发

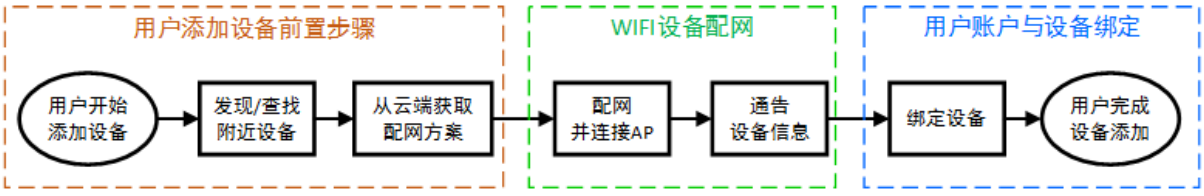
2.11.1 Wi-Fi设备配网方案介绍

对于无屏的设备（如插座、灯泡等），用户无法通过输入Wi-Fi热点信息让设备接入网络，此时需要对这些设备进行配网操作。生活物联网平台为Wi-Fi设备提供了多种配网技术方案，使不具备人机交互能力的设备可以借助于一些特殊方式连上网络。

配网概述

配网是将路由器的Wi-Fi SSID和PASSWORD通过某种方式传递到终端设备，让终端设备可以接入Wi-Fi网络的过程。配网示意图如下所示。

通常情况下，用户使用App添加一个Wi-Fi设备的流程如下。

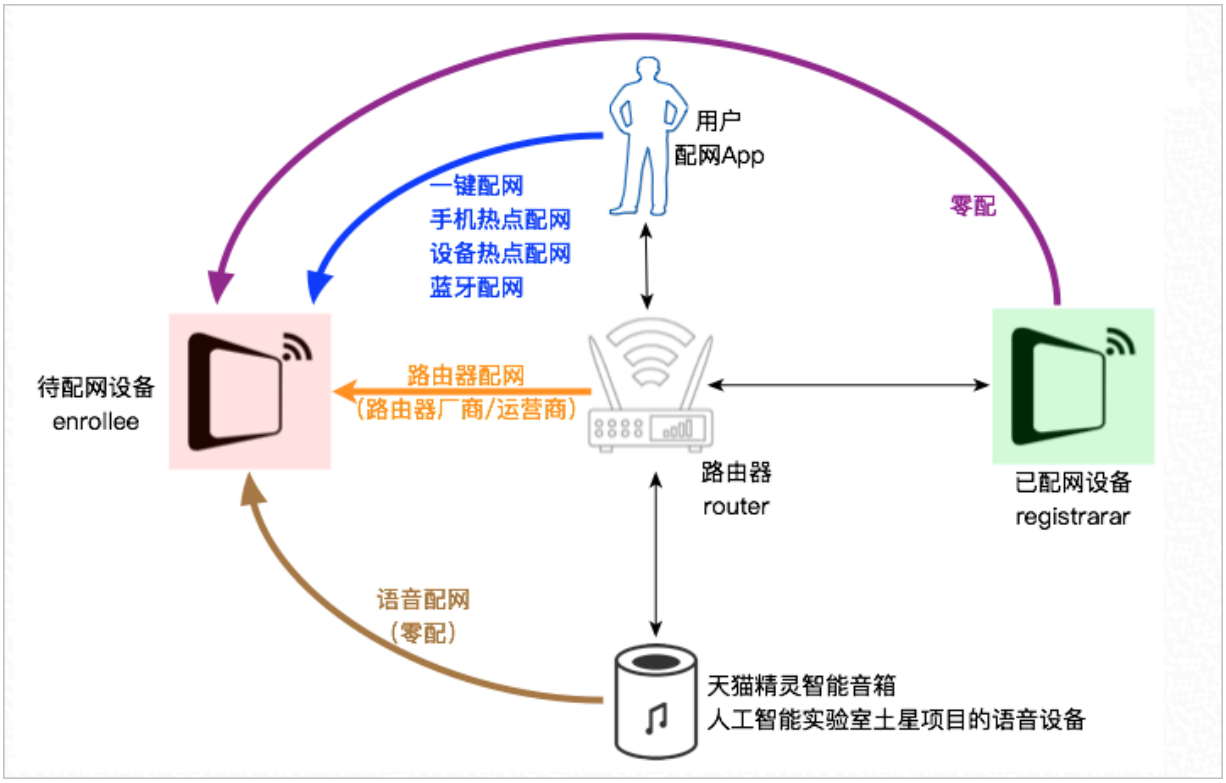


1. 用户添加设备前的准备操作。
2. Wi-Fi设备配网（通过某种方式将目标AP的SSID和Password给到设备，使设备可连接上目标AP）。

3. 用户账户与设备之间绑定。

Wi-Fi配网方式对比

生活物联网平台提供了多种Wi-Fi设备的配网方式（各种方式的对比如下表所示），Wi-Fi设备配网方式的总体场景示意图如下。

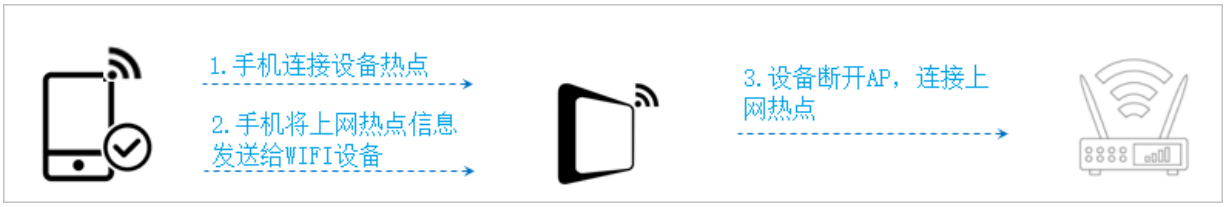


配网方式	配网技术	简要说明	补充说明
设备热点配网（dev-ap-config）	手机连设备热点传数据	App连接设备起的热点传输配网信息给设备	推荐使用该配网方式
零配配网（zero-config）	Wi-Fi管理帧传数据	已配网设备传输配网信息给待配网设备	当前环境中，具有可以为待配设备进行配网的设备，该方案无需用户输入Wi-Fi密码，推荐您使用
蓝牙辅助配网（ble-config）	BLE（Bluetooth）传数据	App通过蓝牙通道传输配网信息给设备	设备必须支持Wi-Fi与BLE双模通信（combo chip），推荐您使用该配网方式
一键配网（smart-config或广播配网）	广播密文包长编码（bcast）	App通过广播报文传输配网信息给设备	-

配网方式	配网技术	简要说明	补充说明
	P2P密文编码（p2p）	App通过P2P报文传输配网信息给设备	仅Android支持，SSID和Password的长度之和不能大于32Byte（部分三星手机不能大于25Byte）
	组播密文编码（mcast）	App通过组播报文传输配网信息给设备	-
手机热点配网（phone-ap-config）	手机起热点，设备连接传数据	App起固定热点，设备自动发起连接，通过连接通道传输配网信息	iOS需要手动设置热点再切回App
摄像头扫码配网（webcam-config）	摄像头扫码解析数据	App生成二维码，放置于摄像头设备前识别解析其中的配网信息	设备须具备摄像头，且支持二维码解码能力

设备热点配网

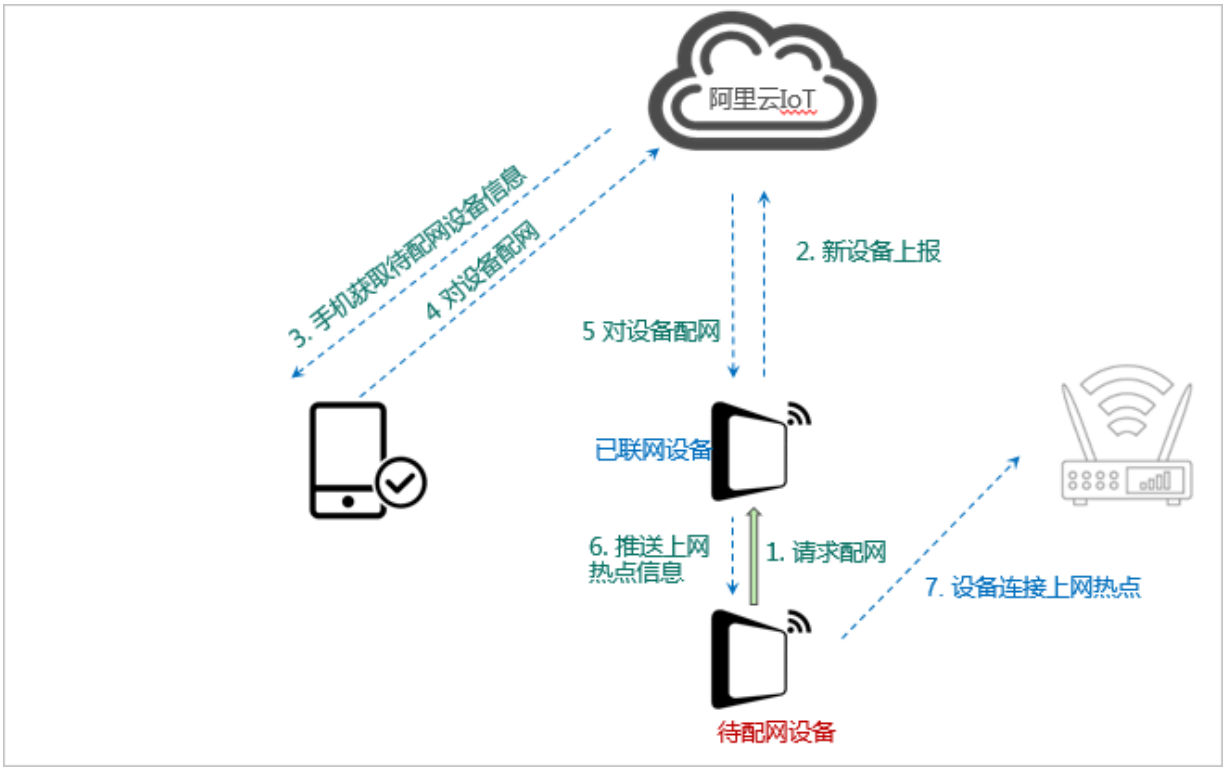
设备热点配网（dev-ap-config）的配网流程如下。



1. 设备开启自带的Wi-Fi热点。手机搜索并发现热点后，连接到该设备的热点。
2. 建立连接通道后，手机将Wi-Fi热点（路由器）的SSID/密码发送给设备。
3. Wi-Fi设备使用该SSID/密码连接Wi-Fi热点（路由器）。

零配配网

零配配网（zero-config）不需要用户在手机端输入Wi-Fi热点的SSID/密码，配网流程如下。



1. 待配网设备向已连接Wi-Fi热点（路由器）的设备发送配网请求。
2. 已连接Wi-Fi热点的设备向生活物联网平台上报新设备。
3. 生活物联网平台向手机（App）获取待配网设备的信息。
4. 手机向平台反馈待配网设备的详细信息。
5. 平台向已连接Wi-Fi热点的设备下发对新设备配网的许可。
6. 已连接Wi-Fi热点的设备向待配网设备推送Wi-Fi热点的SSID/密码信息。
7. 待配网设备连接Wi-Fi热点。

蓝牙辅助配网

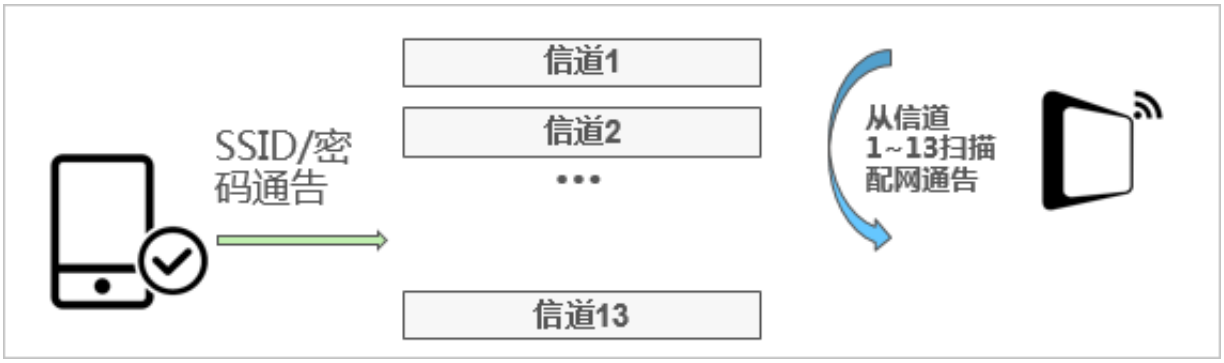
蓝牙辅助配网（ble-config）方案无需区分手机的操作系统（iOS或Android的手机体验一样），且该方案的配网成功率和可靠性较高。使用该方案，设备需支持Wi-Fi与BLE双模通信能力。配网流程如下。

1. 手机通过蓝牙连接待配网的双模设备。
2. 手机通过蓝牙将Wi-Fi热点（路由器）的SSID/密码信息传送给设备。

3. 设备连接Wi-Fi热点。

一键配网

一键配网（smart-config）方案只需要用户输入Wi-Fi的密码即可。配网流程如下。



1. 手机通过广播发送Wi-Fi热点的SSID/密码信息。
2. 设备通过监听方式获取信息，并连接上网热点的方式。

手机热点配网

手机启动一个预定义热点（例如：热点名为**aha**，密码为**12345678**），设备先连接到手机热点获取路由器上网热点信息，再连接上网热点的方式。该方案需要Wi-Fi切换相关的权限，如果没有授予权限可能会导致配网失败，建议将该方案作为备选方案。

1. 手机启动一个预定义热点（例如：热点名为**aha**，密码为**12345678**），设备先连接到手机热点上。
2. 手机将Wi-Fi热点的SSID/密码信息发送给Wi-Fi设备。
3. 设备断开手机热点，连接Wi-Fi热点。

摄像头扫码配网

摄像头扫码配网（webcam-config）针对拥有摄像头及二维码解码能力设备。配网流程如下。

1. 通过App生成携带SSID、password等信息的二维码。
2. 摄像头根据App的二维码画面解码出信息，连接路由器。

您将SSID、password信息传入SDK后，SDK会将完整二维码信息（二维码中携带的信息如下表所示）回调并开启局域网监听，您需要展现二维码并监听配网结果。

Key	描述	是否必选
v	version，版本号，区分为阿里的二维码规则，当前为Ali_1	必选
s	ssid，家庭路由SSID	必选
p	password，家庭路由密码	必选
b	BSSID，家庭路由的BSSID信息，Hex String，解决gbk编码等问题	必选
t	token，绑定需要的Token信息，3 Bytes random string	可选

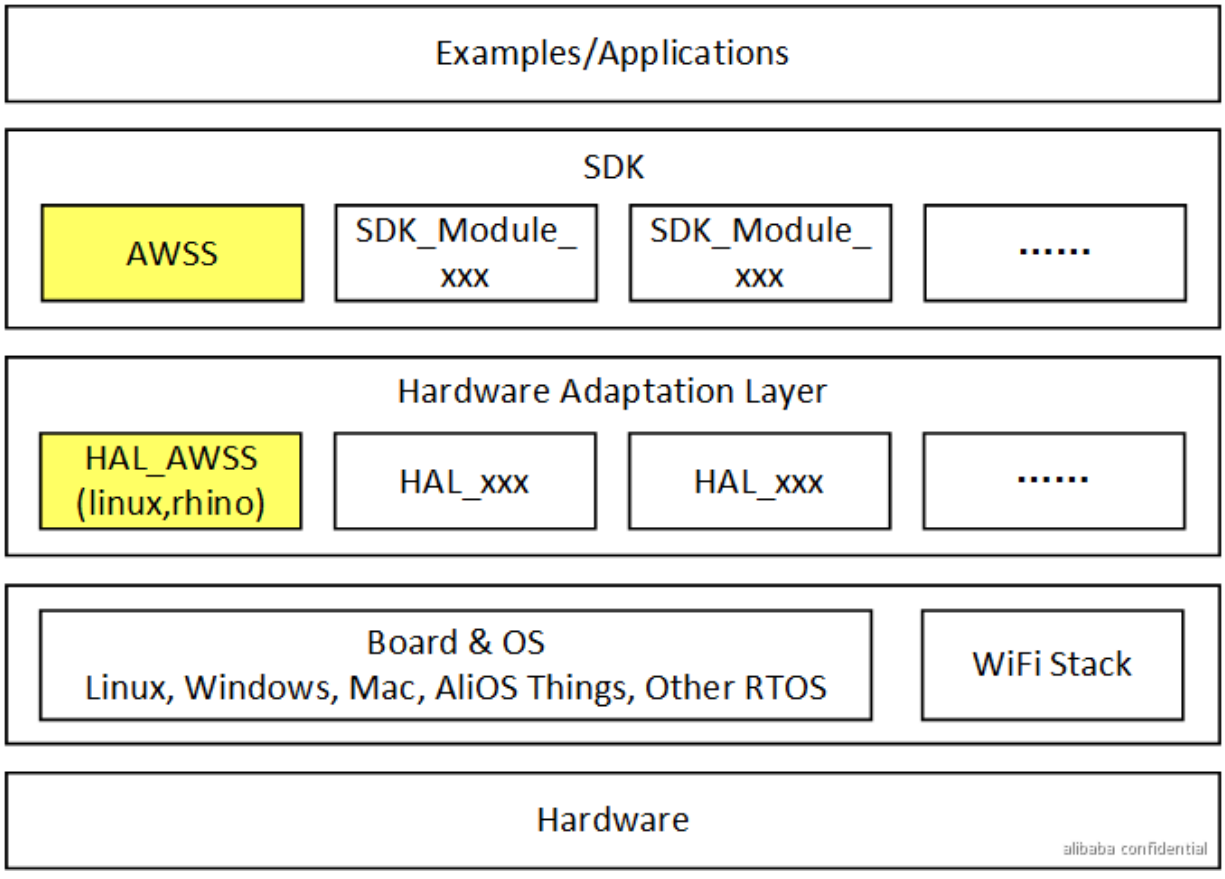
二维码按JSON规则拼接，示例如下。

```
{"v":"Ali_1","s":"Router","p":"12345678","b":"EC26CAB68CC","t":"93DAB7"}
```

2.11.2 Wi-Fi设备配网适配开发

生活物联网平台提供多种Wi-Fi设备的配网技术方案，对于不具备人机交互能力的设备，可以通过借助其他设备来协助配网。为此，生活物联网平台提供了开放的配网能力，便于您在不同的设备硬件平台上快速适配和实现Wi-Fi配网功能。

下图为生活物联网平台SDK中Wi-Fi设备配网模块的分层框架。



- HAL_AWSS实现接口和OS、硬件的适配对接，为上层SDK的配网服务提供基础的Wi-Fi通信能力。

  在生活物联网平台SDK中，HAL_AWSS的适配实现，主要提供了Wi-Fi配网需要适配的接口。
- AWSS模块是SDK中提供的配网服务，可以提供多种配网方式，依赖于HAL层的HAL_AWSS部分，以及HAL层其他相关接口的正确适配。

  AWSS模块为上层的应用开发提供了配网相关的用户编程接口，Examples为调用这些接口的完整示例的实现。

  在生活物联网平台SDK中，AWSS用户编程接口，主要用来调用实现设备的Wi-Fi配网功能。

通用HAL说明

Wi-Fi配网模块中依赖生活物联网平台SDK的公共HAL接口（如下表所示）。请您参照生活物联网平台SDK中对应模块的说明完成对接。

序号	适配接口名	说明
1	HAL_MutexCreate	创建一个互斥量对象，返回指向所创建互斥量的指针，用于同步访问，对于仅支持单线程应用，可实现为空函数
2	HAL_MutexDestroy	销毁一个互斥量对象，释放资源
3	HAL_MutexLock	锁住一个互斥量
4	HAL_MutexUnlock	解锁一个互斥量
5	HAL_UptimeMs	获取设备从上电到当前时刻所经过的毫秒数
6	HAL_Malloc	申请一块堆内存
7	HAL_Free	释放参数ptr指向的一块堆内存，当传入的参数为NULL时不执行任何操作
8	HAL_SleepMs	睡眠函数，使当前执行线程睡眠指定的毫秒数
9	HAL_GetProductKey	获取设备的ProductKey，用于标识设备的品类，设备证书之一
10	HAL_GetProductSecret	获取设备的ProductSecret，用于标识设备的品类，设备证书之一
11	HAL_GetDeviceName	获取设备的DeviceName，用于标识设备单品的名字，设备证书之一
12	HAL_GetDeviceSecret	获取设备的DeviceSecret， 用于标识设备单品的密钥，设备证书之一
13	HAL_Kv_Set	Flash中写入键值对（Key-Value）
14	HAL_Kv_Get	Flash中读取键值对的Value

序号	适配接口名	说明
15	HAL_Aes128_Init	初始化AES加密的结构体
16	HAL_Aes128_Destroy	销毁AES加密的结构体
17	HAL_Aes128_Cbc_Encrypt	以AES-CBC-128方式，根据HAL_Aes128_Init()时传入的密钥，加密指定的明文
18	HAL_Aes128_Cbc_Decrypt	以AES-CBC-128方式，根据HAL_Aes128_Init()时传入的密钥，解密指定的密文
19	HAL_Aes128_Cfb_Encrypt	以AES-CFB-128方式，根据HAL_Aes128_Init()时传入的密钥，加密指定的明文
20	HAL_Aes128_Cfb_Decrypt	以AES-CFB-128方式，根据HAL_Aes128_Init()时传入的密钥，解密指定的密文
21	HAL_Timer_Create	根据Name、TimerFunc和用户上下文创建Timer
22	HAL_Timer_Start	Timer开始计时，Timer超时时调用回调函数TimerFunc
23	HAL_Timer_Stop	停止Timer计时
24	HAL_Timer_Delete	删除Timer，释放资源

说明：

带AliOS Things的生活物联网平台SDK的HAL接口文件说明，请参见以下目录中的各个文件内容。

- /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

HAL适配概述

Wi-Fi配网模块需要适配的HAL接口的详细说明如下表所示。

说明：

表中分类为**配网通用**的接口在适配时必须要实现，否则会影响Wi-Fi设备生活物联网平台SDK的正常运行。其他配网特定的分类，您可以根据自己硬件平台的情况以及产品的使用场景和需要进行选择支持，但是同一配网方式类别的所有接口需要适配完整，否则会影响此种配网方式的正常运行。

序号	适配接口名	说明	分类
1	HAL_Wifi_Get_Mac	获取设备的MAC地址， 格式应当是”XX:XX:XX:XX:XX:XX”	配网通用

序号	适配接口名	说明	分类
2	HAL_Wifi_Get_IP	获取设备的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，Station模式为连接AP时的IP地址，SoftAP模式为设备作为AP的Gateway IP地址	配网通用
3	HAL_Awss_Get_Channelscan_Interval_Ms	获取在每个信道（channel）上扫描的时间长度，单位是毫秒	配网通用
4	HAL_Awss_Get_Timeout_Interval_Ms	获取配网服务（AWSS）的超时时间长度，单位是毫秒	配网通用
5	HAL_Awss_Get_Encrypt_Type	获取一键配网服务的安全等级	一键配网
6	HAL_Awss_Get_Conn_Encrypt_Type	获取基于连接的配网服务（热点配网/零配）的安全等级	• 零配配网 • 设备热点配网 • 蓝牙辅助配网 • 手机热点配网
7	HAL_Awss_Open_Monitor	设备工作在监听（Monitor）模式，并在收到802.11帧的时候调用被传入的回调函数（包括管理帧和数据帧）	配网通用
8	HAL_Awss_Close_Monitor	关闭监听（Monitor）模式	配网通用
9	HAL_Wifi_Scan	启动一次Wi-Fi的空中扫描	配网通用
10	HAL_Awss_Switch_Channel	设置Wi-Fi设备切换到指定的信道（channel）上	配网通用
11	HAL_Awss_Connect_Ap	要求Wi-Fi网卡连接指定热点（Access Point）的函数，bssid指定特定AP，另外bssid也可能为空或无效值（全0或全0xff）	配网通用
12	HAL_Wifi_Send_80211_Raw_Frame	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）	配网通用
13	HAL_Wifi_Enable_Mgmt_Frame_Filter	使能或禁用对管理帧的过滤	配网通用
14	HAL_Sys_Net_Is_Ready	检查Wi-Fi网卡、芯片或模组当前的IP地址是否有效	配网通用

序号	适配接口名	说明	分类
15	HAL_Wifi_Get_Ap_Info	获取设备所连接的热点（Access Point）的信息（SSID、Password、BSSID等）	配网通用
16	HAL_Wifi_Get_Link_Stat	获取当前与AP连接链路状态的信息（channel、RSSI等）	配网通用
17	HAL_Awss_Open_Ap	开启设备热点（SoftAP模式）	设备热点配网
18	HAL_Awss_Close_Ap	关闭当前设备热点，并把设备由SoftAP模式切换到Station模式	设备热点配网

HAL_Wifi_Get_Mac

原型

```
char *HAL_Wifi_Get_Mac(_OU_ char mac_str[HAL_MAC_LEN]);
```

接口说明

获取Wi-Fi网口的MAC地址，格式为XX:XX:XX:XX:XX:XX。

参数说明

参数	数据类型	方向	说明
mac_str	char	输出	指向缓冲区数组起始位置的字符指针

返回值说明

指向缓冲区数组起始位置的字符指针。

HAL_Wifi_Get_IP

原型

```
uint32_t HAL_Wifi_Get_IP(_OU_ char ip_str[NETWORK_ADDR_LEN], _IN_ const char *ifname);
```

接口说明

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。

参数说明

参数	数据类型	方向	说明
ip_str	char[]	输出	存放点分十进制格式的IP地址字符串的数组
ifname	const char*	输入	指定Wi-Fi网络接口的名字（如果只有一个网口，可以忽略此参数，该参数可能为NULL）

返回值说明

二进制形式的IP地址，以网络字节序（大端）组织。

HAL_Awss_Get_Channelscan_Interval_Ms

原型

```
int HAL_Awss_Get_Channelscan_Interval_Ms(void);
```

接口说明

获取在每个信道（channel）上扫描的时间长度，单位是毫秒，建议设置为200ms~400ms（默认250ms）。

参数说明

void

返回值说明

时间长度，单位为毫秒。

HAL_Awss_Get_Timeout_Interval_Ms

原型

```
int HAL_Awss_Get_Timeout_Interval_Ms(void);
```

接口说明

获取配网服务（AWSS）的超时时间长度，单位是毫秒，建议配置为60s或60000ms。

参数说明

void

返回值说明

超时时长，单位是毫秒。

HAL_Awss_Get_Encrypt_Type

原型

```
int HAL_Awss_Get_Encrypt_Type(void);
```

接口说明

获取一键配网服务的安全等级。

参数说明

void

返回值说明

值	说明
0	open (no encrypt)
1	aes256cfb with default aes-key and aes-iv
2	aes128cfb with default aes-key and aes-iv
3	aes128cfb with aes-key per product and aes-iv = 0
4	aes128cfb with aes-key per device and aes-iv = 0
5	aes128cfb with aes-key per manufacture and aes-iv = 0
others	无效

HAL_Awss_Get_Conn_Encrypt_Type

原型

```
int HAL_Awss_Get_Conn_Encrypt_Type(void);
```

接口说明

获取零配，热点配网的安全等级。

参数说明

void

返回值说明

值	说明
3	aes128cfb with aes-key per product and aes-iv = random
4	aes128cfb with aes-key per device and aes-iv = random

值	说明
5	aes128cfb with aes-key per manufacture and aes-iv = random
others	无效

HAL_Awss_Open_Monitor

原型

```
void HAL_Awss_Open_Monitor(_IN_ awss_recv_80211_frame_cb_t cb);
```

接口说明

设置Wi-Fi网卡工作在监听（Monitor）模式，并在收到802.11帧的时候调用被传入的回调函数。

参数说明

参数	数据类型	方向	说明
cb	awss_recv_80211_frame_cb_t	输入	回调函数指针，当Wi-Fi接收到帧时会调用此函数

```
/**
 * @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] length @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, range of [-127, -1]
 */
typedef int (*awss_recv_80211_frame_cb_t)(char *buf, int length,
       enum AWSS_LINK_TYPE link_type, int with_fcs, signed char rssi);
```

返回值说明

void

HAL_Awss_Close_Monitor

原型

```
void HAL_Awss_Close_Monitor(void);
```

接口说明

设置Wi-Fi网卡离开监听（Monitor）模式，并开始以站点（Station）模式工作。

参数说明

void

返回值说明

void

HAL_Wifi_Scan

原型

```
int HAL_Wifi_Scan(awss_wifi_scan_result_cb_t cb);
```

接口说明

启动一次Wi-Fi的空中扫描，该API是一个阻塞操作，扫描没有完成不能结束。所有的AP列表收集完成后，一个一个通过回调函数告知AWSS，最好不要限制AP的数量，否则可能导致中文GBK编码的SSID热点配网失败。

参数说明

参数	数据类型	方向	说明
cb	awss_wifi_scan_result_cb_t	输入	扫描通知回调函数

返回值说明

值	说明
0	扫描正常结束
-1	其他情况

HAL_Awss_Switch_Channel

原型

```
void HAL_Awss_Switch_Channel(
        _IN_ char primary_channel,
```

```
        _IN_OPT_ char secondary_channel,
        _IN_OPT_ uint8_t bssid[ETH_ALEN]);
```

接口说明

设置Wi-Fi网卡切换到指定的信道（channel）上。

参数说明

参数	数据类型	方向	说明
primary_channel	char	输入	首选信道
secondary_channel	char	输入	辅助信道，信道带宽为40MHz时才会使用，信道宽度为20MHz时忽略该参数
bssid	uint8_t	输入	生活物联网平台SDK1.3.0及以上版本中，此参数已废弃，您若兼容1.3.0以前版本使用

返回值说明

void

HAL_Awss_Connect_Ap

原型

```
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);
```

接口说明

要求Wi-Fi网卡连接指定热点（Access Point）的函数，bssid指定特定AP，另外bssid也可能为空或无效值（全0或全0xff）。

参数说明

参数	数据类型	方向	说明
connection_timeout_ms	uint32_t	输入	连接AP的超时时间
ssid	char	输入	目的AP的SSID

参数	数据类型	方向	说明
passwd	char	输入	目的AP的PASSWORD
auth	enum	输入	目的AP的加密方式，HAL可以忽略
encry	enum	输入	目的AP的认证方式，HAL可以忽略
bssid	uint8_t	输入	目的AP的BSSID，该字段可能为NULL或设置为全0
channel	uint8_t	输入	目的AP的信道，该字段可以忽略

返回值说明

值	说明
0	连接AP和DHCP成功
-1	连接AP或DHCP失败

HAL_Wifi_Send_80211_Raw_Frame

原型

```
int HAL_Wifi_Send_80211_Raw_Frame(_IN_ enum HAL_Awss_Frame_Type type,
      _IN_ uint8_t *buffer, _IN_ int len);
```

接口说明

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）。

参数说明

参数	数据类型	方向	说明
type	enum HAL_Awss_Frame_Type	输入	查看HAL_Awss_Frame_Type_t定义
buffer	uint8_t *	输入	802.11裸数据帧，包括完整的MAC头和FCS域
len	int	输入	802.11裸帧字节长度

```
/* 80211 frame type */
typedef enum HAL_Awss_Frame_Type {
    FRAME_ACTION,               // 802.11中的Action帧
    FRAME_BEACON,               // 802.11中的Beacon帧
    FRAME_PROBE_REQ,            // 802.11中的Probe Request帧
```

```
    FRAME_PROBE_RESPONSE,      // 802.11中的Probe Response帧
    FRAME_DATA                 // 802.11中的数据帧
} HAL_Awss_Frame_Type_t;
```

返回值说明

值	说明
0	发送成功
-1	发送失败

HAL_Wifi_Enable_Mgmt_Frame_Filter

原型

```
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);
```

接口说明

使能或禁用对特定管理帧的过滤。

参数说明

参数	数据类型	方向	说明
filter_mask	uint32_t	输入	帧过滤参数
vendor_oui	uint8_t	输入	Wi-Fi联盟分配的厂商OUI，如果OUI为NULL，表示不对OUI过滤，反之要根据OUI过滤
callback	awss_wifi_mgmt_frame_cb_t	输入	用于接收802.11帧或者信息元素（IE）的回调函数

返回值说明

值	说明
= 0	发送成功
= -1	发送失败
= -2	不支持

HAL_Sys_Net_Is_Ready

原型

```
int HAL_Sys_Net_Is_Ready();
```

接口说明

检查系统网络是否可用（设备是否已经成功获得IP地址并且当前IP地址可用）。

参数说明

void

返回值说明

值	说明
0	网络不可用
1	网络可用

HAL_Wifi_Get_Ap_Info

原型

```
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]);
```

接口说明

获取所连接的热点（Access Point）的信息。

参数说明

参数	数据类型	方向	说明
ssid	char	输出	AP的SSID，该参数可能为NULL
passwd	char	输出	AP的Password，该参数为NULL
bssid	uint8_t	输出	AP的BSSID，该参数可能为NULL，如果bssid不对将导致零配设备发现失败

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Wifi_Get_Link_Stat

原型

```
int HAL_Wifi_Get_Link_Stat(_OU_ int *p_rssi,
                           _OU_ int *p_channel);
```

接口说明

获取当前与AP连接链路状态的信息（channel、RSSI等）。

参数说明

参数	数据类型	方向	说明
p_rssi	char	输出	设备当前关联的AP的信号强度值
p_channel	char	输出	设备当前关联的AP所在的信道

返回值说明

值	说明
0	操作成功
-1	操作失败

HAL_Awss_Open_Ap

原型

```
int HAL_Awss_Open_Ap(const char *ssid,
                     const char *passwd,
                     int beacon_interval,
                     int hide);
```

接口说明

开启设备热点（SoftAP模式）。

参数说明

参数	数据类型	方向	说明
ssid	const char *	输入	热点的ssid字符
passwd	const char *	输入	热点的passwd字符
beacon_interval	int	输入	热点的Beacon广播周期（广播间隔）
hide	int	输入	• 0：非隐藏 • 非0（其它值）：隐藏

返回值说明

值	说明
0	success
-1	unsupported
-2	failure with system error
-3	failure with no memory
-4	failure with invalid parameters

HAL_Awss_Close_Ap

原型

```
int HAL_Awss_Close_Ap(void);
```

接口说明

关闭设备热点。

参数说明

void

返回值说明

值	说明
0	success
-1	unsupported
-2	failure

2.11.3 蓝牙辅助配网开发

蓝牙辅助配网基于已有的WiFi+BLE的Combo芯片方案，利用BLE的通信能力，将WiFi连接所需的SSID、无线密码等信息传输给WiFi+BLE Combo设备，使设备顺利接入互联网与云端，继而完设备绑定、数据上下行传输等功能。您可以根据生活物联网平台提供的设备端SDK代码、应用示例等，快速集成蓝牙辅助配网功能。

蓝牙辅助配网的方案示意图如下。

生活物联网平台设备端蓝牙辅助配网方案的特性说明如下。

- 数据加密传输：确保物联网复杂环境下的通信安全，保护用户隐私。
- 支持国内与海外环境不同区域的物联网设备接入。
- 支持设备功能自检（包括配网、连云等），并通过移动端App页面实时反馈，增强设备可运维性。
- 商业化高水准的专业测试，保障功能品质。
- 配套提供专门的Combo Application设备应用，便于客户快速评估和验证功能。
- ComboApp固件支持蓝牙辅助配网和一键配网功能，备用的配网功能增强用户配网的稳定性。
- ComboApp固件支持便捷的串口命令交互功能，适用于不同功能场景的操作使用。
- 支持一机一密的高安全级别的防护，每个设备需烧录生活物联网平台颁发的设备证书。

设备端资源需求

生活物联网平台的蓝牙辅助配网功能模块需运行于支持WiFi + BLE的Combo芯片或模组上，该模块完整功能的支持需要的硬件资源推荐指标列表如下，供您选型时参考。

指标类目	指标项	指标推荐要求
硬件性能类	处理器	• ≥ 32位MCU • 主频 ≥ 100MHz
	内存资源	• Flash ≥ 2MByte • RAM ≥ 256KByte

指标类目	指标项	指标推荐要求
协议栈指标类	WiFi协议栈	• 802.11 b/g/n 1x1 • 2.4G频段支持必选，5G频段支持可选 • Station模式 • Monitor模式
	BLE协议栈	≥BLE 4.2
生活物联网平台SDK指标类	内存资源	• Flash 300KByte • RAM 50KByte

SDK获取

设备端SDK中包含了蓝牙辅助配网的示例代码ComboApp，该示例代码目录与编译指令在不同设备端SDK版本中区别如下。

SDK版本	版本比较	代码目录	编译指令
1.4.0	• 海外建连速度优化 • 配网时设备异常自检	/Products/example/comboapp/	**./build.sh example comboapp bk7231udevkitc SINGAPORE ONLINE 1**
1.3.0	支持设备身份信息设置、蓝牙辅助配网等实用cli指令	/Living_SDK/example/comboapp/	**cd Living_SDK** **aos make clean** **aos make comboapp@bk7231udevkitc btstack=vendor**
1.1.0	初始蓝牙辅助配网版本，设备需使用一机一密的认证方式	/example/comboapp/	**aos make clean** **aos make comboapp@bk7231udevkitc btstack=vendor**

固件移植与开发

生活物联网平台SDK中，蓝牙辅助配网移植了部分芯片平台（在控制台的产品**调试设备**页面查询已认证的Combo类型的芯片或模组）。如果是没有经过认证的模组/芯片，则需要您根据开发文档进行移植开发。

- HAL层接口移植

  蓝牙辅助WiFi配网是专用于同时支持BLE+WiFi的设备，因此其功能依赖于底层的BLE协议栈和WiFi协议栈的正确适配。

  - BLE协议栈移植

    生活物联网平台定义了相应的BLE HAL接口，您需将BLE HAL接口根据自己使用的芯片平台进行对接和实现。蓝牙辅助配网的BLE部分是基于生活物联网平台的蓝牙设备端开发，详细请参见蓝牙设备端SDK移植接口。

  - WiFi协议栈移植

    生活物联网平台定义了相应的WiFi HAL接口，您需将WiFi HAL接口根据自己使用的芯片平台进行对接和实现。蓝牙辅助配网的WiFi部分是基于生活物联网平台设备端SDK的，移植详细说明请参见Wi-Fi芯片移植中“HAL移植”与“WiFi和配网移植”的内容。

- 用户编程接口指南

  基于已移植好的BLE和WiFi的协议栈后，生活物联网平台的SDK向上层应用开发提供了相应的用户编程接口，同时生活物联网平台的SDK中也提供了蓝牙辅助配网应用的示例程序ComboApp，通过对相应的用户编程接口进行调用，完成蓝牙辅助配网的功能。

  Breeze awss相关的编程接口说明请参见蓝牙设备端SDK用户编程接口中breeze_awss_xxx接口。

  蓝牙辅助配网的WiFi部分的编程接口说明请参见基于已认证的模组开发中“WiFi配网”的内容（重点关注连接路由器相关的接口）。

基于该开发指南，生活物联网平台提供了蓝牙辅助配网开发的最佳实践文档，详细请参见Combo设备蓝牙辅助配网适配最佳实践。

3 App端开发指南

3.1 用户账号开发指南

生活物联网平台中支持使用内置账号体系，也支持集成您自己的账号体系。您可以基于用户账号体系实现自有App开发中的注册、登录、忘记密码、登出、修改用户信息等功能。同一个项目创建多个自有品牌App时，App的账号体系相通。

内置账号体系

内置账号体系为平台提供的服务和能力，客户端集成平台账号及用户SDK即可使用。账号及用户SDK支持唤起登录页面，包括账号注册、登录、登出、忘记密码、多语言切换、修改头像、修改昵称、注销账号等功能，并支持在此页面基础上修改UI风格。详细请参见Android账号及用户SDK和iOS账号及用户SDK。

说明：

如果您的App需发行到海外，建议您在App中实现以下内容。

- 邮箱注册：邮箱注册更符合海外用户使用习惯。
- 多语言切换：当前平台支持多种语言，并持续添加中。
- 注销账号：由于海外严格的合规要求（如GDPR），App中需要提供注销用户的功能，注销时需要删除用户的所有数据。

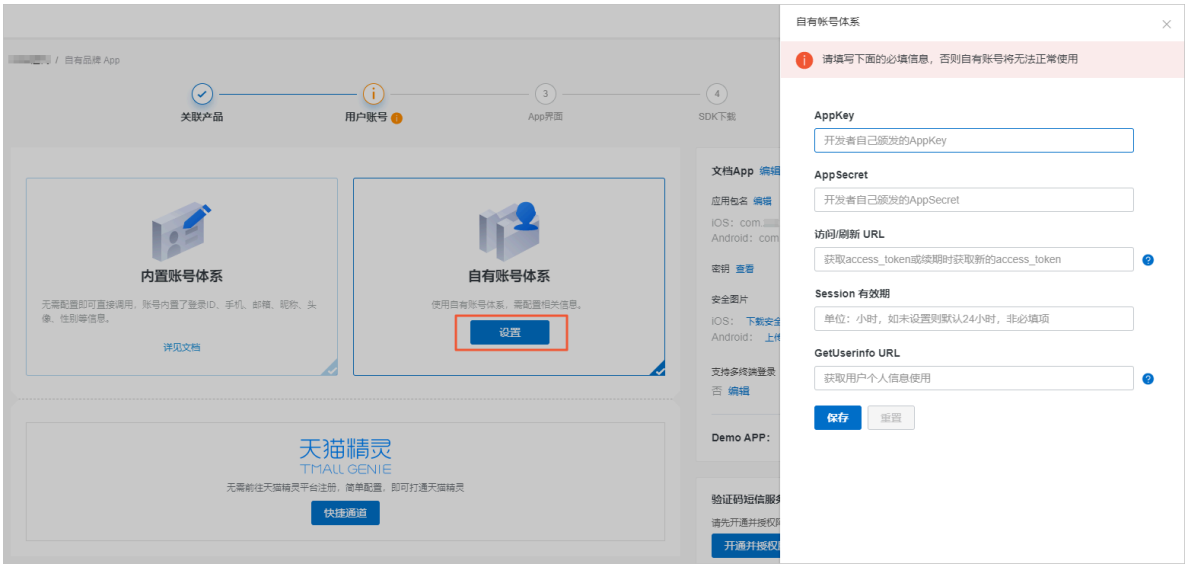
自有账号体系

当您拥有自己的账号体系，可以将您自己的账号体系和平台关联，实现设备绑定关系、设备分享、设备消息推送等功能。您自己账户体系中的用户信息不会保留在平台上，保障您的用户隐私信息。

自有账号体系对接基于Oauth2.0 API协议（代理用户访问的授权开放网络标准协议，详细请参见RFC定义参考文档），请您按照以下步骤配置和开发自有账号对接功能。

1. 进入自有品牌App的**用户账号**配置页面。

2. 选择**自有账号体系**，并单击**设置**。

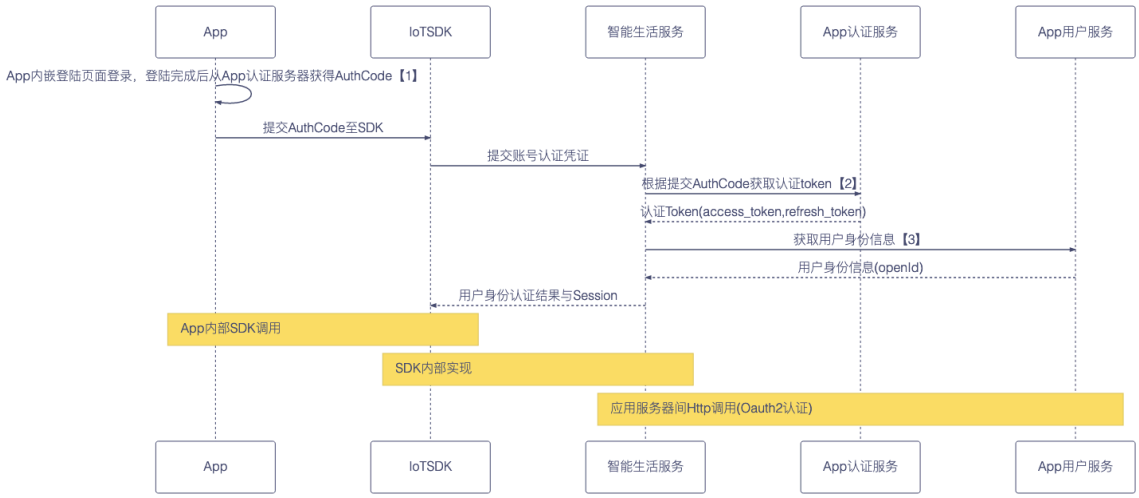


3. 配置自有账号体系，并单击**确认**保存。

   其中**访问/刷新URL**和**GetuserinfoURL**需厂商提供URL。

4. 开发自有账号体系。

   根据以下流程图完成开发工作。



   详细流程如下。

   a. 实现App登录，并从App认证服务中获取当前登录用户的AuthCode，并将AuthCode传递给SDK（使用SDK的方法请参见账号及用户SDK（Android）与账号及用户SDK（iOS）中的“三方自有账号”内容）。

   说明：

   App向App认证服务（一般包含认证登录、颁发**AuthCode**、验证**AuthCode**、颁发Token、验证Token等）获取AuthCode，该部分您自行实现。一般实现中建议使用安全随机数生成随

机字符串，并将申请的**client_id(AppKey)**与登录账号相关联，保证流程2中申请**access_token**时携带的**AuthCode**能且仅能使用一次。

b. 生活物联网平台通过HTTP POST方式，向您的App认证服务中发送请求来获取token，请求中含有**client_id(AppKey)**、**AuthCode**、**client_secret(AppSecret)**。

说明:

物联网生活平台账号互联的所有HTTP请求中，请求相应内容Content-Type需为application/json。

- **AuthCode**换取**access_token**请求

  此处为OAuth2.0标准token换取请求，以**访问/刷新URL**配置为https://test.net/api/users/oauth/token为例，示例如下。

```
POST /api/users/oauth/token?grant_type=authorization_code&client_id=testxxx
&client_secret=testxxxxx&code=test222224tD2fVtexxxxojFZL6&redirect_uri=none
HTTP/1.1
Host: test.net
Content-Type: application/x-www-form-urlencoded
```

字段说明	类型	描述
grant_type	String	授权类型，固定为字符串authorization_code
client_id	String	appKey
client_secret	String	appSecret
code	String	第一步返回的authCode

成功返回示例如下。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "result_code":"0",
  "access_token":"2YotnxxxxMWpAA",
  "token_type":"Bearer",
  "expires_in":3600,
  "openid":"cjtexxxx8test",
  "refresh_token":"tGzvxxxx2TlKWIA",
  "example_parameter":"example_value"
}
```

字段说明	类型	描述
access_token	String	授权令牌，Access_Token。

字段说明	类型	描述
expires_in	String	该access token的有效期，单位为秒。
refresh_token	String	在授权自动续期步骤中，获取新的Access_Token时需要提供的参数。
openid	String	用户唯一标识
result_code	String	- 0：成功 - 100000：client_id或者client_secert无效 - 100002：AuthCode换取AccessToken失败 - 100007：无效的授权码 - 110000：系统通用错误代码
message	String	响应结果描述

- 刷新**access_token**请求

```
POST /api/users/oauth/token?grant_type=refresh_token&client_id=testxx&
client_secret=test2222&refresh_token=test222testaaaL6 HTTP/1.1
Host: test.net
Content-Type: application/x-www-form-urlencoded
```

字段说明	类型	描述
grant_type	String	授权类型，固定为字符串refresh_token
client_id	String	appKey
client_secret	String	appSecret
refresh_token	String	获取accessToken返回的refresh_token

成功返回示例如下。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
 "result_code":"0",
 "access_token":"2YotnFxxxxsicMWpAA",
 "token_type":"Bearer",
 "expires_in":3600,
 "refresh_token":"tGzv3Jxxxx2TlKWIA",
```

```
  "example_parameter":"example_value"
}
```

字段说明	类型	描述
access_token	String	授权令牌，Access_Token。
expires_in	String	该access token的有效期，单位为秒。
refresh_token	String	在授权自动续期步骤中，获取新的Access_Token时需要提供的参数。
result_code	String	- 0：成功 - 100000：client_id或者client_secert无效 - 100003：RefreshToken已经过期或失效 - 110000：系统通用错误代码
message	String	响应结果描述

c. 通过流程2中获取到的**access_token**，生活物联网平台向App用户服务**OpenId**获取用户信息。

App用户服务即为App自有账号体系内自己的账号体系管理服务，一般包含基本的账户注册、账户信息管理（此处须实现一个HTTP请求，通过OAuth token验证鉴权获取用户基础信息）。

URL示例为：`https://thrid.com/sns/userinfo`，请求方法为POST。

请求示例如下。

```
POST /api/users/oauth/userinfo?access_token=testaaaatest222a779537c6687c3&
openid=cjtestwwd458test  HTTP/1.1
Host: testxx.net
Content-Type: application/x-www-form-urlencoded
```

字段说明	类型	描述
access_token	String	第二步获取的access token

字段说明	类型	描述
openid	String	用户唯一标识（非必填，如果需要支持Google Assistant和Amazon Alexa等语音对接能力，此处需要允许不传递该标识）

响应示例如下。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "result_code": "0",
  "message": "成功",
  "openid":" OPENID",
  "nick_name": "NICKNAME",
  "avatar_url": "image.com/xxxx.png",
  "gender": "1"
}
```

字段说明	类型	描述
result_code	String	• 0：成功 • 100000：client_id或者client_secert无效 • 100001：AccessToken 过期 • 100002：AuthCode换取AccessToken失败 • 100003：RefreshToken已经过期或失效 • 100004：用户修改密码或者解除授权导致AccessToken失效 • 100005：AccessToken非法 • 100006：无效的OpenId • 100007：无效的授权码 • 110000：系统通用错误代码

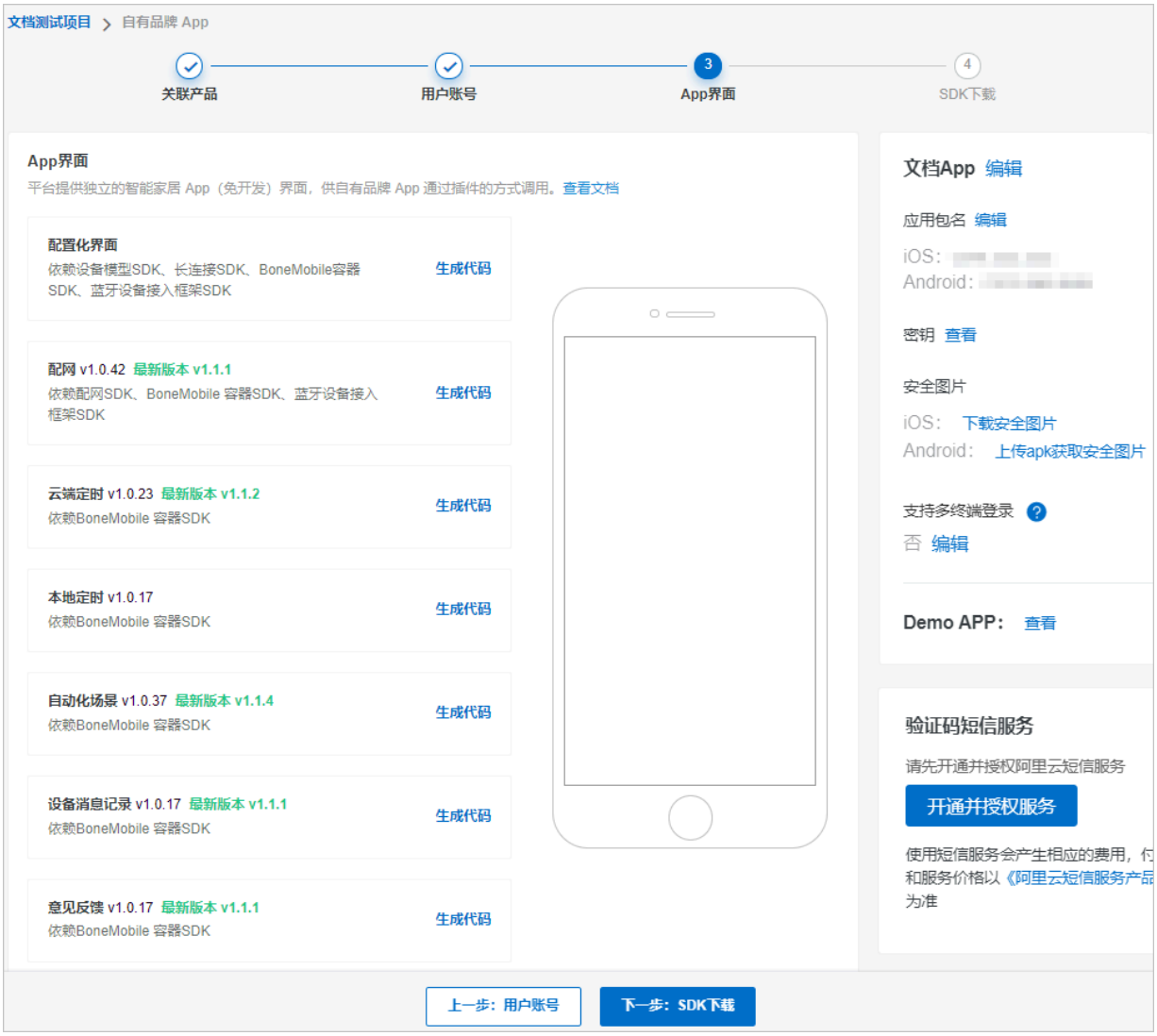
字段说明	类型	描述
openid	String	用户唯一标识（非必填，如果需要支持Google Assistant和Amazon Alexa等语音对接能力，此处需要允许不传递该标识）
nick_name	String	昵称
avatar_url	String	头像URL
gender	String	性别 • 0:未知 • 1: 男性 • 2:女性

3.2 插件使用指南

平台为了让您更快搭建自有品牌App，提供了免开发的App页面插件。您只需通过简单的调用即可呈现一个完整功能。

概述

平台当前提供的插件包括：配网页面（支持WiFi、以太网、蜂窝网）、配置化界面、云端定时、本地定时、场景自动化、消息列表。


文档测试项目 > 自有品牌 App
关联产品
用户账号
3
App界面
4
SDK下载
App界面
平台提供独立的智能家居 App（免开发）界面，供自有品牌 App 通过插件的方式调用。查看文档
配置化界面
依赖设备模型SDK、长连接SDK、BoneMobile容器SDK、蓝牙设备接入框架SDK
生成代码
配网 v1.0.42 最新版本 v1.1.1
依赖配网SDK、BoneMobile 容器SDK、蓝牙设备接入框架SDK
生成代码
云端定时 v1.0.23 最新版本 v1.1.2
依赖BoneMobile 容器SDK
生成代码
本地定时 v1.0.17
依赖BoneMobile 容器SDK
生成代码
自动化场景 v1.0.37 最新版本 v1.1.4
依赖BoneMobile 容器SDK
生成代码
设备消息记录 v1.0.17 最新版本 v1.1.1
依赖BoneMobile 容器SDK
生成代码
意见反馈 v1.0.17 最新版本 v1.1.1
依赖BoneMobile 容器SDK
生成代码
文档App 编辑
应用包名 编辑
iOS:
Android:
密钥 查看
安全图片
iOS: 下载安全图片
Android: 上传apk获取安全图片
支持多终端登录
否 编辑
Demo APP: 查看
验证码短信服务
请先开通并授权阿里云短信服务
开通并授权服务
使用短信服务会产生相应的费用，付
和服务价格以《阿里云短信服务产品
为准
上一步：用户账号
下一步：SDK下载

插件使用

1. 进入项目主页，单击创建的自有品牌App，进入App配置页面。

2. 在自有品牌App的**App界面**中，选择插件，生成代码（此处是插件更新的地方）。

   - 如果第一次使用插件，单击**生成代码**后，将插件代码和参数复制到App中，App才能打开该插件。
   - 当插件右侧出现“最新版本v1.0.xx”的更新字样，表明平台有新版本发布。如需使用最新版本插件功能，可单击进行更新。

3. 复制插件路由代码，粘贴到您的App中。

   - SDK依赖：每个插件依赖一些SDK，也请确保在“SDK下载”中获得这些SDK。
   - 多语言：所有插件都已支持国际化（即App语言切换），插件会跟着一起切换。

   示例代码如下。

   - Android端

     ```
     import android.os.Bundle;
     ```

```
import com.aliyun.iot.aep.component.router.Router;

...

String code = "link://router/devicenotices";
Bundle bundle = new Bundle();
bundle.putString("key","value"); // 传入插件参数，没有参数则不需要这一行
Router.getInstance().toUrlForResult(getActivity(), code, 1, bundle);
```

- iOS端

```
#import <IMSBoneKit/IMSRouter/IMSRouter.h>
[[IMSRouterService sharedService] openURL:[NSURL URLWithString:@跳转的url]
options:nil];

...

NSURL *url = [NSURL URLWithString:@"link://router/connectConfig"];
NSDictionary *options = @{@"key": @"value"}; // 传入插件参数，没有参数则不需要这一
行
[[IMSRouterService sharedService] openURL:url options:options completionHandler
:^(BOOL success) {
    if (!success) {
        [self ims_showHUDWithMessage:[NSString ls_loadFailed]];
    }
}];
```

插件更新

如果插件配合您的设备固件（如本地定时），请确保固件升级与旧版本的兼容，再单击插件更新，否则会引起线上问题，请谨慎操作。

插件参数介绍

- 配置化界面

参数	类型	描述
iotId	String	（必须）设备 ID

- 配网插件

参数	类型	描述
productKey	String	（必须）产品 ProductKey
deviceName	String	（GPRS设备必选）设备名称

- 云端定时

参数	类型	描述
iotId	String	（必选） 设备 ID

- 本地定时

参数	类型	描述
iotId	String	（必选）设备 ID

- 自动化场景

参数	类型	描述
sceneType	enum	场景的类型，请根据场景服务的版本选择对应的参数，默认为IFTTT - 场景服务1.0仅支持 ilop（单品）、hc（全屋） - 场景服务2.0仅支持 IFTTT、CA

参数	类型	描述
sceneId	String	仅在编辑时传入

- 不传**sceneId**时为新增场景

新建场景
保存
场景图标
如果
满足以下任一条件
手动触发
点击“执行”按钮触发场景
并且
满足以下所有条件
就
执行

- 传sceneId时为编辑场景

向手机发送通知
保存
场景开关
场景名称
向手机发送通知
场景图标
如果
满足以下任一条件
手动触发
点击“执行”按钮触发场景
并且
满足以下所有条件
就

- 设备消息记录

参数	类型	描述
iotId	String	设备ID

  - 传设备ID时，单个设备的消息记录

9:41
消息记录
设置
所有日期
所有类别
12/14 星期一
设备名称
信息内容
15:33
设备名称
信息内容
15:33
12/14 星期日
设备名称
信息内容
15:33

- 不传设备ID时，用户绑定的所有设备消息记录

9:41
消息记录
设置
智能插座
所有时间
所有类别
12/14 星期一
设备名称
信息内容
15:33
设备名称
信息内容
15:33
12/14 星期日
设备名称
信息内容
15:33

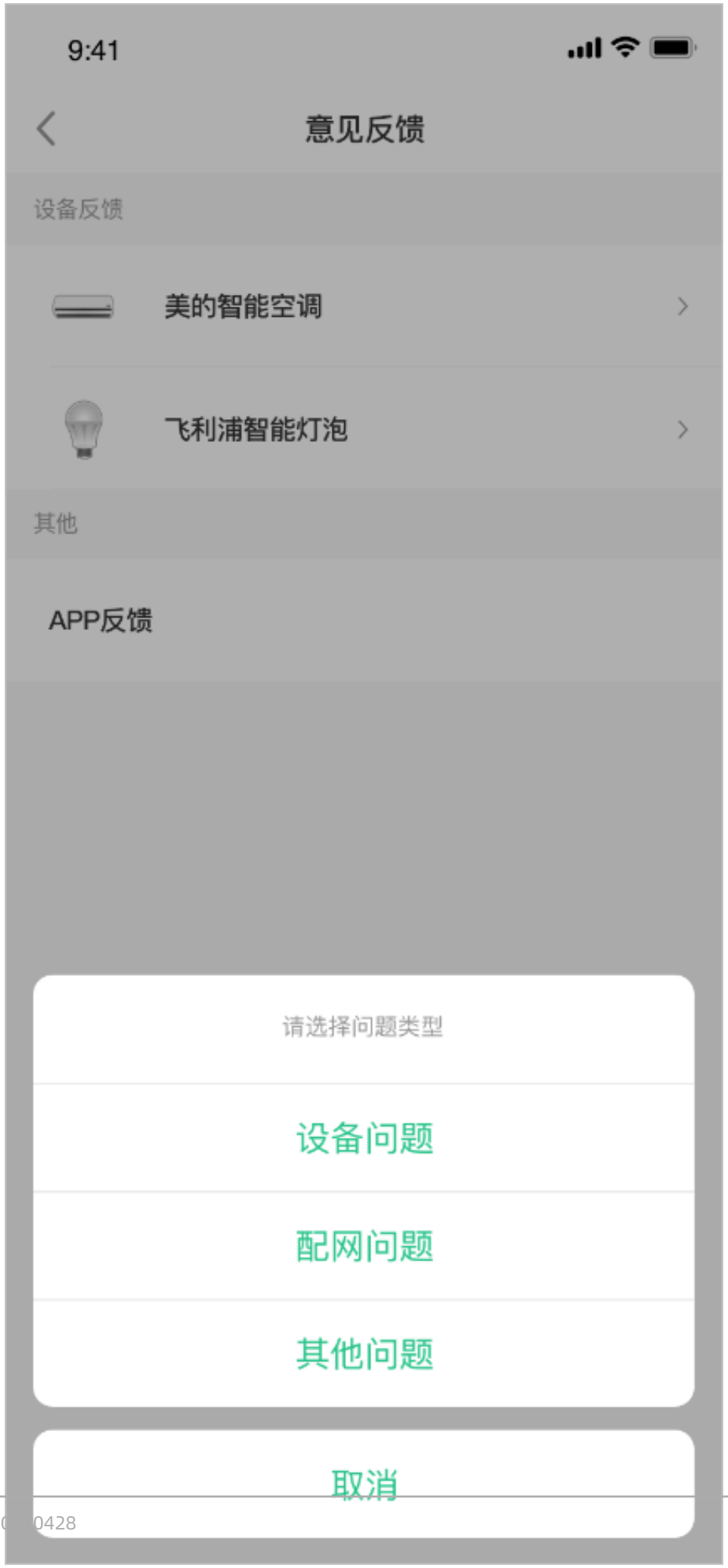
- 意见反馈插件

参数	类型	描述
mobileModel	String	手机型号
mobileSystem	String	手机系统
appVersion	String	App版本

意见反馈说明

- App端用户
  - 反馈首页显示和当前用户有关联的设备，单击设备选择反馈类型。

  - 选择设备的反馈类型（App的问题类型为：反馈故障，功能建议，其他问题）。


9:41
意见反馈
设备反馈
美的智能空调
飞利浦智能灯泡
其他
APP反馈
请选择问题类型
设备问题
配网问题
其他问题
取消

- 填写反馈信息，问题描述为必填项

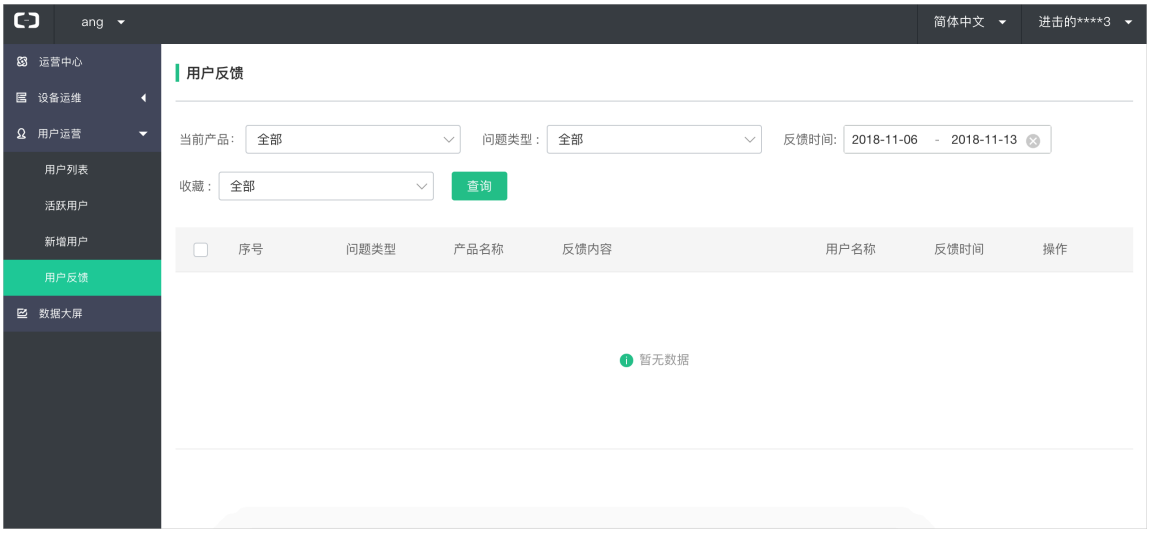
9:41
填写反馈信息
设备问题–美智能的空调
请详细描述您遇到的问题或建议
0/200
联系方式
请留下电话或邮箱，方便我们联系您
提交

- 反馈成功后可在反馈记录中查看。（当出现红点时说明有新的回复）

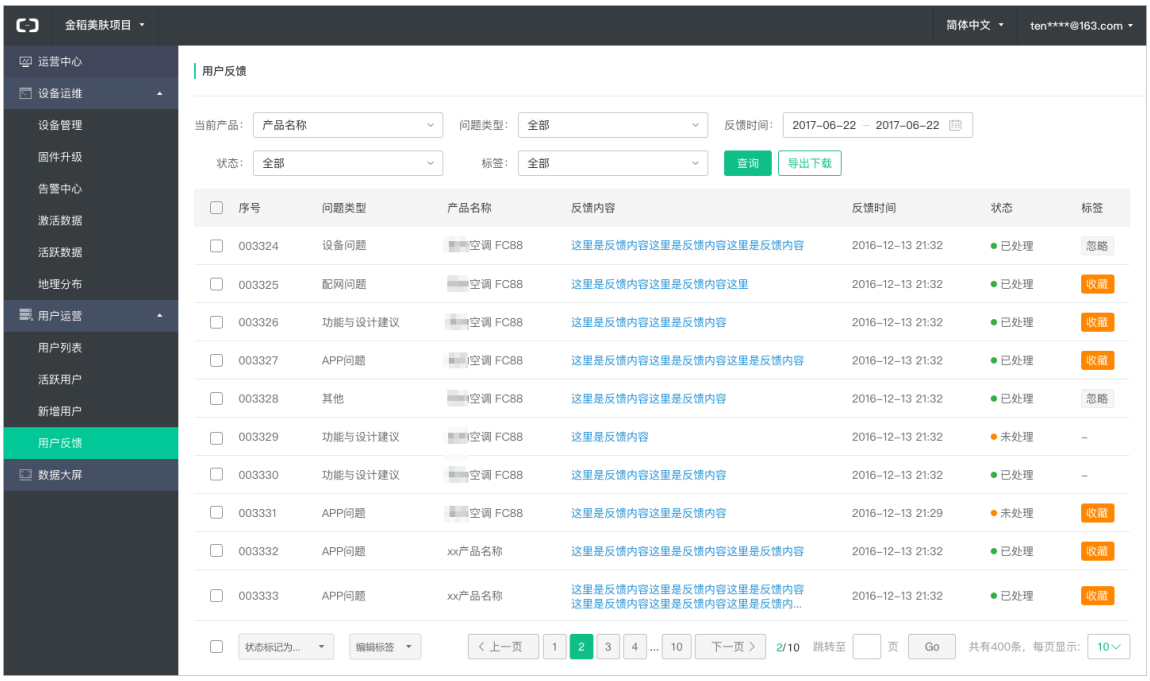
- 在反馈记录中查看反馈

9:41
反馈记录
空调连不上网络
设备问题–智能空调
2018/09/04
谢谢您对阿里云移动服务的支持与...
配网问题–智能空调
2018/09/04

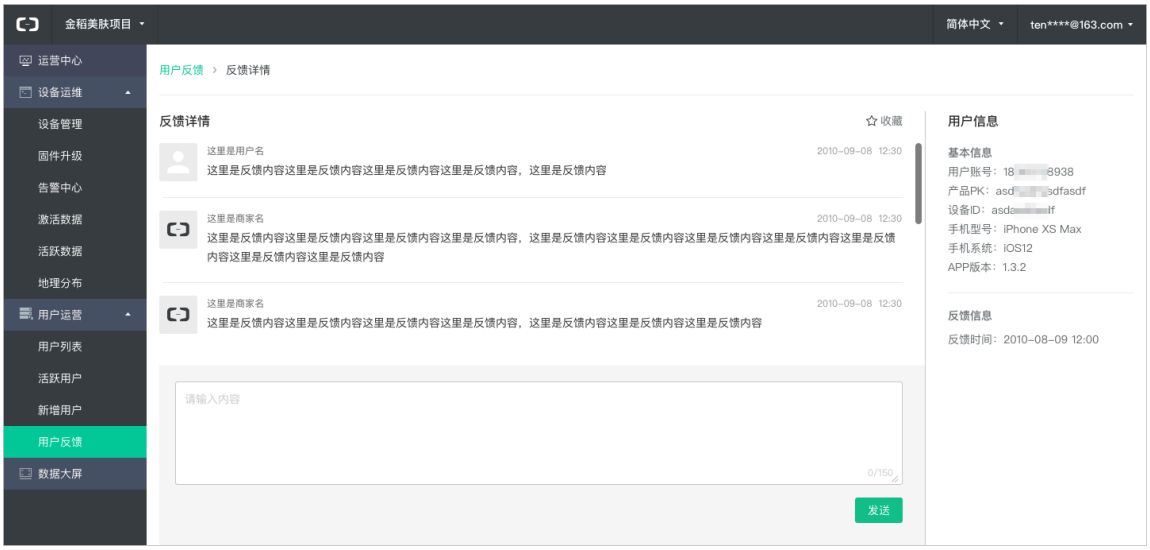
- PC端商家用户
  - 商家通过生活物联网的控制台，在运营中心中查看用户反馈（商家账号显示产品名称，小二账号不显示）



  - 单击反馈内容可查看反馈详情



  - 反馈详情页，可对用户反馈内容进行回复



3.3 配网开发指南

本节将会介绍如何开发App端的设备配网、以及和用户绑定的功能。

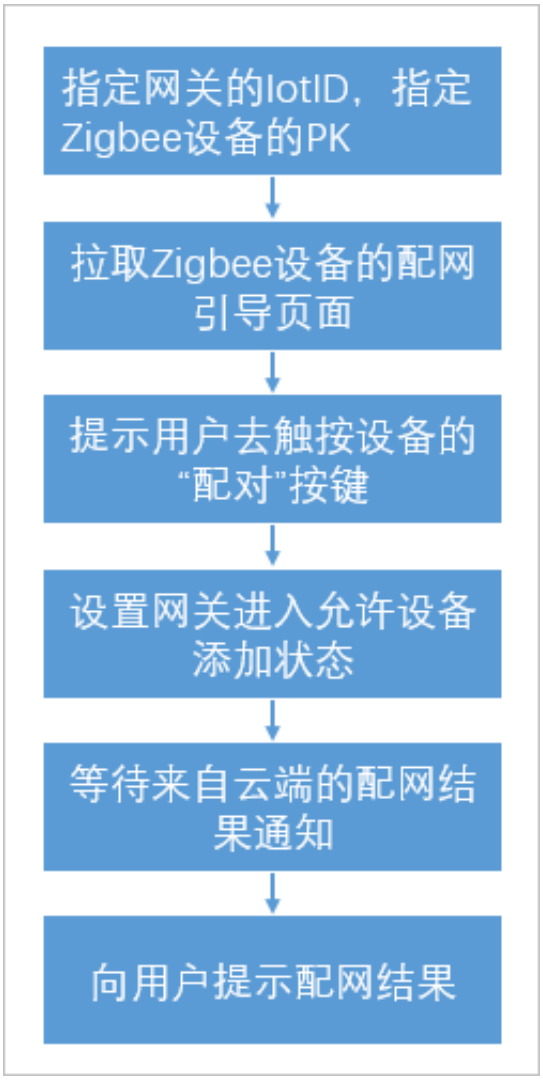
- 配网部分，提供了两种页面的开发方式：
  - 基于配网SDK自行开发配网界面（设备配网和设备绑定）
  - 调用平台提供的配网插件
- 绑定部分，介绍了不同设备类型的绑定接口：
  - 基于token方式的设备绑定（WiFi、以太网）
  - 基于时间窗口方式的设备绑定（蜂窝网、蓝牙、Zigee、其他）

基于配网SDK自行开发配网界面

- WiFi设备配网界面开发
  - 配网方案介绍参见：WiFi 配网方案介绍
  - Android APP SDK： 配网SDK
  - iOS APP SDK：配网SDK
  - 最佳实践：#unique_59

- Zigbee设备配网界面开发

  Zigbee设备的配网指的是将Zigbee设备连接到某个Zigbee网关的过程，Zigbee的配网流程为：



  - 指定网关的IoTID

    开发者可以根据用户绑定的设备去拉取用户绑定的Zigbee网关设备，并让用户从中选择希望接入Zigbee设备的网关，从而明确网关的**IoTID**。

    拉取用户已绑定设备列表的接口为：/uc/listBindingByAccount，详细调用方法如下。

    ■ Android系统：Andriod API通道使用说明

    ■ iOS系统：iOS API通道使用说明

  - 指定Zigbee设备的ProductKey

    开发者可以通过扫码方式获取设备的**ProductKey**，或者通过列出Zigbee产品列表让用户选择的方式获知Zigbee设备的**ProductKey**。

  - 配网引导

由于不同Zigbee设备的**配对**按钮位置与外观不同，厂家要将自己Zigbee设备的**配对**按键的指示图片放在生活物联网平台上，开发者可以通过待配网Zigbee设备的**ProductKey**去生活物联网平台获取设备的配网引导页面，显示给用户显示如何启动Zigbee设备的配网。

获取产品配网引导页面的接口为：/awss/enrollee/guide/get，详细调用方法参见 产品管理服务。

- 网关添加子设备

  调用云端接口让网关进入允许添加设备状态，并等待来自云端的配网结果。

  设置网关允许添加设备的接口为：/thing/gateway/permit，详细调用方法参见配网服务。

  等待配网结果的topic为：/thing/topo/add/status，详细调用方法如下。

  ■ Android系统：Andriod API通道使用说明

  ■ iOS系统：iOS API通道使用说明

  最后将接收到的配网结果向用户显示即可。

- 以太网设备开发

  对于接入设备端SDK的以太网设备，配网SDK提供以太网设备的发现和获取绑定token的能力。

  - 以太网设备发现

    以太网设备的发现依赖配网SDK来实现，详细调用方法如下。

    ■ Android系统：配网SDK以太网设备发现

    ■ iOS系统：配网SDK以太网设备发现

  - 以太网设备绑定

    以太网设备的绑定分为两个部分：获取绑定token和调用绑定接口。

  - 获取以太网设备绑定token

    以太网设备获取token和WiFi设备获取绑定token的逻辑是一致的，详细调用方法如下。

    ■ Android系统：配网SDK以太网设备获取绑定token

    ■ iOS系统：配网SDK以太网设备获取绑定token

调用平台提供的配网插件

在**产品-人机交互**的**配网引导**中，手动填写配网引导页面中的图片和文字。该插件中兼容了WiFi、以太网和蜂窝网的配网流程。其中WiFi仅支持一键配网、设备热点配网和手机热点配网，如需其他配网方式，请使用基于配网SDK开发配网界面。

调用配网的插件如下。

- 插件ID国内版：a123kfz2KdRdrfYc
- 插件ID国际版：a223c2beCJQ2Xpk2

插件入参参数介绍如下。

属性	说明	类型	是否必选	默认值
productKey	产品的唯一标识符	String	是	无
deviceName	同一产品下， 设备的唯一标示符。本地发现设备进入配网、GPRS设备扫码配网时需用到该参数	String	否	无
token	设备令牌，用于绑定设备时鉴权，本地发现已配网设备是需使用该参数	String	否	无
addDeviceFrom	标识该设备是由路由器发现的或零配设备发现的待配设备，调用本地发现接口时会返回该值。配置如下 • 零配：ZERO_DEVICE • 路由器配网：ROUTER	String	否	无

插件出参，退出配网流程：

属性	类型	必选	默认值
productKey	String	是	无
deviceName	String	是	无

代码示例

- Android代码示例：

```
// 启动插件
Bundle bundle = new Bundle();
bundle.putString("productKey", pk);
Router.getInstance().toUrlForResult(activity, "link://router/connectConfig",{your_reque
st_code}, bundle);

// 接收配网结果
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  ...
  if (REQUEST_CODE_PRODUCT_ADD == requestCode) {
    if (Activity.RESULT_OK != resultCode) {
      // 配网失败
      return;
    }
```

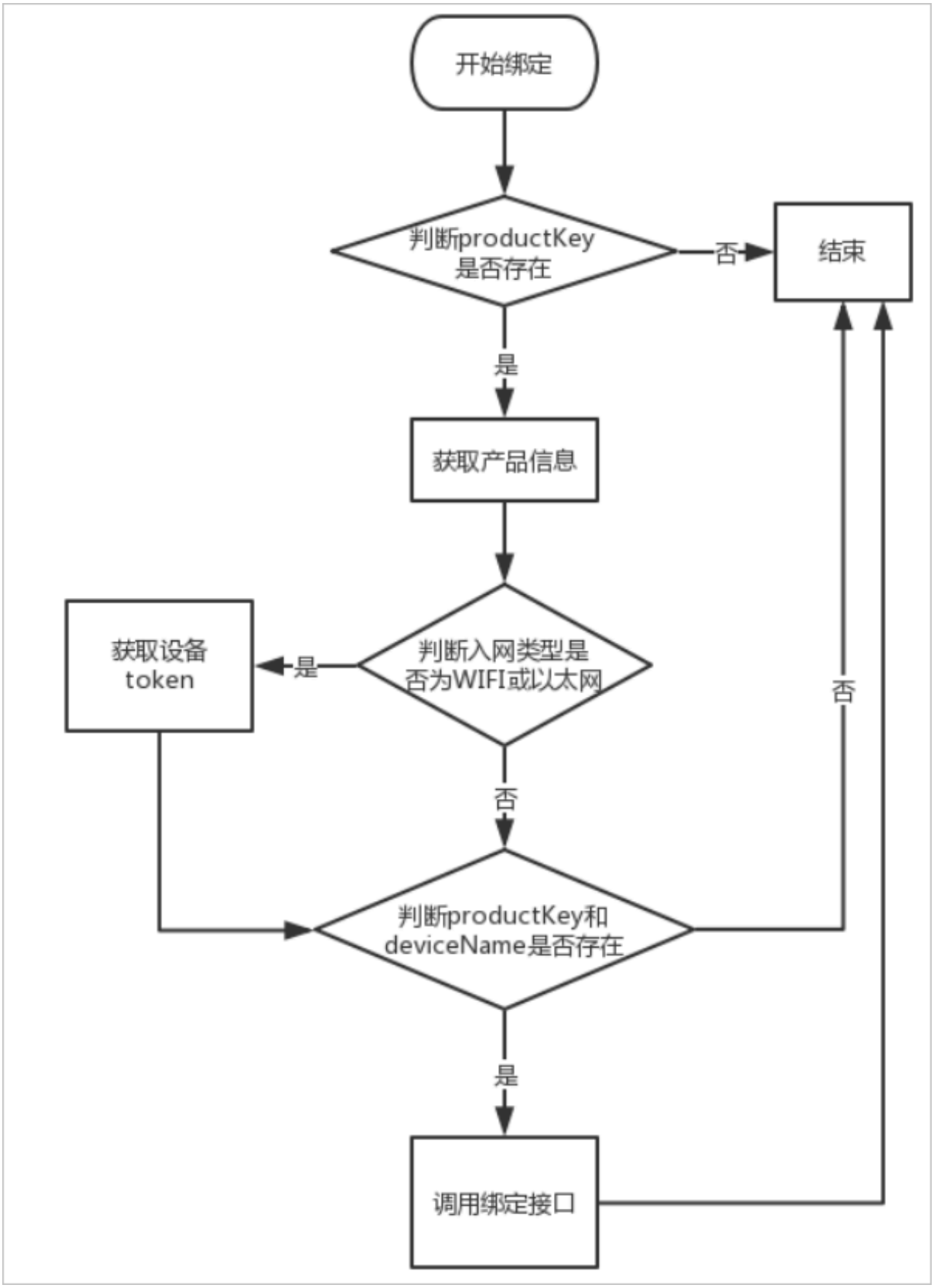
```
        String productKey = data.getStringExtra("productKey");
        String deviceName = data.getStringExtra("deviceName");
        // 配网成功
    }

}
```

- iOS代码示例：

```
NSMutableDictionary *options = [NSMutableDictionary dictionary];
options[@"productKey"] = device.productKey;
options[@"deviceName"] = device.deviceName;
options[@"token"] = device.token;
options[@"addDeviceFrom"] = device.addDeviceFrom;

IMSRouterCallback block = ^(NSError *error, NSDictionary *info) {
    [self.navigationController popToViewController:self animated:YES];

    if (error) {
        // 处理错误信息
    } else if (info && [info count] > 0) {
        // 配网成功
    } else {
        // 配网正常退出...
    }
};
options[AKRouterCompletionHandlerKey] = block;
// 配网插件id
NSURL *url = [NSURL URLWithString:@"link://router/connectConfig"];
//NSString *pluginId = @"a123kfz2KdRdrfYc"; //以国内版插件Id举例，此方法是老方法，现在统一用 router的方式调用
//NSURL *url = [NSURL URLWithString:[NSString stringWithFormat:@"link://plugin/%@", pluginId]];
[[IMSRouterService sharedService] openURL:url options:options completionHandler:^(BOOL success) {
    if (!success) {
        // 进入配网插件失败
    }
}];
```

设备绑定

设备绑定的流程图如下。



API相关介绍如下。

- WiFi、以太网，使用API “基于token方式的设备绑定”。
- 蜂窝网、蓝牙、Zigee、其他，使用API “基于时间窗口方式的设备绑定”。

- 具体配网相关API，如本地发现、配网、绑定等，请查看配网服务。
- 具体和用户关系API，如解绑、查询绑定关系、分享设备等，请查看用户服务。
- Android获取设备token代码示例如下。

```
LocalDeviceMgr.getInstance().getDeviceToken(productKey, deviceName, 60*1000, new
IOnDeviceTokenGetListener() {
    @Override
    public void onSuccess(String token) {
        // 拿到绑定需要的token
        //TODO 用户根据具体业务场景调用
    }

    @Override
    public void onFail(String reason) {
    }
});
```

- iOS获取设备token代码示例如下。

```
[[IMLLocalDeviceMgr sharedMgr] getDeviceToken:self.productKey deviceName:self.
deviceName timeout:60 resultBlock:^(NSString *token, BOOL boolSuccess) {
    if (token && boolSuccess) {
        // 获取token成功
    } else {
        // 获取token失败
    }
}];
```

3.4 移动应用推送开发指南

当您开发自有品牌App的消息推送时，请按照本文档的指导来配置移动应用服务，并开发移动应用的推送功能。

前提条件

- 在控制台产品开发的**人机交互**页面中，已设置了设备告警规则（例如，门锁开启时给App推送一条消息；或当PM2.5超过200时给App推送一条消息）。详细请参见#unique_66。
- 已完成自有品牌App的创建，详细请参见#unique_67。
- 下载SDK套餐包时勾选了消息推送，详细请参见#unique_68。

背景信息

阿里云移动推送服务为Alibaba Cloud Mobile Push，在线送达率超过98%，承载了双十一超过5000条/秒的消息推送峰值。具体产品信息请参见移动推送。生活物联网平台基于阿里云移动推送服务，整合了设备管理能力，将设备的消息推送到App上。

配置移动应用推送服务

1. 进入生活物联网平台控制台。
2. 选择项目名称，在项目主页面，单击已创建的自有App名称。

3. 进入自有品牌App开发的**SDK下载**页面，单击消息推送对应的**设置**。

4. 配置iOS应用的消息推送服务。

   iOS应用的推送服务没有在线推送和离线推送之分，需统一采用苹果官方提供给开发者的推送服务。请您根据以下步骤操作。

   a) 单击**上传文件**，并上传APNs推送证书。

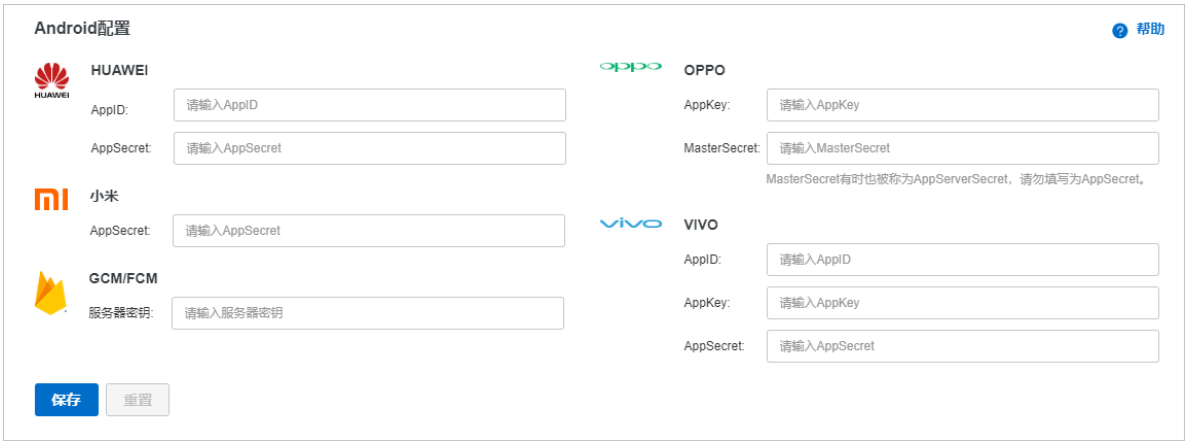
   b) 输入推送证书的密码，并单击**验证证书**。

   只有正确配置了证书和密码，提示验证成功后，才能正常调用iOS推送服务。

   c) 单击**测试推送**，在弹出的输入框中填入deviceToken信息（参见SDK初始化中的didRegisterForRemoteNotificationsWithDeviceToken方法），并单击**测试**，可以推送一条消息进行测试。

5. （可选）配置Android离线推送服务。

   Android应用默认支持在线推送，无需额外配置。针对目前市场上的常用移动设备（华为、小米、FCM、OPPO、VIVO），平台还提供了Android应用离线推送能力。



> **说明：**
> 以下为三方品牌消息推送大致的配置步骤，仅供您参考，具体操作请根据三方品牌的实际情况来定。

- 华为

  a. 在华为开发者联盟注册App。应用审核通过后，可得到华为的AppID和AppSecret。

  b. 打开华为开发者平台push功能区的消息推送开关。

  c. 在华为开发者平台添加消息回执回调地址为https://agoo-ack.m.taobao.com/hw/。

  d. 在生活物联网控制台**消息推送设置**页面（如上图所示）配置华为AppID和AppSecret。

  > **说明：**
  > 确保您在华为控制台激活了推送通道功能，且您的App处于审核中或通过审核的状态（不能为草稿状态），否则通道不会生效。

- 小米

  a. 在小米开放平台注册App，得到相应的小米AppID、小米AppKey、小米AppSecret。

  b. 打开小米开发者平台push功能区的消息推送开关。

  c. 在生活物联网控制台**消息推送设置**页面配置小米AppSecret。

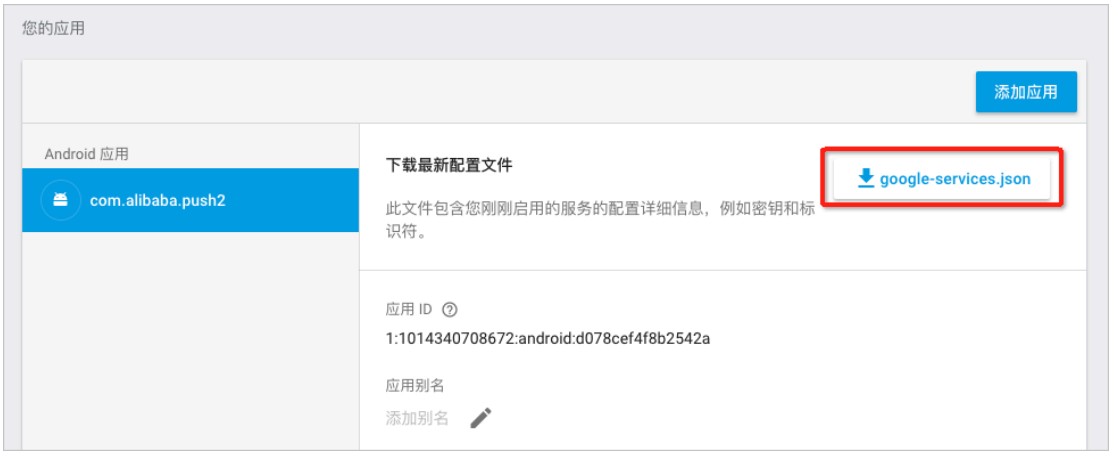
- FCM（Google已将GCM推送迁移至Firebase，改称FCM）

  > **说明：**

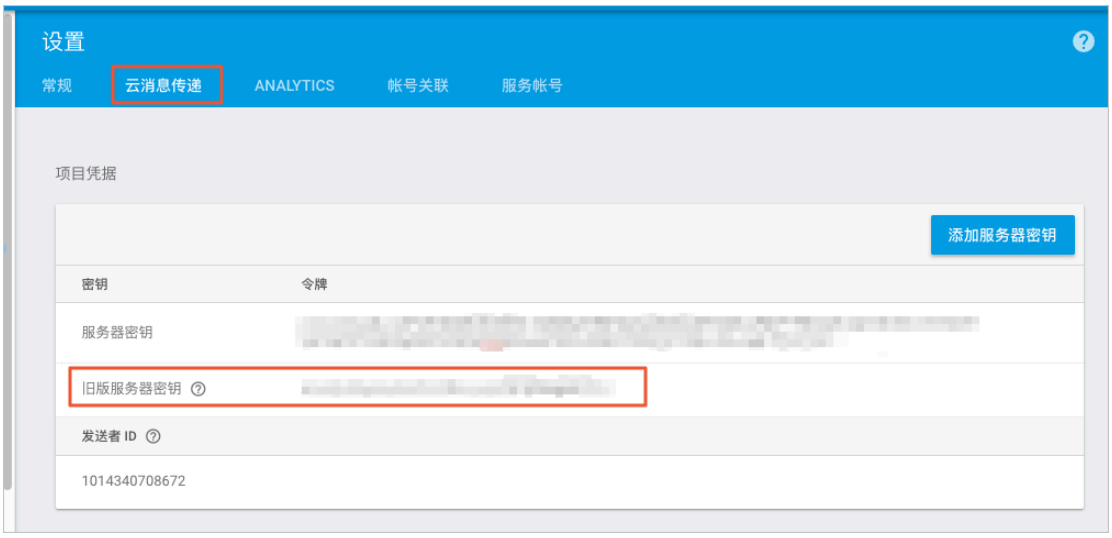
接入前确保手机已安装Google Play Services，否则无法注册成功（目前大部分国内手机的谷歌服务被剥离了）。

a. 在FCM推送平台中创建一个项目，并在项目下新增App。

b. 下载对应App的google-services.json文件，获取json文件中的**project_number**、**mobilesdk_app_id**这两个key对应的value，分别记录为sendId/applicationId（下面注册过程中需要用到）。



c. 在Firebase控制台获取服务器密钥。



d. 在生活物联网控制台**消息推送设置**页面设置GCM/FCM的服务器密钥。

- OPPO

a. 在OPPO市场上架应用，且应用评级为A（评级为A才能使用推送服务，具体政策可咨询OPPO客服）。

b. 在OPPO开放平台应用配置注册OPPO企业开发者账号，得到相应的AppKey和MasterSecret。

c. 在OPPO开放平台添加应用并开通OPPO推送服务。

d. 在生活物联网控制台**消息推送设置**页面设置您的OppoAppkey和OppoMasterSecret。

- VIVO

  > **说明：**
  > 国际站的自有品牌App中，暂不支持VIVO推送。

  a. 在VIVO开放平台注册VIVO账号及创建应用（需要使用企业开发者账号）。审核通过后，能够得到应用的AppID、AppKey和AppSecret。

  b. 在生活物联网控制台**消息推送设置**页面设置VIVO应用的AppID、AppKey以及AppSecret。

开发应用推送

请您参照以下SDK开发移动应用推送功能。

- Android参见移动应用推送SDK。
- iOS参见移动应用推送SDK。

推送消息介绍

生活物联网平台提供的SDK支持推送的消息分为以下两类。

- 设备告警消息

  在控制台中配置告警消息，当设备触发该消息规则时，会主动给App推送一条消息。

  推送范围：该设备的所有用户，包括设备的管理者、以及设备的被分享者。

  配置方式：在平台**产品-人机交互**的**设备告警**中，配置设备告警信息，详细参见#unique_66。

- 设备分享消息

  当设备的管理员（即ownerName，以下示例中的甲）将设备分享给其他用户（即memberName，以下示例中的乙）时，平台会主动给App的用户推送设备分享相关的消息。

  - ownerName：设备拥有者别名标识（例如用户名称）
  - memberName：被分享者（或非拥有者）别名标识（例如用户名称）

  设备分享相关的消息包括以下六种情形。在设备分享过程中，分别收到的消息以示例说明如下。

消息类型	设备分享的场景描述
设备拥有者向其他用户发起设备分享	甲向乙发起设备分享时，乙收到消息：ownerName+“向您共享设备”。
设备拥有者取消设备分享	甲向乙取消设备分享时，乙收到消息：ownerName+“取消了设备共享”。

消息类型	设备分享的场景描述
被分享者接收分享消息	乙接收甲的设备分享时，甲收到消息：memberName+“添加了您的设备”。
被分享者拒绝分享消息	乙拒绝甲的设备分享时，甲收到消息：memberName+“取消了设备共享”。
设备被抢占	乙抢占了甲的设备时，甲收到消息：memberName+“抢占了您的设备”。
发起者已解绑	乙接收甲分享的设备后，甲取消设备分享时，乙收到消息：ownerName+“删除了您的设备”。

说明：

Android应用中，仅“设备拥有者向其他用户发起设备分享”支持离线推送和在线推送，其余设备分享相关的消息都仅支持在线推送。

3.5 场景自动化开发指南

通过平台提供的客户端的场景自动化API，App开发者可以实现场景自动化功能，让用户可以自行配置一个自动化任务，当触发某些条件时，执行某些任务。例如，用户可以通过App设置：到达早上6时空调自动关闭的任务。

名词解释

- 场景：需要用户手动在App上单击触发后，执行一系列任务。如，用户可以创建一个“回家模式”，包括打开灯、打开空调、打开电视、拉开窗帘等多个任务。当用户在App上单击执行**回家模式**场景时，所有任务将被执行。
- 自动化：不需要用户手动触发，当符合某些条件时，系统自动执行一系列任务。如，用户可以创建一个“自动开启摄像头”的任务，当触发条件“红外人体传感器识别到有人经过”时，自动执行“摄像头开始录像”的任务。
- IFTTT：If this then that规则，即自动化。
- Trigger：自动化中的触发器
- Condition：自动化中的触发条件
- Action：场景和自动化中的任务执行
- CA模型（推荐）：当满足Condition（多个Condition之间可设规则，all：当所有触发条件都满足，any：任一触发条件满足）的触发条件时，执行Action。

- TCA模型（旧模型）：当Trigger（多个Trigger之间是and关系）触发，且满足Condition（多个Condition之间是or关系）的触发条件时，执行Action。

开发指南

App端的场景自动化功能，可以通过两种方式实现。

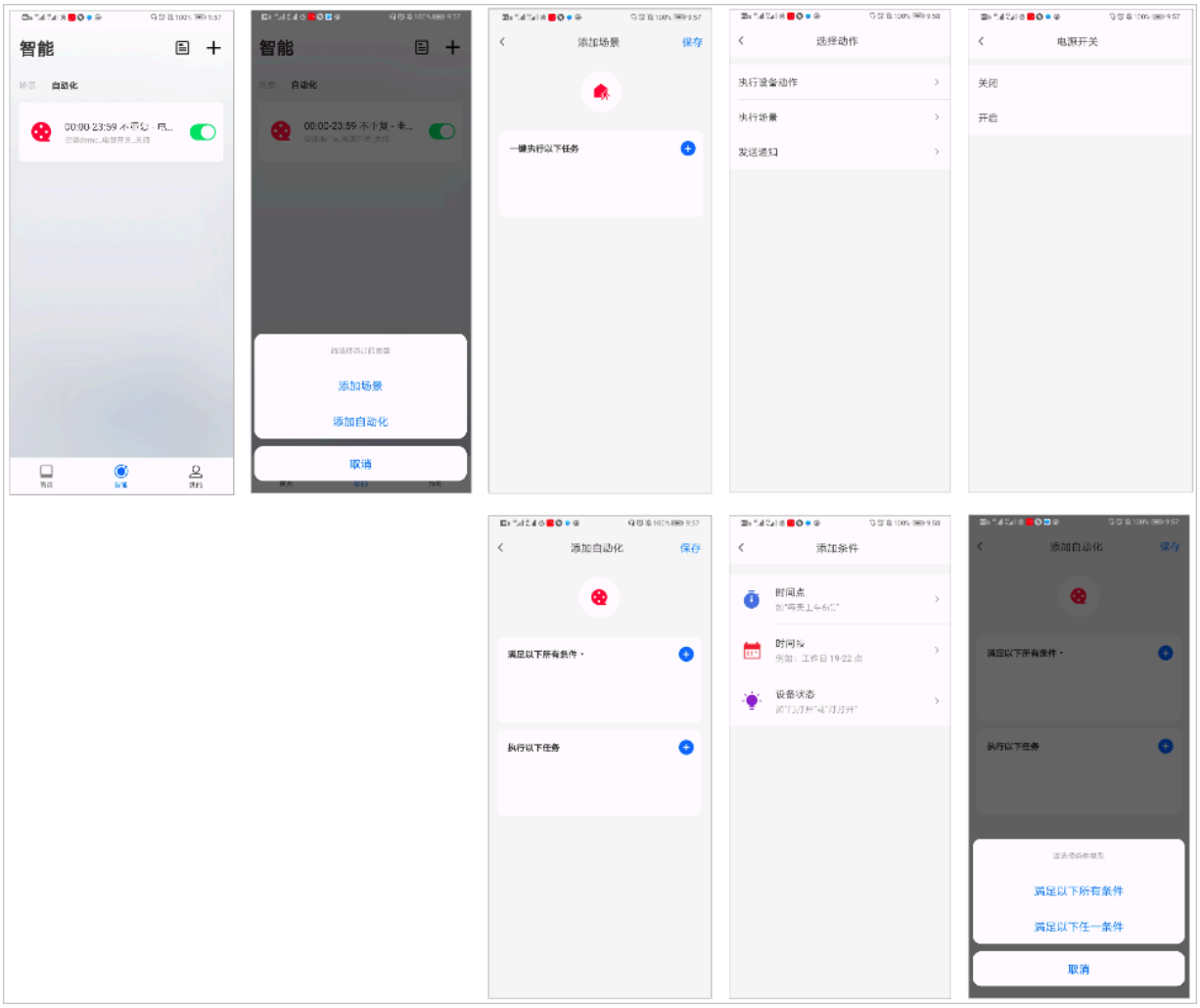
- 基于场景服务2.0API自行开发页面
- 调用场景插件

基于场景服务2.0API自行开发页面

支持功能如下。

- 场景的创建、删除、查看、修改
- 自动化的创建、删除、查看、修改
- 触发条件支持：时间点、时间段、设备动作（后续会扩展更多）
- 触发条件的关系：满足所有条件、满足以下任意条件
- 执行任务支持：设备动作、执行场景、发送手机推送通知

开发后效果实例如下图，该实例基于CA模型开发（不包括界面，界面需要开发者基于API自行开发），具体参见场景服务2.0。



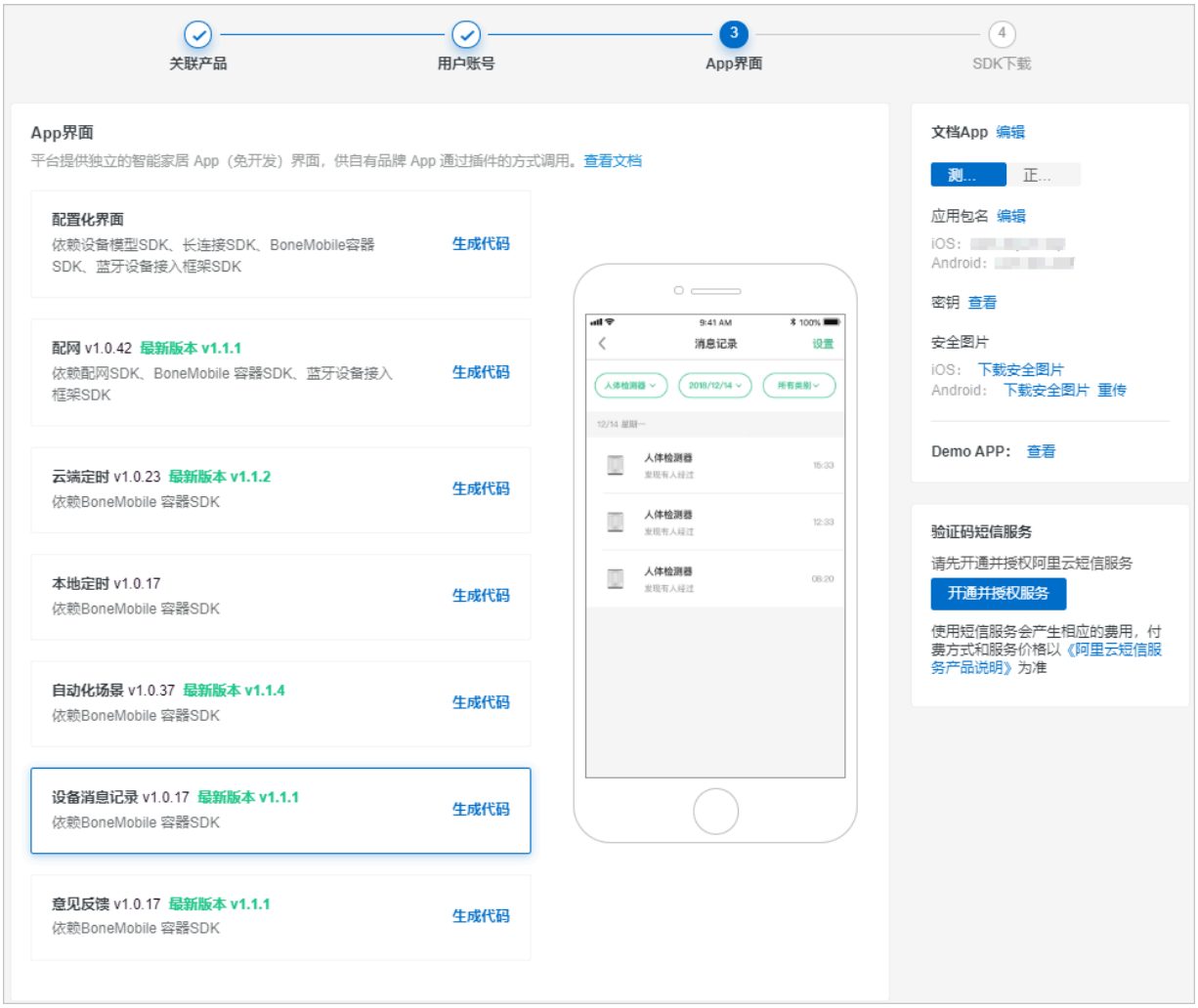
调用场景插件

支持功能如下：

- 场景的创建、删除、查看、修改
- 自动化的创建、删除、查看、修改
- 触发条件支持：时间点、设备动作
- 触发条件的关系：TCA模型，需要用户分别设置Trigger、Condition和Action
- 执行任务支持：设备动作、执行场景、发送手机推送通知

在平台**自有品牌App-App界面**中找到该插件以及调用方法。该插件包含了自动化场景功能的完整界面，可简化App的开发工作量。具体请查看插件使用指南。

该插件基于TCA模型，所以如果开发者希望实现更灵活和体验更佳的功能和界面，建议使用基于场景服务2.0API自行开发界面。

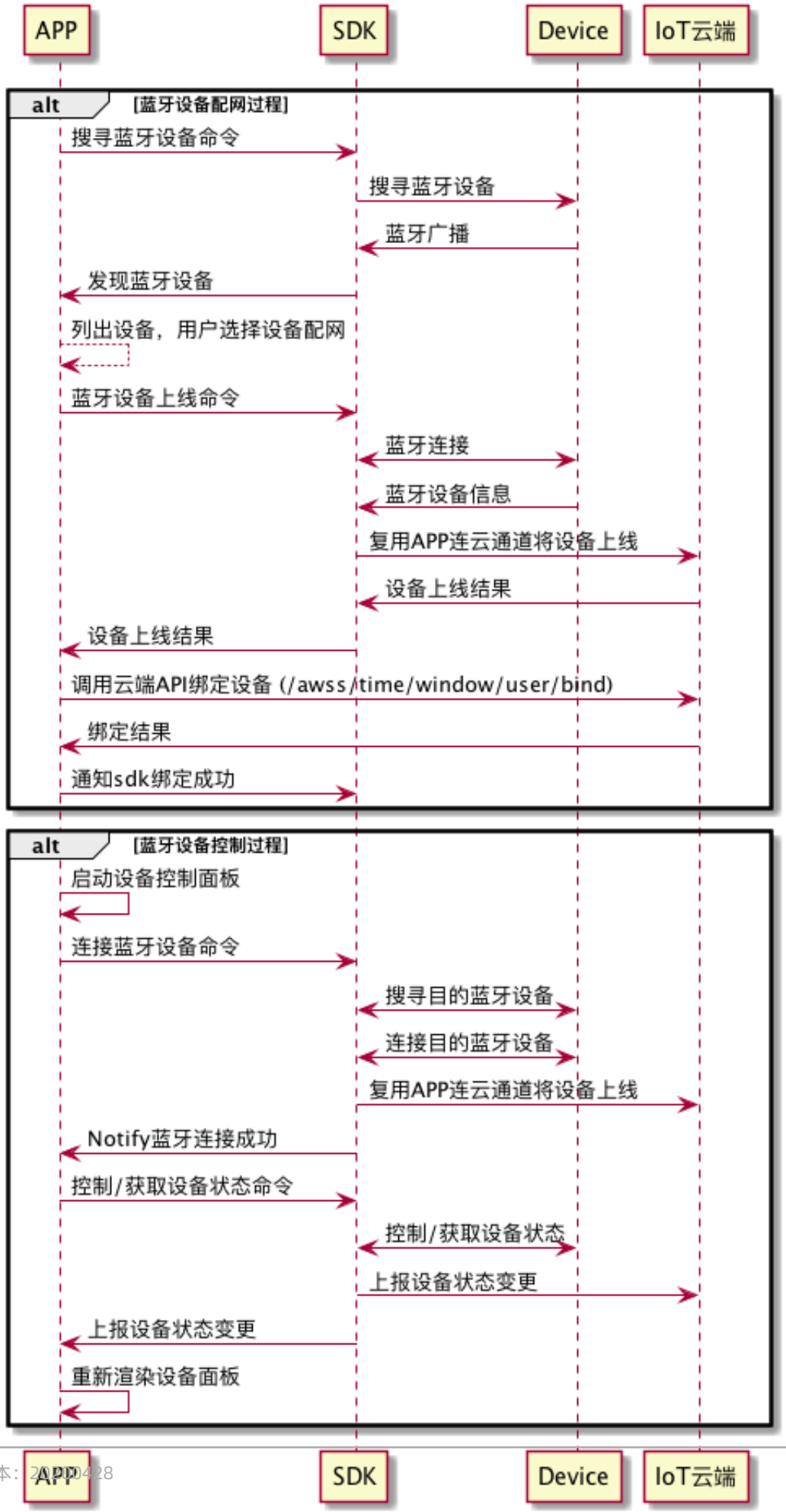


3.6 蓝牙连接开发指南

对于蓝牙设备，阿里物联网平台提供了一套完整的解决方案，即蓝牙设备接入框架。

概述

蓝牙设备接入框架的流程图如下所示。


APP
SDK
Device
IoT云端
alt
[蓝牙设备配网过程]
搜寻蓝牙设备命令
搜寻蓝牙设备
蓝牙广播
发现蓝牙设备
列出设备，用户选择设备配网
蓝牙设备上线命令
蓝牙连接
蓝牙设备信息
复用APP连云通道将设备上线
设备上线结果
设备上线结果
调用云端API绑定设备 (/awss/time/window/user/bind)
绑定结果
通知sdk绑定成功
alt
[蓝牙设备控制过程]
启动设备控制面板
连接蓝牙设备命令
搜寻目的蓝牙设备
连接目的蓝牙设备
复用APP连云通道将设备上线
Notify蓝牙连接成功
控制/获取设备状态命令
控制/获取设备状态
上报设备状态变更
上报设备状态变更
重新渲染设备面板
APP
SDK
Device
IoT云端

蓝牙设备接入框架可支撑以下两种形式的蓝牙接入。

- 阿里云IoT定义一套蓝牙通用规范，简化了厂商进行复杂蓝牙设备的开发（如：智能手表，蓝牙门锁等）

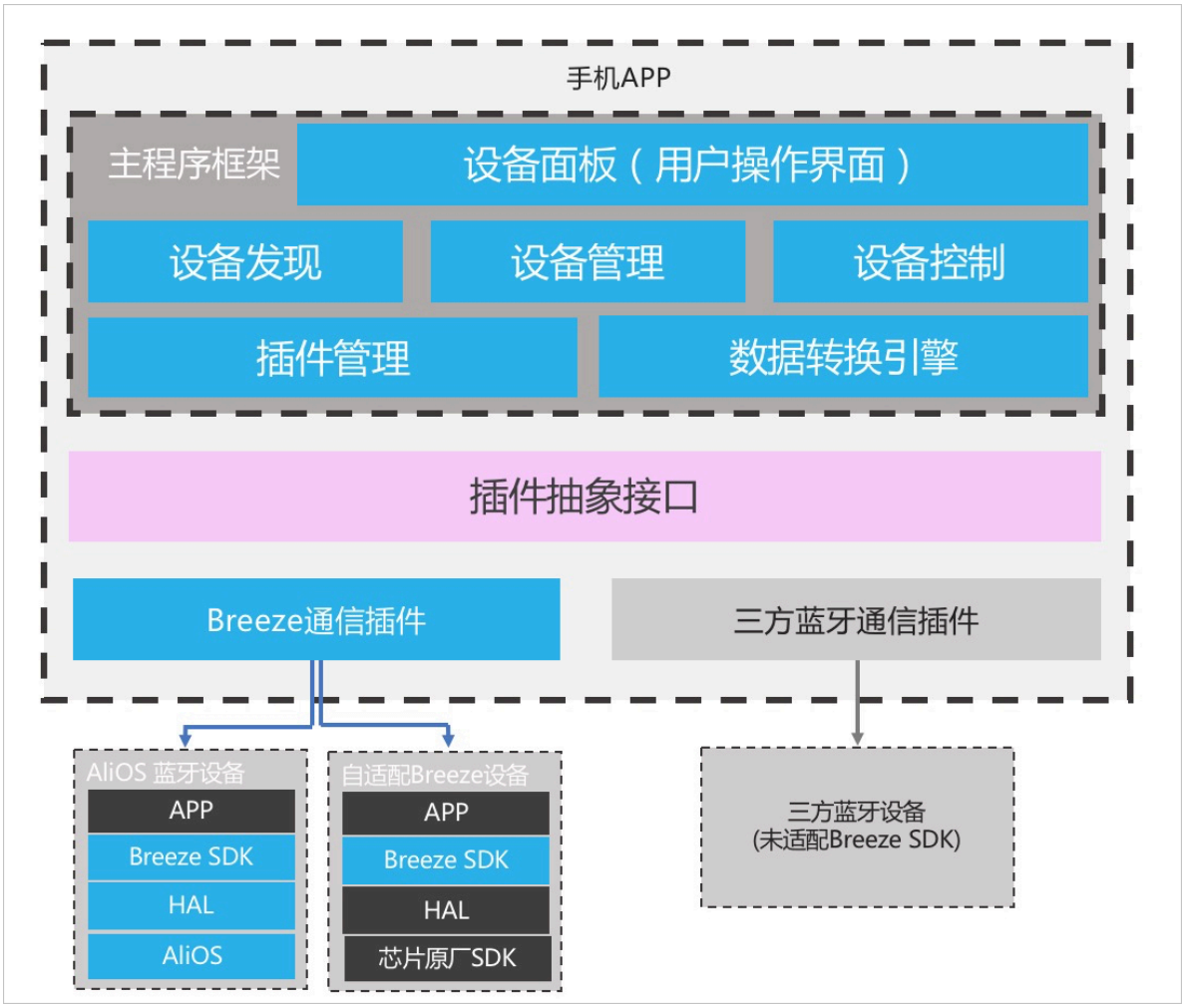
  该规范对蓝牙广播，服务，通信协议等进行了标准化，简称为蓝牙Breeze方案。蓝牙Breeze方案提供了手机端和设备端的SDK，主要包含设备管理，设备发现，加密通信，大数据传输等功能。并对蓝牙芯片和主流手机进行了严格的认证，保证了兼容性和稳定性。

  此接入方案需要按蓝牙设备端开发开发您的设备端。

- 开发者也可以不采用蓝牙Breeze方案，使用自己的私有蓝牙协议通信

  开发者可以开发三方蓝牙通信插件来跟第三方的蓝牙设备进行通信，而无需要在蓝牙设备里集成阿里云IoT提供的设备端Breeze SDK，如下图所示。

  第三方蓝牙通信插件的开发，参见#unique_74。



  框架中主要模块介绍如下。

  - 设备面板：即设备控制的相关UI。
  - 设备发现：蓝牙设备扫描，在这个过程中需要扫描本地的蓝牙设备，并获取设备的**ProductKey**和**DeviceName**。
  - 设备控制：用户操作设备的相关流程。
  - 插件抽象接口：约定了开发者在移动端开发三方蓝牙通信插件时必须实现的API，请参见#unique_74中的相关示例代码。
  - 数据转换引擎：当设备端的产品功能定义没有按阿里云IoT建议的物模型进行设计时，开发者需要在生活物联网控制台创建产品时上传数据格式进行脚本转换，App端的数据格式转换引擎会

在Runtime将脚本下载下来，在数据从手机App将发往设备会做一次数据转换，转换成设备端能识别的格式，同理设备上报给App的数据也会反向转换，转换成物模型定义的格式。

对于第三方蓝牙设备，其产品功能定通常未遵循阿里云IoT物模型规范，需要将设备格式转换脚本后上传。

可以在生活物联网平台控制台的**产品-设备调试**中，单击**编辑脚本**，在产品脚本编辑中可实现脚本转换。

创建蓝牙产品

1. 进入生活物联网控制台，选择某个项目。

2. 创建蓝牙产品。

   - **是否接入网关**选择**是**。
   - **接入协议**选择**BLE**。

移动端开发介绍

开发者在生活物联网平台中，可以通过以下两种方式实现蓝牙设备的移动端开发。

- 使用公版App

  添加设备右上角的 ，进入蓝牙配网界面。

- 开发自有品牌App

  蓝牙SDK下载勾选**蓝牙设备接入框架**套餐项，平台会自动生成SDK套餐包，下载集成开发即可。

  - 套餐包介绍：#unique_75
  - 移动端SDK介绍：Android开发文档，iOS开发文档
  - 蓝牙OTA SDK下载：Android开发文档，iOS开发文档

设备端开发介绍

如果选择了阿里云IoT提供的Breeze蓝牙方案，蓝牙连接开发需要移动端SDK和设备端SDK配合使用。设备端开发需要根据蓝牙芯片选择不同的SDK使用。

1. 进入运营中心。

2. 在固件升级页面，单击**新增固件**，添加一个OTA任务。

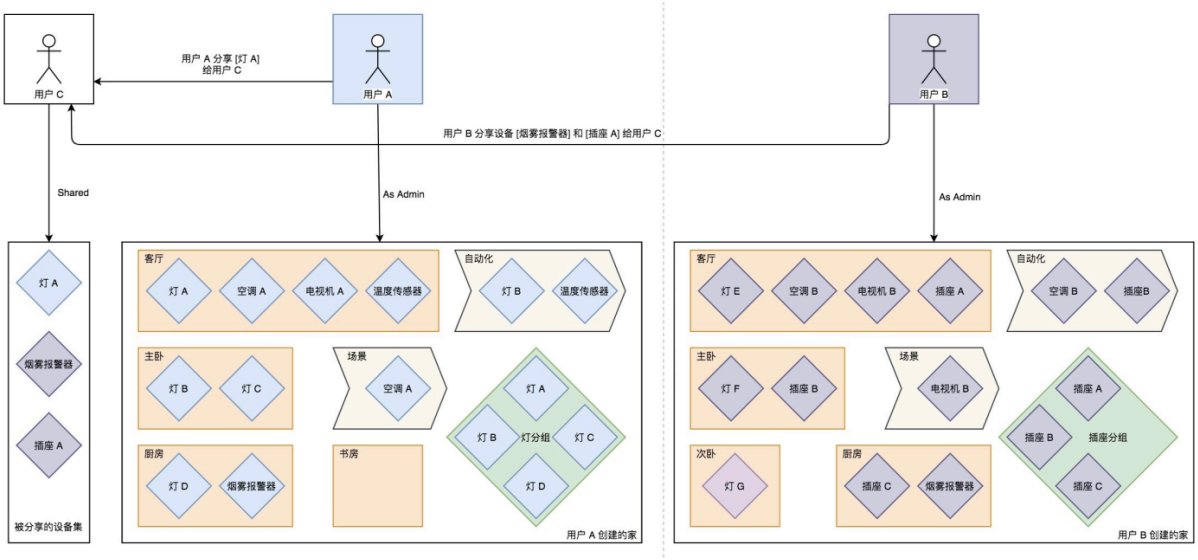
3. 验证固件，固件能过批量升级前必现先验证，在验证页面，可以选择待升级版本（必现由蓝牙设备通过手机上报），DeviceName，验证完成后，即可进行常规批量OTA升级工作。

3.7 家空间开发指南

本节将介绍如何基于平台提供的家空间管理相关API搭建自有App的家空间模型。

概述

家空间管理API服务主要是提供了基于智能生活领域搭建空间模型的能力。“人-空间-设备”的拓扑关系如下图所示。



规则说明

生活物联网平台提供了家空间服务，同时为配合该能力，对原有的配网服务和场景服务2.0进行了升级改造，具体请您参考对应的相关接口文档。

您在开发家空间时，需遵守相关的规则说明如下。

- 用户首次创建家（创建第一个家）时，会进行数据订正，将老版本用户维度下的设备归属到新创建的家下面。
- 设备不能独立于家存在，即绑定设备时必须传入 homeId。
- 设备组是归属在同一款产品下的设备集合，可用于组控。
- 设备可以被归属到房间，也可以独立于房间存在（只归属在家下面）。
- 被分享的设备集属于用户维度的数据，不属于任何一个家，使用/uc/listBindingByAccount接口来获取（接口说明参见用户服务）。
- 被分享的设备集不支持创建场景和自动化。
- 当用户下存在空间数据（创建过家）后，下述版本的 API 将不再提供支持（会统一返回 28806 的错误码）。

服务	path	不支持的版本列表
配网服务	/awss/token/user/bind	1.0.7及以下版本
配网服务	/awss/time/window/user/bind	1.0.7及以下版本
配网服务	/uc/scanBindByShareQrCode	1.0.6及以下版本
配网服务	/awss/enrollee/user/bind	1.0.2及以下版本
配网服务	/awss/ble/user/bind	1.0.2及以下版本

服务	path	不支持的版本列表
配网服务	/awss/gprs/user/bind	1.0.2及以下版本
配网服务	/awss/subdevice/bind	1.0.2及以下版本
场景服务	/living/scene/create	1.0.0
场景服务	/scene/create	1.0.5及以下版本
场景服务	/scene/info/get	1.0.5及以下版本

3.8 客户端调用API

3.8.1 如何调用API

阿里云IoT提供了数量众多的API供客户端使用，为了减少客户的接入成本，将调用云端API过程进行了封装，并向客户提供了客户端SDK（分Android SDK和iOS SDK）。客户端只需要按照以下方式调用API即可。

iOS客户端如何调用云端API

参见iOSAPI通道SDK里的使用说明。

Android客户端如何调用云端API

参见Android API通道SDK里的使用方式。

通用说明

公共参数

生活物联网平台提供的所有客户端API都遵循一致的接口定义规范，其中如下字段为接口调用必须传入或一定会返回的公共字段，此处统一列出，后续接口详细文档说明中将不再单独包含该参数。

请求参数

参数	子键	类型	必有	描述
id	-	String	是	请求id，如使用生活物联网平台提供的SDK，则会自动生成；如果您自行调用API，则需要自己生成。该参数主要用于关联请求应答及问题定位
version	-	String	是	开放平台的版本号，当前固定为1.0
request	-	-	是	系统请求参数
-	apiVer	String	是	所调用接口对应的版本号，具体参见相应的接口文档

参数	子键	类型	必有	描述
-	iotToken	String	是	请求参数，如使用生活物联网平台提供的SDK，则会自动生成；如果您自行调用API，则由您调用token接口获取。主要用于确认请求发起者的系统参数
-	language	String	否	请求多语言的参数，用于传递多语言信息（目前支持：中文 zh-CN、英文 en-US、西班牙语 es-ES、法语 fr-FR、俄语 ru-RU、德语 de-DE、日语 ja-JP、韩语 ko-KR、印地语 hi-IN、意大利语 it-IT）
params	-	-	否	业务请求参数，具体内容因不同的接口而不同，部分接口可能不需要入参，仅通过默认的用户鉴权信息即可操作，具体请参见相应的接口文档

返回参数

参数	类型	必有	描述
id	String	是	用于识别请求记录，即对应的请求id
code	Int	是	接口返回码字段，生活物联网平台客户端API所有接口公用。200表示请求成功；非200表示请求失败（具体失败原因请参见错误码字典）
message	String	是	接口返回信息字段，生活物联网平台客户端API所有接口公用。请求成功返回“success“，否则返回错误原因
data	-	否	接口返回值字段，每个接口的返回值皆可自定义（部分接口没有返回值，仅通过返回码标识请求成功或失败）

网关协议与地址

- 中国站API网关地址：https://api.link.aliyun.com
- 国际站API网关地址：https://api-iot.ap-southeast-1.aliyuncs.com

3.8.2 基础服务

提供基础能力的接口服务，通过接口可以方便的测试调用方与云端的连通性，确认接口调用方式的正确性。

连通性测试接口

定义描述

path	版本	描述	是否需要用户身份的鉴权
/kit/debug/ping	1.0.0	通过该连通性测试接口，测试云端接口调用的连通性	否

请求参数

参数	类型	必填	描述
input	String	是	连通测试数据，input的最大长度不超过1M

返回参数

参数	类型	必有	描述
data	String	是	与入参**input**的数据一致

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0"
  },
  // 以上为生活物联网平台客户端API请求的公共参数
  "params": {
    // 以下为该接口的业务参数
    "input" : "hello"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": "hello"
}
```

3.8.3 账号服务

提供一组与用户账号体系相关的服务接口，通过本组服务接口可以实现用户账户信息的查询和设置。

- 查询统一身份列表
- 修改账号属性

- 注销账号
- 检测账号是否已注册
- 查询统一身份
- 根据登录状态获取当前用户的详细信息
- 获取用户上传头像的url及加密签名
- 绑定外部平台（淘宝）账号
- 获取绑定的外部平台（淘宝）账号
- 接触绑定的外部平台（淘宝）账号

查询统一身份列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/iotx/account/queryIdentityList	1.0.4	查询统一身份列表	否

请求参数

参数	类型	必填	描述
identityIds	List	是	账号身份id列表

返回参数

名称	类型	必有	描述
identityId	String	是	账号身份id
loginId	String	是	OA loginId
loginSource	String	是	账号来源，目前为openAccount
loginName	String	否	登录名
phone	String	是	手机号
email	String	是	邮箱
nickName	String	否	昵称
avatarUrl	String	否	头像链接
gmtCreate	Long	是	创建时间
gmtModified	Long	是	修改时间

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "apiVer": "1.0.4"
     },
     "params": {
       "request": {"identityIds":["506bop1ef3a1cexxxxxxxxf57ae175ac94618"]}
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": [
      {
         "gmtModified": 1508314232000,
         "loginId": "4398047084811",
         "gmtCreate": 1508314232000,
         "identityId": "506bop1ef3a1cexxxxxxxx57ae175ac94618",
         "loginSource": "openAccount",
      }
   ],
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

修改账号属性

定义描述

path	版本	描述	是否需要用户身份的鉴权
/iotx/account/modifyAccount	1.0.5	修改账号属性	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
identityId	String	是	identityId
accountMetaV2	JSON	是	账户属性信息

accountMetaV2结构

参数	类型	必填	描述
loginName	String	否	登录名

参数	类型	必填	描述
phone	String	手机号和邮箱至少一个必填	手机号
email	String	手机号和邮箱至少一个必填	邮箱
nickName	String	否	昵称
appKey	String	是	OA的appKey
avatarUrl	String	否	头像链接

返回参数

无业务返回参数

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "iotToken": "109049c8xxxxxxxx62e29a3ba",
       "apiVer": "1.0.5"
     },
     "params": {
       "request": {"identityId":"50e5opda16exxxxxxxxc9632a038c2479", "accountMetaV2
":{"phone":"157xxxx6621", "appKey":"6xxxx75","nickName":"test"}}
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "class": "com.aliyun.iotx.common.base.service.IoTxResult"
}
```

注销账号

定义描述

path	版本	描述	是否需要用户身份的鉴权
account/unregister	1.0.6	账号注销	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无业务请求参数

返回参数

无业务返回参数

示例

- 请求示例

```
{
    "id":1508232047196,
     "request": {
       "iotToken": "109049cxxxxxxxxb15f6f62e29a3ba",
       "apiVer": "1.0.6"
     },
     "params": {
      },
     "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": null,
  "class": "com.aliyun.iotx.common.base.service.IoTxResult"
}
```

检测账号是否已注册

定义描述

path	版本	描述	是否需要用户身份的鉴权
/user/account/regcheck	1.0.0	检测账号是否已注册	否

请求参数

参数	类型	必填	描述
phone	String	否	手机号
email	String	否	邮箱地址

返回参数

名称	类型	必有	描述
data	Bool	是	是否已经注册，true：已经注册；false：未注册

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "apiVer": "1.0.0"
     },
     "params": {
       "request": {
         "phone": "186xxxx4567",
         "email": "xxxx@xxx.com"
       }
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
  "id": "8d1a665e-ce92-xxxx-xxxx-99d2e81814f3",
  "code": 200,
  "message": null,
  "localizedMsg": null,
  "data": false
}
```

查询统一身份

定义描述

path	版本	描述	是否需要用户身份的鉴权
/user/account/identity/query	1.0.0	查询统一身份	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
loginId	String	否	OA loginId
loginSource	String	否	账号来源类型（OA）
phone	String	否	手机号

参数	类型	必填	描述
email	String	否	邮箱地址
opType	Int	是	1-根据loginId & loginSouce；2-根据手机号码；3-根据邮箱；其他值非法

返回参数

名称	类型	描述
phone	String	手机号
identityId	String	统一身份Id
nickName	String	用户昵称
loginId	String	OA loginId
avatarUrl	String	头像地址
loginName	String	登录名
loginSource	String	账号来源

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "apiVer": "1.0.0"
     },
     "params": {
      "request": {
        "loginId": "value1",
        "loginSource": "value2",
        "phone": "value3",
        "email": "value4",
        "opType": "value5",
      }
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
  "id": "8d1a665e-ce92-xxxx-xxxx-99d2e81814f3",
  "code": 200,
  "message": null,
  "localizedMsg": null,
  "data": {
    "identityId": "5053opfxxxxxxxx473f5db0e73982",
```

```
    "loginId": "493265",
    "loginSource": "openAccount",
    "modifier": "system",
    "nickName": "ktt",
    "phone": "138xxxx3662",
    "loginName": "tester"
  }
}
```

根据登录态获取当前用户的详细信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/user/account/session/authidentity	1.0.1	根据登录态获取当前用户的详细信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotToken	String	是	用户登录态

返回参数

名称	类型	描述
companyId	String	公司Id
companyName	String	公司名
identityId	String	统一身份Id
loginName	String	登录名
nickName	String	昵称
phone	String	手机号
email	String	邮箱地址

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "apiVer": "1.0.1"
     },
     "params": {
       "request": {
```

```
        "iotToken": "xxxxxxxxxxxxx",
      }
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
 "id": "8d1a665e-ce92-xxxx-xxxx-99d2e81814f3",
 "code": 200,
 "message": null,
 "localizedMsg": null,
 "data": {
  "companyId": "95BDDAxxxxxxxxt8240A19D5",
  "companyName": "test物联技术有限公司",
  "identityId": "5022al4d34xxxxxxxxaf3fac06085",
  "loginName": "test物联",
  "nickName": "test物联",
  "phone": "xxxxxxxxxxx",
  "email": "abc@cc.com"
 }
}
```

获取用户上传头像的url及加密签名

当用户需要修改头像时流程如下。

1. 调用此接口获取上传的url和签名等参数。
2. App取得上传的url和签名等参数后，通过POST方法将头像上传（请参见#unique_86）。

   上传时header中的**x-oss-forbid-overwrite**参数不能设置为true，否则可能导致上传失败。上传成功后，根据流程1中调用接口的返回参数中的**host**、**dir**拼接得出头像图片url地址（拼接规则为“https://” + host + “/” + dir）。
3. 通过修改账号属性接口更新图像url。

path	版本	描述	是否需要用户身份的鉴权
/living/user/avatar/upload/signature/get	1.0.0	用户头像上传url以及加密签名获取	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
fileSize	int	是	头像图片文件大小，单位为字节，取值范围最小为1；最大为10485760（10M）。

返回参数

名称	类型	描述
accessKey	String	OSS的accessKey
dir	String	图片在OSS桶里的存储目录
expire	Long	过期时间戳，毫秒为单位，在此时间戳之前可以向OSS上传头像
host	String	OSS访问域名地址
signature	String	签名
policy	String	请求表单域的合法性校验信息

示例

- 请求示例

```
{
    "id":1508232047194,
     "request": {
       "apiVer": "1.0.0"
     },
     "params": {
       "request": {
         "fileSize": 300888
       }
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "id":1508232047194,
   "code":200,
   "data": {
      "signature":"b45CXXXXXXXXXXXXXXXXXXXXX",
      "accessKey":"LTAXXXXXXXXXXXXX",
      "expire":1578077777777,
      "host":"xxxx.oss-ap-shanghai.aliyuncs.com",
      "dir":"images/third_app_user/profile/50adopbXXXXXX_MmI2NGJjZD.png",
      "policy":"eyJleHBpcmFXXXXXXXXXXX0aW9uIjoiMjAy"
     }
```

```
}
```

绑定外部平台（淘宝）账号

path	版本	描述	是否需要用户身份的鉴权
/account/taobao/bind	1.0.5	将App账号与淘宝账号进行绑定，用于同时支持天猫精灵控制用户设备，详情请参见#unique_87	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
authCode	String	是	淘宝OAuth认证返回autoCode

返回参数

名称	类型	描述
accountId	String	账号id，此处为淘宝账号的id
accountType	String	账号类型，此处固定为**TAOBAO**

示例

- 请求示例

```
{
    "id":1508232047194,
    "request": {
      "apiVer": "1.0.0"
    },
    "params": {
      "request": {
        "authCode": "value1"
      }
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": {
      "accountId":"value2",
      "accountType":"TAOBAO"
```

```
    }
}
```

获取绑定的外部平台（淘宝）账号

path	版本	描述	是否需要用户身份的鉴权
/account/thirdparty/get	1.0.5	查询用户当前已绑定的外部平台账号ID，此处为淘宝账号ID，用于同时支持天猫精灵控制用户设备，详情请参见#unique_87	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
accountType	String	是	外部平台类型，此处固定为**TAOBAO**

返回参数

名称	类型	描述
accountId	String	账号id，此处为是淘宝账号的id
accountType	String	外部平台类型，此处固定为**TAOBAO**

示例

- 请求示例

```
{
    "id":1508232047194,
    "request": {
      "apiVer": "1.0.0"
    },
    "params": {
      "request": {
        "authCode": "TAOBAO"
      }
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
```

```
      "accountId":"value1",
      "accountType":"TAOBAO"
    }
}
```

解除绑定的外部平台（淘宝）账号

path	版本	描述	是否需要用户身份的鉴权
/account/thirdparty/unbind	1.0.5	解绑用户当前已绑定的外部平台账号ID，此处为淘宝账号ID，用于同时支持天猫精灵控制用户设备，详情请参见#unique_87	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
accountType	String	是	外部平台类型，此处固定为**TAOBAO**

返回参数

名称	类型	描述
accountId	String	账号id，此处为淘宝账号的id
accountType	String	外部账号类型，固定为**TAOBAO**

示例

- 请求示例

```
{
    "id":1508232047194,
    "request": {
      "apiVer": "1.0.0"
    },
    "params": {
      "request": {
        "authCode": "TAOBAO"
      }
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
```

```
  "code": 200,
  "data": {
    "accountId":"value1",
    "accountType":"TAOBAO"
  }
}
```

3.8.4 长连接服务

提供基于用户维度的长连接通道服务，包括连接通道的绑定、解绑和消息接收。通过长连接服务客户端可以轻松的实现监听设备的实时变更消息。

用户中心长连接

- 绑定MQTT消息通道
- 解绑MQTT消息通道

设备消息下行到App

- 设备属性变更推送
- 设备事件变更推送
- 设备连接状态变更推送
- 子设备添加结果推送
- 固件升级进度推送

绑定MQTT消息通道

上行，绑定用户和clientId对应的MQTT通道。

```
TOPIC: /sys/${productKey}/${deviceName}/app/up/account/bind
REPLY TOPIC:  /sys/${productKey}/${deviceName}/app/down/account/bind_reply

request:
{
 "id": "msg_id_bind_xxx",   // 自定义 msg_id
 "version": "1.0",
 "request": {
  "clientId": "client_id_xxx"   // 自定义 client_id
 },
 "params": {
  "iotToken": "iot_token_xxx"   // 表示身份 id 的 iotToken
 }
}

response:
{
  "id":"msg_id_bind_xxx",   // 和 request 中的 id 保持一致
  "code":200,
  "message":"success"
}
```

说明:

通常情况下，开发者不需要主动调用这个接口，长连接SDK中已经封装了本接口的实现。

解绑MQTT消息通道

上行，解绑用户和clientId对应的MQTT通道。

```
TOPIC: /sys/${productKey}/${deviceName}/app/up/account/unbind
REPLY TOPIC:  /sys/${productKey}/${deviceName}/app/down/account/unbind_reply

request:
{
  "id": "msg_id_unbind",    // 自定义 msg_id
  "version": "1.0",
  "request": {
    "clientId": "client_id_xxx"    // 自定义 client_id
  },
  "params": {
  }
}

response:
{
    "id":"msg_id_unbind",    // 和 request 中的 id 保持一致
    "code":200,
    "message":"success"
}
```

说明:

通常情况下，开发者不需要主动调用这个接口，长连接SDK中已经封装了本接口的实现。

设备属性变更推送

```
TOPIC: /sys/${productKey}/${deviceName}/app/down/thing/properties

MESSAGE:
{
  "id": "157355878660810192",
  "version": "1.0",
  "params": {
        "iotId":"0300MSKL03667c544f69342a74Sv4Za4",
        "productKey":"X5eCzh6fEH7",
        "deviceName":"5gJtxDVeGAkaEztpisjX",
        "items":{
          "temperature":{
            "time":1510292697471,
            "value":30
          },
          "meterReadingMonth":{
            "time":1510292697470,
            "value":715665571
          },
          "alarm":{
            "time":1510292697470,
            "value":"normal"
          }
        }
    },
  "method":"thing.properties"
}
```

```
// 其中：items字段表示变更的属性列表（key-value健对值）。
// 以其中一个key-value举例说明：
// temperature表示属性标识符，对应属性的identifier。
// time表示属性上报value对应的时间戳。
// value表示设备属性值（value数据规范可参考TSL描述）。
```

设备事件变更推送

```
TOPIC: /sys/${productKey}/${deviceName}/app/down/thing/events

MESSAGE:
{
  "id": "157355878660810192",
  "version": "1.0",
  "params":{
     "identifier":"BrokenInfo"
     "name":"损坏率上报",
     "type":"info",
     "iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
     "productKey":"X5eCzh6fEH7",
     "deviceName":"5gJtxDVeGAkaEztpisjX",
     "value":{
       "Power":"on"
       "structParam":{
            "param1":"abc",
            "param2":123
         }
     },
      "time":1510799670074
   }
   "method":"thing.events"
}

// 其中：identifier表示事件标识符，对应事件的 identifier。
// value表示变更的事件属性列表（key-value 健值对）。
// time表示事件上报 value 对应的时间戳。
// type表示事件类型，可取值包括 info，error，block。
```

设备连接状态变更推送

```
TOPIC: /sys/${productKey}/${deviceName}/app/down/thing/status

MESSAGE:
{
  "id": "157355878660810193",
  "version": "1.0",
  "params":{
      "iotId":"0300MSKL03667c544f69342a74Sv4Za4",
      "productKey":"X5eCzh6fEH7",
      "deviceName":"5gJtxDVeGAkaEztpisjX",
      "status":{
         "time":1510292697471,
         "value":0
       }
  }
  "method":"thing.status"
}

//其中，status字段：
//value 表示设备生命周期，可取值包括，0：未激活，1：上线，3：离线，8：禁用。
```

```
//time 表示当前状态的开始时间。
```

子设备添加结果推送

```
TOPIC: /sys/${productKey}/${deviceName}/app/down/thing/topo/add/status

MESSAGE:
{
   "id":"157355878660810192",
   "version":"1.0",
   "param":{
      "status":0,
      "subIotId":"xxx",
      "subProductKey":"xxx",
      "subDeviceName":"xxx",
      "oldGwIotId":"xxx",
      "oldGwProductKey":"xxx",
      "oldGwDeviceName":"xxx",
      "newGwIotId":"xxx",
      "newGwProductKey":"xxx",
      "newGwDeviceName":"xxx"
   },
   "method":"thing.topo.add.status"
}

//其中，status表示子设备的注册结果，可取值包括，0：添加成功，1：子设备注册成功，但设备已被其它网关绑定，2：子设备注册失败。
```

固件升级进度推送

```
TOPIC: /sys/${productKey}/${deviceName}/app/down/ota/device/forward

MESSAGE:
{
   "id": "157355878660810192",
   "version": "1.0",
   "params": {
      "iotId": "",
      "srcVersion": "",
      "destVersion": "",
      "step": 50,
      "desc": "",
      "upgradeStatus": 1
   },
   "method":"ota.device.forward"
}

//其中：srcVersion表示最近一次升级的源版本。
// destVersion表示最近一次升级目标版本。
// step表示固件升级进度信息，可取值包括，-1：表示升级失败，-2：表示下载失败，-3：表示校验失败，-4：表示烧写失败，[1，100] 之间的数字：表示升级进度百分比。
// desc当前步骤的描述信息，如果发生异常，此字段可承载错误信息。
```

```
// upgradeStatus表示升级结果，可取值包括，0：待升级或待确认，1：升级中，2：升级异常，3：升级失败，4：升级完成。
```

3.8.5 产品管理服务

提供一组与产品相关的服务接口，客户端应用通过本组服务接口可以实现产品信息的查询，包括可配网产品列表、配网引导页信息以及具体产品信息的查询。

- 根据设备id查询产品详情
- 根据产品key查询产品详情
- 根据品类id查询产品列表
- 查询配网引导信息
- 获取支持配网的产品列表

根据设备id查询产品详情

定义描述

path	版本	描述	是否需要登录
/thing/detailInfo/queryProductInfo	1.1.3	查询产品详情信息	否

请求参数

参数	类型	必填	描述
iotId	String	是	设备id

返回参数

参数	类型	必有	描述
productKey	String	是	产品的Key
dataFormat	String	是	数据格式 • CUSTOM_FORMAT：透传/自定义格式 • ALINK_FORMAT：Alink协议

参数	类型	必有	描述
netType	String	是	设备入网类型： • lora：NET_LORA • gprs：NET_CELLULAR • wifi：NET_WIFI • zigbee：NET_ZIGBEE • 蓝牙：NET_BT • 以太网：NET_ETHERNET • 其他：NET_OTHER
productSecret	String	否	产品密钥
nodeType	String	是	设备的节点类型：DEVICE，GATEWAY
domain	String	否	产品域
name	String	是	产品名称
region	String	是	地域
ownerDomain	String	否	所有者领域
categoryId	Long	是	归属品类id
categoryKey	String	否	品类标识符
categoryName	String	否	品类名称
accessMethod	String	是	访问方式 • DATA_DIRECT：数据直连 • AIR_FORWARD：空中转发 • APP：APP • VIRTUAL：虚拟设备
status	String	是	产品状态 • DEVELOPMENT_STATUS：开发中 • RELEASE_STATUS：已发布
gmtCreate	Date	是	创建时间

参数	类型	必有	描述
gmtModified	Date	是	修改时间
creator	String	是	创建者
modifier	String	是	修改者
productId	Long	是	产品Id
aliyunCommodityCode	String	否	商品码
connectMode	String	是	配网方式
rbacTenantId	String	是	租户id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.1.3",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D2429xxxxxxxxF31A2697"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "accessMethod": "DATA_DIRECT",
    "gmtModified": 1517217645000,
    "modifier": "5030ala9cd20xxxxxxxxc63b454243668090",
    "productKey": "a1IxxxxW",
    "creator": "5030ala9cdxxxxxxxxc63b454243668090",
    "productId": 87634,
    "dataFormat": "ALINK_FORMAT",
    "netType": "NET_WIFI",
    "aliyunCommodityCode": "tmp",
    "productSecret": "tfbMxxxxKKj",
    "nodeType": "DEVICE",
    "gmtCreate": 1517208303000,
    "domain": "a1IxxxxCW",
    "name": "风扇",
    "tenantId": 3312,
    "connectMode": "2,3",
    "region": "cn-shanghai",
    "ownerDomain": "tmp_5030alxxxxxxxx0fbdc63b454243668090",
    "rbacTenantId": "755ED7C7xxxxxxxx026BBC48BF0247",
    "categoryId": 285,
```

```
    "status": "RELEASE_STATUS"
  },
  "id": "6aad0b12-2192-xxxx-xxxx-08a2bc0b5c2a"
}
```

根据产品key查询产品详情

定义描述

path	版本	描述	是否需要登录
/thing/allProductInfo/getByProductKey	1.1.3	根据ProductKey查询产品信息	否

请求参数

参数	类型	必填	描述
productkey	String	是	产品Key

返回参数

参数	类型	必有	描述
productKey	String	是	产品的Key
dataFormat	String	是	数据格式 • CUSTOM_FORMAT：透传/自定义格式 • ALINK_FORMAT：Alink协议
netType	String	是	设备入网类型： • lora：NET_LORA • gprs：NET_CELLULAR • wifi：NET_WIFI • zigbee：NET_ZIGBEE • 蓝牙：NET_BT • 以太网：NET_ETHERNET • 其他：NET_OTHER
productSecret	String	否	产品密钥
nodeType	String	是	节点类型，DEVICE：设备；GATEWAY：网关

参数	类型	必有	描述
domain	String	否	产品域
name	String	是	产品名称
region	String	是	地域
ownerDomain	String	否	所有者领域
categoryId	Long	是	归属品类id
categoryKey	String	否	品类标识符
categoryName	String	否	品类名称
accessMethod	String	是	访问方式 • DATA_DIRECT：数据直连 • AIR_FORWARD：空中转发 • APP：APP • VIRTUAL：虚拟设备
status	String	是	产品状态 • DEVELOPMENT_STATUS：开发中 • RELEASE_STATUS：已发布
gmtCreate	Date	是	创建时间
gmtModified	Date	是	修改时间
creator	String	是	创建者
modifier	String	是	修改者
productId	Long	是	产品Id
aliyunCommodityCode	String	否	商品码
connectMode	String	是	配网方式
rbacTenantId	String	是	rbac租户id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
```

```
  "request": {
    "apiVer": "1.1.3",
    "iotToken": "token"
  },
  "params": {
    "productKey": "a1xxxxVAH"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "accessMethod": "DATA_DIRECT",
    "gmtModified": 1517217645000,
    "modifier": "5030alxxxxxxxxfbdc63b454243668090",
    "productKey": "a1lxxxxCW",
    "creator": "5030ala9cdxxxxxxxxc63b454243668090",
    "productId": 87634,
    "dataFormat": "ALINK_FORMAT",
    "netType": "NET_WIFI",
    "aliyunCommodityCode": "tmp",
    "productSecret": "tfbxxxxxxxxqPKKj",
    "nodeType": "DEVICE",
    "gmtCreate": 1517208303000,
    "domain": "a1I0TQKlMCW",
    "name": "风扇",
    "tenantId": 3312,
    "connectMode": "2,3",
    "region": "cn-shanghai",
    "ownerDomain": "tmp_5030ala9cdxxxxxxxxb454243668090",
    "rbacTenantId": "755ED7Cxxxxxxxx26BBC48BF0247",
    "categoryId": 285,
    "status": "RELEASE_STATUS"
  },
  "id": "6aad0b12-2192-xxxx-xxxx-08a2bc0b5c2a"
}
```

根据品类id查询产品列表

定义描述

path	版本	描述	是否需要登录
/thing/productInfo/getByCategory	1.1.4	根据品类查询租户创建的所有产品，无需租户id	否

请求参数

参数	类型	必填	描述
categoryId	Long	是	品类Id
rbacTenantId	String	否	rbac租户Id

返回参数

参数	类型	必有	描述
productKey	String	是	产品的Key
dataFormat	String	是	数据格式：0-透传/自定义格式；1-Alink协议
netType	String	是	入网类型：0-LORA；3-WIFI；4-:ZIGBEE；5-BT；6-CELLULAR；7-ETHERNET；8-NET_OTHER
productSecret	String	否	产品密钥
nodeType	String	是	节点类型：0-DEVICE；1-GATEWAY
domain	String	否	产品域
name	String	是	产品名称
region	String	是	地域
ownerDomain	String	否	所有者领域
categoryId	Long	是	归属品类id
categoryKey	String	否	品类标识符
categoryName	String	否	品类名称
accessMethod	String	是	访问方式：0-数据直连；1-空中转发；2-APP
status	Integer	是	产品状态：0-开发中；1-已发布
gmtCreate	Date	是	创建时间
gmtModified	Date	是	修改时间
creator	String	是	创建者
modifier	String	是	修改者
productId	Long	是	产品Id
aliyunCommodityCode	String	否	商品码
connectMode	String	是	配网方式
rbacTenantId	String	是	租户id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.1.3",
    "iotToken": "token"
  },
  "params": {
    "categoryId": 12
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [{
    "accessMethod": "DATA_DIRECT",
    "gmtModified": 1517562992000,
    "productModel": "niki_test_prodcut_model",
    "modifier": "50a1al58xxxxxxxx201b3f85e2ff9014",
    "productKey": "a1xxxxCf",
    "image": "logo",
    "creator": "50a1alxxxx201b3f85e2ff9014",
    "productId": 95959,
    "dataFormat": "CUSTOM_FORMAT",
    "netType": "NET_CELLULAR",
    "aliyunCommodityCode": "tmp",
    "productSecret": "Y1kxxxxxxxxkCw",
    "nodeType": "DEVICE",
    "gmtCreate": 1517562992000,
    "domain": "a15iEO9rxCf",
    "name": "test_name",
    "tenantId": 21,
    "region": "cn-hangzhou",
    "rbacTenantId": "2BF8238E3xxxxxxxx9C7180BD2CF6",
    "ownerDomain": "tmp_50a1al5xxxxxxxxf3350201b3f85e2ff9014",
    "categoryId": 61,
    "status": "DEVELOPMENT_STATUS"
  }],
  "id": "db13e9ae-c617-xxxx-xxxx-a1e4f43e36ec"
}
```

查询配网引导信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/enrollee/guide/get	1.1.3	查询配网引导页的模版定义信息，用于客户端展示友好的配网流程的引导信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
productKey	String	是	productKey

返回参数

参数	类型	必有	描述
productKey	String	是	产品Key
helpTitle	String	否	帮助标题
dnGuideIcon	String	否	配网引导图标
helpCopywriting	String	否	帮助文案
buttonCopywriting	String	否	按钮文案
dnCopywriting	String	否	配网文案
helpIcon	String	否	帮助图标
id	Long	是	配网引导id
gmtCreate	Date	是	创建时间
gmtModified	Date	是	修改时间
creator	String	是	创建者
modifier	String	是	修改者
rbacTenantId	String	是	rbac租户Id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.1.3",
    "iotToken": "token"
  },
  "params": {
    "productKey": "a1xxxxAH"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [{
    "gmtModified": 1517476367000,
```

```
      "creator": "755ED7CxxxxxxxxC48BF0247",
      "helpTitle": "指示灯没有闪烁？",
      "dnGuideIcon": "http://iotx-ilop.oss-cn-hangzhou.aliyuncs.com/iotx-ilop-portal/
p/a12xxxxQfi/networkConfig/1517476344732/timg.jpg",
      "helpCopywriting": "请长按重置3-5秒，确保看到指示灯闪烁后松开",
      "modifier": "755ED7CxxxxxxxxBC48BF0247",
      "buttonCopywriting": "我确认在闪烁",
      "gmtCreate": 1517476347000,
      "productKey": "a1xxxxW",
      "id": 87,
      "dnCopywriting": "接通电源，确认指示灯闪烁",
      "helpIcon": "http://iotx-ilop.oss-cn-hangzhou.aliyuncs.com/iotx-ilop-portal/p/
a12xxxxfi/networkConfig/1517476333895/timg (1).jpg",
      "rbacTenantId": "755ED7C7810B4255BA026BBC48BF0247"
   }],
   "id": "e6d8bd5e-b79b-xxxx-xxxx-b3f9c65cdd2b"
}
```


获取支持配网的产品列表

定义描述

path	版本	描述	是否需要登录
/thing/productInfo/getByAppKey	1.1.5	获取当前支持配网操作的产品列表	是

请求参数

无

返回参数

参数	类型	必有	描述
tenantId	String	是	租户id
productKey	String	是	产品的Key
dataFormat	Integer	是	设备上传的数据格式，0表示透传或自定义格式；1表示使用Alink协议数据格式
netType	Integer	是	产品入网类型：0表示LoRa；3表示WiFi；4表示ZigBee；5表示Bluetooth；6表示蜂窝网；7表示以太网；8表示其他入网方式
nodeType	Integer	是	节点类型，0表示设备；1表示网关
name	String	是	产品名称

参数	类型	必有	描述
region	String	是	产品所在地域
categoryKey	String	是	产品所属品类的品类标识符
categoryName	String	是	产品所属品类的名称
accessMethod	Integer	是	访问方式：0表示数据直连；1表示空中转发；2表示APP
status	Integer	是	产品状态，0表示开发中；1表示已发布
gmtCreate	Date	是	产品创建时间
gmtModified	Date	是	上一次产品信息变更的时间
description	String	否	产品描述

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.1.5",
    "iotToken": "token"
  },
  "params": {
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200,
  "data":[
    {
      "accessMethod": 0,
      "gmtModified": 1515829586000,
      "productKey": "a1oOl0IW16u",
      "dataFormat": 1,
      "netType": 3,
      "nodeType": 0,
      "gmtCreate": 1515829586000,
      "name": "灯",
      "tenantId": "D99AEBxxxxxxxxCF008B2DCE1E313",
      "region": "cn-shanghai",
      "categoryKey":"Light",
      "categoryName":"灯"
      "status": 0
```

```
    }
  ]
}
```

3.8.6 物的模型服务

提供一组与物（设备）相关的服务接口，客户端应用通过本组服务接口可以实现物信息的查询和控制，查询的内容涉及物的基本信息，物的上报数据信息以及物的TSL定义。

设备状态查询和控制

- 获取物的基本信息
- 获取物的属性timeline数据
- 获取物的事件timeline数据
- 设置物的属性
- 获取物的事件
- 根据设备id获取物的模板
- 根据产品key获取物的模板
- 获取物的属性
- 调用物的服务
- 获取物的连接状态
- 执行恢复出厂设置任务
- 查询恢复出厂设置任务的执行状态

设备扩展属性

- 设置设备扩展信息
- 获取设备扩展信息
- 删除设备扩展信息

设备定时

- 创建预约定时场景
- 更新预约定时场景
- 删除预约定时场景
- 获取预约定时场景详情
- 查询预约定时场景列表

App小组件控制设备

- 查询用户的小组件设备列表
- 更新用户的小组件设备列表

- 查询用户小组件设备的属性列表

获取物的基本信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/info/get	1.0.2	获取物的信息，例如设备名称、MAC地址、设备激活时间等内容	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符

返回参数

参数	类型	必有	描述
gmtModified	Long	是	设备记录数据修改时间（系统内部使用字段，可无需关心）
activeTime	Long	否	设备首次激活时间
gmtCreate	Long	是	设备记录创建时间，对于生活物联网平台而言，是量产时生成设备证书的时间
productKey	String	是	产品key
statusLast	Integer	是	上一次设备状态
mac	String	否	mac地址
deviceSecret	String	是	设备secret
iotId	String	是	设备唯一标识符
name	String	是	设备名称
sdkVersion	String	否	sdk版本
sn	String	否	设备sn

参数	类型	必有	描述
thingType	String	是	设备类型 • VIRTUAL：虚拟设备 • VIRTUAL_SHADOW：虚拟设备 • DEVICE：真实设备
firmwareVersion	String	否	固件版本号
rbacTenantId	String	是	租户id
status	Integer	是	设备状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
netAddress	String	否	ip地址

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D2429xxxxE4F31A2697"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "gmtModified": 1517918765000,
    "activeTime": null,
    "gmtCreate": 1500017648000,
    "productKey": "kQxxxxqA",
    "statusLast": null,
    "mac": null,
    "netAddress": null,
    "deviceSecret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "iotId": "D95D242xxxxxxxx4F31A2697",
    "name": "Kq7rgxxxxxwJh",
    "nickname": null,
    "sdkVersion": null,
    "sn": null,
    "thingType": "DEVICE",
```

```
      "firmwareVersion": "1.0.0",
      "rbacTenantId": "ADSBCxxxx918327917",
      "status": 1
    },
    "message": "success",
    "localizedMsg": null
}
```

获取物的属性timeline数据

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/device/property/timeline/get	1.0.0	获取物的属性timeline数据（最多可以获取设备一年内的数据）。 **说明：** 调用/living/device/reset后，再调用该接口时不再获取到设备的历史数据。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
identifier	String	是	属性标识符
start	Long	是	timeline起始时间
end	Long	是	timeline终止时间
limit	Int	是	最大查询条数，上限200
order	String	是	• asc：时间顺序 • desc：时间逆序

返回参数

参数	类型	必有	描述
iotid	String	是	设备标识符
data	String	是	设备timeline数据

参数	类型	必有	描述
modifytime	Long	是	属性更新时间
property	String	是	属性标识符
batchId	String	是	批次id，设备端同一批上报的属性具有相同的批次Id
timestamp	Long	是	属性采集时间戳

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxxCCE4F31A2697",
    "identifier": "xxxx",
    "start": 1517217645000,
    "end": 1517217645000,
    "limit": 10,
    "order": "asc"
  }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": {
  "items": [
   {
    "iotid": "D95D24294xxxxxxxx4F31A2697",
    "data": "1.23",
    "modifytime": 1511812747287,
    "property": "Lixxxxt",
    "batchId": "2fc766cxxxxxxxxed1f3e4b61803",
    "timestamp": 1511812747245
   },
   {
    "iotid": "xqxyZjSKzxxxxxxxx010851c00",
    "data": 1.24,
    "modifytime": 1511812747288,
    "property": "Lxxxxlt",
    "batchId": "2fc766cxxxx1f3e4b61803",
    "timestamp": 1511812747245
   }
  ],
  "timestamp": 1511812747245
 }
```

```
}
```

获取物的事件timeline数据

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/device/event/timeline/get	1.0.0	获取物的事件timeline数据，最多可以获取设备一年内的数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
identifier	String	是	属性标识符
eventType	String	是	事件类型
start	Long	是	timeline起始时间
end	Long	是	timeline终止时间
limit	Int	是	最大查询条数，上限200
order	Boolean	是	• asc：时间顺序 • desc：时间逆序

返回参数

参数	类型	必有	描述
eventCode	String	是	事件标识符
iotId	String	是	物的唯一标识符
eventName	String	是	事件名称
eventType	String	是	事件类型
eventBody	String	是	事件参数
batchId	String	是	批次id，设备同一批上报的事件具有相同的批次id
timestamp	Long	是	时间采集时间

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxxE4F31A2697",
    "identifier": "xxxx",
    "eventType": "xxxx",
    "start": 1517217645000,
    "end": 1517217645000,
    "limit": 10,
    "order": "asc"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "localizedMsg": "",
  "data": {
    "items": [
      {
        "eventCode": "Error",
        "iotId": "D95D24xxxxECCE4F31A2697",
        "eventName": "故障上报",
        "eventType": "info",
        "eventBody": "{\"ErrorCode\": 0}",
        "batchId": "5ebc6axxxx3edde6d28c8fb3",
        "timestamp": 1516342985261
      },
      {
        "eventCode": "Error",
        "iotId": "YzqEnI5xxxxo0010840500",
        "eventName": "故障上报",
        "eventType": "info",
        "eventBody": "{\"ErrorCode\": 0}",
        "batchId": "4a0b5a7xxxxxxxx8d5ff77456f1",
        "timestamp": 1516342995305
      }
    ],
    "timestamp": 1516343075699
  }
}
```

设置物的属性

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/properties/set	1.0.2	设置物的属性	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
items	JSON	是	set参数

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D242xxxxE4F31A2697",
    "items": {
      "LightSwitch1": 0,
      "LightSwitch2": 0
    }
  }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": null,
 "message": "success",
 "localizedMsg": ""
}
```

获取物的事件

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/events/get	1.0.2	获取物最新上报的事件，如果该事件未上报，则不会返回	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符

返回参数

参数	类型	必有	描述
identifier	String	是	事件标识符
time	Long	是	时间采集时间
type	String	是	事件类型
value	Object	是	事件出参

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxxCE4F31A2697"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [
    {
      "identifier": "Error",
      "time": 1517420842314,
      "type": "info",
      "value": {
        "Shrink5": 12,
        "Shrink20": {
          "testKey2": "val2",
          "testKey": "val"
```

```
            }
          }
        }
    ],
    "message": "success",
    "localizedMsg": null
}
```

根据设备id获取物的模板

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/tsl/get	1.0.2	获取物的tsl功能模板（包含属性、事件、服务的定义）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符

返回参数

参数	类型	必有	描述
data	JSON String	否	物的模板json对象

示例

- 请求示例

```
{
    "id": "1509086454180",
    "version": "1.0",
    "request": {
        "apiVer": "1.0.2",
        "iotToken": "token"
    },
    "params": {
        "iotId": "D95D2xxxxCE4F31A2697"
    }
}
```

- 正常返回示例

```
{
 "code": 200,
    "data": {
        "schema": "http://aliyun/iot/thing/desc/schema",
        "profile": {
            "productKey": "xxxxxxx",
```

```
        "deviceName": "xxxxxxxxxxxxxxxx"
      },
      "link": "/sys/a1xxxxpck/YzqEnxxxx2pjjo/thing/",
      "services": [
        {
          "outputData": [

          ],
          "identifier": "ToggleLightSwitch",
          "inputData": [

          ],
          "method": "thing.service.ToggleLightSwitch",
          "name": "翻转主灯开关",
          "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
        }
      ],
      "events": [
        {
          "outputData": [
            {
              "identifier": "ErrorCode",
              "dataType": {
                "specs": {
                  "0": "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
        }
      ]
    },
    "message": "success",
    "localizedMsg": null
}
```


根据产品key获取物的模板

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/tsl/getByProducyKey	1.0.2	获取物的tsl功能模板（包含属性、事件、服务的定义）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
productKey	String	是	产品Key
deviceName	String	否	deviceName

返回参数

参数	类型	必有	描述
data	JSON String	否	物的模板json对象

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "ProductKey": "xxxxxxxx",
    "deviceName": "xxxxxxxx"
  }
}
```

- 正常返回示例

```
{
 "code": 200,
  "data": {
    "schema": "http://aliyun/iot/thing/desc/schema",
    "profile": {
      "productKey": "xxxxxxx",
      "deviceName": "xxxxxxxxxxxxxxxx"
    },
    "link": "/sys/a1xxxck/YzqEnxxxxpjjo/thing/",
    "services": [
      {
        "outputData": [

        ],
        "identifier": "ToggleLightSwitch",
```

```
            "inputData": [

            ],
            "method": "thing.service.ToggleLightSwitch",
            "name": "翻转主灯开关",
            "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
          }
        ],
        "events": [
          {
            "outputData": [
              {
                "identifier": "ErrorCode",
                "dataType": {
                  "specs": {
                    "0": "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
          }
        ]
      },
      "message": "success",
      "localizedMsg": null
}
```


获取物的属性

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/properties/get	1.0.2	获取物的所有属性快照数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的标识符

返回参数

参数	子参数	类型	必有	描述
identifier				属性id
	value	Object	是	属性对应的数据
	time	Long	是	属性采集时间

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.2",
      "iotToken": "token"
   },
   "params": {
      "iotId": "D95D24xxxxCE4F31A2697"
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
   "_sys_device_mid": {
     "time": 1516356290173,
     "value": "example.demo.module-id"
   },
   "WorkMode": {
     "time": 1516347450295,
     "value": 0
   },
   "_sys_device_pid": {
     "time": 1516356290173,
     "value": "example.demo.partner-id"
   }
 },
 "message": "success",
 "localizedMsg": null
}
```

调用物的服务

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/service/invoke	1.0.2	调用物的服务	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的标识符
identifier	String	是	服务标识符
args	JSON String	是	服务入参

返回参数

参数	类型	必有	描述
data	String	是	该返回值类型及值取决于设备端实现逻辑

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.2",
      "iotToken": "token"
   },
   "params": {
      "iotId": "D95D2xxxxCE4F31A2697",
      "identifier": "xxxx",
      "args": {}
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": xxx，//xxx为服务的实际返回值
 "message": "success",
 "localizedMsg": ""
}
```

获取物的连接状态

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/status/get	1.0.2	获取物的连接状态	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符

返回参数

参数	类型	必有	描述
status	Integer	是	设备状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
time	Long	是	状态变更时间

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.2",
      "iotToken": "token"
   },
   "params": {
      "iotId": "D95D2xxxxCCE4F31A2697"
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
  "time": 1517923297000,
  "status": 3
 },
 "message": "success",
 "localizedMsg": null,
```

```
}
```

执行恢复出厂设置任务

定义描述

path	版本	描述	是否需要登录
/living/device/reset	1.0.0	恢复设备的出厂设置，并通知设备端，同时会清除云端设备快照数据；该接口调用会返回一个恢复出厂设置任务的id，任务后台异步执行，通过该任务id可查询任务的执行状态（配合“查询恢复出厂设置任务的执行状态”接口使用） **说明:** 调用该接口时，可删除设备历史数据。	是

请求参数

参数	类型	必填	描述
iotId	String	是	设备的唯一标识

返回参数

参数	类型	描述
taskId	String	执行恢复出厂设置任务的id

示例

- 请求示例

```
{
  "id":"xxx",  // 请求的 requestId
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"  // 接口的版本号
  },
  "params":{
    "iotId":"xxx"
```

```
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"taskId"
}
```

查询恢复出厂设置任务的执行状态

定义描述

path	版本	描述	是否需要登录
/living/device/reset/task/state/get	1.0.0	查询恢复出厂设置任务的执行状态	是

请求参数

参数	类型	必填	描述
taskId	String	是	执行恢复出厂设置任务的id

返回参数

参数	类型	描述
data	String	任务的执行状态，可取值包括：SUCCESS、FAILED、TIMEOUT、RUNNING

示例

- 请求示例

```
{
  "id":"xxx",  // 请求的 requestId
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"  // 接口的版本号
  },
  "params":{
    "taskId":"xxx"
  }
```

```
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data": "SUCCESS"
}
```

设置设备扩展信息

单个设备的标签不能超过10个。

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/extended/property/set	1.0.2	设置物的扩展属性数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
dataKey	String	是	扩展属性标识符（小于64个字符）
dataValue	String	是	扩展属性的值（小于128个字符）

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D242xxxxxxxx4F31A2697",
    "dataKey": "xxxx",
```

```
    "dataValue": "xxxx"
  }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": null
}
```

获取设备扩展信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/extended/property/get	1.0.2	获取物的扩展属性的数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
dataKey	String	是	扩展属性标识符

返回参数

参数	类型	必有	描述
data	String	是	扩展属性的值

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D2xxxxxxxxF31A2697",
    "dataKey": "xxxx"
  }
```

```
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": "xxxx"
}
```

删除设备扩展信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/extended/property/delete	1.0.0	删除物的扩展属性数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	物的唯一标识符
dataKey	String	是	扩展属性标识符

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454181",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "iotToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxx4F31A2698",
    "dataKey": "xxxx"
  }
}
```

- 正常返回示例

```
{
 "code": 200,
```

```
  "message": "success",
  "localizedMsg": null,
  "data": null
}
```

查询用户的小组件设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/iotx/ilop/queryComponentProduct	1.0.0	查询用户的小组件设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无

返回参数

参数	类型	必有	描述
data	JSON Array	是	小组件设备列表；单个设备数据结构请参考下文“小组件设备详情”表格

小组件设备详情

参数	类型	必有	描述
productKey	string	是	产品pk
deviceName	string	是	设备名称
iotId	string	是	设备id
properties	JSON Array	是	属性列表，详细结构参见下文表格
nickName	string	是	别名
productName	string	是	产品名称
iconUrl	string	是	图片地址
deviceStatus	int	是	设备状态

属性列表详情

参数	类型	必有	描述
propertyIdentifier	string	是	属性id
propertyName	string	是	属性名称
propertyValue	string	是	属性取值
propertyDataType	string	是	属性取值类型

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params": {
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": [{
  "deviceName":"xxx",
  "deviceStatus":1,
  "iconUrl":"http://iotx-paas-admin.oss-cn-shanghai.aliyuncs.com/publish/image/
xxxxxx.png",
  "iotId":"xxxxxx",
  "productKey":"xxx",
  "productName":"a",
  "properties":[{
   "propertyDataType":"bool",
   "propertyIdentifier":"PowerSwitch",
   "propertyName":"电源开关",
   "propertyValue":"1"}]
   },{
  "deviceName":"xxxx",
  "deviceStatus":1,
  "iconUrl":"http://iotx-paas-admin.oss-cn-shanghai.aliyuncs.com/publish/image/
xxxxx.png",
  "iotId":"xxxx",
  "productKey":"xxx",
  "productName":"xxx",
  "properties":[{
   "propertyDataType":"bool",
   "propertyIdentifier":"LightSwitch",
   "propertyName":"主灯开关",
   "propertyValue":"0"
  }]
 }]
```

```
}
```

更新用户的小组件设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/iotx/ilop/updateComponentProduct	1.0.0	更新用户的小组件设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
data	数组	是	小组件设备列表

小组件设备详情

参数	类型	必有	描述
productKey	string	是	产品productKey
iotId	string	是	设备id
properties	数据	否	属性列表

属性列表详情

参数	类型	必有	描述
propertyIdentifier	string	否	属性id

返回参数

无

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params": {
    "componentProductList": [{
      "iotId":"xxxxxx",
```

```
      "productKey":"xxx"
    },{
      "iotId":"xxxxxx",
      "productKey":"xxx"
    }]
  }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": null
}
```

查询用户小组件设备的属性列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/iotx/ilop/queryComponentProperty	1.0.0	查询用户小组件设备的属性列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
productKey	string	是	产品key
iotId	string	是	设备id
query	struct	否	查询条件

查询条件详情

参数	类型	必填	描述
dataType	string	否	属性数据类型
I18Language	string	否	多语言
languageList	array	否	多语言列表
allLanguage	bool	否	是否取全部多语言
replaceOrigin	bool	否	是否替换默认语言

返回参数

参数	类型	必有	描述
propertyIdentifier	string	是	属性id
propertyName	string	是	属性名称
propertyValue	string	是	属性取值
propertyDataType	string	是	属性取值类型
added_component	bool	是	是否添加至小组件中

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params": {
    "iotId":"xxxxxx",
    "productKey":"xxx",
    "query":{
     "dataType":"BOOL"
    }
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": [
  {"added_component":false,
   "dataType":"BOOL",
   "i18nDTO":{
    "i18nData":{"name":{"zh-CN":"电源开关"}}
   },
   "productKey":"xxxx",
   "propertyIdentifier":"PowerSwitch",
   "propertyName":"电源开关"}
 ]
}
```

3.8.7 用户服务

提供一组与用户相关的服务接口，客户端应用通过本组服务接口可以实现用户绑定设备列表信息的获取，同时可以实现设备和push通道的解绑操作。

说明:

用户服务已升级，新增了家、空间、群组控制等功能。若您无需使用这些功能，请查看历史版本用户服务。

用户和设备绑定关系

- 获取用户绑定的设备列表
- 根据设备获取绑定关系
- 设备列表排序
- 获取网关的子设备列表
- 查询用户和设备的关系
- 解绑用户和设备
- 管理员解绑设备

设备分享

- 生成分享用的二维码
- 通过扫描二维码绑定设备
- 设备分享给指定的用户
- 获取共享通知列表
- 发起者取消分享
- 被分享者同意或拒绝分享
- 设置设备昵称
- 清空共享消息列表
- 管理员批量查询设备和目标用户的被分享关系

用户和消息绑定关系

- 绑定push通道
- 解绑push通道

虚拟用户

- 创建虚拟用户
- 删除虚拟用户
- 更新虚拟用户信息
- 查询虚拟用户列表
- 查询虚拟用户详情

获取用户绑定的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/listBindingByAccount	1.0.8	获取用户绑定的设备列表	是

请求参数

参数	类型	必填	描述
thingType	String	否	设备的类型： • VIRTUAL（虚拟设备） • VIRTUAL_SHADOW（虚拟影子设备） • WEB • APP • DEVICE（普通设备）
nodeType	String	否	设备的节点类型，可取值包括：DEVICE，GATEWAY
subDevice	Boolean	否	是否为子设备 • ture 表示获取子设备列表 • false 表示获取非子设备（直连设备）列表 • null 表示获取全部设备列表 归属在某一个网关设备下的设备称为子设备。注意该参数只对2020.01.08 之后新绑定的子设备生效，该日期之前绑定的子设备数据不支持根据该字段过滤
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于100

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的设备总数。
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	设备的详情列表
-	identityId	String	用户的身份id
-	iotId	String	设备的 iotId，设备的唯一标识
-	deviceName	String	设备的名称
-	productKey	String	设备所属产品的ProductKey
-	productName	String	设备所属产品的名称
-	productImage	String	设备所属产品的图标，如果未定义产品图标，该字段和 categoryImage 字段一致
-	productModel	String	设备所属产品的型号
-	categoryImage	String	设备所属产品品类的图标
-	nickName	String	设备的昵称，可为空
-	netType	String	设备的入网类型，可取值： • NET_LORA（LoRa） • NET_CELLULAR（GPRS） • NET_WIFI（WiFi） • NET_ZIGBEE（ZigBee） • NET_ETHERNET（以太网） • NET_OTHER（其他）

参数	子参数	类型	描述
-	thingType	String	设备的类型： • VIRTUAL（虚拟设备） • VIRTUAL_SHADOW（虚拟影子设备） • WEB • APP • DEVICE（普通设备）
-	status	Byte	设备的状态，可取值：0（未激活），1（在线），3（离线），8（禁用）
-	owned	Integer	设备和用户的关系，可取值：0（分享者），1（拥有者）
-	nodeType	String	设备的节点类型，可取值包括：DEVICE，GATEWAY
-	identityAlias	String	用户的显示名（mobile 或 loginName 或 email）
-	subDevice	Boolean	标识是否为子设备，true：表示为子设备，false 或者 null：表示非子设备（为直连设备）
-	gmtModified	Date	修改时间（绑定时间）

示例

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
   },
   "params":{
```

```
    "isSubDevice":false,
    "pageNo":1,
    "pageSize":10
  }
}
```

- 正常返回示例

```
{
  "id":"4de2c367-xxxx-xxxx-aa15-8c585e595d92",
  "code":200,
  "message":"success",
  "localizedMsg":null,
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "identityId":"xxx",
        "iotId":"edwB6TqvxxxxxxNa000100",
        "deviceName":"摄像头1",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片",
        "productModel":"产品型号",
        "categoryImage":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
        "thingType":"DEVICE",
        "status":1,
        "owned":1,
        "nodeType":"DEVICE",
        "identityAlias":"133xxx",
        "subDevice":false,
        "gmtModified":"xxx"
      },
      {
        "identityId":"xxx",
        "iotId":"xadqskxxxxxx9h000100",
        "deviceName":"摄像头2",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片",
        "productModel":"产品型号",
        "categoryImage":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
        "thingType":"DEVICE",
        "status":1,
        "owned":1,
        "nodeType":"DEVICE",
        "identityAlias":"133xxx",
        "subDevice":false,
        "gmtModified":"xxx"
      }
    ]
  }
```

```
}
```

根据设备获取绑定关系

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/listBindingByDev	1.0.2	根据设备获取绑定关系	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id
pageNo	Int	是	当前页号，从1开始的页序号
pageSize	Int	是	页大小，单页的item数量上限
owned	Int	否	表示设备绑定的类型：0：作为分享设备被他人分享；1：设备拥有者；null：所有

返回参数

参数	子键	类型	描述
total	-	Integer	云端总数
pageNo	-	Integer	当前页号
pageSize	-	Integer	页大小
data	-	List<>	用户绑定设备列表
-	identityId	String	用户的身份ID
-	iotId	String	设备的iotId
-	productKey	String	设备的productKey
-	deviceName	String	设备的deviceName
-	productName	String	设备的产品名称
-	productImage	String	设备的产品图片

参数	子键	类型	描述
-	productModel	String	设备的产品型号
-	categoryImage	String	品类图标
-	nickName	String	用户对设备的昵称，用户通过setDeviceNickName设置的昵称
-	netType	String	设备入网类型
-	thingType	String	设备的类型： • VIRTUAL（虚拟设备） • VIRTUAL_SHADOW（虚拟影子设备） • WEB • APP • DEVICE（普通设备）
-	nodeType	String	设备的节点类型：DEVICE，GATEWAY
-	status	Byte	设备的状态 0：未激活；1：在线；3：离线；8：禁用
-	owned	Byte	0：分享者；1：拥有者
-	identityAlias	String	用户的显示名：mobile，loginName或email
-	gmtModified	Date	修改绑定的时间
-	description	String	描述

示例

- 请求示例

```
{
  "request": {
    "iotToken": "109049c80xxxxxxxxf62e29a3ba",
    "apiVer": "1.0.2"
  },
  "id": 1508212818676,
  "params": {
    "pageNo":1,
    "pageSize":10,
    "iotId":"fy2Z1oZFxxxxxxxx101edf00"
```

```
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [
    {
      "productModel": "testProduct",
      "iotId": "fy2Z1oZFxxxxxxx00101edf00",
      "netType": "NET_WIFI",
      "identityId": "5082opabf5192xxxxxxxx980ae6f6093",
      "thingType": "DEVICE",
      "nodeType": "DEVICE",
      "productKey": "a1xxxxJNb",
      "deviceName": "b65cb59xxxxxxxx08592cd695fb",
      "productName": "0dxxxx95",
      "status": 3
    }
  ],
  "id": "e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
}
```

- 异常返回示例

```
{
 "code":2064,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"need authorize to bind"
}
```

设备列表排序

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/sortDevices	1.0.2	对设备进行排序	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	子键	类型	必填	描述
sortDeviceDTOList	-	List<>	是	设备排序有变化的数据
-	iotId	String	是	设备ID

参数	子键	类型	必填	描述
-	fromOrder	Integer	是	排序变化前设备的位置
-	toOrder	Integer	是	排序变化后设备的位置

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxxf6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "sortDeviceDTOList":[
         {"iotId": "Z75bTa4dxxxxxxxxRW001056af10",
          "fromOrder": 1,
          "toOrder":  3
         },
         {"iotId": "faHSHCOxxxxxxxxxZQoN00108dcf00",
          "fromOrder": 2,
          "toOrder":  1
         },
         {"iotId": "Spb4eatt1Lxxxxxxxx00108a4410",
          "fromOrder": 3,
          "toOrder":  2
         }
      ]
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2000,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"sortDevice param error!"
```

```
}
```

生成分享用的二维码

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/generateShareQrCode	1.0.2	生成设备分享用的二维码，支持批量设备，有效期30分钟	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotIdList	List< String >	否	设备iotId列表，列表最大个数为20，该参数不能为空

返回参数

参数	类型	描述
qrKey	String	生成的二维码key

示例

- 请求示例

```
{
    "request": {
        "iotToken": "109049cxxxxxxxx6f62e29a3ba",
        "apiVer": "1.0.2"
    },
    "id": 1508212818676,
    "params": {
        "iotIdList":["xxxx"],
        "sceneIdList":["xxxx"]
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
    "code": 200,
    "data": {
        "qrKey": "xxxxxx"
    },
    "id": "6ec222eb-87fc-xxxx-xxxx-351df196ceb3"
```

```
}
```

- 异常返回示例

```
{
  "code":2081,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"scene not found by sceneId"
}
```

通过扫描二维码绑定设备或者场景

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/scanBindByShareQrCode	1.0.8	通过扫描二维码绑定设备或者场景，绑定后是普通成员	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
qrKey	String	是	扫描的二维码key

返回参数

参数	类型	描述
ScanShareQrCodeV2Resp	Object	绑定的设备或者场景ID列表

ScanShareQrCodeV2Res结构

参数	类型	描述
iotIdList	List	绑定的设备或者场景ID列表

示例

- 请求示例

```
{
   "request":{
      "iotToken":"109049c80xxxxxxxx6f62e29a3ba",
      "apiVer":"1.0.2"
   },
   "id":1508212818676,
   "params":{
```

```
      "homeId":"50f5opxxxxxxxxxxxxf28104512",
      "qrKey":"xxxx"
   },
   "version":"1.0"
}
```

- 正常返回示例

```
{
   "code":200,
   "message":"success",
   "localizedMsg":"",
   "data":{
      "iotIdList":[
         "CKuX1xxxxxxxxxxxxxxx00100",
         "vbiM2xxxxxxxxxxxxxxx00100"
      ]
   },
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2067,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"qrKey not exist"
}
```

设备分享给指定的用户

定义描述

path	版本	描述	是否需要鉴权
/uc/shareDevicesAndScenes	1.0.7	通过指定手机、邮箱或具体identityId的方式将设备和场景分享给其他人；对于使用自有账号体系的用户，在智能生活平台一般不会保存对应账号的手机和邮箱等信息，此时可以借助自有账号id和智能生活平台identityId映射关系（用户可自行存储该映射关系），来实现该分享接口的调用；每次分享最多可以分享20个设备	是，客户端SDK需启用用户身份的鉴权

请求参数

参数	类型	必填	描述
iotIdList	List< String >	是	设备iotId列表，每次最多分享20个
targetIdentityId	String	否	分享目标身份ID（用户属性对/身份ID 两者须至少有一个）
accountAttr	String	否	唯一标识目标用户的属性（如：手机号、邮箱的值）
accountAttrType	String	否	唯一标识目标用户的属性类型（如：手机号、邮箱等），手机号：MOBILE；邮箱：EMAIL
mobileLocationCode	String	否	手机号区位码，如86

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxx5f6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "accountAttr":"13656637684",
      "accountAttrType": "MOBILE",
       "iotIdList":[
                  "inqfGrp0xxxxxxxxpD00100fd600",
                  "ETytXLwjxxxxxxxx4qG0010629e00"
               ]
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
```

```
}
```

- 异常返回示例

```
{
 "code":2064,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"need authorize to bind"
}
```

获取共享通知列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/getShareNoticeList	1.0.7	获取共享通知列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
pageNo	Int	是	当前页号，从1开始的页序号
pageSize	Int	是	页大小，单页item的数量上限，最大值为200。如果传入的值超过200，系统会自动按照200条来查询
groupBy	String	否	是否按照批次的方式来获取通知列表。可取值为 NONE, BATCH，如果传 BATCH，那么同一批次的分享，只会有一条通知。默认值为 NONE，表示不作 groupBy 处理。

返回参数

参数	子键	子键	类型	描述
total	-	-	Integer	云端总量

参数	子键	子键	类型	描述
pageNo	-	-	Integer	当前页号
pageSize	-	-	Integer	页大小
data	-	-	List<>	分享列表
-	initiatorAlias	-	String	发起分享用户的账号，手机号码或者邮箱地址
-	receiverAlias	-	String	被分享用户的账号，手机号码或者邮箱地址
-	targetId	-	String	共享的物的ID。 **说明：** 按照 BATCH 查询时，如果批次内有多个记录，此字段不会返回
-	targetType	-	String	共享的物的类型
-	productImage	-	String	设备产品图片
-	deviceName	-	String	设备的deviceName **说明：** 按照 BATCH 查询时，如果批次内有多个记录，此字段不会返回
-	isReceiver	-	Integer	当前用户是否是消息接收者
-	gmtCreate	-	Long	创建时间
-	gmtModified	-	Long	修改时间

参数	子键	子键	类型	描述
-	recordId	-	String	分享记录唯一标识 **说明：** 按照 BATCH 查询时，如果批次内有多个记录，此字段不会返回
-	batchId	-	String	分享批次Id，批量分享设备时，同一批次id相同，客户端可以根据此判断是否为同一批分享，由此做聚合展示等
-	status	-	Integer	状态 • -1: 初始化 • 0：同意 • 1：拒绝 • 2：取消 • 3：过期 • 4：抢占 • 5：删除 • 6：发起者已解绑 • 99：异常
-	description	-	String	描述
-	categoryImage	-	String	品类图标 **说明：** 按照 BATCH 查询时，如果批次内有多个记录，此字段不会返回

参数	子键	子键	类型	描述
-	productName	-	String	产品名称 **说明：** 按照 BATCH 查询时，如果批次内有多个记录，此字段不会返回
-	batchInfo	-	Struct	如果批次内包含多个记录，此字段不为空，其中包含了该批次的信息。
-	-	recordCount	Integer	当前批次中所包含的分享记录的数量，例如在一个批次中分享了 10 个设备，那么对应包含有 10 个分享记录。

示例

- 请求示例

```
{
    "request":{
        "iotToken":"109049c80xxxxxxxx5fe29a3ba",
        "apiVer":"1.0.2"
    },
    "id":1508212818676,
    "params":{
        "pageNo":1,
        "pageSize":10,
        "groupBy":"BATCH"
    },
    "version":"1.0"
}
```

- 正常返回示例

```
{
    "code":200,
    "data":{
        "total":1,
        "pageNo":1,
        "pageSize":10,
        "data":[
```

```
        {
          "gmtCreate":"15275845255",
          "gmtModified":"15235845255",
          "description":"向15163587365共享设备中，待对方确认",
          "targetType":"DEVICE",
          "batchId":"ACCOUNT_DEV_SHARE_3fbd910a-xxxxxx-09aabf5fe5b5",
          "initiatorAlias":"15632547896",
          "receiverAlias":"15163587365",
          "isReceiver":0,
          "status":-1,
          "batchInfo":{
            "recordCount":10
          }
        }
      ]
  },
  "id":"e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
}
```


- 异常返回示例


```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```


获取批次内共享通知列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/user/device/shared/record/query	1.0.0	查询设备分享记录。支持按照批次 id 进行查询。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
pageNo	Int	是	当前页号，从1开始的页序号
pageSize	Int	是	页大小，单页的item数量上限，最大为 20
batchId	String	是	要查询的批次id

返回参数

参数	子键	类型	描述
total	-	Integer	云端总量
pageNo	-	Integer	当前页号
pageSize	-	Integer	页大小
data	-	List<>	分享列表
-	initiatorAlias	String	发起分享用户别名
-	receiverAlias	String	被分享用户别名
-	targetId	String	共享的物的ID。
-	targetType	String	共享的物的类型
-	productImage	String	设备产品图片
-	deviceName	String	设备的deviceName
-	isReceiver	Integer	当前用户是否是消息接收者
-	gmtCreate	Long	创建时间
-	gmtModified	Long	修改时间
-	recordId	String	分享记录唯一标识
-	batchId	String	分享批次Id，批量分享设备时，同一批次id相同，客户端可以根据此判断是否为同一批分享，由此做聚合展示等
-	status	Integer	状态 • -1: 初始化 • 0：同意 • 1：拒绝 • 2：取消 • 3：过期 • 4：抢占 • 5：删除 • 6：发起者已解绑 • 99：异常

参数	子键	类型	描述	
-	description	String	描述	
-	categoryImage	String	品类图标	
-	productName	String	产品名称	

示例

- 请求示例

```
{
  "request":{
    "iotToken":"109049c80xxxxxxxx5fe29a3ba",
    "apiVer":"1.0.2"
  },
  "id":1508212818676,
  "params":{
    "pageNo":1,
    "pageSize":10,
    "batchId":"ACCOUNT_DEV_SHARE_3fbd910a-xxxxxx-09aabf5fe5b5"
  },
  "version":"1.0"
}
```

- 正常返回示例

```
{
  "code":200,
  "data":{
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "gmtCreate":"15275845255",
        "gmtModified":"15235845255",
        "targetId":"fy2Z1oZFWxxxxxxxx0101edf00",
        "categoryImage":"http://xxx",
        "description":"向15163587365共享设备中，待对方确认",
        "targetType":"DEVICE",
        "recordId":"a1OwEjYxxxxdfdFJNb",
        "batchId":"ACCOUNT_DEV_SHARE_3fbd910a-xxxxxx-09aabf5fe5b5",
        "deviceName":"b65cb59xxxxxxxx92cd695fb",
        "productName":"0dxxxx195",
        "initiatorAlias":"15632547896",
        "receiverAlias":"15163587365",
        "isReceiver":0,
        "status":-1
      }
    ]
  },
  "id":"e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
```

```
}
```

- 异常返回示例

```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

发起者取消分享

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/cancelShare	1.0.7	在分享用户接受分享前，发起者取消分享	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
recordIdList	List< String >	否	分享记录列表。recordIdList 和 batchId 其中一个必传。
batchId	String	否	批次 id，recordIdList 和 batchId 其中一个必传。 传入 batchId 时，将取消批次内所有设备的分享

返回参数

无

示例

- 请求示例

```
{
   "request":{
      "iotToken":"109049c8xxxxxxxx6f62e29a3ba",
      "apiVer":"1.0.2"
   },
   "id":1508212818676,
```

```
  "params":{
    "recordIdList":[
      "d4eb352adxxxxxxxx1d53febd503",
      "2568degt2xxxxxxxxe91fb61d53febd3"
    ]
  },
  "version":"1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
  "code":2093,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"forbidden to repeat cancel sharing."
}
```

被分享者同意或拒绝分享

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/confirmShare	1.0.7	被分享者同意/拒绝分享	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
recordIdList	List< String >	否	分享记录列表，单次同意（拒绝）分享的数量上限为 10 个。recordIdList 和 batchId 其中一个必传
agree	Integer	是	0：不同意；1：同意

参数	类型	必填	描述
batchId	String	否	批次 id，recordIdList 和 batchId 其中一个必传。 传入 batchId 时，将同意或拒绝批次内所有设备的分享。

返回参数

无

示例

- 请求示例

```
{
   "request":{
      "iotToken":"109049c80xxxxxxxxf62e29a3ba",
      "apiVer":"1.0.2"
   },
   "id":1508212818676,
   "params":{
      "recordIdList":[
         "d4eb352adxxxxxxxx53febd503",
         "2568degt2xxxxxxxx91fb61d53febd3"
      ],
      "agree":1
   },
   "version":"1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2092,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"forbidden to repeat reject sharing."
}
```

设置设备昵称

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/setDeviceNickName	1.0.2	设置设备昵称	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
nickName	String	是	昵称

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049cxxxxxxxxf6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "iotId": "5070op867xxxxxxxx49d87c207471699",
       "nickName":"nickName"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
  "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
```

```
}
```

解绑用户和设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/unbindAccountAndDev	1.0.2	解绑用户和设备 • 若用户不是拥有者（管理员），则只解绑自己 • 若用户是拥有者（管理员），则再解绑与该设备绑定的所有用户 • 若用户是管理员且设备是网关，则还会解绑用户和其所有的子设备	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	否	设备ID

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c8xxxxxxxx62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "iotId":"xxxx"
   },
   "version": "1.0"
```

```
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
  "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

查询用户和设备的关系

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/getByAccountAndDev	1.0.2	查询用户和设备的关系，如不存在，返回一个空对象	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID

返回参数

参数	类型	描述
identityId	String	用户的身份ID
iotId	String	设备的iotId
AproductKey	String	设备的productKey
deviceName	String	设备的deviceName
productName	String	设备的产品名称
productImage	String	设备的产品图片
productModel	String	设备的产品型号

参数	类型	描述
nickName	String	用户对设备的昵称
netType	String	设备入网类型
thingType	String	设备的类型： • VIRTUAL（虚拟设备） • VIRTUAL_SHADOW（虚拟影子设备） • WEB • APP • DEVICE（普通设备）
nodeType	String	设备的节点类型：DEVICE；GATEWAY
status	Byte	设备的状态 0：未激活 ； 1：在线；3：离线；8：禁用
owned	Byte	0：分享者；1：拥有者
identityAlias	String	用户的显示名：mobile，loginName或email
gmtModified	Date	修改绑定的时间

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxx6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "iotId":"xxxx"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data":
    {
      "productModel": "testProduct",
      "iotId": "fy2Z1oZxxxxxxxx101edf00",
      "netType": "NET_WIFI",
      "identityId": "5082opabf51xxxxxxxx06fbdae6f6093",
      "thingType": "DEVICE",
      "nodeType": "DEVICE",
```

```
    "productKey": "a1OxxxxNb",
    "deviceName": "b65cb59e6xxxxxxxx2cd695fb",
    "productName": "0d2xxxx95",
    "status": 3
   },
  "id": "e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
}
```

- 异常返回示例

```
{
 "code":2063,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"iotId not exist"
}
```

管理员解绑设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/unbindByManager	1.0.2	用户接受设备分享后，管理员解绑指定的用户和设备。要求当前登录用户是设备的管理员、拥有者。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
targetIdentityId	String	是	分享用户的identityId（设备的普通成员），非管理员的identityId
iotIdList	List< String>	是	要解绑的设备ID列表

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxx62e29a3ba",
      "apiVer": "1.0.2"
   },
```

```
  "id": 1508212818676,
  "params": {
    "targetIdentityId":"xxxx",
    "iotIdList":[
                "d4eb352adxxxxxxxx1d53febd503",
                "2568degt2xxxxxxxx91fb61d53febd3"
              ]
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2083,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"you are not the manager of some devices"
}
```

清空共享消息列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/clearShareNoticeList	1.0.2	清空共享消息列表（包括本人发起和接收全部）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无

返回参数

无

示例

- 请求示例

```
{
  "request": {
    "iotToken": "c4b3f64xxxxxxxe7a06a3c5dcb3",
    "apiVer": "1.0.2"
  },
  "id": 1508212818676,
```

```
    "params": {
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
    "code": 200,
    "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

管理员批量查询设备和目标用户的被分享关系

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/user/device/shared/batch/get	1.0.0	管理员批量查询设备和目标用户的被分享关系。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	子健	类型	必填	描述
targetIdentityId	-	String	是	目标用户的identityId，用户的唯一标识
deviceList		JSONArray	是	设备列表，size的最大限制为50个

参数	子健	类型	必填	描述
	iotId	String	否	设备唯一标识的iotId • 该字段为空时，则**productKey**和**deviceName**字段必填 • 该字段为不为空时，即使配置了**productKey**和**deviceName**字段，仍以**iotId**为设备的唯一标识
	productKey	String	否	设备所属产品的productKey，当**iotId**字段为空时，该字段必填
	deviceName	String	否	设备的名称，当**iotId**字段为空时，该字段必填

返回参数

参数	子参	类型	描述
data	-	JSONArray	设备的被分享关系详情列表
-	iotId	String	设备的iotId，设备的唯一标识
-	productKey	String	设备所属产品的productKey
-	deviceName	String	设备的名称
-	shared	Boolean	设备是否分享给了目标用户

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"xxx"
  },
  "params":{
     "targetIdentityId": "xxx",
    "deviceList":[
      {
        "iotId":"edwB6xxxxNa000100"
      }
    ]
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":[
    {
      "iotId":"edwB6xxxxNa000100",
       "productKey":"xxx",
       "deviceName":"xxx",
      "shared":true
    }
  ]
}
```

绑定push通道

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/bindPushChannel	1.0.2	绑定push通道	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
deviceId	String	是	push通道的deviceId

参数	类型	必填	描述
deviceType	String	是	设备类型：iOS或ANDROID，值由业务方和app协商

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxxf62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "deviceId":"xxxx",
      "deviceType":"iOS"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

解绑push通道

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/unbindPushChannel	1.0.2	绑定push通道	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
deviceId	String	是	push通道的deviceId

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxf62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "deviceId":"xxxx"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

获取网关的子设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/subdevices/list	1.0.2	获取用户绑定的网关子设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
pageNo	Int	是	当前页号，从1开始的页序号
pageSize	Int	是	页大小，单页的item数量上限

返回参数

参数	子键	类型	描述
total	-	Integer	云端总数
pageNo	-	Integer	当前页号
pageSize	-	Integer	页大小
data	-	List<>	子设备列表
-	iotId	String	设备的iotId
-	productKey	String	设备的productKey
-	deviceName	String	设备的deviceName
-	nickName	String	设备的产品名称
-	image	String	品类图片
-	status	Byte	设备的在线状态：0 - 未激活， 1 - 在线， 3 - 离线， 8 - 禁用

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c8xxxxxxxx6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "iotId":"xxxxxxx",
      "pageNo":1,
      "pageSize":10
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
```

```
 "code": 200,
 "data": [
  {
   "iotId": "testIot",
   "productKey": "fy2Z1oZxxxxxxxx0101edf00",
   "deviceName": "NET_WIFI",
   "nickName": "xxxxxx",
   "image": "http://abc.jpg",
   "status": 0
  },
  {
   "iotId": "testIot",
   "productKey": "fy2Z1oZFxxxxxxxx00101edf00",
   "deviceName": "NExxxxFI",
   "nickName": "xxxxxx",
   "image": "http://abc.jpg",
   "status": 0
  }
 ],
 "id": "e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
}
```


- 异常返回示例


```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```


创建虚拟用户

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/virtual/user/create	1.0.4	创建当前登录用户名下的虚拟用户	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
attrList	List	否	用户的属性及属性值。若为空，则表示只创建一个无属性的用户。属性Key必须是在规定的系统属性中选择

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

返回参数

参数	类型	描述
userId	String	创建的虚拟用户ID
attrList	List	用户的属性值，意义同请求参数

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
  "request": {
    "iotToken": "109049xxxxxxxxf62e29a3ba",
    "apiVer": "1.0.4"
  },
  "id": 1508212818676,
  "params": {
    "attrList": [
        {
          "attrKey":"name",
          "attrValue":"小明"
        }
    ]
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": {
    "userId":"10B6336Exxxxxxxx428F58AA5A18",
    "attrList":[
      {
        "attrKey":"name",
        "attrValue":"小明"
      }
    ]
  }
```

```
}
```

- 异常返回示例

```
{
 "code":503,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"服务不可用",
 "message":"service not available."
}
```

删除虚拟用户

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/virtual/user/delete	1.0.4	删除当前登录用户名下的虚拟用户	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID

返回参数

无

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxxf6f62e29a3ba",
      "apiVer": "1.0.4"
   },
   "id": 1508212818676,
   "params": {
        "userId":"10B633xxxxxxxx99BDD428F58AA5A18"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "message": "success",
   "data": {
```

```
    "userId":"10B6336EFxxxxxxxx8F58AA5A18",
    "attrList":[
      {
        "attrKey":"name",
        "attrValue":"小明"
      }
    ]
  }
}
```

- 异常返回示例

```
{
 "code":503,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"服务不可用",
 "message":"service not available."
}
```

更新虚拟用户信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/virtual/user/update	1.0.4	更新当前登录用户名下的虚拟用户信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID
opType	Integer	是	更新时属性时的操作类型（opType）的类型，目前仅支持三种：1-ADD；2-UPDATE；3-DELETE
attrList	List	是	用户的属性及属性值。属性Key必须是在规定的系统属性中选择，具体系统属性定义

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key

参数	类型	必填	描述
attrValue	String	是	属性值

返回参数

参数	类型	描述
userId	String	创建的虚拟用户ID
attrList	List	用户的属性值，意义同请求参数

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
  "request": {
    "iotToken": "109049c8xxxxxxxx6f62e29a3ba",
    "apiVer": "1.0.4"
  },
  "id": 1508212818676,
  "params": {
      "userId":"10B6336EFxxxxxxxx28F58AA5A18",
      "opType":2,
      "attrList":[
        {
          "attrKey":"name",
          "attrValue":"小李"
        }
      ]
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": {
    "userId":"10B6336EFxxxxxxxx428F58AA5A18",
    "attrList":[
      {
        "attrKey":"name",
        "attrValue":"小明"
      }
    ]
  }
```

```
}
```

- 异常返回示例

```
{
 "code":503,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"服务不可用",
 "message":"service not available."
}
```

查询虚拟用户列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/virtual/user/list	1.0.4	查询当前登录用户名下的虚拟用户列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
pageNo	Integer	是	当前页号，从1开始的页序号
pageSize	Integer	是	页大小，单页的item数量上限

返回参数

参数	类型	描述
total	Long	虚拟用户总数
pageNo	Integer	当前页号
pageSize	Integer	页大小
data	List	虚拟用户列表

data列表结构

参数	类型	必填	描述
userId	String	是	虚拟用户ID

参数	类型	必填	描述
attrList	List	是	用户的属性及属性值。属性Key必须是在规定的系统属性中选择

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
  "request": {
    "iotToken": "109049c80xxxxxxxxf62e29a3ba",
    "apiVer": "1.0.4"
  },
  "id": 1508212818676,
  "params": {
      "pageNo":1,
      "pageSize":10
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": {
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "userId":"10B633xxxxxxxxBDD428F58AA5A18",
        "attrList":[
          {
            "attrKey":"name",
            "attrValue":"小明"
          }
        ]
      }
    ]
  }
}
```

- 异常返回示例

```
{
 "code":503,
```

```
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"服务不可用",
 "message":"service not available."
}
```

查询虚拟用户详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/virtual/user/query	1.0.4	查询当前登录用户名下的虚拟用户详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID

返回参数

参数	类型	是否必选	描述
userId	String	是	虚拟用户ID
attrList	List	是	用户的属性及属性值。属性Key必须是在规定的系统属性中选择

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049cxxxxxxxxf6f62e29a3ba",
      "apiVer": "1.0.4"
   },
   "id": 1508212818676,
   "params": {
        "userId":"10B6336Exxxxxxxx428F58AA5A18"
   },
```

```
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "message": "success",
   "data": {
      "userId":"10B633xxxxxxxx28F58AA5A18",
      "attrList":[
          {
              "attrKey":"name",
              "attrValue":"小明"
          }
      ]
   }
}
```

- 异常返回示例

```
{
 "code":503,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"服务不可用",
 "message":"service not available."
}
```

虚拟用户属性定义表

参数	类型	描述
displayName	String	显示名
name	String	姓名
gender	Integer	性别（1男；2女）
birthday	String	生日
QQ	String	QQ号码
employeeID	String	工号
extNum	String	办公分机号
homeAddress	String	家庭地址
companyAddress	String	公司地址
companyName	String	公司名称
country	String	国家名称
region	String	所在地区
avatar	String	头像（头像的图片URL地址）

3.8.8 消息服务

提供一组消息记录的操作服务接口，客户端应用通过本组服务接口可以实现对推送到用户客户端的消息记录的查询、统计和删除等操作。

消息接收

- 绑定push通道
- 解绑push通道

消息查询、删除、更新

- 统计用户的消息记录数量
- 查询用户的消息记录
- 逻辑删除用户的消息记录
- 物理删除用户的消息记录
- 更新用户的消息记录
- 拉取设备对应的告警配置列表
- 拉取指定用户的共享设备的告警消息开关配置
- 设备所有者设置共享设备的告警提醒配置

消息接收设置

- 拉取设备对应的告警配置列表
- 设置设备告警提醒配置
- 设置设备全量告警的提醒配置

绑定push通道

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/bindPushChannel	1.0.2	绑定push通道	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
deviceId	String	是	push通道的deviceId（消息推送SDK初始化时，根据utdid生成）
deviceType	String	是	设备类型：iOS或Android

返回参数

无业务返回参数

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxx6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "deviceId":"xxxx",
      "deviceType":"iOS"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

解绑push通道

定义描述

path	版本	描述	是否需要用户身份的鉴权
/uc/unbindPush Channel	1.0.2	绑定push通道	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
deviceId	String	是	push通道的deviceId

返回参数

无业务返回参数

示例

- 请求示例

```
{
   "request": {
      "iotToken": "109049c80xxxxxxxx5f6f62e29a3ba",
      "apiVer": "1.0.2"
   },
   "id": 1508212818676,
   "params": {
      "deviceId":"xxxx"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

- 异常返回示例

```
{
 "code":2062,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "localizedMsg":"请求错误",
 "message":"identityId not exist"
}
```

统计用户的消息记录数量

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/record/messagetype/count	1.0.6	统计当前用户的消息记录数量	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

字段名	子键	类型	是否必填	备注
requestDTO		JSON		
	startCreateTime	Date	N	查询开始时间
	endCreateTime	Date	N	查询结束时间
	messageType	String	N	消息类型分为设备（device）、通知（announcement）、分享（share），如果三个类型都不指定（该参数为空），则查询返回所有的数据
	type	String	Y	• MESSAGE：透传消息 • NOTICE：通知消息
	maxId	Long	N	查询条件：id < maxId
	minId	Long	N	查询条件：id > minId
	isRead	Integer	N	0: 未读 1：已读
	eventId	String	N	告警事件ID（由拉取设备对应的告警配置列表接口可以获得）
	iotId	String	N	设备ID

说明:

MESSAGE类型的消息只可以通过接口查询到；而NOTICE类型的消息，除了可以提供接口查询，还会主动通过手机的推送通道Push给用户（前提是用户App中使用了生活物联网平台提供的消息推送的SDK）。

返回参数

字段名	类型	描述
share	Integer	分享消息数量
device	Integer	设备消息数
announcement	Integer	通知消息数

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "requestDTO": {
      "type": "NOTICE"
    }
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "share": 6,
    "device": 0,
    "announcement": 0
  },
  "message": "success"
}
```

查询用户的消息记录

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/record/query	1.0.6	查询当前用户的消息记录	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

字段名	子键	类型	是否必填	备注
requestDTO		JSON		
	startCreateTime	Long	N	查询的开始时间，单位毫秒
	endCreateTime	Date	N	查询的结束时间，单位毫秒
	messageType	String	N	消息类型分为设备（device）、通知（announcement）、分享（share），如果三个类型都不指定（该参数为空），则查询返回所有的数据
	type	String	Y	• MESSAGE：透传消息 • NOTICE：通知消息
	maxId	Long	N	查询条件：id < maxId
	minId	Long	N	查询条件：id > minId
	pageNo	Integer	Y	分页页数，从1开始
	pageSize	Integer	Y	分页大小，最大100
	isRead	Integer	N	0：未读； 1：已读

字段名	子键	类型	是否必填	备注
	sortType	Integer	N	null or 0: gmtCreate倒排 1 : gmtCreate正排
	iotId	String	N	设备id：从 1.0.6 起开放

返回参数

子键	二级子键	三级子键	四级子键	类型	描述
				JSON	
count				Long	总数
data				JSON List	
	id			Long	消息id
	gmtCreate			Long	创建时间
	gmtModified			Date	修改时间
	appKey			Long	
	title			String	
	body			String	
	messageType			String	消息类型：设备、共享、通知
	isRead			Integer	阅读状态，0：未读； 1 ：已读
	extData			JSON	附加信息
		device		JSON	设备信息
			iotId	String	设备id
			productKey	String	产品key
			productName	String	产品名称
			icon	String	图标
			nickName	String	昵称

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "requestDTO": {
      "type": "NOTICE",
      "pageNo": 1,
      "pageSize": 20
    }
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "count": 102,
    "data": [{
      "appKey": 1234567,
      "body": "温度高",
      "extData": {
        "device": {
          "categoryId": 100,
          "iotId": "iotId1",
          "productKey": "p1",
          "productName": "1"
        }
      },
      "gmtCreate": 1525351832000,
      "gmtModified": 1525351851000,
      "id": 1000000000000200439,
      "isRead": 1,
      "messageId": "-1",
      "messageType": "device",
      "title": "测试",
      "type": "NOTICE"
    }]
  },
  "message": "success"
}
```

逻辑删除用户的消息记录

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/record/delete	1.0.6	逻辑删除当前用户的消息记录	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

字段名	子键	类型	是否必填	备注
requestDTO		JSON		
	type	String	Y	• MESSAGE：透传消息 • NOTICE：通知消息
	messageType	String	N	消息类型：设备、共享、通知
	id	Long	N	消息记录id
	maxId	Long	N	删除条件：id < maxId
	minId	Long	N	删除条件：id > minId

说明：

messageType、id、maxId、minId四个参数不能同时为空（即至少要配置其中一个）。

返回参数

无业务返回参数

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "requestDTO": {
      "type": "NOTICE"
    }
```

```
    }
}
```

- 正常返回示例

```
{
    "code": 200,
    "data": 10,
    "message": "success"
}
```

物理删除用户的消息记录

定义描述

path	版本	描述	是否需要用户的身份鉴权
/message/center/record/delete/physical	1.0.6	物理删除当前用户的消息记录	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

字段名	子键	类型	是否必填	备注
requestDTO		JSON		
	type	String	Y	• MESSAGE：透传消息 • NOTICE：通知消息
	messageType	String	N	消息类型：设备、共享、通知
	id	Long	N	消息记录id
	maxId	Long	N	删除条件：id < maxId
	minId	Long	N	删除条件：id > minId

返回参数

字段名	类型	描述
data	Integer	删除的记录数量

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
      "requestDTO": {
         "type": "NOTICE"
      }
   }
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": 10,
   "message": "success"
}
```

更新用户的消息记录

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/record/modify	1.0.6	更新当前用户的消息记录目前仅支持更新只读状态	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

字段名	子键	类型	是否必填	备注
requestDTO		JSON		
	id	Long	N	消息记录id
	type	String	Y	• MESSAGE：透传消息 • NOTICE：通知消息
	messageType	String	N	
	maxId	Long	N	查询条件：id < maxId

字段名	子键	类型	是否必填	备注
	minId	Long	N	查询条件：id > minId
	isRead	Integer	N	需要更新的阅读状态， 0：未读； 1：已读

返回参数

无业务返回参数

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "requestDTO": {
      "type": "NOTICE"
    }
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": 5,
  "message": "success"
}
```

设备所有者设置共享设备的告警提醒配置

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/device/notice/sharing/config/set	1.0.0	设备所有者设置共享设备的告警提醒配置	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
targetId	String	是	被分享者身份ID
eventId	String	是	告警事件ID
noticeEnabled	Boolean	是	消息推送提醒开关，true：开启； false：关闭

返回参数

无业务返回参数

示例

- 请求示例

```
{
 "id": "1509086454180",
 "version": "1.0",
 "request": {
  "apiVer": "1.0.0",
  "iotToken": "token"
 },
 "params": {
  "iotId" : "ChFrkX2CEx6DO88EHXNH0010803d10",
  "targetId":"******************************",
  "eventId" : "18012",
  "noticeEnabled" : false
 }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": null
}
```

拉取指定用户的共享设备的告警消息开关配置

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/device/notice/sharing/config/list	1.0.0	拉取指定设备对应的告警提醒配置列表（只有配置为进行App应用推送的告警信息才会被获取到，同时只有设备所有者才能使用这个接口）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
identityId	String	是	被分享者的统一身份ID

返回参数

参数	类型	必有	描述
alarmNotices	List	否	告警提醒配置列表

表 3-1: 参数alarmNotices列表结构

参数	类型	必填	描述
eventId	String	是	告警事件ID
eventName	String	是	告警事件名称
noticeEnabled	Boolean	是	消息推送提醒开关 true：开启 false: 关闭 如果用户之前从未配置过告警提醒配置，则默认返回true

示例

- 请求示例

```
{
  "id": "1509086454180",
```

```
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
   "iotToken": "token"
  },
  "params": {
     "iotId" : "ChFrkX2CEx6DO88EHXNH0010803d10",
   "identityId": "******************************"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "",
  "data": [
    {
       "eventId" : "18012",
       "eventName": "开门提醒",
       "noticeEnabled" : true
    },
    {
       "eventId" : "18013",
       "eventName": "劫持报警",
       "noticeEnabled" : false
    }
  ]
}
```

拉取设备对应的告警配置列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/device/notice/list	1.0.5	拉取指定设备对应的告警提醒配置列表（只有配置为进行app应用推送的告警信息才会被获取到）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID

返回参数

参数	类型	必有	描述
alarmNotices	List	否	告警提醒配置列表

alarmNotices

参数	类型	必填	描述
eventId	String	是	告警事件ID
eventName	String	是	告警事件名称
noticeEnabled	Boolean	是	消息推送提醒开关 true：开启 false: 关闭 如果用户之前从未配置过告警提醒配置，则默认返回true

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.5",
   "iotToken": "token"
  },
  "params": {
     "iotId" : "ChFrkX2CExxxxxxxxH0010803d10"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "",
  "data": [
    {
       "eventId" : "18012",
       "eventName": "开门提醒",
       "noticeEnabled" : true
    },
    {
       "eventId" : "18013",
       "eventName": "劫持报警",
       "noticeEnabled" : false
    }
  ]
}
```

设置设备告警提醒配置

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/device/notice/set	1.0.5	配置指定设备的某个告警提醒的开关；无论当前登录用户是设备管理员还是非管理员，该接口仅对用户自身生效	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
eventId	String	是	告警事件ID
noticeEnabled	Boolean	是	消息推送提醒开关 true：开启 false: 关闭 如果用户之前从未配置过告警提醒配置，则默认返回true

返回参数

无业务返回参数

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.5",
    "iotToken": "token"
   },
   "params": {
      "iotId" : "ChFrkX2Cxxxxxxxx10803d10",
      "eventId" : "18012",
      "noticeEnabled" : false
   }
}
```

- 正常返回示例

```
{
   "code": 200,
```

```
    "message": "success",
    "data": null
}
```

设置设备全量告警的提醒配置

定义描述

path	版本	描述	是否需要用户身份的鉴权
/message/center/device/global/notice/set	1.0.5	设置指定设备上的所有告警提醒条目的状态；例如，一个设备上共有10条告警提醒，如果调用本接口传入true/false，则所有的10条告警提醒都会被设置为true/false 另外，该全量设置不会覆盖到未来新增的告警配置，即假如在此处设置了false后，如果产品所有者在后台又新增了一条告警，则对应的提醒配置默认仍为true；	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
noticeEnabled	Boolean	是	消息推送提醒开关 true：开启 false: 关闭 如果用户之前从未配置过告警提醒配置，则默认返回true

返回参数

无业务返回参数

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.5",
    "iotToken": "token"
   },
   "params": {
      "iotId" : "ChFrkX2CxxxxxxxxH0010803d10",
      "noticeEnabled" : false
   }
}
```

- 正常返回示例

```
{
   "code": 200,
   "message": "success",
   "data": null
}
```

3.8.9 家空间服务

提供一组家空间相关的接口，您可以根据这些接口开发家、房间以及结合家房间的设备组、场景等相关功能。

创建家

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/create	1.0.0	创建家	是

说明

- 首次创建家（创建第一个家）时，会自动进行数据订正（将老版本用户维度下的设备归属到新建的家下面）。
- 创建家，家下面会默认创建6个房间，分别是：客厅、主卧、次卧、书房、餐厅和厨房。
- 单用户下最多支持创建10个家。
- 单用户下家名称不允许重名。
- 家的名称规则：支持字母、数字、空格和汉字，长度最大支持40个字符（中文算一个字符）。

请求参数

参数	类型	必填	描述
name	String	是	家的名称

返回参数

参数	类型	描述
homeId	String	家的id，家的唯一标识

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"xxx"
  },
  "params":{
    "name":"我的家"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "homeId":"50f5op155xxxx4f2f28104512"
  }
}
```

删除家

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/delete	1.0.0	删除家	是

说明

- 单用户下至少保留1个家。
- 删除家，会自动解绑当前家下的所有设备。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识

返回参数

无

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
   },
   "params":{
      "homeId":"50f5op15xxxx4f2f28104512"
   }
}
```

- 返回示例

```
{
   "code":200,
   "message":"success",
   "localizedMsg":"",
   "data":null
}
```

更新家的基本信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/update	1.0.0	更新家的基本信息	是

说明

- 允许更新家的名称。
- 家的名称不允许重名。
- 家的名称规则：支持字母、数字、空格和汉字，长度最大支持40个字符（中文算一个字符）。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
name	String	否	家的名称

返回参数

无

示例

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
   },
   "params":{
      "homeId":"50f5op1556xxxx2f28104512",
      "name":"我的家"
   }
}
```

- 返回示例

```
{
   "code":200,
   "message":"success",
   "localizedMsg":"",
   "data":null
}
```

设置用户当前所在的家

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/current/update	1.0.0	设置用户当前所在的家	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识

返回参数

无

示例

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
   },
   "params":{
      "homeId":"50f5op155xxxxa4f2f28104512"
   }
}
```

- 返回示例

```
{
   "code":200,
   "message":"success",
   "localizedMsg":"",
   "data":null
}
```

获取家的详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/get	1.0.0	获取家的详情	是

请求参数

参数	类型	必填	描述
homeId	String	是	家id，家的唯一标识

返回参数

参数	类型	描述
homeId	String	家id，家的唯一标识
name	String	家的名称
myRole	String	用户和家的关系 • ADMIN • MEMBER

参数	类型	描述
currentHome	Boolean	表示该home是否为用户当前所在的家
roomCnt	Integer	家的房间总数。
deviceCnt	Integer	家的设备总数
createMillis	Long	家的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op155xxxx4f2f28104512"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "homeId":"50f5op1556xxxx2f28104512",
    "name":"我的家",
    "myRole":"ADMIN",
    "currentHome":false,
    "roomCnt":10,
    "deviceCnt":36,
    "createMillis":1569232952000
  }
}
```

获取家的列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/query	1.0.0	获取家的列表	是

请求参数

参数	类型	必填	描述
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于20

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的家总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	家的详情列表
-	homeId	String	家id，家的唯一标识
-	name	String	家的名称。
-	myRole	String	用户和家的关系 • ADMIN • MEMBER
-	currentHome	Boolean	表示该home是否为用户当前所在的家
-	createMillis	Long	家的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "pageNo":1,
    "pageSize":10
  }
```

```
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "homeId":"50f5op1556xxxxf2f28104512",
        "name":"我的家",
        "myRole":"ADMIN",
        "currentHome":true,
        "createMillis":1569232952000
      }
    ]
  }
}
```

获取家的全部设备下的元素（设备或者设备组）列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/element/query	1.0.0	获取家的全部设备下的元素（设备或者设备组）列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
propertyIdentifiers	List < String >	否	物模型的identifier，用于获取物模型的当前状态
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于20

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的元素总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	满足条件的元素详情列表
-	elementType	String	元素类型 • DEVICE • CONTROL_GROUP
-	elementId	String	元素（设备或者设备组）id，元素的唯一标识
-	masterDeviceIotId	String	设备组主控设备的iotId，elementType为CONTROL_GROUP时有效
-	name	String	元素（设备或者设备组）名称
-	roomId	String	设备所在房间的id，elementType为DEVICE时有效
-	roomName	String	设备所在房间的名称，elementType为DEVICE时有效
-	productKey	String	元素（设备或者设备组）所属产品的ProductKey
-	productName	String	元素（设备或者设备组）所属产品的名称，elementType为DEVICE时有效
-	productImage	String	设备所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致

参数	子参数	类型	描述
-	productModel	String	设备所属产品的型号，elementType为DEVICE时有效
-	categoryKey	String	设备所属产品的品类key，elementType为DEVICE时有效
-	categoryImage	String	设备所属产品品类的图标，elementType为DEVICE时有效
-	panelPageRouterUrl	String	设备所属产品品类的面板页面路由URL。elementType为DEVICE且categroyKey为Camera时有效
-	nickName	String	设备的昵称，elementType为DEVICE时有效，可为空
-	netType	String	设备的入网类型，elementType为DEVICE时有效
-	thingType	String	设备的类型，elementType为DEVICE时有效 • VIRTUAL • WEB • APP • DEVICE
-	status	Integer	设备的状态，elementType为DEVICE时有效 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用

参数	子参数	类型	描述
-	controlGroupStatus	String	设备组的状态 • NORMAL • INVALID 当设备组下的设备全部离线或者数量<= 1时展示异常态，elementType为CONTROL_GRO有效
-	nodeType	String	设备的节点类型，elementType为DEVICE时有效 • DEVICE • GATEWAY
-	description	String	设备的描述，elementType为DEVICE时有效
-	propertyList	List < DeviceProperty >	设备的TSL属性状态列表

DeviceProperty数据结构说明:

名称	类型	描述
identifier	String	物模型属性的标识符。
name	String	物模型属性的名称。
value	String	物模型属性当前状态值。
dataType	String	物模型属性值的类型。
imageUrl	String	物模型属性的图标 URL。

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
```

```
  "params":{
    "homeId":"50f5op1556fxxxxxf2f28104512",
    "propertyIdentifiers":[
      "LightSwitch"
    ],
    "pageNo":1,
    "pageSize":10
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "elementType":"DEVICE",
        "elementId":"edwB6Tqxxxxa000100",
        "deviceName":"摄像头1",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片 URL",
        "productModel":"产品型号",
        "categoryKey":"Camera",
        "categoryImage":"xxx",
        "panelPageRouterUrl":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
        "thingType":"DEVICE",
        "status":1,
        "nodeType":"DEVICE",
        "description":"设备描述",
        "propertyList":[
          {
            "identifier":"LightSwitch",
            "name":"电源开关",
            "value":"0",
            "dataType":"BOOL",
            "imageUrl":""
          }
        ]
      },
      {
        "elementType":"GROUP",
        "elementId":"bc04e8c1dxxxxb696e6a5",
        "masterDeviceIotId":"edwB6Tqxxxx00100",
        "name":"摄像头组",
        "productKey":"产品 pk",
        "productImage":"产品图片 URL",
        "controlGroupStatus":"NORMAL",
        "propertyList":[
          {
            "identifier":"LightSwitch",
            "name":"电源开关",
            "value":"0",
            "dataType":"BOOL",
            "imageUrl":""
```

```
            }
          ]
        }
      ]
    }
}
```

移动家的全部设备下元素（设备或者设备组）的顺序

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/element/reorder	1.0.0	移动家的全部设备下元素（设备或者设备组）的顺序	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
elementId	String	是	元素id，元素的唯一标识
targetOrder	Integer	是	元素的目标位置，从0开始

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op155xxxxca4f2f28104512",
    "elementId":"elementId",
    "targetOrder":0
  }
```

```
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

获取家下的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/device/query	1.0.0	获取家下的设备列表，支持多维度检索。	是

请求参数

参数	类型	必填	描述
deviceNodeType	String	否	根据设备节点类型过滤 • DEVICE • GATEWAY
productKey	String	否	产品的productKey
categoryKey	String	否	产品的品类key
homeId	String	否	家的id，家的唯一标识，用于过滤归属在某一个家的设备列表
roomId	String	否	房间的id，房间的唯一标识，用于过滤归属在某一个房间的设备列表
excludedHomeId	String	否	家的id，家的唯一标识，用于过滤未归属在某一个家的设备列表
excludedRoomId	String	否	房间的id，房间的唯一标识，用于过滤未归属在某一个房间的设备列表

参数	类型	必填	描述
boundControlGroup	Boolean	否	设备组的绑定状态 • true：已绑定设备组 • false：未绑定设备组
propertyIdentifiers	List < String >	否	物模型的identifier，用于获取物模型的当前状态
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于50

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的设备总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	设备的详情列表
-	iotId	String	设备的iotId，设备的唯一标识
-	deviceName	String	设备的名称
-	homeId	String	设备所在家的id
-	homeName	String	设备所在家的名称
-	roomId	String	设备所在房间的id
-	roomName	String	设备所在房间的名称
-	productKey	String	设备所属产品的ProductKey
-	productName	String	设备所属产品的名称
-	productImage	String	设备所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致
-	productModel	String	设备所属产品的型号

参数	子参数	类型	描述
-	categoryKey	String	设备所属产品的品类key
-	categoryImage	String	设备所属产品品类的图标
-	panelPageRouterUrl	String	设备所属产品品类的面板页面路由URL，categroyKey为Camera时有效
-	nickName	String	设备的昵称，可为空
-	netType	String	设备的入网类型
-	thingType	String	设备的类型 • VIRTUAL • WEB • APP • DEVICE
-	status	Integer	设备的状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
-	nodeType	String	设备的节点类型 • DEVICE • GATEWAY
-	description	String	设备的描述
-	subDeviceIotIdList	List < String >	绑定该用户的网关子设备列表，nodeType为GATEWAY时有效
-	propertyList	List < DeviceProperty >	设备的TSL属性状态列表

DeviceProperty数据结构说明

名称	类型	描述
identifier	String	物模型属性的标识符
name	String	物模型属性的名称

名称	类型	描述
value	String	物模型属性当前状态值
dataType	String	物模型属性值的类型
imageUrl	String	物模型属性的图标URL

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "boundControlGroup":false,
    "pageNo":1,
    "pageSize":10
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "iotId":"edwB6Tqxxxx00100",
        "myRole":"OWNED",
        "deviceName":"摄像头1",
        "homeId":"50f5op1556f6xxxx2f28104512",
        "homeName":"我的家",
        "roomId":"50f5op1556fxxxxf2f28104517",
        "roomName":"Xbox 游戏室",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片",
        "productModel":"产品型号",
        "categoryKey":"Camera",
        "categoryImage":"xxx",
        "panelPageRouterUrl":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
        "thingType":"DEVICE",
        "status":1,
        "nodeType":"DEVICE",
        "description":"设备描述",
        "subDeviceIotIdList":[
```

```
            "xadqsxxxxL89h000100"
          ],
          "propertyList":[
            {
              "identifier":"LightSwitch",
              "name":"电源开关",
              "value":"0",
              "dataType":"BOOL",
              "imageUrl":""
            }
          ]
        },
        {
          "iotId":"xadqskpxxxx000100",
          "myRole":"OWNED",
          "deviceName":"摄像头2",
          "homeId":"50f5op1556xxxxca4f2f28104512",
          "homeName":"我的家",
          "roomId":"50f5op155xxxxf2f28104517",
          "roomName":"Xbox 游戏室",
          "productKey":"产品 pk",
          "productName":"产品名称",
          "productImage":"产品图片",
          "productModel":"产品型号",
          "categoryKey":"Camera",
          "categoryImage":"xxx",
          "panelPageRouterUrl":"xxx",
          "nickName":"设备昵称",
          "netType":"NET_WIFI",
          "thingType":"DEVICE",
          "status":1,
          "nodeType":"DEVICE",
          "description":"设备描述",
          "propertyList":[
            {
              "identifier":"LightSwitch",
              "name":"电源开关",
              "value":"0",
              "dataType":"BOOL",
              "imageUrl":""
            }
          ]
        }
      ]
    }
                }
```


创建房间

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/create	1.0.0	创建房间	是

说明

- 单个家下最多支持创建20个房间。
- 单个家下房间名称不允许重名。
- 房间的名称规则：支持字母、数字、空格和汉字，长度最大支持20个字符（中文算一个字符）。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
name	String	是	房间的名称

返回参数

参数	类型	描述
roomId	String	房间id

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1556xxxxa4f2f28104512",
    "name":"Xbox 游戏室"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "roomId":"50f5op15xxxxbca4f2f28104516"
  }
}
```

删除房间

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/delete	1.0.0	删除房间	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op15xxxxbca4f2f28104512",
    "roomId":"50f5op1556f6xxxxca4f2f28104517"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

更新房间的基本信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/update	1.0.0	更新房间的基本信息	是

说明

- 允许更新房间的名称。
- 单个家下房间名称不允许重名。
- 房间的名称规则：支持字母、数字、空格和汉字，长度最大支持20个字符（中文算一个字符）。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识
name	String	否	房间的名称

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op15xxxx5bca4f2f28104512",
    "roomId":"50f5op1556fxxxxa4f2f28104517",
    "name":"Switch 游戏室"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
```

```
}
```

获取房间的详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/get	1.0.0	获取房间的详情	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识

返回参数

参数	类型	描述
roomId	String	房间的id，房间的唯一标识
name	String	房间的名称
deviceCnt	Integer	房间的设备总数
createMillis	Long	房间的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1xxxxa4f2f28104512",
    "roomId":"50f5op1556fxxxx4f2f28104517"
  }
```

```
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "roomId":"50f5op155xxxxbca4f2f28104517",
    "name":"房间名称",
    "deviceCnt":10,
    "createMillis":1569232952000
  }
}
```

获取家的房间列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/query	1.0.0	获取家的房间列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，不大于20

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的房间总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	房间的详情列表
-	roomId	String	房间的id，房间的唯一标识
-	name	String	房间的名称

参数	子参数	类型	描述
-	deviceCnt	Integer	归属于该房间的设备总数
-	createMillis	Long	房间的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1556xxxxca4f2f28104512",
    "pageNo":1,
    "pageSize":10
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "roomId":"50f5op1556xxxxca4f2f28104516",
        "name":"Xbox 游戏室",
        "deviceCnt":10,
        "createMillis":1569232952000
      },
      {
        "roomId":"50f5op15xxxxd5bca4f2f28104517",
        "name":"Switch 游戏室",
        "deviceCnt":6,
        "createMillis":1569232957000
      }
    ]
  }
}
```

获取房间的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/device/query	1.0.0	获取房间的设备列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识
propertyIdentifiers	List < String >	否	物模型的identifier，用于获取物模型的当前状态
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于50

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的设备总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	设备的详情列表
-	iotId	String	设备的iotId，设备的唯一标识
-	deviceName	String	设备的名称
-	homeId	String	设备所在家的id
-	homeName	String	设备所在家的名称
-	roomId	String	设备所在房间的id
-	roomName	String	设备所在房间的名称
-	productKey	String	设备所属产品的ProductKey
-	productName	String	设备所属产品的名称

参数	子参数	类型	描述
-	productImage	String	设备所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致
-	productModel	String	设备所属产品的型号
-	categoryKey	String	设备所属产品的品类
-	categoryImage	String	设备所属产品品类的图标key
-	panelPageRouterUrl	String	设备所属产品品类的面板页面路由URL，categroyKey为Camera时有效
-	nickName	String	设备的昵称，可为空
-	netType	String	设备的入网类型
-	thingType	String	设备的类型 • VIRTUAL • WEB • APP • DEVICE
-	status	Integer	设备的状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
-	nodeType	String	设备的节点类型 • DEVICE • GATEWAY
-	description	String	设备的描述
-	propertyList	List < DeviceProperty >	设备的TSL属性状态列表

DeviceProperty数据结构说明:

名称	类型	描述
identifier	String	物模型属性的标识符
name	String	物模型属性的名称
value	String	物模型属性当前状态值
dataType	String	物模型属性值的类型
imageUrl	String	物模型属性的图标URL

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op155xxxxxa4f2f28104512",
    "roomId":"50f5op155xxxx5bca4f2f28104517",
    "pageNo":1,
    "pageSize":10
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "iotId":"edwB6TqvxxxxNa000100",
        "deviceName":"摄像头1",
        "homeId":"50f5op1556f6xxxx2f28104512",
        "homeName":"我的家",
        "roomId":"50f5op1556f65xxxx28104517",
        "roomName":"Xbox 游戏室",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片 URL",
        "productModel":"产品型号",
        "categoryKey":"Camera",
        "categoryImage":"xxx",
        "panelPageRouterUrl":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
```

```
            "thingType":"DEVICE",
            "status":1,
            "nodeType":"DEVICE",
            "description":"设备描述",
            "propertyList":[
              {
                "identifier":"LightSwitch",
                "name":"电源开关",
                "value":"0",
                "dataType":"BOOL",
                "imageUrl":""
              }
            ]
          }
      ]
    }
}
```

移动房间下设备的顺序

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/device/reorder	1.0.0	移动房间下设备的顺序	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识
iotId	String	是	设备的iotId，设备的唯一标识
targetOrder	Integer	是	设备的目标位置，从0开始

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
```

```
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
   },
   "params":{
      "homeId":"50f5op155xxxxca4f2f28104512",
      "roomId":"50f5op1556fxxxx4f2f28104517",
      "iotId":"edwB6Tqvxxxxa000100",
      "targetOrder":0
   }
}
```

- 返回示例

```
{
   "code":200,
   "message":"success",
   "localizedMsg":"",
   "data":null
}
```

家下房间的顺序调整

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/order/refresh	1.0.0	家下房间的顺序调整	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomIdList	List < String >	是	roomId列表

返回参数

无

示例

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.0"
```

```
  },
  "params":{
    "homeId":"50f5op155xxxxa4f2f28104512",
    "roomIdList":[
      "2656fd48b1xxxx598a5ced85",
      "2656fd4xxxxd598a5ced87"
    ]
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

房间的设备更新

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/room/device/full/update	1.0.0	房间的设备更新	是

说明

- 支持设备的增加，删除和排序。
- 该接口支持跨家移动和跨房间移动设备。
- 单个房间最多添加100个设备。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
roomId	String	是	房间的id，房间的唯一标识
iotIdList	List < String >	是	设备iotId列表

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5oxxxxa4f2f28104512",
    "roomId":"50f5op15xxxxbca4f2f28104517",
    "iotIdList":[
      "xadqskxxxx000100",
      "UfPAmFxxxx6OF000100"
    ]
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

创建设备组

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/create	1.0.0	创建设备组	是

说明

- 单个家下最多支持创建20个设备组。
- 单个家下设备组名称不允许重名。
- 设备组的名称规则：支持字母、数字、空格和汉字，长度最大支持60个字符（中文算一个字符）。
- 单个设备组下最少2个设备，最多100个设备。
- 一个设备只能添加到一个设备组。
- 必须同一个productKey下的设备才能组建设备组。
- 目前只支持“灯”和“插座”两个品类。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
name	String	是	设备组的名称
productKey	String	是	设备组所属产品的productKey
iotIdList	List < String >	是	设备的iotId列表

返回参数

参数	类型	描述
controlGroupId	String	设备组的id，设备组的唯一标识

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op15xxxxa4f2f28104512",
    "name":"球泡灯组",
    "productKey":"a1xxxxsk",
    "iotIdList":[
      "xadqskpxxxxL89h000100",
      "UfPAmxxxxW6OF000100"
    ]
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"bc04e8c1d4xxxxeb696e6a5"
}
```

删除设备组

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/delete	1.0.0	删除设备组	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupId	String	是	设备组的id，设备组的唯一标识

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op155xxxxa4f2f28104512",
    "controlGroupId":"bc04e8xxxxa5eb696e6a5"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

批量删除设备组

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/batch/delete	1.0.0	批量删除设备组	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupIdList	List < String >	是	设备组id列表

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1xxxxca4f2f28104512",
    "controlGroupIdList":[
      "bc04e8c1xxxx5eb696e6a5",
      "bc04exxxxa5eb696e6a7"
    ]
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

更新设备组的基本信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/update	1.0.0	更新设备组的基本信息	是

说明

- 允许更新设备组的名称。
- 单个家下设备组的名称不允许重名。
- 设备组的名称规则：支持字母、数字、空格和汉字，长度最大支持60个字符（中文算一个字符）。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupId	String	是	设备组的id，设备组的唯一标识
name	String	是	设备组的名称

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1556fxxxx2f28104512",
    "controlGroupId":"bc04e8c1xxxx9a5eb696e6a5",
    "name":"球泡灯组"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
```

```
  "localizedMsg":"",
  "data":null
}
```

获取设备组的详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/get	1.0.0	获取设备组的详情	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupId	String	是	设备组id，设备组的唯一标识

返回参数

参数	类型	描述
controlGroupId	String	设备组id，设备组的唯一标识
name	String	设备组的名称
productKey	String	设备组所属产品的productKey
productImage	String	设备组所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致
status	String	设备组的状态 • NORMAL • INVALID 当设备组下的设备数量 <= 1 时展示异常态
createMillis	Long	设备组的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
```

```
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5opxxxx5bca4f2f28104512",
    "controlGroupId":"bc04exxxx59a5eb696e6a5"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "controlGroupId":"bc04e8xxxxb696e6a5",
    "name":"球泡灯组",
    "productKey":"a1xxxxsk",
    "productImage":"xxx",
    "status":"NORMAL",
    "createMillis":1569232952000
  }
}
```

获取家的设备组列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/query	1.0.0	获取家的设备组列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，大于等于1，小于等于20

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的设备组总数

参数	子参数	类型	描述
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	设备组的详情列表
-	controlGroupId	String	设备组的id，设备组的唯一标识
-	name	String	设备组的名称
-	productKey	String	设备组所属产品的productKey
-	productImage	String	设备组所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致
-	status	String	设备组的状态 • NORMAL • INVALID 当设备组下的设备数量<= 1 时展示异常态
-	createMillis	Long	设备组的创建时间，UNIX时间戳，单位为ms

示例

- 请求示例

```
{
    "id":"xxx",
    "version":"1.0.0",
    "request":{
        "language":"zh-CN",
        "appKey":"xxx",
        "iotToken":"xxx",
        "apiVer":"1.0.0"
    },
    "params":{
        "homeId":"50f5op1556xxxx104512",
        "pageNo":1,
        "pageSize":10
    }
```

```
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "controlGroupId":"bc04e8cxxxx5eb696e6a5",
        "name":"球泡灯组",
        "status":"NORMAL",
        "productKey":"a1Uxxxxsk",
        "parductImage":"xxx",
        "createMillis":1569232952000
      },
      {
        "controlGroupId":"bc04exxxx5eb696e6a7",
        "name":"插座灯组",
        "status":"INVALID",
        "productKey":"a1xxxxvj",
        "parductImage":"xxx",
        "createMillis":1569232952000
      }
    ]
  }
}
```

获取设备组的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/device/query	1.0.0	获取设备组的设备列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupId	String	是	设备组的id，设备组的唯一标识
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，取值范围：1~20

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的设备总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	设备的详情列表
-	iotId	String	设备的iotId，设备的唯一标识
-	deviceName	String	设备的名称
-	productKey	String	设备所属产品的ProductKey
-	productName	String	设备所属产品的名称
-	productImage	String	设备所属产品的图标，如果未定义产品图标，该字段和categoryImage字段一致
-	productModel	String	设备所属产品的型号
-	categoryKey	String	设备所属产品的品类key
-	categoryImage	String	设备所属产品品类的图标
-	nickName	String	设备的昵称，可为空
-	netType	String	设备的入网类型
-	thingType	String	设备的类型 • VIRTUAL • WEB • APP • DEVICE
-	status	Integer	设备的状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用

参数	子参数	类型	描述
-	nodeType	String	设备的节点类型 • DEVICE • GATEWAY
-	description	String	设备的描述

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op1556fxxxxf2f28104512",
    "controlGroupId":"bc04e8c1xxxxeb696e6a5",
    "pageNo":1,
    "pageSize":10
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "data":[
      {
        "iotId":"edwB6Tqxxxx000100",
        "deviceName":"摄像头1",
        "productKey":"产品 pk",
        "productName":"产品名称",
        "productImage":"产品图片 URL",
        "productModel":"产品型号",
        "categoryKey":"Camera",
        "categoryImage":"xxx",
        "nickName":"设备昵称",
        "netType":"NET_WIFI",
        "thingType":"DEVICE",
        "status":1,
        "nodeType":"DEVICE",
        "description":"设备描述"
      }
    ]
  }
```

```
}
```

设备组的设备更新

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/device/full/update	1.0.0	更新设备组内的设备列表	是

说明

- 支持设备的增加，删除和排序。
- 单个设备组下最少2个设备，最多100个设备。
- 一个设备只能添加到一个设备组。
- 必须同一个 productKey 下的设备才能组建设备组。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
controlGroupId	String	是	设备组的id，设备组的唯一标识
iotIdList	List < String >	是	设备iotId列表

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "homeId":"50f5op155xxxxxbca4f2f28104512",
    "controlGroupId":"bc04exxxxf59a5eb696e6a5",
    "iotIdList":[
```

```
        "xadqskxxxx89h000100",
        "UfPxxxxW6OF000100"
      ]
    }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
}
```

获取可创建设备组的产品列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/home/controlgroup/product/query	1.0.1	获取可创建设备组的产品列表	是

请求参数

参数	类型	必填	描述
homeId	String	否	家的id，家的唯一标识
pageNo	Integer	是	分页页码，从1开始
pageSize	Integer	是	分页大小，取值范围：1~50

返回参数

参数	子参数	类型	描述
total	-	Long	满足条件的产品总数
pageNo	-	Integer	分页页码
pageSize	-	Integer	分页大小
data	-	JSONArray	产品的详情列表
-	productKey	String	产品的 productKey
-	productName	String	产品的名称

参数	子参数	类型	描述
-	productImage	String	产品的图标，如果未定义产品图标，该字段和categoryImage字段一致

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.1"
  },
  "params":{
    "homeId":"50f5op1xxxxa4f2f28104512"
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":{
    "total":2,
    "pageNo":1,
    "pageSize":50,
    "data":[
      {
        "productKey":"a1xxxxvsk",
        "productName":"球泡灯",
        "productImage":"xxx"
      },
      {
        "productKey":"a1bxxxxvj",
        "productName":"插座",
        "productImage":"xxx"
      }
    ]
  }
}
```

设置设备组属性

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/controlgroup/properties/set	1.0.0	设置设备组的属性	是

请求参数

参数	类型	必填	描述
controlGroupId	String	是	设备组id，设备组的唯一标识
items	JSON	是	属性信息，组成为key：value，数据格式为JSON；key为属性的标识符（identifier），类型为String，value为属性值，类型为Object

返回参数

无

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.0"
  },
  "params":{
    "controlGroupId":"bc04e8cxxxxeb696e6a5",
    "items":{
      "PowerSwitch":1,
      "Temperature":20
    }
  }
}
```

- 返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":null
```

```
}
```

在家中创建场景

定义描述

path	版本	描述	是否需要登录
/living/scene/create	1.0.1	在家中创建场景。	是

说明:

目前平台提供了家空间管理服务，可以使用相关API搭建自有App的家空间模型。场景服务也对家空间进行了相应支持，提供了管理家中场景的能力。

您一旦使用家空间服务为App用户创建了家，App用户之前创建的场景就会归属于这个家下。此时，App用户需要使用家场景接口方可管理场景。以下接口不能访问家空间中的场景：

- /scene/create
- /scene/info/get

请求参数

参数	类型	必填	描述
homeId	String	是	场景所在家的唯一标识
catalogId	String	是	用于对场景进行分类 • 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景
enable	Boolean	是	是否在创建时启用场景，true表示启用，false表示停用
name	String	是	用户给场景起的名称
description	String	否	对场景的描述
icon	String	是	场景图标的下载链接
iconColor	String	是	场景图标色，例如#FFFFFF
triggers	JSON Object	否	IFTTT 规则的trigger对象

参数	类型	必填	描述
conditions	JSON Object	否	IFTTT规则的condition对象
caConditions	JSON Array	否	CA规则的condition对象
actions	JSON Array	是	action对象
sceneType	String	否	场景规则类型，可取值为：IFTTT和CA，缺省值为IFTTT
mode	String	否	CA规则的模式 • all caConditions 中的条件全部满足时，才会执行action中的内容 • any caConditions 中的任何一个条件被满足，就会执行action中的内容

返回参数

参数	类型	描述
sceneId	String	创建的场景Id

请求示例

```
{
   "homeId": "testHomeId"
   "catalogId":"1",
   "enable":true,
   "name":"testScene",
   "icon":"http://www.xxx.com/xxx.png",
   "iconColor":"#FFFFFF",
   "caConditions":[
      {
         "params":{
            "cron":"42 00 * * *",
            "cronType":"linux",
            "timezoneId":"Asia/Shanghai"
```

```
        },
        "uri":"condition/timer"
      },
      {
        "params":{
          "compareValue":0,
          "compareType":"==",
          "propertyName":"PowerSwitch",
          "productKey":"xxxx",
          "deviceName":"xxxx"
        },
        "uri":"condition/device/property"
      }
    ],
    "actions":[
      {
        "params":{
          "iotId":"xxxx",
          "propertyName":"PowerSwitch",
          "propertyValue":1
        },
        "uri":"action/device/setProperty"
      }
    ],
    "sceneType":"CA",
    "mode":"all"
}
```

返回示例

```
{
    "code": 200,
    "data": "sceneId",
    "message": "success"
}
```

更新家中场景

定义描述

path	版本	描述	是否需要登录
/living/scene/update	1.0.0	更新场景	是

请求参数

参数	类型	必填	描述
catalogId	String	是	用于对场景进行分类 • 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景 **说明：** 在创建场景时指定catalogId，场景创建后catalogId不可以修改，即不能通过修改catalogId的方式，将手动场景变更为自动场景。
sceneId	String	是	场景Id
enable	Boolean	是	true表示启用场景；false表示停用场景
name	String	是	场景的名称
icon	String	是	场景图标URL
iconColor	String	是	场景图标色
description	String	否	对场景的描述
triggers	JSON Object	否	IFTTT规则的trigger对象
conditions	JSON Object	否	IFTTT规则的condition对象
caConditions	JSON Array	否	CA 规则的condition对象
actions	JSON Array	是	action对象
sceneType	String	否	场景规则类型，可取值为IFTTT和CA，缺省值为IFTTT

参数	类型	必填	描述
mode	String	否	CA规则的模式 • all caConditions 中的条件全部满足时，才会执行action中的内容 • any caConditions 中的任何一个条件被满足，就会执行action中的内容

返回参数

参数	类型	描述
sceneId	String	更新的场景Id

请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
   "enable" : true,
   "iconColor" : "#DE4545",
   "reBind" : true,
   "mode" : "any",
   "caConditions" : [
    {
     "uri" : "condition\/timer",
     "params" : {
      "cron" : "3g3 10 * * 1,2,3,4",
      "cronType" : "linux"
     }
    }
   ],
   "sceneType" : "CA",
   "icon" : "http:\\xxx.xxx.com/xxx.png",
   "sceneId" : "2420d2xxxxf93c54fc4e",
   "catalogId" : "1",
   "name" : "10:27 周一 周二 - 电源开关 开启",
   "actions" : [
```

```
    {
     "uri" : "action\/device\/setProperty",
     "params" : {
      "iotId" : "uumR5xxxx00101",
      "propertyName" : "PowerSwitch",
      "propertyValue" : 1
     }
    }
   ]
  }
}
```

返回示例

```
{
   "code": 200,
   "data": "bdc60bexxxxd617697",
   "message": "success"
}
```

删除家中场景

定义描述

path	版本	描述	是否需要登录
/living/scene/delete	1.0.0	删除场景	是

请求参数

参数	类型	必填	描述
sceneId	String	是	场景Id

返回参数

参数	类型	描述
sceneId	String	删除的场景Id

请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
      "sceneId": "0c965xxxx5306a6657",
      "groupId": null
   }
```

```
}
```

正常返回示例

```
{
  "code": 200,
  "data": "0c9653xxxx06a6657",
  "message": "success"
}
```

查询家中场景列表

定义描述

path	版本	描述	是否需要登录
/living/scene/query	1.0.1	查询场景列表	是

请求参数

参数	类型	必填	描述
homeId	String	是	场景所在家的唯一标识
catalogId	String	是	• 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景
pageNo	Integer	是	页序号，不小于1
pageSize	Integer	是	每页数量，不小于1，不大于50

返回参数

参数	子参数	类型	描述
total		Integer	总条目数
pageNo		Integer	从1开始的页序号
pageSize		Integer	每页数量
scenes		Array	返回当前页条目
	id	String	场景id
	status	Integer	场景运行时状态：1表示场景上线；2表示场景下线

参数	子参数	类型	描述
	enable	Boolean	场景开关（App上的开关）：true-场景置为打开；false-场景置为关闭
	icon	String	场景图标连接
	iconColor	String	场景图标颜色
	name	String	场景名称
	description	String	场景描述
	valid	Boolean	是否有效
	catalogId	String	类型：null-场景1.0版本；0-场景；1-自动化

请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "homeId": "testHomeId",
    "catalogId": "1",
    "pageSize": 20,
    "pageNo": 1,
  }
}
```

正常返回示例

```
{
  "code": 200,
  "data": {
    "pageNo": 1,
    "pageSize": 20,
    "scenes": [{
      "description": "电热毯-hy-信噪比--127 ",
      "enable": true,
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "260de82dxxxx691be22",
      "name": "电热毯",
      "status": 2,
      "iconColor":"#19BBFF",
      "valid":false,
      "catalogId":"1"

    }, {
      "description": "light-hy-LightSwitch-1, shuijin-hy-WiFI_SNR--127 ",
      "enable": true,
```

```
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "45dc6fcxxxx6974e5f7",
      "name": "凉宫",
      "status": 1,
       "iconColor":"#19BBFF",
      "valid":true,
      "catalogId":"1"
    }, {
      "description": "fengsan-hy-PowerSwitch-关闭, light-hy-LightSwitch-1 ",
      "enable": true,
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "3ce636xxxx1367f2dc88",
      "name": "风扇灯0522",
      "status": 1,
       "iconColor":"#19BBFF",
      "valid":false,
      "catalogId":"1"
    }, {
      "description": "light-hy-LightSwitch-1 ",
      "enable": true,
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "5b4bdexxxx2612f81",
      "name": "灯0522",
      "status": 2,
       "iconColor":"#19BBFF",
      "valid":false,
      "catalogId":"1"
    }],
    "total": 4
  },
  "message": "success"
}
```


改变家中场景列表表项的排序

定义描述

path	版本	描述	是否需要登录
/scene/list/reorder	1.0.6	改变场景列表中表项的排序	是

请求参数

参数	子参数	类型	必填	描述
homeId		String	是	场景所属家的唯一标识

参数	子参数	类型	必填	描述
groupId		String	是	对应于 /living/scene/create接口的catalogId参数，用于对场景进行分类。 • 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景
newOrders		Array	是	序号变化的场景数组
	sceneId	String	是	场景id
	targetOrder	Integer	是	需要将场景调整到哪个位置，序号从 0 开始

返回参数

参数	类型	描述
code	Integer	操作成功状态

请求示例

```
{
    "id": "1509086454180",
    "version": "1.0",
    "request": {
        "apiVer": "1.0.1",
        "iotToken": "token"
    },
    "params": {
        "groupId": "1",
        "newOrders": [{
            "sceneId": "a596dexxxx9685d25",
            "targetOrder": 2
        }, {
            "sceneId": "c1a8de4xxxxb473c3",
            "targetOrder": 1
        }]
    }
```

```
}
```

正常返回示例

```
{
  "code": 200,
  "message": "success"
}
```

获取家中场景详情

定义描述

path	版本	描述	是否需要登录
/living/scene/info/get	1.0.0	获取场景详情	是

请求参数

参数	类型	必填	描述
sceneId	String	是	待查询的场景ID
catalogId	String	否	• 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景

返回参数

参数	类型	描述
id	String	场景id
enable	Boolean	场景开关状态 • true：打开状态，场景可以正常被触发执行 • false：关闭状态，即使达到触发条件，场景也不会执行
name	String	用户给场景起的名称
description	String	对场景的描述
icon	String	场景图标
iconColor	String	场景图标色

参数	类型	描述
triggersJson	JSON Object	IFTTT规则Trigger内复杂对象的JSON序列化，用于前端展示
conditionsJson	JSON Object	IFTTT规则的condition对象
caConditionsJson	JSON Array	Since场景2.0，CA规则Condition内复杂对象的JSON序列化，用于前端展示
actionsJson	JSON Object	Action内复杂对象的JSON序列化，用于前端展示
sceneType	String	Since场景2.0，场景规则：IFTTT和CA（缺省值为IFTTT）
mode	String	Since场景2.0，CA规则的模式 • all caConditions 中的条件全部满足时， 才会执行action中的内容 • any caConditions 中的任何一个条件被满足，就会执行action中的内容
valid	Boolean	场景的节点是否有效 • true：场景的节点有效 • false：场景的节点失效 设备作为Action节点被解绑后，将不再被场景所控制，此时场景的节点失效 **说明：** 场景中存在多个Action节点时，只要一个Action节点失效，整个场景的**valid**字段就为**false**。

请求示例

```
{
```

```
    "id": "1509086454180",
    "version": "1.0",
    "request": {
        "apiVer": "1.0.1",
        "iotToken": "token"
    },
    "params": {
        "catalogId": "0",
        "sceneId": "debf2dxxxx51496c2"
    }
}
```

返回示例

```
{
    "code":200,
    "data":{
        "id":"testSceneId",
        "name":"测试场景名",
        "description":"这是一个测试场景",
        "enable":true,
        "icon":"https://www.xxx.com/xxx.png",
        "iconColor":"#FFFFFF",
        "sceneId":"testSceneId",
        "valid":true,
        "triggersJson":"{"items":[{"params":{"cron":"0 7 * * 1,2,3,4,5,6,7","cronType":"linux","
timezoneID":"Asia/Shanghai"},"uri":"trigger/timer"}],"uri":"logical/or"}",
        "actionsJson":"[{"uri":"action/device/setProperty","params":{"propertyName":"
PowerSwitch_2","propertyValue":1,"iotId":"testDeviceIotId01","productKey":"testProduct","
deviceName":"testDeviceName01"}}]"
    },
    "message":"success"
}
```

3.8.10 配网服务

提供一组与设备配网和绑定相关的服务接口，通过本组接口可以实现蓝牙、GPRS、Wi-Fi设备的用户绑定流程，本组接口还包含了子设备入网流程所需的相关操作接口。

说明：

配网服务已升级，新增了家、空间、群组控制等功能。若您无需使用这些功能，请查看历史版本配网服务。

配网前准备

- 获取发现设备列表
- 本地发现设备列表信息过滤

设备配网

- 获取秘钥
- 网关接入子设备
- 解除子设备网络拓扑

设备绑定（推荐）

- 根据token查询设备的证书信息
- 基于token方式的设备绑定（Wi-Fi、以太网）
- 基于时间窗口方式的设备绑定（Zigbee、蜂窝网、蓝牙、其他）

历史设备绑定API（不推荐）

- Wi-Fi设备绑定
- 蓝牙设备绑定
- GPRS设备绑定
- 子设备绑定
- 解绑子设备

获取发现设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/enrollee/list/get	1.0.7	分页查询发现设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
pageSize	Integer	是	分页大小，小于等于99
pageNum	Integer	是	页编号，从1开始

返回参数

参数	子键	类型	描述
totalNum		Integer	总数
items		JSON List	对象数组
	gmtCreate	Date	创建时间
	gmtModified	Date	修改时间
	regIotId	String	已配网设备iotId
	regProductKey	String	已配网设备productKey

参数	子键	类型	描述
	regDeviceName	String	已配网设备deviceName
	regProductName	String	已配网设备的产品名称
	enrolleeIotId	String	待配网设备iotId
	enrolleeProductKey	String	待配网设备productKey
	enrolleeDeviceName	String	待配网设备deviceName
	enrolleeProductName	String	待配网设备的产品名称
	bssid	String	路由器bssid
	ssid	String	热点ssid
	rssi	String	热点rssi
	type	Integer	发现者设备类型，0表示device；1表示router
	status	Integer	配网过程状态，0表示初始态； 1表示配网中

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "pageSize": 10,
    "pageNum": 1
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
```

```
      "totalNum":12,
      "items":[
        {
          "gmtCreate":xxx,
          "gmtModified":xxx,
          "regIotId":"xxx",
          "regProductKey":"xxx",
          "regDeviceName":"xxx",
          "enrolleeIotId":"xxx",
          "enrolleeProductKey":"xxx",
          "enrolleeDeviceName":"xxx",
          "bssid":"xxx",
          "ssid":"xxx",
          "rssi":"xxx",
          "type":0,
          "status":1
        }
      ]
    },
    "message": "success"
}
```

本地发现设备列表信息过滤

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/enrollee/product/filter	1.0.7	根据过滤规则从客户端通过本地通讯发现的本地设备列表中，过滤出还未被用户绑定的设备信息 **说明：** 该接口默认只处理已发布的产品，如果您需要同时处理开发中的产品，调用该接口前需在SDK中设置**PRODUCT_SCOPE_ALL**参数： • Android：setProductScope • iOS：[[IMSIotSmartsharedInstance] configProductScope:PRODUCT_SCOPE_ALL];	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotDevices	JSON List	是	本地发现的设备信息列表

iotDevices的列表结构如下

字段名	字段类型	必填	详细描述
productKey	String	是	产品的Key
deviceName	String	是	设备名称

返回参数

参数	类型	描述
productKey	String	产品的Key（与入参一致）
deviceName	String	设备名称（与入参一致）
productName	String	产品名称
netType	Integer	入网类型 • 0：LORA • 3：WIFI • 4：ZIGBEE • 5：BT • 6：CELLULAR • 7：ETHERNET • 8：NET_OTHER
nodeType	Integer	节点类型，0表示设备；1表示网关
categoryKey	String	产品所属品类的品类标识符
categoryName	String	产品所属品类的名称
categoryId	Long	产品归属品类的ID

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params":{
    "iotDevices": [{
      "productKey": "a1xxxxm",
      "deviceName": "xxxxxx"
    },
    {
      "productKey": "a1xxxxb",
      "deviceName": "xxxxxx"
    }]
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data":[{
```

```
            "productKey":"a1xxxxm",
            "deviceName":"xxxxxx",
            "netType":3,
            "nodeType":0,
            "categoryKey":"xxx",
            "categoryName":"xxx",
            "categoryId":14
        }
    ],
    "message": "success"
}
```

根据token查询设备的证书信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/token/check	1.0.9	token校验	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
token	String	是	设备上报的token
productKey	String	否	设备的productKey
deviceName	String	否	设备的deviceName，该参数不为空时，**productKey**不能为空

返回参数

参数	类型	描述
productKey	String	待配网设备productKey
deviceName	String	待配网设备deviceName
iotId	String	设备的iotId

示例

- 请求示例

```
{
    "request": {
        "iotToken": "109049c80bxxxx62e29a3ba",
        "apiVer": "1.0.9"
```

```
    },
    "id": 1508212818676,
    "params": {
        "token":"xxx"
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
      "iotId": "xxxx"
      "productKey": "xxxx",
      "deviceName": "xxxx"
    },
  "id": "e2d74ffe-308e-xxxx-xxxx-9b44a43eabc9"
}
```

- 异常返回示例

```
{
 "code":6608,
 "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
 "message":"token not found"
}
```

基于token方式绑定设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/token/user/bind	1.0.8	根据token校验并绑定设备	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
productKey	String	是	待配网设备productKey
deviceName	String	是	待配网设备deviceName
token	String	是	设备上报的token

返回参数

参数	类型	描述
data	String	已配网设备iotId

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.8"
  },
  "params":{
    "homeId":"50f5op1556xxxxf2f28104512",
    "productKey":"xxx",
    "deviceName":"xxx",
    "token":"xxx"
  }
}
```

- 正常返回示例

```
{
  "id":"xxx",
  "code":200,
  "message":"success",
  "data":"xxxxxxxxxxxxx"
}
```

基于时间窗口的方式绑定设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/time/window/user/bind	1.0.8	根据时间窗口（600秒）校验并绑定设备，从设备上电开始计算	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
productKey	String	是	待配网设备productKey

参数	类型	必填	描述
deviceName	String	是	待配网设备deviceName

返回参数

参数	类型	描述
data	String	已配网设备iotId

示例

- 请求示例

```
{
   "id":"xxx",
   "version":"1.0.0",
   "request":{
      "language":"zh-CN",
      "appKey":"xxx",
      "iotToken":"xxx",
      "apiVer":"1.0.8"
   },
   "params":{
      "homeId":"50f5op15xxxxca4f2f28104512",
      "productKey":"xxx",
      "deviceName":"xxx"
   }
}
```

- 正常返回示例

```
{
   "id":"xxx",
   "code":200,
   "message":"success",
   "data":"xxxxxxxxxxxxx"
}
```

获取秘钥

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/cipher/get	1.0.7	获取秘钥	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	子键	类型	必填	描述
deviceInfo ForCipher			是	JSON对象
	deviceName	String	是	deviceName
	productKey	String	是	productKey，非蓝牙设备必传，蓝牙设备不用传
	productId	String	是	productId为productKey的十六进制字符串，蓝牙设备必传，非蓝牙设备不用传
	cipherType	Integer	是	生成秘钥的类型（具体介绍参见表下方介绍），为空默认为一型一密
	random	String	是	随机数（32字节16进制字符串）
	params	JSON	是	扩展参数（蓝牙设备需要传mac，非蓝牙设备传空map）

cipherType取值介绍如下

```
cipherType=3 一型一密：根据productKey查询productSecret，密钥算法sha256(contentByteArray)。
contentByteArray=productSecret.getByte()+0x2c+hexToByte(random)

cipherType=4 一机一密：根据productKey和deviceName查询deviceSecret，密钥算法sha256(contentByteArray)。
contentByteArray=deviceSecret.getByte()+0x2c+hexToByte(random)

cipherType=6 BLE一型一密：密钥算法sha256(contentByteArray)。
contentByteArray=productSecret.getByte()+0x2c+hexToByte(mac)+0x2c+hexToByte(random)

cipherType=7 BLE一机一密：密钥算法sha256(contentByteArray)
contentByteArray=deviceSecret.getByte()+0x2c+hexToByte(mac)+0x2c+hexToByte(random)
```

一型一密和一机一密区别如下。

- 一型一密：是指使用产品的秘钥，即productSecret作为参数来生成配网时本地通信的加密秘钥，此时仅需提供productKey即可。

- 一机一密：是指使用设备的秘钥，即deviceSecret作为参数来生成配网时本地通信的加密秘钥，此时必须同时提供productKey和deviceName。

返回参数

名称	类型	描述
secret	Integer	用于本地配网时通信的加密秘钥

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "deviceInfoForCipher": {
      "deviceName": "xxxx",
      "productKey": "xxxx",
      "productId": "001e7",
      "cipherType": 3,
      "random": "00000000000000000000000000000000",
      "params": {
        "mac":"xxx"
      }
    }
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "xxxx",
  "message": "success"
}
```

网关接入子设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/gateway/permit	1.0.2	允许网关处理子设备接入的请求	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	网关设备iotId
productKey	String	是	允许接入网关的子设备产品标识符
time	Integer	是	• 0：网关一直允许添加子设备 • 0～65535：网关允许添加子设备的时间长度，单位为秒 • 65535：网关不允许添加子设备

返回参数

无业务返回参数

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.2",
      "iotToken": "token"
   },
   "params": {
      "iotId": "xxxxxxxxxxxxxx",
      "productKey": "xxxxxxxxx",
      "time": 60
   }
}
```

- 正常返回示例

```
{
   "id":"xxx",
   "code":200,
   "message":"success",
   "data":null
}
```

解除子设备网络拓扑

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/topo/remove	1.0.2	解除子设备的网络拓扑	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备iotId
subIotid	String	是	子设备iotId

返回参数

无业务返回参数

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "xxxxxxxxxxxxxx",
    "subIotId": "xxxxxxxxx"
  }
}
```

- 正常返回示例

```
{
  "id":"xxx",
  "code":200,
  "message":"success",
  "data":null
}
```

Wi-Fi设备绑定

一个设备最多被100个用户绑定，一个用户最多能绑定1000个设备。

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/enrollee/user/bind	1.0.7	用于Wi-Fi设备的绑定操作	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
productKey	String	是	设备所属产品productKey
deviceName	String	是	设备deviceName
token	String	是	设备上报的token

返回参数

名称	类型	描述
iotId	String	已绑定设备iotId

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.7"
  },
  "params":{
    "homeId":"50f5op155xxxxa4f2f28104512",
    "productKey":"xxx",
    "deviceName":"xxx",
    "token":"xxx"
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"CKuX18xxxx000100"
```

```
}
```

蓝牙设备绑定

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/ble/user/bind	1.0.7	用于蓝牙设备的绑定操作	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
deviceName	String	是	设备deviceName
productKey	String	是	待配网设备的productKey
sign	String	是	签名，详细见下方说明
signMethod	String	是	signMethod可支持：hmacSha1、hmacSha256、hmacMd5、sha256
signParams	JSON	是	加签扩展参数

说明：

蓝牙设备进行身份认证时，蓝牙传的不是productkey而是productId。验签是将“productKey + deviceName + map”里的参数，按照字典进行升序排序，再组成kv串即为待加密的签名

签名方法支持（忽略大小写）hmacSha1、hmacSha256、hmacMd5，签名规则如下：

- hmac_md5(deviceSecret, clientId***deviceName***productKey***timestamp***)
- hmac_sha1(deviceSecret, clientId***deviceName***productKey***timestamp***)
- hmac_sha256(deviceSecret, clientId***deviceName***productKey***timestamp***)
- sha256(clientId***deviceName***deviceSecret***productKey***timestamp***)

返回参数

名称	类型	描述
data	String	已配网设备的iotId

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.7"
  },
  "params":{
    "homeId":"50f5op1xxxxf2f28104512",
    "productKey":"xxx",
    "deviceName":"xxx",
    "sign":"xxxxxxx",
    "signMethod":"hmacSha1",
    "signParams":{
      "mac":"xxx"
    }
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"CKuX18xxxx000100"
}
```

GPRS设备绑定

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/gprs/user/bind	1.0.7	用于gprs设备的绑定操作	是，客户端SDK需启用身份

对于GPRS设备的绑定需要在设备上线的10分钟内进行，否则将会绑定失败。

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识

参数	类型	必填	描述
productKey	String	是	设备所属产品productKey
deviceName	String	是	设备deviceName

返回参数

名称	类型	描述
iotId	String	已绑定设备iotId

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.7"
  },
  "params":{
    "homeId":"50f5op15xxxxf2f28104512",
    "productKey":"xxx",
    "deviceName":"xxx"
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"CKuX1xxxx7Q000100"
}
```

子设备绑定

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/subdevice/bind	1.0.7	接入网关的子设备绑定	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
homeId	String	是	家的id，家的唯一标识
productKey	String	是	设备所属产品productKey
deviceName	String	是	设备deviceName

返回参数

名称	类型	描述
data	String	已绑定设备iotId

示例

- 请求示例

```
{
  "id":"xxx",
  "version":"1.0.0",
  "request":{
    "language":"zh-CN",
    "appKey":"xxx",
    "iotToken":"xxx",
    "apiVer":"1.0.7"
  },
  "params":{
    "homeId":"50f5op1xxxxf2f28104512",
    "productKey":"xxx",
    "deviceName":"xxx"
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "message":"success",
  "localizedMsg":"",
  "data":"CKuX1xxxxQ000100"
}
```

解绑子设备

定义描述

path	版本	描述	是否需要用户身份的鉴权
/awss/subdevice/unbind	1.0.2	解绑当前用户和指定子设备。如果当前用户是子设备的管理员，则同时自动解除当前子设备和网关的网络拓扑关系	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
productKey	String	是	子设备所属产品productKey
deviceName	String	是	子设备deviceName

返回参数

无业务返回参数

示例

- 请求示例

```
{
  "request": {
    "iotToken": "109049c80bcde4c06b15f6f62e29a3ba",
    "apiVer": "1.0.2"
  },
  "id": 1508212818676,
  "params": {
    "productKey": "xxx",
    "deviceName": "xxx"
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "id":"xxx",
  "code":200,
  "message":"success",
  "data":null
```

```
}
```

3.8.11 场景服务2.0

提供一组客户端用户创建和配置自动化场景的接口，可以实现触发某些条件时，执行某些任务的功能，如当到达某个时间时设备自动开启等功能。

> **说明：**
> 场景服务2.0已升级，新增了家、空间、群组控制等功能。若您无需使用这些功能，请查看历史版本场景服务2.0。

场景规则说明

- IFTTT规则
- CA规则
- Trigger组件
- Condition组件
- Action组件
- CaCondition组件

场景的创建、编辑、删除、查询

- 创建场景/自动化
- 更新场景/自动化
- 删除场景/自动化
- 获取场景/自动化详情
- 批量获取场景/自动化详情
- 查询场景或自动化列表
- 同时查询场景和自动化列表
- 改变场景列表中表项的排序

触发条件和执行任务

- 获取设备上支持trigger/condition/action配置的功能属性列表
- 获取当前用户下支持trigger/condition/action配置的设备列表
- 获取设备的trigger或condition或action功能列表与tsl定义
- 向手机推送消息action

场景自动化的执行

- 执行场景

- 场景生效与失效
- 获取场景日志列表
- 获取失败日志详情

定时场景

- 创建预约定时场景
- 更新预约定时场景
- 删除预约定时场景
- 获取预约定时场景详情
- 查询预约定时场景列表

App小组件

- 智能小组件查询场景
- 智能小组件添加场景

场景规则说明

场景规则用于描述动作由谁触发（Trigger），需要满足哪些条件（Condition），触发后要执行的动作（Action）。目前支持两种规则形式：IFTTT（If This Than That）和CA（Condition&Action）。其中IFTTT支持Trigger、Condition、Action节点；CA是对IFTTT的简化，将Trigger与Condition合并在一起作为“条件”。

IFTTT规则

以下是一个典型的场景规则示例，表示当设备sensor_01的**someone_exist**属性值等于**1**时场景将被触发（Trigger），此时会检查当前时间是否处于6:00至22:00之间（Condition），如果是，则把设备light 的开关属性设置为**on**（Action）。

```
{
  "type":"IFTTT",
  "trigger":{
    "uri":"trigger/device/property",
    "params":{
      "iotId":"sensor_01_iotId",
      "propertyName":"someone_exist",
      "compareType":"==",
      "compareValue":1
    }
  },
  "condition":{
    "uri":"logical/and",
    "items":[
      {
        "params":{
          "cron":"0-0 6-22 * * 1,2,3",
          "cronType":"linux",
          "timezoneID":"Shanghai"
```

```
          },
          "uri":"condition/timeRange"
        }
      ]
    },
    "action":[
      {
        "uri":"action/device/setProperty",
        "params":{
          "iotId":"light_iotId",
          "propertyName":"onOff",
          "propertyValue":"on"
        }
      }
    ]
}
```

以上例子展示了 IFTTT 类型的场景规则，可以看到规则中包含了四个节点， 分别是：

名称	描述
type	表示场景规则的类型，可取值为 IFTTT、CA。
trigger	表示场景的触发器，例如时间点、设备属性。一个场景中可以有多个触发器，彼此之间是“或”的关系。场景中也可以没有触发器，这时可以通过接口调用来手动触发场景。
condition	表示场景被触发后的过滤条件，例如时间段限制、设备属性限制。一个场景中可以有多个过滤条件，彼此之间是“与”的关系。场景中也可以没有过滤条件，这时场景触发后会不经过滤直接执行相应的动作。
action	表示场景被触发，且满足过滤条件时，所需执行的动作，例如设置设备属性、执行另一个场景。一个场景中可以有多个动作。场景中必须至少有一个动作。

以下例子展示了多个trigger多个condition的场景规则。 此时会使用logical/or节点封装多个trigger，使用logical/and节点封装多个condition。

```
{
  "type":"IFTTT",
  "trigger":{
    "uri":"logical/or",
    "items":[
      {
        "uri":"trigger/device/property",
        "params":{
          "productKey":"pk_sensor",
          "deviceName":"sensor_01",
          "propertyName":"someone_exist",
          "compareType":"==",
          "compareValue":1
        }
      },
      {
        "uri":"trigger/timer",
        "params":{
          "cron":"0 0 10 * *",
          "cronType":"linux"
```

```
          }
        }
      ]
    },
    "condition":{
      "uri":"logical/and",
      "items":[
        {
          "uri":"condition/device/property",
          "params":{
            "productKey":"pkxxxxon",
            "deviceName":"txxxxce",
            "propertyName":"soxxxxrty",
            "compareType":"<",
            "compareValue":10
          }
        },
        {
          "params":{
            "cron":"0-0 9-12 * * 1,2,3",
            "cronType":"linux",
            "timezoneID":"Asia/Shanghai"
          },
          "uri":"condition/timeRange"
        }
      ]
    },
    "action":[
      {
        "uri":"action/device/setProperty",
        "params":{
          "productKey":"pk_action",
          "deviceName":"test_device",
          "propertyName":"onOff",
          "propertyValue": "on"
        }
      },
      {
        "uri":"action/scene/trigger",
        "params":{
          "automationRuleId":"some_scene_id"
        }
      }
    ]
}
```

以上规则表示：每天十点或设备sensor_01的someone_exist属性值等于1时场景触发，如果设备test_device的some_property属性小于10 并且当前处于九点和十二点之间，则执行动作，将test_device的onOff属性设置为on，并触发另一个场景执行。

CA规则

除了IFTTT类型外，还支持CA类型的规则。CA规则是IFTTT规则的简化，规则中包含 type，mode，condition和 action节点。其中：

名称	描述
type	表示场景规则的类型，可取值为 IFTTT, CA

名称	描述
mode	CA规则的模式，可取值为 any 表示任何一个条件满足就执行动作，或 all 表示所有条件满足时才执行动作。
condition	场景的触发条件，例如时间点、设备属性、时间段等。一个场景可以有多个触发条件。
action	表示场景被触发时所需执行的动作，例如设置设备属性、执行另一个场景。一个场景中可以有多个动作。场景中必须至少有一个动作。

以下是CA类型场景规则的一个示例。

```
{
  "type":"CA",
  "mode":"all",
  "condition":[
    {
      "uri":"condition/device/property",
      "params":{
        "productKey":"pk_condition",
        "deviceName":"sensor_01",
        "propertyName":"some_property",
        "compareType":">",
        "compareValue":"300"
      }
    },
    {
      "uri":"condition/device/property",
      "params":{
        "productKey":"pk_condition",
        "deviceName":"sensor_02",
        "propertyName":"BarrierState",
        "compareType":"in",
        "compareValue":[
          0,
          1,
          2
        ]
      }
    },
    {
      "uri":"condition/timer",
      "params":{
        "cron":"0 05 10 * *",
        "cronType":"linux"
      }
    }
  ],
  "action":[
    {
      "uri":"action/device/setStatus",
      "params":{
        "productKey":"pk_action",
        "deviceName":"test_device",
        "propertyName":"LightStatus",
        "propertyValue": 1
      }
    }
  ]
```

```
}
```

以上规则表示： 当设备 sensor_01 的 some_property 属性大于 300、设备 sensor_02 的 BarrierState 属性取值在 0,1,2 之间、时间点在 10:05 这几个条件全部满足时，将设备 test_device 的 LightStatus 属性设置为 1。

Trigger组件

目前支持的Trigger组件包括：

url	描述
trigger/timer	定时触发
trigger/device/property	设备属性触发
trigger/device/event	设备事件触发

trigger/timer 参数描述

参数	类型	描述	参考
cron	String	定时表达式	cron表达式格式：http://crontab.org/
cronType	String	表达式类型，可取值为linux, quartz_cron linux：表示crontab类型，5位，不能设置年 quartz_cron：quartz类型cron,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	cron表达式格式：http://crontab.org/
timezoneID	String	时区ID，可空，表示将按照哪个时区来执行定时表达式。如果不设置，默认为智能生活平台云端服务所在区域，建议传入该参数，明确指定时区	示例：Asia/Shanghai

Corn表达式类型与范例

表达式类型	表达式结构	表达式字段	表达式示范
linux	${minute} ${hour} ${day of month} ${month} ${day of week}	• minute 0-59 • hour 0-23 • day of month 0-31 • month 0-12 • day of week 0-7	1 3 * * * **说明：** 每天的03:01触发
quartz_cron	${second} ${minute} ${hour} ${day of month} ${month} ${day of week} ${year} **说明：** 不允许使用无法抵达的过去时间	• second 必须为 0 • minute 0-59 • hour 0-23 • day of month 0-31 • month 0-12 • day of week 0-7 • year（可省略，省略则表示每年会触发）	0 * 14 * * ? 2019 **说明：** 在2019年每天下午2点到下午2:59期间的每1分钟触发

trigger/timer 示例

```
{
   "uri":"trigger/timer",
   "params":{
      "cron":"* * * * *",
      "cronType":"linux",
      "timezoneID":"Asia/Shanghai"
   }
}
```

trigger/device/property 参数描述

参数	类型	描述
productKey	String	设备产品键
deviceName	String	设备名称
propertyName	String	待比较的设备属性名
compareType	String	比较类型，如>、<、>=、==、<=、!=、in、like等
compareValue	Object	比较的值

trigger/device/property示例

```
{
   "uri":"trigger/device/property",
   "params":{
```

```
      "productKey":"test_pk",
      "deviceName":"test_dn",
      "propertyName":"temp",
      "compareType":">",
      "compareValue":30
   }
}
```

trigger/device/event参数描述

参数	类型	描述
productKey	String	设备产品键
deviceName	String	设备名称
eventCode	String	设备事件Code，可为空
propertyName	String	设备属性名称
compareType	String	比较类型，如>、<、>=、==、<=、!=、in、like等
compareValue	Object	比较的值

trigger/device/event示例

```
{
   "uri":"trigger/device/event",
   "params":{
      "productKey":"test_pk",
      "deviceName":"test_dn",
      "eventCode":"temp_warning",
      "propertyName":"temp",
      "compareType":">",
      "compareValue":30
   }
}
```

Condition组件

目前支持的condition组件包括：

url	描述逻辑
condition/timeRange	比较当前时间是否在一个区间内
condition/device/property	设备属性过滤

condition/timeRange 参数描述

参数	类型	描述
cron	String	20-25 11-23 * * 1,2,3 表示每周一、二、三的11:20到23:25之间允许执行
cronType	String	cron表达式类型，取值是linux
timezoneID	String	示例：Asia/Shanghai

condition/timeRange示例

```
{
   "params":{
      "cron":"0-0 9-12 * * 1,2,3",
      "cronType":"linux",
      "timezoneID":"Asia/Shanghai"
   },
   "uri":"condition/timeRange"
}
```

condition/device/property参数描述

参数	类型	描述
productKey	String	设备产品键
deviceName	String	设备名称
propertyName	String	设备属性
compareType	String	比较类型，如>、<、>=、==、<=、!=、in、like等
compareValue	Object	比较值

condition/device/property 示例

```
{
   "uri":"condition/device/property",
   "params":{
      "productKey":"test_product",
      "deviceName":"test_device",
      "propertyName":"temp",
      "compareType":">",
      "compareValue":30
   }
}
```

Action组件

目前支持的action组件包括：

url	描述逻辑
action/device/setProperty	设置设备属性
action/device/invokeService	调用设备服务
action/scene/trigger	触发另一个场景，如果被触发的场景包含condition，并且 condition没有被满足，那么被触发场景的action不会被执行。
action/automation/setSwitch	启用或停用另一个场景设置

action/scene/trigger 参数描述

参数	类型	描述	示例	
sceneId	String	场景id	无	

action/scene/trigger 示例

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

action/device/invokeService 参数描述

参数	类型	描述	示例	
iotId	String	设备Id	无	
serviceName	String	服务名称	无	
serviceArgs	JSON	服务参数	{"warningStatus":"off","level":"init"}	

action/device/invokeService 示例

```
{
   "uri":"action/device/invokeService",
   "params":{
      "iotId":"xxxx",
      "serviceName":"clearWarningStatus",
      "serviceArgs":{
         "warningStatus":"off",
         "level":"init"
      }
   }
}
```

action/device/setProperty 参数描述

参数	类型	描述	示例
iotId	String	设备id	无
propertyName	String	要设置的属性的名称	PowerSwitch
propertyValue	Object	要设置的属性的值	1

action/device/setProperty 示例

```
{
   "uri":"action/device/setProperty",
   "params":{
      "iotId":"xxxx",
      "propertyName":"PowerSwitch",
      "propertyValue":1
   }
}
```

action/automation/setSwitch 参数描述

参数	类型	描述	示例
automationRuleId	String	要启用或停用的场景id	无
switchStatus	Integer	0 表示停用，1 表示启用	1

action/automation/setSwitch 示例

```
{
   "uri":"action/automation/setSwitch",
   "params":{
      "automationRuleId":"xxxx",
      "switchStatus":0
   }
}
```

CaCondition组件

目前支持的caCondition组件包括：

url	描述逻辑
condition/timer	定时器，作用和参数与IFTTT类型下的trigger/timer组件相同

url	描述逻辑
condition/timeRange	时间段，作用与IFTTT类型下的condition/timeRange相同 只能用于all模式；any模式下将被忽略 **说明：** CA 类型下的condition/timeRange节点与IFTTT类型下的 condition/timeRange 节点参数形式不同。
condition/device/property	设备属性，作用和参数与IFTTT类型下的trigger/device/property 相同

condition/timeRange 参数描述

参数	类型	描述
format	String	时间的格式，如HH:mm
timezoneID	String	时区ID，可空，表示将按照哪个时区来进行时间段判断。如果不设置，默认为智能生活平台云端服务所在区域，建议传入该参数，明确指定时区。
beginDate	String	开始时间，需要符合format指定的格式，例如format为HH:mm那么 beginDate可以传入12:30
endDate	String	结束时间，需要符合format指定的格式（例如format为HH:mm），那么endDate可以传入13:30。 **说明：** 如果 beginDate大于endDate，那么将视为跨点，例如beginDate为22:00，endDate为6:00，则视为晚上22:00到第二天6:00。
repeat	String	星期重复，是逗号分隔的1~7数字构成的字符串。可空，不传表示不重复。如1,2,3 表示在星期一、星期二、星期三重复

condition/timeRange format 取值描述

取值	说明
mm:ss	分秒
HH:mm	时分
HH:mm:ss	时分秒
dd HH	日时
dd HH:mm	日时分
dd HH:mm:ss	日时分秒
MM	月
MM-dd	月日
MM-dd HH	月日时
MM-dd HH:mm	月日时分
MM-dd HH:mm:ss	月日时分秒
yyyy-MM-dd	年月日
yyyy-MM-dd HH:mm:ss	年月日时分秒

condition/timeRange 示例

```
{
   "uri":"condition/timeRange",
   "params":{
      "format":"HH:mm",
      "beginDate":"8:30",
      "endDate":"23:00",
      "repeat":"1,2,3,4,5"
   }
}
```

规则限制

场景规则有如下限制：

- Trigger组件可以没有，此时场景不会自动触发，可以调用 执行场景 接口来手动触发场景。例如，只设置了 condition/timeRange，但没有设置trigger，那么场景不会自动触发。
- Condition组件可以没有，此时Trigger触发会直接执行Action。
- 不能没有Action组件。
- 一个用户可以创建200个场景。
- 一个规则内最多可以有10个Trigger。
- 一个规则内最多可以有5个Condition。
- 一个规则内最多可以有10个Action。

- 一个设备最多可以作为20个场景的Action。
- 一个规则内最多只能有一个trigger/timer组件。
- 一个规则内最多只能有一个condition/timeRange组件。
- CA类型规则中，只有all模式支持condition/timeRange组件，any模式下将忽略该组件。

创建场景

定义描述

path	版本	描述	是否需要登录
/scene/create	1.0.5	创建场景	是

请求参数

参数	类型	必填	描述
groupId	String	是	用于对场景进行分类 • 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景
enable	Boolean	是	是否在创建时启用场景，true表示启用，false表示停用
name	String	是	用户给场景起的名称
icon	String	是	场景图标的下载链接
iconColor	String	是	场景图标色，例如#FFFFFF
description	String	否	对场景的描述
triggers	Object	否	IFTTT 规则的trigger对象
conditions	Object	否	IFTTT规则的condition对象
caConditions	JSON Array	否	CA规则的condition对象
actions	JSON Array	是	action对象

参数	类型	必填	描述
sceneType	String	否	场景规则类型：可取值为：IFTTT和CA；缺省值为IFTTT
mode	String	否	CA规则的模式 • all caConditions 中的条件全部满足时，才会执行action中的内容 • any caConditions 中的任何一个条件被满足，就会执行action中的内容

返回参数

参数	类型	描述
sceneId	String	创建的场景Id

示例

- 创建CA规则场景的请求示例

```
{
  "groupId":"1",
  "enable":true,
  "name":"ilopTestScene",
  "icon":"http://www.xxx.com/xxx.png",
  "iconColor":"#FFFFFF",
  "caConditions":[
    {
      "params":{
        "cron":"42 00 * * *",
        "cronType":"linux",
        "timezoneId":"Asia/Shanghai"
      },
      "uri":"condition/timer"
    },
    {
      "params":{
        "compareValue":0,
        "compareType":"==",
        "propertyName":"PowerSwitch",
        "productKey":"xxxx",
```

```
            "deviceName":"xxxx"
          },
          "uri":"condition/device/property"
        }
    ],
    "actions":[
        {
          "params":{
            "iotId":"xxxx",
            "propertyName":"PowerSwitch",
            "propertyValue":1
          },
          "uri":"action/device/setProperty"
        }
    ],
    "sceneType":"CA",
    "mode":"all"
}
```

- 创建IFTTT规则场景的请求示例（单Trigger，单Condition）

```
{
    "sceneType":"IFTTT",
    "enable":true,
    "groupId":"1",
    "name":"TestScene",
    "icon":"http://xxx.xxx.com/xxx.png",
    "iconColor":"#FFFFFF",
    "triggers":{
        "params":{
          "cron":"42 00 * * *",
          "cronType":"linux",
          "timezoneId":"Asia/Shanghai"
        },
        "uri":"trigger/timer"
    },
    "conditions":{
        "uri":"logical/and",
        "items":[
            {
              "params":{
                "cron":"20-25 11-23 * * 1,2,3",
                "cronType":"linux",
                "timezoneID":"Asia/Shanghai"
              },
              "uri":"condition/timeRange"
            }
        ]
    },
    "actions":[
        {
          "params":{
            "iotId":"xxxx",
            "propertyName":"PowerSwitch",
            "propertyValue":1
          },
          "uri":"action/device/setProperty"
        }
    ]
```

```
}
```

- 创建IFTTT规则场景的请求示例（多Trigger，多Condition）

```
{
  "sceneType":"IFTTT",
  "enable":true,
  "groupId":"1",
  "name":"testScene",
  "icon":"http://xxx.xxx.com/xxx.png",
  "iconColor":"#FFFFFF",
  "triggers":{
    "uri":"logical/or",
    "items":[
      {
        "params":{
          "cron":"42 00 * * *",
          "cronType":"linux",
          "timezoneId":"Asia/Shanghai"
        },
        "uri":"trigger/timer"
      },
      {
        "params":{
          "iotId":"testIotId",
          "propertyName":"PowerSwitch",
          "compareType":"==",
          "compareValue":0
        },
        "uri":"trigger/device/property"
      }
    ]
  },
  "conditions":{
    "uri":"logical/and",
    "items":[
      {
        "params":{
          "cron":"20-25 11-23 * * 1,2,3",
          "cronType":"linux",
          "timezoneID":"Asia/Shanghai"
        },
        "uri":"condition/timeRange"
      },
      {
        "params":{
          "iotId":"testIotId",
          "propertyName":"PowerSwitch",
          "compareType":"==",
          "compareValue":0
        },
        "uri":"condition/device/property"
      }
    ]
  },
  "actions":[
    {
      "params":{
        "iotId":"testIotId",
        "propertyName":"PowerSwitch",
        "propertyValue":1
      },
      "uri":"action/device/setProperty"
```

```
      }
    ]
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "sceneId",
  "message": "success"
}
```

更新场景

定义描述

path	版本	描述	是否需要登录
/scene/update	1.0.5	更新场景	是

请求参数

参数	类型	必填	描述
groupId	String	是	用于对场景进行分类 • 0表示手动场景，只包含action场景 • 1表示自动场景，包含Trigger和Condition的场景 **说明：** 在创建场景时指定groupId，场景创建后groupId不可以修改，即不能通过修改groupId的方式，将手动场景变更为自动场景。
sceneId	String	是	场景Id
enable	Boolean	是	true表示启用场景；false表示停用场景
name	String	是	场景的名称
icon	String	是	场景图标URL
iconColor	String	是	场景图标色

参数	类型	必填	描述
description	String	否	对场景的描述
triggers	Object	否	IFTTT规则的trigger对象
conditions	Object	否	IFTTT规则的condition对象
caConditions	Array	否	CA 规则的condition对象
actions	Array	是	action对象
sceneType	String	否	场景规则类型，可取值为IFTTT和CA，缺省值为IFTTT
mode	String	否	CA规则的模式，可取值为all和any

返回参数

参数	类型	描述
sceneId	String	更新的场景Id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
  "enable" : true,
  "iconColor" : "#DE4545",
  "reBind" : true,
  "mode" : "any",
  "caConditions" : [
   {
    "uri" : "condition\/timer",
    "params" : {
     "cron" : "33 10 * * 1,2,3,4",
     "cronType" : "linux"
    }
   }
  ],
  "sceneType" : "CA",
  "icon" : "http:\\xxx.xxx.com/xxx.png",
  "sceneId" : "2420d2xxxxf93c54fc4e",
  "groupId" : "1",
```

```
    "name" : "10:27 周一 周二 - 电源开关 开启",
    "actions" : [
     {
      "uri" : "action\/device\/setProperty",
      "params" : {
       "iotId" : "uumR5xxxx00101",
       "propertyName" : "PowerSwitch",
       "propertyValue" : 1
      }
     }
    ]
   }
}
```

- 正常返回示例

```
{
    "code": 200,
    "data": "bdc60bexxxxd617697",
    "message": "success"
}
```

删除场景

定义描述

path	版本	描述	是否需要登录
/scene/delete	1.0.2	删除场景	是

请求参数

参数	类型	必填	描述
sceneId	String	是	场景Id

返回参数

参数	类型	描述
sceneId	String	删除的场景Id

示例

- 请求示例

```
{
    "id": "1509086454180",
    "version": "1.0",
    "request": {
        "apiVer": "1.0.1",
        "iotToken": "token"
    },
    "params": {
        "sceneId": "0c965xxxx5306a6657",
```

```
    "groupId": null
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "0c9653xxxx06a6657",
  "message": "success"
}
```

查询场景或自动化列表

定义描述

path	版本	描述	是否需要登录
/scene/list/get	1.0.5	查询场景列表	是

请求参数

参数	类型	必填	描述
groupId	String	是	0=场景 1=自动化
pageNo	Integer	是	页序号，不小于1
pageSize	Integer	是	每页数量，不小于1，不大于50

返回参数

参数	子参数	类型	描述
total		Integer	总条目数
pageNo		Integer	从1开始的页序号
pageSize		Integer	每页数量
scenes		Array	返回当前页条目
	scenes.id	String	场景id
	scene.status	Integer	场景运行时状态：1-场景上线；2-场景下线
	scene.enable	Boolean	场景开关（App上的开关）：true-场景置为打开；false-场景置为关闭

参数	子参数	类型	描述
	scene.icon	String	场景图标连接
	scene.iconColor	String	场景图标颜色
	scene.name	String	场景名称
	scene.description	String	场景描述
	scene.valid	Boolean	是否有效
	scene.groupId	String	类型：null-老版场景；0-场景；1-自动化

示例

- 请求示例

```
{
 "id": "1509086454180",
 "version": "1.0",
 "request": {
  "apiVer": "1.0.1",
  "iotToken": "token"
 },
 "params": {
  "groupId": "1",
  "pageSize": 20,
  "pageNo": 1,
 }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
  "pageNo": 1,
  "pageSize": 20,
  "scenes": [{
   "description": "电热毯-hy-信噪比--127 ",
   "enable": true,
   "icon": "https://g.aliplus.com/scene_icons/default.png",
   "id": "260de82dxxxx691be22",
   "name": "电热毯",
   "status": 2,
   "iconColor":"#19BBFF",
   "valid":false,
   "groupId":"1"

  }, {
   "description": "light-hy-LightSwitch-1, shuijin-hy-WiFI_SNR--127 ",
   "enable": true,
   "icon": "https://g.aliplus.com/scene_icons/default.png",
   "id": "45dc6fcxxxx6974e5f7",
   "name": "凉宫",
   "status": 1,
    "iconColor":"#19BBFF",
   "valid":true,
```

```
    "groupId":"1"
   }, {
    "description": "fengsan-hy-PowerSwitch-关闭, light-hy-LightSwitch-1 ",
    "enable": true,
    "icon": "https://g.aliplus.com/scene_icons/default.png",
    "id": "3ce636xxxx1367f2dc88",
    "name": "风扇灯0522",
    "status": 1,
     "iconColor":"#19BBFF",
    "valid":false,
    "groupId":"1"
   }, {
    "description": "light-hy-LightSwitch-1 ",
    "enable": true,
    "icon": "https://g.aliplus.com/scene_icons/default.png",
    "id": "5b4bdexxxx2612f81",
    "name": "灯0522",
    "status": 2,
     "iconColor":"#19BBFF",
    "valid":false,
    "groupId":"1"
   }],
   "total": 4
  },
  "message": "success"
}
```


同时查询场景和自动化列表

定义描述

path	版本	描述	是否需要登录
/scene/list/all	1.0.0	查询我的智能页面中的所有场景信息，包括场景和自动化	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
pageNo	Integer	否	当前页号，缺省默认值1
pageSize	Integer	是	页大小，大于等于1，最大为30

返回参数

参数	二级子参数	三级子参数	四级子参数	类型	描述
data				String	场景及自动化列表
	scene0			String	场景列表信息

参数	二级子参数	三级子参数	四级子参数	类型	描述
		total		Integer	场景总数
		pageNo		Integer	返回当前的页号
		pageSize		Integer	返回的页大小
		scenes		String	场景列表
			id	String	场景id
			name	String	场景名称
			description	String	场景描述
			enable	Boolean	场景开关（App上的开关）：true-场景置为打开；false-场景置为关闭
			icon	String	场景图标链接
			iconColor	String	场景图标颜色
			groupId	String	分类，0-表示场景；1-表示自动化
			valid	Boolean	场景是否有效，当场景中的设备被解绑时，自动化会变为无效状态，此时该字段为true
	scene1			String	自动化列表信息
		total		Integer	自动化总数
		pageNo		Integer	返回的当前页号
		pageSize		Integer	返回的页大小
		scenes		String	自动化列表
			id	String	自动化id
			name	String	自动化名称

参数	二级子参数	三级子参数	四级子参数	类型	描述
			description	String	自动化描述
			enable	Boolean	自动化开关（App上的开关）：true-场景置为打开；false-场景置为关闭
			icon	String	自动化图标
			iconColor	String	自动化图标颜色
			groupId	String	分类，0-表示场景；1-表示自动化
			valid	Boolean	自动化是否有效，当自动化中的设备被解绑时，自动化会变为无效状态，此时该字段为true

示例

- 请求示例

```
{
 "id": "1509086454180",
 "version": "1.0",
 "request": {
  "apiVer": "1.0.0",
  "iotToken": "token"
 },
 "params": {
 "pageNum": 1,
  "pageSize": 10
 }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
   "scene0": {
      "total": 1,
      "scenes": [
        {
          "id": "",
```

```
            "name": "",
            "description": "",
            "enable": true,
            "icon": "",
            "iconColor": "",
            "groupId": "0",
            "valid": true
          }
        ],
        "pageNo": 1,
        "pageSize": 10
      },
      "scene1": {
        "total": 0,
        "scenes": [],
        "pageNo": 1,
        "pageSize": 10
      }
    },
    "message": "success"
}
```


改变场景列表中表项的排序

定义描述

path	版本	描述	是否需要登录
/scene/list/reorder	1.0.5	改变场景列表中表项的排序	是

请求参数

参数	子参数	类型	必填	描述
groupId		String	是	0-场景；1-自动化
newOrders		Array	是	序号变化的场景数组
	sceneId	String	是	场景id
	fromOrder	Integer	是	顺序变化前的序号
	toOrder	Integer	是	新的序号

返回参数

参数	类型	描述
code	Integer	操作成功状态

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
      "groupId": null,
      "newOrders": [{
         "fromOrder": 1,
         "sceneId": "a596dexxxx9685d25",
         "toOrder": 2
      }, {
         "fromOrder": 2,
         "sceneId": "c1a8de4xxxxb473c3",
         "toOrder": 1
      }]
   }
}
```

- 正常返回示例

```
{
   "code": 200,
   "message": "success"
}
```

执行场景

定义描述

path	版本	描述	是否需要登录
/scene/fire	1.0.2	执行场景	是

请求参数

参数	类型	必填	描述
sceneId	String	是	待触发的场景id

返回参数

参数	类型	描述
sceneId	String	触发的场景id

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
      "groupId": null,
      "sceneId": "debf2dxxxx51496c2"
   }
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": "debf2dxxxx51496c2",
   "message": "success"
}
```

获取场景日志列表

定义描述

path	版本	描述	是否需要登录
/scene/log/list/get	1.0.2	获取场景日志列表	是

请求参数

参数	类型	必填	描述
pageNo	Integer	是	请求的页数，不小于1
pageSize	Integer	是	一次拉取的条目数
nowTime	Long	是	查询日志的起始时间（单位毫秒），参照场景日志查询的方式

如果页面要查看T1之后的某个时间点开始的Log，需要重新指定此时的时间点T2，并且把已查询到的结果全部删除掉，令pageNo=1，重新开始查询。[T1-15*3600*24, T1] 这个时间段之间的日志场景日志查询的方式

返回参数

参数	类型	描述
pageNo	Integer	请求的页数，不小于1

参数	类型	描述
pageSize	Integer	一次拉取的条目数
total	Integer	总数
logs	Array	日志列表，按时间从大到小返回
logs.time	Long	日志时间，时间戳，单位毫秒
logs.icon	String	场景图标
logs.sceneId	String	场景id
logs.id	String	日志id
logs.sceneName	String	场景名称
logs.result	Integer	执行结果：0-失败；1-成功

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "pageNo": 1,
    "pageSize": 5,
    "groupId": null,
    "nowTime": 1527129084824
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "logs": [{
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "0bc19f5d15xxxx56576d52bf",
      "result": 0,
      "sceneId": "d5edf3a8xxxx91b15e2028",
      "sceneName": "t时间设备",
      "time": 1527091200000
    }, {
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "0bc19f5xxxx056577d52bf",
      "result": 0,
      "sceneId": "81eb22a2991xxxx2d0993ce",
      "sceneName": "t时间",
      "time": 1527091200000
    }, {
```

```
            "icon": "https://g.aliplus.com/scene_icons/default.png",
            "id": "0bc19f5d15xxxx75d52bf",
            "result": 0,
            "sceneId": "3f522ebxxxx12336709",
            "sceneName": "测",
            "time": 1527091200000
        }, {
            "icon": "https://g.aliplus.com/scene_icons/default.png",
            "id": "0bc19fxxxx578d52bf",
            "result": 0,
            "sceneId": "32020xxxx757ee3",
            "sceneName": "t时间设备",
            "time": 1527091200000
        }, {
            "icon": "https://g.aliplus.com/scene_icons/default.png",
            "id": "0bc19f5xxxx14d52bf",
            "result": 1,
            "sceneId": "70d90fexxxx997188911",
            "sceneName": "灯",
            "time": 1527054987000
        }],
        "pageNo": 1,
        "pageSize": 5,
        "total": 62
    },
    "message": "success"
}
```


获取失败日志详情

定义描述

path	版本	描述	是否需要登录
/scene/failedlog/get	1.0.4	获取失败日志详情	是

请求参数

参数	类型	必填	描述
logId	String	是	待查询的日志id
sceneId	String	是	场景id
time	Long	是	/scene/log/list/get返回的logs.time，单位毫秒
pageNo	Integer	是	请求的页数，第一次为1
pageSize	Integer	是	一次拉取的条目数

返回参数

参数	类型	描述
total	Integer	总数
pageSize	Integer	一次拉取的条目数
pageNo	Integer	请求的页数，不小于1
details	Array	无
details.productKey	String	失败设备的详情列表
details.localizedProductName	String	本地化的产品信息
details.deviceName	String	设备的deviceName
details.aliasName	String	用户给设备起的名称
details.failedReason	String	错误信息，格式如果action是设置属性，则文案拼接规则是：{产品名称}-{属性名}-{属性值} 执行失败如果action是调用服务，则文案拼接规则是：{产品名称}-{服务名称} 执行失败
details.detail	String	下标注释的原因
details.icon	String	图标

示例

- 请求示例

```
{
  "id": "150xxxx4180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "pageNo": 1,
    "pageSize": 1000,
    "logId": "0bc19xxxx00056576d52bf",
    "sceneId": "d5edf3xxxx91b15e2028",
    "time": 1527091200000
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "details": [{
      "detail": "light-hy - NightLightSwitch - 关闭1 设置设备参数",
```

```
            "deviceName": "hyxxxxhy",
            "failedReason": "device in scene's action is unbind",
            "productKey": "b1xxxxrm"
        }],
        "pageNo": 1,
        "pageSize": 1000
    },
    "message": "success"
}
```


获取场景详情

定义描述

path	版本	描述	是否需要登录
/scene/info/get	1.0.5	获取场景详情	是

请求参数

参数	类型	必填	描述
sceneId	String	是	待查询的场景ID
groupId	String	否	场景类型： • 0：场景 • 1：自动化

返回参数

参数	类型	描述
sceneId	String	场景id
enable	Boolean	场景开关状态，取值如下： • true，打开状态，场景可以正常被触发执行 • false，关闭状态，即便达到触发条件，场景也不会执行
name	String	用户给场景起的名称
icon	String	场景图标
iconColor	String	场景图标色
description	String	对场景的描述
triggersJson	Object	IFTTT规则-Trigger内复杂对象的Json序列化，用于前端展示
conditions	Object	IFTTT规则的condition对象

参数	类型	描述
caConditionsJson	Array	Since场景2.0，CA规则 - Condition内复杂对象的Json 序列化，用于前端展示
actionsJson	String	Action内复杂对象的Json序列化，用于前端展示
sceneType	String	Since场景2.0，场景规则：IFTTT和CA（缺省值为IFTTT）
mode	String	Since 场景2.0，CA规则的模式：all和any
valid	Boolean	场景是否有效

场景生效与失效

定义描述

path	版本	描述	是否需要登录
/living/scene/switch	1.0.0	场景自动化开启/关闭	是

请求参数

参数	类型	必填	描述
sceneId	String	是	场景ID
enable	Boolean	是	场景开关状态，取值如下： • true，打开状态，场景可以正常被触发执行 • false，关闭状态，即便达到触发条件，场景也不会执行

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
```

```
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "iotToken": "token"
  },
  "params": {
    "sceneId": "0c96536xxxxc5306a6657",
    "enable": true
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success"
}
```

获取当前用户下支持trigger/condition/action配置的设备列表

定义描述

path	版本	描述	是否需要登录	说明
/scene/thing/list	1.0.2	查询获取支持T/C/A的设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证	包含了被分享的设备

请求参数

参数	类型	必填	描述
flowType	Integer	是	流程类型 0-trigger；1-condition；2-action
pageNum	Integer	是	分页页数，从1开始
pageSize	Integer	是	分页页面大小

返回参数

参数	类型	描述
pageNo	Integer	从1开始的页序号
pageSize	Integer	单页的item数量上限
total	Long	总记录数
data	List	设备列表

data 列表结构

参数	类型	描述
iotId	String	设备ID
deviceName	String	设备deviceName
productKey	String	产品productKey
nickName	String	设备昵称
image	String	品类图标URL

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "flowType": 0,
    "pageNum": 1,
    "pageSize": 20
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "pageNo": 1,
    "pageSize": 20,
    "total":1,
    "data": [{
      "iotId": "45dc6xxxxf6974e5f7",
      "deviceName": "xxxx",
      "productKey": "xxxx",
      "nickName": "测试设备",
      "image": "https://g.aliplus.com/scene_icons/default.png"
    }]
  },
  "message": "success"
}
```

获取设备上支持trigger/condition/action配置的功能属性列表

定义描述

path	版本	描述	是否需要登录
/iotid/scene/ability/list	1.0.2	查询设备支持T/C/A的功能列表信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
flowType	Integer	是	流程类型 0-trigger；1-condition；2-action

返回参数

参数	类型	描述
name	String	功能名称
identifier	String	功能标识符
categoryType	String	品类名称
type	Integer	功能类型 1:-属性；2-服务；3-事件

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "iotId": "45dc6fxxxx29f6974e5f7",
    "flowType": 0
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [{
        "name": "开关",
        "identifier": "switch",
        "categoryType": "xxxx",
        "type": 1
```

```
    }],
    "message": "success"
}
```

获取设备的trigger或condition或action功能列表与TSL定义

定义描述

path	版本	描述	是否需要登录
/iotid/scene/ability/tsl/list	1.0.2	查询设备支持T/C/A的功能列表信息以及对应设备的TSL信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
flowType	Integer	是	流程类型：0-trigger；1-condition；2-action

返回参数

参数	类型	描述
simplifyAbilityDTOs	List	功能定义列表
abilityDsl	JSONObject	产品功能TSL（参考）

simplifyAbilityDTOs结构

参数	类型	描述
name	String	功能名称
identifier	String	功能标识符
categoryType	String	品类名称
type	Integer	功能类型：1-属性；2-服务；3-事件

示例

- 请求示例

```
{
    "id": "1509086454180",
    "version": "1.0",
```

```
  "request": {
    "apiVer": "1.0.1",
    "iotToken": "token"
  },
  "params": {
    "iotId": "45dc6fxxxx6974e5f7",
    "flowType": 0
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
   "simplifyAbilityDTOs":[{
       "name": "开关",
       "identifier": "switch",
       "categoryType": "xxxx",
       "type": 1
    }],
   "abilityDsl": {}
  },
  "message": "success"
}
```

创建设备预约定时场景

定义描述

path	版本	描述	是否需要用户身份的鉴权
/scene/timing/create	1.0.2	创建设备的预约定时场景	是，客户端SDK需启用身份的鉴权，进行身份认证

说明:

预约定时场景与设备相关联，因此不能使用/scene/update和/living/scene/switch等接口来更新、启停。如有这方面需要，请使用/scene/timing/update接口。同时不支持手机消息推送action。

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
enable	Boolean	是	true打开 false关闭
name	String	是	用户给场景起的名称

参数	类型	必填	描述
icon	String	是	场景图标
description	String	否	对场景的描述
triggers	Object	否	trigger对象
conditions	Object	否	condition对象
actions	Array	是	action对象

说明:

action不能为空，trigger可以空，condition可以空。

返回参数

参数	类型	描述
sceneId	String	新建的场景Id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "actions": [{
      "params": {
        "compareValue": 1,
        "iotId": "zjoOvFxxxxc4dc00",
        "compareType": ">",
        "propertyName": "Qxxxxat",
        "propertyValue": 1
      },
      "uri": "action/device/setProperty"
    }],
    "iotId":"zjoOvFZxxxx4dc00",
    "enable": true,
    "icon": "https://g.aliplus.com/scene_icons/default.png",
    "name": "一样",
    "triggers": {
      "params": {},
      "items": [{
        "params": {
          "cron": "1 1 * * *",
          "cronType": "linux",
          "timezoneID": "Asia/Shanghai"
        },
        "uri": "trigger/timer"
      }],
```

```
        "uri": "logical/or"
      }
    }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "0c9653xxxx645c5306a6657",
  "message": "success"
}
```

更新设备预约定时场景

定义描述

path	版本	描述	是否需要用户身份的鉴权
/scene/timing/update	1.0.2	更新设备的预约定时场景	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
sceneId	String	是	场景ID
enable	Boolean	是	true打开 false关闭
name	String	是	用户给场景起的名称
icon	String	是	场景图标
description	String	否	对场景的描述
triggers	Object	否	trigger对象
conditions	Object	否	condition对象
actions	Array	是	action对象

说明：

action不能为空，trigger可以空，condition可以空。

返回参数

参数	类型	描述
sceneId	String	场景Id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "actions": [{
      "params": {
        "compareValue": 0,
        "iotId": "gaI0CRxxxx09b3700",
        "compareType": {
          "i18nKey": "scene.common.compareType.equal",
          "type": "=="
        },
        "propertyName": "Nigxxxxwitch",
        "propertyValue": 0
      },
      "uri": "action/device/setProperty",
      "status": 1
    }],
    "conditions": {
      "params": {},
      "items": [{
        "params": {
          "compareValue": 0,
          "iotId": "gaI0Cxxxx09b3700",
          "compareType": "==",
          "propertyName": "LightSwitch",
          "propertyValue": 0
        },
        "uri": "condition/device/property",
        "status": 1
      }],
      "uri": "logical/and"
    },
    "enable": true,
    "icon": "https://g.aliplus.com/scene_icons/default.png",
    "name": "",
    "sceneId": "bdc60xxxx7d617697",
    "iotId": "gaI0CRKxxxx09b3700",
    "triggers": {
      "params": {},
      "items": [{
        "params": {
          "cron": "0 0 * * *",
          "cronType": "linux",
          "timezoneID": "Asia/Shanghai"
        },
        "uri": "trigger/timer",
        "status": 1
      }],
```

```
        "uri": "logical/or"
      }
    }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "0c9653xxxx06a6657",
  "message": "success"
}
```

删除设备预约定时场景

定义描述

path	版本	描述	是否需要用户身份的鉴权
/scene/timing/delete	1.0.2	删除定时场景	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备Id
sceneId	String	是	场景Id

返回参数

参数	类型	描述
sceneId	String	删除的场景Id

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "sceneId": "0c96xxxx306a6657",
    "iotId": "gaI0Cxxxx109b3700"
  }
```

```
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "0c9653xxxx06a6657",
  "message": "success"
}
```

获取设备预约定时场景详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/scene/timing/info/get	1.0.2	获取定时场景的详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备Id
sceneId	String	是	待查询的场景Id

返回参数

参数	类型	描述
sceneId	String	场景id
enable	Boolean	场景开关：true-打开；false-关闭
status	Integer	场景运行时状态：1-运行；2-失效
name	String	用户给场景起的名称
icon	String	场景图标
triggers	Object	triggers对象的String。trigger对象定义在下面
conditions	Object	conditions对象的String
actions	Array	actions对象的String

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "sceneId": "260de8xxxx4691be22",
    "iotId": "gaI0Cxxxx9b3700"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "actions": [{
      "params": {
        "compareValue": -127,
        "localizedProductName": "电热毯-hy",
        "iotId": "SY7a6Ixxxx0c00",
        "compareType": "==",
        "propertyName": "WiFI_SNR",
        "localizedPropertyName": "信噪比",
        "deviceNickName": "电热毯-hy",
        "propertyValue": -127,
        "localizedCompareValueName": "-127",
        "productKey": "b1xxxxql",
        "propertyItems": {
          "WiFI_SNR": -127
        },
        "deviceName": "hyhyhy4"
      },
      "uri": "action/device/setProperty",
      "status": 0
    }],
    "enable": true,
    "icon": "https://g.aliplus.com/scene_icons/default.png",
    "name": "电热毯",
    "sceneId": "260de82dxxxx4691be22",
    "status": 2
  },
  "message": "success"
}
查询预约定时场景列表
```

查询设备预约定时场景列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/scene/timing/list/get	1.0.2	查询定时场景列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
pageNo	Integer	是	页序号，不小于1
pageSize	Integer	是	每页数量，不小于1，不大于50

返回参数

参数	子参数	类型	描述
total		Integer	总条目数
pageNo		Integer	从1开始的页序号
pageSize		Integer	每页数量
scenes		Array	返回当前页条目
	scenes.id	String	场景id
	scene.status	Integer	场景运行时状态：1-场景上线；2-场景下线
	scene.enable	Boolean	场景开关（App上的开关）：true-场景置为打开；false-场景置为关闭
	scene.icon	String	场景图标连接
	scene.name	String	场景名称
	scene.description	String	场景描述

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
```

```
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "pageSize": 20,
    "pageNo": 1,
    "iotId": "zjoOvFZxxxx0c4dc00"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "pageNo": 1,
    "pageSize": 20,
    "scenes": [{
      "description": "电热毯-hy-信噪比--127 ",
      "enable": true,
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "260de82xxxx691be22",
      "name": "电热毯",
      "status": 2
    }, {
      "description": "light-hy-LightSwitch-1, shuijin-hy-WiFI_SNR--127 ",
      "enable": true,
      "icon": "https://g.aliplus.com/scene_icons/default.png",
      "id": "45dc6fcxxxx6974e5f7",
      "name": "凉宫",
      "status": 1
    }],
    "total": 2
  },
  "message": "success"
}
```

批量获取场景详情

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/scene/batchget	1.0.0	批量请求场景信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
sceneIdList	list	是	待查询的场景Id列表

返回参数

参数	类型	描述
sceneId	String	场景id
enable	Boolean	场景开关：true-打开；false-关闭
name	String	用户给场景起的名称
icon	String	场景图标
triggers	String	triggers对象的String
conditions	String	conditions对象的String
actions	String	actions对象的String

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params":
 {"sceneIdList":["b8fe2xxxx26b76c31"]}
}
```

- 正常返回示例

```
{
  "code":200,
  "data":
  [
  {"sceneType":"IFTTT",
  "name":"主灯开关 开启 - 灯模式 mono - 开启",
  "icon":"https://img.alicdn.com/tfs/TB10G8xxxxUVXa-144-144.png",
  "sceneId":"b8fexxxx26b76c31",
  "sceneSwitch":1,
  "triggers":"{\"params\":{},\"items\":[{\"params\":{\"eventCode\":\"\",\"compareValue\":1,\"iotId\":\"z8nxxxx00101\",\"compareType\":\"==\",\"propertyName\":\"LightSwitch\",\"propertyValue\":1,\"productKey\":\"a1B0kMs35NW\",\"deviceName\":\"z8nxxxx6SFY\"},\"uri\":\"trigger/device/property\"}],\"uri\":\"logical/or\"}",
  "conditions":"{\"params\":{},\"items\":[{\"params\":{\"eventCode\":\"\",\"compareValue\":0,\"iotId\":\"z8nqxxxx000101\",\"compareType\":\"==\",\"propertyName\":\"LightMode\",\"propertyValue\":0,\"productKey\":\"a1B0kMs35NW\",\"deviceName\":\"z8nxxxx6SFY\"},\"uri\":\"condition/device/property\"}],\"uri\":\"logical/and\"}",
  "actions":"[{\"params\":{\"switchStatus\":1,\"automationRuleId\":\"50xxxx520ceed\"},\"uri\":\"action/automation/setSwitch\"}]"}
  ],
  "id":"1509086454180"
}
```

向手机推送消息action

定义描述

path	描述	是否需要用户身份的鉴权
/action/mq/send	向手机推送消息action	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
customData	CustomData	是	推送消息模型
customData.message	String	是	向手机推送消息的内容，最多60个字符
msgTag	String	是	推送消息模型，默认值：IlopBusiness_CustomMsg

返回参数

无

示例

- 请求示例

```
"actions": [
  {
    "uri":"action/mq/send",
    "params":{
      "customData":{
        "message":"电源开启了"
      },
      "msgTag":"IlopBusiness_CustomMsg"
    }
  }
]
```

- 正常返回示例

  无

智能小组件查询场景

定义描述

path	版本	描述	是否需要登录
/living/appwidget/list	1.0.0	智能小组件查询，用于APP的场景快捷触发小组件中查询组件信息	是

请求参数

无

返回参数

参数	类型	描述
id	Stirng	场景ID
name	Stirng	场景名称
description	Stirng	场景描述
enable	Boolean	场景是否启用
icon	String	场景图标
iconColor	String	场景图标色
valid	Boolean	场景是否有效

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "iotToken": "token"
  },
  "params": {
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success"
   "data":[{
    "id": "d5edf3xxxxb15e2028",
    "iconColor": "#19BBFF",
    "name": "向手机发送通知",
    "description": " 描述 ",
    "enable": true
  },
  {
```

```
      "id": "debf2dxxxx1496c2",
      "iconColor": "#FFBBFF",
      "name": "开灯",
      "description": "描述",
      "enable": false
   }
]
}
```

智能小组件添加场景

定义描述

path	版本	描述	是否需要登录
/living/appwidget/create	1.0.0	智能小组件添加，用于APP的场景快捷触发小组件中添加组件	是

请求参数

无

返回参数

参数	类型	描述
sceneIds	Array	场景ID列表，最多8个场景

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params": {
    "sceneIds":["d5edfxxxx5e2028","debfxxxx496c2"]
   }
}
```

- 正常返回示例

```
{
   "code": 200,
   "message": "success"
}
```

3.8.12 数据聚合查询服务

提供设备级别常见的上报数据记录（分属性、服务、事件）和聚合查询的接口，以及支持的上报记录和聚合数据种类。

查询数据

定义描述

path	版本	描述	是否需要用户身份的鉴权
/datacenter/data/aggregate/query	1.0.0	查询设备纬度的的记录/统计聚合数据	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数名	参数类型	详细描述
iotId	String	设备ID
cols	List	需要查询的列
tableName	String	表名称
expr		查询对象表达式对象
orderBy		排序设置 • col：列名 • type： - asc：升序 - desc：降序 示例：[{“col”：“id”，“type”：“asc”}]
pageNo	Integer	分页位置，最小1
pageSize	Integer	分页大小

表 3-2: 请求参数中expr字段描述

参数名	参数类型	详细描述
op	String	and/or
filters	List	多个过滤对象

参数名	参数类型	详细描述
exprs	List	多个查询表达式对象数组，exprs表示一个或多个expr（expr中可以嵌套expr，嵌套的expr个数不限）

表 3-3: expr中filters字段描述

参数名	参数类型	详细描述
col	String	匹配字段
comp	String	比较操作符 = != > < >= <= in not-in like not-like
val	Object	比较值

返回参数

参数名	参数类型	详细描述
code	INT	
data	JSONObject	自建表查询结果
message	String	错误信息提示，不通错误提示不同

示例

- 请求示例

```
{
  "iotId":"example_iot_id",
  "pageNo":1,
  "pageSize":24,
  "expr":{
    "op":"and",
    "filters":[
      {
        "val":"TargetHumidity",
        "comp":"=",
        "col":"attribute"
      },
      {
        "val":"2019-10-12 00:00:00",
        "comp":">=",
        "col":"date_time"
      },
      {
        "val":"2019-10-13 00:00:00",
        "comp":"<",
        "col":"date_time"
      }
```

```
      ]
   },
   "cols":[
      "sum_num",
      "avg_num",
      "date_time"
   ],
   "tableName":"ads_ilop_kv_state_hourly"
}
```

- 正常返回示例

```
{
   "code":200,
   "data":[
      {
         "sum_num":40,
         "date_time":"2019-10-12 00:00:00",
         "avg_num":20
      },
      {
         "sum_num":2260,
         "date_time":"2019-10-12 08:00:00",
         "avg_num":20
      },
      {
         "sum_num":80,
         "date_time":"2019-10-12 09:00:00",
         "avg_num":20
      }
   ],
   "localizedMsg":"",
   "message":""
}
```

接口支持的表和列信息

表 3-4: 按小时聚合统计属性/服务的数据（ads_ilop_kv_state_hourly）

小时聚合数据保存时间为7天

存储列名称	存储列数据类型	存储列数据描述
iot_id	String	设备唯一身份标识
attribute	String	产品功能定义Identifier
sum_num	Int	设备上报的属性/服务数据，按小时求和
avg_num	Int	设备上报的属性/服务数据，按小时求平均
date_time	TimeStamp	统计日期，格式为yyyy-mm-dd hh:MM:ss

表 3-5: 按天聚合统计属性/服务的数据（ads_ilop_kv_state_daily）

天聚合数据保存时间为30天

存储列名称	存储列数据类型	存储列数据描述
iot_id	String	设备唯一身份标识
attribute	String	产品功能定义Identifier
sum_num	Int	设备上报的属性/服务数据，按天求和
avg_num	Int	设备上报的属性/服务数据，按天求平均
date_time	TimeStamp	统计日期，格式为yyyy-mm-dd hh:MM:ss

表 3-6: 按月（自然月）聚合统计属性/服务的数据（ads_ilop_kv_state_monthly）

月聚合数据保存时间为365天

存储列名称	存储列数据类型	存储列数据描述
iot_id	String	设备唯一身份标识
attribute	String	产品功能定义Identifier
sum_num	Int	设备上报的属性/服务数据，按自然月求和
avg_num	Int	设备上报的属性/服务数据，按自然月求平均
date_time	TimeStamp	统计日期，格式为yyyy-mm-dd hh:MM:ss

表 3-7: 历史事件数据的记录（r_ilop_event_aggre_di）

设备所有上报的历史事件数据的记录集合，事件聚合数据保存时间为 30天

存储列名称	存储列数据类型	存储列数据描述
batch_id	String	事件上报request_id
iot_id	String	设备唯一身份标识
event_code	String	事件Identifier
data_value	String	设备上报事件原始内容
client_date	TimeStamp	设备端发送时间

存储列名称	存储列数据类型	存储列数据描述
service_date	TimeStamp	服务端处理时间

3.8.13 固件升级服务

平台提供一组OTA固件升级的接口，客户端App通过本组接口可以获取指定设备的固件信息以及固件升级状态，同时提供了固件升级确认接口（当升级需要用户许可时使用）。只有管理员用户才具有调用固件升级接口的权限（设备的被分享者无法调用该设备的固件升级接口，只能查询固件升级的进度）。

- 获取固件升级信息
- 获取设备固件
- 取消固件升级
- 获取正在升级的设备列表
- 上报固件升级进度
- 上报版本号
- 批量确认设备升级
- 获取待升级的设备列表

获取固件升级信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/progress/getByUser	1.0.2	获取某一个设备的固件升级信息，需要指定具体的固件版本号。如果存在相应的固件，返回当前固件的升级进度等信息。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id
version	String	是	对应的固件版本号

返回参数

参数	类型	必有	描述
iotId	String	是	设备id
step	Integer	是	升级进度或升级错误信息 • 负数时，表示升级错误信息 - -1：升级操作失败 - -2：固件文件下载失败 - -3：固件文件校验失败 - -4：固件文件烧写失败 • 非负数时，表示升级进度，单位为：百分比
desc	String	否	描述信息
success	Boolean	是	是否成功标记，当为true时表示固件升级操作成功，反之表示升级未完成，包括正在升级或升级失败，请优先选择upgradeStatus参数来判断固件升级状态
needConfirm	Boolean	否	是否需要用户进行确认操作
upgradeStatus	Integer	否	当前升级状态，0：待升级或待确认；1：升级中；2：升级异常；3：升级失败；4：升级成功

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
```

```
  },
  "params": {
    "iotId": "",
    "version": "1.0.1"
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200,
  "data": {
    "iotId": "xxxx",
    "step": 10,
    "desc": "xxxxxx",
    "success": false,
    "needConfirm": true,
    "upgradeStatus": 0
  }
}
```

获取设备固件

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/info/queryByUser	1.0.2	获取某一个设备当前的升级固件信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id

返回参数

参数	类型	必有	描述
currentVersion	String	是	当前设备实际烧录的版本号
version	String	是	待升级设备的目标版本号，如果目标版本号比实际烧录固件版本号低，则返回的是当前设备实际烧录的版本号

参数	类型	必有	描述
currentTimestamp	Date	否	设备当前烧录的固件，对应的文件上传到云端时记录的时间
timestamp	Date	是	设备当前最新固件文件上传到云端时记录的时间
size	String	是	文件包大小，单位Byte
md5	String	是	版本固件文件md5
name	String	是	版本固件名称
url	String	是	版本固件地址，sts加密
desc	String	否	版本详情描述

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "xxxxx"
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200,
  "data": {
    "currentVersion": "1.2.2",
    "version": "1.2.3",
    "currentTimestamp": "2017-09-08 16:08:08",
    "timestamp": "2017-09-10 12:11:34",
    "size": "4000000",
    "md5": "xxxxxxx",
    "name": "xxxxxxx",
    "url": "xxxxxxx",
    "desc": "xxxxxxx"
  }
}
```

取消固件升级

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/unupgradeByUser	1.0.2	取消某一个设备的固件升级流程	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
version	String	是	对应的固件版本号
iotid	String	是	设备id

返回参数

参数	类型	必有	描述
data	Boolean	是	取消操作指令处理是否成功True：成功False：失败

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "xxxxxxx",
    "version": "1.0.1"
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200,
  "data": "true"
}
```

获取正在升级的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/upgrade/listByUser	1.0.2	获取当前正在升级的设备列表信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无

返回参数

参数	子参数	类型	必有	描述
data		JSON Array	是	当前正在升级的设备信息列表
	iotId	String	是	设备id
	step	Integer	是	升级进度或升级错误信息 • 负数时，表示升级错误信息 - -1：升级操作失败 - -2：固件文件下载失败 - -3：固件文件校验失败 - -4：固件文件烧写失败 • 非负数时，表示升级进度，单位为：百分比
	desc	String	否	描述信息

参数	子参数	类型	必有	描述
	success	Boolean	是	是否成功标记，当为true时表示固件升级操作成功，反之表示升级未完成，包括正在升级或升级失败，请优先选择upgradeStatus参数来判断固件升级状态
	needConfirm	Boolean	是	是否需要用户进行确认操作
	upgradeStatus	Integer	是	当前升级状态，0：待升级或待确认；1：升级中；2：升级异常；3：升级失败；4：升级成功

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200,
  "data": [
    {
      "iotId": "xxxx",
      "step": 10,
      "desc": "xxxxxx",
      "success": false,
      "needConfirm": true,
      "upgradeStatus": 0
    },
    {
      "iotId": "xxxx",
```

```
        "step": 10,
        "desc": "xxxxxx",
        "success": false,
        "needConfirm": true,
        "upgradeStatus": 0
      }
  ]
}
```

上报固件升级进度

定义描述

path	版本	描述	是否需要用户身份的鉴权
/ota/progress/postByUser	1.0.2	上报某一个设备的固件升级的进度信息给到云端进行记录	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id
step	String	是	固件升级进度信息（百分比）
desc	String	是	升级信息的描述
version	String	是	对应的固件版本号信息

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "xxxx",
    "step": "1",
    "desc": "xxxx",
    "version": "1.0.1"
  }
```

```
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200
}
```

上报版本号

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/version/reportByUser	1.0.2	上报某一个设备当前烧录的版本号信息给云端	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id
version	String	是	固件版本号

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotId": "xxxx",
    "version": "1.0.1"
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200
```

```
}
```

批量确认设备升级

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/batchUpgradeByUser	1.0.2	批量确认设备进入升级状态，用于需要客户端进行人为确认方可进行升级的场景（可以支持单个设备，也可以支持批量设备）	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotIds	List	是	设备id信息列表

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
    "iotIds": ["iotid1","iotid2"]
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
  "code": 200
}
```

获取待升级的设备列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/thing/ota/listByUser	1.0.2	获取当前用户名下的待升级（待确认）的设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无

返回参数

参数	子参数	类型	必有	描述
data		JSON Array	是	当前用户名下的待升级或待确认的设备列表
	iotId	String	是	设备id
	image	String	否	设备对应的品类图标，当云端系统配置了设备所属产品的品类图标时有效
	deviceName	String	否	设备显示名称，会优先使用用户配置的设备昵称返回，如果不存在昵称的配置则返回对应的产品名称

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "iotToken": "token"
  },
  "params": {
  }
}
```

- 正常返回示例

```
{
  "id": "94e0871c-0770-xxxx-xxxx-e840d86b3ba6",
```

```
  "code": 200,
  "data": [
    {
      "iotId": "xxxx",
      "image": "xxxx",
      "deviceName": "xxxxxx"
    },
    {
      "iotId": "xxxx",
      "image": "xxxx",
      "deviceName": "xxxxxx"
    }
  ]
}
```


3.8.14 意见反馈

意见反馈（Feedback）是智能生活平台提供的连接用户、商家和开发者的一种通道。通过意见反馈，开发者和商家可以及时和用户保持沟通，提升用户满意度。

- 提交意见反馈
- 继续意见反馈
- 判断是否显示小红点
- 查询当前登录用户的意见反馈列表
- 根据意见反馈id查询意见反馈信息
- 批量更新反馈回复记录状态为已读

提交意见反馈

定义描述

path	版本	描述	是否需要登录
/feedback/add	1.0.1	提交一条意见反馈	是

请求参数

参数名	类型	是否必须	描述
mobileSystem	String	是	手机系统
appVersion	String	是	App版本号
type	Int	是	枚举值 • 1：设备问题 • 2：配网问题 • 3：其他问题
productKey	String	是	产品key
content	String	是	意见反馈内容

参数名	类型	是否必须	描述
iotId	String	是	设备Id
mobileModel	String	是	手机型号
contact	String	是	联系方式
topic	String	是	意见反馈主题
devicename	String	是	设备名称

返回参数

参数	类型	必有	描述
data	Long	是	意见反馈记录ID

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
         "appVersion": "10.0.0",
         "type": 1,
         "productKey": "a1xxxxxNb",
         "content": "重试多次配网失败",
         "mobileModel": "iPhoneX",
         "contact": "15066666666",
         "mobileSystem": "ios",
         "appVersion": "1.1",
         "iotId": "fy2Z1oZFxxxxxxxx01edf00",
         "topic": "设备无法配网",
         "devicename": "手环"
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": 3,
 "message": "success",
 "localizedMsg": null,
}
```

继续意见反馈

定义描述

path	版本	描述	是否需要登录
/feedback/reply/add	1.0.1	在已有的反馈意见上，累加新的意见反馈	是

请求参数

参数名	类型	是否必须	描述
type	Integer	是	用户为1，运营为2
topicid	Long	是	反馈意见ID
content	String	是	回复内容
mobileSystem	String	是	手机型号
mobileModel	String	是	手机系统
appVersion	String	是	app版本

返回参数

参数	类型	必有	描述
data	Long	是	继续意见反馈记录ID

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
         "type": 1,
         "content": "家里用的wifi网络配网",
         "mobileModel": "iPhoneX",
         "mobileSystem": "ios",
         "appVersion": "1.1",
         "topicid": 3
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": 8,
 "message": "success",
 "localizedMsg": null,
```

```
}
```

判断是否显示小红点

定义描述

path	版本	描述	是否需要用户身份的鉴权
/feedback/redpoint/get	1.0.1	判断是否在意见反馈菜单上显示小红点，小红点可以醒目提示用户该菜单有内容更新	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

无

返回参数

参数	类型	必有	描述
data	Boolean	是	是否需要在菜单上显示小红点：true需要显示；false不需要显示

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": true,
 "message": "success",
 "localizedMsg": null,
}
```

查询当前登录用户的意见反馈列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/feedbacklist/querybyuid	1.0.1	根据用户信息查询意见反馈列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数类型		必填	描述
pageSize	Integer	是	页大小，单页的记录数量上限是200
pageNo	Integer	是	当前页号，从1开始

返回参数

参数	类型	必有	描述
data	json	是	意见反馈的内容

data结构中详细字段说明如下：

mobileSystem	String	手机系统
appVersion	String	app版本号
type	Int	枚举值 • 1：设备问题 • 2：配网问题 • 3：其他问题
productKey	String	产品key
content	String	意见反馈内容
iotId	String	设备Id
mobileModel	String	手机型号
contact	String	联系方式
topic	String	意见反馈主题
devicename	String	设备名称
gmtModified	Date	记录最近修改时间
gmtCreate	Date	记录创建时间
productKey	String	产品key

mobileSystem	**String**	**手机系统**	
id	Long	意见反馈记录ID	
devicename	String	设备名称	
replyStatus	Int	回复状态，0-未读；1-已读	

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
         "pageNo": 1,
         "pageSize": 10,
   }
}
```

- 正常返回示例

```
返回值示例：
{
 "code": 200,
 "data": {
  "total": 1,
  "data": [
   {
    "mobileSystem": "安卓",
    "appVersion": "1.0",
    "gmtModified": 1539423792000,
    "gmtLastFeedback": 1539423676000,
    "isolationId": "2",
    "type": 2,
    "productKey": "2",
    "gmtCreate": 1539423792000,
    "content": "2",
    "tags": "2",
    "uid": "2",
    "lastFeedback": "2",
    "iotId": "2",
    "gmtLastReply": 1539423676000,
    "mobileModel": "华为",
    "lastReply": "2",
    "contact": "138",
    "tenantId": "2",
    "replyStatus": 0,
    "topic": "2",
    "devicename": "2",
    "id": 3,
    "feedbackStatus": 0
   }
  ],
```

```
    "pageSize": 20,
    "pageId": null,
  },
  "message": "success",
  "localizedMsg": null,
}
```

根据意见反馈id查询意见反馈信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/feedback/getbytopicId	1.0.1	根据意见反馈ID查询意见反馈信息	是

请求参数

参数	类型	必填	描述
feedbackTopicId	Long	是	意见反馈Id

返回参数

参数	类型	必有	描述
data	json	是	意见反馈的内容

data结构中详细字段说明如下

mobileSystem	String	手机系统
appVersion	String	app版本号
type	Int	枚举值 • 1：设备问题 • 2：配网问题 • 3：其他问题
productKey	String	产品key
content	String	意见反馈内容
iotId	String	设备Id
mobileModel	String	手机型号
contact	String	联系方式
topic	String	意见反馈主题

mobileSystem	**String**	**手机系统**
devicename	String	设备名称
gmtModified	Date	记录最近修改时间
gmtCreate	Date	记录创建时间
productKey	String	产品key
id	Long	意见反馈记录ID，系统生成
devicename	String	设备名称
replyStatus	Int	回复状态，0-未读；1-已读
feedBackReplyList	List	意见反馈回复以及再次反馈记录列表

FeedbackReplyList中的字段说明如下

id	**Long**	**回复以及再次反馈记录ID**
appVersion	String	app版本号
type	Int	2-回复；1-继续反馈
content	String	回复内容或者再次反馈内容
gmtCreate	Date	回复或者再次反馈的时间
topicid	Long	意见反馈记录ID

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "iotToken": "token"
   },
   "params": {
      "feedbackTopicId": 10
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "gmtModified": 1540460427000,
    "appVersion": "V1.2.0",
    "gmtLastFeedback": 1539683646000,
    "type": 103,
```

```
    "productKey": "",
    "content": "其他",
    "uid": "5021op82c076xxxxdd0e1788f5424b5384",
    "gmtLastReply": 1540460402000,
    "mobileModel": "M3X",
    "contact": "18373637272",
    "tenantNickName": "iot开发管理者",
    "id": 10,
    "userNickName": "180xxxx6420",

    "feedbackStatus": 0,
    "mobileSystem": "Android 6.0.1",
    "isolationId": "a103cxxxxTrF8",
    "gmtCreate": 1539683646000,
    "tags": "",
    "lastFeedback": "其他",
    "iotId": "",
    "lastReply": "<script>alert(1);</script>",
    "tenantId": "D99AEB09xxxxF008B2DCE1E313",
    "replyStatus": 0,
    "topic": "其他反馈",
    "devicename": "测试灯",
    "feedBackReplyList": [
     {
      "uid": "1584611943708270",
      "topicid": 10,
      "gmtModified": 1540xxxx1000,
      "id": 446,
      "type": 2,
      "gmtCreate": 1540xxxx1000,
      "content": "<script>alert(1);</script>"
     },
     {
      "uid": "1584xxxx708270",
      "topicid": 10,
      "gmtModified": 15404xxxx1000,
      "id": 445,
      "type": 2,
      "gmtCreate": 15404xxxx1000,
      "content": "春节将近",
     }
    ]
  },
  "message": "success",
  "localizedMsg": null,
}
```


批量更新反馈回复记录状态为已读

定义描述

path	版本	描述	是否需要用户身份的鉴权
/feedback/reply/status/readable/update	1.0.1	当用户打开某一条意见反馈时，调用该接口通知云端将所有回复置成已读状态	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
feedbackTopicId	Long	是	意见反馈记录ID

返回参数

参数	类型	必有	描述
data	Boolean	是	状态更新是否成功，true为成功；false为失败

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.0",
      "iotToken": "token"
   },
   "params": {
        "feedbackTopicId": 10
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": true,
  "message": "success",
  "localizedMsg": null,
}
```

3.8.15 锁的服务

提供一组锁SaaS专用的接口，方便对锁相关产品进行快速开发。

- 钥匙与虚拟用户绑定
- 钥匙与虚拟用户解绑
- 查询虚拟用户和某批钥匙的绑定关系
- 查询虚拟用户和某个钥匙的绑定关系
- 过滤还未绑定虚拟用户的钥匙信息
- 查询虚拟用户绑定的钥匙列表
- 查询设备下的虚拟用户列表

- 删除设备的钥匙信息
- 查询钥匙对应的虚拟用户信息
- 设置锁设备的昵称
- 查询锁的事件记录列表
- 设置钥匙的昵称

钥匙与虚拟用户绑定

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/bind	1.0.0	钥匙与虚拟用户进行绑定，只允许锁设备绑定的管理员进行操作	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID，与账户下创建的虚拟用户对应
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	是	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）
iotId	String	是	设备ID

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
```

```
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
     "virtualUserId" : "10B6336xxxx28F58AA5A18",
    "lockUserId" : "10",
    "lockUserType" : 1,
    "lockUserPermType" : 2,
    "iotId" : "ChFrkX2xxxx010803d10"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": null
}
```

钥匙与虚拟用户解绑

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/unbind	1.0.0	钥匙与虚拟用户进行解除绑定，只允许锁设备绑定的管理员进行操作	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID，与账户下创建的虚拟用户对应
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
iotId	String	是	设备ID

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
     "virtualUserId" : "10B63xxxx428F58AA5A18",
    "lockUserId" : "10",
    "lockUserType" : 1,
    "iotId" : "ChFrkX2xxxxH0010803d10"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": null
}
```

查询虚拟用户和某批钥匙的绑定关系

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/users/bindinfo/get	1.0.0	查询虚拟用户和某个设备下某批钥匙的绑定关系，如果存在绑定关系则返回相应的关系信息。限定锁设备绑定的管理员查询方有效	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID，与账户下创建的虚拟用户对应
originalLockUsers	List	是	钥匙ID列表（对应锁端的锁用户ID，由设备上报获得）

originalLockUsers列表结构

参数	类型	描述
iotId	String	设备ID
lockUserId	String	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）

返回参数

参数	类型	必有	描述
lockUsers	List	否	存在绑定关系的钥匙列表

lockUsers列表结构

参数	类型	描述
iotId	String	设备ID
lockUserId	String	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）
keyNickName	String	钥匙昵称（通过setKeyNickName设置过昵称后，在此处便可以获得昵称，否则为空）
userId	String	虚拟用户ID
identityId	String	账户身份ID
tenantId	String	租户ID
isolationId	String	资源隔离ID

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
     "virtualUserId" : "10B63xxxx428F58AA5A18",
    "originalLockUsers" : [
       {
         "iotId" : "ChFrkX2Cxxxx010803d10",
         "lockUserId": "10",
         "lockUserType" : 1
       },
       {
         "iotId" : "ChFrkX2Cxxxx0803d10",
         "lockUserId": "12",
         "lockUserType" : 2
       }
    ]
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": [
    {
       "iotId" : "ChFrkX2Cxxxx0010803d10",
       "lockUserId": "10",
       "lockUserType" : 1,
       "lockUserPermType": 2,
       "keyNickName":"keyNickNameTest",
       "userId": "10B6336xxxxD428F58AA5A18",
       "identityId": "5070xxxxe6970731e258b2250",
       "tenantId": "C9A56xxxxAE9875CDE5F01B3",
       "isolationId": "a10xxxxrF8"
    }
  ]
}
```

查询虚拟用户和某个钥匙的绑定关系

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/bindinfo/get	1.0.0	查询虚拟用户和某个钥匙的绑定关系，如果存在绑定关系则返回相应的关系信息，不存在则返回空。限定锁设备绑定的管理员查询方有效	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID，与账户下创建的虚拟用户对应
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
iotId	String	是	设备ID

返回参数

参数	类型	必有	描述
iotId	String	是	设备ID
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	是	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）

参数	类型	必有	描述
keyNickName	String	否	钥匙昵称（通过setKeyNickName设置过昵称后，在此处便可以获得昵称，否则为空）
userId	String	是	虚拟用户ID
identityId	String	是	账户身份ID
tenantId	String	是	租户ID
isolationId	String	是	资源隔离ID

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "virtualUserId" : "10B63xxxxD428F58AA5A18",
    "iotId" : "ChFrkX2CExxxx10803d10",
    "lockUserId": "10",
    "lockUserType" : 1
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": {
       "iotId" : "ChFrkX2Cxxxx10803d10",
       "lockUserId": "10",
       "lockUserType" : 1,
       "lockUserPermType": 2,
       "keyNickName": "keyNickNameTest",
       "userId": "10B6336Exxxx28F58AA5A18",
       "identityId": "5070opbxxxx3520731e258b2250",
       "tenantId": "C9A562xxxxCDE5F01B3",
       "isolationId": "a10xxxxLYnTrF8"
    }
}
```

过滤还未绑定虚拟用户的钥匙信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/unbind/filter	1.0.0	过滤出当前还未绑定的钥匙列表。限定锁设备绑定的管理员查询方有效	是，客户端SDK需启用身份的鉴权，进行身份认证
/lock/key/user/unbind/filter	1.0.2	调整入参，要求iotId、lockUserId等参数必填	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
originalLockUsers	List	是	钥匙ID列表（对应锁端的锁用户ID，由设备上报获得）

originalLockUsers列表结构

参数	类型	描述
iotId	String	设备ID
lockUserId	String	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）
keyNickName	String	钥匙昵称（通过setKeyNickName设置过昵称后，在此处便可以获得昵称，否则为空）

返回参数

参数	类型	必有	描述
originalLockUsers	List	否	过滤后的钥匙ID列表（对应锁端的锁用户ID）

originalLockUsers列表结构

参数	类型	描述
iotId	String	设备ID
lockUserId	String	钥匙ID（对应锁端的锁用户ID，由设备上报获得）
lockUserType	Integer	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "originalLockUsers": [
      {
        "iotId" : "ChFrkXxxxx010803d10",
        "lockUserId": "10",
        "lockUserType" : 1,
        "lockUserPermType": 2
      },
      {
        "iotId" : "ChFrkX2xxxx010803d10",
        "lockUserId": "12",
        "lockUserType" : 2,
        "lockUserPermType": 1
      }
    ]
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": [
   {
    "iotId" : "ChFrkXxxxx010803d10",
    "lockUserId": "10",
    "lockUserType" : 1,
    "lockUserPermType": 2
   }
  ]
}
```

查询虚拟用户绑定的钥匙列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/bindlist/get	1.0.0	查询虚拟用户下绑定的钥匙列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
virtualUserId	String	是	虚拟用户ID，与账户下创建的虚拟用户对应
lockUserType	Integer	否	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）不填表示查询全部类型
lockUserPermType	Integer	否	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ）不填表示查询全部类型

返回参数

参数	类型	必有	描述
lockUsers	List	否	存在绑定关系的钥匙列表

lockUsers列表结构

参数	类型	描述
iotId	String	设备ID
lockUserId	String	钥匙ID（对应锁端的锁用户ID）
lockUserType	Integer	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
lockUserPermType	Integer	钥匙权限（1-普通用户； 2-管理员用户； 3-胁迫用户 ）
keyNickName	String	钥匙昵称（通过setKeyNickName设置过昵称后，在此处便可以获得昵称，否则为空）
userId	String	虚拟用户ID
identityId	String	账户身份ID
tenantId	String	租户ID
isolationId	String	资源隔离ID

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "virtualUserId" : "10B633xxxx28F58AA5A18",
    "lockUserType" : 1,
    "lockUserPermType": 2
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": [
   {
      "iotId" : "ChFrkX2xxxx10803d10",
      "lockUserId": "10",
```

```
      "lockUserType" : 1,
      "lockUserPermType": 2,
      "userId": "10B6336xxxxF58AA5A18",
      "identityId": "5070opxxxx1e258b2250",
      "tenantId": "C9A56xxxxDE5F01B3",
      "isolationId": "a1xxxxTrF8"
    },
    {
      "iotId" : "ChFrkX2xxxx0803d10",
      "lockUserId": "12",
      "lockUserType" : 2,
      "lockUserPermType": 1,
      "userId": "109C2DxxxxD8FA1673",
      "identityId": "5070opxxxx731e258b2250",
      "tenantId": "C9A5xxxxCDE5F01B3",
      "isolationId": "a1xxxxTrF8"
    }
  ]
}
```

查询设备下的虚拟用户列表

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/dev/virtual/user/get	1.0.0	查询某个设备下绑定了钥匙的虚拟用户列表信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID

返回参数

参数	类型	必有	描述
userInfos	List	否	虚拟用户列表

userInfos列表结构

参数	类型	必填	描述
userId	String	是	虚拟用户ID
attrList	List	是	用户的属性值（参看虚拟用户对用户属性的定义）

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "iotId" : "ChFrkX2xxxx10803d10"
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": [
   {
      "userId" : "10B6336xxxx28F58AA5A18",
      "attrList": [
         {
           "attrKey": "name",
           "attrValue": "小明"
         },
         {
           "attrKey": "birthday",
           "attrValue": "1985-07-03"
         }
      ]
    },
    {
      "userId" : "109C2D6ED58xxxx7D8FA1673",
      "attrList": [
         {
           "attrKey": "name",
           "attrValue": "妈妈"
         },
         {
           "attrKey": "birthday",
           "attrValue": "1965-02-15"
         }
      ]
    }
  ]
}
```

删除设备的钥匙信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/user/delete	1.0.0	删除指定设备的钥匙信息，如果钥匙已与虚拟用户绑定，那么该操作会优先触发虚拟用户的解绑，不管最终钥匙删除是否成功，均会解绑虚拟用户，只允许锁设备绑定的管理员进行操作	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
iotId	String	是	设备ID

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "lockUserId" : "10",
    "lockUserType" : 1,
    "iotId" : "ChFrkX2xxxx803d10"
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": null
}
```

查询钥匙对应的虚拟用户信息

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/key/virtual/user/get	1.0.0	查询钥匙对应的虚拟用户信息	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
lockUserId	String	是	钥匙ID（对应锁端的锁用户ID）
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
iotId	String	是	设备ID

返回参数

参数	类型	必有	描述
userId	String	是	虚拟用户ID
attrList	List	否	用户的属性值

attrList列表结构

参数	类型	必填	描述
attrKey	String	是	属性Key
attrValue	String	是	属性值

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "lockUserId" : "10",
    "lockUserType" : 1,
    "iotId" : "ChFrkXxxxx10803d10"
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": {
        "userId" : "10B633xxxx8F58AA5A18",
       "attrList": [
          {
             "attrKey": "name",
             "attrValue": "小明"
          },
          {
             "attrKey": "birthday",
             "attrValue": "1985-07-03"
          }
       ]
   }
}
```

设置锁设备的昵称

定义描述

path	版本	描述	是否需要用户身份的鉴权
/lock/nickname/set	1.0.0	设置门锁设备的昵称，设备绑定的用户均可以设置，设置结果将同步所有关联用户	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
nickName	String	是	昵称信息
iotId	String	是	设备ID

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "nickName" : "大门锁",
    "iotId" : "ChFrkXxxxx10803d10"
   }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": null
}
```

查询锁的事件记录列表

定义描述

由于数据量巨大，该接口数据的查询依赖于大数据等基础设施能力，所以当前无法提供完全的实时性；当前该接口的数据时延情况如下：

- 90%的数据查询时延<1分钟，即在设备上报数据后1分钟内可以查询到
- 100%的数据查询延迟<2小时，即所有数据均可在2小时内查询到

path	版本	描述	是否需要用户身份的鉴权
/lock/event/history/query	1.0.0	查询锁设备的事件记录列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotid	String	是	设备ID

参数	类型	必填	描述
events	List	否	需要查询的设备事件类型列表（对应产品功能定义中事件定义的标识符）
start	Long	是	查询时间区间的开始时间（毫秒）
end	Long	是	查询时间区间的结束时间（毫秒）
lockUserId	String	否	钥匙ID（对应锁端的锁用户ID），与lockUserType同时有值或同时为null时有效，否则直接忽略设置
lockUserType	Integer	否	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙），与lockUserId同时有值或同时为null时有效，否则直接忽略设置
virtualUserId	String	否	虚拟用户ID，当同时设置了lockUserId、lockUserType和virtualUserId时，将会同时生效，即在虚拟用户绑定的钥匙列表的基础上加上配置的（lockUserId、lockUserType）
pageNo	Integer	是	当前页号，从1开始的页序号
pageSize	Integer	是	页大小，单页的记录数量上限（最大50）

返回参数

参数	类型	必有	描述
pageNo	Integer	是	当前页号
pageSize	Integer	是	页大小

参数	类型	必有	描述
total	Long	是	记录总数
data	List	否	事件记录列表

data列表结构

参数	类型	必填	描述
iot_id	String	是	设备ID
event_code	String	是	上报事件的标识符
service_date	String	是	服务端记录时间（格式：2018-06-25 17:41:18）
client_date	String	是	客户端上报时间（格式：2018-06-25 17:41:18）
KeyID	String	否	钥匙ID（对应锁端的锁用户ID，由设备上报获得），当对应的事件具备该出参时，才包含该字段
LockType	Integer	否	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙），当对应的事件具备该出参时，才包含该字段
UserLimit	Integer	否	钥匙权限（1-普通用户；2-管理员用户；3-胁迫用户 ），当对应的事件具备该出参时，才包含该字段
...	...	否	其他事件具备的出参信息，以具体的产品功能列表中事件的定义为准

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
```

```
    "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "iotId" : "ChFrkX2xxxx0803d10",
    "events" : ["DoorOpenNotification","HijackingAlarm","DoorUnlockedAlarm"]
    "start" : 1629500044000,
    "end" : 1629587444000,
    "virtualUserId":"10Bxxxx28F58AA5A18",
    "pageNo" : 1,
    "pageSize" : 50
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "message": "success",
  "data": [
    {
      "iot_id" : "ChFrkX2xxxx0803d10",
      "event_code": "DoorOpenNotification",
      "service_date" : "2018-06-25 17:41:18",
      "client_date": "2018-06-25 17:41:18",
      "KeyID": "10",
      "LockType": 2
    },
    {
      "iot_id" : "ChFrkXxxxx10803d10",
      "event_code": "HijackingAlarm",
      "service_date" : "2018-06-25 17:40:18",
      "client_date": "2018-06-25 17:40:18",
      "KeyID": "10",
      "LockType": 2
    },
    {
      "iot_id" : "ChFrkXxxxx10803d10",
      "event_code": "DoorUnlockedAlarm",
      "service_date" : "2018-06-25 17:38:18",
      "client_date": "2018-06-25 17:38:18"
    }
  ]
}
```

设置钥匙的昵称

定义描述

path	版本	描述	是否需要用户身份的鉴权
/living/lock/key/nickname/set	1.0.0	设置锁钥匙的别名	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
lockUserId	String	是	钥匙id
lockUserType	Integer	是	钥匙类型（1-指纹，2-密码，3-卡，4-钥匙）
iotId	String	是	设备id
nickName	String	是	钥匙别名
userId	String	否	淘宝账号ID信息
userType	String	否	账号类型（淘宝-TAOBAO）
origin	String	否	标示来源（目前只支持TmallGenie）

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
     "apiVer": "1.0.0",
  "iotToken": "token"
  },
  "params": {
    "lockUserId" : "10",
    "lockUserType" : 1,
    "iotId" : "ChFrkX2xxxx0803d10",
    "nickName":"nickNameTest",
  }
}
```

- 正常返回示例

```
{
  "code": 200,
 "message": "success",
  "data": null
}
```

3.8.16 Link Visual视频服务

提供关于Link Visual视频服务的API接口。

- 录像计划相关
  - 配置录像计划
  - 更新录像计划
  - 查询录像计划列表
  - 获取录像计划详情
  - 删除录像计划
  - 绑定录像计划到设备
  - 解绑设备的录像计划
  - 根据设备id查询绑定的录像计划
  - 查询绑定此录像计划的设备列表
- 事件联动录像计划相关
  - 配置事件联动录像计划
  - 更新事件联动录像计划
  - 查询事件联动录像计划列表
  - 删除事件联动录像计划
  - 获取事件联动录像计划详情
  - 绑定事件联动录像计划到设备
  - 解绑设备的事件联动录像计划
  - 根据设备id查询事件联动录像计划
  - 查询事件联动录像计划所关联的设备列表

- 云存录像相关
  - 根据时间范围查询云存的录像列表
  - 获取云存录像点播地址
  - 查询月录像
  - 查询事件列表
  - 根据事件id获取云存录像文件播放地址
  - 批量删除录像
  - 云端录像下载
  - 主动触发设备进行短录像任务
  - 设置连续录像文件时长
  - 查询连续录像文件时长配置
- 云存套餐相关
  - 查询赠送的云存储套餐详情
  - 领取赠送的云存储套餐
- 图片相关
  - 主动抓图
  - 根据时间查询图片列表
  - 根据图片id列表获取图片地址
  - 批量删除图片
  - 根据事件ID查询图片
- 事件相关
  - 设置业务事件的消息推送间隔
  - 查询业务事件的消息推送间隔
  - 根据事件ID批量查询事件详情

配置录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/set	2.1.0	配置录像计划，在计划时间范围内则通知设备推流，云存录像单个文件长度不超过30分钟，需要与设备绑定后才计划才会生效	是

请求参数

参数	类型	必填	默认值	描述
name	String	是		录像计划名称
allDay	int	是		是否全天 • 0：为非全天 • 1：为全天
timeSectionList	List	是		入参allDay值为0（非全天）时的时段信息
timeZone	int	否	8	UTC时区，取值区间-11～12

timeSection

参数	类型	必填	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

timeSection

返回参数

参数	类型	必有	描述
planId	String	是	录像计划ID

更新录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/update	2.1.0	更新录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		录像计划
name	String	否		录像计划名称
allDay	int	否		是否全天：0为非全天；1为全天
timeSectionList	List	否		入参allDay值为0（非全天）时的时段信息
timeZone	int	否	8	UTC时区，取值区间-11～12

timeSection

参数	类型	必填	描述	
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六	
begin	int	是	每天开始时间，单位秒，范围0-86399	
end	int	是	每天结束时间，单位秒，范围0-86399	

timeSection

返回参数

参数	类型	必有	描述
planId	String	是	录像计划ID

查询录像计划列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/query	2.1.0	查询录像计划列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
pageStart	int	否	0	起始页数，从0开始
pageSize	int	否	20	每页个数，最大值是500
timeZone	int	否	8	UTC时区，取值区间-11～12

返回参数

参数名	类型	必有	描述
total	int	是	总数
currentCount	int	是	当前页返回记录数
recordPlanList	List	是	录像计划列表

- recordPlan

参数	类型	必有	描述
planId	String	是	录像计划ID
name	String	是	录像计划名称
allDay	int	是	是否全天：0为非全天；1为全天
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

- timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

获取录像计划详情

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/get	2.1.0	获取录像计划详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		录像计划ID
timeZone	int	否	8	UTC时区，取值区间-11～12

返回参数

参数	类型	必有	描述
planId	String	是	录像计划ID
name	String	是	录像计划名
allDay	int	是	是否全天：0为非全天；1为全天
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

删除录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/delete	2.0.0	删除录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		录像计划ID

返回参数

无

绑定录像计划到设备

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/bind	2.0.0	设备与录像计划关联，绑定后设备按该计划执行录像，需要为设备开通云存储	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		录像计划ID
iotId	String	是		设备ID

参数	类型	必填	默认值	描述
streamType	int	否	0	码流类型：0主码流，1辅码流

返回参数

无

解绑设备的录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/unbind	2.0.0	解除设备与录像计划关系，解除后设备停止录像	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
streamType	int	否	0	码流类型：0主码流，1辅码流

返回参数

无

根据设备id查询绑定的录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/getbyiotid	2.1.0	根据设备IotId查询绑定的录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
streamType	int	否	0	码流类型：0主码流，1辅码流

参数	类型	必填	默认值	描述
timeZone	int	否8	8	UTC时区，取值区间-11～12

返回参数

参数	类型	必有	描述
planId	String	是	录像计划ID
name	String	是	录像计划名
allDay	int	是	是否全天：0非全天；1全天
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

查询绑定此录像计划的设备列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/plan/bind/device/query	2.0.0	查询绑定此录像计划的设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		录像计划ID
pageStart	int	否	0	起始页数，从0开始

参数	类型	必填	默认值	描述
pageSize	int	否	20	分页大小，最大值是500

返回参数

参数	类型	必有	描述
total	int	是	总数
currentCount	int	是	当前页列数
deviceList	List	是	设备列表

device：

参数	类型	必有	描述
iotId	String	是	设备ID
streamType	int	是	码流类型：0主码流，1辅码流

配置事件联动录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/set	2.1.0	配置事件联动录像计划，设备上报事件若落在计划范围内，则会通知设备推流并转存为云存录像，需要绑定到设备后才生效	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
name	String	是		计划名称
preRecordDuration	int	否	0	预录时间，单位秒
recordDuration	int	是		录像时间（总时间，包含预录部分），单位秒

参数	类型	必填	默认值	描述
allDay	int	是		是否全天：0非全天，1全天
eventTypeList	List	是		事件类型列表，1移动侦测
timeSectionList	List	是		入参allDay值为0（非全天）时的时段信息
timeZone	int	否	8	UTC时区，取值区间-11～12

timeSection

参数	类型	必填	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

返回参数

参数	类型	必有	描述
planId	String	是	事件联动录像计划ID

更新事件联动录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/update	2.1.0	更新事件联动录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		事件录像计划ID
name	String	否		事件录像计划名称

参数	类型	必填	默认值	描述
preRecordDuration	int	否	0	预录时间， 单位秒
recordDuration	int	否		录像时间（总时间，包含预录部分），单位为秒
allDay	int	否		是否全天：0非全天，1全天
eventTypeList	List	否		事件类型列表，1移动侦测
timeSectionList	List	否		入参allDay值为0（非全天）时的时段信息
timeZone	int	否	8	UTC时区，取值区间-11～12

timeSection

参数	类型	必填	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

返回参数

无

查询事件联动录像计划列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/query	2.1.0	查询事件联动录像计划列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
pageStart	int	否	0	起始页数，从0开始
pageSize	int	否	20	分页大小，最大值是500
timeZone	int	否	8	UTC时区，取值区间-11～12

返回参数

参数	类型	必有	描述
total	int	是	总数
currentCount	int	是	当前页列数
eventRecordPlanList	List	是	事件录像计划列表

- eventRecordPlan

参数	类型	必有	描述
planId	String	是	事件录像计划ID
name	String	是	事件录像计划名称
preRecordDuration	int	是	预录时间，单位秒
recordDuration	int	是	录像时间（总时间，包含预录部分），单位秒
allDay	int	是	是否全天：0非全天；1全天
eventTypeList	List	是	事件类型列表，1移动侦测
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

- timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399

参数	类型	必有	描述
end	int	是	每天结束时间，单位秒，范围0-86399

删除事件联动录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/delete	2.0.0	删除事件联动录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		事件录像计划ID

返回参数

无

获取事件联动录像计划详情

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/getbyid	2.1.0	获取事件联动录像计划详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		事件联动录像计划ID
timeZone	int	否	8	UTC时区，取值区间-11 ~ 12

返回参数

参数	类型	必有	描述
planId	String	是	事件录像计划ID

参数	类型	必有	描述
name	String	是	事件录像计划名称
preRecordDuration	int	是	预录时间，单位秒
recordDuration	int	是	录像时间（总时间，包含预录部分），单位秒
allDay	int	是	是否全天：0非全天；1全天
eventTypeList	List	是	事件类型列表。1移动侦测
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

绑定事件联动录像计划到设备

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/device/bind	2.0.0	绑定事件联动录像计划到设备，需要为设备开通云存储	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		事件联动录像计划ID
iotId	String	是		设备ID
streamType	int	否	0	码流类型：0主码流，1辅码流

返回参数

无

解绑设备的事件联动录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/unbind	2.0.0	解绑设备的事件联动录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
streamType	int	否	0	码流类型：0主码流，1辅码流

返回参数

无

根据设备id查询事件联动录像计划

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/plan/getbyiotid	2.1.0	根据设备IotId查询事件联动录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
streamType	int	否		码流信息：0主码流 ，1辅码流
timeZone	int	否	8	UTC时区，取值区间-11 ~ 12

返回参数

参数	类型	必有	描述
planId	String	是	事件联动录像计划ID
name	String	是	事件联动录像计划名称
preRecordDuration	int	是	预录时间，单位秒
recordDuration	int	是	录像时间（总时间，包含预录部分），单位秒
allDay	int	是	是否全天：0非全天，1全天
eventTypeList	List	是	事件类型列表，1移动侦测
timeSectionList	List	是	入参allDay值为0（非全天）时的时段信息

timeSection

参数	类型	必有	描述
dayOfWeek	int	是	周日到周六，0代表周日，6代表周六
begin	int	是	每天开始时间，单位秒，范围0-86399
end	int	是	每天结束时间，单位秒，范围0-86399

查询事件联动录像计划所关联的设备列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/eventrecord/bind/device/query	2.0.0	查询事件联动录像计划所关联的设备列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
planId	String	是		事件录像计划ID
pageStart	int	否	0	起始页数，从0开始

参数	类型	必填	默认值	描述
pageSize	int	否	20	每页个数，最大值500

返回参数

参数	类型	必有	描述
total	int	是	总数
currentCount	int	是	当前页列数
deviceList	List	是	设备列表

device

参数	类型	必有	说明
iotId	String	是	设备id
streamType	int	是	码流类型：0主码流，1辅码流

根据时间范围查询云存的录像列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/query	2.1.3	查询时间范围内的云端录像文件列表，支持分页查询，可通过查询出来的列表项中的filename来获取播放地址	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
streamType	int	否	0	码流类型：0主码流，1辅码流

参数	类型	必填	默认值	描述
beginTime	int	是		查询开始时间，1970年1月1日开始的秒数
endTime	int	是		查询结束时间，1970年1月1日开始的秒数
recordType	int	否	99（全部）	录像类型：0 计划录像，1 报警录像，2 主动录像，3 边看边录
pageStart	int	否	0	起始页数，从0开始
pageSize	int	否	20	每页个数，最大值500
needSnapshot	boolean	否	false	是否需要录像缩略图

返回参数

参数	类型	必有	描述
recordFileList	JSON	是	云端录像文件列表
record	JSON	是	录像文件信息
beginTime	String	是	文件开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	文件结束时间，格式yyyy-MM-dd HH:mm:ss
beginTimeUTC	String	是	文件开始UTC时间，格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	文件结束UTC时间，格式yyyy-MM-ddTHH:mm:ssZ

参数	类型	必有	描述
recordType	int	是	录像类型：0 计划录像，1 报警录像，2 主动录像，3 边看边录
streamType	int	是	码流类型：0主码流，1辅码流
fileName	String	是	文件名
fileSize	int	是	文件大小
snapshotUrl	String	否	录像缩略图URL
intelligentTypeList	String	否	智能事件列表

获取云存录像点播地址

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/vod/getbyfilename	2.0.0	按文件名获取设备云存录像播放地址	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数名	类型	是否必须	说明
iotId	String	是	设备ID
fileName	String	是	文件名，录像查询返回

返回参数

参数名	类型	说明
vodUrl	String	点播地址，带鉴权信息，无法重复使用

查询月录像

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/monthrecord/query	2.0.0	查询设备在某月的各天是否有云存储录像	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
month	String	是		格式为yyyyMM，如201806

返回参数

参数	类型	必有	描述
recordFlags	String	是	长度为当月天数，每位表示当天是否有录像

查询事件列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/event/query	2.1.2	查询事件列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
beginTime	long	是	查询开始时间，1970年1月1日开始的毫秒数
endTime	long	是	查询结束时间，1970年1月1日开始的毫秒数
eventType	int	否	事件类型，0表示全部，1表示移动侦测
pictureType	int	否	图片类型：0表示全部（默认），1表示原图，2表示缩率图
pageStart	int	否	起始页数，从0开始
pageSize	int	否	分页大小，默认为20

返回参数

参数	类型	必有	描述
eventList	List	是	事件列表
nextValid	boolean	是	下一页是否有效
nextBeginTime	int	是	下一页起始时间

event：

参数	类型	必有	描述
eventId	String	是	事件ID，若有生效的事件联动录像计划，则可根据事件ID来获取云存计划录像播放地址
eventTime	String	是	事件发生时间，格式yyyy-MM-ss HH:mm:ss
eventType	int	是	事件类型， 1移动侦测
eventDesc	String	是	事件描述
eventPicId	String	是	事件关联图片ID
eventData	String	是	事件附加信息
eventTimeUTC	String	是	事件发生UTC时间，格式yyyy-MM-ssTHH:mm:ssZ
eventPicUrl	String	否	报警事件图片URL地址
eventPicThumbUrl	String	否	报警事件图片缩率图URL
intelligentTypeList	String	否	智能事件列表，其中3代表人形侦测

根据事件id获取云存录像文件播放地址

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/vod/getbyeventid	2.1.0	根据事件ID获取云存录像文件播放地址，需要设备提前配置了生效的事件联动录像计划	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
eventId	String	是	事件ID

返回参数

参数	类型	必有	描述
vodList	List	是	播放列表

vodFile

参数	类型	必有	描述
fileName	String	是	文件名
vodUrl	String	是	播放地址
beginTime	String	是	文件开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	文件结束时间，格式yyyy-MM-dd HH:mm:ss
beginTimeUTC	String	是	文件开始UTC时间，格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	文件结束UTC时间，格式yyyy-MM-ddTHH:mm:ssZ

批量删除录像

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/batchdelete	2.0.0	批量删除录像	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备ID
fileNameList	List	是		录像文件名列表，一次删除上限为100

返回参数

参数	类型	必有	描述
deletedCount	int	是	删除成功个数

云端录像下载

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/vod/cloudfile/get	2.1.0	请求云端录像下载地址。云端录像需要进行MP4转码，这一过程会比较耗时，APP需要轮询该接口获取进度。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
fileName	String	是	录像文件名

返回参数

参数	类型	必有	描述
url	String	是	云端录像下载url。转码中url为空，转码成功后返回生成url。
status	int	是	录像转码状态，0成功，1转码中，-1转码异常
progress	int	是	录像转码进度，百分比0-100

主动触发设备进行短录像任务

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/trigger	2.1.0	触发设备主动推流，作为事件联动录像上传并存储在云端，需要事先为设备配置事件联动录像计划并为开通云存储	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
recordDuration	int	是	录制时长，单位秒，最大值1800
streamType	int	否	码流类型，0表示主码流（默认），1表示辅码流
preRecordDuration	int	否	预录时长，单位秒

返回参数

参数	类型	必有	描述
recordId	String	是	云存录像id

设置连续录像文件时长

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/durationconfig/set	1.0.0	设置连续录像文件时长	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id

参数	类型	必填	描述
recordDuration	int	否	连续录像文件时长配置，单位秒，最小600，最大3600，默认值为1800

返回参数

无

查询连续录像文件时长配置

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/record/durationconfig/get	1.0.0	查询连续录像文件时长配置	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是	无	设备id

返回参数

参数	类型	必有	描述
recordDuration	int	是	连续录像时长配置

查询赠送的云存储套餐详情

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/cloudstorage/presented/get	2.1.2	查询赠送的云存储套餐详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	默认值	描述
iotId	String	是		设备id

返回参数

参数	类型	必有	描述
type	int	是	云存储套餐类型：连续型为1，事件型2
lifecycle	int	是	数据在云端的存储周期， 只有1/3/7/30天
months	int	是	云存储服务有效期， 从1开始，单位月
consumed	int	是	是否已领取，0未领取，1已领取
startTime	String	否	开始时间，格式为yyyy-MM-dd HH:mm:ss
endTime	String	否	结束时间，格式为yyyy-MM-dd HH:mm:ss
expired	int	是	是否已过期，0未过期，1过期
startTimeUTC	String	否	开始时间UTC，格式为yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	否	结束时间UTC，格式为yyyy-MM-ddTHH:mm:ssZ

领取赠送的云存储套餐

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/cloudstorage/presented/consume	2.1.3	领取赠送的云存储套餐	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备id
enableDefaultPlan	boolean	否	是否设置7*24小时默认计划，默认值为false

返回参数

参数	类型	必有	描述
type	int	是	云存储套餐类型：事件型0，连续型1
lifecycle	int	是	数据在云端的存储周期，只有1、3、7、30天
months	int	是	云存储服务有效期，从1开始，单位月
consumed	int	是	是否已领取，0为领取，1已领取
startTime	String	否	开始时间，格式为yyyy-MM-dd HH:mm:ss
endTime	String	否	结束时间，格式为yyyy-MM-dd HH:mm:ss
expired	int	是	是否已过期
startTimeUTC	String	是	开始时间UTC，格式为yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间UTC，格式为yyyy-MM-ddTHH:mm:ssZ

主动抓图

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/picture/trigger	2.0.0	触发设备抓图上传到云端	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID

返回参数

参数	类型	必有	描述
pictureId	String	是	图片ID

根据时间查询图片列表

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/picture/querybytime	2.1.1	根据时间查询图片列表	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
startTime	long	是	开始时间，1970年1月1日开始的毫秒数
endTime	long	是	结束时间，1970年1月1日开始的毫秒数
pageStart	int	否	起始页，从0开始
pageSize	int	否	每页个数，最大值100，默认值为20
type	int	否	图片类型：0全部（默认），1原图，2缩略图
source	int	否	图片来源：0全部（默认），1报警抓图，2主动抓图，3其他

返回参数

参数	类型	必有	描述
pictureList	List	是	图片列表

picture

参数	类型	必有	说明
iotId	String	是	设备ID
pictureId	String	是	图片ID

参数	类型	必有	说明
pictureTime	String	是	图片创建时间
pictureUrl	String	是	图片URL
thumbUrl	String	是	图片缩略图URL
pictureTimeUTC	String	是	图片创建时间UTC
eventType	int	是	事件类型
source	int	是	图片来源：0全部（默认），1报警抓图，2主动抓图，3其他
intelligentTypeList	int	否	智能事件列表

根据图片id列表获取图片地址

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/picture/querybyids	2.1.0	根据图片id列表获取图片地址	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
pictureIdList	List	是	图片ID列表
type	int	否	图片类型：0表示全部（默认）；1表示原图；2表示缩率图

返回参数

参数	类型	必有	描述
pictureList	List	是	图片列表

picture

参数	类型	必有	描述
iotId	String	是	设备ID

参数	类型	必有	描述
pictureId	String	是	图片ID
pictureTime	String	是	图片创建时间
pictureUrl	String	是	图片URL
thumbUrl	String	是	图片缩略图URL
pictureTimeUTC	String	是	图片创建时间UTC

批量删除图片

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/picture/batchdelete	2.0.0	批量删除图片	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必有	描述
iotId	String	是	设备ID
pictureIdList	List	是	图片ID列表

返回参数

参数	类型	必有	描述
deleteCount	int	是	删除成功个数

设置业务事件的消息推送间隔

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/bizevent/config/set	1.0.0	设置业务事件的消息推送间隔。	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID

参数	类型	必填	描述
eventType	int	是	业务事件类型。1侦测告警事件；2云存到期提醒事件
eventInterval	int	否	业务事件的消息推送间隔，默认值为180

返回参数

无

查询业务事件的消息推送间隔

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/bizevent/config/get	1.0.0	查询业务事件的消息推送间隔	是

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
eventType	int	是	业务事件类型，1侦测告警事件；2云存到期提醒事件

返回参数

eventConfig

参数	类型	必有	描述
eventType	int	是	业务事件类型
eventInterval	int	是	业务事件的消息推送间隔

根据事件ID查询图片

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/pic/getbyevent	2.1.0	根据事件ID查询图片	是

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
eventIds	Array	是	事件ID列表

返回参数

pictureList

参数	类型	描述
iotId	String	设备ID
eventId	String	事件ID
pictureTime	String	图片创建时间
alarmPicId	String	图片ID
picUrl	String	图片URL
thumbUrl	String	缩略图URL
pictureTimeUTC	String	图片创建UTC时间

根据事件ID批量查询事件详情

定义描述

path	版本	描述	是否需要用户身份鉴权
/vision/customer/event/query/byeventids	1.0.0	根据事件ID批量查询事件详情	是，客户端SDK需启用身份的鉴权，进行身份认证

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
eventIds	List	是	事件ID列表
needAlarmPicture	boolean	否	是否查询事件对应的告警图片，默认值为false

参数	类型	必填	描述
needRecordSnapshot	boolean	否	是否查询事件对应的录像缩略图，默认值为false

返回参数

参数	类型	描述
eventId	String	事件ID
eventType	int	事件类型
eventTime	String	事件时间
eventTimeUTC	String	事件UTC时间
eventPicId	String	事件对应的图片ID
eventPicUrl	String	事件对应的图片URL
eventPicThumbUrl	String	事件对应的图片缩略图URL
eventFileName	String	事件对应的录像名
recordSnapshotUrl	String	事件对应的录像缩略图URL

3.9 Android SDK 手册

3.9.1 SDK升级

本文介绍API Level 7及以下版本升级至API Level 8的操作指导。

概述

生活物联网平台发布的App端SDK最新版本为API Level 8，该版本优化了整体SDK框架，对下载的SDK采对统一接口进行初始化（更多介绍请参见#unique_101）。

- 如果您是第一次集成SDK的新用户，请直接使用最新版本的SDK，不涉及SDK升级。

  您可以按照#unique_68来下载最新版本的SDK，并按照SDK初始化来初始化SDK。
- 如果您是已经集成过API Level 7或之前的SDK的用户，推荐您升级至最新版本的SDK，并使用新的初始化方案，新的方案对后期增加新功能、开拓海外市场等，具有更大的优势。当然您也可以保留之前的初始化方法，只更新SDK版本。本文档提供了两种升级方案，请自行选择。

升级SDK并使用新的初始化方案

1. 在**自有品牌App-SDK下载**页面，选择最新的API Level 8，并下载新的SDK套餐项。详细请参见#unique_68。

   下载到本地的文件为压缩包，解压后包含安全图片和dependency.gradle文件。

   > **说明：**
   > 由于下载Android安全图片需要先上传apk签名文件，如果您没有上传apk签名文件，压缩包里就不包含安全图片。安全图片的介绍请参见#unique_103。

2. 删除当前App工程中初始化相关的代码，如APIGatewaySDKDelegate、OpenAccountSDKDelegate、DeviceCenterSDKDelegate等。
3. 使用新的统一初始化接口，并完成SDK的初始化。详细请参见SDK初始化。
4. （可选）添加原来的定制化逻辑，如用户账号的定制化UI等。

升级SDK但不用新的初始化方案

1. 在**自有品牌App-SDK下载**页面，选择最新的API Level 8，并下载新的SDK套餐项。详细请参见#unique_68。

   下载到本地的文件为压缩包，解压后包含安全图片和dependency.gradle文件。

   > **说明：**
   > 由于下载Android安全图片需要先上传apk签名文件，如果您没有上传apk签名文件，压缩包里就不包含安全图片。安全图片的介绍请参见#unique_103。

2. 拷贝至App工程build.gradle同级目录，替换原来的同名文件。
3. 在build.gradle里添加以下代码。

```
apply from: "dependencies.gradle"
```

3.9.2 SDK初始化

当您开发自有App，下载并集成SDK后，需要对所有SDK进行初始化。

概述

从API Level 8开始，SDK的初始化不再需要挨个初始化SDK，而可以使用统一的初始化接口，一次性完成所有的所需SDK的初始化。如果您使用的是API Level 7或以下版本，建议您升级至最新版本，详细请参见SDK升级。

根据开发者下载SDK时勾选的**SDK配置项**，统一接口对如下SDK逐一初始化。

- API通道SDK （必选初始化）

- 账号及用户SDK （必选初始化）
- 身份认证SDK（必选初始化）
- 长连接通道SDK
- 设备模型SDK
- 移动应用推送SDK
- BoneMobile容器SDK

统一初始化支持默认初始化和带参数初始化两种方式，区别如下。

- 如果您的设备仅在中国内地使用，建议使用默认初始化。默认初始化不可以自定义，固定为直连中国内地接入点的正式版环境。
- 如果您的设备可能销往除中国内地以外地区，建议使用带参数初始化。用带参数初始化可以自定义，包括选择接入类型（中国内地或全球）、是否打开日志，以及定制三方通道离线推送参数等。

SDK初始化

1. 集成安全图片。详细操作请参见#unique_103。

   说明:
   安全图片文件名根据站点、版本的不同而存在差异。请勿修改安全图片名称，下载后直接拷贝到App工程目录下。

2. 下载并集成SDK。详细操作请参见#unique_68。
3. 初始化SDK。
   - 默认初始化

     ```
     IoTSmart.init(application); //初始化，App须继承自AApplication，否则会报错
     ```

   - 带参数初始化

     ```
     // 初始化参数配置
     IoTSmart.InitConfig initConfig = new IoTSmart.InitConfig()
                 // REGION_ALL表示连接全球多个接入点；REGION_CHINA_ONLY表示直连中国内地接入点
                 .setProductEnv(IoTSmart.PRODUCT_ENV_PROD)

                 // 是否打开日志
                 .setDebug(true);

         // 定制三方通道离线推送，目前支持华为、小米、FCM、OPPO、VIVO
         IoTSmart.PushConfig pushConfig = new IoTSmart.PushConfig();
           pushConfig.fcmApplicationId = "fcmid"; // 替换为从FCM平台申请的id
           pushConfig.fcmSendId = "fcmsendid"; // 替换为从FCM平台申请的sendid
           pushConfig.xiaomiAppId = "XiaoMiAppId"; // 替换为从小米平台申请的AppID
           pushConfig.xiaomiAppkey = "XiaoMiAppKey"; // 替换为从小米平台申请的AppKey
     ```

```
        pushConfig.oppoAppKey = "oppoAppKey"; // 替换为从OPPO平台申请的AppKey
        pushConfig.oppoAppSecret = "oppoAppSecret"; // 替换为从OPPO平台申请的
AppSecret
        // 华为与VIVO的推送通道在AndroidManifest.xml里面添加，此处无需配置

        initConfig.setPushConfig(pushConfig);

        /**
         * 设置App配网列表的产品范围，PRODUCT_SCOPE_ALL表示当前项目中已发布和未
发布的所有产品，
         * PRODUCT_SCOPE_PUBLISHED表示只包含已发布产品，正式发布的App请选择
PRODUCT_SCOPE_PUBLISHED
         */
        IoTSmart.setProductScope(IoTSmart.PRODUCT_SCOPE_PUBLISHED);

        // 初始化，App须继承自AApplication，否则会报错
        IoTSmart.init(app, initConfig);
```

三方通道的离线推送的详细介绍请参见移动应用推送开发指南。

4. 设置国家。

   阿里云IoT的云端服务为多区域部署，如果您将**config.regionType**设置成**REGION_ALL**后，还需要设置相应的国家，请参见#unique_11完成。

   > **说明:**
   > 如果您未设置国家，SDK初始化流程会挂起，直至国家设置完毕。在国家设置完成前请勿使用SDK的任一API，否则会造成报错。

5. （可选）定制登录注册页面。

   - 使用内置账号：请参照demo App里面的**SDKInitHelper.postInit**，来定制自己的登录页面。请参见Android Demo App模板。
   - 使用自有账号：您需要自行开发登录UI，登录成功后再授权认证。请参见 账号及用户SDK。

SDK API Reference

在使用生活物联网平台提供的SDK时，相关的SDK API注释请参见SDK API Reference。

3.9.3 通用SDK

通用SDK用来帮助所有SDK做初始化和统一设置，比如设置语言、设置国家等。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

设置国家相关

1. 显示国家、地区列表页面

   - 调用默认页面

     ```
     IoTSmart.showCountryList(final ICountrySelectCallBack callBack);
     ```

   - 自定义UI页面

     ```
     IoTSmart.getCountryList(final ICountryListGetCallBack callback);
     ```

2. 设置国家

   ```
   IoTSmart.setCountry(Country country, ICountrySetCallBack callBack);
   IoTSmart.setCountry(String domainAbbreviation, ICountrySetCallBack callBack);
   ```

   说明:

   App在中国内地与除中国内地以外地区之间切换时，需要重启才能正确初始化。

设置多语言

生活物联网平台目前支持中文（zh-CN）、 英文（en-US）、法文（fr-FR）、德文（de-DE）、日文（ja-JP）、韩文（ko-KR）、西班牙文（es-ES）、俄文（ru-RU）、意大利文（it-IT）、印地文（hi-IN）、葡萄牙文（pt-PT）等十一种语言。

- 设置语言

  统一切换API网关、用户账号、推送、插件等SDK的语言环境。

  ```
  IoTSmart.setLanguage(String languagename);
  ```

- 获取当前语言

  获取API网关、用户账号、推送、插件等SDK的语言环境。

  ```
  IoTSmart.getLanguage(String languageName);
  // 查看SDK当前设置的语言，如果您没有设置过语言，此处会返回当前系统语言
  ```

调试未发布产品

设置App配网列表的产品范围，取值如下。

- PRODUCT_SCOPE_ALL：表示当前项目中已发布和未发布的所有产品。

- PRODUCT_SCOPE_PUBLISHED：表示只包含已发布产品。正式发布的App请选择**PRODUCT_SCOPE_PUBLISHED**。

```
IoTSmart.setProductScope(String productScope);
```

设置日志开关

设置日志开关的状态，取值如下。

- true：表示显示所有日志
- fasle：表示只显示error日志

```
IoTSmart.setDebug(boolean debug);
```

3.9.4 账号及用户SDK

账号和用户SDK，提供账号能力，包括注册，登录，登出，获取账号，会话管理，人机校验等功能。集成了UI显示并同时提供UI定制的能力。

依赖SDK	概述
API通道	提供API通道能力

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

内置账号

- 基础功能
  - 注册

    调用登录页面后，在登录页面内会出现注册和忘记密码的功能，无需额外开发。
  - 登录

    目前平台已经支持两种登录方式：账号登录、邮箱登录。登录页面调起后，两种登录方式都可以使用。

```
LoginBusiness.login(new ILoginCallback() {
    @Override
    public void onLoginSuccess() {
        Log.i(TAG,"登录成功");
   }
    @Override
    public void onLoginFailed(int code, String error) {
        Log.i(TAG,"登录失败");
    }
```

```
});
```

- 退出登录

```
LoginBusiness.logout(new ILogoutCallback() {
    @Override
    public void onLogoutSuccess() {
        Log.i(TAG,"登出成功");
    }

    @Override
    public void onLogoutFailed(int code, String error) {
        Log.i(TAG,"登出失败");
    }
});
```

- 忘记密码

  在调起登录页面后，在登录页面内会出现注册和忘记密码的功能，无需额外开发。

- 注销账号

  注销账号的时候，同时会把用户绑定的设备都解除绑定关系，请参见注销账号。

- 修改个人信息

```
Map<String, Object> map = new LinkedHashMap<>();
        map.put("displayName", newName);
        OpenAccountUIService oas = OpenAccountSDK.getService(OpenAccountUIService
.class);
        oas.updateProfile(getApplicationContext(), map, new LoginCallback() {
            @Override
            public void onSuccess(OpenAccountSession openAccountSession) {
            }

            @Override
            public void onFailure(int i, String s) {
            }
        });
```

- 修改头像

  需要先将头像图片存储到云端，获取该图片的URL，再用如下方式更新。

```
Map<String, Object> map = new LinkedHashMap<>();
        map.put("avatarUrl", avatarUrl);
        OpenAccountUIService oas = OpenAccountSDK.getService(OpenAccountUIService
.class);
        oas.updateProfile(getApplicationContext(), map, new LoginCallback() {
            @Override
            public void onSuccess(OpenAccountSession openAccountSession) {
            }

            @Override
            public void onFailure(int i, String s) {
            }
        });
```

- 刷新会话

```
LoginBusiness.refreshSession(true, new IRefreshSessionCallback() {
```

```
    @Override
    public void onRefreshSuccess() {
        Log.i(TAG,"刷新Session成功");
    }

    @Override
    public void onRefreshFailed() {
        Log.i(TAG,"刷新Session失败");
    }
});
```

- 获取会话ID

  添加以下代码，获取当前会话的ID号。

```
LoginBusiness.getSessionId();
```

- 获取用户信息

  添加以下代码，获取当前登录用户的信息。

```
LoginBusiness.getUserInfo();
```

- 自定义登录页面

  参照Demo App代码里的SDKInitHelper类postInit方法来定制您的登录页面。获取Demo App源码请参见#unique_67。

三方自有账号

在对接三方自有账号之前，须先确认默认的账号页面（内置账号体系）可以正常注册和登录，并完成SDK的初始化。

选择自有账号通过OAuth2.0协议接入，配置流程参见用户账号开发指南。

三方自有账号在自有服务器上登录成功之后，获取Authorization Code（请参见用户账号开发指南），然后调用以下方法完成认证。

```
    LoginBusiness.authCodeLogin(authCode, new ILoginCallback() {
        @Override
        public void onLoginSuccess() {

        }

        @Override
        public void onLoginFailed(int i, String s) {
            Log.d(TAG, "code: " + i + ", str: " + s);
        }
    });
```

说明：

在调用登录或者登出接口时，直接调用LoginBusiness.login和LoginBusiness.logout两个方法即可。

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

3.9.5 身份认证SDK

提供基于iotToken的用户身份认证方案，通过和账号及用户SDK、API通道SDK的集成，完成用户身份凭证的生成和管理，以及发起API请求的用户身份的鉴权。

依赖 SDK	概述
API通道	提供API通道能力
账号及用户SDK	提供账号能力

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

当使用API通道执行某些需要IoT身份鉴权的业务请求时，统一身份认证SDK为API通道提供了一个自动完成请求认证信息填充的模块IoTCredentialProviderImpl，使用方式如下。

- 构建需要鉴权的业务请求

  构建IoTRequest时需要增加AuthType参数（初始化时已默认注册为“iotAuth”），代码如下所示。

```
IoTRequest request = new IoTRequestBuilder()
    .setPath("/kit/debug/ping") // 参考业务API文档，设置path
    .setApiVersion("1.0.0")  // 参考业务API文档，设置API Version
    .addParam("request", paramMap)
    .setAuthType("iotAuth")   //此处固定使用iotAuth
    .build();
```

- 处理认证失败

  目前服务端逻辑不支持同一个账号多端登录，如果一个账号在多个设备登录，那么只有最后一个登录的账号才可以正常访问IoT服务，其他账号则会返回认证错误，SDK提供一个接口，可以方便开发者随时监听这个错误，接口如下所示。

```
IoTCredentialManageImpl.getInstance(app).setIotCredentialListenerList(IoTTokenInvalidListener listener)
```

  其中IoTTokenInvalidListener定义如下。

```
public interface IoTTokenInvalidListener {
    void onIoTTokenInvalid();
```

```
}
```

说明：

这个API全局只能设置一次，推荐在初始化完成之后，调用这个API，监听IoTToken失效的情况，当onIotTokenInvalid被触发时，需要提示用户当前会话已失效，需要重新登录。

获取/刷新用户凭证

获取用户认证数据代码如下，如果结果返回NULL，可以调用异步刷新接口重新获取。

```
IoTCredentialManage ioTCredentialManage = IoTCredentialManageImpl.getInstance(app
);
if(ioTCredentialManage!=null){
    ioTCredentialManage.getIoTCredential();
}
```

其中IotCredentialData常用字段解释如下。

```
public class IoTCredentialData {
    /**
     * 临时令牌，对请求做身份校验
     */
    public String iotToken;

    /**
     * iotToken 创建时间，标准Unix时间戳
     */
    public long iotTokenCreateTime;

    /**
     * iotToken失效时间 -- 时间与云端同步，单位为毫秒
     */
    public long iotTokenExpireTime;

    /**
     * refreshToken ，用于刷新 iotToken
     */
    public String refreshToken;

    /**
     * refreshToken 创建时间，标准Unix时间戳
     */
    public long refreshTokenCreateTime;

    /**
     * refreshToken失效时间 -- 时间与云端同步，单位为毫秒
     */
    public long refreshTokenExpireTime;

    /**
     * iot 用户唯一身份标识 identityId
     */
    public String identity;
```

```
}
```

刷新用户认证数据代码如下。

```
IoTCredentialManage ioTCredentialManage = IoTCredentialManageImpl.getInstance(app
);
if (ioTCredentialManage != null) {
    ioTCredentialManage.asyncRefreshIoTCredential(new IoTCredentialListener() {
        @Override
        public void onRefreshIoTCredentialSuccess(IoTCredentialData ioTCredentialData) {
            Log.i(TAG, "refresh IoTCredentailData success :" + ioTCredentialData.toString());
        }

        @Override
        public void onRefreshIoTCredentialFailed(IoTCredentialManageError ioTCredent
ialManageError) {
            Log.i(TAG, "refresh IoTCredentailData failed ");
            if (ioTCredentialManageError != null) {
                Log.i(TAG, "error code is:" + ioTCredentialManageError.errorCode);
            }
        }
    });
}
```

混淆配置

在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keep public class com.aliyun.iot.aep.sdk.credential** {
    public <methods>;
    public <fields>;
}
```

常见错误码含义

错误码	含义
0	账号未登录
1	账号类型错误
2	账号AuthCode无效（即Session失效）
3	refreshToken过期，需要重新登录
4	服务器相应的报文格式错误（请将request和response提工单）
5	账号AuthCode校验错误
-1	其他错误，请查看APIClient返回的具体业务错误或者通过detail字段查看

3.9.6 API通道SDK

API通道SDK是IoT官方服务的API通道。 API通道基于HTTPS协议，并通过整合安全组件来提升通道的安全性。集成该SDK后，可以通过调用SDK的请求接口，完成对生活物联网平台云端接口的调用。

引入方式

- Maven仓库

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

- gradle依赖

```
api 'com.aliyun.iot.aep.sdk:apiclient:0.0.9.1'
api 'com.aliyun.alink.linksdk:api-client-biz:1.0.1'
```

- 混淆配置

  在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keep public class com.aliyun.iot.aep.sdk.apiclient.** {
    public <methods>;
    public <fields>;
}
```

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

- 请求调用

  以下接口示例是根据设备端iotId查询设备的详情和绑定关系的接口，云端介绍请参见根据设备获取绑定关系。

```
// 构建请求
Map<String, Object> params = new HashMap<>();
params.put("pageNo", 1);
params.put("pageSize", 100);
// iotId获取当前账号绑定设备列表的时候可以拿到，对应唯一设备
params.put("iotId", "xxxx");

IoTRequest request = new IoTRequestBuilder()
    .setScheme(Scheme.HTTPS) // 设置Scheme方式，取值范围：Scheme.HTTP或Scheme.HTTPS，默认为Scheme.HTTPS
    .setPath("uc/listBindingByDev") // 参照API文档，设置API接口描述中的Path，本示例为uc/listBindingByDev
    .setApiVersion("1.0.2")  // 参照API文档，设置API接口的版本号，本示例为1.0.2
    .authType("iotAuth") // 当云端接口需要用户身份鉴权时需要设置该参数，反之则不需要设置
    .addParam("input", "测试") // 参照API文档，设置API接口的参数，也可以使用.setParams(Map<Strign,Object> params)来设置
    .build();
```

```
// 获取Client实例，并发送请求
IoTAPIClient ioTAPIClient = new IoTAPIClientFactory().getClient();
ioTAPIClient.send(request, new IoTCallback() {
    @Override
    public void onFailure(IoTRequest request, Exception e) {
        // TODO根据e，处理异常
    }

    @Override
    public void onResponse(IoTRequest request, IoTResponse response) {
        int code = response.getCode();

        // 200 代表成功
        if(200 != code){
            //失败示例，参见 "异常数据返回示例"
            String mesage = response.getMessage();
            String localizedMsg = response.getLocalizedMsg();
            //TODO，根据mesage和localizedMsg，处理失败信息
            return;
        }
        Object data = response.getData();
        //TODO，可以将data转成一个本地的对象或者直接使用JSONObject进行数据解析

        /**
         * 解析data，data示例参见"正常数据返回示例"
         * 以下解析示例采用fastjson针对"正常数据返回示例"，解析各个数据节点
        */
        JSONObject jsonObject = JSON.parseObject(data);
        //获取业务层code
        String codeBiz = jsonObject.getString("code");
        //获取业务返回的数据
        JSONObject dataBizJsonObject = jsonObject.getJSONObject("data");
        //获取data，data数据是一个JSONArray，即设备列表
        JSONArray devListJsonArray = dataBizJsonObject.getJSONArray("data");
        //后续具体设备信息，则是对devListJsonArray进行一个遍历解析了
        if （devListJsonArray != null） {
            for (int i = 0; i < devListJsonArray.size(); i++) {
                JSONObject devJsonObject = devListJsonArray.getJSONObject(i);
                // TODO 从 devJsonObject 解析出各个字段
            }
        }
    }
});
```

- 正常数据返回示例

```
{
    "code":200,
    "data":{
        "total":1,
        "data":[
            {
                "productModel":"X1",
                "gmtModified":1581772608000,
                "categoryImage":"http://iotx-paas-admin.oss-cn-shanghai.aliyuncs.com/
publish-sg/image/xxxx.png",
                "netType":"NET_WIFI",
                "description":"February 15, 2020 9:16:48 PM CST Add binding",
                "nodeType":"DEVICE",
                "productKey":"xxx",
                "deviceName":"xxxx",
                "productName":"xxxx",
```

```
            "identityAlias":"xxxxxx",
            "iotId":"xrHAYrQDFSEpexxxx",
            "owned":1,
            "identityId":"5053ope232f4xxxx",
            "thingType":"DEVICE",
            "status":3
         }
      ],
      "pageNo":1,
      "pageSize":100
   },
   "id":"5168fe23-xxx-xxx-962c-1f61b8bdbd2d"
}
```

- 异常数据返回示例

```
{
    "code":2064,
    "id":"4fa207ca-fffd-xxxx-xxxx-e6f7ca6c99c3",
    "localizedMsg":"请求错误",
    "message":"need authorize to bind"
}
```

添加日志

API通道SDK本身没有输出任何请求日志（从性能和安全性的角度考虑）。您在开发过程中，可以选择以下任一方式来打印日志。

- 自行添加Tracker
- 使用内置的LogTracker（如以下代码所示）
- 在初始化中，传入debug=true
- 初始化后调用IoTSmart.setDebug(true)打开SDK调试日志开关

```
IoTAPIClientImpl.getInstance().registerTracker(new Tracker() {
    @Override
    public void onSend(IoTRequest ioTRequest) {
        // 收到上层接口请求，请求发送前触发
    }

    @Override
    public void onFailure(IoTRequest ioTRequest, Exception e) {
        // 请求失败时触发
    }

    @Override
    public void onResponse(IoTRequest ioTRequest, IoTResponse ioTResponse) {
        // 请求成功时触发
    }

    @Override
    public void onRealSend(IoTRequestWrapper ioTRequestWrapper) {
        // 上层接口发送之后触发
    }

    @Override
    public void onRawFailure(IoTRequestWrapper ioTRequestWrapper, Exception e) {
        // API 网关 SDK 接口请求失败时触发
    }
```

```
    @Override
    public void onRawResponse(IoTRequestWrapper ioTRequestWrapper, IoTResponse
ioTResponse) {
        // API 网关 SDK 接口响应成功时触发
    }
});
```

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

常见错误

错误码列表包含了初始化常见的一些错误，可以在logcat中看到如下异常信息。其它业务相关接口错误码、含义及解决方式请参见 常见问题。

错误码	含义	解决方法
103	安全组件so加载失败	过滤掉x86架构
202	安全图片与当前apk的签名不匹配导致的	修改当前apk签名或者重新上传apk生成新的安全图片
203	未找到安全图片	• 检查安全图片是否存在或者authCode是否正确 • 是否开启资源优化导致 • 是否启用instant run导致无法加载

3.9.7 BoneMobile容器SDK

BoneMobile容器SDK为可选模块，提供了加载插件的功能。如果您需要开发或者使用插件，则需要在App中集成BoneMobile容器SDK。

依赖SDK	概述
API通道	提供API通道能力

说明：

Google Play已于2019年8月1日停掉尚未支持64位体系的App，如果您App要在中国内地之外的国家和地区（包括港澳台地区）的Google Play应用商店上架，且用到了BoneKit SDK，那需要尽快升级，以免无法上架。后续如果推出新插件或老插件升级，仅针对已升级到v0.59 BoneKit SDK的自有App。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

- 打开插件面板

```
// 第一参数，当前上下文，通常是 Activity
        // 第二参数，格式为“link://plugin/{插件 Id}”，可以参见想要打开的插件的具体说明，以下以打开国内配网插件为例
        Router.getInstance().toUrl(context, "link://router/connectConfig"); // "link://plugin/a123kfz2KdRdrfYc"已不再推荐使用
```

- 打开插件面板并接收返回值

  请参见调用配网插件，并接受配网结果的示例。

```
// 启动插件
Bundle bundle = new Bundle();
bundle.putString("productKey", pk);
Router.getInstance().toUrlForResult(activity, "link://router/connectConfig",{your_request_code}, bundle);

// 接收配网结果
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
    if (REQUEST_CODE_PRODUCT_ADD == requestCode) {
        if (Activity.RESULT_OK != resultCode) {
            // 配网失败
            return;
        }

        String productKey = data.getStringExtra("productKey");
        String deviceName = data.getStringExtra("deviceName");
        // 配网成功
    }

}
```

- 打开调试面板

  请参见如下代码使用本地调试功能。

```
String ip = "{IP地址}"; //开发电脑上要开启 Bone 调试服务
        new BoneDevHelper().getBundleInfoAsync(this, ip, new BoneDevHelper.OnBondBundleInfoGetListener() {
        @Override
        public void onSuccess(BoneDevHelper.BoneBundleInfo boneBundleInfo) {
            BoneDevHelper.RouterInfo info = new BoneDevHelper().handleBundleInfo(MainActivity.this, boneBundleInfo);

            if (null == info) {
                return;
            }

            Router.getInstance().toUrl(MainActivity.this, info.url, info.bundle);
        }

        @Override
        public void onError(String message, Exception e) {
            Toast.makeText(MainActivity.this, message, Toast.LENGTH_SHORT).show();
```

```
            if (null != e) {
                e.printStackTrace();
            }
        }
    });
```

集成账号能力

插件中如果需要访问当前用户相关信息（例如：当前用户是否登录，用户的昵称是什么等），需要集成账号及用户 SDK，并注册API到容器。

账号SDK的集成请参见 账号及用户SDK。

API注册到容器，请参考如下代码如下。

```
BonePluginRegistry.register(BoneUserAccountPlugin.API_NAME, BoneUserAccountPlugin.class);
```

集成设备模型能力

插件中如果需要使用设备模型API（如设备面板的场景），则需要集成设备模型SDK，并注册API到容器。

设备模型SDK的集成，请参见设备模型SDK。

API注册到容器，请参考如下代码。

```
BonePluginRegistry.register("BoneThing", BoneThing.class);
```

集成配网能力

插件中如果需要使用配网能力，则需要集成配网SDK，并注册API到容器。

配网 SDK 的集成，请参见 配网SDK。

API 的注册，请参考如下代码。

```
BonePluginRegistry.register("BoneAddDeviceBiz",BoneAddDeviceBiz.class);
BonePluginRegistry.register("BoneLocalDeviceMgr",BoneLocalDeviceMgr.class);
BonePluginRegistry.register("BoneHotspotHelper",BoneHotspotHelper.class);

// 如果需要绑定蓝牙设备，需要添加如下代码
BonePluginRegistry.register("BoneThing", BoneThing.class);
```

集成长连接能力

插件中如果需要订阅云端消息，则需要集成长连接通道 SDK，并注册 API 到容器。

长连接通道 SDK 的基础，请参见长连接通道SDK。

API 的注册，请参考如下代码：

```
BonePluginRegistry.register("BoneChannel", BoneChannel.class);
```

Native和JS共享配置

某些业务场景下， Native 端和 JS 端可能需要共享一些配置。为了满足这个需求，我们开辟了一个 Native 和 JS 都可以访问的配置区。

JS 端访问配置区，请参见：环境配置信息。

Android 端访问配置区，可以参考如下代码：

```
// 设置
BoneConfig.set("region", "china");

// 获取
String region = BoneConfig.get("region");
```

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

图片库替换

为了减小BoneKit的SDK大小，降低接入成本及运行期的CPU/内存/文件系统资源消耗。BoneMobileRN容器允许开发者定制自己想用的图片库组件。

如果您已经有了成熟的APK包，并且使用了Fresco以外的图片库，请参见以下步骤替换图片库。

说明:

各个图片库支持的图片格式有所差异，在替换图片库时，需要注意以下内容。

- gif的支持，请使用支持gif动画的图片库或者自己实现gif的支持，否则可能导致使用了gif图片的页面显示异常。
- 图片圆角的支持，请实现对于圆角的支持，否则设置了圆角的图片可能显示异常。

1. 实现ImageLoaderModule

   实现 ImageLoaderModule，参见Facebook的官方文档Native Modules。

   请参见Facebook基于Fresco实现的ImageLoaderModule， 来实现您的ImageLoaderModule。

2. 实现 ReactImageManager

   实现 ReactImageManager，请参见 Facebook 的官方文档Native UI Components

   请参见Facebook基于Fresco实现的ReactImageManager.java，来实现您的ReactImageManager。

3. 实现ReactTextInlineImageViewManager

   实现ReactTextInlineImageViewManager与实现ReactImageViewManager类似，都继承自ViewManager，但差别在于，这次需要实现ShadowNode。ShadowNode需要继承自ReactTextInlineImageShadowNode。

   请参见基于Fresco实现的 FrescoBasedReactTextInlineImageViewManager和FrescoBasedReactTextInlineImageShadowNode来实现您的ReactTextInlineImageViewManager。

4. 实现ImagePackage

   ImagePackage的实现比较简单，把前面步骤实现的API和组件展现出来即可。参见下面的代码。

```
public class FrescoPackage implements ReactPackage {
 @Override
 public List<NativeModule> createNativeModules(ReactApplicationContext reactContext)
{
   return Arrays.<NativeModule>asList(
    new ImageLoaderModule(reactContext)
   );
 }

 @Override
 public List<Class<? extends JavaScriptModule>> createJSModules() {
  return Collections.emptyList();
 }

 @Override
 public List<ViewManager> createViewManagers(ReactApplicationContext reactContext)
{
   return Arrays.<ViewManager>asList(
    new ReactImageManager(),
    new FrescoBasedReactTextInlineImageViewManager()
   );
 }
}
```

5. 注册ImagePackage

   最后把刚才实现的ImagePackage注册到BoneKit的全局配置即可。

```
// 添加你自己的图片组件支持
      RNGlobalConfig.addBizPackage(new ImagePackage());
```

6. 删除Fresco的相关依赖和代码

   删除Fresco的依赖，仅保留BoneKit的依赖，参见如下代码。

```
compile ('com.aliyun.iot.aep.page:rn:0.0.3.2-SNAPSHOT'){
      exclude group:'com.aliyun.iot.aep.sdk', module:'rn-external-fresco' //移除Fresco的依赖
   }
   compile 'com.aliyun.alink.external:flog:1.0.0@aar'//ReactNative使用了Fresco内部的Flog相关类，Fresco移除后，需要额外添加flog
```

   删除FrescoPackage的添加代码如下。

```
// 添加基于 Fresco 的图片组件支持
```

```
        // RNGlobalConfig.addBizPackage(new FrescoPackage());
```

注：仅支持apiLevel<=6

支持GIF WebP

Android平台下，默认不支持GIF、WebP格式。可以增加支持库如下。

```
dependencies {
   compile 'com.facebook.fresco:animated-gif:0.11.0'  //需要GIF动画支持添加本行语句
   compile 'com.facebook.fresco:webpsupport:0.11.0'   //需要WebP格式支持添加本行语句
   compile 'com.facebook.fresco:animated-webp:0.11.0'  //需要WebP动画支持添加本行语句
}
```

混淆配置

请参见混淆配置。

3.9.8 长连接通道SDK

长连接通道SDK，提供IoT业务协议封装的云端数据下行能力。为App提供订阅、发布消息的能力和支持请求响应模型。

依赖SDK	概述
API通道	提供API通道能力

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

SDK封装了上行RPC请求、订阅、取消订阅等接口 。

SDK中描述的Topic都是简短的Topic。例如完整的上行请求Topic/sys/{productKey}/{deviceName}/app/up/test/publish，上行请求SDK 内部会判断补齐/sys/{productKey}/{deviceName}/app/up/，所以在调用SDK入参的时候只需要输入/test/publish即可。

对应的下行Topic，例如完整的设备状态变化下行Topic是 /sys/{productKey}/{deviceName}/app/down/things/status，SDK回调里面只会出现/things/status，自动阶段调用/sys/{productKey}/{deviceName}/app/down前缀。

业务请求响应模型

该接口实际上是封装了一个 Remote Procedure Call 的过程。以用户账号绑定通道的示例来说明内部逻辑，例如用户账号绑定通道的Topic为/sys/{productKey}/{deviceName}/app/up/account/bind。

向该Topic 发布数据前，需先订阅该Topic对应的Reply Topic。其格式如下所示 /sys/{productKey}/{deviceName}/app/down/account/bind_reply。

订阅成功后开始发布数据，IoT用户中心在收到SDK发布到/sys/{productKey}/{deviceName}/app/up/account/bind这个Topic的数据后，并完成账号绑定的业务逻辑后，会往 /sys/{productKey}/{deviceName}/app/down/account/bind_reply这个Topic发布响应数据。

SDK在收到reply Topic的数据后，将响应结果通过**onSuccess**回调给用户，从而完成整个业务逻辑。

业务请求响应模型示例如下。

```
//带请求响应的 RPC 请求
String topic = "path/of/topic";
JSONObject params = new JSONObject();
params.put("key","value");
MobileChannel.getInstance().asyncSendRequest(topic, null, params, new IMobileReq
uestListener() {
    @Override
    public void onSuccess(String jsonData) {
        ALog.d(TAG,"onSuccess, rsp = "+jsonData);
    }

    @Override
    public void onFailure(AError error) {
        ALog.d(TAG,"onFailure");
    }
});
```

订阅Topic

```
//订阅请求
String topic = "path/of/topic";
MobileChannel.getInstance().subscrbie(topic, new IMobileSubscrbieListener() {
    @Override
    public void onSuccess(String topic) {
        ALog.d(TAG,"onSuccess, topic = "+topic);

    }

    @Override
    public void onFailed(String topic, AError error) {
        ALog.d(TAG,"onFailed, topic = "+topic);
    }

    @Override
    public boolean needUISafety() {
        return false;
    }
});
```

取消订阅Topic

```
//取消订阅
String topic = "path/of/topic";
MobileChannel.getInstance().unSubscrbie(topic, new IMobileSubscrbieListener() {
```

```
    @Override
    public void onSuccess(String topic) {
        ALog.d(TAG,"onSuccess, topic = "+topic);

    }

    @Override
    public void onFailed(String topic, AError error) {
        ALog.d(TAG,"onFailed, topic = "+topic);
    }

    @Override
    public boolean needUISafety() {
        return false;
    }
});
```

Publish数据

上行请求推荐使用API通道。

```
//Publish 请求
String topic = "path/of/topic";
JSONObject params = new JSONObject();
params.put("key","value");
MobileChannel.getInstance().ayncSendPublishRequest(topic, params, new IMobileReq
uestListener() {
    @Override
    public void onSuccess(String jsonData) {
        ALog.d(TAG,"onSuccess, rsp = "+jsonData);
    }

    @Override
    public void onFailure(AError error) {
        ALog.d(TAG,"onFailure");
    }
});
```

注册下行Listener

设置长连接通道连接变化以及云端推送的下行消息监听事件。目前云端下行接口包括设备属性、事件及连接状态变更推送，相关Topic参见长连接服务API。

```
/**
 设置通道的下推回调，如果不需要用的时候，记得调用 unRegisterDownstreamListener
 回调里的Method即为Topic，e.g. /thing/properties,/thing/events,/thing/status
*/
MobileChannel.getInstance().registerDownstreamListener(true, new IMobileDow
nstreamListener() {
    @Override
    public void onCommand(String method, String data) {
        ALog.d(TAG,"接收到Topic = "+method+", data="+data);
    }

    @Override
    public boolean shouldHandle(String method) {
        // method 即为Topic，如果该Topic需要处理，返回true后onCommand才会回调。
        return true;
    }
```

```
});

/** 注册通道的状态变化,记得调用 unRegisterConnectListener */
MobileChannel.getInstance().registerConnectListener(true, new IMobileConnectListener()
 {
   @Override
   public void onConnectStateChange(MobileConnectState state) {
      ALog.d(TAG,"通道状态变化，state="+state);
   }
});
```

解绑长连接通道与账号

在初始化时，已实现长连接通道与账号的绑定，此时如果您需要解绑长连接通道与账号，参照以下代码执行。

如需解绑长连接通道跟用户账号的绑定，请在账号登出前操作，否则会导致解绑失败。

```
MobileChannel.getInstance().unBindAccount(new IMobileRequestListener() {
        @Override
        public void onSuccess(String jsonData) {
        }

        @Override
        public void onFailure(AError error) {
        }
    });
```

混淆配置

在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keep class com.aliyun.alink.linksdk.channel.**{*;}
```

3.9.9 配网SDK

提供了把WiFi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、蓝牙辅助配网、智能路由器配网以及设备间相互配网等。

依赖SDK	概述
API 通道	提供云端API通道能力，需要完成该SDK的初始化
身份认证	提供云端API通道身份认证（需要零配发现能力的时候需要该依赖）
Breeze SDK	提供蓝牙辅助配网支持，依赖breeze-biz SDK

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

本配网SDK对外主要提供设备发现，设备配网，以及获取设备端token的能力。

设备发现

提供局域网内发现已配网的设备，包括WiFi设备以及以太网设备，以及局域网内已上电的待配设备（前提是当前局域网内已经有一个已配好网的智能设备，且该设备具有发现待配网设备的能力）的能力。

设备发现相关API，请参见LocalDeviceMgr。

启动设备发现

发现本地的已配网设备，或者零配、路由器发现的待配设备，或者符合adh{pk}{mac}格式的设备热点设备。发现的待配设备信息可以作为后续设备配网的入参信息。如果需要对已发现的已配网设备做过滤（如过滤已绑定的设备，不支持的设备等），可以参见配网服务。

```
// 开始发现设备
// enumSet 是需要使用的防发现方式 EnumSet<DiscoveryType>, 请根据需要选择发现方式，并添加对应的依赖;
// 第三个参数是获取零配或智能路由器发现的待配设备 请求时需要携带的参数
LocalDeviceMgr.getInstance().startDiscovery(context, enumSet, null, new IDeviceDiscoveryListener() {
    @Override
    public void onDeviceFound(DiscoveryType discoveryType, List<DeviceInfo> list) {
        // 发现的设备
        // LOCAL_ONLINE_DEVICE 当前和手机在同一局域网已配网在线的设备
        // CLOUD_ENROLLEE_DEVICE 零配或智能路由器发现的待配设备
        // BLE_ENROLLEE_DEVICE 发现的是蓝牙Wi-Fi双模设备（蓝牙模块广播的subType=2即为双模设备）
        // SOFT_AP_DEVICE 发现的设备热点
        // BEACON_DEVICE 一键配网发现的待配设备
          // 注意：发现蓝牙设备需添加 breeze-biz SDK依赖
    }
});
```

停止发现设备

停止发现本地已配和线上待配设备，调用该接口会清除已发现设备列表，确保与startDiscovery成对调用。

```
// 停止设备发现
LocalDeviceMgr.getInstance().stopDiscovery();
```

设备配网

设备配置添加相关API说明，参见配网服务。

- 设置待配网设备信息

```
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = "xx"; // 商家后台注册的 productKey，不可为空
deviceInfo.deviceName = "xxx";// 设备名, 可为空
```

```
deviceInfo.productId = "xxx";// 产品 ID， 蓝牙辅助配网必须
deviceInfo.id= "xxx";// 设备热点的id，在发现热点设备返回到APP的时候会带这个字段
// 设备热点配网 ForceAliLinkTypeSoftAP  蓝牙辅助配网 ForceAliLinkTypeBLE
// 二维码配网 ForceAliLinkTypeQR  手机热点配网 ForceAliLinkTypePhoneAP
// 一键配网 ForceAliLinkTypeBroadcast  零配 ForceAliLinkTypeZeroAP
deviceInfo.linkType = "ForceAliLinkTypeNone"; // 默认一键配网

//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(DeviceInfo deviceinfo);
```

配网时，指定的配网方式 linkType，请参见 LinkType；设备信息相关实体类，请参见DeviceInfo。

- 开始设备配网

  上述设置好待添加设备信息，进入配网。调用startAddDevice接口进入配网流程。如果出现某些高版本手机无法获取WiFi的**SSID**，需要申请位置权限，并打开GPS。

  - 如使用手机热点方式配网，linkType= "ForceAliLinkTypePhoneAP"，需要用户确保应用已具备了 **WRITE_SETTINGS**权限，该权限需要 App动态申请。
  - 设备热点配网必需要位置权限和GPS服务。

```
// 开始添加设备
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) {
      // 手机热点配网、设备热点配网、一键配网、蓝牙辅助配网、二维码配网会走到该流程，
       // 零配和智能路由器配网不会走到该流程。
      // prepareType = 1提示用户输入账号密码
      // prepareType = 2提示用户手动开启指定热点 aha 12345678
      // 执行完上述操作之后，调用toggleProvision接口继续执行配网流程
   }

   @Override
   public void onProvisioning() {
      // 配网中
   }

   @Override
   public void onProvisionStatus(ProvisionStatus provisionStatus) {
     // 二维码配网
     // provisionStatus=ProvisionStatus.QR_PROVISION_READY表示二维码ready了
     // ProvisionStatus.QR_PROVISION_READY.message() 获取二维码内容
     // 注意：返回二维码时已开启监听设备是否已配网成功的通告，并开始计时，UI端应提示用户尽快扫码，
     // 如果在指定时间配网超时了，重新调用开始配网流程并刷新二维码。

     // 设备热点配网
     // 只有需要提示用户需要手动连接设备热点或者恢复WiFi连接的场景才会回调，
     // 比如android 10，或者非android 10发现或连接设备热点失败。
     // provisionStatus=ProvisionStatus.SAP_NEED_USER_TO_CONNECT_DEVICE_AP 表示需要用户手动连接设备热点，
```

```
    // provisionStatus=ProvisionStatus.SAP_NEED_USER_TO_RECOVER_WIFI 表示需要用户手动恢复到配网之前的WiFi,
    // 针对android 10不支持的场景，可以根据这两个事件增加交互，让用户处理。
  }

  @Override
  public void onProvisionedResult(boolean b, DeviceInfo deviceInfo, DCErrorCode errorCode) {
    // 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。
  }
});
```

配网回调接口，请参见 IAddDeviceListener。

- 输入账号密码

  在收到onProvisionPrepare回调后，引导用户输入**ssid**和**password**，并调用此方法传入**ssid**、**password**、**timeout**信息启动配网。只有广播配网场景下，才会触发onProvisionPrepare回调。如果出现部分手机一直无法获取到SSID，打开GPS再尝试获取SSID（部分手机需要开启GPS才能获取到）。

```
String ssid = "ssid";// 热点配网的时候注意 要先获取ssid，然后再开启热点，否则无法正确获取到SSID
String ssid = "xxxxxxxx";
int timeout = 60;//单位秒 目前最短只能设置60S
AddDeviceBiz.getInstance().toggleProvision(ssid, password, timeout);
```

- 停止配网

```
// 停止配网
AddDeviceBiz.getInstance().stopAddDevice();
```

- 设置配网模式

```
String linkType = "ForceAliLinkTypeBroadcast";
// 设置配网模式
AddDeviceBiz.getInstance().setAliProvisionMode(linkType);
```

获取设备绑定Token

设备绑定时除了需要**productKey**、**deviceName**，还需要设备端的token，该接口提供从设备端获取绑定token的功能。WiFi设备或者以太网设备均可以使用该接口获取绑定需要的Token。以太网设备需要先获取到设备的**productKey**、**deviceName**，可以参考使用startDiscovery接口去获取。

```
// 获取设备绑定 token
LocalDeviceMgr.getInstance().getDeviceToken(context, productKey, deviceName, 60*1000, 5*1000， new IOnDeviceTokenGetListener() {
        @Override
        public void onSuccess(String token) {
            // TODO bind
        }

        @Override
        public void onFail(String reason) {
        }
```

```
    });
```

SDK依赖

- Maven仓库地址

```
// 阿里云仓库地址，包括了IOT事业部开发的SDK
maven {
    url "http://maven.aliyun.com/nexus/content/repositories/releases/"
}
```

- Gradle依赖

```
compile ('com.aliyun.alink.linksdk:ilop-devicecenter:1.7.3'){
    transitive = true
}
// 如果需要蓝牙辅助配网能力，请加上如下依赖；不需要蓝牙辅助配网能力，则不需要加下面这个依赖；
// compile ('com.aliyun.alink.linksdk:breeze-biz:1.1.4')
```

混淆配置

在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keep public class com.aliyun.alink.business.devicecenter.** {*;}

-keep public class com.aliyun.alink.linksdk.alcs.coap.**{*;}
```

在插件中使用配网能力

如果需要在插件中使用配网能力，需要向BoneMobile容器中注册配网API，请在插件加载前，调用如下注册代码。

```
BonePluginRegistry.register("BoneAddDeviceBiz",BoneAddDeviceBiz.class);
BonePluginRegistry.register("BoneLocalDeviceMgr",BoneLocalDeviceMgr.class);
BonePluginRegistry.register("BoneHotspotHelper",BoneHotspotHelper.class);
```

3.9.10 移动应用推送SDK

阿里移动推送（Alibaba Cloud Mobile Push）是基于大数据的移动智能推送服务，帮助App快速集成移动推送的功能。在实现高效、精确、实时的移动推送的同时，降低了开发成本并提高了用户活跃度和应用的留存率。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

1. 在应用中注册和启动移动推送。该操作封装在SDK初始化中，无需额外操作。
2. 创建消息接收Receiver，继承自com.alibaba.sdk.android.push.MessageReceiver，并在对应回调中添加业务处理逻辑，参照以下代码执行。

```
public class MyMessageReceiver extends MessageReceiver {
```

```
    // 消息接收部分的LOG_TAG
    public static final String REC_TAG = "receiver";
    @Override
    public void onNotification(Context context, String title, String summary, Map<String
, String> extraMap) {
        // TODO 处理推送通知
        Log.e("MyMessageReceiver", "Receive notification, title: " + title + ", summary: " +
summary + ", extraMap: " + extraMap);
    }
    @Override
    public void onMessage(Context context, CPushMessage cPushMessage) {
            Log.e("MyMessageReceiver", "onMessage, messageId: " + cPushMessage.
getMessageId() + ", title: " + cPushMessage.getTitle() + ", content:" + cPushMessage.
getContent());
    }
    @Override
    public void onNotificationOpened(Context context, String title, String summary,
String extraMap) {
        Log.e("MyMessageReceiver", "onNotificationOpened, title: " + title + ", summary: "
 + summary + ", extraMap:" + extraMap);
    }
    @Override
    protected void onNotificationClickedWithNoAction(Context context, String title,
String summary, String extraMap) {
        Log.e("MyMessageReceiver", "onNotificationClickedWithNoAction, title: " + title +
 ", summary: " + summary + ", extraMap:" + extraMap);
    }
    @Override
    protected void onNotificationReceivedInApp(Context context, String title, String
summary, Map<String, String> extraMap, int openType, String openActivity, String
openUrl) {
        Log.e("MyMessageReceiver", "onNotificationReceivedInApp, title: " + title + ",
summary: " + summary + ", extraMap:" + extraMap + ", openType:" + openType + ",
openActivity:" + openActivity + ", openUrl:" + openUrl);
    }
    @Override
    protected void onNotificationRemoved(Context context, String messageId) {
        Log.e("MyMessageReceiver", "onNotificationRemoved");
    }
}
```

3. 将该receiver添加到AndroidManifest.xml中。

```
<!-- 消息接收监听器 （用户可自主扩展） -->
<receiver
    android:name=".MyMessageReceiver"
    android:exported="false"> <!-- 为保证receiver安全，建议设置不可导出，如需对其他应
用开放可通过android：permission进行限制 -->
    <intent-filter>
        <action android:name="com.alibaba.push2.action.NOTIFICATION_OPENED" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.alibaba.push2.action.NOTIFICATION_REMOVED" />
    </intent-filter>
    <intent-filter>
        <action android:name="com.alibaba.sdk.android.push.RECEIVE" />
    </intent-filter>
</receiver>
```

三方辅助推送通道初始化

目前仅支持小米、华为、FCM的推送通道，当App不在运行的时候，三方辅助推送通道可以将消息推送到手机。

初始化之前，需要申请和配置好三方辅助的推送通道，参见移动应用推送开发指南。

三方辅助推送通道的初始化在SDK初始化时已完成，无需另外操作。

使用方式

支持两种类型的消息下发，通知类型和消息类型。

- 通知类型

  服务器下发的通知，SDK会自动处理，根据下发配置标题和内容自动弹出通知，无需在MyMessageRecevier中添加业务代码。

- 消息类型

  服务器下发的消息可以在初始化中创建的MyMessageReceiver对应的回调方法中获得。如需弹出通知，需要自己根据下发的内容在此模式下弹出通知。

告警功能

告警功能依赖设备和用户绑定，用户和设备的绑定已经封装在SDK初始化中，您只需关注告警功能的业务逻辑即可。

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

混淆配置

在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keepclasseswithmembernames class ** {
    native <methods>;
}

-keepattributes Signature

-keep class sun.misc.Unsafe { *; }

-keep class com.taobao.** {*;}

-keep class com.alibaba.** {*;}

-keep class com.alipay.** {*;}

-keep class com.ut.** {*;}

-keep class com.ta.** {*;}
```

```
-keep class anet.**{*;}
-keep class anetwork.**{*;}
-keep class org.android.spdy.**{*;}
-keep class org.android.agoo.**{*;}
-keep class android.os.**{*;}
-dontwarn com.taobao.**
-dontwarn com.alibaba.**
-dontwarn com.alipay.**
-dontwarn anet.**
-dontwarn org.android.spdy.**
-dontwarn org.android.agoo.**
-dontwarn anetwork.**
-dontwarn com.ut.**
-dontwarn com.ta.**
```

3.9.11 设备模型SDK

设备模型SDK提供了App端的设备模型（属性、事件、服务），用来开发设备界面，实现手机对设备的查看和控制。

依赖 SDK	概述
API通道SDK	API通道SDK，提供IoT业务协议封装的HTTPS请求能力，并通过整合安全组件来提升通道的安全性。
长连接通道SDK	长连接通道SDK，提供IoT业务协议封装的云端数据下行能力；为App 提供订阅、发布消息的能力，和支持请求响应模型。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

本地通信功能是设备模型SDK提供的一项基础能力，它提供了在外网断开的情况下，局域网内设备控制的特性。

当外网络断开时，本地通信模块会搜寻当前局域网内的设备，如果发现的设备是App用户曾经控制过的设备，此时可以通过本地通信链路去控制设备。

在外网断开时，向云端拉取用户账号下的设备列表会失败，此时可以使用以下接口获取当前可以本地通信控制的设备列表。此API在SDK2.1.4版本中新增。

```
DeviceManager.getInstance().getLocalAuthedDeviceDataList();
```

返回值是一个JSON Array对象，对象内的数据格式具体请参见根据设备获取绑定关系协议的AccountDevDTO结构。

设备创建

```
PanelDevice panelDevice = new PanelDevice(iotid);
```

iotid可以通过云端接口获取。具体请参见 物的模型服务。

```
panelDevice.init(context, new IPanelCallback() {
        @Override
        public void onComplete(boolean bSuc, Object o) {
        }
    });
 //context是应用的上下文，IPanelCallback是初始化回调接口
 // bSuc 表示初始化结果，true为成功，false为失败
 // o 忽略
```

设备控制

设备控制是基于物的模型对设备定义的属性、事件、服务进行操作。关于属性、事件、服务的描述，请参见 生成物的模型TSL。

- 获取设备状态

```
panelDevice.getStatus(new IPanelCallback() {
            @Override
            public void onComplete(boolean bSuc, Object o) {
               ALog.d(TAG,"getStatus(), request complete," + bSuc);
               JSONObject data = new JSONObject((String)o);
            }
        });
  // bSuc表示是否获取成功，true为成功，false为失败
  // o 表示具体的数据,失败时是一个AError结构，成功时是json字符串格式如下
  /* {
       "code":200,
       "data":{
          "status":1
          "time":1232341455
       }
    }
    说明：status表示设备生命周期，目前有以下几个状态，
     0：未激活；1：上线；3：离线；8：禁用；time表示当前状态的开始时间
  */
```

- 获取设备属性

```
panelDevice.getProperties(new IPanelCallback() {
            @Override
            public void onComplete(boolean bSuc, Object o) {
               ALog.d(TAG,"getProps(), request complete," + bSuc);
```

```
                JSONObject data = new JSONObject((String)o);
            }
        });
    //bSuc表示是否获取成功，true为成功，false为失败
    //o表示具体的数据，失败时是一个AError结构，成功时是json字符串格式如下
    /* {
        "code":200,
        "data":{
            "WorkMode": {
              "time": 1516347450295,
              "value": 0
            }
        }
    }
    */
```

通过**panelDevice**的getProperties接口，获取到设备当前的所有属性值。

- 设置设备属性

```
 panelDevice.setProperties(paramsStr, new IPanelCallback() {
            @Override
            public void onComplete(boolean bSuc, Object o) {
                ALog.d(TAG,"setProps(), request complete,"+bSuc);
                JSONObject data = new JSONObject((String)o);
            }
        });
//paramsStr 格式参考如下:
/*
{
   "items":{
      "LightSwitch":0
   },
   "iotId":"s66CDxxxxXH000102"
}
*/
//bSuc表示是否获取成功，true为成功，false为失败
//o表示具体的数据，失败时是一个AError结构，成功时忽略
```

paramsStr是设置属性协议中**params**参数的字符串，请参见生成物的模型TSL。

- 调用服务

```
panelDevice.invokeService(paramsStr, new IPanelCallback() {
            @Override
            public void onComplete(boolean bSuc, Object o) {
                ALog.d(TAG,"callService(), request complete,"+bSuc);
                JSONObject data = new JSONObject((String)o);
            }
        });

//paramsStr 格式参考如下
/*
{
   "args":{
      "Saturation":80,
      "LightDuration":50,
      "Hue":325,
      "Value":50
   },
   "identifier":"Rhythm",
```

```
   "iotId":"s66CDxxxxItXH000102"
}
*/

//bSuc表示是否获取成功，true为成功，false为失败
//o表示具体的数据，失败时是一个AError结构，成功时忽略
```

- 订阅所有事件

```
panelDevice.subAllEvent(
      new IPanelCallback() {
         @Override
         public void onComplete(boolean bSuc, Object data) {
             ALog.d(TAG,"doTslTest data:" + data) ;
         }
      }
      ,new IPanelEventCallback() {
               @Override
               public void onNotify(String iotid,String topic, Object data) {
                  ALog.d(TAG,"onNofity(),topic = "+topic);
                  JSONObject jData = new JSONObject((String)data);
               }
      }
      ,null);
//IPanelCallback订阅成功或者失败时回调
//IPanelCallback的onComplete接口回调其中的参数
//bSuc表示是否获取成功，true为成功，false为失败
//o 表示具体的数据。失败时是一个AError结构，不会有事件回调，成功时忽略

//IPanelEventCallback在事件触发时回调
//iotid参数是设备iotid
//topic参数是回调的事件主题字符串
//Object data参数是触发事件的内容，类型为json字符串，格式参考如下
/*
{
   "params": {
    "iotId":"0300MSKL03667c544f69342a74Sv4Za4",
    "productKey":"X5eCzh6fEH7",
    "deviceName":"5gJtxDVeGAkaEztpisjX",
    "items":{
     "temperature":{
      "time":1510292697471,
      "value":30
     }
    }
 },
 "method":"thing.properties"
}
*/
```

清理缓存

账号退出时需要清理账号缓存的数据。

```
DeviceManager.getInstance().clearAccessTokenCache();
```

获取物的模型

```
panelDevice.getTslByCache(new IPanelCallback() {
      @Override
      public void onComplete(boolean bSuc, Object data) {
```

```
            ALog.d(TAG,"doTslTest data:" + data) ;
        }
    });
//bSuc表示是否获取成功，true为成功，false为失败
//data 为具体的返回数据，格式为json字符串，失败时为一个AError结构
```

开发者也可以使用云端接口来获取原始的物的模型。具体请参见 物的模型服务。

混淆配置

在proguard-rules.pro文件中，加入以下代码，排除不需要被混淆的类和方法。

```
-keep class com.aliyun.alink.linksdk.tmp.**{*;}
-keep class com.aliyun.alink.linksdk.cmp.**{*;}
-keep class com.aliyun.alink.linksdk.alcs.**{*;}
-keep class com.aliyun.iot.ble.**{*;}
-keep class com.aliyun.iot.breeze.**{*;}
```

蓝牙设备支持

蓝牙设备受连接特性的约束，往往无法直接跟生活物联网云端平台连接，因而需要借助一个网关设备来实现蓝牙设备与云端平台的连接通道。手机在这一过程中可以充当网关角色。

此部分API提供以下几方面的能力如下。

- 提供发现蓝牙设备/连接蓝牙设备的能力
- 提供连云通道，可以供蓝牙设备数据上下云
- 提供蓝牙设备控制与数据获取的能力

依赖 SDK	概述
蓝牙 Breeze SDK	是按照规范实现的手机端蓝牙 SDK，方便合作厂商在手机端快速接入蓝牙功能。Breeze SDK包含的主要功能有：设备发现连接，设备通信，加密传输，大数据传输等
移动端设备网关 SDK	移动端设备网关SDK，运行于App上的子设备网关，对于无法直连网络的子设备，如蓝牙设备，提供子设备的管理功能，如子设备添加拓扑，删除拓扑，上线，下线以及数据上下行等

- 蓝牙API依赖导入

  在设备模型SDK支持蓝牙设备时，需要导入如下的依赖。您可以在生活物联网控制台，SDK下载页面获取相关的版本（相关操作请参见#unique_68）。

  ```
  compile 'com.aliyun.alink.linksdk:lpbs-plugin-breeze:${version}'
     compile 'com.aliyun.alink.linksdk:breeze-biz:${version}'
     compile 'com.aliyun.alink.linksdk:breeze:${version}'
  ```

```
compile 'com.aliyun.alink.linksdk:ble-library:${version}'
```

- 初始化移动端设备网关SDK

  此功能模块依赖移动端设备网关SDK。使用前请先初始化该SDK。

- 使用蓝牙接入注意事项

  使用蓝牙设备前，App必须有如下权限，缺一不可。

  - 蓝牙权限
  - 蓝牙管理权限

  同时在apilevel 21（含level 21）之上的Anroid系统，须额外具有如下权限，缺一不可。

  - 低精度位置权限
  - 高精度位置权限

  说明:

  除了上述权限，在API level 21（含level 21）之上的Anroid系统上，系统必须开启位置服务，否则扫描将无法正常工作，如何开启位置服务，参见这里，需求位置权限及开启位置服务的原因，参见官方说明。

  更多介绍，请参见官方文档。

- 发现蓝牙设备

```
//discoverDevices 接口为发现本地设备接口
//第一个参数是userdata，一般设置为null
//第二个参数表示是否清理之前发现的设备。true表示清理；false表示不清理，一般使用false
//第三个参数表示发现设备多久超时，单位为毫秒
//第四个参数是过滤接口，返回true表示该设备为业务层需要的设备；返回false表示将该设备排除
//第五个参数为返现回调接口，会将本地发现的设备通过该回调接口返回

      DeviceManager.getInstance().discoverDevices(null, false, 5000, new IDiscovery
Filter() {
         @Override
         public boolean doFilter(DeviceBasicData basicData) {
            return true;
         }
      },listener);
```

- 添加绑定蓝牙设备

  蓝牙设备的绑定流程分为两个步骤。

  1. 蓝牙设备通过App去云端上线，可以使用SDK提供的API完成。

```
DevService.subDeviceAuthenLogin(params, new DevService.ServiceListener() {
         @Override
         public void onComplete(boolean isSuccess,Object bundle) {
            ALog.e(TAG,"subDeviceAuthenLogin onComplete isSuccess:" + isSuccess +
 " bundle:" + bundle);
```

```
            String productKey = null;
            String deviceName = null;
            if(bundle != null && bundle instanceof Bundle){
                Bundle resultBundle = (Bundle)bundle;
                productKey = resultBundle.getString(DevService.BUNDLE_KEY
_PRODUCTKEY);
                deviceName = resultBundle.getString(DevService.BUNDLE_KEY
_DEVICENAME);
            }
            if(boneCallback != null){
                boneCallback.success(getRspObject(isSuccess,productKey,deviceName
));
            }
        }
    });


//params 参数是子设备信息，参考格式如下：
/*
{
   "iotId":"your iotid",
   "productKey":"your productkey",
   "deviceName":"your deviceName "
}
*/

//ServiceListener 是回调接口，
//isSuccess表示是否成功，true表示成功，false表示失败
//bundle是回调的扩展参数，包含设备的一些信息
```

2. 上线成功后，调用云端API去做绑定账号。绑定成功后通知SDK已经绑定成功。

```
IoTCallback callback = new IotCallback(){
  @Override
  void onFailure(IoTRequest var1, Exception var2){
  }

  @Override
  void onResponse(IoTRequest var1, IoTResponse var2){
     SubDevInfo subDevInfo = new
 SubDevInfo(deviceInfo.productKey,deviceInfo.deviceName);
     DevService.notifySubDeviceBinded();
  }
}

Map<String, Object> params = new HashMap();
params.put("deviceName", deviceInfo.deviceName);
params.put("productKey", deviceInfo.productKey);
IoTRequest request = (new
 IoTRequestBuilder()).setApiVersion("1.0.3").setAuthType("iotAuth").setPath("/awss/
time/window/user/bind").setParams(params).build();
IoTAPIClient ioTAPIClient = (new IoTAPIClientFactory()).getClient();
ioTAPIClient.send(request, callback);


//api 网关请求，接口需要设备的productKey，deviceName
//具体请参见
```

API通道SDK

- 连接/断开本地的蓝牙设备

  此SDK相关功能在初始化后成功后，本地链路已经建立成功，不需要上层主动调用链接和断开接口。在一段时间后，遇见设备断开链接后，可以通过本地连接接口再次建立链接。

  1. 建立本地链接

```
panelDevice.startLocalConnect(new IPanelCallback() {
  @Override
  public void onComplete(boolean bSuc, Object o) {
  ALog.d(TAG,"startLocalConnect, onComplete,"+bSuc);
  }
});
// bSuc 表示初始化结果，true为成功，false为失败
// o 为扩展参数，成功时忽略，失败时是一个AError结构
```

  2. 断开本地链接

```
panelDevice.stopLocalConnect(new IPanelCallback() {
  @Override
  public void onComplete(boolean bSuc, Object o) {
  ALog.d(TAG,"stopLocalConnect, onComplete,"+bSuc);
  }
});
// bSuc 表示初始化结果，true为成功，false为失败
// o 为扩展参数，成功时忽略，失败时是一个AError结构
```

- 控制蓝牙设备

  蓝牙设备的控制以及信息获取跟WiFi设备的API一致，参照本文档前部分的内容。

3.9.12 蓝牙OTA SDK

SDK提供蓝牙OTA业务的App端解决方案，提供了蓝牙设备固件升级的能力。

依赖 SDK	概述
蓝牙	Breeze SDK是按照规范实现的手机端蓝牙SDK，方便合作厂商在手机端快速接入蓝牙功能。Breeze SDK 包含的主要功能有：设备发现连接，设备通信，加密传输，大数据传输等。
API通道	提供API通道能力，和基础环境配置信息。

初始化

按照以下步骤进行初始化。

1. 通过android蓝牙SDK连接上设备，获取IBreezeDevice实例。
2. 通过IBreezeDevice实例，`BreezeHelper.getDeviceInfo()`获取**deviceinfo**。

```
getDeviceInfo(breezeDevice,new IDeviceInfoCallback(){
  void onDeviceInfo(DeviceInfo info){
    mInfo = info;
```

```
};
```

3. 通过DeviceInfoBreezeHelper.bindBreezeDevice()绑定到当前用户，获取**iotID**。

```
BreezeHelper.bindBreezeDevice(breezeDevice, mInfo, new BindCallback(){
    void onBindResult(BindResult result, int error){
    if (null == error){
      mIotId = result.iotId;
  }
});
```

使用方式

1. 在生活物联网后台配置OTA任务，参见蓝牙连接开发指南。
2. 获取OTA实例。

   获取OTA对象实例，并初始化（注意不再使用的时候务必释放资源），device是breeze连接对象。

```
mBusiness = Factory.create(device);
      mBusiness.init();
```

3. 检查有无可以更新的固件。

   检查是否有新更新，mIotId在成功绑定时获取会返回。

```
mBusiness.inquiryNewVersion(mIotId, new ILinkOTABusiness.IInquiryNewVersionCa
llback() {
                @Override
                public void onResult(boolean needUpgrade, String error, String result) {
                    // 提示用户升级
                }
            });
```

4. 开始OTA升级。

   如果有新的OTA固件可以升级，且用户同意升级，调用下列接口开始升级，升级进度通过IOtaListener回调。

```
mBusiness.startUpgrade(iotId, false, ILinkOTABusiness.DEVICE_TYPE_BLE, new
ILinkOTABusiness.IOtaListener() {
            @Override
            public void onNotification(int type, ILinkOTABusiness.IOtaError error) {
```

```
Log.d(TAG, "onNotification type:" + type + " error:" + error);
```

5. 停止OTA升级。

   但OTA结束后（失败或成功）或者用户向终止OTA，调用下列接口停止升级。

```
mBusiness.stopUpgrade();
```

6. 释放OTA资源。

   当不需要OTA时，务必释放OTA资源。

```
mBusiness.deInit();
```

3.9.13 Link Visual视频Media SDK

本节为Link Visual音视频模块使用说明，包含视频播放、语音对讲的功能。

依赖SDK	概述
API通道	提供API通道能力
长连通道	P2P需要长连接通道

初始化

在初始化Link Visual视频Media SDK前，需要正确集成安全图片。详细请参见#unique_103。

依赖引入

```
// 1. 根build.gradle添加对aliyun maven仓库的引用
allprojects {
    repositories {
        maven {
            url "http://maven.aliyun.com/nexus/content/repositories/releases"
        }
    }
}

// 2. app build.gradle中添加依赖
implementation('com.aliyun.iotx:linkvisual-media:1.2.7.2')
```

混淆配置

```
# keep and don't warn linkvisual

-dontwarn com.aliyun.iotx.linkvisual.**
-dontwarn com.google.android.exoplayer2.**

-keep class com.aliyun.iotx.linkvisual.media.** { *; }
```

使用方式

视频播放器按功能分为三种。

- 直播播放器
  - 用于RTMP直播源，具有时延低的特点。
  - 支持P2P（需使用接入Link Visual设备端SDK的摄像头）。
- 设备录像点播播放器

  用于设备录像回放的播放，可调整播放进度。
- HLS播放器
  - 用于基于HLS的云端录像回放的播放，支持MPEG-TS和FMP4容器，AES-128加密方式。
  - 提供两种HLS播放器供选择：自研HLS播放器和基于ExoPlayer封装的播放器。

功能	直播播放器	设备录像点播播放器	HLS播放器（自研）	HLS播放器（ExoPlayer）
视频播放	#	#	#	#
音频播放	#	#	#	#
暂停/恢复	x	#	#	#
跳至指定位置播放	x	#	#	#
总时长	x	#	#	#
当前播放进度	x	#	#	#
播放器状态变更通知	#	#	#	#
设置播放音量	#	#	#	#
变速播放	x	x	#	#
循环播放	x	x	x	#
画面缩放模式设置	#	#	#	#
播放器截图	#	#	#	#
截图到文件	#	#	#	x
边播边录	#	#	#	x
提供YUV数据	#	#	#	x
提供SEI数据	#	#	#	x

播放器的使用样例代码如下：

- 直播播放器

```
// 构造播放器实例
LivePlayer player = new LivePlayer(getApplicationContext());
```

```
// 设置textureview
player.setTextureView(textureView);
// 设置必要的状态监听
player.setOnPlayerStateChangedListener(new OnPlayerStateChangedListener() {
    @Override
    public void onPlayerStateChange(int playerState) {
        Log.d(TAG, "play state= " + playerState);
        switch (playerState) {
            case Player.STATE_BUFFERING:
                break;
            case Player.STATE_IDLE:
                break;
            case Player.STATE_READY:
                break;
            case Player.STATE_ENDED:
                break;
            default:
                break;
        }
    }
});
// 设置错误监听
player.setOnErrorListener(new OnErrorListener() {
    @Override
    public void onError(PlayerException exception) {
        makeToast("errorcode: " + exception.getCode() + "\n" + exception.getMessage());
    }
});

// 设置rtmp地址
player.setDataSource("rtmp://58.200.131.2:1935/livetv/hunantv");
// 设置数据源就绪监听器
player.setOnPreparedListener(new OnPreparedListener() {
    @Override
    public void onPrepared() {
        // 数据源就绪后开始播放
        player.start();
    }
});
player.prepare();
...
// 停止播放
player.stop();
...
// 释放播放器资源
player.release();
```

- 设备录像点播播放器

```
// 构造播放器实例
VodPlayer player = new VodPlayer(getApplicationContext());

// 设置textureview
player.setTextureView(textureView);
// 设置必要的状态监听
player.setOnPlayerStateChangedListener(new OnPlayerStateChangedListener() {
    @Override
    public void onPlayerStateChange(int playerState) {
        Log.d(TAG, "play state= " + playerState);
        switch (playerState) {
            case Player.STATE_BUFFERING:
                break;
```

```
            case Player.STATE_IDLE:
                break;
            case Player.STATE_READY:
                break;
            case Player.STATE_ENDED:
                break;
            default:
                break;
        }
    }
});
// 设置错误监听
player.setOnErrorListener(new OnErrorListener() {
    @Override
    public void onError(PlayerException exception) {
        makeToast("errorcode: " + exception.getCode() + "\n" + exception.getMessage());
    }
});

// 设置支持点播的rtmp地址
player.setDataSource("rtmp://xxxx");
// 设置数据源就绪监听器
player.setOnPreparedListener(new OnPreparedListener() {
    @Override
    public void onPrepared() {
        // 数据源就绪后开始播放
        player.start();
    }
});
player.prepare();
...
// 暂停播放
player.pause();
...
// 恢复播放
player.start();
...
// 停止播放
player.stop();
...
// 释放播放器资源
player.release();
```

- HLS播放器（ExoPlayer）

```
ExoHlsPlayer player = new ExoHlsPlayer(getApplicationContext());
// 设置textureview
player.setTextureView(textureView);
// 也可以使用Exo的SimpleExoPlayerView来作为播放器的UI组件
// simpleExoPlayerView.setPlayer(player.getExoPlayer());
// simpleExoPlayerView.requestFocus();

// 设置错误监听
player.setOnErrorListener(new OnErrorListener() {
    @Override
    public void onError(PlayerException exception) {
        makeToast("errorcode: " + exception.getCode() + "\n" + exception.getMessage());
    }
});
// 设置状态监听
player.setOnPlayerStateChangedListener(new OnPlayerStateChangedListener() {
    @Override
    public void onPlayerStateChange(int playerState) {
```

```
        switch (playerState) {
            case Player.STATE_BUFFERING:
                break;
            case Player.STATE_IDLE:
                break;
            case Player.STATE_READY:
                break;
            case Player.STATE_ENDED:
                break;
            default:
                break;
        }
    }
});

// 设置m3u8地址
player.setDataSource("http://devimages.apple.com.edgekey.net/streaming/
examples/bipbop_4x3/gear3/prog_index.m3u8");
// 设置数据源就绪监听器
player.setOnPreparedListener(new OnPreparedListener() {
    @Override
    public void onPrepared() {
        // 数据源就绪后开始播放
        player.start();
    }
});
player.prepare();
...
// 暂停播放
player.pause();
...
// 恢复播放
player.start();
...
// 停止播放
player.stop();
...
// 释放播放器资源
player.release();
...
// 截图
textureview.getBitmap();
```

- HLS播放器（自研）

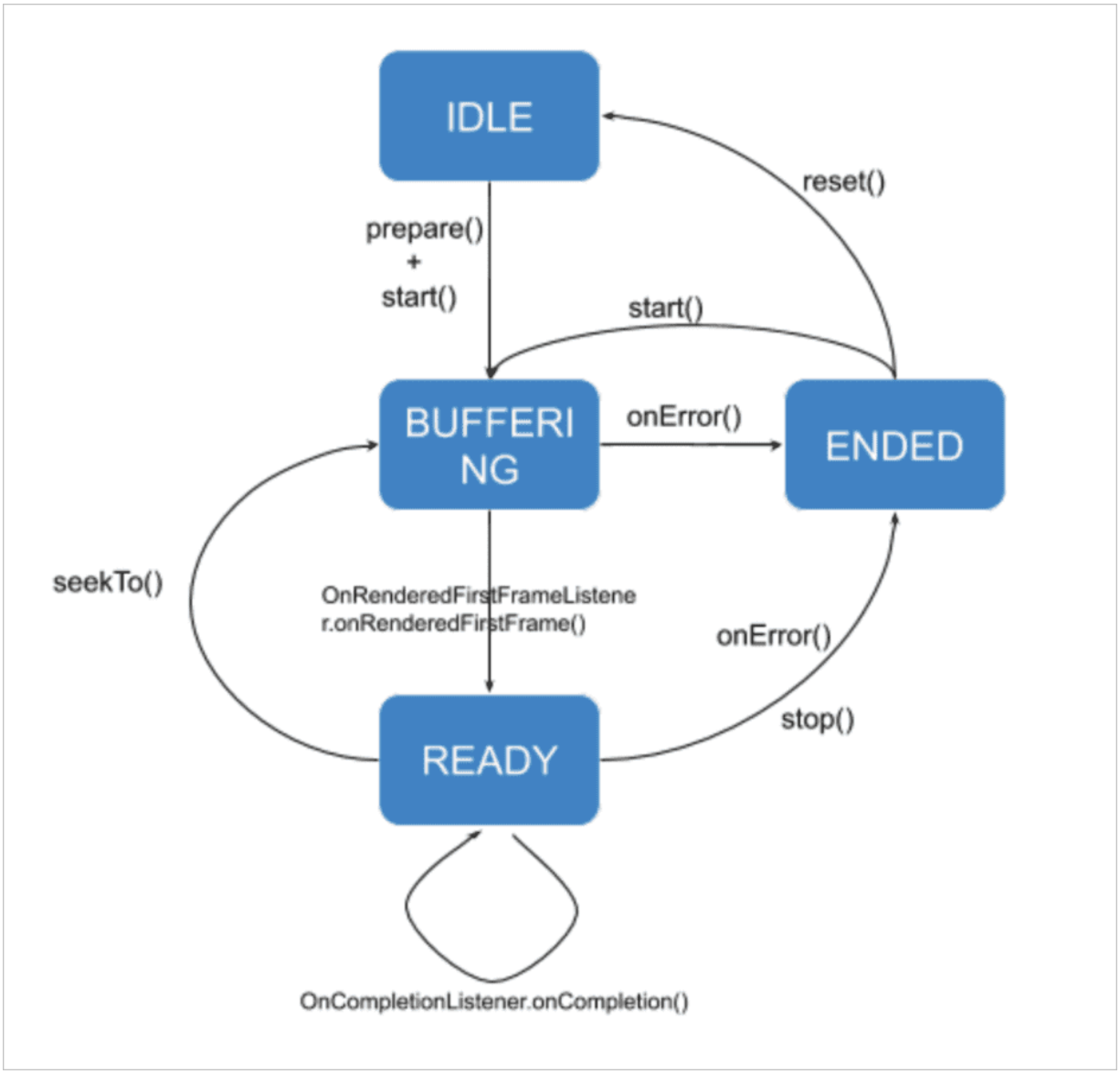
```
HlsPlayer player = new HlsPlayer(getApplicationContext());
// 设置textureview
player.setTextureView(textureView);

// 设置错误监听
player.setOnErrorListener(new OnErrorListener() {
    @Override
    public void onError(PlayerException exception) {
        makeToast("errorcode: " + exception.getCode() + "\n" + exception.getMessage());
    }
});
// 设置状态监听
player.setOnPlayerStateChangedListener(new OnPlayerStateChangedListener() {
    @Override
    public void onPlayerStateChange(int playerState) {
        switch (playerState) {
            case Player.STATE_BUFFERING:
                break;
```

```
            case Player.STATE_IDLE:
                break;
            case Player.STATE_READY:
                break;
            case Player.STATE_ENDED:
                break;
            default:
                break;
        }
    }
});

// 设置m3u8地址
player.setDataSource("http://devimages.apple.com.edgekey.net/streaming/
examples/bipbop_4x3/gear3/prog_index.m3u8");
// 设置数据源就绪监听器
player.setOnPreparedListener(new OnPreparedListener() {
    @Override
    public void onPrepared() {
        // 数据源就绪后开始播放
        player.start();
    }
});
player.prepare();
...
// 暂停播放
player.pause();
...
// 恢复播放
player.start();
...
// 停止播放
player.stop();
...
// 释放播放器资源
player.release();
...
```

播放器状态介绍

通过设置播放器状态监听器，可接收到状态变更事件，用于相关UI元素的变更。

状态变更事件如：播放中发生错误、主动停止播放等。



- IDLE：播放器没有任何内容播放时的状态。
- BUFFERING：播放器正在缓冲，当前的位置还不可以播放。状态变更事件如：开始播放时缓冲、seek后重新缓冲。
- READY：播放器已经有内容在播放。状态变更事件如：首帧数据已经渲染、seek缓冲完成开始播放新内容。对于点播播放器，若已seek或播放到文件结尾，则会回调OnCompletionListener.onCompletion()方法，状态不会切换到ENDED。
- ENDED：播放器已结束播放。播放出错或stop后会切换到该状态。

接口说明

- LivePlayer
  - 构造方法

```
/**
 * 构造方法
```

```
   * @param applicationContext ApplicationContext
   */
  LivePlayer(Context applicationContext);
```

- 设置非加密播放源

```
/**
   * 设置播放源
   * @param url RTMP地址
   */
  void setDataSource(String url) throws IllegalArgumentException;
```

- 设置加密播放源

```
/**
   * 设置加密播放源（请确保源是做过AES加密）
   * @param url         rtmp源地址
   * @param isEncrypted 是否是加密源
   * @param decryptIv   解密向量，16 byte array
   * @param decryptKey  解密密钥，16 byte array
   * @throws IllegalArgumentException
   */
  void setDataSource(String url, boolean isEncrypted, byte[] decryptIv, byte[]
decryptKey) throws IllegalArgumentException;
```

- 设置IPC直播数据源

```
/**
   * 设置IPC直播数据源.
   * @param iotId           IPC设备的iotId
   * @param streamType      流的类型：C.STREAM_TYPE_MAJOR0-主码流，C.
STREAM_TYPE_MINOR-辅码流，若有多路码流请关注该参数，否则请填0
   * @param relayEncrypted   云转是否加密，强烈建议开启，对于出海产品，务必保持开
启
   * @param relayEncryptType 云转加密类型：目前只支持C.ENCRYPTE_AES_128(AES-
128)加密方式
   * @param forceIFrame      是否需要强制I帧，默认开启
   */
   void setIPCLiveDataSource(String iotId, int streamType, boolean relayEncrypted,
int relayEncryptType, boolean forceIFrame);
```

- 校验和准备数据

```
/**
   * 校验和准备数据
   */
  void prepare();
```

- 开始播放视频

```
/**
   * 开始播放视频
   */
  void start();
```

- 停止播放

```
/**
   * 停止播放
   */
```

```
  void stop();
```

- 重置播放器

```
/**
  * 重置播放器
  */
  void reset();
```

- 释放播放器资源

```
/**
  * 释放播放器资源
  */
  void release();
```

- 设置重连次数

```
   /**
    * 设置重连次数
    * 只针对发生{@link PlayerException#SUB_CODE_UNEXPECTED_PULL_STREA
M_ERROR}错误时做重连
    * @param count 默认为0，建议不超过3次
    */
   void setReconnectCount(int count);
```

- 截图

```
/**
  * 当前视频画面截图
  * @return 如果当前无画面则返回 null
  */
  Bitmap snapShot();
```

- 截图到文件

```
/**
    * 当前视频画面截图
    * @param jpegFile 保存截图jpeg的文件
    * @return true - 文件保存成功
    */
   boolean snapShotToFile(File jpegFile);
```

- 开始录屏

```
/**
  * 开始录制当前播放内容，生成MPEG-4格式转存到指定的文件中
  * 文件名后缀必须为.mp4
  * 须在{@link PlayerState#STATE_READY}时调用有效
  * @param contentFile
  * @return 操作成功与否
  */
  boolean startRecordingContent(File contentFile) throws IOException;
```

- 获取当前录制的视频时长

```
   /**
    * 获取当前录制的视频时长
    * @return 单位ms
    */
```

```
    long getCurrentRecordingContentDuration();
```

- 结束录屏

```
/**
  * 停止记录播放内容
  * @return 操作成功与否
  */
 boolean stopRecordingContent();
```

- 设置播放器音量

```
/**
  * 设置播放器音量
  * @param audioVolume 范围 0-1，0为静音
  */
 void setVolume(float audioVolume);
```

- 设置音频流通道类型

```
/**
  * 设置音频流通道类型，see {@link android.media.AudioManager}，默认使用
AudioManager.STREAM_MUSIC
  * 如果音频正在播放，则会因为重新创建AudioTrack导致有短暂停顿
  * @param audioStreamType
  */
 void setAudioStreamType(int audioStreamType);
```

- 设置画面缩放模式

```
/**
  * 设置视频画面缩放模式，默认为{@link android.media.MediaCodec#VIDEO_SCAL
ING_MODE_SCALE_TO_FIT_WITH_CROPPING}
  * @param videoScalingMode 参见：
  * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FIT}
  * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FI
T_WITH_CROPPING}
  */
 void setVideoScalingMode(int videoScalingMode);
```

- 设置播放停止时画面绘制策略

```
/**
   * 设置播放停止时画面绘制策略
   * @param playerStoppedDrawingMode
   * ALWAYS_KEEP_LAST_FRAME        播放停止时始终保留最后一帧画面
   * KEEP_LAST_FRAME_WITHOUT_ERROR   播放停止时只有未出现错误时才保留最后一
帧画面（默认是此模式）
   * ALWAYS_BLACK                 播放停止时始终显示黑色
   */
  void setPlayerStoppedDrawingMode(PlayerStoppedDrawingMode playerStop
pedDrawingMode)
```

- 设置播放器固定缓存帧数

```
/**
  * 设置播放器固定缓存帧数
  * @param frameCount 播放器固定缓存帧数，范围0-16帧，数值越大播放器延迟越
大，流畅性越好，默认5帧
  */
```

```
void setBufferedFrameCount(int frameCount);
```

- 设置surfaceview

```
/**
    * 设置SurfaceView, 必须为GLSurfaceView，同一个Window中同时只允许一个播放器播放，若需要支持多播放实例，请使用TextureView
    * 注意:GLSurfaceView必须在Activity的onResume和onPause回调方法中调用GLSurfaceView的onResume和onPause方法;
    * GLSurfaceView及其容器不可设置背景色.
    * @param surfaceview
    */
 void setSurfaceView(SurfaceView surfaceview);
```

- 清除surfaceview

```
/**
  * 清除surfaceview
  */
 void clearSurfaceView();
```

- 设置textureview

```
/**
    * 设置TextureView，同一个TextureView不可被多个播放器共享，提供ZoomableTextureView带手势放缩处理可做参照
    * @param textureview
    */
   void setTextureView(TextureView textureView);
```

- 清除textureview

```
/**
    * 清除textureview
    */
   void clearTextureView();
```

- 设置是否由外部来完成YUV数据的渲染

```
   /**
    * 设置是否由外部来完成YUV数据的渲染
    * @param useExternalRender true为由外部渲染，false为由内部渲染
    */
   void setUseExternalRender(boolean useExternalRender);
```

- 获取是否使用外部渲染

```
   /**
    * 获取是否使用外部渲染
    * @return true为由外部渲染，false为由内部渲染
    */
   boolean useExternalRender();
```

- 设置外部渲染监听器

```
   /**
    * 设置外部渲染监听器
    * @param onExternalRenderListener
    */
```

```
    void setOnExternalRenderListener(OnExternalRenderListener onExternal
RenderListener);public interface OnExternalRenderListener {

public interface OnExternalRenderListener {
    /**
     * 通知已经有YUV数据帧需要被渲染
     * @param width 视频帧宽
     * @param height 视频帧高
     * @param timestamp 时间戳，单位为ms
     */
    void onVideoFrameUpdate(int width, int height ,long timestamp);
}
```

- 获取YUV帧数据用于渲染

```
    /**
     * 获取YUV帧数据用于渲染，只在外部渲染模式下有效
     * @return Yuv420p格式视频帧
     */
    Yuv420pFrame getYuvFrame();
```

- 设置SEI信息监听器

```
    /**
     * 设置SEI信息监听器
     *
     * @param seiInfoBuffer  创建用于存储SEI帧数据的缓存，必须保证码流中的SEI信息长
度不会超过缓存大小，超过的帧将会被丢弃
     * @param onSeiInfoListener
     */
    void setOnSeiInfoListener(SeiInfoBuffer seiInfoBuffer, OnSeiInfoListener
onSeiInfoListener);

interface OnSeiInfoListener {
    /**
     * 当SEI信息更新时回调
     * 请不要执行阻塞的操作，并在该回调中及时处理seiInfoBuffer数据
     * @param seiInfoBuffer
     */
    void onSeiInfoUpdate(SeiInfoBuffer seiInfoBuffer);
}
```

- 设置数据源准备就绪事件监听器

```
/**
   * 设置数据源准备就绪事件监听器
   * @param listener
   */
  void setOnPreparedListener(OnPreparedListener listener);

public interface OnPreparedListener {
    /**
     * 数据源准备就绪回调
     */
    void onPrepared();
}
```

- 设置播放器错误事件监听器

```
/**
   * 设置播放器错误事件监听器，错误类型参见：
```

```
   * {@link PlayerException.SOURCE_ERROR}
   * {@link PlayerException.RENDER_ERROR}
   * {@link PlayerException.UNEXPECTED_ERROR}
   * @param listener
   */
  void setOnErrorListener(OnErrorListener listener);

 public interface OnErrorListener {
    /**
     * 播放器错误回调，参见
     * {@link PlayerException}
     * @param exception
     */
    void onError(PlayerException exception);
}
```

- 设置播放状态变更事件监听器

```
/**
  * 设置播放状态变更事件监听器，请参见状态图
  * @param listener
  */
 void setOnPlayerStateChangedListener(OnPlayerStateChangedListener listener);

public interface OnPlayerStateChangedListener {
    /**
     * 播放器状态变更回调
     *
     * @param playerState 参见
     *                {@link PlayerState#STATE_IDLE}
     *                {@link PlayerState#STATE_BUFFERING}
     *                {@link PlayerState#STATE_READY}
     *                {@link PlayerState#STATE_ENDED}
     */
    void onPlayerStateChange(int playerState);
}
```

- 设置首帧被渲染事件监听器

```
/**
  * 设置首帧被渲染事件监听器
  * @param listener
  */
 void setOnRenderedFirstFrameListener(OnRenderedFirstFrameListener listener);

public interface OnRenderedFirstFrameListener {
    /**
     * 首帧被渲染回调
     */
    void onRenderedFirstFrame();
}
```

- 设置视频内容宽高变更回调

```
    /**
     * 设置视频内容宽高变更回调
     * @param listener
     */
    void setOnVideoSizeChangedListener(OnVideoSizeChangedListener listener);
public interface OnVideoSizeChangedListener {

    /**
```

```
     * 内容宽高改变时的回调
     *
     * @param width  视频内容宽，单位像素
     * @param height 视频内容高，单位像素
     */
    void onVideoSizeChanged(int width, int height);
}
```

- 获取音量

```
/**
  * 获取音量
  * @return 范围0-1，0为静音
  */
 float getVolume();
```

- 获取播放器状态

```
/**
  * 获取播放状态
  * @return 状态枚举：
  * {@link PlayerState#STATE_IDLE} 播放器初始状态
  * {@link PlayerState#STATE_BUFFERING} 缓冲中状态
  * {@link PlayerState#STATE_READY} 缓冲结束开始播放状态
  * {@link PlayerState#STATE_ENDED} 播放完成状态
  */
 int getPlayState();
```

- 获取播放器当前流的连接类型

```
/**
  * 获取播放器当前流的连接类型
  * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
  *
  * @return {@link StreamConnectType}
  */
 StreamConnectType getStreamConnectType();
```

- 获取播放器当前的帧率/码率等信息

```
/**
  * 获取播放器当前的帧率/码率等信息
  * 播放器状态为{@link PlayerState#STATE_BUFFERING}及{@link PlayerState#
STATE_READY}时调用有效
  * @return 包含帧率/码率等信息的json string
  */
 PlayInfo getCurrentPlayInfo();
```

- VodPlayer
  - 构造方法

```
/**
  * 构造方法
  * @param applicationContext ApplicationContext
  */
```

```
VodPlayer(Context applicationContext);
```

- 设置非加密播放源

```
/**
 * 设置播放源
 * @param url RTMP地址
 */
void setDataSource(String url) throws IllegalArgumentException;
```

- 设置加密播放源

```
/**
 * 设置加密播放源（请确保源是做过AES加密）
 * @param url         rtmp源地址
 * @param isEncrypted 是否是加密源
 * @param decryptIv   解密向量，16 byte array
 * @param decryptKey  解密密钥，16 byte array
 * @throws IllegalArgumentException
 */
void setDataSource(String url, boolean isEncrypted, byte[] decryptIv, byte[]
decryptKey) throws IllegalArgumentException;
```

- 设置播放源为IPC录像（按录像文件名）

```
/**
 * 设置播放源为IPC录像（按录像文件名）
 *
 * @param iotId        设备iotId
 * @param fileName     录像文件名
 * @param encrypted    是否流需要加密，强烈建议开启，对于出海产品，务必保持开启
 * @param encryptType  加密类型：目前只支持C.ENCRYPTE_AES_128(AES-128)加密
方式
 */
void setDataSourceByIPCRecordFileName(String iotId, String fileName, boolean
encrypted, int encryptType);
```

- 设置播放源为IPC录像（按录像时间段）

```
/**
 * 设置播放地址为已接入生活物联网平台的IPC设备，指定录像时间段的本地录像文件地
址，播放偏移量为0
 * see {@link #setDataSourceByIPCRecordTime(String, int, int, boolean, int, long)}
 */
void setDataSourceByIPCRecordTime(String iotId, String beginTime, String
endTime, boolean encrypted, int encryptType);

/**
 * 设置播放地址为已接入生活物联网平台的IPC设备，指定录像时间段的本地录像文件地址
 *
 * @param iotId            设备iotId
 * @param beginTimeInS     录像时间段开始时间，UTC时间，1970年1月1日开始的秒
数
 * @param endTimeInS       录像时间段结束时间，UTC时间，1970年1月1日开始的秒数
 * @param encrypted        是否流需要加密，强烈建议开启，对于出海产品，务必保持开
启
 * @param encryptType      加密类型：目前只支持C.ENCRYPTE_AES_128(AES-128)加
密方式
 * @param seekToPositionInMs 相对于beginTimeInS的播放偏移量，单位为ms
 */
```

```
void setDataSourceByIPCRecordTime(String iotId, int beginTimeInS, int endTimeInS
, boolean encrypted, int encryptType, long seekToPositionInMs)
```

- 校验和准备数据

```
/**
 * 校验和准备数据
 */
void prepare();
```

- 开始或恢复播放视频

```
/**
 * 开始播放或恢复播放视频
 */
void start();
```

- 暂停播放

```
/**
 * 暂停播放，调用start()恢复播放
 */
void pause();
```

- seek到指定位置

```
/**
 * seek到指定位置
 * @param position 毫秒
 */
void seekTo(long positionInMs);
```

- 停止播放

```
/**
 * 停止播放
 */
void stop();
```

- 重置播放器

```
/**
 * 重置播放器
 */
void reset();
```

- 释放播放器资源

```
/**
 * 释放播放器资源
 */
void release();
```

- 截图

```
/**
 * 当前视频画面截图
 * @return 如果当前无画面则返回 null
 */
```

```
  Bitmap snapShot();
```

- 截图到文件

```
/**
    * 当前视频画面截图
    * @param jpegFile 保存截图jpeg的文件
    * @return true - 文件保存成功
    */
   boolean snapShotToFile(File jpegFile);
```

- 开始录屏

```
/**
  * 开始录制当前播放内容，生成MPEG-4格式转存到指定的文件中
  * 文件名后缀必须为mp4
  * 须在{@link PlayerState#STATE_READY}时调用有效
  * @param contentFile
  * @return 操作成功与否
  */
 boolean startRecordingContent(File contentFile) throws IOException;
```

- 获取当前录制的视频时长

```
   /**
    * 获取当前录制的视频时长
    * @return 单位ms
    */
   long getCurrentRecordingContentDuration();
```

- 结束录屏

```
/**
  * 停止记录播放内容
  * @return 操作成功与否
  */
 boolean stopRecordingContent();
```

- 设置播放器音量

```
/**
  * 设置播放器音量
  * @param audioVolume 范围 0-1，0为静音
  */
 void setVolume(float audioVolume);
```

- 设置音频流通道类型

```
/**
  * 设置音频流通道类型，参见{@link android.media.AudioManager}
  * 如果音频正在播放，则会因为重新创建AudioTrack导致有短暂停顿
  * @param audioStreamType
  */
 void setAudioStreamType(int audioStreamType);
```

- 设置画面缩放模式

```
/**
  * 设置视频画面缩放模式，默认为{@link android.media.MediaCodec#VIDEO_SCAL
ING_MODE_SCALE_TO_FIT_WITH_CROPPING}
```

```
   * @param videoScalingMode 参见:
   * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FIT}
   * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FI
T_WITH_CROPPING}
   */
  void setVideoScalingMode(int videoScalingMode);
```

- 设置播放停止时画面绘制策略

```
/**
    * 设置播放停止时画面绘制策略
    * @param playerStoppedDrawingMode
    * ALWAYS_KEEP_LAST_FRAME         播放停止时始终保留最后一帧画面
    * KEEP_LAST_FRAME_WITHOUT_ERROR   播放停止时只有未出现错误时才保留最后一
帧画面（默认是此模式）
    * ALWAYS_BLACK                  播放停止时始终显示黑色
    */
   void setPlayerStoppedDrawingMode(PlayerStoppedDrawingMode playerStop
pedDrawingMode)
```

- 设置surfaceview

```
/**
    * 设置SurfaceView，必须为GLSurfaceView，同一个Window中同时只允许一个播放器
播放，若需要支持多播放实例，请使用TextureView
    * 注意GLSurfaceView必须在Activity的onResume和onPause回调方法中调用
GLSurfaceView的onResume和onPause方法
    * GLSurfaceView及其容器不可设置背景色
    * @param surfaceview
    */
  void setSurfaceView(SurfaceView surfaceview);
```

- 清除surfaceview

```
/**
   * 清除surfaceview
   */
  void clearSurfaceView();
```

- 设置textureview

```
/**
    * 设置TextureView，同一个TextureView不可被多个播放器共享，提供ZoomableTe
xtureView带手势放缩处理可做参照
    * @param textureview
    */
   void setTextureView(TextureView textureView);
```

- 清除textureview

```
/**
    * 清除textureview
    */
   void clearTextureView();
```

- 设置是否由外部来完成YUV数据的渲染

```
    /**
    * 设置是否由外部来完成YUV数据的渲染
    * @param useExternalRender true为由外部渲染，false为由内部渲染
```

```
    */
    void setUseExternalRender(boolean useExternalRender);
```

- 获取是否使用外部渲染

```
    /**
     * 获取是否使用外部渲染
     * @return true为由外部渲染，false为由内部渲染
     */
    boolean useExternalRender();
```

- 设置外部渲染监听器

```
    /**
     * 设置外部渲染监听器
     * @param onExternalRenderListener
     */
    void setOnExternalRenderListener(OnExternalRenderListener onExternal
RenderListener);public interface OnExternalRenderListener {

public interface OnExternalRenderListener {
    /**
     * 通知已经有YUV数据帧需要被渲染
     * @param width 视频帧宽
     * @param height 视频帧高
     * @param timestamp 时间戳，单位为ms
     */
    void onVideoFrameUpdate(int width, int height ,long timestamp);
}
```

- 获取YUV帧数据用于渲染

```
    /**
     * 获取YUV帧数据用于渲染，只在外部渲染模式下有效
     * @return Yuv420p格式视频帧
     */
    Yuv420pFrame getYuvFrame();
```

- 设置SEI信息监听器

```
    /**
     * 设置SEI信息监听器
     *
     * @param seiInfoBuffer  创建用于存储SEI帧数据的缓存，必须保证码流中的SEI信息长
度不会超过缓存大小，超过的帧将会被丢弃
     * @param onSeiInfoListener
     */
    void setOnSeiInfoListener(SeiInfoBuffer seiInfoBuffer, OnSeiInfoListener
onSeiInfoListener);

interface OnSeiInfoListener {
    /**
     * 当SEI信息更新时回调
     * 请不要执行阻塞的操作，并在该回调中及时处理seiInfoBuffer数据
     * @param seiInfoBuffer
     */
    void onSeiInfoUpdate(SeiInfoBuffer seiInfoBuffer);
}
```

- 设置数据源准备就绪事件监听器

```
/**
  * 设置数据源准备就绪事件监听器
  * @param listener
  */
 void setOnPreparedListener(OnPreparedListener listener);

interface OnPreparedListener {
    /**
     * 数据源准备就绪回调
     */
    void onPrepared();
}
```

- 设置播放器错误事件监听器

```
/**
  * 设置播放器错误事件监听器，错误类型参见：
  * {@link PlayerException.SOURCE_ERROR}
  * {@link PlayerException.RENDER_ERROR}
  * {@link PlayerException.UNEXPECTED_ERROR}
  * @param listener
  */
 void setOnErrorListener(OnErrorListener listener);

interface OnErrorListener {
    /**
     * 播放器错误回调，参见
     * {@link PlayerException}
     * @param exception
     */
    void onError(PlayerException exception);
}
```

- 设置播放状态变更事件监听器

```
/**
  * 设置播放状态变更事件监听器
  * @param listener
  */
 void setOnPlayerStateChangedListener(OnPlayerStateChangedListener listener);

interface OnPlayerStateChangedListener {
    /**
     * 播放器状态变更回调
     *
     * @param playerState 参见
     *                    {@link PlayerState#STATE_IDLE}
     *                    {@link PlayerState#STATE_BUFFERING}
     *                    {@link PlayerState#STATE_READY}
     *                    {@link PlayerState#STATE_ENDED}
     */
    void onPlayerStateChange(int playerState);
}
```

- 设置首帧被渲染事件监听器

```
/**
  * 设置首帧被渲染事件监听器
```

```
   * @param listener
   */
  void setOnRenderedFirstFrameListener(OnRenderedFirstFrameListener listener);

 interface OnRenderedFirstFrameListener {
    /**
     * 首帧被渲染回调
     */
    void onRenderedFirstFrame();
 }
```

- 设置视频内容宽高变更回调

```
    /**
     * 设置视频内容宽高变更回调
     * @param listener
     */
    void setOnVideoSizeChangedListener(OnVideoSizeChangedListener listener);

 interface OnVideoSizeChangedListener {

    /**
     * 内容宽高改变时的回调
     *
     * @param width  视频内容宽，单位像素
     * @param height 视频内容高，单位像素
     */
    void onVideoSizeChanged(int width, int height);
 }
```

- 设置播放到内容结束事件监听器

```
 /**
   * 设置播放到内容结束事件监听器
   * 收到该事件后，需要调用stop，播放器状态才会切到STATE_END
   * @param listener
   */
  void setOnCompletionListener(OnCompletionListener listener);

 interface OnCompletionListener {
    /**
     * 播放到文件末尾回调
     */
    void onCompletion();
 }
```

- 获取当前播放进度

```
 /**
   * 获取当前播放进度，相对于开始位置的偏移量
   * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
   * @return 单位MS
   */
  long getCurrentPosition();
```

- 获取视频总时长

```
 /**
   * 获取视频总时长
   * 播放器状态为{@link #STATE_READY}时调用有效
   * @return 单位MS
   */
```

```
long getDuration();
```

- 设置回放速率

```
    /**
     * 设置播放速率
     * 须在{@link PlayerState#STATE_BUFFERING}及{@link PlayerState#STATE_READY}时
调用有效
     * 非1倍速率下默认关闭声音
     * @param speed 只支持1/16、1/8、1/4、1/2、1、2、4、8、16倍速
     */
    void setPlaybackSpeed(float speed)
```

- 获取音量

```
/**
 * 获取音量
 * @return 范围0-1，0为静音
 */
float getVolume();
```

- 获取播放器状态

```
/**
 * 获取播放状态
 * @return 状态枚举：
 * {@link PlayerState#STATE_IDLE} 播放器初始状态
 * {@link PlayerState#STATE_BUFFERING} 缓冲中状态
 * {@link PlayerState#STATE_READY} 缓冲结束开始播放状态
 * {@link PlayerState#STATE_ENDED} 播放完成状态
 */
int getPlayState();
```

- 获取播放器当前流的连接类型

```
/**
 * 获取播放器当前流的连接类型
 * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
 * @return {@link StreamConnectType}
 */
StreamConnectType getStreamConnectType();
```

- 获取播放器当前的帧率/码率等信息

```
/**
 * 获取播放器当前的帧率/码率等信息
 * 播放器状态为{@link PlayerState#STATE_BUFFERING}及{@link PlayerState#
STATE_READY}时调用有效
 * @return 包含帧率/码率等信息的json string
 */
PlayInfo getCurrentPlayInfo();
```

- HlsPlayer
  - 构造方法

```
/**
 * 构造方法
 * @param applicationContext ApplicationContext
 */
```

```
  HlsPlayer(Context applicationContext);
```

- 设置播放地址

```
/**
  * 设置播放源
  * @param url HLS点播地址
  */
  void setDataSource(String url) throws IllegalArgumentException;
```

- 设置播放地址为已接入生活物联网平台的IPC设备指定录像名的云端录像文件地址

```
   /**
    * 设置播放地址为已接入生活物联网平台的IPC设备指定录像名的云端录像文件地址
    *
    * @param iotId    设备iotId
    * @param fileName 录像文件名
    */
   void setDataSourceByIPCRecordFileName(String iotId, String fileName)
```

- 设置播放地址为已接入生活物联网平台的IPC设备指定录像名的云端录像文件地址

```
   /**
    * 设置播放地址为已接入生活物联网平台的IPC设备指定录像名的云端录像文件地址
    *
    * @param iotId    设备iotId
    * @param fileName 录像文件名
    * @param seekToPositionInMs 起始偏移量，单位ms
    */
   void setDataSourceByIPCRecordFileName(String iotId, String fileName, long
seekToPositionInMs)
```

- 校验和准备数据

```
/**
  * 校验和准备数据
  */
  void prepare();
```

- 开始或恢复播放视频

```
/**
  * 开始播放或恢复播放视频
  */
  void start();
```

- 暂停播放

```
/**
  * 暂停播放，调用start()恢复播放
  */
  void pause();
```

- seek到指定位置

```
/**
  * seek到指定位置
  * @param position 毫秒
  */
```

```
void seekTo(long positionInMs);
```

- 停止播放

```
/**
 * 停止播放
 */
void stop();
```

- 重置播放器

```
/**
 * 重置播放器
 */
void reset();
```

- 释放播放器资源

```
/**
 * 释放播放器资源
 */
void release();
```

- 截图

```
/**
 * 当前视频画面截图
 * @return 如果当前无画面则返回 null
 */
Bitmap snapShot();
```

- 截图到文件

```
/**
   * 当前视频画面截图
   * @param jpegFile 保存截图jpeg的文件
   * @return true - 文件保存成功
   */
  boolean snapShotToFile(File jpegFile);
```

- 开始录屏

```
/**
 * 开始录制当前播放内容，生成MPEG-4格式转存到指定的文件中
 * 文件名后缀必须为mp4
 * 须在{@link PlayerState#STATE_READY}时调用有效
 * @param contentFile
 * @return 操作成功与否
 */
boolean startRecordingContent(File contentFile) throws IOException;
```

- 获取当前录制的视频时长

```
  /**
   * 获取当前录制的视频时长
   * @return 单位ms
   */
```

```
    long getCurrentRecordingContentDuration();
```

- 结束录屏

```
/**
  * 停止记录播放内容
  * @return 操作成功与否
  */
 boolean stopRecordingContent();
```

- 设置回放速率

```
    /**
    * 设置播放速率
    * 须在{@link PlayerState#STATE_BUFFERING}及{@link PlayerState#STATE_READY}时
调用有效
    * 非1倍速率下默认关闭声音
    * @param speed 只支持1/16、1/8、1/4、1/2、1、2、4、8、16倍速
    */
   void setPlaybackSpeed(float speed)
```

- 设置播放器音量

```
/**
  * 设置播放器音量
  * @param audioVolume 范围 0-1，0为静音
  */
 void setVolume(float audioVolume);
```

- 设置音频流通道类型

```
/**
  * 设置音频流通道类型，参见{@link android.media.AudioManager}
  * 如果音频正在播放，则会因为重新创建AudioTrack导致有短暂停顿
  * @param audioStreamType
  */
 void setAudioStreamType(int audioStreamType);
```

- 设置画面缩放模式

```
/**
  * 设置视频画面缩放模式，默认为{@link android.media.MediaCodec#VIDEO_SCAL
ING_MODE_SCALE_TO_FIT_WITH_CROPPING}
  * @param videoScalingMode 参见：
  * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FIT}
  * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FI
T_WITH_CROPPING}
  */
 void setVideoScalingMode(int videoScalingMode);
```

- 设置播放停止时画面绘制策略

```
/**
    * 设置播放停止时画面绘制策略
    * @param playerStoppedDrawingMode
    * ALWAYS_KEEP_LAST_FRAME        播放停止时始终保留最后一帧画面
    * KEEP_LAST_FRAME_WITHOUT_ERROR   播放停止时只有未出现错误时才保留最后一
帧画面（默认是此模式）
    * ALWAYS_BLACK                播放停止时始终显示黑色
    */
```

```
    void setPlayerStoppedDrawingMode(PlayerStoppedDrawingMode playerStop
pedDrawingMode)
```

- 设置surfaceview

```
/**
    * 设置SurfaceView，必须为GLSurfaceView，同一个Window中同时只允许一个播放器
播放，若需要支持多播放实例，请使用TextureView
    * 注意GLSurfaceView必须在Activity的onResume和onPause回调方法中调用
GLSurfaceView的onResume和onPause方法
    * GLSurfaceView及其容器不可设置背景色
    * @param surfaceview
    */
 void setSurfaceView(SurfaceView surfaceview);
```

- 清除surfaceview

```
/**
  * 清除surfaceview
  */
 void clearSurfaceView();
```

- 设置textureview

```
/*
    * 设置TextureView，同一个TextureView不可被多个播放器共享，提供ZoomableTe
xtureView带手势放缩处理可做参照
    * @param textureview
    */
   void setTextureView(TextureView textureView);
```

- 清除textureview

```
/**
    * 清除textureview
    */
   void clearTextureView();
```

- 设置是否由外部来完成YUV数据的渲染

```
    /**
    * 设置是否由外部来完成YUV数据的渲染
    * @param useExternalRender true为由外部渲染，false为由内部渲染
    */
   void setUseExternalRender(boolean useExternalRender);
```

- 获取是否使用外部渲染

```
    /**
    * 获取是否使用外部渲染
    * @return true为由外部渲染，false为由内部渲染
    */
   boolean useExternalRender();
```

- 设置外部渲染监听器

```
    /**
    * 设置外部渲染监听器
    * @param onExternalRenderListener
    */
```

```
    void setOnExternalRenderListener(OnExternalRenderListener onExternal
RenderListener);public interface OnExternalRenderListener {

public interface OnExternalRenderListener {
    /**
     * 通知已经有YUV数据帧需要被渲染
     * @param width 视频帧宽
     * @param height 视频帧高
     * @param timestamp 时间戳，单位为ms
     */
    void onVideoFrameUpdate(int width, int height ,long timestamp);
}
```

- 获取YUV帧数据用于渲染

```
    /**
     * 获取YUV帧数据用于渲染，只在外部渲染模式下有效
     * @return Yuv420p格式视频帧
     */
    Yuv420pFrame getYuvFrame();
```

- 设置SEI信息监听器

```
    /**
     * 设置SEI信息监听器
     *
     * @param seiInfoBuffer  创建用于存储SEI帧数据的缓存，必须保证码流中的SEI信息长
度不会超过缓存大小，超过的帧将会被丢弃
     * @param onSeiInfoListener
     */
    void setOnSeiInfoListener(SeiInfoBuffer seiInfoBuffer, OnSeiInfoListener
onSeiInfoListener);

interface OnSeiInfoListener {
    /**
     * 当SEI信息更新时回调
     * 请不要执行阻塞的操作，并在该回调中及时处理seiInfoBuffer数据
     * @param seiInfoBuffer
     */
    void onSeiInfoUpdate(SeiInfoBuffer seiInfoBuffer);
}
```

- 设置数据源准备就绪事件监听器

```
/**
   * 设置数据源准备就绪事件监听器
   * @param listener
   */
  void setOnPreparedListener(OnPreparedListener listener);

interface OnPreparedListener {
    /**
     * 数据源准备就绪回调
     */
    void onPrepared();
}
```

- 设置播放器错误事件监听器

```
/**
   * 设置播放器错误事件监听器，错误类型参见：
```

```
     * {@link PlayerException.SOURCE_ERROR}
     * {@link PlayerException.RENDER_ERROR}
     * {@link PlayerException.UNEXPECTED_ERROR}
     * @param listener
     */
    void setOnErrorListener(OnErrorListener listener);

    interface OnErrorListener {
        /**
         * 播放器错误回调，参见
         * {@link PlayerException}
         * @param exception
         */
        void onError(PlayerException exception);
    }
```

- 设置播放状态变更事件监听器

```
    /**
     * 设置播放状态变更事件监听器
     * @param listener
     */
    void setOnPlayerStateChangedListener(OnPlayerStateChangedListener listener);

    interface OnPlayerStateChangedListener {
        /**
         * 播放器状态变更回调
         *
         * @param playerState 参见
         *                    {@link PlayerState#STATE_IDLE}
         *                    {@link PlayerState#STATE_BUFFERING}
         *                    {@link PlayerState#STATE_READY}
         *                    {@link PlayerState#STATE_ENDED}
         */
        void onPlayerStateChange(int playerState);
    }
```

- 设置首帧被渲染事件监听器

```
    /**
     * 设置首帧被渲染事件监听器
     * @param listener
     */
    void setOnRenderedFirstFrameListener(OnRenderedFirstFrameListener listener);

    interface OnRenderedFirstFrameListener {
        /**
         * 首帧被渲染回调
         */
        void onRenderedFirstFrame();
    }
```

- 设置视频内容宽高变更回调

```
        /**
         * 设置视频内容宽高变更回调
         * @param listener
         */
        void setOnVideoSizeChangedListener(OnVideoSizeChangedListener listener);

    interface OnVideoSizeChangedListener {
```

```
    /**
     * 内容宽高改变时的回调
     *
     * @param width  视频内容宽，单位像素
     * @param height 视频内容高，单位像素
     */
    void onVideoSizeChanged(int width, int height);
}
```

- 设置播放到内容结束事件监听器

```
/**
  * 设置播放到内容结束事件监听器
  * 收到该事件后，需要调用stop，播放器状态才会切到STATE_END
  * @param listener
  */
 void setOnCompletionListener(OnCompletionListener listener);

interface OnCompletionListener {
    /**
     * 播放到文件末尾回调
     */
    void onCompletion();
}
```

- 获取当前播放进度

```
/**
  * 获取当前播放进度，相对于开始位置的偏移量
  * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
  * @return 单位MS
  */
 long getCurrentPosition();
```

- 获取视频总时长

```
/**
  * 获取视频总时长
  * 播放器状态为{@link #STATE_READY}时调用有效
  * @return 单位MS
  */
 long getDuration();
```

- 获取音量

```
/**
  * 获取音量
  * @return 范围0-1，0为静音
  */
 float getVolume();
```

- 获取播放器状态

```
/**
  * 获取播放状态
  * @return 状态枚举：
  * {@link PlayerState#STATE_IDLE} 播放器初始状态
  * {@link PlayerState#STATE_BUFFERING} 缓冲中状态
  * {@link PlayerState#STATE_READY} 缓冲结束开始播放状态
  * {@link PlayerState#STATE_ENDED} 播放完成状态
  */
```

```
int getPlayState();
```

- 获取播放器当前流的连接类型

```
/**
 * 获取播放器当前流的连接类型
 * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
 * @return {@link StreamConnectType}
 */
StreamConnectType getStreamConnectType();
```

- 获取播放器当前的帧率/码率等信息

```
/**
 * 获取播放器当前的帧率/码率等信息
 * 播放器状态为{@link PlayerState#STATE_BUFFERING}及{@link PlayerState#
STATE_READY}时调用有效
 * @return 包含帧率/码率等信息的json string
 */
PlayInfo getCurrentPlayInfo();
```

- ExoHlsPlayer
  - 构造方法

```
ExoHlsPlayer(Context context);
```

  - 设置m3u8播放地址

```
/**
 * 设置播放源
 * @param url m3u8地址
 */
void setDataSource(String url);
```

  - 设置播放地址为IPC云存录像（按文件名）

```
/**
 * 设置播放地址为IPC云存录像（按文件名）
 * @param iotId    设备iotId
 * @param fileName 录像文件名
 */
void setDataSourceByIPCRecordFileName(String iotId, String fileName);
```

  - 校验和准备数据

```
/**
 * 校验和准备数据
 */
void prepare();
```

  - 开始或恢复播放视频

```
/**
 * 开始播放或恢复播放视频
 */
```

```
  void start();
```

- 暂停播放

```
/**
  * 暂停播放, 调用start()恢复播放
  */
  void pause();
```

- seek到指定位置

```
/**
  * seek到指定位置
  * @param position 毫秒
  */
  void seekTo(long positionInMs);
```

- 停止播放

```
**
  * 停止播放
  */
  void stop();
```

- 重置播放器

```
/**
  * 重置播放器
  */
  void reset();
```

- 释放播放器资源

```
/**
  * 释放播放器资源
  */
  void release();
```

- 设置是否循环播放

```
/**
  * 设置是否循环播放
  * @param circlePlay true为循环播放
  */
  void setCirclePlay(boolean circlePlay);
```

- 设置回放速率

```
/**
  * 设置回放的播放速率
  * @param speed 速率因子，(0,2]之间，1为正常速率
  */
  void setPlaybackSpeed(float speed);
```

- 设置播放器音量

```
/**
  * 设置播放器音量
  * @param audioVolume 范围 0-1，0为静音
```

```
   */
  void setVolume(float audioVolume);
```

- 设置音频流通道类型

```
/**
   * 设置音频流通道类型，参见{@link android.media.AudioManager}
   * 如果音频正在播放，则会因为重新创建AudioTrack导致有短暂停顿
   * @param audioStreamType
   */
  void setAudioStreamType(int audioStreamType);
```

- 设置画面缩放模式

```
/**
   * 设置视频画面缩放模式，默认为{@link android.media.MediaCodec#VIDEO_SCAL
ING_MODE_SCALE_TO_FIT_WITH_CROPPING}
   * @param videoScalingMode 参见：
   * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FIT}
   * {@link android.media.MediaCodec#VIDEO_SCALING_MODE_SCALE_TO_FI
T_WITH_CROPPING}
   */
  void setVideoScalingMode(int videoScalingMode);
```

- 设置surfaceview

```
/**
   * 设置SurfaceView
   * @param surfaceview
   */
  void setSurfaceView(SurfaceView surfaceview);
```

- 清除surfaceview

```
/**
   * 清除surfaceview
   */
  void clearSurfaceView();
```

- 设置textureview

```
/**
    * 设置TextureView，同一个TextureView不可被多个播放器共享，提供ZoomableTe
xtureView带手势放缩处理可做参照
    * @param textureview
    */
   void setTextureView(TextureView textureView);
```

- 清除textureview

```
/**
    * 清除textureview
    */
   void clearTextureView();
```

- 设置数据源准备就绪事件监听器

```
/**
   * 设置数据源准备就绪事件监听器
   * @param listener
```

```
   */
  void setOnPreparedListener(OnPreparedListener listener);

 interface OnPreparedListener {
    /**
     * 数据源准备就绪回调
     */
    void onPrepared();
 }
```

- 设置播放器错误事件监听器

```
 /**
   * 设置播放器错误事件监听器，错误类型参见：
   * {@link PlayerException.SOURCE_ERROR}
   * {@link PlayerException.RENDER_ERROR}
   * {@link PlayerException.UNEXPECTED_ERROR}
   * @param listener
   */
  void setOnErrorListener(OnErrorListener listener);

 interface OnErrorListener {
    /**
     * 播放器错误回调，参见
     * {@link PlayerException}
     * @param exception
     */
    void onError(PlayerException exception);
 }
```

- 设置播放状态变更事件监听器

```
 /**
   * 设置播放状态变更事件监听器
   * @param listener
   */
  void setOnPlayerStateChangedListener(OnPlayerStateChangedListener listener);

 interface OnPlayerStateChangedListener {
    /**
     * 播放器状态变更回调
     *
     * @param playerState 参见
     *                    {@link PlayerState#STATE_IDLE}
     *                    {@link PlayerState#STATE_BUFFERING}
     *                    {@link PlayerState#STATE_READY}
     *                    {@link PlayerState#STATE_ENDED}
     */
    void onPlayerStateChange(int playerState);
 }
```

- 设置首帧被渲染事件监听器

```
 /**
   * 设置首帧被渲染事件监听器
   * @param listener
   */
  void setOnRenderedFirstFrameListener(OnRenderedFirstFrameListener listener);

  interface OnRenderedFirstFrameListener {
    /**
     * 首帧被渲染回调
```

```
     */
    void onRenderedFirstFrame();
}
```

- 设置视频内容宽高变更回调

```
    /**
     * 设置视频内容宽高变更回调
     * @param listener
     */
    void setOnVideoSizeChangedListener(OnVideoSizeChangedListener listener);

 interface OnVideoSizeChangedListener {

    /**
     * 内容宽高改变时的回调
     *
     * @param width  视频内容宽，单位像素
     * @param height 视频内容高，单位像素
     */
    void onVideoSizeChanged(int width, int height);
}
```

- 获取当前播放进度

```
/**
  * 获取当前播放进度，相对于开始位置的偏移量
  * 播放器状态为{@link PlayerState#STATE_READY}时调用有效
  * @return 单位MS
  */
 long getCurrentPosition();
```

- 获取视频总时长

```
/**
  * 获取视频总时长
  * 播放器状态为{@link #STATE_READY}时调用有效
  * @return 单位MS
  */
 long getDuration();
```

- 获取音量

```
/**
  * 获取音量
  * @return 范围：0-1
  */
 float getVolume();
```

- 获取播放器状态

```
/**
  * 获取播放状态
  * @return 状态枚举：
  * {@link PlayerState#STATE_IDLE} 播放器初始状态
  * {@link PlayerState#STATE_BUFFERING} 缓冲中状态
  * {@link PlayerState#STATE_READY} 缓冲结束开始播放状态
  * {@link PlayerState#STATE_ENDED} 播放完成状态
  */
```

```
int getPlayState();
```

- ZoomableTextureView
  - 设置画面放大倍数上限

```
/**
 * 设置画面放大倍数上限
 * 默认为4倍
 *
 * @param scale 倍数
 */
void setMaxScale(float scale);
```

  - 将画面缩放比重置为1

```
/**
 * 缩放比重置为1
 *
 * @param smooth 是否带有平滑变化效果
 */
void zoomOut(boolean smooth);
```

  - 获取当前缩放比

```
/**
 * 获取当前缩放比
 *
 * @return  未放缩时为1.0f
 */
float getScale();
```

  - 设置监听器

```
/* 设置监听器
 * @param listener
 */
void setOnZoomableTextureListener(OnZoomableTextureListener listener);

public interface OnZoomableTextureListener {

    /**
     * 当画面缩放比例发生变化时回调
     *
     * @param zoomableTextureView
     * @param scale             画面缩放比例
     */
    void onScaleChanged(ZoomableTextureView zoomableTextureView, float scale
);

    /**
     * view双击事件回调
     *
     * @param zoomableTextureView
     * @param e                 MotionEvent
     * @return 事件是否被处理，如果返回false，则会启用内部缩放逻辑
     */
    boolean onDoubleTap(ZoomableTextureView zoomableTextureView,
MotionEvent e);

    /**
```

```
        * view单击事件回调
        *
        * @param zoomableTextureView
        * @param e                MotionEvent
        * @return 事件是否被处理
        */
       boolean onSingleTapConfirmed(ZoomableTextureView zoomableTextureView,
MotionEvent e);

       /**
        * view长按事件回调
        *
        * @param zoomableTextureView
        * @param e                MotionEvent
        */
       void onLongPress(ZoomableTextureView zoomableTextureView, MotionEvent e
);
   }
```

- 设置边缘监听器

```
    void setOnViewEdgeListener(OnViewEdgeListener listener)

    interface OnViewEdgeListener {

      /**
       * 当第一次拖动到view的边缘
       */
      void onViewEdgeFirstTouched();

      /**
       * 当拖动到view下边缘，持续拖动会持续回调
       *
       * @param zoomableTextureView
       * @param delta    持续处于边缘时触摸移动相对上次的差值
       */
      void onBottomEdge(ZoomableTextureView zoomableTextureView, float delta);

      /**
       * 当拖动到view上边缘，持续拖动会持续回调
       *
       * @param zoomableTextureView
       * @param delta            持续处于边缘时触摸移动相对上次的差值
       */
      void onTopEdge(ZoomableTextureView zoomableTextureView, float delta);

      /**
       * 当拖动到view右边缘，持续拖动会持续回调
       *
       * @param zoomableTextureView
       * @param delta            持续处于边缘时触摸移动相对上次的差值
       */
      void onRightEdge(ZoomableTextureView zoomableTextureView, float delta);

      /**
       * 当拖动到view左边缘，持续拖动会持续回调
       *
       * @param zoomableTextureView
       * @param delta            持续处于边缘时触摸移动相对上次的差值
       */
      void onLeftEdge(ZoomableTextureView zoomableTextureView, float delta);
```

```
}
```

错误列表

错误主码	描述	子码	描述
SOURCE_ERROR	数据源相关错误	SUB_CODE_SOURCE_STREAM_CONNECT_ERROR(1005)	与数据源建立连接失败
		SUB_CODE_SOURCE_INVALID_DECRYPTE_KEY(1006)	无效的解密密钥
		SUB_CODE_SOURCE_INVALID_RTMP_URL(1007)	无效的播放地址
		SUB_CODE_SOURCE_PARAMETER_ERROR(1008)	错误的数据源参数
		SUB_CODE_SOURCE_QUERY_URL_FAILED(1009)	请求播放地址失败
RENDER_ERROR	渲染相关错误	SUB_CODE_RENDER_DECODE_ERROR(1000)	解码错误
UNEXPECTED_ERROR	不符合预期错误	SUB_CODE_UNEXPECTED_PULL_STREAM_ERROR(1100)	拉流失败，8S未拉取到流或连接被异常断开

语音对讲

提供App和IPC设备之间端到端的单/双向实时对讲能力。

- 单讲：App端采集并发送音频数据到设备端进行播放，App端采集音频期间手机保持声音静默。
- 双讲：App端和设备端都需要同时做采音和放音，设备端必须支持AEC，否则不建议使用该方案。

支持以下格式的音频。

类型	采样率	编码	解码
G711A	8Khz/16Khz	#	#
G711U	8Khz/16Khz	#	#

使用指南

语音对讲集成分为以下几个步骤。

1. 创建语音对讲实例，并设置对讲模式和音频参数。

```
// 创建语音对讲实例
```

```
liveIntercomV2 = new LiveIntercomV2(context, iotId, LiveIntercomV2.LiveIntercomMode.DoubleTalk, AudioParams.AUDIOPARAM_MONO_8K_G711A);
```

2. 注册监听器，并处理语音对讲回调。

   请在对应的事件回调中处理，包括对讲开始、录音开始和结束、录音数据回调（用于UI展示，如音量计算）。

   在语音通道建立和对讲过程中，可能发生的错误详见错误列表。

```
// 设置语音对讲错误回调
liveIntercomV2.setLiveIntercomV2Listener(new LiveIntercomV2Listener() {
        @Override
        public void onTalkReady() {
            showToast("可以开始说话了");
        }
        @Override
        public void onError(LiveIntercomException error) {
            showToast("code:" + error.getCode() + " msg:" + error.getMessage());
        }

        @Override
        public void onRecordStart() {
            // 直播播放器设置为静音
            player.setVolume(0f);
            showToast("录音机已启动");
        }

        @Override
        public void onRecordEnd() {
            // 直播播放器恢复音量
            player.setVolume(1f);
            showToast("录音机结束");
        }

        @Override
        public void onRecordBufferReceived(byte[] buffer, int offset, int size) {
            // nothing
            Log.d(TAG, "onBufferReceived:" + size);
        }
    });
```

3. 设置增益水平。

   可以设置App端声音采集的增益值，提供“无、低、中、高、强”五种水平供选择（默认值为高），请根据设备效果来调整。

```
liveIntercomV2.setGainLevel(LiveIntercomV2.GAIN_LEVEL_NONE);
```

4. 开始与停止对讲。

   对讲开始后，会请求音频焦点，并设置到通话模式，同时将对端播放声音强制路由到扬声器；对讲结束后释放音频焦点，并设置到常规模式，同时取消强制音频播放路由到扬声器规则。对讲过程中默认开启手机回声消除。

```
// 开始对讲
liveIntercomV2.start();
```

```
// 主动停止
liveIntercomV2.stop();
```

接口说明

- LiveIntercomV2
  - 构造方法

```
    /**
     * 创建对讲实例
     *
     * @param context         application context
     * @param iotId           iotId
     * @param liveIntercomMode 对讲模式，单讲SingleTalk、双讲DoubleTalk
     * @param audioParams      音频参数（采样率、通道数、采样位宽、编码格式）
     */
    LiveIntercomV2(Context context, final String iotId, final LiveIntercomMode
liveIntercomMode, final AudioParams audioParams);
```

  - 开始语音对讲

```
/**
  * 开始语音对讲
  */
 void start();
```

  - 关闭语音对讲

```
/**
  * 关闭语音对讲，停止录音，关闭对讲通道，停止播放
  */
 void stop();
```

  - 释放对讲实例

```
    /**
     * 释放对讲实例
     */
    void release();
```

  - 设置静音

```
    /**
     * 设置静音
     *
     * @param mute true为静音，false为取消静音
     */
    void setMute(boolean mute)
```

  - 当前对讲是否静音

```
    /**
     * 当前对讲是否静音
     *
     * @return
     */
```

```
boolean isMute()
```

- 设置增益水平

```
/**
 * 设置增益水平
 *
 * @param gainLevel
 */
void setGainLevel(int gainLevel)
```

- 设置对讲监听器

```
/**
 * 设置对讲监听器
 *
 * @param listener
 */
void setLiveIntercomV2Listener(LiveIntercomV2Listener listener);

interface LiveIntercomV2Listener {

    /**
     * 语音对讲对端就绪
     */
    void onTalkReady();

    /**
     * 语音对讲发生错误
     * @param error
     */
    void onError(LiveIntercomException error);

    /**
     * 开始录音
     */
    void onRecordStart();

    /**
     * 结束录音
     */
    void onRecordEnd();

    /**
     * 接收录音数据
     * @param buffer
     * @param offset
     * @param size
     */
    void onRecordBufferReceived(byte[] buffer, int offset, int size);
}
```

错误列表

错误枚举	描述
LiveIntercomException.INVALID_AUDIO_PARAMS	无效的语音对讲音频参数（对端上报的音频参数SDK不支持）

错误枚举	描述
LiveIntercomException.START_LIVE_INTERCOM_REQUEST_FAILED	启动语音对讲失败
LiveIntercomException.CONNECTION_STREAM_FAILED	语音流建立失败（5S未建立成功会超时）
LiveIntercomException.SEND_STREAM_DATA_FAILED	发送语音流数据失败
LiveIntercomException.RECEIVE_STREAM_DATA_FAILED	接收语音流数据失败
LiveIntercomException.INIT_RECORD_FAILED	录音机初始化失败
LiveIntercomException.START_RECORD_FAILED	录音机启动错误
LiveIntercomException.READ_RECORD_BUFFER_FAILED	录音数据读取错误
LiveIntercomException.INIT_AUDIO_PLAYER_FAILED	音频播放器创建失败

3.9.14 常见问题

介绍Android SDK在开发过程中遇到的常见问题和解决方法。

Q：登录App或注册时，提示网络不顺畅

A：检查以下内容是否正确。

- 确认手机时间是否设置正确，如果有偏差，则会被安全监测拦截而导致无法登录。
- 确认安全图片是否在src/drawable目录下，如果在初始化代码中设置了图片后缀为 114d ，请务必保证安全图片的名称为yw_1222_114d（API level 7以及以下版本适用，如下图所示），API level 8版本参见#unique_103来重新集成安全图片。

```
//设置图片后缀
ConfigManager.getInstance().setSecGuardImagePostfix("114d");
```

- 确认初始化中是否有抛出异常。
  - 其中ErrorCode=110的异常码可以忽略
  - 以下几种Warn可以忽略（用来检查社交账号SDK）

```
W/oa_Oauth: [] : Umeng is not available, Umeng Oauth Service Provider is disabled
W/oa_Oauth: [] : Taobao MemberSDK is not available, Taobao 3rd Oauth Service Provider is disabled
W/oa_Oauth: [] : Taobao login4android SDK is not available, Taobao 2nd Oauth Service Provider is disabled
W/oa_Oauth: [] : Alipay sdk is not available, Alipay Oauth Service Provider is disabled
```

- 如果是用Demo中提供的OALoginAdapter，下面异常日志可以忽略。

```
java.lang.NullPointerException: Attempt to invoke virtual method 'long java.lang.
Long.longValue()' on a null object reference
at com.aliyun.iot.aep.sdk.login.oa.OALoginAdapter.a(OALoginAdapter.java:343)
```

无法打开登录页面

A：检查以下内容是否正确。

- 确认日志中是否有以下异常输出。

```
init failed code = 10010 message = 发生错误，消息为null，请使用LogCat查看细节，
或者搜索Failed resolution of: Lcom/ut/mini/UTHitBuilders@UTCustomHitBuilder
```

这种情况说明主工程缺少UT SDK的依赖，请在主工程里增加如下两个依赖，或者在平台上重新下载SDK依赖文件。

```
compile 'com.aliyun.ams:alicloud-android-utdid:1.1.5.4'
compile 'com.aliyun.ams:alicloud-android-ut:5.1.0'
```

- 如果启动直接Crash，查看日志有提示XML相关的报错，请检查Manifest.xml中的**packageName**和build.gradle中的**applicationID**必须保持一致。

Q：API通道SDK初始化时，出现ErrorCode=103错误

A：查看logcat中是否有类似如下的报错日志。

```
12-16 16:26:11.430 10486-10486/? W/System.err: ErrorCode = 103
12-16 16:26:11.430 10486-10486/? W/System.err: com.alibaba.wireless.security.open
.SecException: java.lang.UnsatisfiedLinkError: dlopen failed: "/data/data/com.aliyun.
iot.demo.wuxi.demoapp/app_SGLib/app_1513412765/libsgmainso-5.3.38.so" is 32-bit
instead of 64-bit
12-16 16:26:11.430 10486-10486/? W/System.err:     at com.alibaba.wireless.security.
mainplugin.SecurityGuardMainPlugin.onPluginLoaded(Unknown Source)
12-16 16:26:11.430 10486-10486/? W/System.err:     at com.alibaba.wireless.security.
framework.e.a(Unknown Source)
12-16 16:26:11.430 10486-10486/? W/System.err:     at com.alibaba.wireless.security.
framework.e.c(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.alibaba.wireless.security.
framework.e.c(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.alibaba.wireless.security.open.
initialize.b.a(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.alibaba.wireless.security.open.
initialize.a.loadLibrarySync(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.alibaba.wireless.security.open.
initialize.a.initialize(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.alibaba.wireless.security.jaq.
SecurityInit.Initialize(Unknown Source)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.aliyun.iot.demo.wuxi.demoapp
.DemoApplication.onCreate(DemoApplication.java:34)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.app.Instrumentation.
callApplicationOnCreate(Instrumentation.java:1025)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.app.ActivityThread.
handleBindApplication(ActivityThread.java:5403)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.app.ActivityThread.-wrap2(
ActivityThread.java)
```

```
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.app.ActivityThread$H.
handleMessage(ActivityThread.java:1545)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.os.Handler.dispatchMe
ssage(Handler.java:102)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.os.Looper.loop(Looper.java
:154)
12-16 16:26:11.431 10486-10486/? W/System.err:     at android.app.ActivityThread.main(
ActivityThread.java:6119)
12-16 16:26:11.431 10486-10486/? W/System.err:     at java.lang.reflect.Method.invoke(
Native Method)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.android.internal.os.ZygoteInit$
MethodAndArgsCaller.run(ZygoteInit.java:886)
12-16 16:26:11.431 10486-10486/? W/System.err:     at com.android.internal.os.ZygoteInit.
main(ZygoteInit.java:776)
```

如果出现以上日志，一般是so文件加载失败导致的，可以在build.gradle文件中添加如下配置。

```
android {
    compileSdkVersion 28
    defaultConfig {

        ...

        ndk {
            abiFilters "arm64-v8a","armeabi-v7a"  //API level 7及以上版本的SDK
           // abiFilters "armeabi", "x86"   //API level 6以及低版本的SDK，过滤除 armeabi 和 x86
 以外的 so 文件
        }
    }
}
```

Q：API通道SDK初始化时，出现app key or app secret must be initialed错误

A：查看logcat中是否有类似如下的报错日志如下。

```
10-22 03:03:37.555 18552-18552/? E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.rnd.kx, PID: 18552
    java.lang.RuntimeException: Unable to create application com.rnd.kx.MyApplication:
com.alibaba.cloudapi.sdk.exception.SdkException: app key or app secret must be initialed
        at android.app.ActivityThread.handleBindApplication(ActivityThread.java:4710)
        at android.app.ActivityThread.-wrap1(ActivityThread.java)
        at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1405)
        at android.os.Handler.dispatchMessage(Handler.java:102)
        at android.os.Looper.loop(Looper.java:148)
        at android.app.ActivityThread.main(ActivityThread.java:5417)
        at java.lang.reflect.Method.invoke(Native Method)
        at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
    Caused by: com.alibaba.cloudapi.sdk.exception.SdkException: app key or app secret
must be initialed
```

如果出现以上日志，检查以下内容是否正确。

- 检查是否出现203错误码，并根据导致错误的可能原因进行详细排查。
- 请检查libsgmain.so是否被打到apk包内，要确保lib/armeabi和lib/x86文件夹内包含该文件。

Q：API 通道 SDK 初始化的时候，出现ErrorCode=202错误

A：查看logcat中是否有类似如下的报错日志如下。

```
12-16 13:19:06.586 9344-9344/com.aliyun.iot.demo W/System.err: ErrorCode = 202
12-16 13:19:06.587 9344-9344/com.aliyun.iot.demo W/System.err: com.alibaba.wireless.
security.open.SecException:
12-16 13:19:06.594 9344-9344/com.aliyun.iot.demo W/System.err:     at com.taobao.
wireless.security.adapter.JNICLibrary.doCommandNative(Native Method)
12-16 13:19:06.595 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.
wireless.security.mainplugin.a.doCommand(Unknown Source:0)
12-16 13:19:06.595 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.
wireless.security.a.j.a.getAppKeyByIndex(Unknown Source:21)
12-16 13:19:06.596 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.sdk.
android.push.securitybox.alipush.a.getAppKey(Unknown Source:7)
12-16 13:19:06.597 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.sdk.
android.push.vip.AppRegister.h(Unknown Source:17)
12-16 13:19:06.597 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.sdk.
android.push.vip.AppRegister.a(Unknown Source:34)
12-16 13:19:06.597 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.sdk.
android.push.impl.j.a(Unknown Source:30)
12-16 13:19:06.597 9344-9344/com.aliyun.iot.demo W/System.err:     at com.alibaba.sdk.
android.push.impl.j.register(Unknown Source:1)
12-16 13:19:06.599 9344-9344/com.aliyun.iot.demo W/System.err:     at com.aliyun.iot.aep
.demo.DemoApplication.initPush(DemoApplication.java:115)
12-16 13:19:06.599 9344-9344/com.aliyun.iot.demo W/System.err:     at com.aliyun.iot.aep
.demo.DemoApplication.onCreate(DemoApplication.java:27)
12-16 13:19:06.599 9344-9344/com.aliyun.iot.demo W/System.err:     at android.app.
Instrumentation.callApplicationOnCreate(Instrumentation.java:1118)
12-16 13:19:06.599 9344-9344/com.aliyun.iot.demo W/System.err:     at android.app.
ActivityThread.handleBindApplication(ActivityThread.java:5791)
12-16 13:19:06.599 9344-9344/com.aliyun.iot.demo W/System.err:     at android.app.
ActivityThread.-wrap1(Unknown Source:0)
12-16 13:19:06.601 9344-9344/com.aliyun.iot.demo W/System.err:     at android.app.
ActivityThread$H.handleMessage(ActivityThread.java:1661)
12-16 13:19:06.601 9344-9344/com.aliyun.iot.demo W/System.err:     at android.os.
Handler.dispatchMessage(Handler.java:105)
12-16 13:19:06.601 9344-9344/com.aliyun.iot.demo W/System.err:     at android.os.Looper
.loop(Looper.java:164)
12-16 13:19:06.601 9344-9344/com.aliyun.iot.demo W/System.err:     at android.app.
ActivityThread.main(ActivityThread.java:6541)
12-16 13:19:06.601 9344-9344/com.aliyun.iot.demo W/System.err:     at java.lang.reflect.
Method.invoke(Native Method)
12-16 13:19:06.604 9344-9344/com.aliyun.iot.demo W/System.err:     at com.android.
internal.os.Zygote$MethodAndArgsCaller.run(Zygote.java:240)
12-16 13:19:06.604 9344-9344/com.aliyun.iot.demo W/System.err:     at com.android.
internal.os.ZygoteInit.main(ZygoteInit.java:767)
```

如果出现以上日志，通常是安全图片与当前apk的签名不匹配导致的，可以修改当前apk签名或者重新上传apk生成新的安全图片。

Q：API通道SDK初始化时，出现ErrorCode=203错误

A：查看logcat中是否有类似如下的报错日志如下。

```
12-16 13:29:59.005 10232-10267/? W/System.err: ErrorCode = 203
12-16 13:29:59.005 10232-10267/? W/System.err: com.alibaba.wireless.security.open.
SecException:
```

```
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.taobao.wireless.security.
adapter.JNICLibrary.doCommandNative(Native Method)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.wireless.security.
mainplugin.a.doCommand(Unknown Source:0)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.wireless.security.a.j.a.
getAppKeyByIndex(Unknown Source:21)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.
securitybox.alipush.a.getAppKey(Unknown Source:7)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.
securitybox.alipush.a.a(Unknown Source:8)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.
securitybox.alipush.a.getMpsDeviceId(Unknown Source:2)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.vip.
AppRegister$a.b(Unknown Source:68)
12-16 13:29:59.005 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.vip.
AppRegister$a.a(Unknown Source:0)
12-16 13:29:59.006 10232-10267/? W/System.err:     at com.alibaba.sdk.android.push.vip.b
.handleMessage(Unknown Source:54)
12-16 13:29:59.006 10232-10267/? W/System.err:     at android.os.Handler.dispatchMe
ssage(Handler.java:105)
12-16 13:29:59.006 10232-10267/? W/System.err:     at android.os.Looper.loop(Looper.java
:164)
12-16 13:29:59.006 10232-10267/? W/System.err:     at android.os.HandlerThread.run(
HandlerThread.java:65)
```

如果出现以上日志，检查以下内容是否正确。

- 没有找到图片文件，请确保安全图片在res\drawable目录下，并检查安全图片的名称的末四位和当前authCode是否一致（authCode默认为 114d）。
- 安卓环境下可能是因为资源优化被优化成了0，请检查APK中的图片。
  - 如果开启混淆，需要检查发布包335大小不为0，shrinkResources true会导致yw_1222_0335和yw_1222图片大小为0。

    ```
    release {
            minifyEnabled true // 是否混淆
            shrinkResources true //会导致335或者122图片大小为0
            proguardFiles getDefaultProguardFile("proguard-android.txt"), "proguard-rules
    .pro"
          }
    ```

    可通过放弃资源压缩或新建res/raw/keep.xml文件，并在文件中添加以下内容。

    ```
    <?xml version="1.0" encoding="UTF-8"?>
    resources xmlns:tools="http://schemas.android.com/tools"
        tools:keep="@drawable/yw_1222_0335, @drawable/yw_1222"/>
    ```
  - 如果是在android studio下调试发现找不到图片，但是确认过图片是正常的，请关闭Android studio的instant run功能，在instant run下apk并非一个完整bundle，其资源文件被拆分到特定bundle中，图片路径发生改变会导致保镖找不到图片。

Q：无法初始化登录SDK，提示发生错误且消息为null

```
init failed code = 10010 message =
```

A：可以使用LogCat查看更多日志细节。可能原因为缺少SDK依赖，请在主工程的build.gradle中添加以下2个依赖，或者在平台上重新生成一份dependency.gradle文件。

```
compile 'com.aliyun.ams:alicloud-android-utdid:1.1.5.4'
compile 'com.aliyun.ams:alicloud-android-ut:5.1.0'
```

Q：访问API时，API网关出现以下报错

```
code:403, message:request forbidden, localizedMsg:请求被禁止
```

A：可能原因为该项目APPKEY没有访问API的权限，请通过控制台右上角的**工单**联系我们，并备注APPKEY以及请求API的名称。

Q：出现“unsupported auth type iotAuth”报错

```
unsupported auth type iotAuth, maybe you forgot to register IoTAuthProvider
```

A：参见SDK初始化文档初始化SDK，初始化IoTCredentialProviderImpl模块代码也放在应用内。如需要主动调用接口还需添加setAuthType("iotAuth")。

Q：Demo App切换国际站或更换安全图片后，提示网络不顺畅或无法连接

A：解决办法如下。

- 设置BuildConfig.BUILD_COUNTRY=OVERSEA
- build.gradle里面的**CHINA**更改为**OVERSEA**
- Demo App里面的**SINGAPORE**都更改为**OVERSEA**
- drawable里面的安全图片需要更改为国际站的安全图片
- 将EnvConfigure.putEnvArg(RNContainerComponentDelegate.KEY_RN_CONTAINER_PLUGIN_ENV, "test");更改为EnvConfigure.putEnvArg(RNContainerComponentDelegate.KEY_RN_CONTAINER_PLUGIN_ENV, "release");
- RNContainerComponentDelegate.java里面的BoneConfig.set("region", "china");更改为BoneConfig.set("region", "singapore");
- 如采用release.gradle里面打包apk方式，更改src/oversea/res和/configure/oversea/里的安全图片

Q：提示libreactnativejni.so无法找到

A: 查看logcat有如下提示。

```
java.lang.UnsatisfiedLinkError: couldn't find DSO to load: libreactnativejni.so
    at com.facebook.soloader.SoLoader.doLoadLibraryBySoName(SoLoader.java:738)
    at com.facebook.soloader.SoLoader.loadLibraryBySoName(SoLoader.java:591)
    at com.facebook.soloader.SoLoader.loadLibrary(SoLoader.java:529)
    at com.facebook.soloader.SoLoader.loadLibrary(SoLoader.java:484)
    at com.facebook.react.bridge.ReactBridge.staticInit(ReactBridge.java:31)
    at com.facebook.react.bridge.NativeMap.<clinit>(NativeMap.java:19)
```

```
    at com.facebook.react.jscexecutor.JSCExecutorFactory.create(JSCExecutorFactory.java:
25)
    at com.facebook.react.ReactInstanceManager$5.run(ReactInstanceManager.java:944)
    at java.lang.Thread.run(Thread.java:764)
```

出现上面的错误的可能原因是CPU架构不支持（目前我们不再支持armeabi和x86架构），您可以通过添加以下代码解决。

```
android {
    compileSdkVersion 28
    defaultConfig {

        ...

        ndk {
            abiFilters "armeabi-v7a", "arm64-v8a" //过滤armeabi和x86的so文件
        }
    }
}
```

3.10 iOS SDK手册

3.10.1 SDK升级

本文介绍API Level 7及以下版本升级至API Level 8的操作指导。

概述

生活物联网平台发布的App端SDK最新版本为API Level 8，该版本优化了整体SDK框架，对下载的SDK采对统一接口进行初始化（更多介绍请参见#unique_101）。

- 如果您是第一次集成SDK的新用户，请直接使用最新版本的SDK，不涉及SDK升级。

  您可以按照#unique_68来下载最新版本的SDK，并按照SDK初始化来初始化SDK。
- 如果您是已经集成过API Level 7或之前的SDK的用户，推荐您升级至最新版本的SDK，并使用新的初始化方案。新的方案对后期增加新功能、开拓海外市场等，具有更大的优势。当然您也可以保留之前的初始化方法，只更新SDK版本。本文档提供了两种升级方案，请自行选择。

升级SDK并使用新的初始化方案

1. 在**自有品牌App-SDK下载**页面，选择最新的API Level 8，并下载新的SDK套餐项。详细请参见#unique_68。

   下载后的压缩包解压后，包含安全图片和Podfile文件。
2. 删除当前App工程中初始化相关的代码，如API通道SDK初始化代码、账号及用户SDK初始化代码、身份认证SDK初始化代码等。
3. 使用新的统一初始化接口，并完成SDK的初始化。详细请参见SDK初始化。

4. （可选）添加原来的定制化逻辑，如用户账号的定制化UI等。

升级SDK但不用新的初始化方案

1. 在**自有品牌App-SDK下载**页面，选择最新的API Level 8，并下载新的SDK套餐项。详细请参见#unique_68。

   下载后的压缩包解压后，包含安全图片和Podfile文件。

2. 将代码工程中原有Podfile里的SDK版本号，替换为新下载Podfile里的SDK版本号。

3.10.2 SDK初始化

当您开发自有App，下载并集成SDK后，需要对所有SDK进行初始化。

概述

从API Level 8开始，SDK的初始化不再需要逐一初始化SDK，而可以使用统一的初始化接口，一次性完成所有的所需SDK的初始化。如果您使用的是API Level 7或以下版本，建议您升级至最新版本，详细请参见SDK升级。

根据开发者下载SDK时勾选的**SDK配置项**，统一接口对如下SDK逐一初始化。

- API通道SDK （必选初始化）
- 账号及用户SDK （必选初始化）
- 身份认证SDK（必选初始化）
- 长连接通道SDK
- 设备模型SDK
- 移动应用推送SDK
- BoneMobile容器SDK

SDK初始化

1. 集成安全图片。详细操作请参见#unique_103。

   说明:
   安全图片文件名根据站点、版本的不同而存在差异。请勿修改安全图片名称，下载后直接拷贝到App工程目录下。

2. 下载并集成SDK。详细操作请参见#unique_68。
3. 初始化SDK。

   a) 配置SDK。

```
#import <IMSIotSmart/IMSIotSmart.h>
```

```
IMSIotSmartConfig *config = [IMSIotSmartConfig new];
config.regionType = REGION_ALL; //取值范围参见枚举类型 `IMSRegionType`
//这是默认配置，可以根据实际情况调整

[IMSIotSmart sharedInstance].config = config;
```

说明：

如果使用默认配置，可忽略此步骤。

b) 启动初始化。

```
#import <IMSIotSmart/IMSIotSmart.h>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(
NSDictionary *)launchOptions {

    IMSIotSmartConfig *config = [IMSIotSmartConfig new];
    config.regionType = REGION_ALL;  // 设置地域类型，如果您的设备仅在国内销售，请
设置为“REGION_CHINA_ONLY”；
                // 如果您的设备还需要往海外销售，则需设置为“REGION_ALL”，且还需设置
相应的国家，参见步骤3-3。
    //如使用默认配置，此处可以省略
    [IMSIotSmart sharedInstance].config = config;

    [[IMSIotSmart sharedInstance] application:application didFinishLaunchingWi
thOptions:launchOptions];

    return YES;
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(
NSDictionary *)userInfo {
   //如果APP没有集成移动应用推送能力，此处无需要调用
    [[IMSIotSmart sharedInstance] application:application didReceiveRemoteNoti
fication:userInfo];
}

- (void)application:(UIApplication *)application didRegisterForRemoteNotificati
onsWithDeviceToken:(NSData *)deviceToken {
    //如果APP没有集成移动应用推送能力，此处无需要调用
    [[IMSIotSmart sharedInstance] application:application didRegisterForRemote
NotificationsWithDeviceToken:deviceToken];
}

- (void)application:(UIApplication *)application didFailToRegisterForRemoteNoti
ficationsWithError:(NSError *)error {
    //如果APP没有集成移动应用推送能力，此处无需要调用
    [[IMSIotSmart sharedInstance] application:application didFailToRegisterFor
RemoteNotificationsWithError:error];
}
```

c) 设置国家。

阿里云IoT的云端服务为多区域部署，如果您将**config.regionType**设置成**REGION_ALL**后，还需要设置相应的国家，请参见#unique_11完成。

说明：

如果您未设置国家，SDK初始化流程会挂起，直至国家设置完毕。在国家设置完成前请勿使用SDK的任一API，否则会造成报错。

4. 设置App可配网的产品列表。

```
#import <IMSIotSmart/IMSIotSmart+scope.h>
[[IMSIotSmart sharedInstance] configProductScope:PRODUCT_SCOPE_ALL];
   /// 配置App上能看到的产品的范围，PRODUCT_SCOPE_ALL：表示当前项目中已发布和未发布的所有产品；PRODUCT_SCOPE_PUBLISHED：表示只包含已发布产
   /// 正式发布的App请选择PRODUCT_SCOPE_PUBLISHED
```

SDK API Reference

在使用生活物联网平台提供的SDK时，相关的SDK API注释请参见SDK API Reference。

3.10.3 通用SDK

通用SDK用来帮助所有SDK做初始化和统一设置，比如设置语言、设置国家等。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

设置国家相关

1. 显示国家、地区列表页面

   - 调用默认页面

```
#import <IMSIotSmart/IMSIotSmart.h>
// 使用SDK提供的国家选择UI
   [[IMSIotSmart sharedInstance] showCountryListInNavigationController:self.navigationController callback:^(IMSIotCountry * _Nullable country) {
       // 使用SDK提供的国家选择UI，选中某个国家时的回调
       NSLog(@"selcted country:%@",country);
   }];
```

   - 自定义UI页面

```
#import <IMSIotSmart/IMSIotSmart.h>
[[IMSIotSmart sharedInstance] getCountryList:^(NSError * _Nullable error, NSArray<IMSIotCountry *> * _Nullable countryList) {
       // 返回的countryList为原始数据
       // 您可以基于这些原始数据设计UI呈现国家列表
   }];
```

2. 设置国家

```
#import <IMSIotSmart/IMSIotSmart.h>
[[IMSIotSmart sharedInstance] setCountry:_selectCountry callback:^(BOOL needRestartApp) {
      if (needRestartApp) {
         ////needRestartApp为YES时，必须退出应用，否则无法正常使用。
      }
```

```
}];
```

说明：

App在中国内地与除中国内地以外地区之间切换时，需要重启才能正确初始化。

设置多语言

生活物联网平台目前支持中文（zh-CN）、 英文（en-US）、法文（fr-FR）、德文（de-DE）、日文（ja-JP）、韩文（ko-KR）、西班牙文（es-ES）、俄文（ru-RU）、意大利文（it-IT）、印地文（hi-IN）、葡萄牙文（pt-PT）等十一种语言。

- 设置语言

  统一切换API网关、用户账号、推送、插件等SDK的语言环境。

```
#import <IMSIotSmart/IMSIotSmart.h>
NSString *language = @"zh-CN";
[[IMSIotSmart sharedInstance] setLanguage:language];
```

- 获取当前语言

  获取API网关、用户账号、推送、插件等SDK的语言环境。

```
#import <IMSIotSmart/IMSIotSmart.h>
// 查看SDK当前设置的语言，如果您没有设置过语言，此处会返回当前系统语言
NSString *language = [[IMSIotSmart sharedInstance] getLanguage];
```

调试未发布产品

设置App配网列表的产品范围，取值如下。

- PRODUCT_SCOPE_ALL：表示当前项目中已发布和未发布的所有产品。
- PRODUCT_SCOPE_PUBLISHED：表示只包含已发布产品。正式发布的App请选择**PRODUCT_SCOPE_PUBLISHED**。

```
#import <IMSIotSmart/IMSIotSmart+scope.h>
[[IMSIotSmart sharedInstance] configProductScope:PRODUCT_SCOPE_ALL];
```

设置日志开关

设置日志开关，请参见 日志SDK。

3.10.4 账号及用户SDK

提供自建账号体系能力，包括注册、登录、登出、获取账号、会话管理、人机校验、登录UI定制等功能。同时基于Oauth2.0协议，提供快速对接自有账号的能力，满足开发者在App开发中对接自有账号体系的需求。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

> **说明:**
> SDK集成使用Pod集成，添加完依赖后需要将ALBBOpenAccountUI.framework中的xib目录放置到主工程目录下。

内置账号

- 基础功能
  - 注册

```
// 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>
#import <ALBBOpenAccountCloud/ALBBOpenAccountUser.h>

// 获取账号UI服务
id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUISer
vice);

// 显示手机注册窗口，presentingViewController将通过present方式展示登陆界面
viewcontroller
[uiService presentRegisterViewController:presentingViewController success:^(
ALBBOpenAccountSession *currentSession) {
  // 注册成功，currentSession为当前会话信息
} failure:^(NSError *error) {
    // 只有用户取消注册才会执行
}];

// 显示邮箱注册窗口，presentingViewController将通过present方式展示登陆界面
viewcontroller
[uiService presentEmailRegisterViewController:presentingViewController success:^(
ALBBOpenAccountSession *currentSession) {
  // 注册成功，currentSession为当前会话信息
} failure:^(NSError *error) {
    // 只有用户取消注册才会执行
}];
```

  - 登录

    目前平台已经支持两种登录方式：账号登录、邮箱登录。登录页面调起后，两种登录方式都可以使用。

```
// 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>
#import <ALBBOpenAccountCloud/ALBBOpenAccountUser.h>
```

```
// 获取账号UI服务
id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUISer
vice);
// 显示登录窗口，presentingViewController将通过present方式展示登陆界面
viewcontroller
[uiService presentLoginViewController:presentingViewController 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
}];
```

- 退出登录

```
// 引入头文件
#import <IMSAccount/IMSAccountService.h>

// 退出登录
 [[IMSAccountService sharedService] logout];
```

- 忘记密码

  调起登录页面之后，在登录页面内会出现注册和忘记密码的功能，无需额外开发。

- 注销账号

  注销账号时，会同时解除用户绑定的设备关系，请参见注销账号。

- 修改个人信息
  - 修改昵称

```
/ 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>
#import <ALBBOpenAccountCloud/ALBBOpenAccountService.h>

[ALBBService(ALBBOpenAccountService) updateAccountProfile:@{@"displayName
" : @"name1"} Callback:^(NSError *error) {
   if (error == nil) {
      NSLog(@"modify nickname successfully ");
   } else {
      NSLog(@"fail to modify nickname");
   }
}];
```

  - 修改头像

    需要先将头像图片存储到云端，获取该图片的URL，再用如下方式更新。

```
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>
#import <ALBBOpenAccountCloud/ALBBOpenAccountService.h>
```

```
[ALBBService(ALBBOpenAccountService) updateAccountProfile:@{@"avatarUrl" : url
} Callback:^(NSError *error) {
    if (error == nil) {
        NSLog(@"modify photo image successfully ");
    } else {
        NSLog(@"fail to set photo image");
    }
}];
```

- 刷新会话

```
// 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>

// 获取当前会话
ALBBOpenAccountSession *session = [ALBBOpenAccountSession sharedInstance];
// 刷新当前会话
[session refreshSessionIDWithCallback:^(NSString *sid, NSError *err) {
  // 如果失败则返回error, 否则返回新的会话Id:sid
}];
```

- 获取会话ID

```
// 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>

ALBBOpenAccountSession *session = [ALBBOpenAccountSession sharedInstance];
NSString *sessionID = session.sessionID;
```

- 获取用户信息

```
// 引入头文件
#import <ALBBOpenAccountCloud/ALBBOpenAccountSDK.h>

// 获取当前会话
ALBBOpenAccountSession *session = [ALBBOpenAccountSession sharedInstance];
if ([session isLogin]) {
    // 获取用户信息
    ALBBOpenAccountUser *user = session.getUser;
}
```

- 登录失效处理

  参见身份认证SDK中API请求的认证错误处理章节。

- OA语言切换
  - 使用默认语言

    默认语言不需要进行任何语言设置，默认中文，只需要在“xib”格式的文件中自行调整UI即可。
  - 自定义方式（简单集成）
    - pod 'IMSOpenAccountCustom', '1.1.1'
    - 将IMSOpenAccountCustom的IMSOpenAccountCustomResource.bundle下的xib放到主工程，如果已经有了，可自行决定是否替换
    - 设置语言，只需要指定前缀即可。可设置OA模块语言有：

      zh、en、es、de、fr、ja、ko、ru、hi、it。

```
// 使用默认的翻译资源文件
  [[IMSiLopOALanguageManage shareInstance]
  setOpenAccountModuleLanguageWithLanguagePrefix:@"ko"];

// 想使用自己的资源文件（xxxx）：将 IMSOpenAccountCustom的IMSOpenAccountCustomResource.bundle的.lproj文件中的内容拷贝，自行整理一份相同key的八国语言翻译
  [[IMSiLopOALanguageManage shareInstance]
  setOpenAccountModuleLanguageWithLanguagePrefix:@"ko" bundleName:@"xxxx"];
```

  - 高度自定义方式（自定义较高）

```
//一、设置云端语言：zh-CN、en-US、fr-FR、de-DE、ja-JP、ko-KR、es-ES、ru-RU、hi-IN、it-IT
[[ALBBOpenAccountSDK sharedInstance] setRpcLocale:locale];

//二、修改xib上显示语言，默认显示中文
//1、将ALBBOpenAccountUI下拷贝ALBBOpenAccount.bundle中的.lproj文件中的内容拷贝，自行整理一份相同key的八国语言翻译
//2、设置显示的国际化文件
```

```
[[ALBBOpenAccountSDK sharedInstance] setLocale:@"zh.lproj"];
//3、指定显示的国际化文件的具体路径
NSString *bundlePath = [NSString stringWithFormat:@"%@/%@.bundle/zh.lproj",[
NSBundle mainBundle].bundlePath, bundleName];
[[ALBBOpenAccountSDK sharedInstance] setLocaleBundle:[NSBundle bundleWith
Path:bundlePath]];
//4、通过设置xib各个控制器的代理，自行在代理方法中对控件进行取多语言文案，实现赋
值操作
id<ALBBOpenAccountUIService> uiService = ALBBService(ALBBOpenAccountUISer
vice);
[uiService setLoginViewDelegate:self];

- (void)loginViewDidLoad:(ALBBOpenAccountLoginViewController *)viewController
 {
    viewController.usernameField.placeholder = @"xxx";// 自行取资源赋值
    viewController.usernameLabel.text = @"xxx";// 自行取资源赋值
}
```

- 自定义UI

  请参见 UI界面自定义完成。

三方自有账号

> **说明:**
> 在对接三方自有账号时，必须先完成SDK的初始化。

1. 引入依赖的头文件

```
#import <ALBBOpenAccountSSO/ALBBOpenAccountSSOSDK.h>
#import <IMSAccount/IMSAccountService.h>
```

实现自有账号自定义登录，获取 Oauth2.0 的 AuthCode，并调用 ALBBOpenAccountSSOSe

rvice 进行 Oauth 授权登录。

如何实现**asyncLoginGetAuthCode**，请参见用户账号开发指南。

```
@implementation IMSAccountThirdViewController

- (void)asyncLoginGetAuthCode:(void (^)(NSError * _Nullable error, NSString *
_Nullable authCode))completionHandler {
    //自有账号登录并通过 OAuth2.0 服务获取 AuthCode
}

- (IBAction)onClickLogin:(id)sender {
    [self asyncLoginGetAuthCode:^(NSError * _Nullable error, NSString * _Nullable
authCode) {
        if (error) {
            // 错误处理
            return;
        } else {
            id<ALBBOpenAccountSSOService> ssoService = ALBBService(ALBBOpenAc
countSSOService);
            [ssoService oauthWithThirdParty:authCode delegate:self];
        }
    }];
}
```

```
@end
```

2. 实现Oauth授权登录回调

```
@interface IMSAccountThirdViewController () <SSODelegate>
@end

@implementation IMSAccountThirdViewController

- (void)openAccountOAuthError:(NSError *)error Session:(ALBBOpenAccountSession *)
session {
    if (!error) {
        //登录成功，发送登录成功通知，身份认证 SDK 会监听该通知进行用户身份凭证创建和管
理
        NSString *loginNotificationName = [[IMSAccountService sharedService].sessionPro
vider accountDidLoginSuccessNotificationName];
        [[NSNotificationCenter defaultCenter] postNotificationName:loginNotificationNam
e object:nil];
    } else {
        //处理登录失败
    }
}
@end
```

3.10.5 身份认证SDK

提供基于iotToken的用户身份认证方案，通过和账号及用户SDK、API通道SDK的集成，完成用户身份凭证的生成和管理，以及发起API请求的用户身份的鉴权。

依赖SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力
AP通道	提供API通道能力，和基础环境配置信息

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用方式

- 发送带身份认证的API请求

```
// 引入头文件
#import <IMSAuthentication/IMSIoTAuthentication.h>
#import <IMSApiClient/IMSApiClient.h>

// 构建请求
NSString *path = @"/uc/listByAccount";
NSString *apiVer = @"1.0.0";
NSDictionary *params = @{};
IMSIoTRequestBuilder *builder = [[IMSIoTRequestBuilder alloc] initWithPath:path
apiVersion:apiVer params:params];

// 指定身份认证类型
[builder setAuthenticationType:IMSAuthenticationTypeIoT];
```

```
//通过 IMSRequestClient 发送请求
[IMSRequestClient asyncSendRequest:builder.build responseHandler:^(NSError *
_Nullable error, IMSResponse * _Nullable response) {
    if (error) {
        //处理Error，非服务端返回的错误都通过该Error回调
    } else {
        if (response.code == 200) {
            //成功，处理response.data
        }
        else {
            //处理服务端错误
        }
    }
}];
```

- API请求的认证错误处理

  目前服务端逻辑不支持同一个账号多端登录。如果一个账号在多个设备登录，那么只有最后一个登录的账号才可以正常访问IoT服务，其他账号在发送API通道请求时则会返回认证错误（错误码为401）。另外，如果账号未登录或登录信息过期（或长时间未登录）时，也会在发送API通道请求时返回认证错误。

  通常情况下只需要处理API请求的错误即可。

```
[IMSRequestClient asyncSendRequest:builder.build responseHandler:^(NSError *
_Nullable error, IMSResponse * _Nullable response) {
 if (error) {
     //...
 } else {
     if (response.code == 200) {
         //...
     }
     else if (response.code == 401) {
         //处理认证错误
     }
     else {
         //...
     }
 }
}];
```

  如果您希望统一处理这种类型的认证错误，可根据以下流程来实现。

  1. 继承IMSIoTAuthentication来实现自定义的身份认证功能。

```
@interface XXCustomAuthentication : IMSIoTAuthentication
@end
@implementation XXCustomAuthentication

- (void)handleRequestBeforeSend:(IMSRequest * _Nonnull)request
                        payload:(IMSRequestPayload * _Nonnull)payload
                     completion:(void (^ _Nonnull)(NSError * _Nullable error,
                                                   IMSResponse *_Nullable mockResponse,
                                                   IMSRequestPayload * _Nullable newPayload))
completionHandler {
```

```
    [super handleRequestBeforeSend:request payload:payload completion:^(
NSError * _Nullable error, IMSResponse * _Nullable mockResponse, IMSRequest
Payload * _Nullable newPayload) {
        completionHandler(error, mockResponse, newPayload);

        if (mockResponse && mockResponse.code == 401) {
            //自定义处理 401, 比如 toast
            NSLog(@"before: 401");
        }
    }];
}

- (void)handleResponse:(IMSResponse * _Nonnull)response
          completion:(void (^ _Nonnull)(NSError * _Nullable error, IMSResponse *
_Nullable response))completionHandler {
    [super handleResponse:response
              completion:^(NSError * _Nullable error, IMSResponse * _Nullable response)
 {
                  completionHandler(error, response);

                  if (response && response.code == 401) {
                      //自定义处理 401, 比如 toast
                      NSLog(@"after: 401");
                  }
              }];
}
@end
```

2. 注册自定义的身份认证。

```
XXCustomAuthentication *iotAuthDelegate = [[XXCustomAuthentication alloc]
initWithCredentialManager:IMSCredentialManager.sharedManager];
[IMSRequestClient registerDelegate:iotAuthDelegate forAuthenticationType:
IMSAuthenticationTypeIoT];
```

- 获取用户身份凭证

  在账号登录成功后，可通过下面的方法同步获取用户身份凭证信息。

```
#import <IMSAuthentication/IMSCredentialManager.h>

IMSCredential *credential = [IMSCredentialManager sharedManager].credential;
NSString *identityId = credential.identityId;
NSString *iotToken = credential.iotToken;
```

- 刷新用户身份凭证

  当同步方法获取到的用户身份凭证为空时，可以通过下面的方法强制异步刷新一个新的用户身份凭证信息。

```
#import <IMSAuthentication/IMSCredentialManager.h>

[[IMSCredentialManager sharedManager] asyncRefreshCredential:^(NSError *
_Nullable error, IMSCredential * _Nullable credential) {
    if (error) {
        //刷新出错，参考错误码 IMSCredentialManagerErrorCode 处理
    } else {
        NSString *identityId = credential.identityId;
        NSString *iotToken = credential.iotToken;
    }
```

```
}];
```

3.10.6 API通道SDK

API通道SDK，提供IoT业务协议封装的https请求能力，并通过整合安全组件来提升通道的安全性。

依赖 SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用说明

API 调用示例如下。

```
#import <IMSApiClient/IMSApiClient.h>

// 构建请求
NSDictionary *params = @{@"input":@"测试"};
IMSIoTRequestBuilder *builder = [[IMSIoTRequestBuilder alloc] initWithPath:@"/kit/debug
/ping"
                                                    apiVersion:@"1.0.0"
                                                       params:params];
// 可选参数
// [builder setHost:@"xxx"];//指定API host
// [builder setScheme:@"https"];

//通过 IMSRequestClient 发送请求
[IMSRequestClient asyncSendRequest:builder.build responseHandler:^(NSError *
_Nullable error, IMSResponse * _Nullable response) {
    if (error) {
        //处理Error，非服务端返回的错误都通过该Error回调
    }
    else {
        if (response.code == 200) {
            //成功，处理response.data
        }
        else {
            //处理服务端错误，可通过response.localizedMsg展示错误Toast
        }
    }
}];
```

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

超时时长设置

指定API请求的超时时长。

```
// 设置全局超时时长，对所有API请求都生效，不设置默认10s
[IMSConfiguration sharedInstance].timeoutInterval = 10;
```

```
// 针对局部请求超时时长设置
NSDictionary *params = @{@"input":@"测试"};
IMSIoTRequestBuilder *builder = [[IMSIoTRequestBuilder alloc] initWithPath:@"/kit/debug
/ping"
                                                    apiVersion:@"1.0.0"
                                                       params:params];
// 可选参数，不设置默认使用全局配置超时时长
builder.timeoutInterval = 10;

//通过 IMSRequestClient 发送请求
[IMSRequestClient asyncSendRequest:builder.build responseHandler:^(NSError *
_Nullable error, IMSResponse * _Nullable response) {
    if (error) {
        //处理Error，非服务端返回的错误都通过该Error回调
    }
    else {
        if (response.code == 200) {
            //成功，处理response.data
        }
        else {
            //处理服务端错误，可通过response.localizedMsg展示错误Toast
        }
    }
}];
```

打开调试日志

如需打开调试日志，请参见 日志SDK。

3.10.7 长连接通道SDK

长连接通道SDK，提供IoT业务协议封装的云端数据下行能力，为App提供订阅、发布消息的能力和支持请求响应模型。

依赖SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力
API 通道	提供API通道能力，和基础环境配置信息

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用说明

SDK封装了上行RPC请求、订阅、取消订阅等接口，详细接口参见 长连接服务。

SDK中描述的Topic都是简短的Topic。例如完整的上行请求Topic“/sys/{productKey}/{deviceName}/app/up/test/publish”。

上行请求SDK 内部会判断补齐/sys/{productKey}/{deviceName}/app/up/，在调用SDK入参的时候只需要输入/test/publish即可。

对应的下行Topic，例如完整的设备状态变化下行Topic是/sys/{productKey}/{deviceName}/app/down/things/status。SDK回调里面只会露出/things/status，自动截断掉/sys/{productKey}/{deviceName}/app/down前缀。

业务请求响应模型

这个接口实际上是封装了一个Remote Procedure Call的过程。我以用户账号绑定通道的示例来说明内部逻辑：用户账号绑定通道的Topic为：/sys/{productKey}/{deviceName}/app/up/account/bind。

在向这个Topic发布数据前，先订阅这个Topic对应的Reply Topic，其格式如下所示 /sys/{productKey}/{deviceName}/app/down/account/bind_reply。订阅成功后才开始发布数据，IoT用户中心在收到SDK 发布到/sys/{productKey}/{deviceName}/app/up/account/bind这个Topic的数据后，完成账号绑定的业务逻辑后，会往/sys/{productKey}/{deviceName}/app/down/account/bind_reply这个Topic发布响应数据。SDK在收到这个reply Topic的数据后，将响应结果通过respHandler回调给用户，从而完成整个业务逻辑。

```
#import <AlinkAppExpress/LKAppExpress.h>
//由于长连接通道 SDK，会在内部逻辑中补齐 '/sys/{productKey}/{deviceName}/app/up' 部分，
//所以使用者在这个 API 时，只要传这一段的后边部分即 '/account/bind' 即可了。
[[LKAppExpress sharedInstance] invokeWithTopic : @"/account/bind"  opts:nil       params:@{@"iotToken":iotToken}
                            respHandler:^(LKAppExpResponse * _Nonnull response) {
                               LKAELogDebug(@"bindAccount result : %@", response);
                            }];
```

订阅Topic

长连接通道SDK，采用MQTT协议，基于订阅/发布模型设计。订阅某个Topic后，当其他服务或者终端往这个Topic发布消息时，便能收到消息。下边以订阅用户所绑定设备属性变化的Topic为例。

Topic全路径为：/sys/{productKey}/{deviceName}/app/down/thing/properties。由于长连接通道 SDK，会在内部逻辑中补齐/sys/{productKey}/{deviceName}/app/down部分，所以使用者在调用订阅Topic API 时，只要整个Topic 的后边部分即/thing/properties即可。其他Topic依次类推。

```
#import <AlinkAppExpress/LKAppExpress.h>
//以订阅用户所绑定的设备属性变化事件为例，详细的使用说明请参考 api reference
[[LKAppExpress sharedInstance]subscribe:@"/thing/properties" complete:^(NSError * _Nullable error) {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (error == nil) {
            [_tipsLabel setText:@"订阅成功"];
        } else {
            [_tipsLabel setText:@"订阅失败"];
        }
    });
```

```
}];
```

取消订阅Topic

取消订阅是订阅的逆过程，二者遵循同样的Topic规则。仍然以取消订阅用户所绑定设备属性变化的Topic为例。

```
#import <AlinkAppExpress/LKAppExpress.h>
// 详细的使用说明请参考 api reference
[[LKAppExpress sharedInstance]unsubscrbie:@"/thing/properties" complete:^(NSError *
_Nullable error) {
    dispatch_async(dispatch_get_main_queue(), ^{
        if (error == nil) {
            [_tipsLabel setText:@"取消订阅成功"];
        } else {
            [_tipsLabel setText:@"取消订阅失败"];
        }
    });
}];
```

Publish数据

长连接通道SDK基于订阅/发布模型设计。既可以订阅Topic，也可以往某个Topic发布数据。下边以往Topic：/sys/{productKey}/{deviceName}/app/up/test/publish发送数据为例。

由于长连接通道SDK，会在内部逻辑中补齐/sys/{productKey}/{deviceName}/app/up部分，所以使用者在调用Publish数据API时，只要传这一段的后边部分即/test/publish即可。

```
#import <AlinkAppExpress/LKAppExpress.h>

NSString * text = @"{\"input\":\"Hello World\"}";
NSData *data = [text dataUsingEncoding:NSUTF8StringEncoding];
if (data == nil) {
    return;
}
NSError *error;
NSDictionary *params = [NSJSONSerialization JSONObjectWithData:data options:0 error:&
error];
if (error) {
    return;
}
[[LKAppExpress sharedInstance]publish:@"/test/publish" params:params
                             complete:^(NSError * _Nonnull error) {
                                 dispatch_async(dispatch_get_main_queue(), ^{
                                     if (error == nil) {
                                         [_tipsLabel setText:@"publish成功"];
                                     } else {
                                         [_tipsLabel setText:@"publish失败"];
                                     }
                                 });
                             }];
```

注册下行Listener

注册下行Listener，才能接收到订阅过的Topic数据，参见长链接服务消息下行API。

```
#import <AlinkAppExpress/LKAppExpress.h>
```

```
@interface TestDownstreamListener：NSObject <LKAppExpDownListener>
@end
@implementation TestDownstreamListener

- (void)onDownstream:(NSString * _Nonnull)topic data:(id  _Nullable)data {

    NSLog(@"onDownstream topic : %@", topic);
    NSLog(@"onDownstream data : %@", data);
    NSDictionary * replyDict = nil;
    if ([data isKindOfClass:[NSString class]]) {
        NSData * replyData = [data dataUsingEncoding:NSUTF8StringEncoding];
        replyDict = [NSJSONSerialization JSONObjectWithData:replyData options:NSJSONRead
ingMutableLeaves error:nil];

    } else if ([data isKindOfClass:[NSDictionary class]]) {
        replyDict = data;
    }
    if (replyDict == nil) {
        return;
    }
}

- (BOOL)shouldHandle:(NSString * _Nonnull)topic {
    if ([topic isEqualToString:@"/thing/properties"]) {
        return YES;//返回YES，说明对此topic感兴趣,SDK会调用[listener onDownstream:data:]
    }
    return NO;
}
@end

self.testListner = [TestDownstreamListener new];//sdk不会strong持有此listener，开发者自
己保证listener不被释放.
[[LKAppExpress sharedInstance]addDownStreamListener:YES listener:self.testListner]
```

解绑长连接通道与账号

在初始化时，已实现长连接通道与账号的绑定，此时如果您需要解绑长连接通道与账号，参照以下代码执行。

如需解绑长连接通道跟用户账号的绑定，请在账号登出前操作，否则会导致解绑失败。

```
#import <AlinkAppExpress/LKAppExpress.h>
#import <IMSAuthentication/IMSCredentialManager.h>

#pragma mark - 取消长连接通道关联

NSString *topic = @"/account/unbind";
    [[LKAppExpress sharedInstance] invokeWithTopic:topic opts:nil params:@{} respHandle
r:^(LKAppExpResponse * _Nonnull response) {
        if (![response successed]) {
            IMSLifeLogVerbose(@"解绑长连接推送失败");
        }
    }];
```

3.10.8 BoneMobile容器SDK

BoneMobile容器SDK为可选模块，提供加载插件的功能。如果您需要开发或者使用插件，则需要在App中集成BoneMobile容器SDK。

依赖 SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力
API通道	提供IoT业务协议封装的API网关HTTPS请求能力，和集成无线保镖SDK进行请求报文安全加签的能力
MJRefresh	github上开源的下拉刷新控件，版本：3.1.15
ZipArchive	github上开源的zip解压库，版本：1.4.0

说明：

Google Play已于2019年8月1日停掉尚未支持64位体系的App，如果您App要在海外Google Play应用商店上架，且用到了BoneKit SDK，那需要尽快升级，以免无法上架。后续如果推出新插件或老插件升级，仅针对已升级到v0.59 BoneKit SDK的自有App。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

切换语言（国际化）

为了满足国际化需求，自有App可以支持多种语言。如果您需要切换语言，请参见#unique_11。

使用方式

- 通过路由打开UIViewController

```
#import <IMSRouter/IMSRouter.h>

NSURL *url = [NSURL URLWithString:@"link://router/{routerName}"];
NSDictionary *para = @{@"key": @"value"};
[[IMSRouterService sharedService] openURL:url
                                  options:para
                        completionHandler:^(BOOL success) {
                            if (success) {
                                NSLog(@"插件打开成功");
                            } else {
                                NSLog(@"插件打开失败");
                            }
```

```
                    }];
```

- 打开插件面板并接收返回值

  参见调用配网插件，并接受配网结果的示例。

```
#import <IMSRouter/IMSRouter.h>

NSURL *url = [NSURL URLWithString:@"link://router/connectConfig"];
NSDictionary *para = @{AKRouterCompletionHandlerKey: ^(NSError *error, NSDictiona
ry *result) {
        // result handler
    }};
[[IMSRouterService sharedService] openURL:url
                                  options:para
                        completionHandler:^(BOOL success) {
                            if (success) {
                                NSLog(@"插件打开成功");
                            } else {
                                NSLog(@"插件打开失败");
                            }
                        }];
```

- 直接打开UIViewController

```
#import <IMSBoneKit/BoneRCTViewController.h>

NSURL *url = [NSURL URLWithString:@"{插件URL地址}"];
NSDictionary *para = @{@"key": @"value"};
BoneRCTViewController *controller = [BoneRCTViewController new];
[controller openUrl:newURL props:params];
[self.navigationController pushViewController:controller animated:YES];
```

- 本地Debug方式打开UIViewController

  调试时需使用Debug的ReactNative库，这样才会出现调试菜单，请修改Podfile文件如下。

```
#pod 'AKReactNative', '0.41.2'
pod 'AKReactNative', '0.41.2-debug' #这是Debug版本，请勿使用该版本发上appStore
```

  示例代码如下。

```
__weak typeof(self) wSelf = self;
NSURLSession *session = [NSURLSession sharedSession];
NSString *host = @"/*启用Bone服务的电脑IP*/";
NSURL *url = [NSURL URLWithString:[NSString stringWithFormat:@"http://%@:8081/
boneDebugUrl?platform=ios&ip=%@", host, host]];
NSURLRequest *request = [NSURLRequest requestWithURL:url];

NSURLSessionDataTask *task = [session dataTaskWithRequest:request
                                        completionHandler:^(NSData *data, NSURLResponse *response,
NSError *error)
{
    dispatch_async(dispatch_get_main_queue(), ^{
        if (!error) {
            NSDictionary *responseJSON = [NSJSONSerialization JSONObjectWithData:data
options:NSJSONReadingAllowFragments error:nil];
            int responseCode = [[responseJSON objectForKey:@"code"] intValue];
            if (responseCode == 200) {
                NSDictionary *responseData= [responseJSON objectForKey:@"data"];
```

```
            NSString *urlString = responseData[@"url"];
            BoneRCTViewController *controller = [BoneRCTViewController new];
            [controller openUrl:[NSURL URLWithString:urlString] props:[responseData
mutableCopy]];
            controller.hidesBottomBarWhenPushed = YES;
            [self.navigationController pushViewController:controller animated:YES];
        }
    }
  });
}];
[task resume];
```

集成账号能力

BoneKit支持账号插件，可以支持自定义账号接入，可以自行实现IMSAccountProtocol，IMSAccountUIProtocol协议，然后设置IMSAccountService的sessionProvider，accountProvider 来完成。如账号使用OpenAccount对接的话，可以参考账号Demo下IMSOpenAccount的实现。

代码示例如下。

```
// 引入头文件
#import <IMSAccount/IMSAccountService.h>

IMSOpenAccount *openAccount = [IMSOpenAccount sharedInstance];
[IMSAccountService sharedService].sessionProvider = openAccount;
[IMSAccountService sharedService].accountProvider = openAccount;
```

Native和JS共享配置

某些业务场景下，Native端和JS端可能需要共享一些配置。为了满足这个需求，我们开辟了一个Native和JS都可以访问的配置区。

- JS端访问配置区，请参见环境配置信息。
- iOS端访问配置区，参考如下代码。

```
// 设置
[[IMSBoneConfiguration sharedInstance] set:@"region" value:@"china"];

// 获取
[[IMSBoneConfiguration sharedInstance] get:@"region"];
```

3.10.9 日志SDK

日志SDK，是一个基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力。

初始化

工程引入头文件如下。

```
#import <IMSLog/IMSLog.h>
```

```
//统一设置所有模块的日志 tag 输出级别
[IMSLog setAllTagsLevel:IMSLogLevelAll];
```

```
//可选：设置是否开启日志的控制台输出，建议在release版本中不要开启。
[IMSLog showInConsole:YES];
```

Level级别如下。

```
IMSLogLevelError
IMSLogLevelWarning
IMSLogLevelInfo
IMSLogLevelDebug
IMSLogLevelVerbose
```

使用说明

API 调用示例如下。

```
//IMSLog拥有分模块日志隔离的能力。
//使用时必须先注册tag，推荐使用模块名称作为tag。
[IMSLog registerTag:@"IMSApiClient"];

//可选：可以指定详细的tag对应的日志输出级别。
[IMSLog setLevel:IMSLogLevelDebug forTag:@"IMSApiClient"]

...

IMSLogError(tag,frmt,...)
IMSLogWarn(tag,frmt,...)
IMSLogInfo(tag,frmt,...)
IMSLogDebug(tag,frmt,...)
IMSLogVerbose(tag,frmt,...)

eg:
IMSLogError(@"IMSApiClient", @"错误：%@", @"token失效");
```

3.10.10 配网SDK

提供了把WiFi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、手机热点配网、蓝牙辅助配网、智能路由器配网以及设备间相互配网等。

依赖SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力
API通道	提供API通道能力，和基础环境配置信息
Breeze SDK	提供蓝牙辅助配网支持

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用说明

- 设备发现

  提供获取局域网内已经配网的设备，包括WiFi设备、以太网设备，以及局域网内已上电的待配设备（前提是当前局域网内已经有一个已配网的智能设备，且该待配设备支持本地发现的配网能力）。

  - 启动发现设备

    发现本地的已配网设备，或者已配网设备、路由器等发现的待配设备。发现的待配设备信息可以作为后续设备配网的入参信息。

    ```
    // 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:
          // 0代表wifi设备；1代表ethernet设备（网线连接）：2代表路由器；@"ble_subtyp
    e_2" 代表蓝牙辅助配网设备*/
       }];
    ```

  - 获取所有已发现设备

    ```
    // 引入头文件
    #import <IMLDeviceCenter/IMLDeviceCenter.h>

    NSArray *allLanDevicesArray = [kLKLocalDeviceMgr getLanDevices];
    ```

  - 终止发现设备

    停止发现本地已配和线上待配设备，调用该接口会清除已发现设备列表，确保与**startDiscovery**成对调用。

    ```
    // 引入头文件
    #import <IMLDeviceCenter/IMLDeviceCenter.h>

    // 停止发现设备
    [kLKLocalDeviceMgr stopDiscovery];
    ```

- 通用配网流程

  - 设置待添加设备信息

    待添加设备信息来源可以为上面本地发现的待配设备，也可以通过扫码等其他途径获取待配设备信息。

    ```
    // 引入头文件
    #import <IMLDeviceCenter/IMLDeviceCenter.h>

    // 选取本地发现的待配设备
    IMLCandDeviceModel *model = self.localDeviceList[index];
    // 自行创建待配设备信息
    ```

```
IMLCandDeviceModel *model = [[IMLCandDeviceModel alloc] init];
model.productKey = @"xxxx";
model.linkType = ForceAliLinkTypeXXX;
...

[kLkAddDevBiz setDevice:model];
```

其中，（IMLCandDeviceModel *）model为本地发现待配设备或者云端拉取产品列表组装的model。 **IMLCandDeviceModel**属性说明如下。

属性	类型	是否必选	描述	备注
productKey	NSString	是	设备productKey	N/A
deviceName	NSString	否	设备名称	N/A
productId	NSString	否	待配设备产品ID	在蓝牙辅助配网中必须填入
linkType	assign	否	指定配网方式	■ ForceAliLinkTypeBroadcast 一键广播配网；ForceAliLinkTypeHotspot 手机热点配网方案 ■ ForceAliLinkTypeSoftap 设备热点配网方案 ■ ForceAliLinkTypeQR 摄像头扫码配网方案 ■ ForceAliLinkTypeBLE 蓝牙辅助配网方案

- 设置配网模式

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

[kLkAddDevBiz setAliProvisionMode:ForceAliLinkTypeHotspot];
```

- 开始设备配网

设置好待添加设备信息，进入配网。调用startAddDevice接口进入配网流程，并实现监听的协议方法。

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

[kLkAddDevBiz startAddDevice:self];

其中 self 为配网过程中 notifier 监听回调对象（代理）

- (void)notifyPrecheck:(BOOL)success withError:(NSError *)err
{
    NSLog(@"notifyPrecheck callback err : %@", err);
    dispatch_async(dispatch_get_main_queue(), ^{
```

```
        [self.addVC notifyProgress:LKAddStatePrechecking result:nil withError: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){
        // TODO:一键广播配网相关引导
    } else if(guideCode == LKPGuideCodeWithUserGuide){
        // TODO:手机热点配网相关引导
    } else if(guideCode == LKPGuideCodeWithUserGuideForSoftAp) {
        // TODO:设备热点配网相关引导
    } else if(guideCode == LKPGuideCodeWithUserGuideForQR) {
        // TODO:摄像头扫码配网相关引导
    }
}

-(void)notifyProvisioning
{
    NSLog(@"notifyProvisioning callback(正在进行配网...) ");
}

/**
手机热点配网状态回掉调，提示用户关闭热点并切回以前的wifi，optional。
 */

- (void)notifyProvisioningNotice{
    NSLog(@"notifyProvisioningNotice");
}

/*
 设备热点状态回调，optional。
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);
}

/**
摄像头二维码配网模式相关回调，optional
 @param qrcode 需要UI展现的二维码内容
 */

- (void)notifyProvisioningNoticeForQR:(NSString *) qrcode;


/**
 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(@"配网成功：%@",candDeviceModel);
```

```
}
```

- 输入配网WiFi名称以及密码信息

在收到- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode回调引导完成相关操作（一键广播指引用户输入ssid和密码，热点配网指引用户开启热点，输入ssid和密码等）后，调用- (void)toggleProvision:(NSString) ssid pwd:(NSString) pwd timeout:(int) timeout方法，传入WiFi的ssid和password信息。

说明:

一键广播配网和热点配网才会有notifyProvisionPrepare回调。

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

- (void)notifyProvisionPrepare:(LKPUserGuideCode)guideCode
{
    NSLog(@"notifyProvisionPrepare callback guide code : %ld", guideCode);
    [self inputSsidAndPassword];
}

- (void)inputSsidAndPassword
{
    NSSstring *ssid = @"example ssid";
    NSString *password = @"1qaz@WSX";
    NSInterger timeout = 60;(单位秒,s);
    [kLkAddDevBiz toggleProvision:ssid pwd:password timeout:timeout];
}
```

- 配网过程关键节点监听处理

针对热点配网必须监听该回调

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

- (void)notifyProvisionStatus:(LKProvisonStatus)provisionStatus boolSuccess:(BOOL
)boolSuccess;
{
    NSLog(@"notifyProvisionStatus callback provisionStatus:%d boolSuccess:%d",
provisionStatus, boolSuccess);
    if(provisionStatus == LKProvisonStatusSwitchAP){
        // 设备回复 switch ap 请求，提示用户切换回之前的 wifi
        NSLog(@"请立即切换回开启热点之前的 wifi 网络");
    }
}
```

- 配网结果监听

```
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvis
ionError:(NSError *)provisionError
{
    if(candDeviceModel != nil){
        NSLog(@"配网成功： %@",candDeviceModel);
    } else{
        NSLog(@"配网失败，错误信息:%@", provisionError);
    }
```

```
}
```

- 停止配网

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

[kLkAddDevBiz stopAddDevice];
```

- 设备绑定

当完成设备配网之后，还需要将设备与用户或者与家庭进行关联绑定，绑定过程中需要调用SDK的获取token方法，取得设备token，然后进行设备绑定。获取token的代码如下。

```
// 引入头文件
#import <IMLDeviceCenter/IMLDeviceCenter.h>

// self.productKey 和 self.deviceName 是配网成功后返回的设备模型中的 productKey 和 deviceName
[[IMLLocalDeviceMgr sharedMgr] getDeviceToken:self.productKey deviceName:self.deviceName 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)");
    }
}];
```

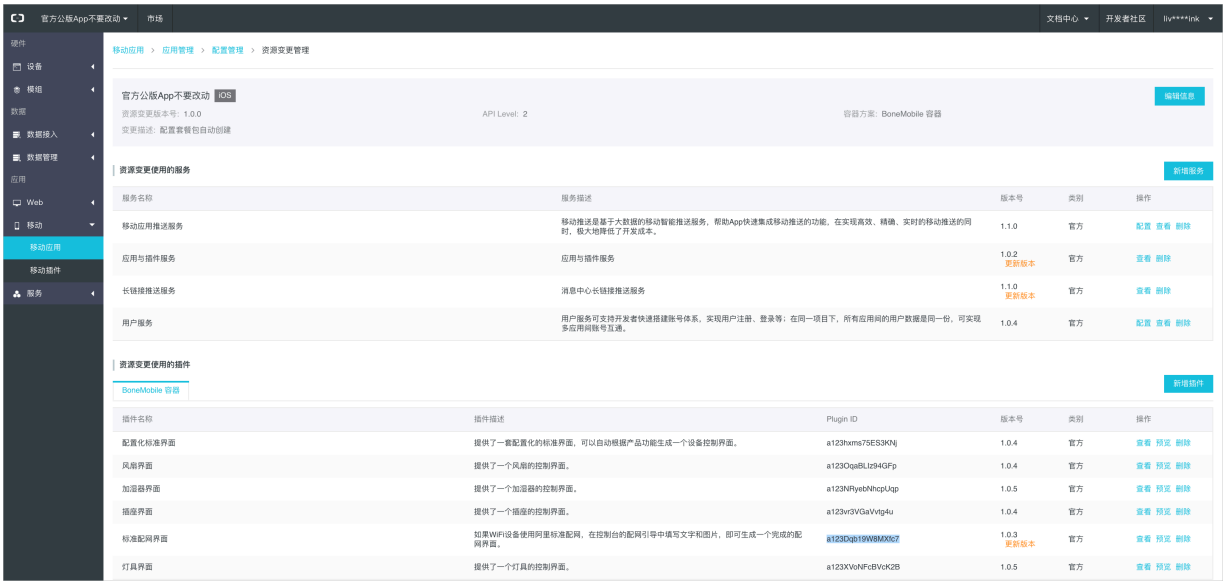
使用配网插件

您可以接入Bone容器，使用配网插件来完成配网的业务逻辑，这样只需关注配网之后的业务逻辑即可。

目前配网插件ID如下所示。

- 插件ID国内版：a123kfz2KdRdrfYc
- 插件ID国际版：a223c2beCJQ2Xpk2

插件ID的具体使用方式，参考见Bone容器部分。也可以参见配网开发指南中调用平台提供的配网插件的内容。



3.10.11 移动应用推送SDK

移动推送是基于大数据技术的移动云服务。帮助App快速集成移动推送的功能，在实现高效、精确、实时的移动推送的同时，降低了开发成本，提高用户活跃度和留存率。

依赖 SDK	概述
API 通道	提供API通道能力，和基础环境配置信息。

配置服务

本SDK涉及的功能依赖移动应用推送服务，需要先在控制台配置后才可以正常使用。

如何配置服务请参见移动应用推送开发指南。其中，iOS应用开发环境/生产环境为必配。

说明：

苹果公司于2019年9月正式发布iOS 13系统，为了不影响您正常使用移动推送功能，建议您马上更换新版本的SDK（pod 'AlicloudPushIoT', '1.9.5.5'）。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用说明

- 向苹果公司申请 APNs。
- 申请APNs的token，初始化SDK时会一并完成，无须额外操作。
- 推送通知到来监听。

```
/*
 * App处于启动状态时，通知打开回调
 */
```

```
- (void)application:(UIApplication*)application didReceiveRemoteNotification:(
NSDictionary*)userInfo {
    NSLog(@"Receive one notification.");
    // 取得APNS通知内容
    NSDictionary *aps = [userInfo valueForKey:@"aps"];
    // 内容
    NSString *content = [aps valueForKey:@"alert"];
    // badge数量
    NSInteger badge = [[aps valueForKey:@"badge"] integerValue];
    // 播放声音
    NSString *sound = [aps valueForKey:@"sound"];
    // 取得Extras字段内容
    NSString *Extras = [userInfo valueForKey:@"Extras"]; //服务端中Extras字段，key是自己
定义的
    NSLog(@"content = [%@], badge = [%ld], sound = [%@], Extras = [%@]", content, (long
)badge, sound, Extras);
    // iOS badge 清0
    application.applicationIconBadgeNumber = 0;
    // 通知打开回执上报
    [[IMSIotSmart sharedInstance] application:application didReceiveRemoteNoti
fication:userInfo];

}
```

以下业务逻辑已封装到SDK初始化中，开发者无需额外进行，只需关注接口调用即可。

- 关联移动推送到某账号，参见API服务：/uc/bindPushChannel
- 取消关联移动推送到某账号，参见API服务：/uc/unbindPushChannel

3.10.12 设备模型SDK

设备模型SDK提供了App端的设备模型（属性、事件、服务），用来开发设备界面，实现手机对设备的查看和控制。

依赖 SDK	概述
日志	基础依赖SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力
API通道SDK	API通道SDK，提供IoT业务协议封装的https请求能力，并通过整合安全组件来提升通道的安全性。
长连接通道SDK	长连接通道SDK，提供IoT业务协议封装的云端数据下行能力，为App提供订阅、发布消息的能力和支持请求响应模型。

初始化

初始化前需确保已#unique_103，初始化的操作请参见SDK初始化。

使用说明

- 获取本地发现且支持本地控制的设备列表

  本地通信功能是设备模型SDK提供的一项基础能力，它提供了在外网断连的情况下，局域网内设备控制的特性。当外网断连的情况下，本地通信模块会去搜寻当前局域网内的设备，如果被发现的设备是之前用户控制过的设备，那么此时可以通过本地通信链路去控制设备。

  在外网断连时，如向云端拉取用户账号下的设备列表会失败，此时可以使用以下接口获取当前可以本地通信控制的设备列表。由于本地发现设备是一个较长的过程，所以第一次调用此接口时有可能返回为空，此时需要允许用户刷新设备列表。

```
#import <IMSThingCapability/IMSThingCapability.h>

(NSArray<NSDictionary *> *)  devices = [kIMSThingManager getLocalAuthedDevice
DataList];
//其中 NSDictionary 保存了设备的详细信息。
```

- 创建设备

  设备抽象类IMSThing封装了设备对外提供的所有接口，包括获取设备的模型、设备的操作方法、设备的基础信息等。

```
#import <IMSThingCapability/IMSThingCapability.h>
/**
APP端可以使用 API 通道 SDK，请求接口 '/uc/listByAccount'，可以拿到当前账号下
所有绑定的设备列表，返回的设备信息中有设备的 iotId。

*/
_thingShell = [kIMSThingManager buildThing:_iotId];// _iotId为云端给设备颁发的唯一标
识
// 当不再需要使用时，请记得销毁，如下所示：
// [kIMSThingManager destroyThing:_thingShell];
// 如果需要获取设备的 iotId
// NSString * iotId = [_thingShell iotId];
```

- 设备控制

  App端需要控制设备时，SDK会根据当前的实际情况来决定控制请求是通过本地还是通过云端发送至设备。设备控制的接口协议为IMSThingActions。

  该协议定义了几个接口，用来对设备做控制，内部逻辑会实现通道选择。设备控制是基于物的模型对设备定义的属性、事件、服务进行操作。关于属性、事件、服务的描述，请参见：物的模型TSL介绍。

  方法调用时的入参、出参，请参见 物的模型服务。

  - 获取设备状态

```
[[_thingShell getThingActions] getStatus:^(IMSThingActionsResponse * _Nullable
response) {
    NSDictionary * properties = [response.dataObject valueForKey:@"data"];
```

```
    //格式如下:
    /* {
         "status":1 //
         "time":1232341455
       }
      说明：status表示设备生命周期，目前有以下几个状态，
       0:未激活；1：上线；3：离线；8：禁用；time表示当前状态的开始时间；
    */
}];
```

- 获取设备属性

```
[[_thingShell getThingActions] getPropertiesFull:^(IMSThingActionsResponse *
_Nullable response) {
    NSDictionary * properties = [response.dataObject valueForKey:@"data"];
    //格式如下:
    /* {
         "_sys_device_mid": {
          "time": 1516356290173,
          "value": "example.demo.module-id"
         },
         "WorkMode": {
          "time": 1516347450295,
          "value": 0
         },
         "_sys_device_pid": {
          "time": 1516356290173,
          "value": "example.demo.partner-id"
         }
       }
    */
}];
```

- 设置设备属性

```
NSDictionary * items = @{@"power":@"on", @"temperature":25};
//items 为key-value对，具体的值请参考 物的模型 TSL-属性以及其 datatype
[[_thingShell getThingActions] setProperties:items
                        responseHandler:^(IMSThingActionsResponse * _Nullable response
) {
                                if (response.success) {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                        [UiUtils showTip:[NSString stringWithFormat:@"设置属性
 %@成功", [property name]]];
                                    });
                                } else {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                        [UiUtils showTip:[NSString stringWithFormat:@"设置属性
 %@失败", [property name]]];

                                    });
                                }
                            }];
```

- 调用服务

```
// identifier 请参见 物的模型 TSL 中 Service 描述
// 调用服务时的入参 请参见 物的模型 TSL 中 Service 的 inputData
// 调用服务时的出参 请参见 物的模型 TSL 中 Service 的 outputData
[[ _thingShell getThingActions] invokeService:[service identifier]
                                params:valueDict
```

```
                    responseHandler:^(IMSThingActionsResponse * _Nullable response
) {
                                if (response.success) {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                        [UiUtils showTip:[NSString stringWithFormat:@"调用
服务 %@成功", [service name]]];
                                    });
                                } else {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                        [UiUtils showTip:[NSString stringWithFormat:@"调用
服务 %@失败", [service name]]];
                                    });
                                } }];
```

- 订阅所有事件

```
[_thingShell registerThingObserver:self]; //注册设备状态、属性变化、以及事件触发的
观察者。
// 具体参见 `IMSThingObserver`，并请注意注册的Observer的生命周期
// SDK只会Weak reference其实例,请开发者自己保证Observer的生命周期。
// 不需要监听时，请注销[_thingShell unregisterThingObserver:self];
```

- 清理缓存

  SDK 在使用过程中会保存本地通信加速的相关数据到手机沙盒目录，当手机账号登出时记得清理这些缓存。

```
[kIMSThingManager clearLocalCache];
```

获取物的模型

物模型是对设备是什么、能做什么的描述，包括设备身份标识、连接状态、描述信息，以及设备的属性（properties）、服务（services）、事件（events），后三者构成了设备的功能定义。阿里云IoT平台通过定义一种物的描述语言来描述物模型，称之为 TSL（即 Thing Specification Language）。

当IMSThingObserver didThingTslLoad方法被回调时，可以获得解析完成的物的模型。

```
IMSThingProfile * Profile = [thingShell getThingProfile];
//其中物的三要素 保存在 IMSThingProfile 中。
_thingProperties = [[_thingShell getThingProfile] allPropertiesOfModel];
_thingEvents =  [[_thingShell getThingProfile] allEventsOfModel];
_thingServices =  [[_thingShell getThingProfile] allServicesOfModel];
// 注意，此处拿到的 properties，events，services 是物的模型中的物的三要素。
// 具体请参见 IoT-TSL 规范
```

开发者也可以使用云端接口来获取原始的物的模型，参见获取物的模板。

蓝牙功能API介绍

蓝牙设备受连接特性的约束，往往无法直接跟生活物联网云端平台连接，因而需要借助一个网关设备来实现蓝牙设备与生活物联网云端平台的连接通道。而手机在这一过程中也可以充当网关的角色。

此 SDK 额外提供以下几方面的能力如下。

- 提供发现蓝牙设备/连接蓝牙设备的能力
- 提供连云通道，可以供蓝牙设备数据上下云
- 提供蓝牙设备控制与数据获取的能力

接入蓝牙设备需要引入相关依赖如下。

依赖 SDK	概述
蓝牙Breeze SDK	Breeze SDK是按照规范实现的手机端蓝牙 SDK，方便合作厂商在手机端快速接入蓝牙功能。Breeze SDK包含的主要功能有：设备发现连接，设备通信，加密传输，大数据传输等。
移动端设备网关SDK	移动端设备网关SDK，运行于App上的子设备网关，对于无法直连网络的子设备，如蓝牙设备，提供子设备的管理功能，如子设备添加拓扑，删除拓扑，上线，下线以及数据上下行等。

- 初始化移动端设备网关SDK

  此功能模块依赖移动端设备网关SDK。使用前请先初始化该SDK。

- 蓝牙设备的发现

```
#import <IMSThingCapability/IMSThingCapability.h>

[[IMSThingDiscoveryRegistry sharedRegistry] startDiscoveryWithFilter:filterParams
                                                       didFoundBlock:^(NSArray * _Nullable result, NSError *
_Nullable error) {
                                                           if ([result count]) {
                                                               [result enumerateObjectsUsingBlock:^(id<
IMSLocalDevice> item, NSUInteger idx, BOOL * _Nonnull stop) {
                                                                   NSString * productKey = item.productKey;
                                                                   NSString * bleMac = item.deviceName;//对于
Breeze 蓝牙设备，这里得到的是设备的MAC地址

                                                               }];
                                                           }
    }];
```

- 蓝牙设备的添加绑定流程

  1. 启动蓝牙连接，需要在使用蓝牙设备前调用。

```
#import <IMSThingCapability/IMSThingCapability.h>

IMSThing * thing = [[IMSThingManager sharedManager] buildThing:iotId];
[thing startLocalConnect];
```

  2. 断开蓝牙连接，需要在不需要使用蓝牙设备时调用。

```
#import <IMSThingCapability/IMSThingCapability.h>

IMSThing * thing = [[IMSThingManager sharedManager] buildThing:iotId];
```

```
[thing stopLocalConnect];
```

3. 蓝牙连接状态变化需要传入Delegate。

```
IMSThing * thing = [[IMSThingManager sharedManager] buildThing:iotId];
[thing registerThingObserver:self];//请参见IMSThingObserver
```

- 蓝牙设备的控制

  蓝牙设备的控制以及信息获取跟WiFi设备的API是一致的，可以参考本文档前述章节。

3.10.13 蓝牙OTA SDK

SDK提供蓝牙OTA业务的App端解决方案，提供了蓝牙设备固件升级的能力。

依赖 SDK	概述
蓝牙	Breeze SDK 是按照规范实现的手机端蓝牙 SDK，方便合作厂商在手机端快速接入蓝牙功能。Breeze SDK 包含的主要功能有：设备发现连接，设备通信，加密传输，大数据传输等。
日志	基础依赖 SDK，提供客户端统一日志打印，日志等级控制，分模块日志隔离等能力。
API 通道	提供 API 通道能力，和基础环境配置信息。

初始化

OTA SDK的初始化依赖API通道的初始化。在进行蓝牙设备OTA固件升级时，依赖蓝牙设备的连接绑定，需要使用到蓝牙 SDK（即BreezeSDK）初始化配置和绑定流程。

OTA SDK初始化如下。

```
#import
//self.breeze如何得到请参见[蓝牙 SDK]的初始化部分
self.otaBiz = [LKLinkOtaBusiness setupOtaBiz:self.breeze];
```

使用说明

OTA基本流程是：用户绑定设备，得到一个iotId后上报版本，再进行OTA升级或取消。

其中上报固件版本：开发者在上述蓝牙设备连接、绑定后调用云端提供的上传设备固件版本号即可，上传固件版本号请参见：上报版本号。

- 开始OTA升级流程

  1. 预检查事件（LKOTANotificationTypeCheck）

  2. 下载升级包事件（LKOTANotificationTypeDownload）

  3. 传输升级包事件（LKOTANotificationTypeTransmit）

  4. 重启升级事（LKOTANotificationTypeReboot）

  5. OTA 结束（LKOTANotificationTypeFinish）

```
#import
[self.otaBiz startUpgrade:self.iotId alcsOTA:NO  type:LKOTADeviceTypeBle lisener:^(
LKOTANotificationType type, NSDictionary *result) {
    if (type == LKOTANotificationTypeCheck) {
        NSError * err = [result objectForKey:@"error"];
        if (err != nil) {
            NSString * descrip = err.localizedDescription;
            [self insertMsgWithColor:@"red" main:@"升级时预检查失败" detail:descrip];
        } else {
            [self insertMsgWithColor:@"blue" main:@"升级时预检查完成" detail:@"成功"];
        }
    } else if (type == LKOTANotificationTypeDownload) {
        NSDictionary * subResult = [result objectForKey:@"result"];
        NSError * err = [result objectForKey:@"error"];
        if (err != nil) {
            NSString * descrip = err.localizedDescription;
             [self insertMsgWithColor:@"red" main:@"下载OTA包失败" detail:descrip];
        } else {
            int progress = [[subResult valueForKey:@"progress"]intValue];
            [self updateProgressLabel:@"下载OTA包" withProgress:progress];
        }
    } else if (type == LKOTANotificationTypeTransmit) {
        NSDictionary * subResult = [result objectForKey:@"result"];
        NSError * err = [result objectForKey:@"error"];
        if (err != nil) {
            NSString * descrip = err.localizedDescription;
            [self insertMsgWithColor:@"red" main:@"传输OTA包失败" detail:descrip];
        } else {
            int progress = [[subResult valueForKey:@"progress"]intValue];
            [self updateProgressLabel:@"传输OTA包" withProgress:progress];
        }
    } else if (type == LKOTANotificationTypeReboot) {
        NSDictionary * subResult = [result objectForKey:@"result"];
        NSError * err = [result objectForKey:@"error"];
        if (err != nil) {
            NSString * descrip = err.localizedDescription;
            [self insertMsgWithColor:@"red" main:@"重启设备升级失败" detail:descrip];
        } else {
            int progress = [[subResult valueForKey:@"progress"]intValue];
            [self updateProgressLabel:@"重启设备升级" withProgress:progress];
        }
    } else if (type == LKOTANotificationTypeFinish) {
        //NSDictionary * subResult = [result objectForKey:@"result"];
        NSError * err = [result objectForKey:@"error"];
        if (err != nil) {
            NSString * descrip = err.localizedDescription;
            [self insertMsgWithColor:@"red" main:@"设备升级失败" detail:descrip];
        } else {
            [self insertMsgWithColor:@"yellow" main:@"设备升级成功" detail:nil];
        }
    }
```

```
}];
```

- 终止 OTA流程

  OTA完成后，必须要通过stopUpgrade停止掉，可以停止正在进行中的OTA过程。

```
#import
[self.otaBiz stopUpgrade];
```

3.10.14 Link Visual视频Media SDK

本节主要为Link Visual iOS版本的音视频模块使用说明，包含视频播放、语音对讲的功能。

依赖SDK	概述
API 通道	提供API通道能力
长连通道	P2P需要长连接通道

配置工程

- 通过CocoaPods工具配置工程

  1. 在podfile中添加引用源。

```
source 'https://github.com/aliyun/aliyun-specs.git'
```

  2. 添加库依赖。

```
//如遇IMS***版本冲突，请按照pod提示修改podfile对应sdk版本
//如podfile中没有的sdk库，请查看podfile.lock对应修改podfile
pod 'IMSLinkVisualMedia', '1.4.0'
```

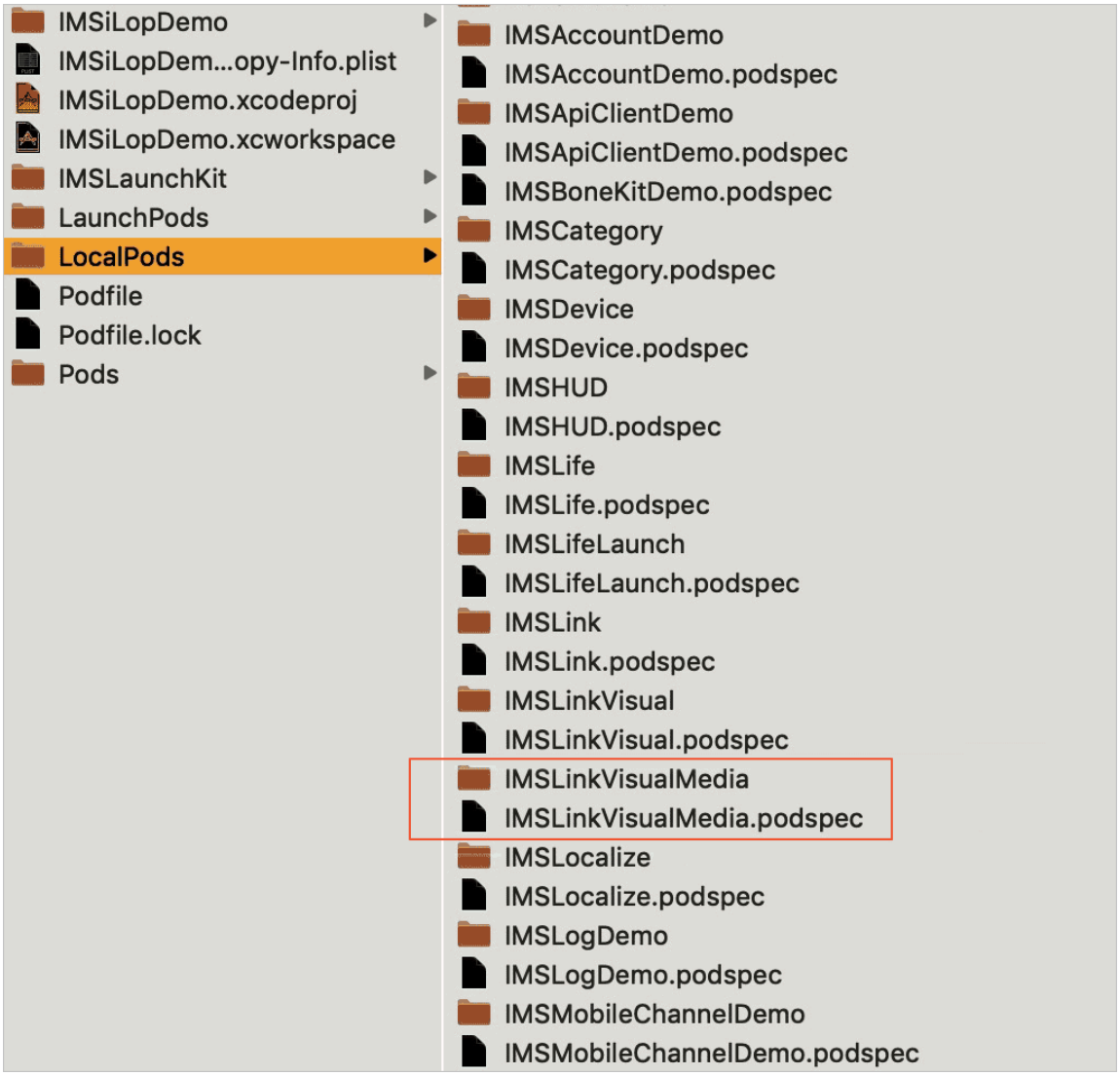
  3. 执行**pod update**，则库安装完毕。

- 通过引入本地库项目配置工程（本地包请使用该方式集成，例如本地包为podspec）

  1. 在podfile中添加本地路径。

```
pod 'IMSLinkVisualMedia', :path => 'LocalPods/IMSLinkVisualMedia.podspec'
```

  2. 拷贝以下SDK文件至项目的LocalPods目录下。



  3. 将podspec拷贝以下文件至项目内LocalPods目录下。

```
IMSLinkVisualMedia.podspec
IMSLinkVisualMedia
//文件夹 IMSLinkVisualMedia文件夹内包含以下framework
IMSLinkVisualMedia.framework
LibRtmp.framework
LinkVisualClientSDK.framework
FFmpeg.framework
```

  4. 执行**pod update**，则库安装完毕。

播放器说明

播放器按功能分为三种，提供了以下功能。

- 直播播放器

  用于RTMP直播源，具有时延低的特点，支持P2P（需使用接入LinkVisual设备端SDK的摄像头）。

- TF卡点播播放器

  用于设备录像回放，可调整播放进度，支持P2P（需使用接入LinkVisual设备端SDK的摄像头）。

- 云端HLS播放器

  用于基于HLS的云端录像回放的播放，支持MPEG-TS和FMP4容器，AES-128加密方式。

功能	直播播放器	TF卡点播播放器	云端HLS播放器
视频播放	#	#	#
音频播放	#	#	#
暂停/恢复	x	#	#
跳至指定位置播放	X	#	#
总时长	x	#	#
当前播放进度	x	#	#
播放器状态变更通知	#	#	#
静音	#	#	#
变速播放	x	x	#
循环播放	x	x	x
画面缩放模式设置	#	#	#
播放器截图	#	#	#
边播边录	#	#	#

播放器使用指南

1. 引入框架。

```
//框架引入
#import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>
```

2. 设置播放器日志。

```
//打开播放器日志  建议为IMSLinkVisualMediaLogInfo
[IMSLinkVisualPlayerViewController setLogLevel:IMSLinkVisualMediaLogInfo];
```

3. 创建播放器。

```
    //创建播放器
```

```
    IMSLinkVisualPlayerViewController *player = [[IMSLinkVisualPlayerViewController
alloc] init];
    //因为播放器为viewController，所以addChildViewController
    [self addChildViewController:player];
    //设置播放器frame
    player.view.autoresizingMask = UIViewAutoresizingFlexibleWidth | UIViewAuto
resizingFlexibleHeight;
    player.view.frame = self.view.bounds;
    //添加播放器view到指定位置
    [self.view insertSubview:player.view atIndex:0];
    //播放器代理
    player.delegate = self;
```

4. 设置播放源。

- 直播播放器

```
    //设置参数 iotId
    [player setDataSource:self.iotId
            streamType:IMSLinkVisualPlayerLiveStreamTypeMain
        needForceIFrame:true];
```

```
    //设置参数 rtmpPath rtmp地址
    //needEncrypt false 不需要加密，参数iv和key都传nil
    [player setDataSource_Live:rtmpPath
                needEncrypt:needEncrypt
                        iv:iv
                       key:key];
```

- TF卡点播播放器

```
    //设置参数（时间）
    //时间接口中vodStartTime为当天0点时间戳，vodEndTime为当天23点59秒时间戳，
 seekTime为本次播放的起始时间（秒）
    [player setDataSource:iotId
          vodStartTime:timestamp
           vodEndTime:timestamp + dayOfSecond seekTime:time];
```

```
    //设置参数（文件）
    [player setDataSource:iotId
           vodFileName:vodFileName];
```

```
    //设置参数 rtmpPath rtmp地址
    //needEncrypt false 不需要加密，参数iv和key都传nil
    [player setDataSource_Vod:rtmpPath
                needEncrypt:needEncrypt
                        iv:iv
                       key:key];
```

- 云端HLS播放器

  设置云端HLS

```
    //设置参数 （文件）
    [player setDataSource:iotId
           hlsFileName:file.fileName
             seekTime:seekTime];
```

```
     //设置参数 HLSPath http***.m3u8 seekTime的单位：秒
```

```
    [player setDataSource_HLS:hlsPath
                    seekTime:seekTime];
```

5. 设置并使用播放器功能。

   - 开始播放

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     // 开始播放
     [self.player start];
     ```

   - 静音播放器

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     //播放器静音，对讲不播放接收到的声音通过mute实现
     self.player.mute = true;
     ```

   - 停止播放

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     // 停止播放，结束播放器
     [self.player stop];
     ```

   - 暂停播放（仅录播）

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     // 暂停播放
     [self.player pause];
     ```

   - 恢复播放（仅录播）

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     // 恢复播放
     [self.player restore];
     ```

   - 倍速播放（仅录播）

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     // 倍速播放
     self.player.playSpeed = PlaySpeed_NORMAL;
     ```

   - 视频截图

     ```
     #import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

     //获取视频截图
     UIImage *image = [self.player videoSnapshot];
     //获取视频截图
     UIImage *image = [hlsPlayer videoSnapshot];
     ```

- 视频截图

```
#import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

//获取视频截图
UIImage *image = [self.player videoSnapshot];
//获取视频截图
UIImage *image = [hlsPlayer videoSnapshot];
```

- 边录边播 （不支持hlsPlayer）

```
#import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>

- (IBAction)recordVideoButtonClick:(UIButton *)sender {
    sender.selected = !sender.selected;
    NSString *tmpPath = [NSTemporaryDirectory() stringByAppendingPathComponent
:@"tmp.mp4"];
    if (sender.selected) {
        [self.player startRecordVideoWithfilePath:tmpPath];
    } else {
        [self.player stopRecordVideo];
    }
}
```

- 设置播放器缓存

```
#import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>
/**
 //设置播放器缓存（缓存越大，延迟越高），一帧缓存延迟40ms，默认为5帧
 @param buffer 缓存大小 0 -- 16
 */
[self.player setDisplyBuffer:10];
```

- 设置播放器渲染模式

```
/// 播放器渲染模式
typedef NS_ENUM(NSUInteger, IMSLinkVisualPlayerDisplayMode){
    /// 客户对YUV数据不可见，完全由SDK处理和渲染，默认使用这个模式
    IMSLinkVisualPlayerDisplayMode_SDK,
    /// 允许客户对YUV数据进行二次加工，然后再还给SDK渲染
    IMSLinkVisualPlayerDisplayMode_Client_SDK,
    /// YUV数据完全交由客户处理和渲染
    IMSLinkVisualPlayerDisplayMode_Client
};

/// 设置播放器渲染模式，默认 IMSLinkVisualPlayerDisplayMode_SDK 模式
@property (assign, nonatomic) IMSLinkVisualPlayerDisplayMode lvDisplayMode;

/**
 直播、点播的共用代理接口
 视频帧YUV数据回调，同步接口，帧数据的内存和SDK是共用的，如外部需要缓存数据，必须把数据拷贝出去
 @see IMSLinkVisualPlayerViewController
 @param frame 视频帧YUV数据
 */
```

```
- (void)linkVisual:(IMSLinkVisualPlayerViewController *_Nullable)player withFrame:
(IMSPlayerVideoFrame *_Nullable)frame;
```

6. 查看播放器状态。

```
/// 播放器状态
typedef NS_ENUM(NSUInteger,IMSLinkVisualPlayerState){
    /// 空闲
    IMSLinkVisualPlayerStateIdle = 0,
    /// 缓冲中
    IMSLinkVisualPlayerStateBuffering = 2,
    /// 开始播放
    IMSLinkVisualPlayerStateStartPlay = 4,
    /// 暂停播放
    IMSLinkVisualPlayerStatePausePlay = 8,
    /// 切换到后台
    IMSLinkVisualPlayerStateBackground = 16,
};
```

7. 监听回调结果。

```
@protocol IMSLinkVisualDelegate <NSObject>

/*------------------直播、点播共用代理接口---------------------*/

/**
 直播、点播 共用接口
 连接成功
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualConnect:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 直播、点播 共用接口
 准备完成进入播放状态
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualReady:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 直播、点播 共用接口
 播放成功停止（正常停止）
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualStop:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 直播、点播 共用接口
 连接错误回调 (所有直播点播错误，都通过这个代理回调)
 @see IMSLinkVisualPlayerViewController
 @param error 错误信息 IMSLinkVisualPlayerError枚举 对应 error.code
 */
- (void)linkVisual:(IMSLinkVisualPlayerViewController *_Nullable)player errorOccurred:
(NSError *_Nullable)error;

/**
 直播、点播 共用接口
 视频帧YUV数据回调，同步接口，帧数据的内存和SDK是共用的，如外部需要缓存数据，必须
把数据拷贝出去
 @see IMSLinkVisualPlayerViewController
 @param frame 视频帧YUV数据
 */
```

```
- (void)linkVisual:(IMSLinkVisualPlayerViewController *_Nullable)player withFrame:(
IMSPlayerVideoFrame *_Nullable)frame;


/*--------------------点播独占代理接口----------------------*/

/**
 点播
 seek成功完成
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualReplaySeekReady:(IMSLinkVisualPlayerViewController *_Nullable)
player;

/**
 点播
 恢复播放成功完成
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualRestore:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 点播
 暂停播放成功完成
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualPause:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 点播
 播放到结尾的回调
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualPlayEnd:(IMSLinkVisualPlayerViewController *_Nullable)player;

/**
 点播
 当前时间回调
 @see IMSLinkVisualPlayerViewController
 */
- (void)linkVisualReplay:(IMSLinkVisualPlayerViewController *_Nullable)player
          currentTime:(NSInteger)currentTime;

/// SEI回调接口
/// @param player  播放器
/// @param data  回调二进制数据 data.length为数据长度
/// @param timeStamp  时间戳
- (void)linkVisualReplay:(IMSLinkVisualPlayerViewController *_Nullable)player
               buffer:(NSData*_Nullable)data
            timeStamp:(NSInteger)timeStamp;
```

```
@end
```

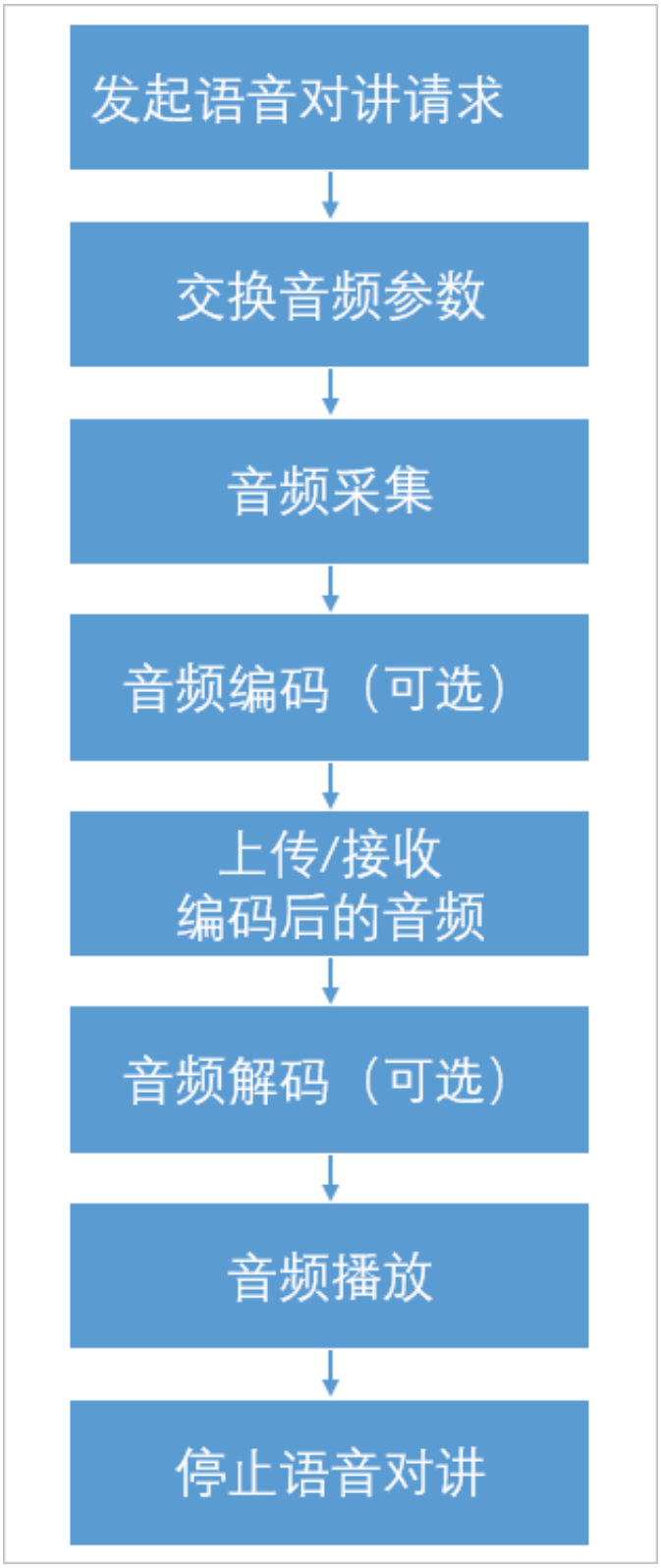
播放器错误列表

错误主码（PlayerMainError）	描述	子码（解析userInfo中subCode获取）	描述
ErrorSource	数据源相关错误	IMSLinkVisualPlayerErrorConnect	建立连接失败
		IMSLinkVisualPlayerErrorEncrypt	无效的解密密钥
		IMSLinkVisualPlayerErrorUrl	无效的播放地址
		IMSLinkVisualPlayerErrorDataSource	数据源错误或未设置
		IMSLinkVisualPlayerErrorGetURL	获取链接失败
		IMSLinkVisualPlayerErrorStop	关闭失败
		IMSLinkVisualPlayerErrorStart	启动失败
ErrorRender	渲染相关错误	IMSLinkVisualPlayerErrorDecode	解码错误
ErrorUnexpected	不符合预期错误	IMSLinkVisualPlayerErrorStream	接收数据流失败

语音对讲介绍

提供App和IPC设备之间端到端的双向实时音频传输能力。

语音对讲支持双工模式，App端需要实现以下流程。



支持的音频格式	编码	解码
AAC_LC	#	#
G711A	#	#
G711U	#	#

语音对讲使用指南

- 创建语音对讲编码参数

  参数为录制语音数据的参数，用来与设备端对齐。

```
//常用对讲模式为以下参数，不建议更改，支持8K和16K
//sampleRate 8000 或 16000 以设备支持为准
   IMSLinkVisualAudioParams *intercomEncodeParams = [[IMSLinkVisualAudioParams
alloc] init];
   intercomEncodeParams.sampleRate = 8000;
   intercomEncodeParams.channel = 1;
   intercomEncodeParams.bitsPerSample = 16;
   intercomEncodeParams.format = IMSLinkVisualAudioFormatG711a;
```

- 设置语音对讲参数

  创建编码器用于编码语音数据。

```
   //player 为播放器实例 IMSLinkVisualPlayerViewController
   player.intercomEncodeParams = intercomEncodeParams;
```

- 设置语音对讲回调代理

  用于语音对讲开始与结束流程的消息回调代理，开始对讲前，先增加语音对讲状态回调方法。

```
player.intercomDelegate = self;
```

- 开语音对讲

  设置播放器为对讲模式并开启对讲功能。

```
[player startIntercom:IMSLinkVisualIntercomAudioModeIntercom];
```

- 停止语音对讲

```
[player stopIntercom];
```

语音对讲状态回调方法与触发规则如下。

```
#pragma mark - IMSLinkVisualIntercomDelegate
#pragma mark 语音对讲连接服务器
- (void)linkVisualIntercomConnect:(IMSLinkVisualPlayerViewController *)player {
   //语音对讲连接服务器成功
   //连接成功不代表可以可以立即发送语音
}#pragma mark 语音对讲ready
//当对讲连接于服务端连接建立后，若对端已就绪，会告知本端事件talk ready. 此时向对端发送
音频数据都会被对端收到并处理.
- (void)linkVisualIntercomReady:(IMSLinkVisualPlayerViewController * _Nullable)player {
}
#pragma mark 对讲接收到设备端音频参数
//当语音对讲通道建立后，若对端支持录音，会先收到对端发送过来的音频参数信息，
//后续对端发送的音频数据按照此音频参数来做解码. 该事件是语音对讲通道建立成功的标志.
//可以在此时构建音频播放器实例用于对端采集音频的实时播放.

- (void)linkVisualIntercom:(IMSLinkVisualPlayerViewController *)player audioParams:(
IMSLinkVisualAudioParams *)params {
}
```

```
#pragma mark 对讲接收到设备端音频数据
//若对端支持录音，语音对讲中会持续不断的收到对端发送过来的音频数据.
- (void)linkVisualIntercom:(IMSLinkVisualPlayerViewController *)intercom audioData:(
NSData *)data {
    //无特殊需求可以不收集数据
    if (data) {
        //playBuffers为自定义可变数组收集数据
        [self.playBuffers addObject:data];
    }
}
#pragma mark 录音数据完成回调
//语音对讲开始后，会收集不断收到mic采集的数据，根据对讲的状态开始发送的数据

- (void)linkVisualIntercom:(IMSLinkVisualPlayerViewController *)intercom didRecordData
:(NSData *)data{
    if (data) {
        //发送录音数据 audioBuffers为自定义可变数组收集数据
        [player sendAudioData:audioData];
    }
}
#pragma mark 语音对讲停止
- (void)linkVisualIntercomStop:(IMSLinkVisualPlayerViewController *)player {
}
#pragma mark 语音对讲出错
- (void)linkVisualIntercom:(IMSLinkVisualPlayerViewController *)player
            errorOccurred:(NSError *)error {
 //处理对讲出错的错误信息
//error.code 与 IMSLinkVisualIntercomError 对应

}
```

对讲错误列表

语音对讲错误状态码	描述
IMSLinkVisualIntercomErrorGetURL	获取链接失败
IMSLinkVisualIntercomErrorParams	对讲参数错误
IMSLinkVisualIntercomErrorStart	启动对讲失败
IMSLinkVisualIntercomErrorConnect	语音流建立失败
IMSLinkVisualIntercomErrorRecorder	录音失败
IMSLinkVisualIntercomErrorStream	接收数据流失败
IMSLinkVisualIntercomErrorDecode	解码错误
IMSLinkVisualIntercomErrorSendData	发送语音对讲数据失败
IMSLinkVisualIntercomErrorStop	关闭失败

3.10.15 常见问题

介绍iOS自有App开发过程中的常见问题以及解决方法。

Q：用户退出登录后，再次登录，无法获取到新的token

A：请确保退出登录时，调用了退出登录API。

```
#import <IMSAccount/IMSAccountService.h>
[[IMSAccountService sharedService] logout];
```

Q：自有App调用内置账号登录时，返回以下错误

```
15:48:04.904551+0800 pelfinpet[40548:1992573] *** Terminating app due to uncaught
exception 'NSInvalidArgumentException', reason: '*** -[__NSPlaceholderArray initWithOb
jects:count:]: attempt to insert nil object from objects[0]'
```

A：xib位置不正确，需要将ALBBOpenAccountUI.framework中的xib放置到主工程目录下。

Q：iOS自有App启动后，日志打印如下错误

出现类似“IMSPackage - AKOPModuleUpdater.m:204 | 解压出错”的内容。

```
2019-01-28 16:33:15.165915+0800 com.hnzhzn.zhzj[4107:1483985] IMSPackage -
AKOPModuleUpdater.m:166 | 解压Zip包：E7142E1B4BEF6E7D58126924D787701F-release-
a123N7GcC7vD4dFH-boneMobile-1.0.4-fdde7a3d0576b493b04b0d5ec83d79c3-source2019
-01-28 16:33:15.209104+0800 com.hnzhzn.zhzj[4107:1483985] 1 entries in the zip file2019
-01-28 16:33:15.210651+0800 com.hnzhzn.zhzj[4107:1483985] IMSPackage - AKOPModule
Updater.m:204 | 解压出错：zip: /Users/zhanhe/Library/Developer/CoreSimulator/
Devices/D0137B06-83AB-4EDA-818D-22E8FABF6520/data/Containers/Data/Application/
D09F0B02-1891-4C5F-A76A-4E990CE41040/Library/Application Support/com.hnzhzn.zhzj
/OfflinePackage/download/E7142E1B4BEF6E7D58126924D787701F-release-a123N7GcC7
vD4dFH-boneMobile-1.0.4-fdde7a3d0576b493b04b0d5ec83d79c3-source.zip, dstPath
: /Users/zhanhe/Library/Developer/CoreSimulator/Devices/D0137B06-83AB-4EDA-
818D-22E8FABF6520/data/Containers/Data/Application/D09F0B02-1891-4C5F-A76A-
4E990CE41040/Library/Application Support/com.hnzhzn.zhzj/OfflinePackage/modules
/E7142E1B4BEF6E7D58126924D787701F-release-a123N7GcC7vD4dFH-boneMobile-1.0.
4-fdde7a3d0576b493b04b0d5ec83d79c3-source/2.2.4, error: Error Domain=ZipArchive
BlocksErrorDomain Code=1 "Failed to open output file for writing" UserInfo={NSLocalize
dDescription=Failed to open output file for writing}2019-01-28 16:33:15.210941+0800
 com.hnzhzn.zhzj[4107:1483985] IMSPackage - AKOPackageUpdater.m:390 | module:
E7142E1B4BEF6E7D58126924D787701F-release-a123N7GcC7vD4dFH-boneMobile-1.0.4-
fdde7a3d0576b493b04b0d5ec83d79c3-source, error: Error Domain=ZipArchiveBlocksErro
rDomain Code=1 "Failed to open output file for writing" UserInfo={NSLocalizedDescription
=Failed to open output file for writing}2019-01-28 16:33:15.211685+0800 com.hnzhzn.zhzj
[4107:1482908] IMSPackage - AKOPackageUpdater.m:407 | 结束更新任务链： {"E7142E1B4B
EF6E7D58126924D787701F-release-a123N7GcC7vD4dFH-boneMobile-1.0.4-fdde7a3d05
76b493b04b0d5ec83d79c3-source" = "<AKOModuleConfig: 0x60000102d840>";}, error:
Error Domain=ZipArchiveBlocksErrorDomain Code=1 "Failed to open output file for writing"
UserInfo={NSLocalizedDescription=Failed to open output file for writing}20
```

A：用户本地使用了ZipArchive类似的库，例如当前客户使用的- SSZipArchive (= 2.1.2)和我们提供的ZipArchive发生冲突，此时需要将- SSZipArchive (= 2.1.2)删除。

Q：三方自有账号登录返回403错误，类似日志如下

```
<NSHTTPURLResponse: 0x600001b7a560> { URL: https://sdk.openaccount.aliyun.com/api/
test/loginbyoauth.json } { Status Code: 403, Headers {
```

```
  "Access-Control-Allow-Origin" =    (
      "*"
  );
  "Content-Length" =    (
      0
  );
  "Content-Type" =    (
      "text/plain;charset=UTF-8"
  );
  Date =    (
      "Mon, 24 Jun 2019 02:42:56 GMT"
  );
  Server =    (
      nginx
  );
  "access-control-allow-headers" =    (
      "X-Requested-With,X-Sequence,X-Ca-Key,X-Ca-Secret,X-Ca-Version,X-Ca-Timestamp
,X-Ca-Nonce,X-Ca-API-Key,X-Ca-Stage,X-Ca-Client-DeviceId,X-Ca-Client-AppId,X-Ca-
Signature,X-Ca-Signature-Headers,X-Ca-Signature-Method,X-Forwarded-For,X-Ca-Date,X-
Ca-Request-Mode,Authorization,Content-Type,Accept,Accept-Ranges,Cache-Control,Range
,Content-MD5"
  );
  "access-control-allow-methods" =    (
      "GET,POST,PUT,DELETE,HEAD,OPTIONS,PATCH"
  );
  "access-control-max-age" =    (
      172800
  );
  "x-ca-error-message" =    (
      Unauthorized
  );
  "x-ca-request-id" =    (
      "2C90C481-0AEE-407B-A970-EC2259C20C5A"
  );
}}
==============================
2019-06-24 10:42:56.695167+0800 zoozee-ios[4126:434355] [Warn]
错误编码：554
错误类型：ALBBOpenAccount - RPC_REQUEST_ERROR
错误消息：原因为Error Domain=com.alibaba.openaccount Code=552 "系统繁忙，请稍后重试"
UserInfo={NSLocalizedDescription=系统繁忙，请稍后重试}
错误建议：请稍后重试
```

A：按照SDK的API level版本，配置以下参数。

- API level 8及以上版本SDK：

```
#import <IMSIotSmart/IMSIotSmart.h>

    IMSIotSmartConfig *config = [IMSIotSmartConfig new];
    config.regionType = REGION_CHINA_ONLY; //中国站；如是国际站：config.regionType
 = REGION_ALL;
    config.appType  = APP_TYPE_PRODUCTION;//取值范围参见枚举类型 `IMSAppType`
    [IMSIotSmart sharedInstance].config = config;
```

- API level 7及其以下SDK版本：

```
[IMSConfiguration initWithHost:@"api.link.aliyun.com" serverEnv:IMSServerRelease];//
API通道SDK初始化
```

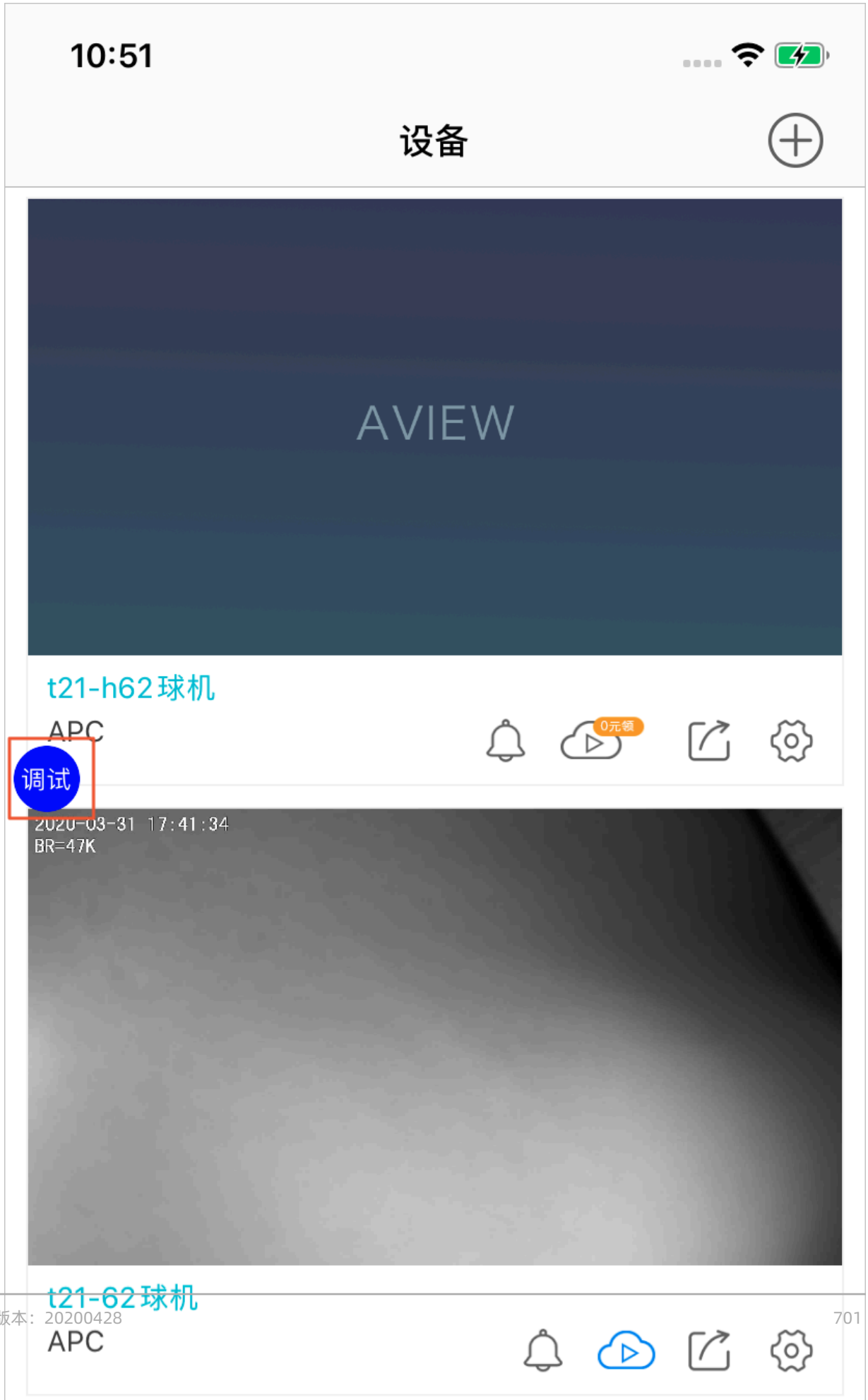
```
[accountSDK setTaeSDKEnvironment:TaeSDKEnvironmentRelease];//账号SDK初始化
```

Q：在运行工程crash时会报错如下

```
*** Terminating app due to uncaught exception 'com.openaccount.SecurityGuard.
SECURITY_PICTURE_NOT_FOUND', reason: '(null)'
```

A：检查工程的**bundleID**与平台上创建App时输入的bundleID（包名）是否一致。

Q：iOS自有App页面上如何去掉“调试”两字



A：选择以下任一方法操作。

- 注释掉Profied中关于IMSDebug的引用。

```
pod 'IMSDebug', :path => 'LocalPods/IMSDebug.podspec'
```

- 在工程代码中搜索@"调试"，注释掉代码中的相关内容。

4 App模板开发指南

4.1 概述

生活物联网平台为您提供了App源码模板，不仅可以满足自定义App的需求，而且可以简化App开发工作。您简单配置后，即可打包构建成一个自有品牌App，并上架应用市场。

了解App模板

生活物联网平台为您提供了两类App源码模板，每套App模板的应用场景、源码开放范围都不相同。整体区别如下。

模板类别	模板介绍	源码开放范围
基础版	基于云智能App（即公版App）开放的智能生活类App，您需简单配置后即可搭建自己的“云智能App”。详细介绍请参见云智能App系列模板介绍。 该模板为付费模板，详细请参见#unique_124。	除了支持App模板的通用修改外，还支持更多自定义内容，详细请参见云智能App系列模板介绍。
免费版	提供了一套免费的便于快速上手的Demo App模板，帮助您更好地熟悉SDK和插件的使用方法。用于初学者熟悉平台。	仅支持App模板的通用修改，请参见App版本的通用修改。

说明：
由于App模板之间的代码差异较大，一旦确定了App模板后，不建议您经常更换模板。

了解模板的版本

每个App模板会持续更新多个版本，当平台有功能更新、问题修复时，平台会基于同一套App模板发布新版本。我们建议您使用最新版本，便于您享受到我们提供的最新功能。

升级App版本时，您需要注意App模板的升级机制。

- 同一种模板内，可以对您的App进行版本升级，但不可以降级。

- 升级后，您需要重新生成App源码。

  同一App模板的各版本之间的源码通常只有少量差异。因此，您重新生成App源码后，可以很便捷地在本地合并代码。

> **说明：**
> 不同的App模板之间，整体代码框架完全不同，请勿合并代码。

4.2 App模板的通用修改

4.2.1 Android App通用修改项

该文档用来汇总所有Android App模板自定义时的通用修改项，如修改App版本号、修改App界面文案、修改App图片、颜色等。

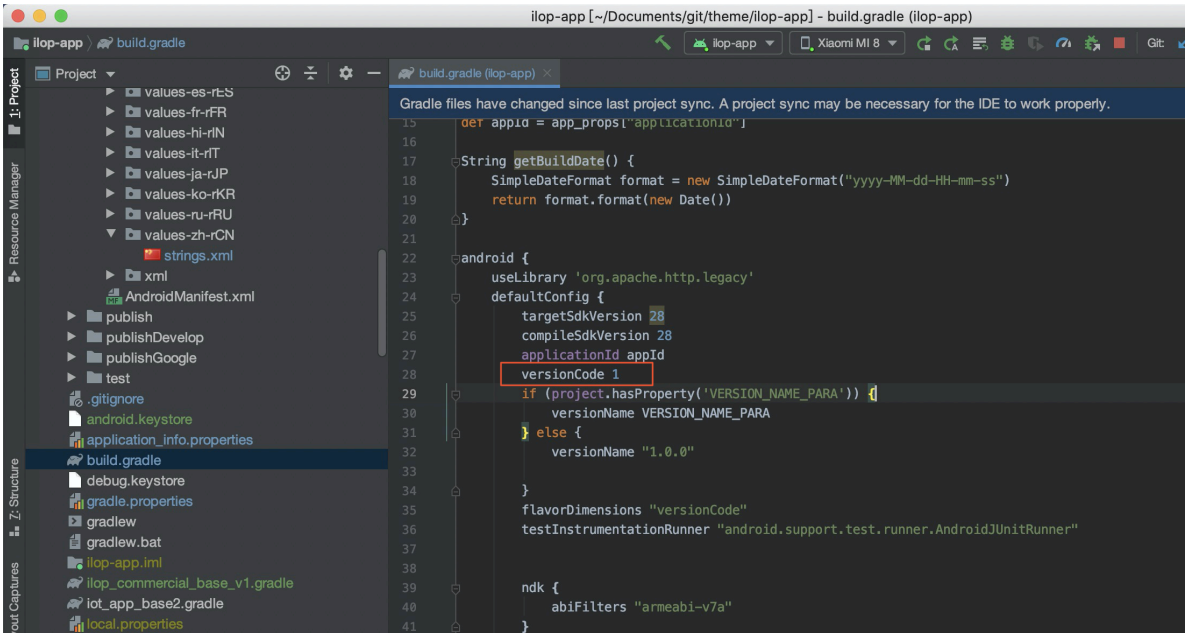
前提条件

- 已完成App模板的源码下载。请参见#unique_67。
- 已安装Android App的开发工具，如Android Studio。

修改App版本号（必选）

每次发布App前，您需要修改App版本号。请根据以下步骤来操作。

1. 使用Android Studio工具打开App工程，并进入project-source/build.gradle文件。
2. 修改**versionCode**的参数值。



修改App的界面文案（可选）

Android App中的所有中文文案及对应的多语言文案都支持修改。如果你希望修改App中的某个文案，请根据以下步骤来操作。

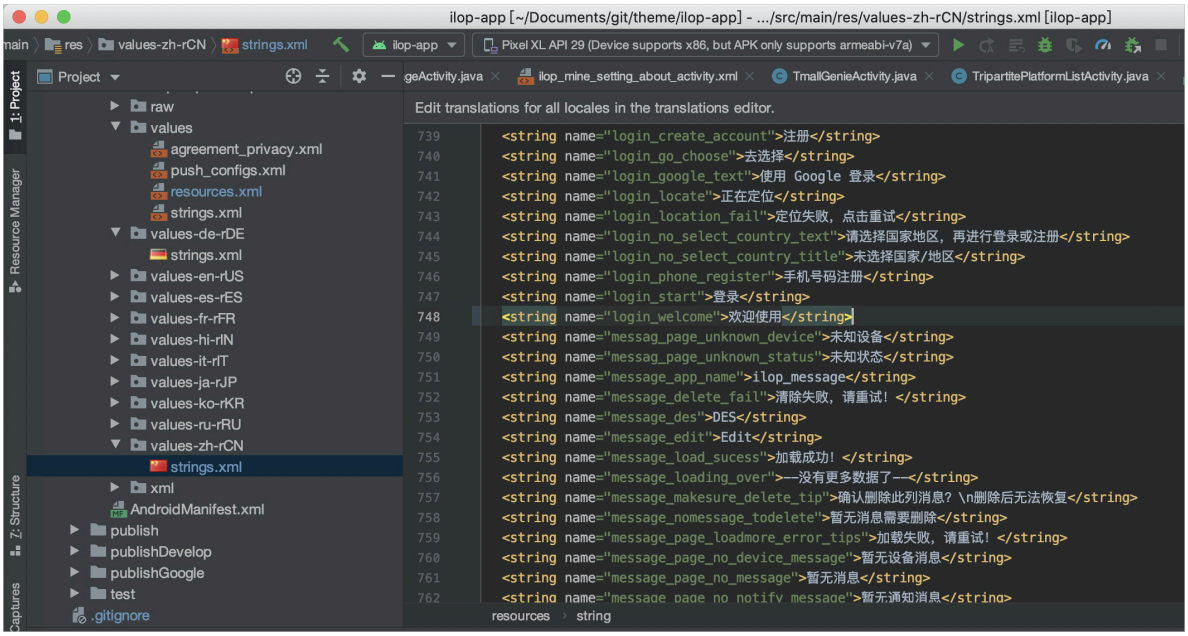
1. 使用Android Studio进入工程的多语言配置文件中。

   不同语言的配置文件不同，各语言的路径如下。

语言	路径
中文	project-source/main/res/values-zh-rCN/strings
英文	project-source/main/res/values-en-rUS/strings
德文	project-source/main/res/values-de-rDE/strings
西班牙文	project-source/main/res/values-es-rES/strings
法文	project-source/main/res/values-fr-rFR/strings
印地文	project-source/main/values-hi-rIN/strings
意大利文	project-source/main/res/values-it-rIT/strings
日文	project-source/main/res/values-ja-rJP/strings
韩文	project-source/main/res/values-ko-rKR/strings
俄文	project-source/main/res/values-ru-rRU/strings

2. 修改App的文案。

App使用key=value的方式来定义文案，请您根据需要修改value的值。



说明：

当您完成App的中文界面文案修改时，建议您同步修改对应的多语言文案。同一个文案在不同语言的配置文件里的**key**相同。

例如：中文为"login_welcome"="欢迎使用"，对应的英文为"login_welcome"="Welcome"。此时中文App的界面文案显示为“欢迎使用”，切换到英文App时界面文案显示为“Welcome”。

修改App中的图片（可选）

如果您需要修改App中的某张图片，请您进入模板源码中的图片所在路径，并替换图片即可。

替换App图片时，需要注意以下内容。

- 替换App中图片时，须确保图片名称保持一致。
- Android工程中的图片，按照分辨率要求，图片所在的目录可能存在以下任一目录下。其中drawable表示默认分辨率，带后缀-xhdpi表示较高分辨率，且x个数表示高级程度。
  - project-source/src/main/res/drawable
  - project-source/src/main/res/drawable-xhdpi
  - project-source/src/main/res/drawable-xxhdpi
  - project-source/src/main/res/drawable-xxxhdpi

修改App中的颜色（可选）

如果您需要修改App中某个文字或背景的颜色，请进入模板源码中的颜色所在路径，并修改相应参数值即可。

App中所有自定义颜色所在的文件为project-source/src/main/res/values/resources.xml，参数请查看源码中的注释。

4.2.2 iOS App通用修改项

该文档用来汇总所有iOS App模板自定义时的通用修改项，如修改App版本号、修改App界面文案、修改App图片、颜色等。

前提条件

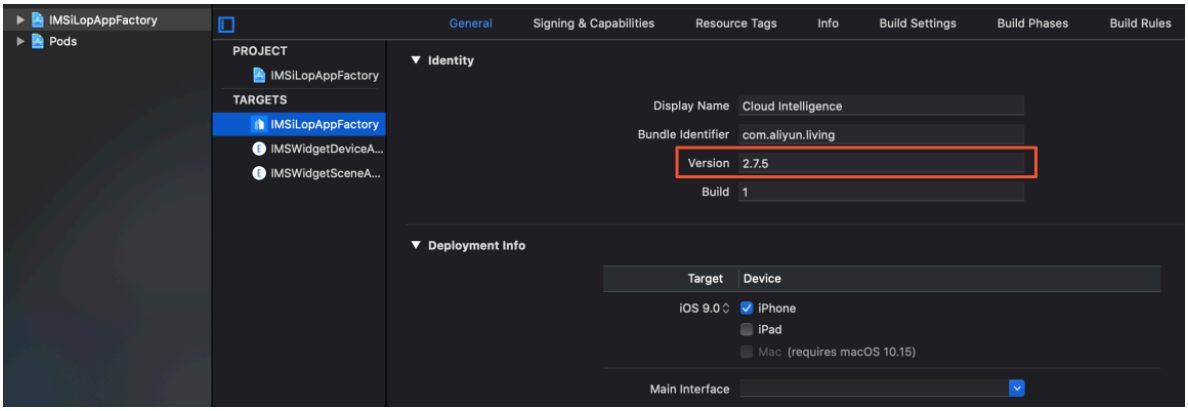
- 已完成App模板的源码下载。请参见#unique_67。
- 已安装iOS App的开发工具xcode。

修改App的版本号（必选）

向苹果应用商店提交App前，您需要修改App的版本号，版本号直接向终端用户展示。如果同一个版本号的App您需要向苹果应用商店提交多次，则还需要修改**Build**值。关于iOS App上架的更多限制说明请查阅苹果公司的官网文档。

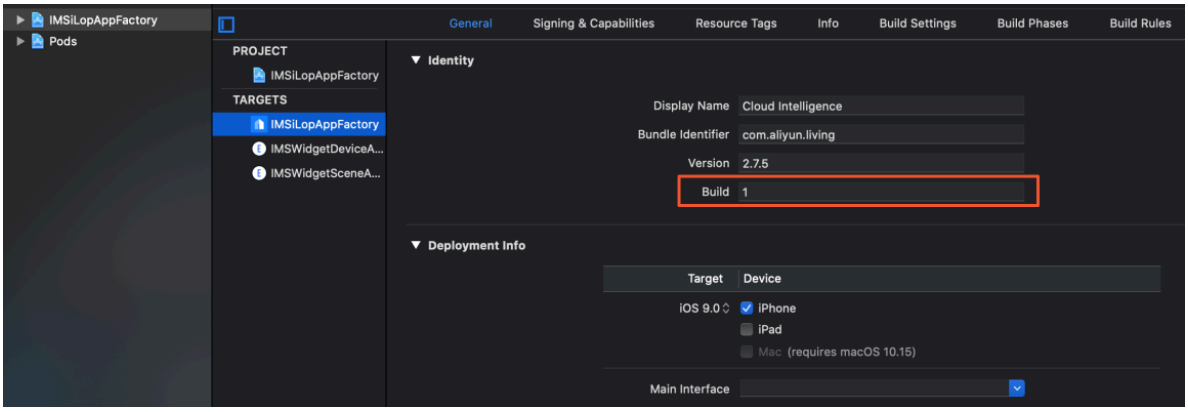
请您根据以下步骤来修改App版本号和Build值。

1. 使用xcode工具打开工程，并单击**General** > **IMSiLopAppFactory** > **IMSiLopAppFactory**。
2. 修改App的版本号。



3. 修改**Build**参数的值。

   根据苹果公司规定，同一版本号的App，每次提交上架应用商店时，**Build**值需保持递增。



4. 修改小组件的版本号。

   您可以使用**xcrun agvtool**命令运行xcode自带的小工具，自动修改所有小组件的版本号。详细操作请您自行查阅苹果公司的官网文档。

   建议小组件的版本号与App的版本号保持一致。

修改App的界面文案（可选）

iOS App中的所有中文文案及对应的多语言文案都支持修改。如果你希望修改App中的某个文案，请根据以下步骤来操作。

1. 使用xocde进入工程的多语言配置文件中。

   不同语言的配置文件不同，各语言的路径如下。

语言	路径
中文	./IMSiLopAppFactory/zh.lproj/Localizable.strings
英文	./IMSiLopAppFactory/en.lproj/Localizable.strings
德文	./IMSiLopAppFactory/de.lproj/Localizable.strings
西班牙文	./IMSiLopAppFactory/es.lproj/Localizable.strings
法文	./IMSiLopAppFactory/fr.lproj/Localizable.strings
印地文	./IMSiLopAppFactory/hi.lproj/Localizable.strings
意大利文	./IMSiLopAppFactory/it.lproj/Localizable.strings
日文	./IMSiLopAppFactory/ja.lproj/Localizable.strings
韩文	./IMSiLopAppFactory/ko.lproj/Localizable.strings
俄文	./IMSiLopAppFactory/ru.lproj/Localizable.strings

2. 修改App的文案。

App使用key=value的方式来定义文案，请您根据需要修改value的值。

说明：

当您完成App的中文界面文案修改时，建议您同步修改对应的多语言文案。同一个文案在不同语言的配置文件里的**key**相同。

例如：中文为"login_welcome"="欢迎使用"，对应的英文为"login_welcome"="Welcome"。此时中文App的界面文案显示为“欢迎使用”，切换到英文App时界面文案显示为“Welcome”。

修改App中的图片（可选）

如果您需要修改App中的某张图片，请在xcode工具中进入模板源码中的图片所在路径，并替换图片即可。

修改App图片时，需要注意以下内容。

- 本文档中提供的图片修改目录由父目录与子目录两部分组成。父目录可以修改，子目录不可以修改。
- 子目录下一般存在多张图片，您都需要修改。
- 修改App图片的建议您使用xcode工具操作，并确保图片的分辨率与原有图片的保持一致。

App中所有的图片所在父目录如下所示。

- App图标和启动图所在目录：./IMSiLopAppFactory/Assets.xcassets/
- 其余自定义图片所在目录：./IMSiLopAppFactory/Assets.xcassets/custom/

修改App中的颜色（可选）

如果您需要修改App中某个文字或背景的颜色，请在xcode工具中进入模板源码中的颜色所在路径，并修改相应参数值即可。

App中所有自定义颜色所在的文件为./IMSiLopAppFactory/ModuleName_Config.h，修改的参数请查看源码中的注释。

4.3 云智能App2.X系列模板

4.3.1 概述

云智能2.X系列模板是基于云智能App开放的自有App模板，仅需简单配置后，即可搭建自己的“云智能App”。

开通服务

您需要先开通智能设备App模板服务，才能使用云智能2.X系列模板。具体操作请参见#unique_129。

模板功能介绍

生活物联网平台提供了一套完整的智能设备App模板，包括配网、家房间分组、设备控制、场景自动化、个人中心等。开通云智能2.X系列模板后，以下功能您可以直接使用，无需额外开发。

App页面	提供的服务
配网	提供扫码配网、本地发现、手动单击产品配网（该App关联的产品，都会在待配网列表中展示）。
家、房间、分组	提供家庭管理、房间管理、分组管理等。
设备控制	在控制台产品的**人机交互**页面中，配置的设备面板，在App中可直接使用。
场景自动化	在手动触发（场景）和自动触发（自动化）中，支持以时间点、时间段、设备状态等作为条件触发；以设备动作、触发场景、发送通知作为动作执行。
我的	提供个人中心和App设置相关，如昵称、头像、消息中心、设备共享、小组件、意见反馈、智能日志、固件升级、多语言切换、温度单位、当前版本、服务协议、语音平台对接。

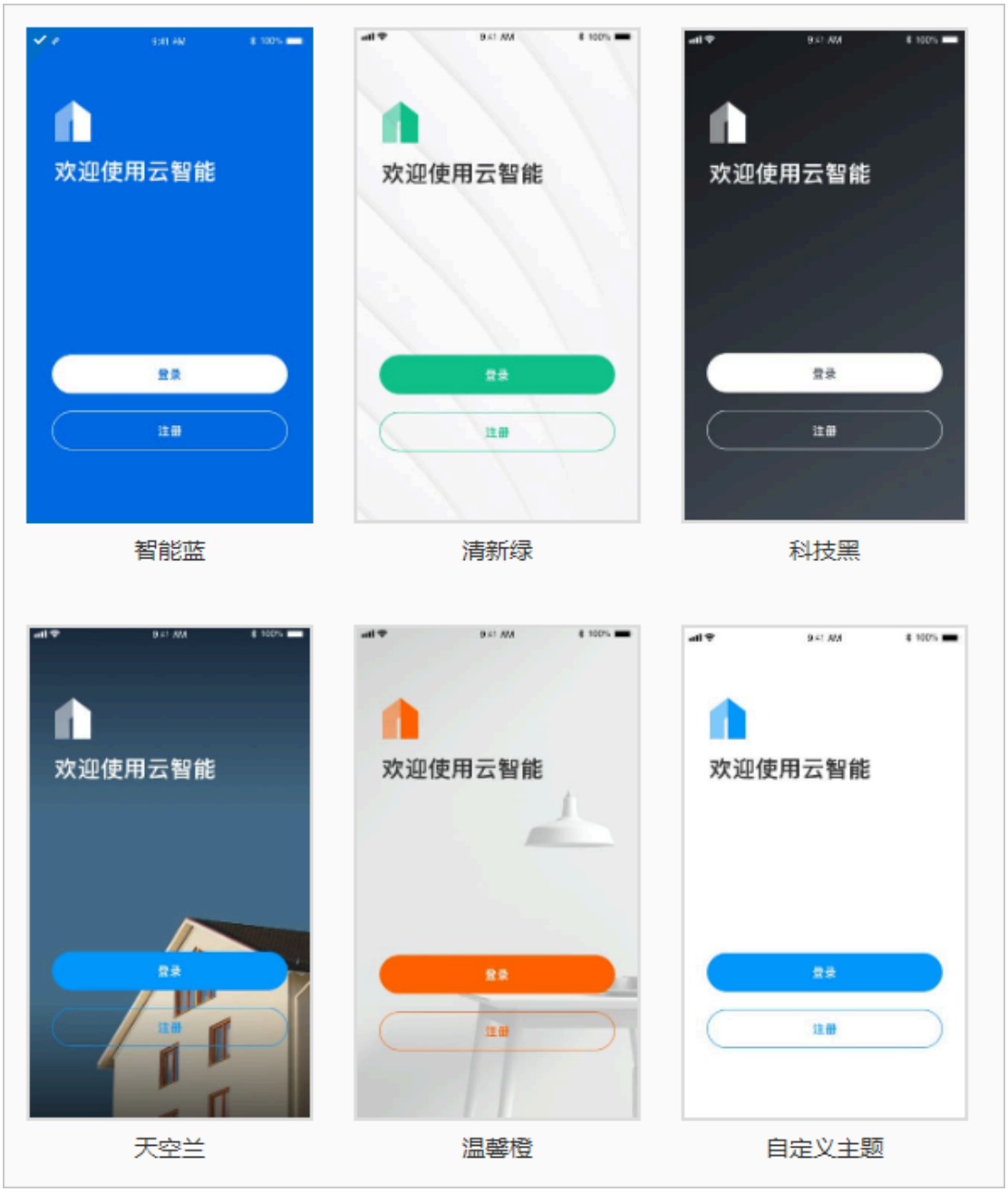
个性化配置介绍

云智能2.X系列模板支持您进行个性化配置，您可以在控制台上配置，也可以在源码中修改。

- 控制台可配置内容
  - App主题色

    主题色影响App中的主要按钮、文字、开关、菜单等UI样式。本模板提供5套固定主题色，和1套自定义主题色。

    ■ 固定主题色

    ■ 自定义主题色

    自定义主题色以白色为底色，可以从色盘中选取任意颜色，作为App的主题色。

- App启动页

  您只需上传一张尺寸超过1242*2688px的图片，即可自动裁剪成可适配各种手机型号的图片。

- 源码可修改内容

  云智能2.X系列模板的源码供您修改的内容主要有主题色、UI的配置项（如全部的文字、部分的颜色、部分的图片等）、天猫精灵的对接源码。除此之外，您还能在源码中修改更多细节的元素，如下表所示。

App界面	可修改内容
注册登录页	- 修改图片：背景图、顶部图标 - 修改颜色：背景色、登录按钮颜色、注册按钮颜色 - 修改文案：修改界面的文案
App底部菜单	- 修改图片：选中和非选中状态的图标 - 修改颜色：背景色、文字颜色 - 修改文案：修改菜单的文案
首页	- 修改图片：背景图、空列表背景图 - 修改颜色：主题色 - 修改文案：修改界面的文案 - 可隐藏：首页页面
智能菜单页	- 修改图片：场景背景图、自动化背景图、场景空列表背景图、自动化空列表背景图 - 修改颜色：主题色 - 修改文案：修改界面的文案 - 可隐藏：智能页面

App界面	可修改内容
我的菜单页	- 修改图片：各列表的图标、默认头像图 - 修改颜色：主题色 - 修改文案：修改界面的文案 - 可隐藏：我的页面、部分功能列表、部分功能列表的子功能模块 除了**个人信息编辑**、**消息中心**、**设备共享**，三个功能模块不支持隐藏外，其余的功能模块都可分别隐藏。其中**设置**、**关于**功能模块中的子功能模块也支持分别隐藏。

4.3.2 iOS 2.X系列模板

除了App模板通用的修改外，云智能App 2.X系列模板还支持的自定义内容，如更换主题色、配置UI菜单、对接天猫精灵等。

前提条件

- 已完成App模板的源码下载。请参见#unique_67。
- 已安装iOS App的开发工具xcode。

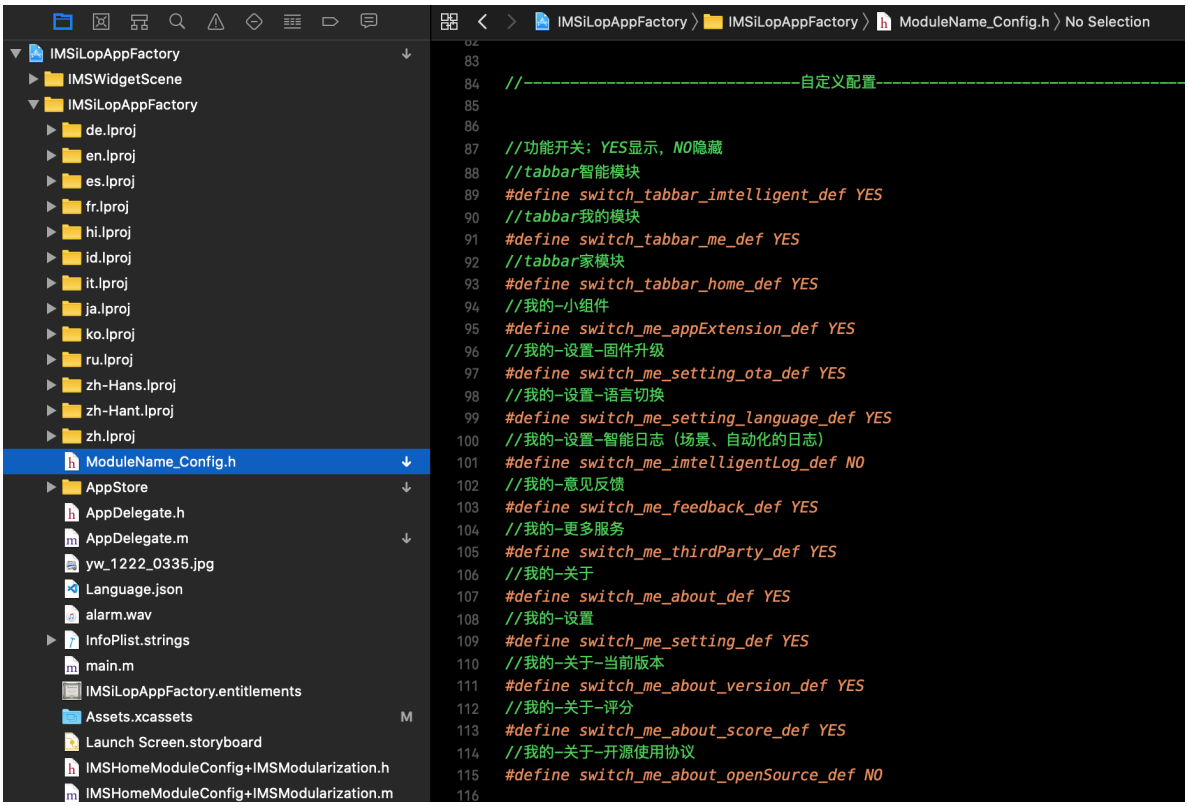
修改说明

自定义iOS系列的App模板的操作主要分为修改颜色、修改图片、隐藏默认菜单等。

请您使用xcode打开App模板的源码，并根据以下方法自定义App。

- 隐藏菜单

  如果您需要隐藏App中的某菜单，请进入待隐藏模块对应的目录后，修改参数对应的值。如下图所示。

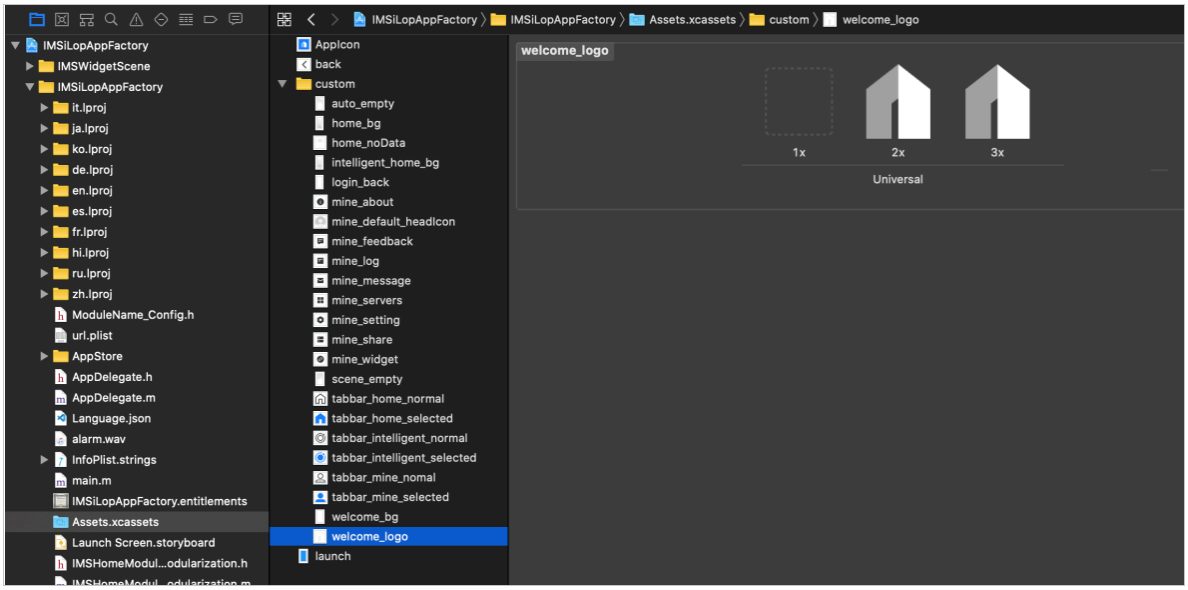


说明：

参数值为**YES**时表示显示菜单；为**NO**时表示隐藏菜单。

- 修改图片

  如果您需要修改App中的某张图片，请进入待修改模块图片所对应的目录下，替换目录下的图片，如下图所示。

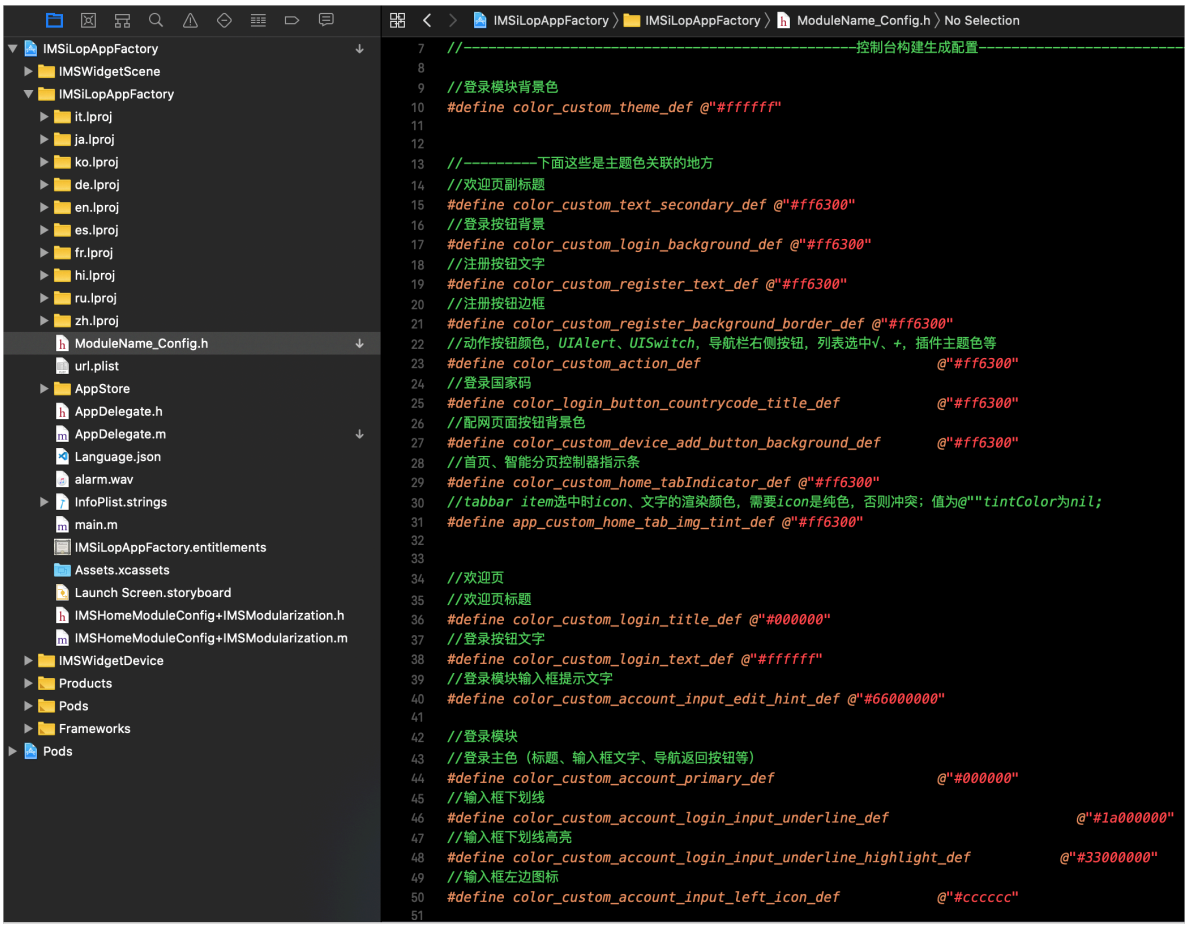


修改App图片时，您需要注意以下内容。

- 本文档中提供的图片修改目录由父目录与子目录两部分组成。父目录可以修改，子目录不可以修改。
- 子目录下一般存在多张图片，您都需要修改。
- 修改App图片的建议您使用xcode工具操作，并确保图片的分辨率与原有图片的保持一致。

- 修改颜色

  如果您需要修改App中的颜色，请进入待修改模块颜色所对应的目录下，修改参数对应的值，如下图所示。



> **说明：**
> 如果您需要修改App的主题色，请在./IMSiLopAppFactory/ModuleName_Config.h文件中，修改**color_custom_action_def**参数的值。

修改注册登录页

云智能2.X系列App模板的默认注册登录页如下图所示。

注册登录页面支持您修改以下内容。

- 修改图片

目的	操作父目录	替换子目录下图片（分辨率一致）
修改背景图	./IMSiLopAppFactory/Assets.xcassets/custom/	welcome_bg.imageset
修改顶部图标		welcome_logo.imageset

- 修改颜色

目的	操作目录（或文件）	修改参数的值
修改背景色	./IMSiLopAppFactory/ModuleName_Config.h	color_custom_login_background_def
修改登录按钮的背景颜色		color_custom_login_background_def
修改登录按钮文字的颜色		color_custom_login_text_def
修改注册按钮边框的颜色		color_custom_register_background_border_def
修改注册按钮文字颜色		color_custom_register_text_def
标题颜色（欢迎使用...）		color_custom_login_title_def

说明：

注册按钮的背景颜色为透明色，因此只能修改**注册**按钮的边框颜色。

修改App底部菜单

云智能2.X系列App模板的底部有三个菜单，分别为：家、智能、我的。如下图所示。

App底部菜单支持您修改以下内容。

- 隐藏菜单

目的	操作目录（或文件）	修改参数的值
隐藏**家**菜单	./IMSiLopAppFactory/ ModuleName_Config.h	switch_tabbar_home_def
隐藏**智能**菜单		switch_tabbar_imtell igent_def
隐藏**我的**菜单		switch_tabbar_me_def

- 修改菜单的图标

图标	是否选中状态	操作父目录	替换子目录下图片（分辨率一致）
家	是	./IMSiLopAppFactory/ Assets.xcassets/custom/	tabbar_home_nomal. imageset
	否		tabbar_home_selected. imageset
智能	是		tabbar_intelligent_normal. imageset
	否		tabbar_intelligent_selected .imageset
我的	是		tabbar_mine_nomal. imageset
	否		tabbar_mine_selected. imageset

- 修改颜色

目的	操作目录（或文件）	修改参数的值
修改选中菜单时的颜色（主菜单图片、文字的颜色）	./IMSiLopAppFactory/ ModuleName_Config.h	app_custom_home_tab_img_tint_def **说明：** 该参数的值，会对图片的颜色进行强制渲染。例如配置为红色后，图片的颜色会变成红色。

修改家菜单页

云智能2.X系列App模板的默认家菜单页（即首页）的界面如下图所示。

菜单页（即首页）支持修改以下内容。

- 修改图片

目的	操作父目录	替换子目录下图片（分辨率一致）
修改背景图	./IMSiLopAppFactory/Assets.xcassets/custom/	home_bg.imageset
修改空列表背景图		home_noData.imageset

- 修改颜色

  请您参照配置示例在./IMSiLopAppFactory/ModuleName_Config.h文件中修改**家**菜单页中的颜色，详细参数请参见源码中的注释。修改方法请参见修改说明。

修改智能菜单页

云智能2.X系列App模板的默认**智能**页如下图所示。

智能菜单页支持修改的内容包括：主题色、背景图、空列表背景图（列表为空时的背景图，即上图中加号位置的背景图）等。

- 修改图片

目的	操作父目录	替换子目录下图片（分辨率一致）
修改背景图	./IMSiLopAppFactory/Assets.xcassets/custom/	intelligent_home_bg.imageset
修改自动化的空列表背景图		auto_empty.imageset
修改场景的空列表背景图		scene_empty.imageset

- 修改颜色

  请您参照配置示例在./IMSiLopAppFactory/ModuleName_Config.h文件中修改**智能**菜单页中的颜色，详细参数请参见源码中的注释。修改方法请参见修改说明。

修改我的菜单页

云智能2.X系列App模板的默认**我的**菜单页如下图所示。

16:52
100
查看并编辑个人资料
小组件
意见反馈
消息中心
智能日志
设备共享
更多服务
设置
关于

我的菜单页中支持修改以下内容。

- 修改功能列表
  - 隐藏功能列表

目的	操作目录（或文件）	修改参数的值
隐藏小组件	./IMSiLopAppFactory/ModuleName_Config.h	switch_me_appExtension_def
隐藏智能日志		switch_me_imtelligentLog_def
隐藏意见反馈		switch_me_feedback_def
隐藏设置		switch_me_setting_def
隐藏关于		switch_me_about_def
隐藏更多服务		switch_me_thirdParty_def

说明：

我的菜单页中，个人信息编辑、消息中心、设备共享，三个功能模块不支持隐藏。

  - 修改功能列表的图标

    您可以修改**我的**菜单页中各项功能列表的图标。

目的	操作父目录	替换子目录下图片（分辨率一致）
修改默认头像	./IMSiLopAppFactory/Assets.xcassets/custom/	mine_default_headIcon.imageset
修改小组件的图标		mine_widget.imageset
修改意见反馈的图标		mine_feedback.imageset
修改消息中心的图标		mine_message.imageset
修改智能日志的图标		mine_log.imageset
修改设备共享的图标		mine_share.imageset
修改更多服务的图标		mine_servers.imageset
修改设置的图标		mine_setting.imageset
修改关于的图标		mine_about.imageset

- 自定义**设置**页面

  您除了可以隐藏整个**设置**功能，还可以分别隐藏**设置**的子功能。

目的	操作目录（或文件）	修改参数的值
隐藏固件升级	./IMSiLopAppFactory/ModuleName_Config.h	switch_me_setting_language_def
隐藏语言		switch_me_setting_ota_def

- 自定义**关于**页面

您除了可以隐藏整个**关于**功能，还可以分别隐藏**关于**的子功能。

目的	操作目录（或文件）	修改参数的值
隐藏当前版本	./IMSiLopAppFactory/ ModuleName_Config.h	switch_me_about_version_def
隐藏去评分		switch_me_about_score_def
隐藏隐私权政策		url_mine_about_privacy_policy_def
隐藏服务协议		url_mine_about_service_agreement_def
隐藏开源软件使用协议		switch_me_about_openSource_def

说明：

隐藏隐私权政策和**隐藏服务协议**的参数值默认为空，即App页面不显示。如果您需要配置相关内容，请参见本文档下方**修改登录的服务协议**的内容来配置。

- 自定义**更多服务**页面

  更多服务中主要展示对接的三方平台，仅对接天猫精灵的源码开源。如果您想对接更多三方平台，可自行对接。对接天猫精灵的详细对接操作请参见#unique_87。

说明:

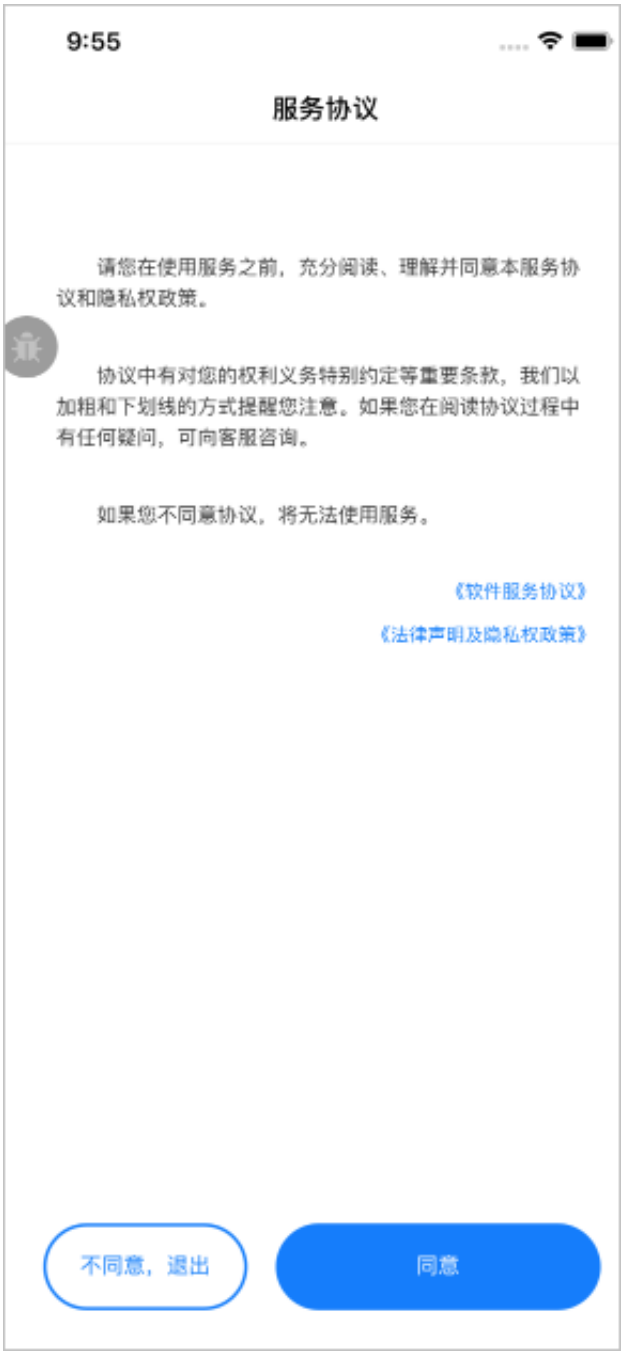
对接三方平台时，为保证页面可以正常跳转，请勿修改以下代码。其中IMSSmartSpeakerAccess表示**我的** > **更多服务**的模块名称，IMSThirdPartyListViewController表示模块的入口类。

```
+ (void)load {
    NSURL *url = [NSURL URLWithString:@"ilopapp://thirdParty/list"];
    [[IMSRouterService sharedService] registerURL:url withHandler:[self class]];
}

+ (UIViewController *)controllerWithParams:(NSDictionary *)params {
    IMSThirdPartyListViewController *thirdVC = [[IMSThirdPartyListViewController alloc]
init];
    thirdVC.hidesBottomBarWhenPushed = YES;

    thirdVC.imageArray = params[@"imageArray"];
    return thirdVC;
```

```
}
```

修改登录的服务协议

云智能App的**我的** > **关于**页面中默认有三个法律协议，分别是：隐私权政策、服务协议、开源软件使用协议。其中隐私权政策和服务协议在登录App时会自动弹出，如下图所示。



云智能2.X系列App模板的**法律声明及隐私权政策**和**软件服务协议**默认为隐藏，即登录App时不会弹出上图**服务协议**的页面。如果您希望登录App后弹出该页面，请根据以下步骤配置服务协议，服务协议为URL格式。

1. 进入工程的./IMSiLopAppFactory/ModuleName_Config.h文件。

2. 修改**url_app_use_service_agreement_def**参数的值。

   - 参数值为空时，隐藏**服务协议**页面。
   - 参数值不为空时，账号登录App后，弹出**服务协议**页面。

   示例代码如下：

   ```
   #define url_app_use_service_agreement_def @"http://xxxxxx"
    //登录后弹出服务协议页面，页面的URL为https://xxxxxx
   ```

3. （可选）当参数值不为空时，处理页面跳转事件。

   服务协议页面包含“软件服务协议”、“法律声明及隐私权政策”、“同意”、“不同意”四个事件。请根据以下示例修改事件的URL。

   - 软件服务协议

     URL中需包含**agree-software**，例如http://x.xxx.com/agree-software.html?visibility=undefined&navbarHide=true。

   - 法律声明及隐私权政策

     URL中需包含**legalNotice**，例如http:/x.xxxx.xxx/legalNotice.html?visibility=undefined&navbarHide=true。

   - 同意

     URL中需包含**action=agree**，例如http://xxx.xxx.com/privacy?action=agree。

   - 不同意

     URL中需包含**action=disagree**，例如http://xxx.xxx.com/privacy?action=disagree。

4.3.3 Android 2.X系列模板

除了App模板通用的修改外，云智能App 2.X系列模板还支持的自定义内容，如更换主题色、配置UI菜单、对接天猫精灵等。

前提条件

- 已完成App模板的源码下载。请参见#unique_67。
- 已安装Android App的开发工具，如Android Studio。

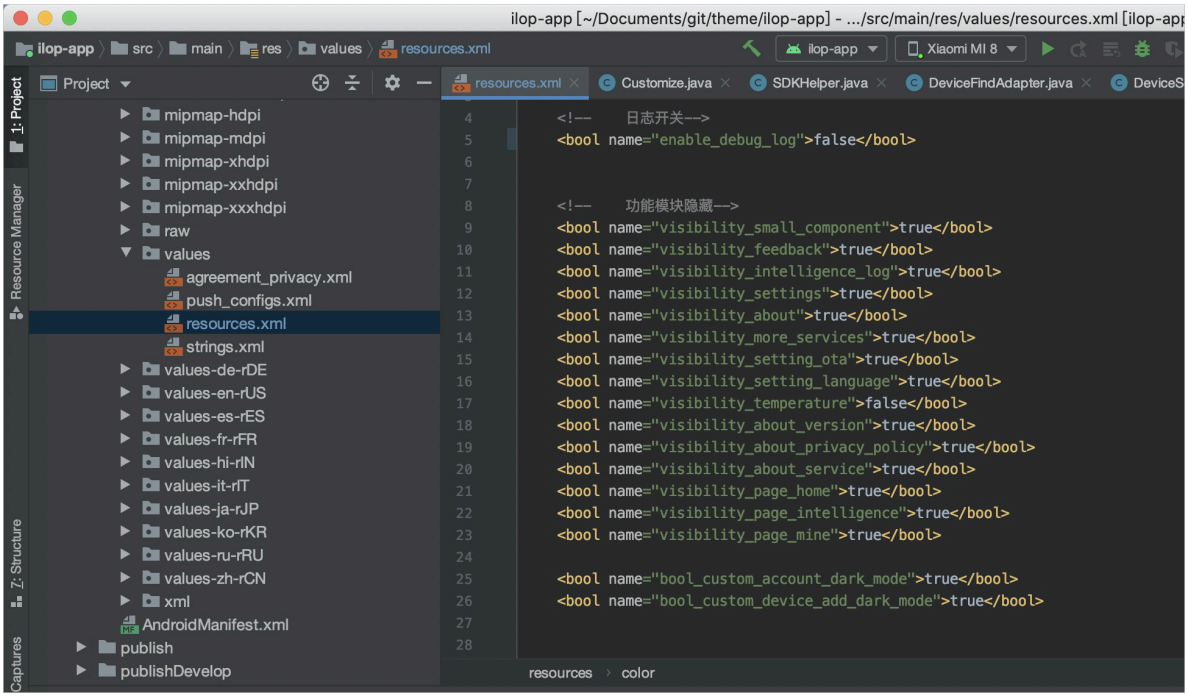
修改说明

自定义Android系列的App模板的操作主要分为修改颜色、修改图片、隐藏默认菜单等。

请您使用Android Studio打开App模板的源码，您根据以下方法自定义App。

- 隐藏菜单

  如果您需要隐藏App中的某菜单，请进入待隐藏模块对应的目录下，修改参数对应的值，如下图所示。

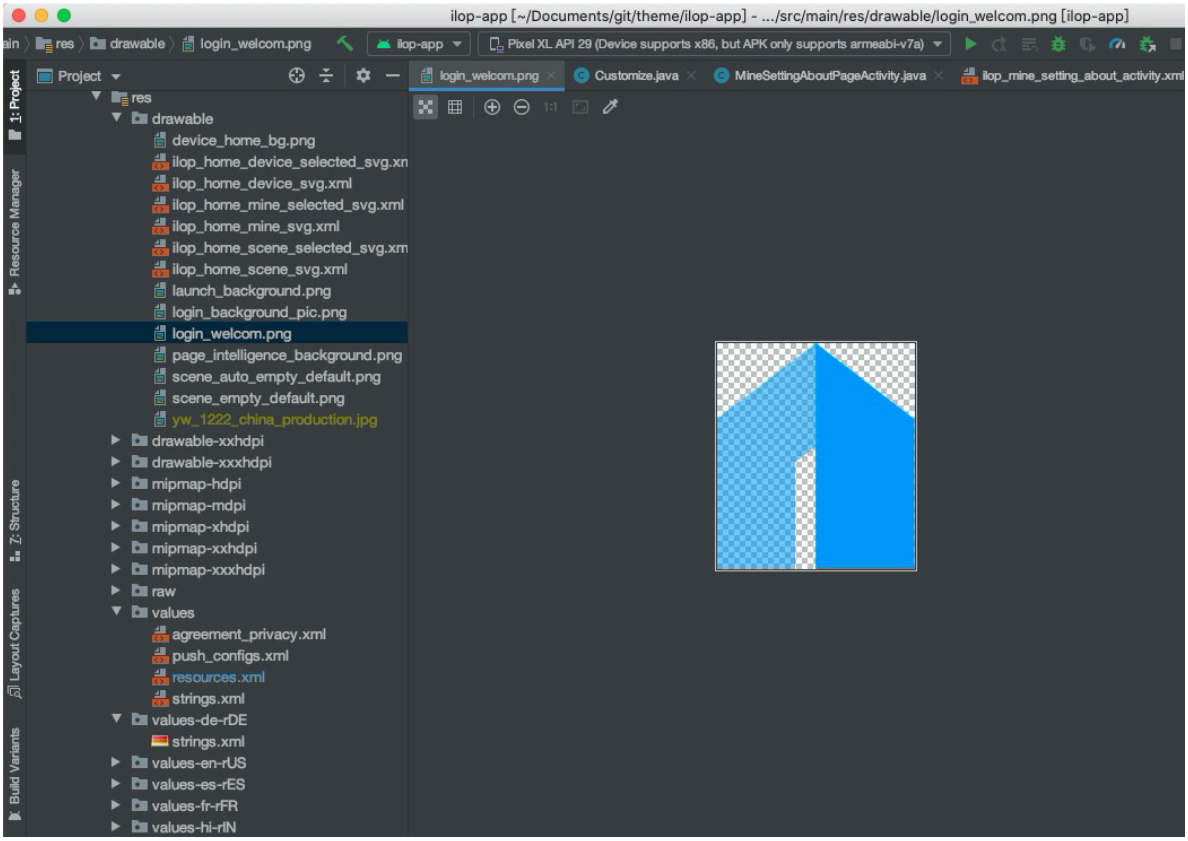


说明：

参数值为**true**时表示显示菜单；为**false**时表示隐藏菜单。

- 修改图片

  如果您需要修改App中的某张图片，请进入待修改模块图片所对应的目录下，替换目录下的图片。

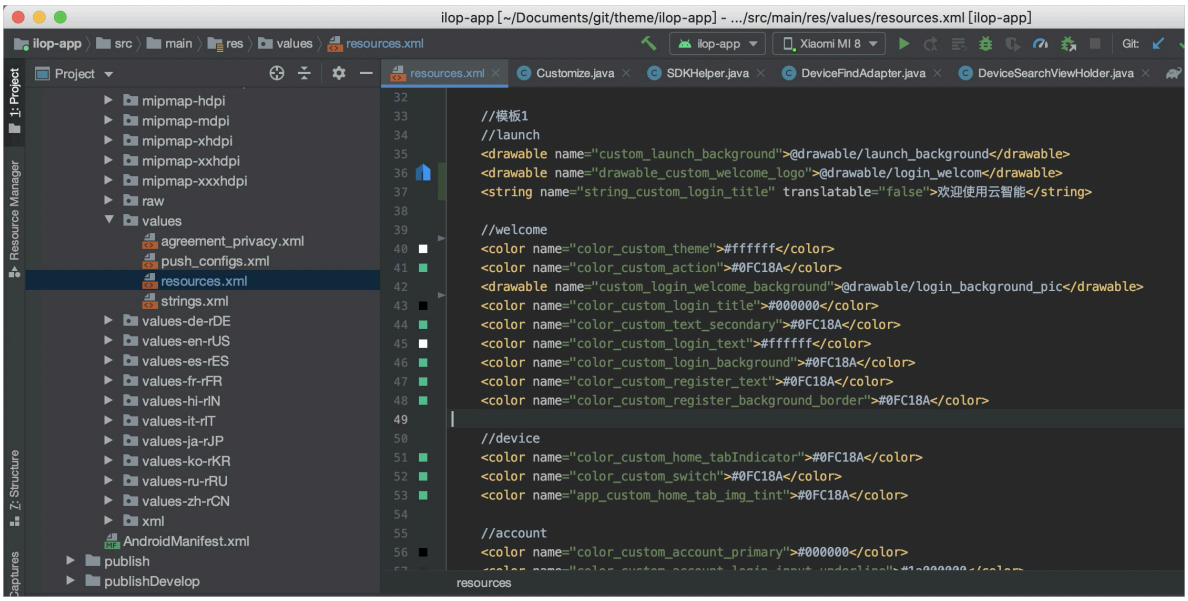


修改App图片时，需要注意以下内容。

- 替换App中图片时，须确保图片名称保持一致。
- Android工程中的图片，按照分辨率要求，图片所在的目录可能存在以下任一目录下。其中drawable表示默认分辨率，带后缀-xhdpi表示较高分辨率，且x个数表示高级程度。
  - project-source/src/main/res/drawable
  - project-source/src/main/res/drawable-xhdpi
  - project-source/src/main/res/drawable-xxhdpi
  - project-source/src/main/res/drawable-xxxhdpi

- 修改颜色

  如果您需要修改App中的颜色，请进入待修改模块颜色所对应的目录下，修改参数对应的值，如下图所示。



> **说明：**
> 如果您需要修改App的主题色，请在project-source/src/main/res/values/resources.xml文件中，修改**color_custom_action**参数的值。

修改注册登录页

云智能2.X系列App模板的默认注册登录页如下图所示。

注册登录页面支持您修改以下内容。

- 修改图片

目的	操作目录（或文件）	替换图片名称
修改背景图	project-source/src/main/res/drawable	login_background_pic.png
修改顶部图标		login_welcom.png

- 修改颜色

目的	操作目录（或文件）	修改参数的值
修改背景色	project-source/src/main/res/values/resources.xml	color_custom_theme
修改登录按钮的背景颜色		color_custom_login_background
修改登录按钮文字的颜色		color_custom_login_text
修改注册按钮边框的颜色		color_custom_register_background_border
修改注册按钮文字颜色		color_custom_register_text
标题颜色（欢迎使用...）		color_custom_login_title

说明：

注册按钮的背景颜色为透明色，因此只能修改**注册**按钮的边框颜色。

修改App底部菜单

云智能2.X系列App模板的底部有三个菜单，分别为：家、智能、我的。如下图所示。

App底部菜单支持您修改以下内容。

- 隐藏菜单

目的	操作目录（或文件）	修改参数的值
隐藏**家**菜单	project-source/src/main/res/values/resources.xml	visibility_page_home
隐藏**智能**菜单		visibility_page_intelligence
隐藏**我的**菜单		visibility_page_mine

- 修改菜单的图标

图标	是否为选中状态	操作目录（或文件）	替换图片（名称须一致）
家	是	project-source/src/main/res/drawable-xxhdpi	ilop_home_device_selected.png
	否		ilop_home_device.png
智能	是		ilop_home_scene_selected.png
	否		ilop_home_scene.png
我的	是		ilop_home_mine_selected.png
	否		ilop_home_mine.png

- 修改颜色

目的	操作目录（或文件）	修改参数的值
修改选中菜单时的颜色（包括菜单图片、文字的颜色）	project-source/src/main/res/values/resources.xml	app_custom_home_tab_img_tint **说明：** 该参数的值，会对图片的颜色进行强制渲染。例如配置为红色后，图片的颜色会变成红色。

修改家菜单页

云智能2.X系列App模板的默认家菜单页（即首页）的界面如下图所示。

菜单页（即首页）支持修改以下内容。

- 修改图片

图标	操作目录（或文件）	替换图片（名称须一致）
修改背景图	project-source/src/main/res/drawable-xxxhdpi	page_device_background.png
修改空列表背景图		device_home_bg.png

- 修改主颜色

  请您参照配置示例在project-source/src/main/res/values/resources.xml文件中修改**家**菜单页中的颜色，详细参数请参见源码中的注释。修改方法请参见修改说明。

修改智能菜单页

云智能2.X系列App模板的默认**智能**页如下图所示。

智能菜单页支持修改的内容包括：主题色、背景图、空列表背景图（列表为空时的背景图，即上图中加号位置的背景图）等。

- 修改图片

图标	操作目录（或文件）	替换图片（名称须一致）
修改背景图	project-source/src/main/res/drawable-xxxhdpi	scene_empty_default.png
修改自动化的空列表背景图		scene_auto_empty_default.png
修改场景的空列表背景图		scene_empty_default.png

- 修改颜色

  请您参照配置示例在project-source/src/main/res/values/resources.xml文件中修改**智能**菜单页中的颜色，详细参数请参见源码中的注释。修改方法请参见修改说明。

修改我的菜单页

云智能2.X系列App模板的默认**我的**菜单页如下图所示。

16:52
100
查看并编辑个人资料
小组件
意见反馈
消息中心
智能日志
设备共享
更多服务
设置
关于

我的菜单页中支持修改以下内容。

- 自定义功能列表
  - 隐藏功能列表

目的	操作目录（或文件）	修改参数的值
隐藏小组件	project-source/src/main/res/values/resources.xml	visibility_small_component
隐藏智能日志		visibility_intelligence_log
隐藏意见反馈		visibility_feedback
隐藏设置		visibility_settings
隐藏关于		visibility_about
隐藏更多服务		visibility_more_services

说明:

我的菜单页中，个人信息编辑、消息中心、设备共享，三个功能模块不支持隐藏。

  - 修改功能列表的图标

目的	操作目录（或文件）	替换图片（名称须一致）
修改默认头像	project-source/src/main/res/drawable-xxhdpi	ilop_mine_home_avatar_default.png
修改小组件的图标		ilop_mine_entry_widget.png
修改意见反馈的图标		ilop_mine_entry_feedback.png
修改消息中心的图标		ilop_mine_entry_message.png
修改智能日志的图标		ic_scene_logcat.png
修改设备共享的图标		ilop_mine_entry_device_share.png
修改更多服务的图标		ilop_mine_entry_more.png
修改设置的图标		ilop_mine_entry_settings.png
修改关于的图标		ilop_mine_entry_about.png

- 自定义**设置**页面

您除了可以隐藏整个**设置**功能，还可以分别隐藏**设置**的子功能。

目的	操作目录（或文件）	修改参数的值
隐藏固件升级	project-source/src/main/res/values/resources.xml	visibility_setting_ota
隐藏语言		visibility_setting_language

- 自定义**关于**页面

您除了可以隐藏整个**关于**功能，还可以分别隐藏**关于**的子功能。

目的	**操作目录（或文件）**	**修改参数的值**
隐藏当前版本	project-source/src/main/res/values/agreement_privacy.xml	visibility_about_version
隐藏隐私权政策		custom_mine_url_privacy_policy
隐藏服务协议		custom_mine_url_service_protocol

目的	操作目录（或文件）	修改参数的值
隐藏开源软件使用协议		visibility_about_privacy_policy

说明：

隐藏隐私权政策和**隐藏服务协议**的参数值默认为空，即App页面不显示。如果您需要配置相关内容，请参见本文档下方**修改登录的服务协议**的内容来配置。

- 自定义**更多服务**页面

  更多服务中展示了对接的三方平台，仅对接天猫精灵的源码开源。对接天猫精灵的详细对接操作请参见#unique_87。您也可以自行对接更多三方平台。

说明：

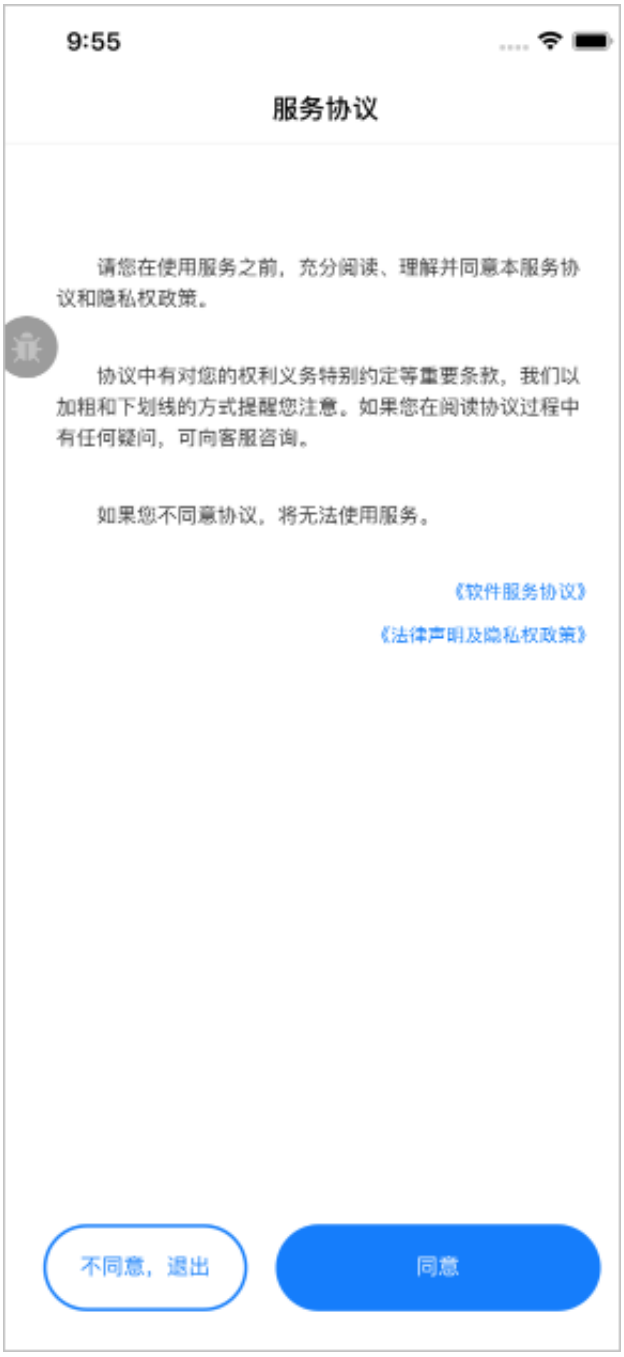
对接三方平台时，为保证页面可以正常跳转，请勿修改以下代码。其中/page/me/tp是**我的** > **更多服务**模块名称，TripartitePlatformListActivity是模块的入口类。

```
<activity
   android:name=".activity.TripartitePlatformListActivity"
   android:exported="false"
```

```
    android:screenOrientation="portrait">
    <intent-filter>
        <action android:name="com.aliyun.iot.aep.action.navigation" />

        <category android:name="android.intent.category.DEFAULT" />

        <data
            android:host="com.aliyun.iot.ilop"
            android:path="/page/me/tp"
            android:scheme="https" />
    </intent-filter>
</activity>
```

修改登录的服务协议

云智能App的**我的** > **关于**页面中默认有三个法律协议，分别是：隐私权政策、服务协议、开源软件使用协议。其中隐私权政策和服务协议在登录App时会自动弹出，如下图所示。



云智能2.X系列App模板的**法律声明及隐私权政策**和**软件服务协议**默认为隐藏，即登录App时不会弹出上图**服务协议**的页面。如果您希望登录App后弹出该页面，请根据以下步骤配置服务协议，服务协议为URL格式。

1. 进入工程的project-source/src/main/res/values/agreement_privacy.xml文件。

2. 配置**custom_privacy_policy_url_china_publish**参数的值。

   - 参数值为空时，隐藏**服务协议**页面。
   - 参数值不为空时，账号登录App后，弹出**服务协议**页面。

   示例如下：

   ```
   <string name="custom_privacy_policy_url_china_publish">https://xxxxxx</string>
    //登录后弹出服务协议页面，页面的URL为https://xxxxxx
   ```

3. （可选）当参数值不为空时，处理页面跳转事件。

   服务协议页面包含“软件服务协议”、“法律声明及隐私权政策”、“同意”、“不同意”四个事件。请根据以下示例修改事件的URL。

   - 软件服务协议

     URL中需包含**agree-software**，例如http://x.xxx.com/agree-software.html?visibility=undefined&navbarHide=true。

   - 法律声明及隐私权政策

     URL中需包含**legalNotice**，例如http:/x.xxxx.xxx/legalNotice.html?visibility=undefined&navbarHide=true。

   - 同意

     URL中需包含**action=agree**，例如http://xxx.xxx.com/privacy?action=agree。

   - 不同意

     URL中需包含**action=disagree**，例如http://xxx.xxx.com/privacy?action=disagree。

4.4 Demo App模板

4.4.1 iOS Demo App模板

Demo App主要介绍App的开发流程，以及App的业务逻辑。在您开发自己的App前，建议您按照本文档完成Demo App的操作。

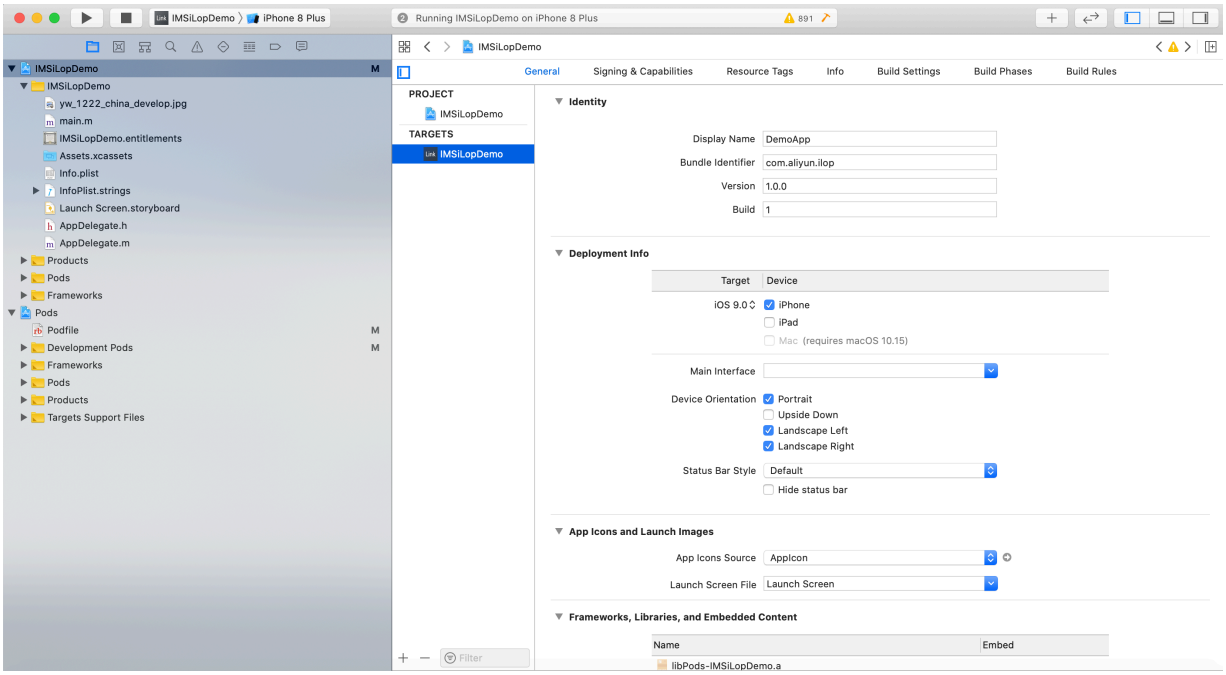
前提条件

- 已在生活物联网平台创建了对应的产品，并完成产品功能定义、配置App等相关操作。
- 已在生活物联网平台创建了一个自有品牌App，并完成Demo App的源码下载。详细操作请参见#unique_67。

使用方式

整个Demo App使用Cocoapods管理依赖库和模块，目前支持Cocoapods 1.2.0版本。

源码下载完成后，解压缩后进入源码目录，执行**pod update**。安装完成，通过Xcode打开IMSAppDemo.xcworkspace文件，然后编译运行。



DemoApp源码结构说明

整个Demo App结构上采用CocoaPods管理业务模块，主工程有安全图片，启动页面和AppDelegate实现，如下图所示。

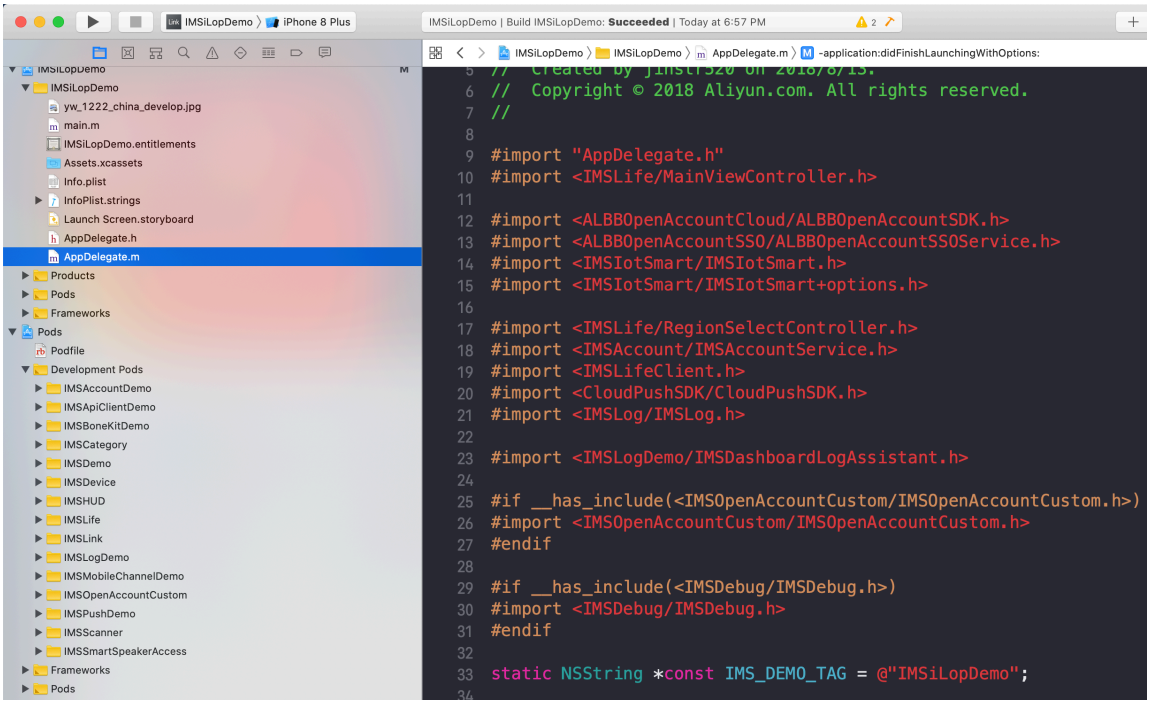
yw_1222_china_production.jpg为安全图片，可以从生活物联网平台下载。**Launch Screen.storyboard**为启动页面，**AppDelegate**主要为SDK的初始化，主页面的切换实现（登录/主页面），还有移动推送逻辑的处理。

SDK的初始化采用统一接口，详细介绍请参见SDK初始化。

Demo App整个UI以tab形式组织，主要由Pods中的IMSLife和IMSLink库实现。

- IMSLife是业务生活Demo对应的功能演示，包含整个App的主框架。支持设备发现、设备配网、设备绑定和设备状态的获取和控制功能。
- IMSLink则以基础SDK为主，主要为tab中间页- 调试页面的实现。

- 主要Pod库为SDK代码示例如下。
  - IMSAccountDemo：账号相关的Demo，提供账号登录、登出等。
  - IMSApiClientDemo：API通道的Demo，用于请求IoT接口业务。
  - IMSBoneKitDemo：BoneMobile容器使用的Demo，用于展示容器的能力和接口。
  - IMSLogDemo：演示日志。
  - IMSMobileChannelDemo：长连接通道的Demo，演示订阅/发送消息等功能。
  - IMSPushDemo：移动推送的Demo，展示通知信息的提交。



运行Demo App

1. 在控制台上创建自有App并下载demo App，详细请参见#unique_67。
2. 集成安全图片。详细请参见#unique_103。

> **说明：**
> 安全图片请勿重命名，否则会导致SDK初始化失败。请您下载后直接放到App代码工程下。

3. 在xcode里编译并运行Demo App。

   Demo App运行后，会显示国家/区域选择页面，请根据实际情况选择国家/地区后，进入登录App界面。

Carrier
5:25 PM
选择国家/地区
国家列表
定位失败，点击重试
A
阿尔巴尼亚
阿尔及利亚
阿富汗
阿根廷
阿联酋
阿鲁巴
阿曼
阿塞拜疆
爱尔兰
埃及
埃塞俄比亚
爱沙尼亚
安道尔共和国
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z #
确定

Demo App业务逻辑介绍

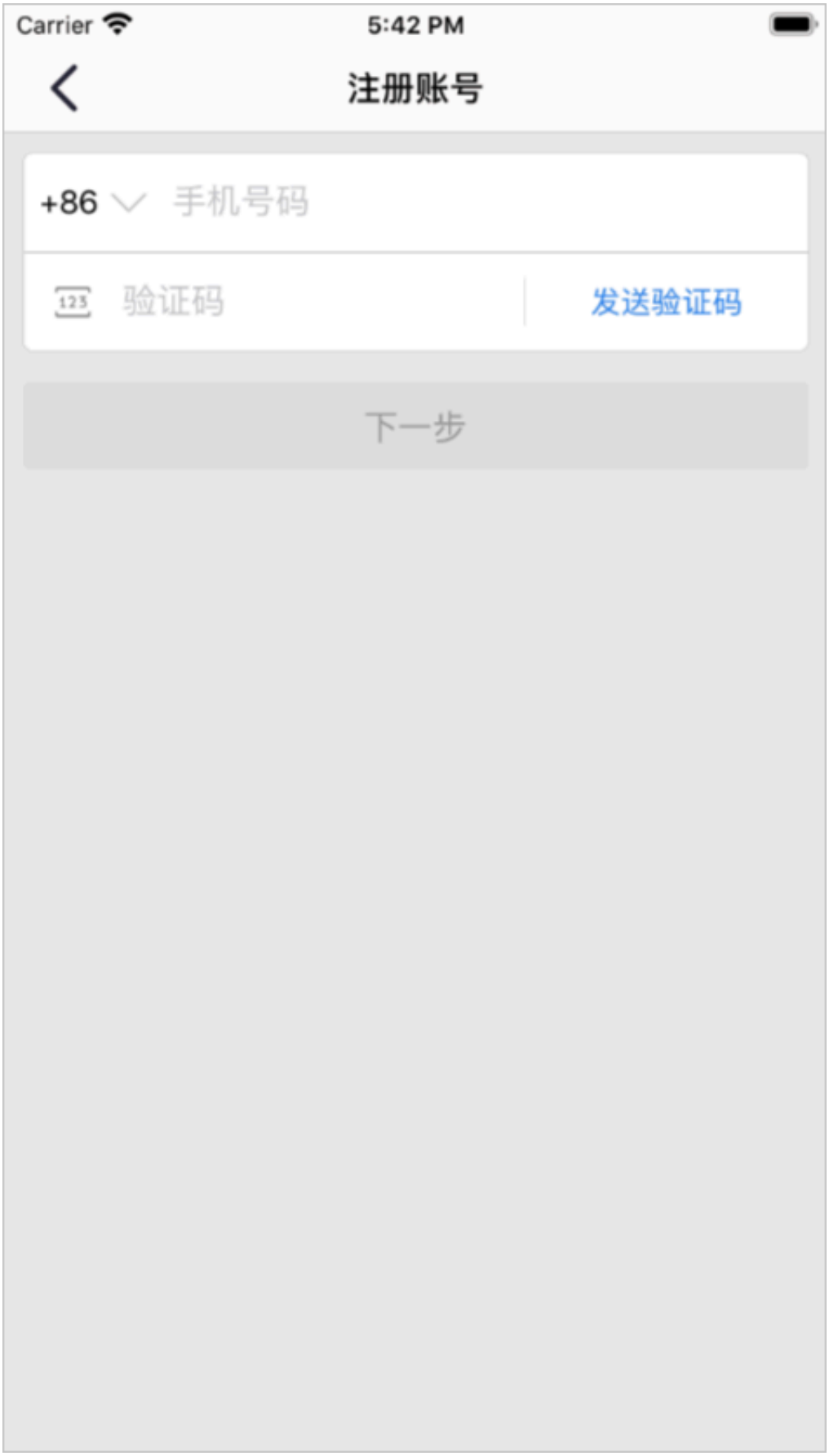
1. 登录自有App。

   a) 启动App，选择国家/区域。

b) 选择一个国家/区域后，进入登录界面。

c) 单击**免费注册**，进入注册页面。

d) 输入要注册的手机号码，并单击下方**发送短信校验码**。



e) 在短信校验码区域输入收到的校验码，单击**下一步**。

f) 设置登录密码，并完成登录。

2. 设置调试界面。

Carrier
6:58 PM
调试
Bone Mobile 容器 SDK
环境切换
本地调试
API 通道
调试
长连接通道
调试
账号和用户
界面展示
首页
调试
我的

IMSSDKEntryProtocol是各SDK模块展现入口协议，这些SDK的使用示例均遵循了该展现入口协议。例如：@interface SDKEntryApiClient () <IMSSDKEntryProtocol>，是API通道SDK的Demo示例。

Demo工程启动后getAllClasses方法会查找遵循了@protocol IMSSDKEntryProtocol协议的类，然后依据其getSDKDisplayConfig中的DisplayLevel按照顺序在首页展示，各个SDK模块示例源码统一放到LocalPods目录。为了便于示例源码学习，可以依据如下子模块源码文件名与展示名称对应进行查找。

文件名	界面展示名
SDKEntryBoneKit	Bone Mobile容器SDK
SDKEntryApiClient	API通道
SDKEntryMobileChannel	长连接通道
SDKEntryAccount	账号和用户
SDKEntryPush	移动应用推送

3. 控制默认的虚拟设备。

a) 进入主页面，如下图所示。

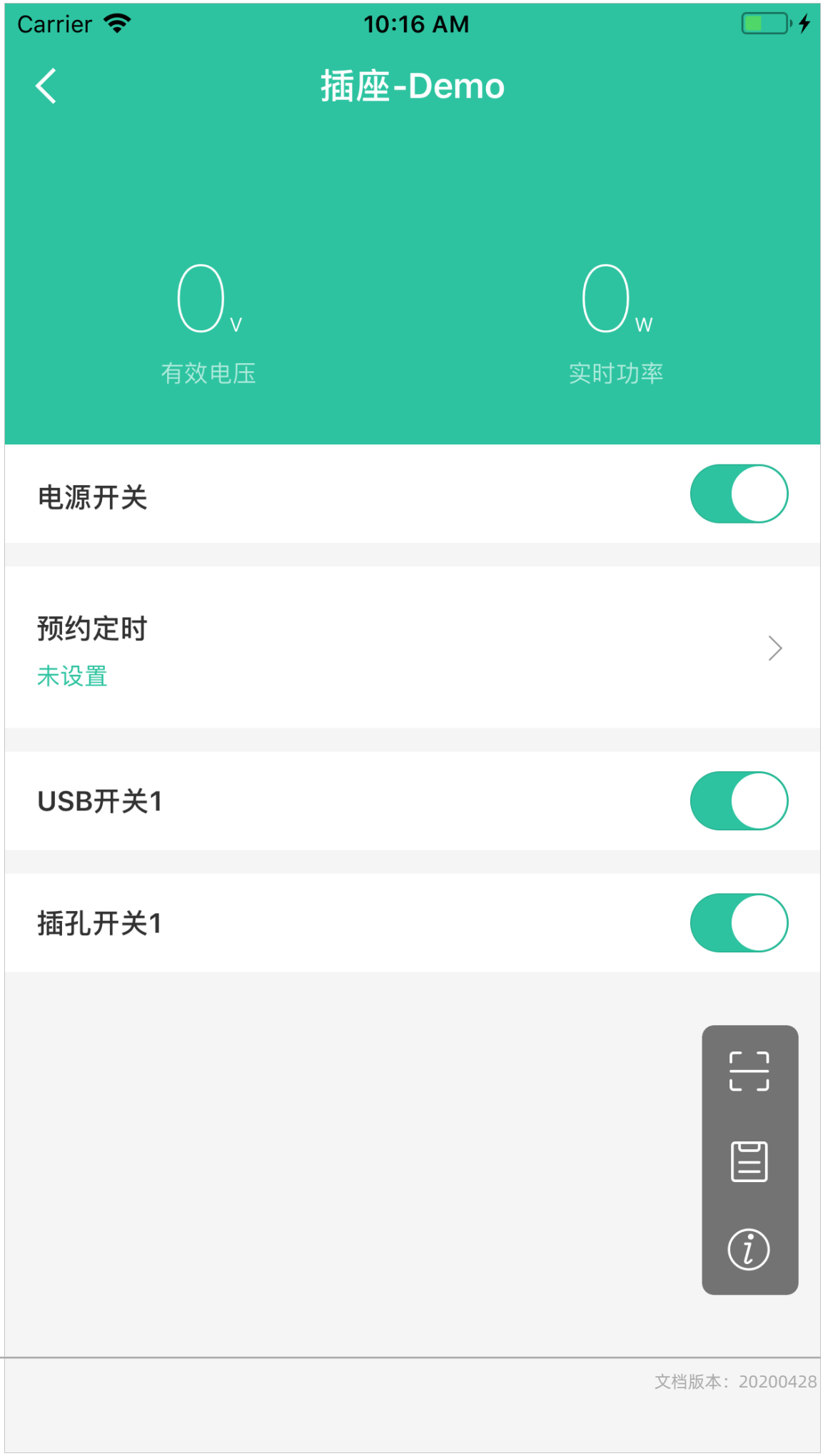
Carrier
10:53 AM
Link
我的设备
添加设备开启智能生活
虚拟设备
插座-Demo
虚拟
在线
风扇-Demo
虚拟
在线
灯-Demo
虚拟
在线
加湿器-Demo
虚拟
在线
首页
调试
我的

上面显示的4个虚拟设备，并不是默认创建的，需要自己更改代码，在LinkViewController.m里面，将自己项目内创建的产品productKey填入下面的supportVirtualList变量中，再编译运行，才可以创建和体验虚拟设备。

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor ims_backgroundColor];

    self.navigationItem.rightBarButtonItem = [[UIBarButtonItem alloc]initWithImage:[
UIImage bundleImageNamed:@"bar_add"] style:UIBarButtonItemStylePlain target:
self action:@selector(addButtonItemDidClicked)];

    // 用户在控制台创建设备后，可以把productKey列表设置进入，会自动生成虚拟设备
    // 例如：@[a1B6cFQldpm, a1AzoSi5TMc, a1nZ7Kq7AG1, a1XoFUJWkPr]
    self.supportVirtualList = @[];
```

```
}
```

b) 单击任意一个设备，进入该设备的面板页面。


Carrier
10:16 AM
插座-Demo
0 V
有效电压
0 W
实时功率
电源开关
预约定时
未设置
USB开关1
插孔开关1

4. 添加设备。

a) 单击首页右上角加号，并选择**添加设备**。

Carrier
10:34 AM
Link
我的设备
扫一扫
添加设备
添加设备开启智能生活
虚拟设备
插座-Demo
虚拟
在线
风扇-Demo
虚拟
在线
灯-Demo
虚拟
在线
加湿器-Demo
虚拟
在线
首页
调试
我的

b) 选择添加设备，进入添加设备页面。

Carrier
10:35 AM
添加设备
扫描本地设备...
本地发现设备
支持添加的设备
风扇-Demo
添加
插座-Demo
添加
灯-Demo
添加
8266灯
添加
加湿器-Demo
添加

通过产品列表添加需要使用官方项目对应的接入产品，这一步目前并没有真实设备可以使用；如果想体验这个流程，建议通过自主研发App的形式来完成。

添加设备支持两种形式。

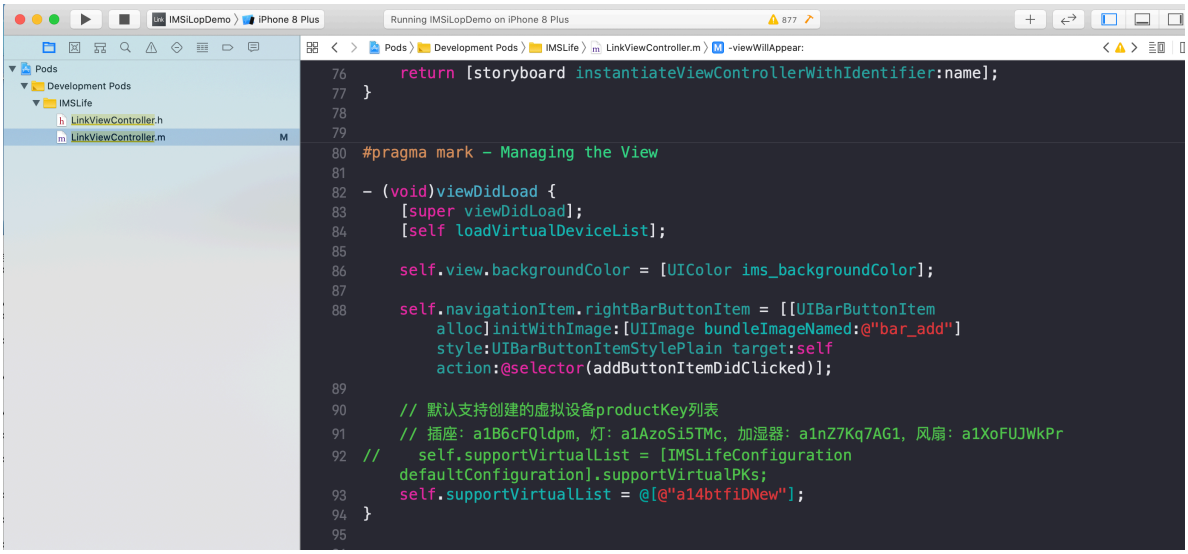
- 通过产品列表的形式进行配置，如上图支持添加的设备。
- 支持零配配网方案的设备，发现到本地设备则会显示出来，如上图本地发现设备。

虚拟创建产品的设备

在没有真实设备可以走通链路的情况下，可以使用虚拟设备进行体验和开发。下面介绍如何为您创建的产品虚拟设备。

1. 在LinkViewController.m中将产品对应的productKey，替换为自己项目的产品productKey。

```
self.supportVirtualList = @[@"axxxxeW",@"axxxxNe"];
```



说明：

每个产品可以申请1个虚拟设备。多个产品productKey之间用逗号间隔。例如：self.supportVirtualList = @[@"a14btFiDNeW", @"a1PCB67lxlP"];。

2. 在xcode里编译并运行App。

   如果工程的localMsg中提示“此路由不存在，请与小二确认”，则需要在**产品-人机交互**中配置设备的面板。

3. 重新登录App，即可使用虚拟设备体验您创建的产品的控制。

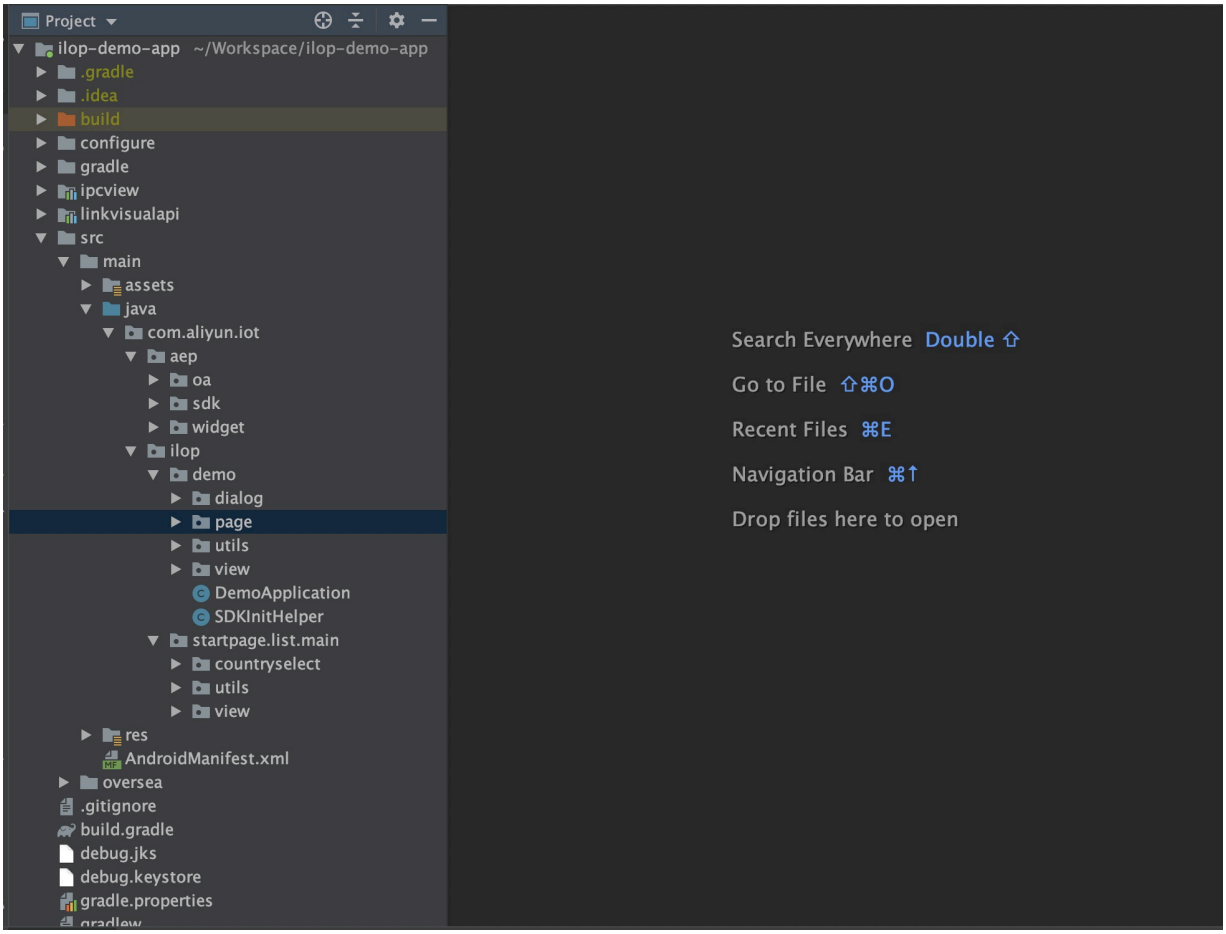
4.4.2 Android Demo App模板

Demo App主要介绍App的开发流程，以及App的业务逻辑。在您开发自己的App前，建议您按照本文档完成Demo App的操作。

前提条件

- 已在生活物联网平台创建了对应的产品，并完成产品功能定义、配置App等相关操作。
- 已在生活物联网平台创建了一个自有品牌App，并完成Demo App的源码下载。详细操作请参见#unique_67。

DemoApp 源码结构说明



App模块负责应用的业务逻辑，其中包括了Demo对应的功能演示：设备本地发现、设备配网、设备绑定和设备状态的获取和控制功能。java/com/aliyun/iot的详细子目录介绍如下。

- ilop/startpage： 演示了如何自定义国家选择页面
- ilop/demo/SDKInitHelper.java：演示了使用统一接口初始化各种SDK
- aep/oa：演示了如何自定义手机区域码选择页面
- aep/sdk/receiver：演示了如何利用广播接收器处理推送消息和通知

- ilop/demo/page/login3rd下
  - AuthCodeFragment.java：演示了如何用三方账号进行授权登录
  - OALoginActivity.java：演示了如何自定义登录页面

运行Demo App

介绍通过Android Studio导入Demo App，并构建APK的操作。

1. 打开Android Studio，选择**File** > **Open...**，导入Demo App源码。
2. 集成安全图片，操作请参见#unique_103。

   说明:

   安全图片请勿重命名，否则会导致SDK初始化失败。请您下载后直接放到App代码工程下。
3. （可选）Demo App中添加摄像头设备。

   a) 打开Demo App工程，在build.gradle中找到**dependencies**，添加Link Visual面板的引用。

   ```
   implementation project(':ipcview')
   ```

   b) 解注释Link Visual面板初始化代码。

   在src\main\java\com\aliyun\iot\ilop\demo\DemoApplication.java中，去掉以下代码前的注释符。

   ```
   IPCViewHelper.getInstance().init(this,"2.0.0");
   ```

   c) 解注释跳转Link Visual的代码。

   在src\main\java\com\aliyun\iot\ilop\demo\page\ilopmain\HomeTabFragment.java中，去掉以下代码前的注释符。

   ```
   Intent intent = new Intent(getActivity(), IPCameraActivity.class);
   intent.putExtra("iotId", iotId);
   intent.putExtra("appKey", EnvConfigure.getEnvArg(EnvConfigure.KEY_APPKEY));
   startActivity(intent);
   ```

   d) 替换产品的**ProductKey**。

   将src\main\java\com\aliyun\iot\ilop\demo\page\ilopmain\HomeTabFragment.java中，示例**ProductKey**替换为自己产品的**ProductKey**。
4. 在Demo App目录下，执行以下命令构建APK。

   ```
   ./gradlew clean build
   ```

   说明:

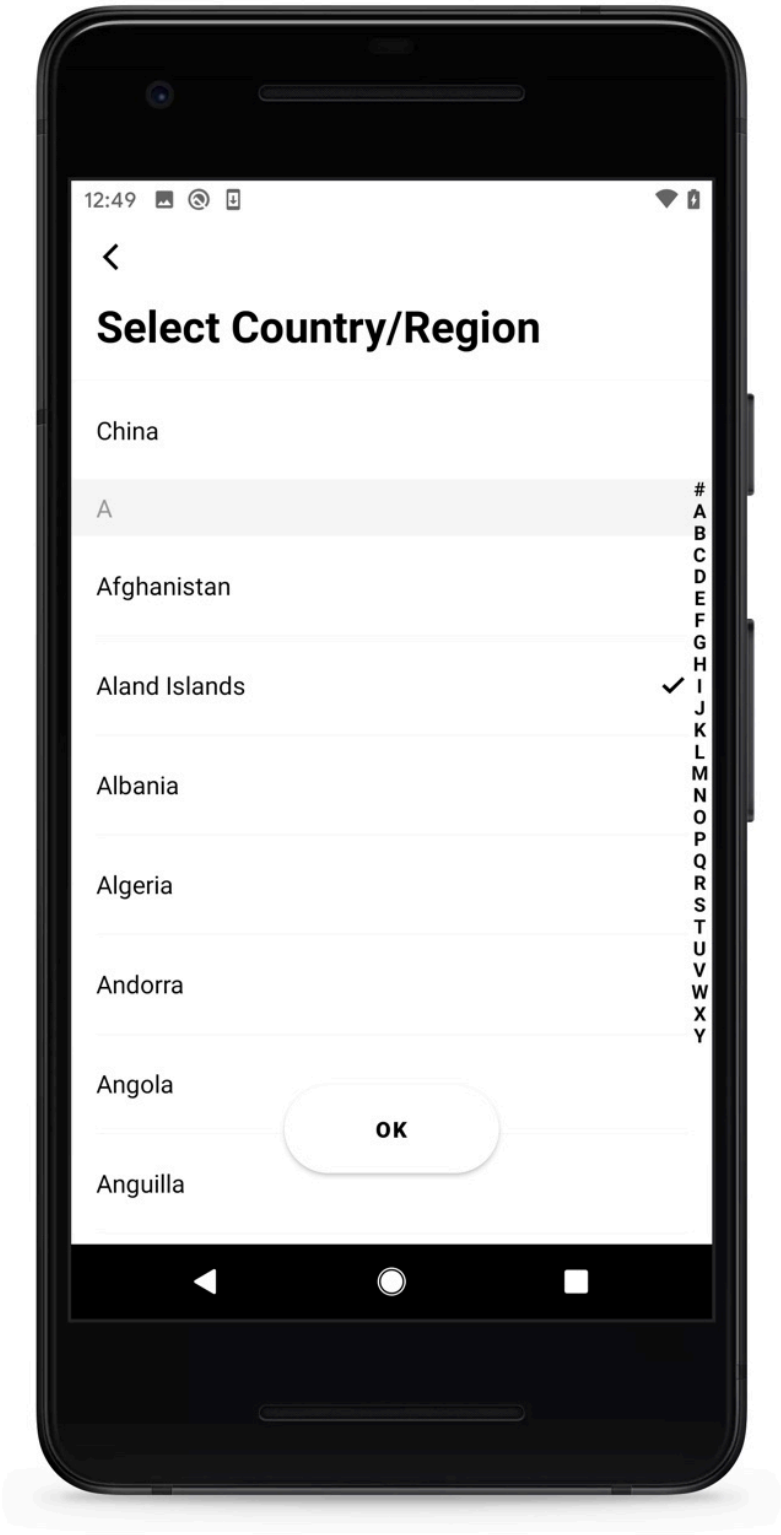
- 请配置JDK为1.8版本。
- 3.0.0版本的打包插件，需要 Android Studio 3.0以上的版本支持。
- 3.0.0版本的打包插件，目前需要访问Google的仓库，请注意网络环境。

Demo App业务逻辑介绍

默认Demo App配置为官方项目和默认支持的几类设备。

1. 选择国家/地区。

   登录之前您须先选择登录国家/地区。



说明:

如果您需要在国内/海外区域之间切换，切换后需重启App以生效。

2. 注册或登录账号。

App启动后会显示如下登录页界面。

a) 单击**免费注册**，弹出注册方式。

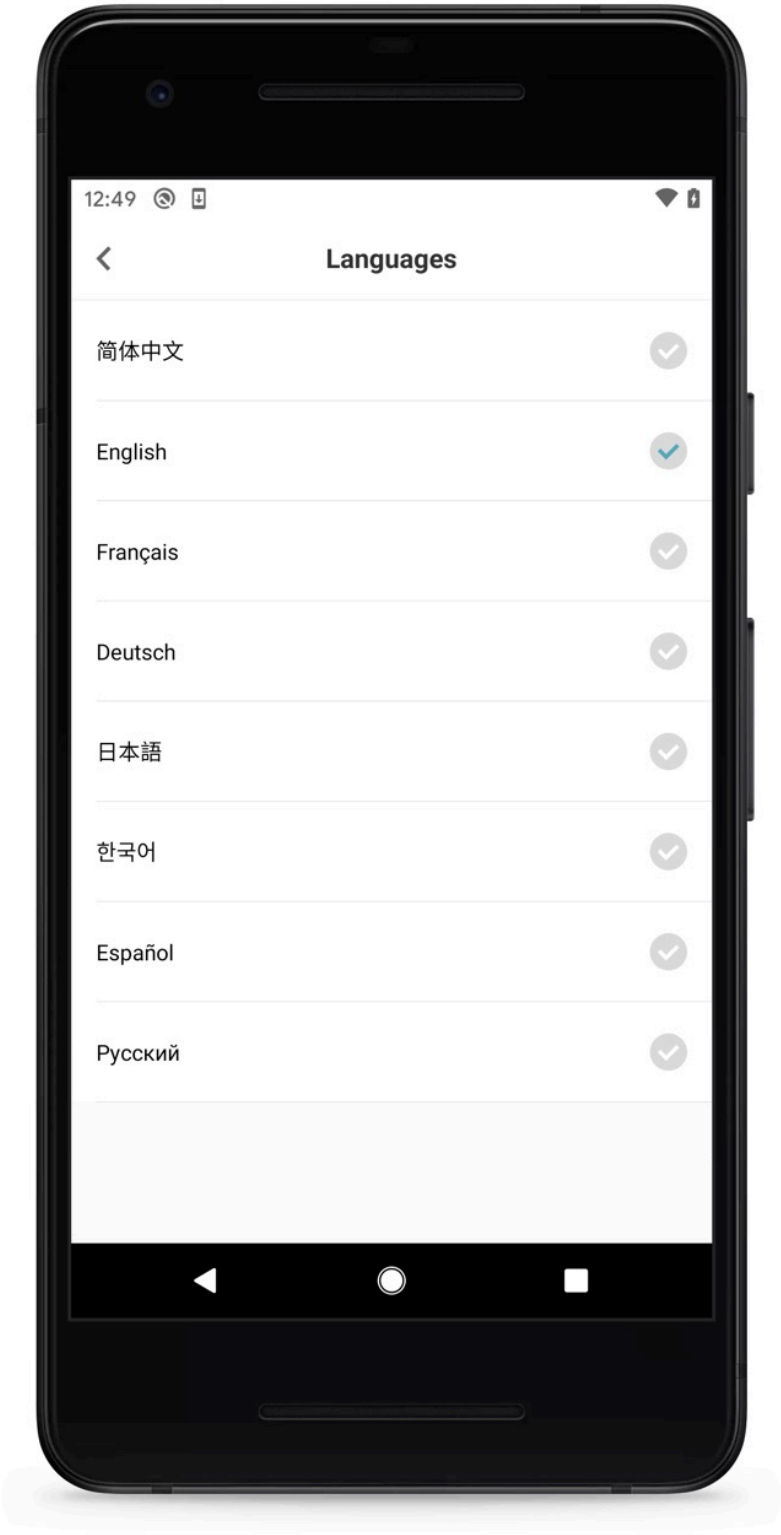
b) 输入要注册的手机号码或邮箱，并单击下方发送短信校验码。

c) 输入收到的校验码，单击**下一步**。

d) 设置密码完成登录。

3. 切换App的语言。

   在**我的** > **切换语言**中切换App的语言，界面如下图所示。



说明:

切换App语言后将跳转到登录界面，用户需要重新登录后才生效。

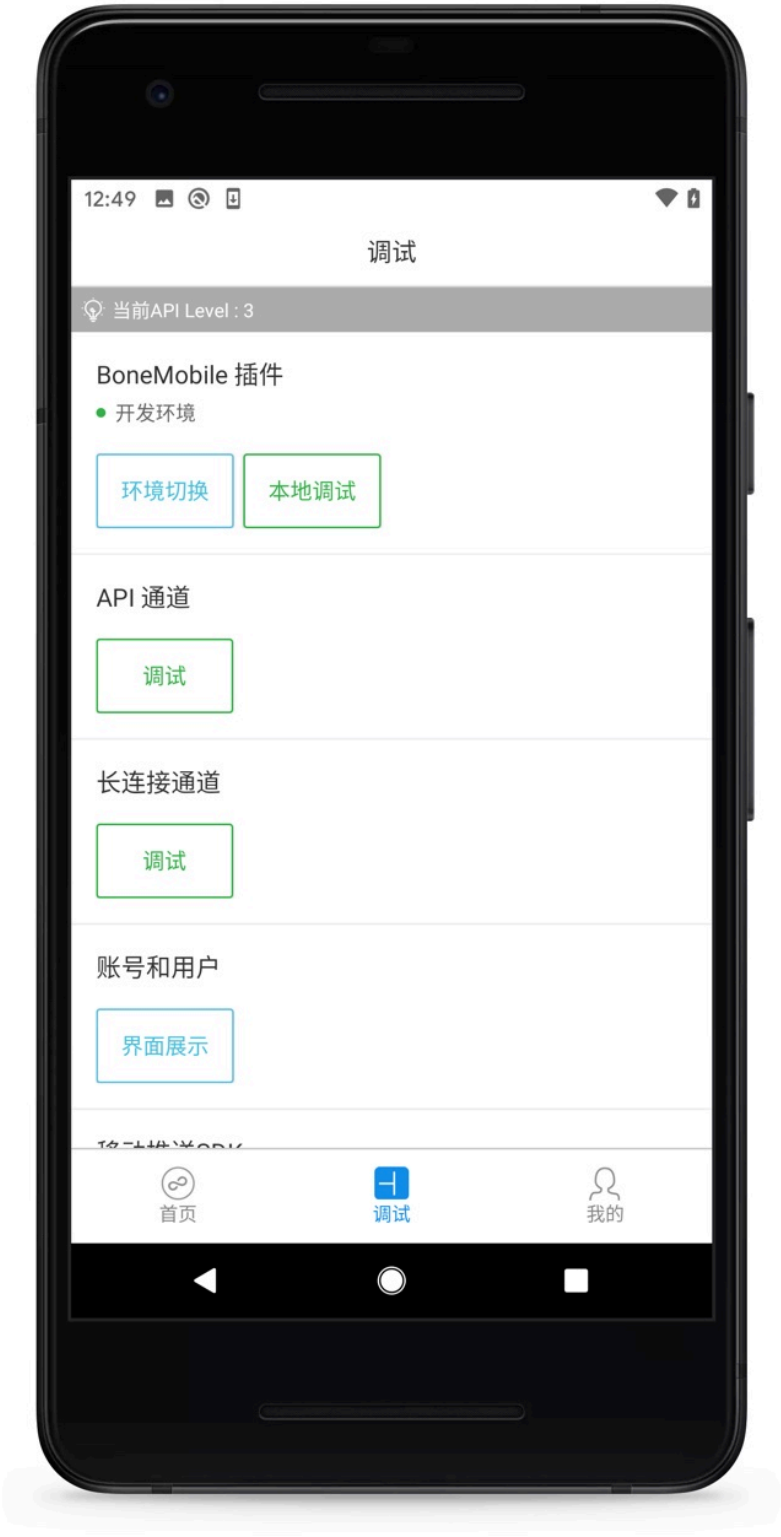
4. 设置调试界面。

   ebugTabFragment调试界面，调用com.aliyun.iot.ilop.demo.view.DebugLayout布局，提供以下功能。

   - API通道：API网管调试
   - 长链接通道：长链接通道调试
   - 账号和用户：登录、退出
   - 移动推送SDK：可以查看AppKey、ClientID、DeviceID等信息

   BoneMobile插件调试分为：

   - 开发环境、生产环境切换
   - 本地调试


12:49
调试
当前API Level : 3
BoneMobile 插件
开发环境
环境切换
本地调试
API 通道
调试
长连接通道
调试
账号和用户
界面展示
首页
调试
我的

5. 控制虚拟设备。

a) 进入主界面，如下图所示。

如果直接扫码下载demoAPP体验，登录后会默认创建5个虚拟设备。

如果是下载代码编译运行，上面显示的5个虚拟设备，并不是默认创建的，需要自己更改代码，在HomeTabFragment里面，将自己项目内创建的产品productKey填入下面的pks，再编译运行，才可以创建和体验虚拟设备。

```
/**
 * 将自己项目内的pk填入，可以体验虚拟设备
 */
//String[] pks = new String[]{"a1AzoSi5TMc", "a1B6cFQldpm", "a1XoFUJWkPr", "
a1nZ7Kq7AG1", "a1OX20YJSLk"};
```

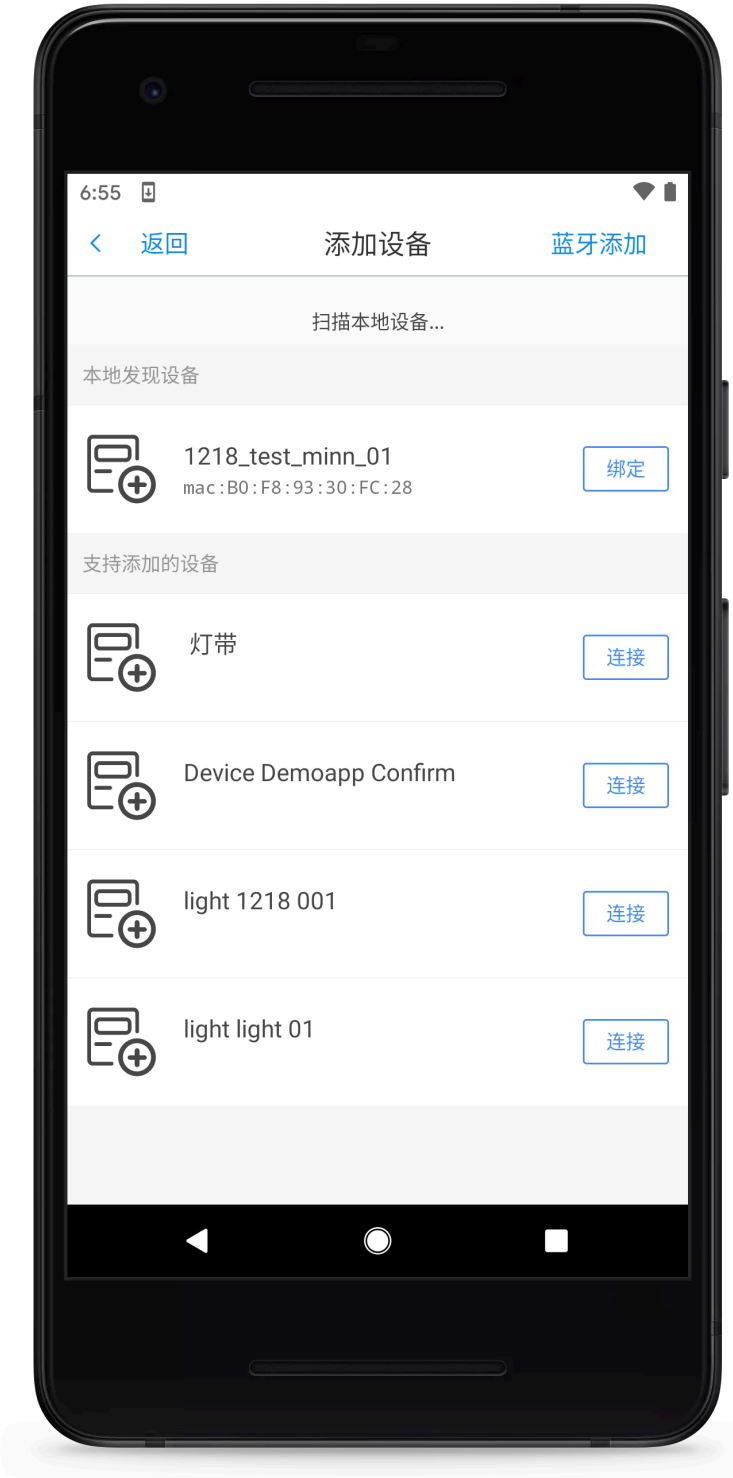
```
String[] pks = new String[]{};
```

b) 单击任意一个设备，进入相应的设备面板界面。

6. 添加设备。

a) 单击首页右上角加号。

b) 选择待添加的设备，进入添加设备界面。



添加设备支持以下两种方式。

- 通过产品列表的形式进行配置（如上图支持添加的设备）

- 通过本地网络发现的设备，或者有支持零配配网方案的设备，如上图本地发现设备（如果能发现到本地设备，则会自动显示出来）。

> **说明：**
> 通过产品列表添加需要使用官方项目对应的接入产品，这一步目前并没有真实设备可以使用；如果想体验这个流程，建议通过自主研发App的形式来完成。

虚拟创建产品的设备

在没有真实设备可以走通链路的情况下，可以使用虚拟设备进行体验和开发。下面介绍如何为您创建的产品虚拟设备。

1. 打开Demo工程com/aliyun/iot/ilop/demo/page/ilopmain/下的HomeTabFragment.java文件。
2. 查找关键字 pks ，快速定位待修改的位置。

```
//注册虚拟设备
Set<String> set = new HashSet<String>();
ArrayList<String> deviceStrList = new ArrayList<>();
for (DeviceInfoBean deviceInfoBean : deviceInfoBeanList) {
    set.add(deviceInfoBean.getProductKey());
    deviceStrList.add(deviceInfoBean.getProductKey() + deviceInfoBean.getDeviceN
ame());
}
mBundle.putStringArrayList("deviceList", deviceStrList);
//noinspection MismatchedReadAndWriteOfArray
String[] pks = new String[]{/*"a1xxxxc", "a1xxxxxm", "a1xxxxr", "a1xxxx1", "a1xxxxk"*/};
mRegisterCount = 0;
mVirtualCount = 0;
//noinspection RedundantOperationOnEmptyContainer
for (String pk : pks) {
    if (set.add(pk)) {
        mRegisterCount++;
        //registerVirtualDevice(pk);
        addVirtualDevice(pk);
    }
```

```
}
```

3. 在控制台上获取待体验产品的**productKey**。

4. 替换**productKey**。

将String[] pks = new String[]{/*"a1xxxxc", "a1xxxxxm", "a1xxxxr", "a1xxxx1", "a1xxxxk"*/};中的默认的productKey替换为自己产品的productKey，示例如下。

```
String[] pks = new String[]{/*"xxxx", "xxxxx"*/};
```

说明：

每个产品可以申请1个虚拟设备，多个产品productKey之间用逗号间隔。

5. 重新登录App，即可使用虚拟设备体验您创建的产品的控制。

5 云端开发指南

5.1 数据HTTP/2方式推送

HTTP/2转储功能适用于生活物联网平台与企业服务器之间的消息流转。通过集成和使用HTTP/2 SDK，即可实现身份认证、消息接收的能力。我们推荐使用HTTP/2的方式推送设备数据（如设备状态数据、设备控制记录等），用户信息数据等。

前提条件

启动设备数据同步，并配置要同步数据的产品。详细参见#unique_136。

说明：

当开启数据同步后，集成HTTP/2客户端SDK来订阅数据。如果通过控制台关闭数据同步再开启时，客户端SDK需要重新进行连接流程，否则无法正常接收数据。如果切换不同的HTTP/2客户端，需要把之前的客户端断开，再连接新的客户端。否则新客户端无法正常接收数据。

HTTP/2 SDK使用

1. 引入依赖。

   在项目中添加maven依赖，maven信息如下。

```
<dependency>
```

```
 <groupId>com.aliyun.openservices</groupId>
 <artifactId>iot-client-message</artifactId>
 <version>1.1.5</version>
</dependency>

<dependency>
   <groupId>commons-codec</groupId>
   <artifactId>commons-codec</artifactId>
   <version>1.11</version>
</dependency>
```

2. 认证身份信息。

   - 身份认证需要使用AppKey和AppSecret，该信息可以从控制台中获取。

   - 认证身份信息需要profile信息，且Profile需要提供EndPoint、AppKey和AppSecret用于鉴权。

```
Profile profile = Profile.getAppKeyProfile("${END_POINT}", "${APP_KEY}", "${
APP_SECRET}");
```

   其中，**EndPoint**是连接节点，具体取值如下表所示。

区域	End Point
中国内地	https://ilop.iot-as-http2.cn-shanghai.aliyuncs.com:443
新加坡	https://ilop.iot-as-http2.ap-southeast-1.aliyuncs.com:443
美国（弗吉尼亚）	https://ilop.iot-as-http2.us-east-1.aliyuncs.com:443
德国（法兰克福）	https://ilop.iot-as-http2.eu-central-1.aliyuncs.com:443

3. 接收云端消息。

   首先需要创建消息接收的客户端对象client，并传入上面身份认证的profile信息。当消息接收的客户端和服务端建立连接后，服务端会立即向消息接收的客户端推送已订阅的消息，因此建立连接时需要提供默认消息接收的回调接口，用于处理云端推送的消息。

   代码片段示例如下。

```
MessageClient messageClient = MessageClientFactory.messageClient(profile);
MessageCallback messageCallback = new MessageCallback() {
      public Action consume(MessageToken messageToken) {
         System.out.println("receive : " + new String(messageToken.getMessage().
getPayload()));
         return Action.CommitSuccess;
      }
  };

messageClient.connect(messageCallback);
```

   - 消息接收的回调接口

     用户需要实现**MessageCallback**接口的consume方法，并调用客户端对象client的`setMessageListener()`方法设置回调接口。

     代码片段示例如下。

```
MessageCallback messageCallback = new MessageCallback() {
   public Action consume(MessageToken messageToken) {
      System.out.println("receive : " + new String(messageToken.getMessage().
getPayload()));
      return Action.CommitSuccess;
   }
};

messageClient.setMessageListener(messageCallback);
```

   - 指定Topic回调

     对于消息接收的回调接口，可以指定带通配符的Topic，如`/${ProductKey}/{$DeviceName}/#`，指定Topic的回调优先级高于通用的回调，一条消息匹配到多个Topic时，按英文字母顺序优先调用，仅会回调一次。

     关于topic的概念请参见#unique_137。

     示例如下。

```
// 当收到消息的Topic匹配到"/Product_A/Device_B/update"时，会优先调用该回调
```

```
messageClient.setMessageListener("/Product_A/Device_B/#",messageCallback);
```

- 通用回调

  设置该回调后，对于未设置指定的topic回调的消息，均会调用该回调。示例如下。

```
// 当收到消息的topic未匹配到已指定的Topic时，调用该回调。该通用回调的设置效果等同于messageClient.connect(messageCallback)的。
messageClient.setMessageListener(messageCallback);
```

完整示例如下。

```
public static void main(String[] args) throws UnsupportedEncodingException {
    Profile profile = Profile.getAppKeyProfile("要连接的<END_POINT>", "您的<AppKey>", "您的<AppSecret>");
    MessageCallback messageCallback = new MessageCallback() {
        public Action consume(MessageToken messageToken) {
            System.out.println("receive : " + new String(messageToken.getMessage().getPayload()));
            return Action.CommitSuccess;
        }
    };

    MessageClient messageClient = MessageClientFactory.messageClient(profile);
    messageClient.setMessageListener(messageCallback);
    messageClient.connect(messageCallback);

    try
    {
        System.in.read();
    } catch (Exception e) {}
}
```

说明：

消息推送失败时，平台会重新推送，重试策略如下。

- 如果对端不在线或未回复ack消息，则会造成消息堆积，堆积的消息转为离线消息。
- 离线消息每隔5min重试推送一次（每次推送10条）。对端如果成功接收了消息，则重试策略会继续推送剩余的离线消息（推送失败的消息，下一次继续推送）。
- 离线消息最多会保存7天，如果7天后仍然无法推送成功，则会被删除。
- 离线消息会进入单独的队列，不会影响后续消息的实时推送。

消息格式

- 物的属性变更消息

  消息字段说明如下。

参数	子参数	子参数	类型	含义
deviceType			String	设备所属品类

参数	子参数	子参数	类型	含义
gmtCreate			Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId			String	设备的唯一id
productKey			String	设备所属产品的唯一标识符
deviceName			String	设备名称
items			JSON	变更的状态列表
	attribute		String	发生变更的属性，具体取值由具体情况确定
		value	具体数据类型由具体情况确定	变更值
		time	Long	设备属性发生变化的时间，自1970-1-1起流逝的毫秒值

消息示例如下。

```
{
  "deviceType": "SmartDoor",
  "iotId": "Xzf15db9xxxxxxxxWR001046b400",
  "productKey": "a17xxxxTYNA",
  "gmtCreate": 153xxxx145304,
  "deviceName": "Xzf15xxxxucTHBgUo6WR",
  "items": {
    "WIFI_Rx_Rate": {
      "value": 74274,
      "time": 1534299145344
    }
  }
}
```

- 物的事件变更消息

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
iotId		String	设备的唯一id
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
identifier		String	事件标识符，对应事件的identifier
name		String	事件名称

参数	子参数	类型	含义
type		String	事件类型
time		Long	设备上报value对应的时间，自1970-1-1起流逝的毫秒值
value		JSON	变更的事件属性列表：key-value键值对
	key	String	属性key
	value	具体数据类型由具体情况确定	属性取值

消息示例如下。

```
{
  "deviceType": "SmartDoor",
  "identifier": "Doorxxxxication",
  "iotId": "Xzf15db9xxxxxxxxx01046b400",
  "name": "开门通知",
  "time": 1534319108982,
  "type": "info",
  "productKey": "a17xxxxTYNA",
  "deviceName": "Xzf15xxxxucTHBgUo6WR",
  "value": {
    "KeyID": "x8xxxxxkDY",
    "LockType": 3
  }
}
```

- 设备服务返回消息

  消息字段说明如下。

参数	类型	含义
gmtCreate	Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId	String	设备的唯一id
productKey	String	设备所属产品的唯一标识符
deviceName	String	设备名称
requestId	String	阿里云产生和设备通信的信息id
code	Integer	调用的结果信息
message	String	结果信息说明
topic	String	服务调用下行时使用的topic

参数	类型	含义
data	Object	设备返回的结果，非透传之间返回设备结果，透传则需要经过脚本转换

消息示例如下。

```
{
  "gmtCreate": 151xxxx39881,
  "iotId": "4z819VQHxxxxxxxxxxxx7ee200",
  "productKey": "p1gxxxxBd",
  "deviceName": "xxxxxxxxxx",
  "requestId": "1234",
  "code": 200,
  "message": "success",
  "topic": "/sys/p1gsv0teUBd/xxxxxxxxxx/thing/service/property/set_reply",
  "data": {}
}
```

- 物的状态变更消息

  为了提高消息有效性，设备上下线过于频繁时，会对消息进行筛检。

  消息字段说明如下。

参数	子参数	类型	含义
deviceType		String	设备所属品类
gmtCreate		Long	数据流转消息产生时间，自1970-1-1起流逝的毫秒值
iotId		String	设备的唯一id
action		String	设备状态变更动作： - online：上线动作 - offline：下线动作
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
status		JSON	状态信息，元素包括：value-状态值，time-发生变化的时间
	time	Long	设备上下线状态发生变化的时间，自1970-1-1起流逝的毫秒值

参数	子参数	类型	含义
	value	String	设备上下线状态 value状态值定义： - 1：在线 - 0：离线

消息示例如下。

```
{
  "deviceType": "SmartDoor",
  "iotId": "Xzf15dxxxxxxxxxxxxxxxx01046b400",
  "action": "online",
  "productKey": "a17xxxxxxTYNA",
  "gmtCreate": 153xxxx1368,
  "deviceName": "Xzf1xxxxxxxxxxxgUo6WR",
  "status": {
    "time": 1534319611368,
    "value": "1"
  }
}
```

- 用户绑定变更消息

  用户绑定/解绑设备产生的回流消息，用于同步用户与设备的绑定、解绑。

  topic: /${pk}/${dn}/thing/awss/enrollee/user

  消息字段说明如下。

参数	子参数	类型	含义
bind		bool	true-绑定；false-解绑
productKey		String	设备所属产品的唯一标识符
deviceName		String	设备名称
iotId		String	设备的唯一id
messageCreateTime		JSON	消息创建时间
identityInfos		list	用户信息列表
	identityId	String	用户身份id
	scopeId	String	隔离id
	tenantId	String	租户id

参数	子参数	类型	含义
	owned	Integer	拥有标记 - 0：分享者 - 1：拥有者
params		Map	扩展参数（暂未使用）

```
{
 "bind":true,
 "productKey": "123xxxx569",
 "deviceName": "deviceNamexxxx34",
 "iotId": "",
 "messageCreateTime": 151xxxx9881,
 "identityInfos":[
   {
    "identityId":"50xxxxxxxxxxx62060259",
    "scopeId":"",
    "tenantId":"1D89B5xxxxxxxxxxxxxxxx861678FF",
    "owned":1
   }
 ],
 "params":{
 }
}
```

5.2 云端调用API

5.2.1 准备工作

介绍用户云端调用阿里生活物联网云端的API流程，获取云端唯一身份AppKey的示例。

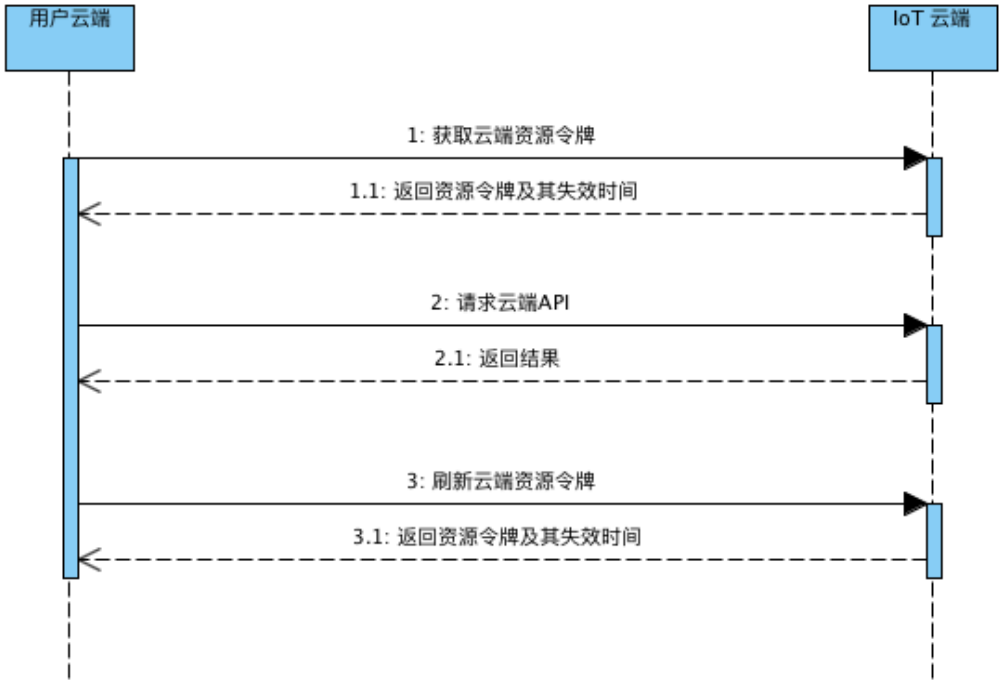
云端服务调用的前置条件

- 云端API使用的AppKey与App使用的AppKey不一致，但是访问环境一致，当需要获取正式环境用户数据时，需要使用云端的正式AppKey。
- 基于安全考虑，所有用户云端接口的调用仅限于HTTPS方式，不支持HTTP方式。
- 所有的API均为POST调用，使用SDK调用的用户可以直接使用API中的postBody方法。

云端服务调用流程

云端接口调用需要资源令牌（cloud token）来做身份鉴权访问。该令牌可通过“云端资源服务”获取。令牌的授权以项目（Project）为粒度，不同的项目会生成不同的令牌，相互之间不能通用。详细参见云端资源服务（云端资源服务默认开通）。

主要流程如下。



获取云端唯一身份Appkey

进入项目设置中获取云云AppKey。

1. 选择项目名称，单击**项目设置**。

2. 在弹出的**权限授权许可**对话框中，单击**确认**。

3. 单击**同意授权**。

4. 选择**设置数据同步**，在**云端接口调用**中，显示线上环境的AppKey和测试环境的AppKey。

调用示例

以下给出Java和PHP两种语言的API调用示例demo，其他语言请参照示例demo中的逻辑和加签算法自行实现。

- 调用API的Java示例

```
// https://github.com/aliyun/iotx-api-gateway-client
public static void main(String[] args) throws UnsupportedEncodingException {
    IoTApiClientBuilderParams ioTApiClientBuilderParams =
      new IoTApiClientBuilderParams();
    ioTApiClientBuilderParams.setAppKey("你的<AppKey>");
    ioTApiClientBuilderParams.setAppSecret("你的<AppSecret>");
    SyncApiClient syncClient = new SyncApiClient(ioTApiClientBuilderParams)   ;

    IoTApiRequest request = new IoTApiRequest();
    //设置API的版本
    //调用/cloud/token接口时，不需要传CloudToken，请参见云端资源服务中/cloud/token
接口的代码示例
    //调用其他接口（除/cloud/token接口外），都需要先调用/cloud/token接口获取
到token，再传入ApiVer和CloudToken，才可以正常调用
    request.setApiVer("1.0.0");
    request.setCloudToken("CloudTokenXXXXX");

    //设置接口的参数
    request.putParam("grantType", "project");
    request.putParam("res", "xxxxx");

    //请求参数域名、path、request
    //host地址 中国站：api.link.aliyun.com 国际站：api-iot.ap-
southeast-1.aliyuncs.com
    ApiResponse response = syncClient.postBody("api.link.aliyun.com",
       "/cloud/token", request, true);
    System.out.println( "response code = " + response.getCode()
                + " response = " + new String(response.getBody(), "UTF-8"));
}
```

- 调用API的PHP示例

```
//https://github.com/aliyun/api-gateway-demo-sign-php (第三方php库)
```

```
<?php
include_once 'Util/Autoloader.php';

//host地址 中国站：api.link.aliyun.com 国际站：api-iot.ap-southeast-1.aliyuncs.com
function example() {
    $path      = "/cloud/token";
    $host      = "https://api.link.aliyun.com";
    $appKey    = "你的<AppKey>";
    $appSecret = "你的<AppSecret>";

    $request = new HttpRequest($host, $path, HttpMethod::POST, $appKey, $appSecret
);
    //设置api版本和参数
    $body = '{"id":"xxx","version":"1.0","request":{"apiVer":"1.0.0"},' .
          '"params":{"grantType":"project","res":"xxx"}}';

    //设定Content-Type
    $request->setHeader(HttpHeader::HTTP_HEADER_CONTENT_TYPE,
                  ContentType::CONTENT_TYPE_JSON);

    //设定Accept
    $request->setHeader(HttpHeader::HTTP_HEADER_ACCEPT,
                  ContentType::CONTENT_TYPE_JSON);

    if (strlen($body) > 0) {
        $request->setHeader(HttpHeader::HTTP_HEADER_CONTENT_MD5,
                    base64_encode(md5($body, true)));
        $request->setBodyString($body);
    }

    //指定参与签名的header
    $request->setSignHeader(SystemHeader::X_CA_TIMESTAMP);

    $response = HttpClient::execute($request);
    var_dump($response);
}

example();
```

5.2.2 云端资源服务

云端资源服务为云端管理的基础服务，通过该服务获取IoT云端资源的令牌，后续请求通过该令牌访问资源。

- 获取云端资源token
- 刷新云端资源token

失效机制

- cloudToken的有效时间通过expireIn来返回，默认有效时长为7200000毫秒（2小时）。
- cloudToken刷新后老cloudToken在5分钟内可用，确保第三方业务可以平滑过渡。

云端资源类型

名称	授权类型（grantType）	资源（res）
项目资源	project	项目ID

获取云端资源token

定义描述

path	请求方法	版本	描述
/cloud/token	POST	1.0.0	获取云端资源Token；该接口对于同一个res（projectId，参见请求参数说明）在token失效前仅需要调用一次，如果再次调用，则会生成新的cloudToken，并会导致之前的cloudToken失效

请求参数

参数	类型	必填	描述
grantType	String	是	授权类型，当前固定为**project**
res	String	是	授权的资源，需填写想要获得授权的**projectId**

返回参数

参数	类型	描述
cloudToken	String	云端Token
expiresIn	long	Token有效期（单位：ms）

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
```

```
    "apiVer": "1.0.0"
  },
  "params": {
    "grantType": "project",
    "res": "xxxxxx"
  }
}
```

- 正常返回示例

```
{
  "id": "1509086454180",
  "code": 200,
  "message": "success",
  "data": {
    "cloudToken" : "cloudToken",
    "expiresIn" : 7200000
  }
}
```

刷新云端资源token

定义描述

path	版本	描述
/cloud/token/refresh	1.0.0	刷新云端Token

请求参数

参数	类型	必填	描述
cloudToken	String	是	-

返回参数

参数	类型	描述
cloudToken	String	云端Token
expiresIn	long	Token有效期（单位：ms）

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "cloudToken":"cloudToken"
  },
  "params": {
    "cloudToken": "cloudToken"
  }
```

```
}
```

- 正常返回示例

```
{
   "id": "1509086454180",
   "code": 200,
   "message": "success",
   "data": {
      "cloudToken" : "cloudToken",
      "expiresIn" : 7200000
   }
}
```

5.2.3 产品管理服务

提供了一组与物的产品相关的服务接口，云端应用通过本组服务接口可以实现对物的产品列表及物的产品查询。

- 查询项目下的产品列表
- 查询单个产品
- 量产生成设备证书（云端自动生成设备名称方式）
- 创建量产批次接口（用户上传设备名称方式）
- 根据量产批次生成设备证书（用户上传设备名称方式）
- 根据批次id查询设备证书列表
- 创建量产批次接口（NVR产品专用）
- 根据批次id查询设备证书列表（NVR产品专用）

查询项目下的产品列表

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/productList/get	1.1.0	查询当前项目下的产品列表	是

请求参数

参数	类型	必填	描述
pageNo	int	是	分页查询页，从1开始
pageSize	int	是	分页大小

返回参数

参数	类型	描述
productKey	String	产品的Key
dataFormat	String	数据格式
netType	String	入网类型
productSecret	String	产品密钥
nodeType	String	节点类型
name	String	产品名称
region	String	地域
categoryId	Long	归属品类id
status	Integer	产品状态 • 0：开发中 • 1：已发布
gmtCreate	Date	创建时间
gmtModified	Date	修改时间
productId	Long	产品Id
rbacTenantId	String	租户Id

示例

- 请求示例

```
{
  "id": 1508232047194,
  "request": {
    "cloudToken": "109049c80xxxxxxxx6f62e29a3ba",
    "apiVer": "1.1.0"
  },
  "params": {
    "pageNo" : 1,
    "pageSize" : 10
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [{
    "gmtModified": 1517562992000,
    "productModel": "niki_test_prodcut_model",
    "productKey": "a1xxxxCf",
    "image": "logo",
    "productId": 95959,
    "dataFormat": "CUSTOM_FORMAT",
```

```
      "netType": "NET_CELLULAR",
      "productSecret": "Y1kxxxxCw",
      "nodeType": "DEVICE",
      "gmtCreate": 1517562992000,
      "name": "test_name",
      "region": "cn-hangzhou",
      "rbacTenantId": "2BF8238E359xxxxxxxx9C7180CF6",
      "categoryId": 61,
      "status": "DEVELOPMENT_STATUS"
    }],
    "id": "db13e9ae-c617-xxxx-xxxx-a1e4f43e36ec"
}
```


查询单个产品

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/product/get	1.1.0	查询指定产品的详细信息	是

请求参数

参数	类型	必填	描述
productKey	String	是	物的产品key

返回参数

参数	类型	描述
productKey	String	产品的Key
dataFormat	String	数据格式
netType	String	入网类型
productSecret	String	产品密钥
nodeType	String	节点类型
name	String	产品名称
region	String	地域
categoryId	Long	归属品类id
status	Integer	产品状态 • 0：开发中 • 1：已发布
gmtCreate	Date	创建时间
gmtModified	Date	修改时间

参数	类型	描述
productId	Long	产品Id
rbacTenantId	String	租户Id

示例

- 请求示例

```
{
  "id": 1508232047194,
  "request": {
    "cloudToken": "109049c8xxxxxxxxf6f62e29a3ba",
    "apiVer": "1.1.0"
  },
  "params": {
    "productKey": "a1dxxxxkl"
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "gmtModified": 1517217645000,
    "productKey": "a1xxxxCW",
    "productId": 87634,
    "dataFormat": "ALINK_FORMAT",
    "netType": "NET_WIFI",
    "productSecret": "tfbxxxxKKj",
    "nodeType": "DEVICE",
    "gmtCreate": 1517208303000,
    "name": "风扇",
    "region": "cn-shanghai",
    "rbacTenantId": "755ED7C7810xxxxxxxxBC48BF0247",
    "categoryId": 285,
    "status": "RELEASE_STATUS"
  },
  "id": "6aad0b12-2192-xxxx-xxxx-08a2bc0b5c2a"
}
```

量产生成设备证书（云端自动生成设备名称方式）

定义描述

path	版本	描述	是否需要cloudToken
/cloud/amount/device/generate	1.1.2	通过该接口可自动生成设备证书（其中设备名称由系统分配），生成设备证书后，可以直接调用/living/device/apply/query接口下载设备证书	是

请求参数

参数	类型	必填	描述
productKey	String	是	物的产品key
amount	Int	是	量产的设备数量

返回参数

参数	类型	描述
batchId	String	量产批次号

示例

- 请求示例

```
{
  "id": "1508232047195",
  "request": {
    "cloudToken": "10904xxxxxxxxf62e29a3ba",
    "apiVer": "1.1.2"
  },
  "params": {
    "productKey": "a1xxxxikl",
    "amount":100
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "12345"
  "id": "1508232047195"
}
```

创建量产批次接口（用户上传设备名称方式）

定义描述

path	版本	描述	是否需要cloudToken
/cloud/device/name/upload	1.1.2	上传设备名称列表获取批次号	是

说明：

该接口需要用户主动上传要量产设备的名称信息，接口会返回该批量产的批次id（batchId），但是此时还未生成设备证书信息，需要再调用“根据量产批次生成设备证书”接口才会真正生成设备证书。

请求参数

参数	类型	必填	描述
productKey	String	是	物的产品key
deviceNames	Array	是	指定的设备名称列表，每批不超过1000个

返回参数

参数	类型	描述
data	String	量产批次号，即batchId

示例

- 请求示例

```
{
   "id": "1508232047196",
   "request": {
      "cloudToken": "109049c8xxxxxxxxf6f62e29a3ba",
      "apiVer": "1.1.2"
   },
   "params": {
      "productKey": "a1xxxxikl",
   "deviceNames": ["deviceName1","deviceName2"]
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": "12345",
```

```
}
```

根据量产批次生成设备证书（用户上传设备名称方式）

定义描述

path	版本	描述	是否需要cloudToken
/cloud/name/device/generate	1.1.2	根据量产批次生成设备证书（在用户上传设备名称的量产场景中使用）	是

说明：

该接口根据批次id（batchId）生成设备证书信息，需配合“创建量产批次接口（用户上传设备名称方式）”使用。

请求参数

参数	类型	必填	描述
productKey	String	是	物的产品key
batchId	String	是	量产的批次号

返回参数

参数	类型	描述
batchId	String	量产批次号

示例

- 请求示例

```
{
  "id": "1508232047195",
  "request": {
    "cloudToken": "109049c8xxxxxxxx2e29a3ba",
    "apiVer": "1.1.2"
  },
  "params": {
    "productKey": "a1dxxxxkl",
  "batchId": "1"
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
```

```
    "data": "1",
    "id": "1508232047195"
}
```

根据批次id查询设备证书列表

定义描述

path	版本	描述	是否需要cloudToken
/living/device/apply/query	1.0.1	根据批次ID查询设备证书列表	是

请求参数

参数	类型	必填	描述
applyId	String	是	批次ID
pageNo	int	是	分页查询页，从1开始
pageSize	int	是	分页大小，最大200

返回参数

参数	类型	描述
totalNum	int	该批次的设备证书总数
items	json列表	设备证书列表

items中参数说明如下

参数	类型	描述
iotId	String	设备的iotId
productKey	String	设备的productKey
deviceName	String	设备的deviceName
deviceSecret	String	设备的deviceSecret
deviceId	String	设备的deviceId
status	int	设备的状态：0-未激活；1-已激活

示例

- 请求示例

```
{
    "request":{
```

```
      "apiVer":"1.0.1",
      "cloudToken":"921fb5617xxxx45c34e9d074c"
   },
   "id":"348f786a-b6b2-xxxx-xxxx-d8a55442399a",
   "params":{
      "applyId":"124639",
      "pageNo":1,
      "pageSize":2
    },
    "version":"1.0"
}
```

- 正常返回示例

```
{
   "code":200,
   "data":{
      "totalNum":90,
      "items":[
         {
            "iotId":"07njQmUOxxxx0101",
            "deviceSecret":"g41b3wEtxxxxHo7Rs9",
            "productKey":"a1rxxxxxTs",
            "deviceName":"07nxxxxxa75U",
            "deviceId":"07nxxxxo8kAa75U",
            "status":0
         },
         {
            "iotId":"0aXlIGxxxxBc2000101",
            "deviceSecret":"bLrDcbxxxxxpk3CrKm",
            "productKey":"axxxxTs",
            "deviceName":"0aXxxxx1Bc2",
            "deviceId":"0aXlxxxxc1Bc2",
            "status":0
         }
      ]
   },
   "id":"348f786a-b6b2-xxxx-xxxx-d8a55442399a"
}
```

创建量产批次接口（NVR产品专用）

定义描述

path	版本	描述	是否需要cloudToken
/living/cloud/product/nvr/device/generate	1.0.0	根据数量量产NVR设备（同时将IPC子产品设备进行量产）	是

请求参数

参数	类型	必填	描述
productKey	String	是	NVR产品的productkey

参数	类型	必填	描述
amount	Int	是	量产的NVR设备数量，最小为1；最大为10000

返回参数

参数	类型	描述
data	String	NVR批次号

示例

- 请求示例

```
{
  "id": "1508232047195",
  "request": {
    "cloudToken": "10904xxxxxxxxf62a3ba",
    "apiVer": "1.0.0"
  },
  "params": {
    "productKey": "a1xxxxikl",
    "amount": 1000
  },
  "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": "11",
  "id": "1508232047195"
}
```

根据批次id查询设备证书列表（NVR产品专用）

定义描述

path	版本	描述	是否需要cloudToken
/living/cloud/product/nvr/device/cert/query	1.0.0	根据批次ID查询设备证书列表	是

请求参数

参数	类型	必填	描述
batchId	String	是	批次ID

参数	类型	必填	描述
productKey	String	是	NVR产品的productkey
pageNo	int	是	分页查询页，从1开始
pageSize	int	是	分页大小，最小为1，最大为20

返回参数

参数	类型	描述
total	int	该批次的NVR设备证书总数
NvrDeviceCertList	json列表	设备证书列表
pageNo	int	分页查询页，从1开始
pageSize	int	分页大小，最小为1，最大为20

NvrDeviceCertList参数说明如下

参数	类型	描述
ipcNum	int	该NVR网关下子设备的数量
subDeviceCertList	DeviceInfo数组	子设备的设备证书列表
nvrDeviceCertInfo	DeviceInfo	NVR设备证书信息

DeviceInfo参数说明如下

iotId	String	设备的iotId
productKey	String	设备的productKey
deviceName	String	设备的deviceName
deviceSecret	String	设备的deviceSecret

示例

- 请求示例

```
{
    "request":{
        "apiVer":"1.0.0",
        "cloudToken":"921fb5617xxxx49d074c"
    },
    "id":"348f786a-b6b2-xxxx-xxxx-d8a55442399a",
    "params":{
        "batchId":"124639",
        "productKey":"axxxxxSbfCe",
```

```
      "pageNo":1,
      "pageSize":2
    },
    "version":"1.0"
}
```

- 正常返回示例

```
{
  "code":200,
  "data":{
    "NvrDeviceCertList":[
      {
        "subDeviceCertList":[
          {
            "iotId":"09cxxxx0101",
            "deviceSecret":"e2RexxxxMepCL4n",
            "productKey":"a1xxxx3Q",
            "deviceName":"09cxxxxFfgn"
          },
          {
            "iotId":"0Ay8zxxxx000101",
            "deviceSecret":"lYQaxxxxSmdMFC",
            "productKey":"a1xxxx3Q",
            "deviceName":"0AyxxxxRpp"
          },
          {
            "iotId":"0Md9bpxxxx00101",
            "deviceSecret":"ZVflxxxxW5pW",
            "productKey":"a1xxxx3Q",
            "deviceName":"0Mdxxxxeoqd"
          }
        ],
        "nvrDeviceCertInfo":{
          "iotId":"vcR91xxxx000101",
          "deviceSecret":"RYV9xxxxXltunD",
          "productKey":"a1xxxxCe",
          "deviceName":"vcRxxxxMUf"
        },
        "ipcNum" :3
      }
    ],
    "totalNum":500,
    "pageNo":235,
    "pageSize":1
  },
  "id":"f0d2acef-xxxx-488b-xxxx-049b95f66a00"
}
```

5.2.4 物的模型服务

提供一组与物（即物联网中的设备）相关的服务接口，云端应用通过本组服务可以实现物信息的查询和控制，查询的内容包含物的基本信息及属性模版等。

- 获取物的属性
- 获取物的模板
- 触发物的服务
- 设置物的属性

- 获取物的连接状态
- 获取物的基本信息
- 批量获取物
- 获取物的事件timeline数据
- 获取物的属性timeline数据
- 查询网关的子设备列表

获取物的属性

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/properties/get	1.0.2	获取物的所有属性快照数据	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空

返回参数

参数	子参数	类型	描述
data		Array	数据列表
	attribute	String	属性名称
	batchId	String	设备属性批量上报的批次
	gmtModified	Date	设备属性采集时间
	iotId	String	设备唯一值
	value	Object	属性值

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D242xxxx4F31A2697"
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "data":[
    {
      "attribute":"AppointmentTime",
      "batchId":"992751fxxxx5b00dac8f4f",
      "gmtModified":1566026333540,
      "iotId":"D95D2xxxxE4F31A2697",
      "value":0
    },
    {
      "attribute":"CO2",
      "batchId":"5729c1c31c6xxxx25fa3b8e",
      "gmtModified":156xxxx430,
      "iotId":"D95D24xxxxF31A2697",
      "value":825
    }
  ],
  "message":"success"
}
```

获取物的模板

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/tsl/get	1.0.2	获取物的tsl功能（包含属性、事件、服务的定义）模板	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D2xxxx4F31A2697"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "schema": "http://aliyun/iot/thing/desc/schema",
    "profile": {
      "productKey": "xxxxxxx",
      "deviceName": "xxxxxxxxxxxxxxxx"
    },
    "link": "/sys/a1xxxxpck/YzqExxxxjjo/thing/",
    "services": [
      {
        "outputData": [

        ],
        "identifier": "ToggleLightSwitch",
        "inputData": [

        ],
        "method": "thing.service.ToggleLightSwitch",
        "name": "翻转主灯开关",
```

```
            "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
          }
        ],
        "events": [
          {
            "outputData": [
              {
                "identifier": "ErrorCode",
                "dataType": {
                  "specs": {
                    "0": "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
          }
        ]
      },
      "message": "success",
    }
```


触发物的服务

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/service/invoke	1.0.2	触发物的服务	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空
identifier	String	是	服务标识符
args	JSON	是	服务入参

返回参数

参数	类型	描述
data	Object	该返回值类型及值取决于设备端实现逻辑

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.1",
      "cloudToken": "token"
   },
   "params": {
      "iotId": "D95D2429xxxx4F31A2697",
      "identifier": "xxxx",
      "args": {}
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": null,
 "message": "success",
 "localizedMsg": ""
}
```

设置物的属性

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/properties/set	1.0.2	设置物的属性	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空
items	JSON	是	set参数

返回参数

无

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxxE4F31A2697",
    "items": {
      "LightSwitch1": 0,
      "LightSwitch2": 0
    }
  }
}
```

```
```

- 正常返回示例

```
{
  "code": 200,
  "data": null,
  "message": "success",
  "localizedMsg": ""
}
```

获取物的连接状态

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/status/get	1.0.2	获取物的连接状态	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空

返回参数

参数	类型	描述
status	Integer	设备状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
time	Long	状态变更时间

示例

- 请求示例

```
{
   "id": "1509086454180",
   "version": "1.0",
   "request": {
      "apiVer": "1.0.2",
      "cloudToken": "token"
   },
   "params": {
      "iotId": "D95D24xxxxCCE4F31A2697"
   }
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
  "time": 1517923297000,
  "status": 3
 },
 "message": "success",
 "localizedMsg": null,
}
```

获取物的基本信息

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/info/get	1.0.2	获取物的基本信息	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空

返回参数

参数	类型	描述
gmtModified	Long	数据修改时间
activeTime	Long	设备激活时间
gmtCreate	Long	设备创建时间
productKey	String	产品唯一标识符
statusLast	Integer	上一次设备状态
mac	String	MAC地址
deviceSecret	String	设备的deviceSecret
iotId	String	设备唯一标识符
name	String	设备名称
nickname	String	设备昵称
sdkVersion	String	SDK版本
sn	String	设备SN
thingType	String	设备类型
region	String	区域
firmwareVersion	String	固件版本号
rbacTenantId	String	租户id
status	Integer	设备状态

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D24xxxxE4F31A2697"
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": {
    "gmtModified": 1517918765000,
    "activeTime": null,
```

```
      "deviceKey": "Kq7rgxxxxexwJh",
      "gmtCreate": 1500017648000,
      "productKey": "kQxxxxqA",
      "statusLast": null,
      "mac": null,
      "netAddress": null,
      "deviceSecret": "xxxxxxxxxxxxxxxxxxxxx",
      "iotId": "03iGXYVlxxxx9ccwgzHG",
      "name": "Kq7rgvxxxxexwJh",
      "tenantId": 7,
      "nickname": null,
      "sdkVersion": null,
      "sn": null,
      "thingType": "DEVICE",
      "region": "daily",
      "firmwareVersion": "1.0.0",
      "rbacTenantId": "ADSBCUYxxxx8327917",
      "ownerDomain": "kQxxxxqA",
      "status": 1
    },
    "message": "success",
    "localizedMsg": null
}
```


批量获取物

定义描述

path	版本	描述	是否需要cloudToken
/cloud/things/get	1.0.2	批量获取物	是

请求参数

参数	类型	必填	描述
productKey	String	是	物的产品唯一标识符
status	Int	是	设备状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用
currentPage	Int	是	分页查询
pageSize	Int	是	分页大小

返回参数

参数	类型	必有	描述
totalNum	Long	是	总记录数

参数	类型	必有	描述
items	List	否	设备列表，列表中参数与获取物的基本信息的返回参数一致

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "productKey": "xxxxxxxxxxxx",
    "status": 3,
    "currentPage": 1,
    "pageSize": 2
  }
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [
    {
      "gmtModified": 1517922329000,
      "offset": null,
      "activeTime": null,
      "deviceKey": "d7Clxxxxu1Lp",
      "gmtCreate": 1501110001000,
      "productKey": "wxxxxtR",
      "statusLast": null,
      "mac": null,
      "netAddress": null,
      "iotId": "d7CllTxxx0010990700",
      "deviceSecret": "u98xxxxzhy/4cV1ThAEGxxxxf48Cf6YX+sqcP1Ky7",
      "name": "\r\n",
      "tenantId": 7,
      "nickname": null,
      "limit": null,
      "sdkVersion": null,
      "sn": null,
      "thingType": "DEVICE",
      "region": "daily",
      "firmwareVersion": null,
      "rbacTenantId": "ADSBCxxxx327917",
      "ownerDomain": "wLxxxxtR",
      "status": 1
    },
    {
      "gmtModified": 1517922329000,
      "offset": null,
      "activeTime": null,
```

```
        "deviceKey": "K1XuExxxxWgiQaq",
        "gmtCreate": 1501123511000,
        "productKey": "wxxxR",
        "statusLast": null,
        "mac": null,
        "netAddress": null,
        "iotId": "K1XuEDskxxxxxxxxq0010d93300",
        "deviceSecret": "bZxxxj/hZs27AJsNKg3xxxxxxxxxxxxjqcP1Ky7",
        "name": "K1XuxxxxgiQaq",
        "tenantId": 7,
        "nickname": null,
        "limit": null,
        "sdkVersion": null,
        "sn": null,
        "thingType": "DEVICE",
        "region": "daily",
        "firmwareVersion": null,
        "rbacTenantId": "ADSBCUYxxxx27918327917",
        "ownerDomain": "wLxxxxtR",
        "status": 1
      }
   ],
   "message": "success",
   "localizedMsg": null
}
```


获取物的事件timeline数据

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/event/timeline/get	1.0.2	获取物的事件timeline数据，最多可以获取设备一年内的数据	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空
identifier	String	是	属性标识符
eventType	String	是	事件类型

参数	类型	必填	描述
start	Long	是	timeline起始时间
end	Long	是	timeline终止时间
pageSize	Int	是	分页查询条数
ordered	Boolean	是	查询方式 • true：顺序 • false：逆序

返回参数

参数	类型	描述
eventCode	String	事件标识符
iotId	String	物的唯一标识符
eventName	String	事件名称
eventType	String	事件类型
eventBody	Object	事件参数
batchId	String	批次id
timestamp	Long	时间采集时间

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D242941xxxxCCE4F31A2697",
    "identifier": "xxxx",
    "eventType": "xxxx",
    "start": 1517217645000,
    "end": 1517217645000,
    "pageSize": 10,
    "ordered": true
  }
}
```

- 正常返回示例

```
{
  "code": 200,
```

```
  "message": "success",
  "localizedMsg": "",
  "data": {
    "items": [
      {
        "eventCode": "Error",
        "iotId": "YzqEnI5Dxxxxjjo0010840500",
        "eventName": "故障上报",
        "eventType": "info",
        "eventBody": {
          "ErrorCode": 0
        },
        "batchId": "5ebc6a9c7dxxxxedde6d28c8fb3",
        "timestamp": 1516342985261
      },
      {
        "eventCode": "Error",
        "iotId": "YzqEnI5Dxxxxxxxxjo0010840500",
        "eventName": "故障上报",
        "eventType": "info",
        "eventBody": {
          "ErrorCode": 0
        },
        "batchId": "4a0b5a7ac8xxxxd5ff77456f1",
        "timestamp": 1516342995305
      }
    ],
    "timestamp": 1516343075699
  }
}
```


获取物的属性timeline数据

定义描述

path	版本	描述	是否需要cloudToken
/cloud/thing/property/timeline/get	1.0.2	获取物的属性timeline数据，最多可以获取设备一年内的数据	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey和deviceName为空时，该入参不允许为空
productKey	String	否（特定条件下必填）	产品唯一标识符，当iotId为空时，该入参不允许为空

参数	类型	必填	描述
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该入参不允许为空
identifier	String	是	属性标识符
start	Long	是	timeline起始时间
end	Long	是	timeline终止时间
pageSize	Int	是	分页查询条数（最大200）
ordered	Boolean	是	查询方式 • true：顺序 • false：逆序

返回参数

参数	类型	描述
iotid	String	设备标识符
data	Object	设备timeline数据
modifytime	Long	属性更新时间
property	String	属性标识符
batchId	String	批次id
timestamp	Long	属性采集时间戳

示例

- 请求示例

```
{
  "id": "1509086454180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.2",
    "cloudToken": "token"
  },
  "params": {
    "iotId": "D95D24294xxxx1ECCE4F31A2697",
    "identifier": "xxxx",
    "start": 1517217645000,
    "end": 1517217645000,
    "pageSize": 10,
    "ordered": true
  }
}
```

- 正常返回示例

```
{
 "code": 200,
 "message": "success",
 "localizedMsg": null,
 "data": {
  "items": [
   {
    "iotid": "xqxyZjSKzCxxxxvbv0O0010851c00",
    "data": 1.23,
    "modifytime": 1511812747287,
    "property": "LightVolt",
    "batchId": "2fc766c5e7xxxx3ed1f3e4b61803",
    "group": null,
    "timestamp": 1511812747245
   },
   {
    "iotid": "xqxyZjSKzxxxx0O0010851c00",
    "data": 1.24,
    "modifytime": 1511812747288,
    "property": "LightVolt",
    "batchId": "2fc766c5exxxx3ed1f3e4b61803",
    "group": null,
    "timestamp": 1511812747245
   }
  ],
  "timestamp": 1511812747245
 }
}
```

查询网关的子设备列表

定义描述

path	版本	描述	是否需要cloudToken
/living/cloud/device/subdevice/query	1.0.0	提供根据网关设备id分页查询子设备列表的能力	是

请求参数

参数	类型	必填	描述
iotId	String	否（特定条件下必填）	物的唯一标识符，当productKey或deviceName为空时，该入参不能为空
productKey	String	否（特定条件下必填）	产品的Key，当iotId为空时，该参数不能为空

参数	类型	必填	描述
deviceName	String	否（特定条件下必填）	设备名称，当iotId为空时，该参数不能为空
pageNo	Int	是	分页查询的页码
pageSize	Int	是	分页查询的每页大小，最大为20

返回参数

参数	一级子参数	二级子参数	类型	描述
data				网关的子设备列表
	total		Long	子设备总数
	pageNo		Int	当前页号，从1开始的页序号
	pageSize		Int	页大小
	items		JSON Array	设备证书列表
		iotId	String	设备ID
		deviceName	String	设备名称
		productKey	String	设备的productKey
		thingType	String	设备的类型： • VIRTUAL • WEB • APP • DEVICE
		firmwareVersion	String	固件版本号，需要设备端上报（如果未上报过，则不会返回该值）
		sdkVersion	String	SDK版本号
		status	Int	设备状态 • 0：未激活 • 1：在线 • 3：离线 • 8：禁用

参数	一级子参数	二级子参数	类型	描述
		nodeType	String	设备节点类型： • DEVICE • GATEWAY
		region	String	设备所属的站点
		rbacTenantId	String	设备所属的租户id
		gmtCreate	Long	记录创建时间，1970年01月01日起至现在的总毫秒数
		gmtModified	Long	记录修改时间，1970年01月01日起至现在的总毫秒数

示例

- 请求示例

```
{
  "id": "15090xxxx4180",
  "version": "1.0",
  "request": {
    "apiVer": "1.0.0",
    "cloudToken": "tokenxxxxxxxx"
  },
  "params": {
   "iotId":"CHWlsAQxxxxxMy4zs000101",
   "pageNo":1,
   "pageSize":10
  }
}
```

- 正常返回示例

```
{
  "code":200,
  "data":{
    "total":1,
    "pageNo":1,
    "pageSize":10,
    "items":[
      {
        "iotId":"c2NBjWQUxxxxxxxxTPFu8000101",
        "gmtModified":15789xxxx6000,
        "thingType":"DEVICE",
        "productKey":"a1HxxxxRiox",
        "region":"cn-hangzhou",
        "gmtCreate":15789xxxx6000,
        "deviceName":"tesxxxxv0",
        "rbacTenantId":"380D6988CxxxxxxxxxD9CBBDAC071",
```

```
            "status":0
          }
        ]
    },
    "id":"3f757e2b-xxxx-xxxx-9926-b435bfe4bb6e"
}
```

5.2.5 用户服务

提供一组与用户相关的服务接口，云端应用通过本组服务可以实现用户的解绑，获取用户绑定的设备列表信息等。

- 分页查询用户列表
- 通过三方外标查询账号信息
- 获取用户绑定的设备列表（包括设备详情）详情内容
- 解绑用户和设备
- 强制解除设备上所有绑定关系
- 根据身份id获取第三方openid
- 根据设备查找所有绑定的用户
- 通过identityid查询账户的详细信息
- 更新自有账号系统在平台内的用户昵称

分页查询用户列表

定义描述

path	版本	描述	是否需要cloudToken
/cloud/account/queryIdentityByPage	1.0.4	分页查询租户下的用户列表	是

请求参数

参数	类型	必填	描述
offset	Integer	否	查询偏移量
count	Integer	否	分页查询数量

返回参数

名称	类型	描述
identityId	String	账号的身份ID
loginId	String	登录ID
loginSource	String	账号来源

名称	类型	描述
gmtCreate	Long	创建时间
gmtModified	Long	修改时间
loginName	String	登录名
phone	String	手机号
email	String	邮箱
nickName	String	昵称
avatarUrl	String	头像
gender	String	性别
lastLoginTime	Date	上次登录时间
onlineStatus	Integer	账号登录态是否在线 • 1：在线 • 0：离线

示例

- 请求示例

```
{
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
    "request": {
     "cloudToken": "109049c8xxxxxxxxf62e29a3ba",
     "apiVer": "1.0.4"
    },
    "params": {
     "offset":0, "count":1
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
  "code": 200,
  "data": [
    {
      "gmtModified": 1508314232000,
      "loginId": "4398047084811",
      "gmtCreate": 1508314232000,
      "identityId": "506bop1ef3axxxxxxxx7ae175ac94618",
      "loginSource": "openAccount",
      "loginName":"test",
      "phone":"13888888888",
      "email":"test@aliyun.com",
      "nickName":"test"
    }
  ],
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
```

```
}
```

通过三方外标查询账号信息

定义描述

path	版本	描述	是否需要cloudToken
/cloud/account/getByOpenId	1.0.4	通过三方外标查询账号信息	是

请求参数

参数	类型	必填	描述
openId	String	是	自有账号外标
openIdAppKey	String	是	应用的appKey

返回参数

名称	类型	描述
identityId	String	账号的身份ID
loginId	String	账号的openAccountId
loginSource	String	账号来源
gmtCreate	Long	创建时间
gmtModified	Long	修改时间
loginName	String	登录名
phone	String	手机号
email	String	邮箱
nickName	String	昵称
tenantId	String	租户ID
gender	String	性别
lastLoginTime	Long	上次登录时间
avatarUrl	String	头像
onlineStatus	Integer	账号登录态是否在线 • 1：在线 • 0：离线

示例

- 请求示例

```
{
    "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
     "request": {
       "cloudToken": "109049c8xxxxxxxx6f62e29a3ba",
       "apiVer": "1.0.4"
     },
     "params": {
       "openId":"4398050099996", "openIdAppKey":"appkey1"
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data":
      {
         "gmtModified":1515590332000,
         "loginId":"4398050099996",
         "gmtCreate":1513181985000,
         "lastLoginTime":1515590332000,
         "phone":"15757245467",
         "identityId":"5091op618xxxxxxxxe93ba653a5928119",
         "loginName":"15757245467",
         "tenantId":"45CFEA7xxxxxxxx05D59452B1437",
         "loginSource":"openAccount"
         "email":"test@aliyun.com",
         "nickName":"test"
      }
,
   "message":"success"
}
```

获取用户绑定的设备列表（包括设备详情）

定义描述

path	版本	描述	是否需要 cloudToken	说明
/cloud/device/queryByUser	1.0.6	获取用户绑定的设备列表（包括设备详情）详情内容	是	查询用户绑定的设备列表；通过设置不同的偏移量来查询任意部分的记录，暂不提供总条数查询

请求参数

参数	类型	必填	描述
openId	String	否	三方用户外标（openId和identityId二者必须有其一）
openIdAppKey	String	否	平台颁发给三方应用的appKey
identityId	String	否	用户身份id（openId和identityId二者必须有其一）
limit	int	否	1.0.5新增参数，获取个数限制，limit取值不能超过100
offset	int	否	1.0.5新增参数，偏移量

返回参数

参数	类型	描述
identityId	String	用户的身份ID
iotId	String	设备的iotId
productKey	String	设备的productKey
deviceName	String	设备的deviceName
productName	String	设备的产品名称
categoryImage	String	设备的产品图片
productModel	String	设备的产品型号
nickName	String	用户对设备的昵称
netType	String	设备入网类型： • lora：NET_LORA • gprs：NET_CELLULAR • wifi：NET_WIFI • zigbee：NET_ZIGBEE • 蓝牙：NET_BT • 以太网：NET_ETHERNET • 其他：NET_OTHER

参数	类型	描述
thingType	String	物的类型： • VIRTUAL（虚拟设备） • VIRTUAL_SHADOW（虚拟影子设备） • WEB • APP • DEVICE（普通设备）
status	Byte	设备的状态，0：未激活 1：在线 3：离线 8：禁用 **说明：** 1.0.6版本之前不支持该字段
owned	int	分享标记 • 0：普通用户（分享者） • 1：管理员（拥有者）

示例

- 请求示例
  - 传入openId和openIdAppKey

```
{
  "id":"55428a70-a740-4xxxx-xxxx-89f727744740",
   "request": {
     "cloudToken": "109049cxxxxxxxx6f62e29a3ba",
     "apiVer": "1.0.6"
   },
   "params": {
     "openId":"10000000001",
     "openIdAppKey":"xxx",
     "limit":50,
     "offset":100
   },
   "version": "1.0"
}
```

  - 传入identityId

```
{
```

```
  "id":"55428a70-a740-xxxx-xxxx-89f727744740",
   "request": {
    "cloudToken": "109049c8xxxxxxxx6f62e29a3ba",
    "apiVer": "1.0.6"
   },
   "params": {
    "identityId": "506bop1efxxxxxxxx57ae175ac94618",
    "limit":50,
    "offset":100
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": {
    "code": 200,
    "data": [{
       "iotId": "VPuHLddp6fixxxxxxxx10dbd910",
       "identityId": "50ceop3441838xxxxxxxx455be61f0356",
       "netType": "NET_WIFI",
       "thingType": "DEVICE",
       "productKey": "a1FNS8TYhpQ",
       "deviceName": "VD_sdDaqyTXeQ",
       "productName": "风扇3-复制"
    }],
    "message": "success"
 },
 "id": "55428a70-a740-xxxx-xxxx-89f727744740"
}
```

解绑用户和设备

定义描述

path	版本	描述	是否需要cloudToken
/cloud/user/device/unbind	1.0.4	解绑用户和设备	是

请求参数

参数	类型	必填	描述
openId	String	否	三方用户外标id（openId和identityId二者必须有其一）
openIdAppKey	String	否	开放平台颁发的appKey（如果使用openId，则该参数为必需）

参数	类型	必填	描述
identityId	String	否	用户身份id（openId和identityId二者必须有其一）
iotId	String	是	iotId

返回参数

无

示例

- 请求示例
  - 传入openId和openIdAppKey

```
{
  "id":"3389580f-c698-xxxx-xxxx-b576c6d6c874",
   "request": {
    "cloudToken": "109049c80xxxxxxxx6f62e29a3ba",
    "apiVer": "1.0.4"
   },
   "params": {
    "openId":"10000000001",
    "openIdAppKey":"xxx"
    "iotId":"VPuHLddpxxxxxxxx0010dbd910"
   },
   "version": "1.0"
}
```

  - 传入identityId

```
{
  "id":"3389580f-c698-xxxx-xxxx-b576c6d6c874",
   "request": {
    "cloudToken": "109049c8xxxxxxxxf62e29a3ba",
    "apiVer": "1.0.4"
   },
   "params": {
    "identityId": "506bop1ef3a1xxxxxxxx7ae175ac94618"
    "iotId":"VPuHLddp6xxxxxxxx010dbd910"
   },
   "version": "1.0"
}
```

- 正常返回示例

```
{
 "code": 200,
 "data": "",
 "id": "3389580f-c698-xxxx-xxxx-b576c6d6c874"
```

```
}
```

强制解除设备上所有绑定关系

定义描述

path	版本	描述	是否需要cloudToken
/living/cloud/user/binding/device/unbind	1.0.0	提供解除指定设备上所有绑定关系的能力	是

请求参数

参数	类型	必填	描述
iotId	String	否	设备唯一标志id；当**productKey**和**deviceName**为空时，该入参不能为空
productKey	String	否	设备的**productKey**，当**iotId**为空时，该参数不能为空
deviceName	String	否	设备的**deviceName**，当**iotId**为空时，该参数不能为空

返回参数

无业务返回参数

示例

- 请求示例

```
{
    "id":"3389580f-c698-4a83-9eee-b576c6d6c874",
     "request": {
       "cloudToken": "109049c80bxxxxx06b15f6f62e29a3ba",
       "apiVer": "1.0.0"
     },
     "params": {
       "iotId":"AawfY6tNxxxxxKOJ9Ffg000101"
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
```

```
  "data": "",
  "id": "3389580f-c698-4a83-9eee-b576c6d6c874"
}
```

根据身份id获取第三方openid

定义描述

Path	版本	描述	是否需要cloudToken
/cloud/account/openId/getByIdentityId	1.0.4	根据身份ID获取第三方OAuth OpenId	是

说明:

当且仅当使用第三方账号体系时，使用该API接口。

请求参数

参数	类型	是否必填	描述
identityId	String	是	身份id
openIdAppKey	String	是	开放平台颁发的appKey

返回参数

参数	类型	是否必填	描述
data	String	是	账号身份id

示例

- 请求示例

```
{
  "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
    "request": {
      "apiVer": "1.0.4"
    },
    "params": {
      "request": {
    "identityId":"506bop1ef3xxxxxxxxf57ae175ac94618",
    "openIdAppKey":"12345678"
    }
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
```

```
   "code": 200,
   "data": "506bop1ef3a1cexxxxxxxx175ac94618",
"id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

根据设备查找所有绑定的用户

定义描述

path	版本	描述	是否需要cloudToken
/living/user/device/binding/query	1.0.0	根据设备iotid查询绑定账户	是

请求参数

参数	类型	必填	描述
iotId	String	是	设备ID
pageNo	Integer	是	分页页数，从1开始
pageSize	Integer	是	分页大小

返回参数

参数	子参数	二级子参数	类型	描述
data			JSON	绑定用户列表
	total		Integer	总记录条数
	accountDevList		jsonArray	绑定列表
		deviceName	String	设备的deviceName
		identityId	String	用户账号id
		iotId	String	设备的iotId
		nickName	String	用户对设备设置的昵称，可为空
		owned	Integer	0：分享者；1：拥有者
		productKey	Date	设备的productKey
		productName	String	设备的产品名称

示例

- 请求示例

```
{
    "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
     "request": {
       "cloudToken": "109049c80bxxxxxxxx62e29a3ba",
       "apiVer": "1.0.0"
     },
     "params": {
       "iotId":"VPuHLddpxxxxxxxx0010dbd910",
       "pageNo": 1,
       "pageSize": 50
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data":
{
   "accountDevList":[
      {
         "deviceName":"00158D00xxxxB3",
         "identityId":"50caop1fxxxxxxxxe55dae27b479165",
         "iotId":"OK61KXgxxxxxxxxtTk000100",
         "nickName":"红外过道2",
         "owned":1,
         "productKey":"a1xxxxyZ",
         "productName":"红外探测器",
      }
   ],
   "total":1
},
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

通过identityid查询账户的详细信息

定义描述

path	版本	描述	是否需要cloudToken
/cloud/account/getByIdentityId	1.0.4	通过IdentityId查询账户的详细信息（IdentityId可以通过/cloud/account/queryIdentityByPage接口获得），无内容值的字段不返回	是

请求参数

参数	类型	必填	描述
identityId	String	是	身份id列表

返回参数

名称	类型	描述
identityId	String	账号的身份ID
loginId	String	登录ID
loginSource	String	账号来源
gmtCreate	Long	创建时间
gmtModified	Long	修改时间
loginName	String	登录名
phone	String	手机号
email	String	邮箱
nickName	String	昵称
avatarUrl	String	头像
gender	String	性别
lastLoginTime	Date	上次登录时间
onlineStatus	Integer	账号登录态是否在线 • 1：在线 • 0：离线

示例

- 请求示例

```
{
    "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
     "request": {
       "cloudToken": "109049c8xxxxxxxxf6f62e29a3ba",
       "apiVer": "1.0.4"
     },
     "params": {
       "identityId":"506bop1ef3xxxxxxxx57ae175ac94618"
     },
     "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data":
```

```
      {
        "gmtModified":1515590332000,
        "loginId":"4398050099996",
        "gmtCreate":1513181985000,
        "lastLoginTime":1515590332000,
        "phone":"157xxxx5467",
        "identityId":"5091op61850061xxxxxxxxba653a5928119",
        "loginName":"157xxxx5467",
        "tenantId":"45CFEA75xxxxxxxxD59452B1437",
        "loginSource":"openAccount"
        "email":"test@aliyun.com",
        "nickName":"test"
      },
   "message":"success"
}
```

更新自有账号系统在平台内的用户昵称

定义描述

path	版本	描述	是否需要cloudToken
/user/account/info/update	1.0.0	-	是

请求参数

参数	类型	必填	描述
openIdAppKey	String	是	平台颁发的OpenId AppKey
openId	String	否	自有账号系统的openId
identityId	String	否（openId/identityId须至少填写一项）	用户账号id
nickName	String	是	用户账号昵称

返回参数

无

示例

- 请求示例

```
{
    "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5",
     "request": {
       "cloudToken": "109049c8xxxxxxxxf6f62e29a3ba",
       "apiVer": "1.0.4"
     },
     "params": {
```

```
      "identityId":"506bop1ef3a1xxxxxxxxf57ae175ac94618",
      "nickName":"飞燕用户001"
    },
    "version": "1.0"
}
```

- 正常返回示例

```
{
   "code": 200,
   "data": {},
   "id":"bb179e80-39fd-xxxx-xxxx-48ca87a3f5c5"
}
```

5.2.6 LinkVisual视频云存储购买服务

本节主要介绍厂商云端调用LinkVisual视频云存储购买服务的流程和API。提供了免费套餐领取及查询接口；还提供了付费套餐相关的查询云存储套餐列表、查询云存储套餐是否可以购买、购买云存储套餐、查询云存储套餐购买订单详情、查询云存储套餐购买订单列表的服务接口。

- 查询免费的云存储套餐详情
- 领取免费的云存储套餐
- 查询视频云存储套餐列表
- 查询云存储套餐是否可以购买
- 购买云存储套餐
- 获取云存储套餐的订单详情
- 查询云存储套餐的订单列表
- 设置免费云存储套餐立即生效
- 设置云存储套餐立即生效
- 用户云存储套餐转移

查询免费的云存储套餐详情

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/freecloudstorage/get	1.0.0	查询免费的云存储套餐详情	可选

请求参数

参数	类型	必有	默认值	描述
iotId	String	是	-	设备id

返回参数

参数	类型	必有	描述
type	int	是	云存储套餐类型：1连续型；2事件型
lifecycle	int	是	存储周期，单位天
months	int	是	服务周期，单位月
consumed	int	是	是否已经领取过，0是尚未领取，1是已领取
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
expired	int	是	是否到期

领取免费的云存储套餐

定义描述

path	版本	描述	是否要cloudToken
/vision/customer/freecloudstorage/consume	1.0.1	领取免费的云存储套餐	可选

请求参数

参数	类型	必有	默认值	描述
iotId	String	是	-	设备id
enableDefaultPlan	boolean	否	false	是否设置默认的7x24小时连续录像计划和事件录像计划

参数	类型	必有	默认值	描述
immediateUse	boolean	否	false	是否立即使用。设置true则为新领取的免费套餐立即生效；设置false则为新领取的免费套餐在之前套餐到期后生效

返回参数

参数	类型	必有	描述
type	int	是	云存储套餐类型：1连续型；2事件型
lifecycle	int	是	存储周期，单位天
months	int	是	服务周期，单位月
consumed	int	是	是否已经领取过，0是尚未领取，1是已领取
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-dd HH:mm:s
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-dd HH:mm:s
expired	int	是	是否到期

查询视频云存储套餐列表

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/commodity/query	1.0.2	查询视频云存储套餐列表	可选

请求参数

无

返回参数

参数	类型	是否必填	描述
cloudStorageCommodityList	List	是	云存储套餐列表

cloudStorageCommodityList

参数	类型	必有	描述
commodityName	String	是	云存储套餐名称
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格
type	int	是	云存储套餐类型：1表示连续型；2表示事件型
price	String	是	云存储套餐价格，单位元
lifecycle	int	是	存储周期，单位天
months	int	是	服务周期，单位月
description	String	否	云存储套餐描述
icon	String	否	云存储套餐图标url

查询云存储套餐是否可以购买

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/commodity/check	1.0.2	查询云存储套餐是否可以购买	可选

请求参数

参数	类型	是否必填	描述
iotId	String	是	设备id
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格

返回参数

参数	类型	必有	描述
available	boolean	是	是否可以购买

购买云存储套餐

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/commodity/buy	1.0.4	购买云存储套餐	可选

请求参数

参数	类型	是否必填	默认值	描述
iotId	String	是	-	设备id
userName	String	是	-	用户名
commodityCode	String	是	-	云存储套餐code
specification	String	是	-	云存储套餐规格
copies	int	是	-	购买数量
enableDefaultPlan	boolean	否	false	是否设置默认的7x24小时连续录像计划和事件录像计划
immediateUse	boolean	否	false	是否立即使用。设置true则为新购套餐立即生效；设置false则为新购套餐在之前套餐到期后生效

返回参数

参数	类型	必有	描述
orderId	String	是	订单id

获取云存储套餐的订单详情

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/order/get	1.0.3	获取云存储套餐的订单详情	可选

请求参数

参数	类型	是否必填	描述
iotId	String	是	设备id
orderId	String	是	订单id

返回参数

参数	类型	必有	说明
orderId	String	是	订单id
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格
copies	int	是	购买数量
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
expired	int	是	是否到期
paymentStatus	int	是	支付状态 • 1：已支付 • 0：未支付
price	String	是	价格元
commodityType	int	是	云存储套餐类型 • 1：连续型 • 2：事件型
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ

查询云存储套餐的订单列表

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/order/query	1.0.3	查询云存储套餐的订单列表	可选

请求参数

参数	类型	是否必填	默认值	描述
iotId	String	是	无	设备id
pageStart	int	否	0	起始页数，默认为0
pageSize	int	否	20	分页大小，最大值为50

返回参数

参数名	类型	必有	说明
pageNo	int	是	当前页号
pageSize	int	是	页大小
pageCount	int	是	总页数
total	int	是	总记录数
orderList	JSON	是	订单列表

orderList

参数	类型	必有	描述
orderId	String	是	订单id
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格
copies	int	是	购买数量
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss

参数	类型	必有	描述
expired	int	是	是否到期
paymentStatus	int	是	支付状态 • 1：已支付 • 0：未支付
price	String	是	订单价格
commodityType	int	是	云存储套餐类型 • 1：连续型 • 2：事件型
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ

设置免费云存储套餐立即生效

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/freecloudstorage/enable	1.0.0	设置免费云存储套餐立即生效	可选

请求参数

参数	类型	是否必填	默认值	描述
iotId	String	是	-	设备id

返回参数

参数	类型	必有	描述
type	int	是	云存储套餐类型：1连续型；2事件型
lifecycle	int	是	存储周期，单位天
months	int	是	服务周期，单位月

参数	类型	必有	描述
consumed	int	是	是否已经领取过，0是尚未领取，1是已领取
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
expired	int	是	是否到期
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ

设置云存储套餐立即生效

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/commodity/enable	1.0.0	设置云存储套餐立即生效	可选

请求参数

参数	类型	是否必填	默认值	描述
iotId	String	是	-	设备id
orderId	String	是	-	订单id

返回参数

参数	类型	必有	描述
orderId	String	是	订单id
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格
copies	int	是	购买数量

参数	类型	必有	描述
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
expired	int	是	是否到期
paymentStatus	int	是	支付状态，0未支付，1已支付
price	String	是	订单价格
commodityType	int	是	云存储套餐类型，1连续型，2事件型
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ

用户云存储套餐转移

定义描述

path	版本	描述	是否需要cloudToken
/vision/customer/cloudstorage/transfer	1.0.0	用户云存储套餐转移。只支持将云存储套餐从同一个管理员账号下的一个设备转移到另一个设备的情形。	可选

请求参数

参数	类型	是否必填	默认值	描述
srcIotId	String	是		设备ID
srcOrderId	String	是		订单ID
dstIotId	String	是		转移目标设备ID
enableDefaultPlan	boolean	否	false	是否设置默认的7x24小时连续录像计划和事件录像计划

参数	类型	是否必填	默认值	描述
immediateUse	boolean	否	false	是否立即使用。设置true则为新购套餐立即生效；设置false则为新购套餐在之前套餐到期后生效

返回参数

参数	类型	必有	描述
orderId	String	是	订单id
commodityCode	String	是	云存储套餐code
specification	String	是	云存储套餐规格
copies	int	是	购买数量
startTime	String	是	开始时间，格式yyyy-MM-dd HH:mm:ss
endTime	String	是	结束时间，格式yyyy-MM-dd HH:mm:ss
expired	int	是	是否到期
paymentStatus	int	是	支付状态，0未支付，1已支付
price	String	是	订单价格
commodityType	String	是	云存储套餐类型，1连续型，2事件型
startTimeUTC	String	是	开始时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ
endTimeUTC	String	是	结束时间（UTC），格式yyyy-MM-ddTHH:mm:ssZ