ALIBABA CLOUD

阿里云

生活物联网平台
App端开发指南

文档版本：20210225

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.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。

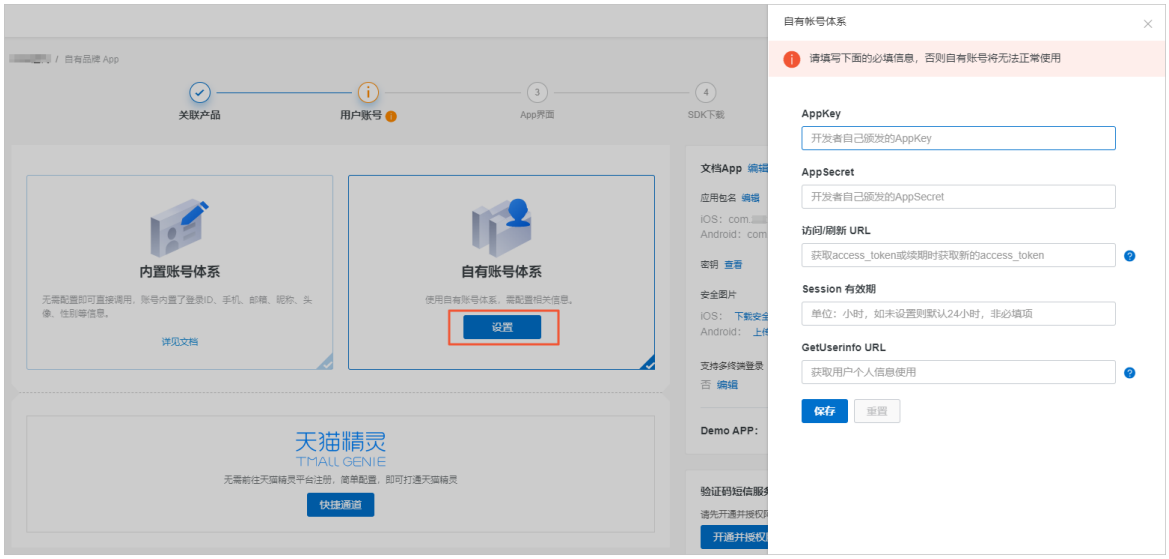
> **说明** 如果您的App需发行到海外，建议您在App中实现以下内容。
>
> - 邮箱注册：邮箱注册更符合海外用户使用习惯。
> - 多语言切换：当前平台支持多种语言，并持续添加中。
> - 注销账号：由于海外严格的合规要求（如GDPR），App中需要提供注销用户的功能，注销时需要删除用户的所有数据。

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。

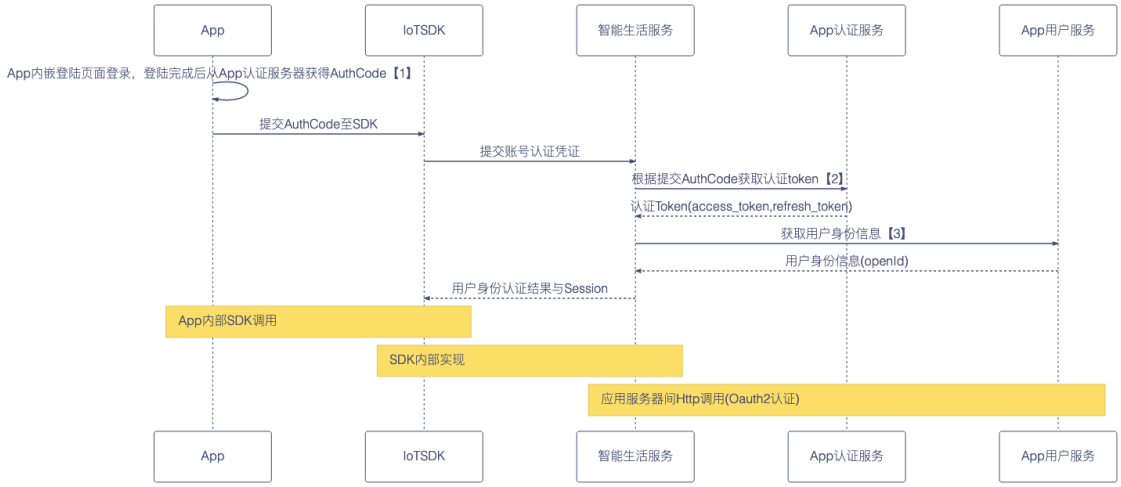
自有账号体系对接基于Oauth 2.0 API协议（代理用户访问的授权开放网络标准协议，详细请参见RFC定义参考文档），请您按照以下步骤配置和开发自有账号对接功能。

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。

i. 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。

ii. 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。

该请求中含有参数client_id、AuthCode、client_secret。对应返回消息中需包含参数result_code、access_token、refresh_token。

> **说明** 生活物联网生活平台账号互联的所有HTTP请求中，请求响应内容 `Content-Type` 需为 `application/json` 。

- AuthCode换取access_token请求

  此处为OAuth 2.0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
```

请求字段	类型	描述
grant_type	String	授权类型，固定为字符串 `authorization_code` 。
client_id	String	生活物联网平台颁发的AppKey。

请求字段	类型	描述
client_secret	String	生活物联网平台颁发的AppSecret。
code	String	流程1中返回的AuthCode。

您的App认证服务向生活物联网平台成功返回的示例如下。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "result_code":"0",
  "openid":"OPENID",
  "access_token":"2YotnxxxxMWpAA",
  "refresh_token":"tGzvxxxx2TlKWIA"
}
```

返回字段	类型	描述
result_code	String	■ 0：成功 ■ 100000：client_id或者client_secert无效 ■ 100002：AuthCode换取access_token失败 ■ 100007：无效的授权码 ■ 110000：系统通用错误代码
openid	String	用户唯一标识
access_token	String	授权令牌
refresh_token	String	获取新的access_token，自动续期授权时需提供该参数。

- 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
```

请求字段	类型	描述
grant_type	String	授权类型，固定为字符串 `refresh_token` 。
client_id	String	生活物联网平台颁发的AppKey。
client_secret	String	生活物联网平台颁发的AppSecret。
refresh_token	String	获取access_token时返回的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",
 "openid":" OPENID",
 "access_token":"2YotnFxxxxsicMWpAA",
 "refresh_token":"tGzv3Jxxxx2TlKWIA",
}
```

返回字段	类型	描述
result_code	String	■ 0：成功 ■ 100000：client_id或者client_secert无效 ■ 100003：refresh_token已经过期或失效 ■ 110000：系统通用错误代码
openid	String	用户唯一标识
access_token	String	授权令牌
refresh_token	String	获取新的access_token，自动续期授权时需提供该参数。

iii. 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 HTTP/1.1
Host: testxx.net
Content-Type: application/x-www-form-urlencoded
```

请求字段	类型	描述
access_token	String	流程2中获取到的access_token。

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"
}
```

返回字段	类型	描述
result_code	String	■ 0：成功 ■ 100000：client_id或者client_secert无效 ■ 100001：access_token过期 ■ 100002：AuthCode换取access_token失败 ■ 100003：refresh_token已经过期或失效 ■ 100004：用户修改密码或者解除授权导致access_token失效 ■ 100005：access_token非法 ■ 100006：无效的OpenId ■ 100007：无效的授权码 ■ 110000：系统通用错误代码

返回字段	类型	描述
message	String	响应结果描述
openid	String	用户唯一标识
nick_name	String	昵称
avatar_url	String	头像URL
gender	String	性别 ■ 0：未知 ■ 1：男性 ■ 2：女性

自有账号全球用户接入

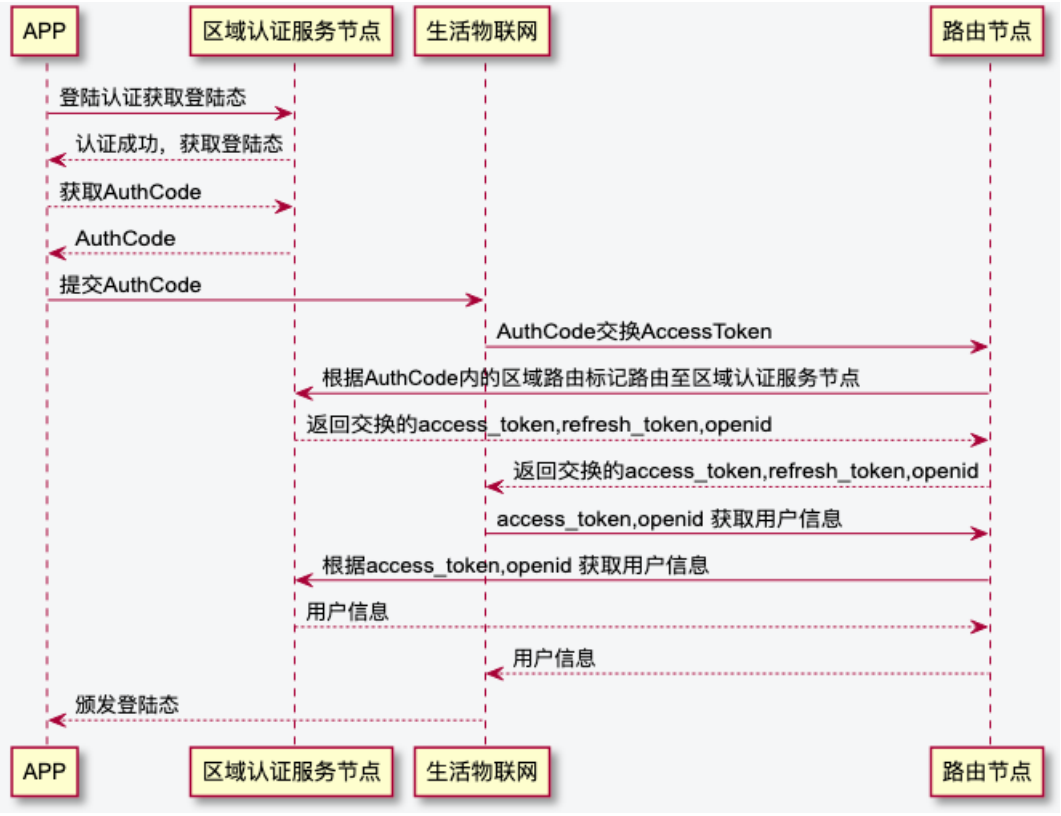
生活物联网平台提供全球统一的账户接入能力，当有多个数据中心时会就近接入，从而达到最快接入响应和合规能力支持。您的自有账号系统只允许配置一个endpoint（即设置的URL的域名，在App账号接入的token地址和getUSerInfo地址中获取）。如果您的自有账号系统在全球具有多个数据中心，可以通过以下方法兼容多数据中心。

1. 在区域认证服务生成authCode时，添加数据中心的标志位。

   需要添加标志位的地方有：AuthCode中、AuthCode交换生成的access_token和refresh_token中。

2. 使用统一的endpoint转发路由。

   详细流程图如下。



以下为您推荐几种统一转发路由的实现方法，供您参考。

- 使用智能CDN，并自建nginx反向代理集群。
- 使用智能CDN，流量通过API网关接入阿里云函数计算。
- 使用阿里云的边缘服务产品，例如EdgeRoutine等。

获取用户上传头像的URL及加密签名

开发自有App的用户账号时，如果App需要支持C端用户上传头像及修改签名的功能，请您根据以下流程来开发。

1. 调用获取用户上传头像的url及加密签名接口，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PostObject）。

   上传时Header中的x-oss-forbid-overwrite参数需设置为false，否则可能导致上传失败。
3. 拼接URL。

   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 ）。
4. 通过修改账号属性接口更新图像URL。

注册登录App开发说明

开发自有App的用户账号时，您还需要注意App注册和登录时国家的选择，如下表所示。

操作	API Level 9及以上版本	API Level 8及以下版本
注册App	需要选择国家	需要选择国家
登录App	不需要选择国家	选择注册时的国家

账号注册时选择的国家，在账号注册成功后无法切换。该账号绑定的设备也将连接到所选国家对应区域的数据中心。如果您需切换数据中心的区域，则要先注销账号，再重新选择新的国家注册，并绑定原来的设备。

因此，您在开发App注册和登录的业务逻辑时，建议参考云智能App页面的实现。当需要选择国家时，强提示C端用户（消费者）选择国家的影响（国家决定了账号所在地和后续该账号绑定设备连接的服务器区域），避免消费者随意选择一个国家，导致设备连接的体验不佳。

2.配网开发指南

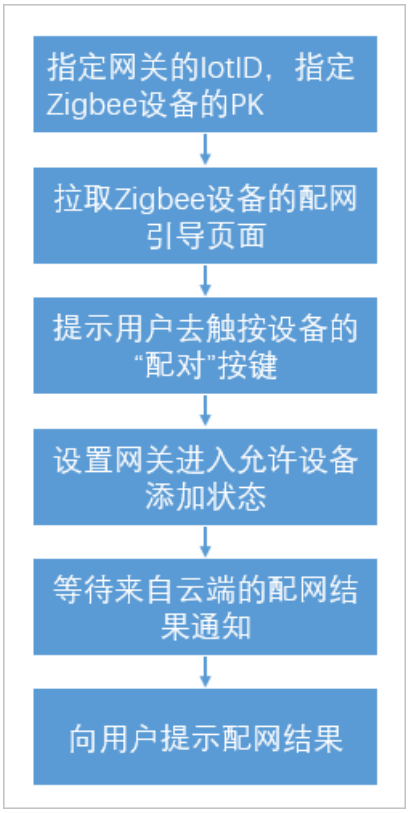
本节将会介绍如何开发App端的设备配网、用户绑定等功能。

- 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（Wi-Fi、以太网）
  - 基于时间窗口方式的设备绑定（蜂窝网、蓝牙、Zigbee、其他）

基于配网SDK自行开发配网界面

- Wi-Fi设备配网界面开发
  - 配网方案介绍，请参见Wi-Fi 配网方案介绍。
  - Android APP SDK的介绍，请参见配网SDK。
  - iOS APP SDK的介绍，请参见配网SDK。
  - 配网相关的最佳实践，请参见Android App Native开发配网。
- 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的配网流程如下图所示。



- 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` 。具体操作，请参见产品管理服务。

- 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` 具体操作，请参见配网服务。

  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Wi-Fi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Wi-Fi、以太网和蜂窝网的配网流程。其中Wi-Fi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时需使用该参数。	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。该参数配置如下。 • 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：

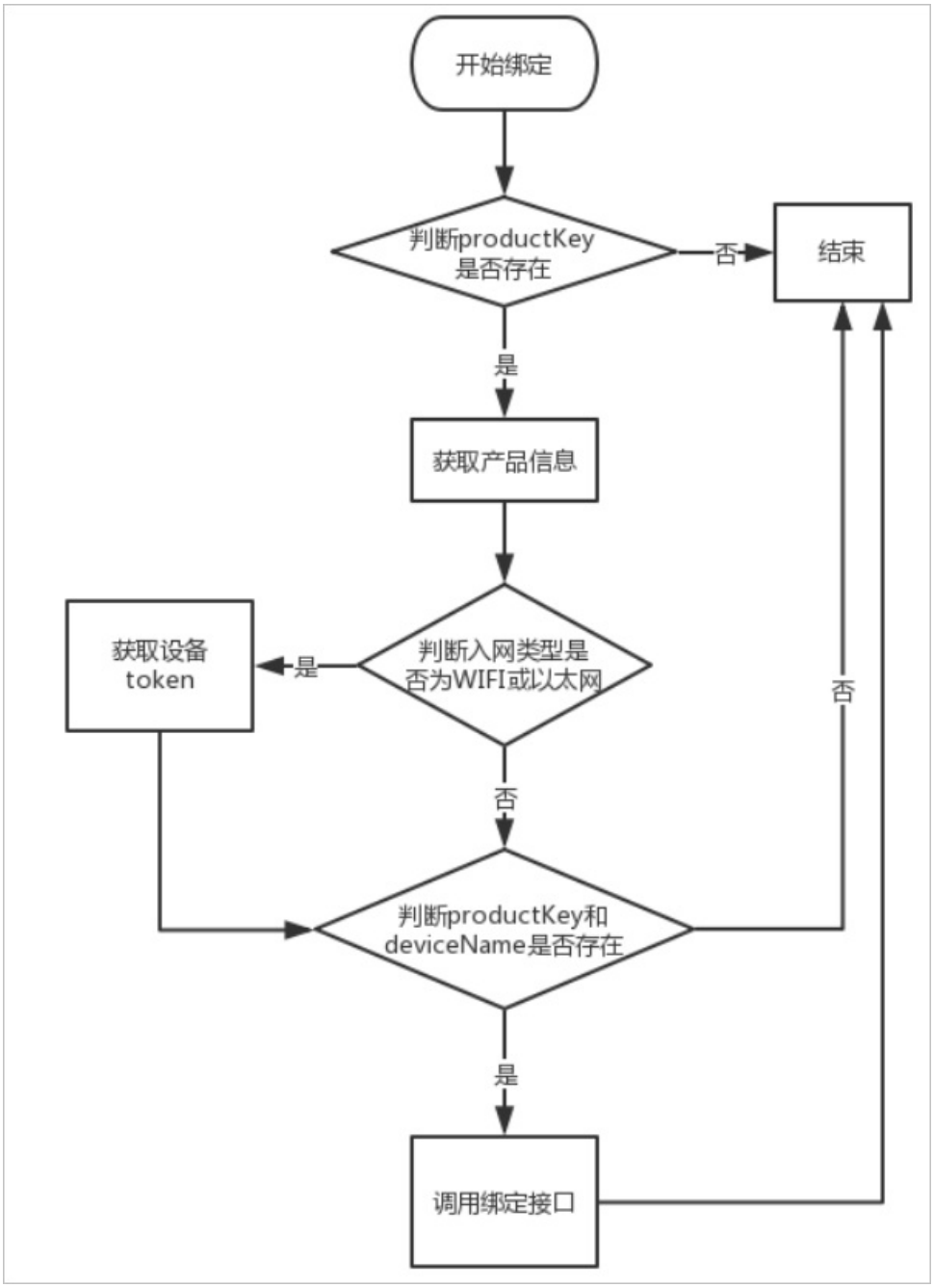
```
// 启动插件
Bundle bundle = new Bundle();
bundle.putString("productKey", pk);
Router.getInstance().toUrlForResult(activity, "link://router/connectConfig",{your_request_code}, bundle
);
// 接收配网结果
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  ...
  if (REQUEST_CODE_PRODUCT_ADD == 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
  }
}
```

- 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。

- Wi-Fi、以太网，使用基于token方式设备绑定接口。
- 蜂窝网、蓝牙、Zigee、其他，使用基于时间窗口方式的绑定设备接口。
- 更多配网相关API的介绍，如本地发现、配网、绑定等，请参见配网服务。
- 用户关系API的介绍，如解绑、查询绑定关系、分享设备等，请参见用户服务。
- 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) {
    // 获取到绑定需要的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.插件使用指南

生活物联网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Wi-Fi、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。

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	设备ID，必选参数

- 配网插件

参数	类型	描述
productKey	String	产品ProductKey，必选参数
deviceName	String	设备名称，GPRS设备必选

- 云端定时

参数	类型	描述
iotId	String	设备ID，必选参数

- 本地定时

参数	类型	描述
iotId	String	设备ID，必选参数

- 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和家空间下的场景仅支持 IFTTT、CA。
sceneId	String	仅在编辑时传入
homeId	String	家空间Id ○ 创建家空间场景必填 ○ 创建非家空间创建不要传

- 不传sceneId时为新增场景

- 传sceneId时为编辑场景

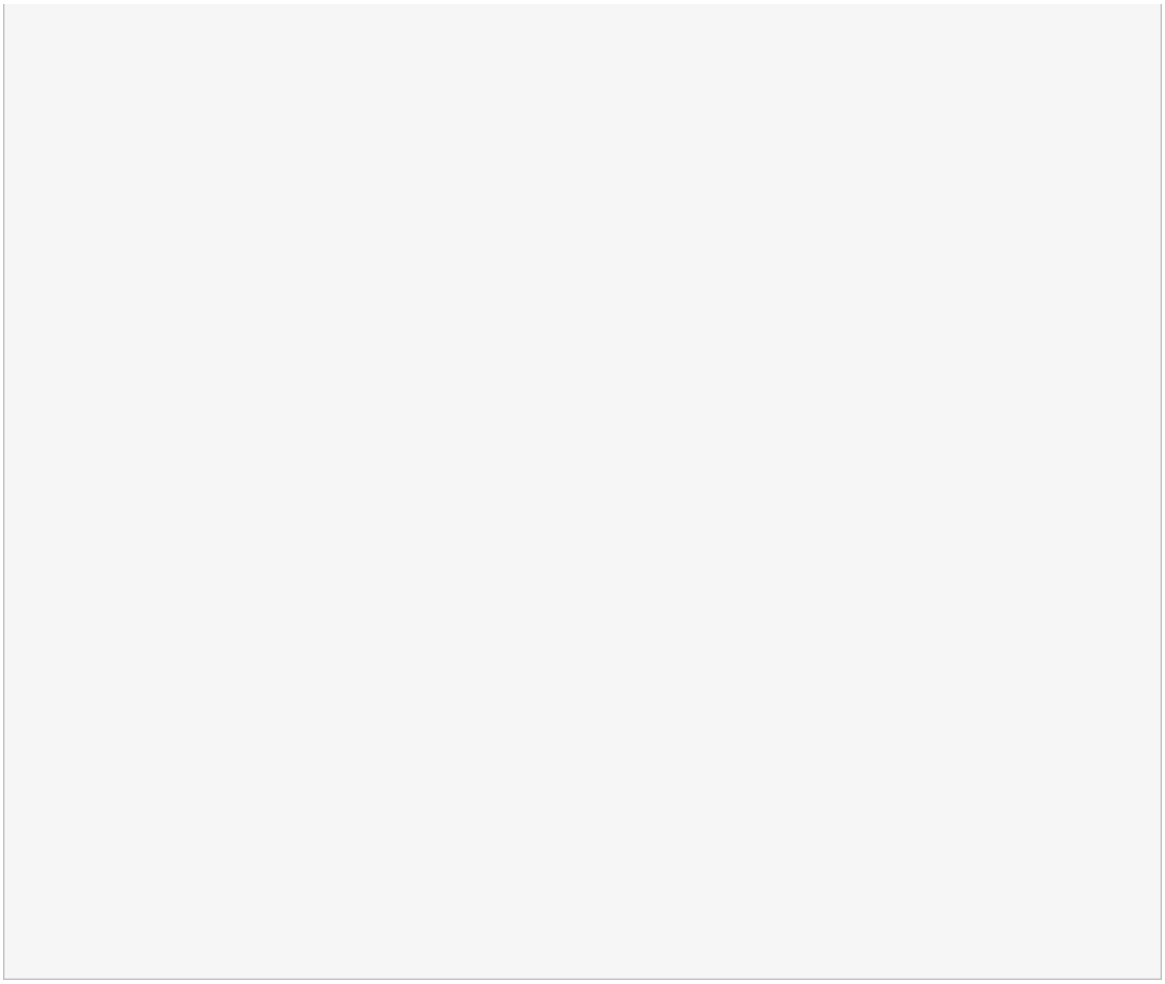
- 设备消息记录

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


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



- 意见反馈插件

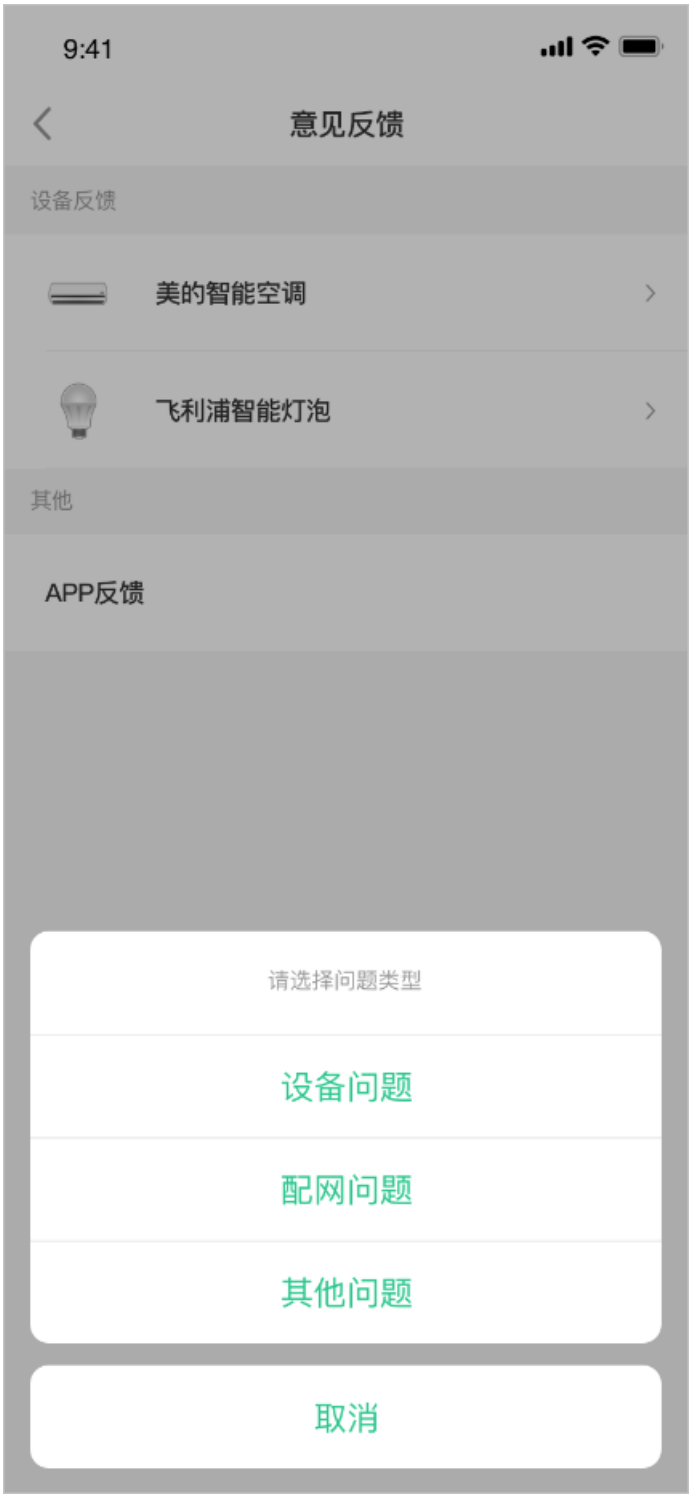
参数	类型	描述
mobileModel	String	手机型号
mobileSystem	String	手机系统
appVersion	String	App版本

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

- 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

- 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）。详细请参见用户反馈。

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）。详细请参见配置App设备告警。
- 已完成自有品牌App的创建，详细请参见创建自有App。
- 下载SDK套餐包时勾选了消息推送，详细请参见下载并集成SDK。

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。

i. 单击**上传文件**，并上传APNs推送证书。

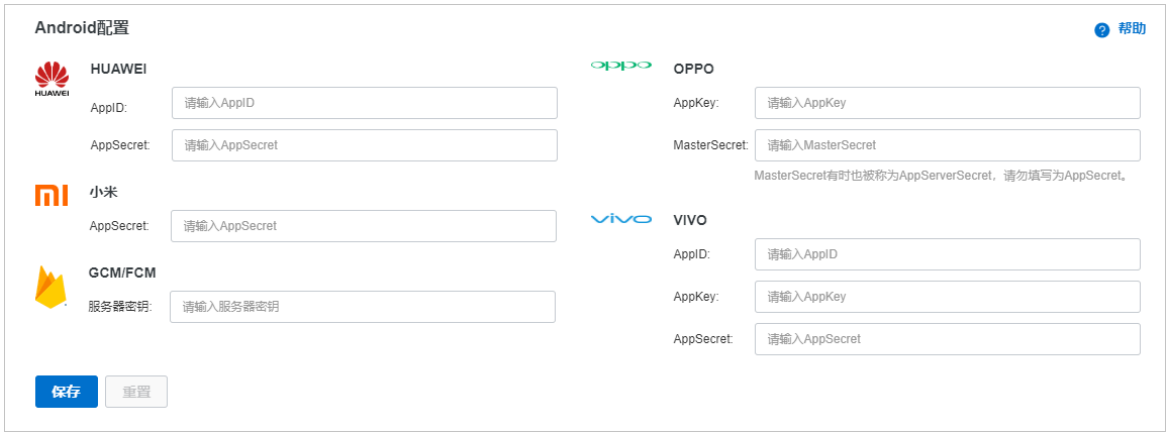
ii. 输入推送证书的密码，并单击**验证证书**。

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。

iii. 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。

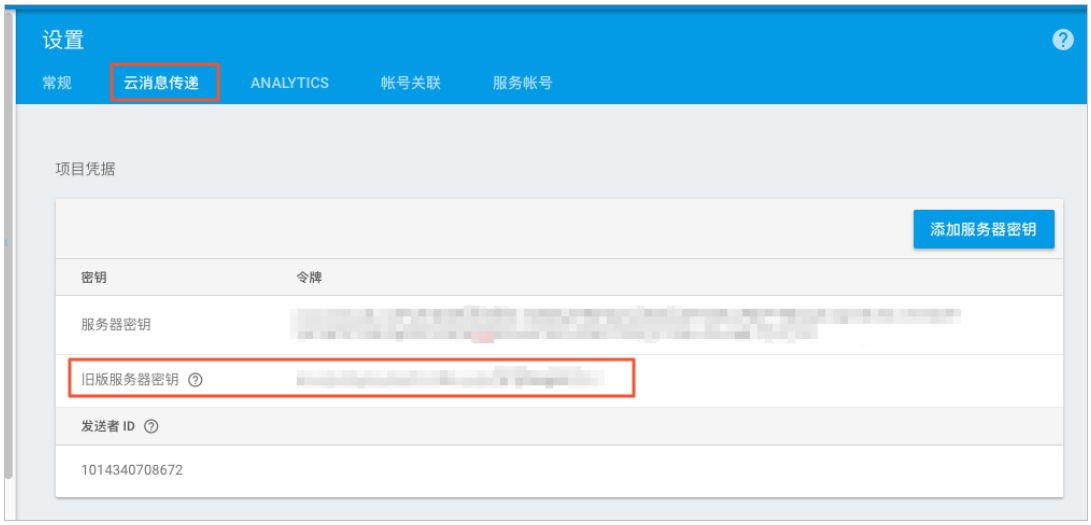
- 小米
  a. 在小米开放平台注册App，得到相应的小米AppID、小米AppKey、小米AppSecret。
  b. 打开小米开发者平台push功能区的消息推送开关。
  c. 在生活物联网控制台**消息推送设置**页面配置小米AppSecret。
- FCM（Google已将GCM推送迁移至Firebase，改称FCM）

  > **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（该参数在SDK初始化中需要用到）、applicationId（该参数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，详细参见配置App设备告警。

- 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”`。
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。

5.场景自动化开发指南

生活物联网平台支持自有App实现场景自动化功能，即支持C端用户（消费者）配置场景和自动化任务。

场景与自动化的区别如下。

- 场景：需要用户手动在App上单击触发后，执行一系列任务。

  例如，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。

  例如，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。

功能开发

场景自动化功能可以通过以下两种方式实现：调用场景插件和基于API自行开发。两种实现方式的对比如下。

对比项	调用场景插件	基于API自行开发
优缺点	该实现方式开发工作量较少，无需您开发业务逻辑，只需集成平台提供的插件即可（关于如何集成插件的操作，请参见插件使用指南）。	该实现方式可以更灵活地体验功能和界面，但对技术要求较高，开发工作量也较大（API接口描述，请参见场景服务2.0）。
支持的功能	• 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，即需要分别设置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	• 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（持续更新中） • 触发条件的关系：满足所有条件、满足任一条件 • 执行任务支持：设备动作、执行场景、发送手机推送通知

规则说明

当您基于API自行开发场景自动化功能时，您需要按照以下规则来设定业务逻辑。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）。目前主要支持以下两种规则形式。

- IFTTT（If This Then That）

  IFTTT规则即自动化，支持Trigger、Condition、Action节点。详细介绍请参见本文档下方“IFTTT规则介绍”。

- CA（Condition & Action）

  CA是对IFTTT的简化，将Trigger与Condition合并在一起作为“条件”。当满足条件（多个Condition之间可设置all或any）时，就执行Action。详细介绍请参见本文档下方“CA规则介绍”。

场景自动化的规则的限制条件如下。

- 一个用户最多可以创建200个场景或自动化。

- 一个设备最多可以作为20个场景规则的Action。
- 一个规则内必须具有Action节点，且最多可以有30个Action。
- 一个规则内可以没有Trigger节点，如果有最多10个Trigger。

  没有Trigger节点时，表示场景不会自动触发。此时可以调用执行场景接口来手动触发场景。例如，仅设置了 `condition/timeRange` 。
- 一个规则内可以没有Condition节点，如果有最多5个Condition。

  没有Condition节点时，Trigger节点触发后直接执行Action。
- 一个规则内最多只能有一个 `trigger/timer` 节点。
- 一个规则内最多只能有一个 `condition/timeRange` 节点。
- CA规则中，只有all模式支持 `condition/timeRange` 节点，any模式下将忽略该节点。

IFTTT规则介绍

IFTTT规则中包含的节点如下表所示。

名称	描述
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。

下面为您提供两个IFTTT的示例。

- 典型示例

  该示例中具有一个Trigger、一个Condition、一个Action的场景规则。

  示例的规则说明：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"
      }
    }
  ]
}
```

- 简化示例

  该示例展示了具有多个Trigger和多个Condition的场景规则。 此时使用*logical/or*节点封装多个Trigger，使用*logical/and*节点封装多个Condition。

示例的规则说明：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，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。

```
{
  "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"
        }
      }
    ]
}
```

CA规则介绍

CA规则是IFTTT规则的简化，规则包含的节点如下表所示。

节点名称	描述
mode	CA规则的模式，可取值如下： • any：表示任何一个条件满足就执行动作。 • 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。

以下为CA规则的示例。

示例的规则说明：当设备sensor_01的some_property属性大于300、设备sensor_02的BarrierState属性取值为0、1、2，且时间点在10:05时，将设备test_device的LightStatus属性设置为1。

```
{
  "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/setProperty",
      "params":{
        "productKey":"pk_action",
        "deviceName":"test_device",
        "propertyName":"LightStatus",
        "propertyValue": 1
      }
    }
  ]
}
```

Trigger节点介绍

目前支持的Trigger节点包括三种。

- trigger/timer

  表示定时触发。参数描述如下。

参数	类型	描述
cron	String	定时表达式，cron表达式格式为 `http://crontab.org/` 。
cronType	String	表达式类型，详细描述参见下方表格。
timezoneID	String	时区ID，表示将按照哪个时区来执行定时表达式。该参数值可为空，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。 配置示例： `Asia/Shanghai` 。

  参数cronType的详细介绍如下。

取值	表达式结构	表达式字段	描述及示例
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	表示crontab类型，共5位，不支持年。 示例： `13***` 表示每天03:01触发。

取值	表达式结构	表达式字段	描述及示例
quartz_cron	${second} ${minute} ${hour} ${day of month} ${month} ${day of week} ${year}	○ 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）	quartz类型cron，7位，支持年。禁止配置为无法抵达的过去时间。当前只能最小设置分钟级别，即第一位秒级必须为0。 示例：`0 * 14 * * ? 2019` 表示在2019年每天下午2点到下午2:59期间的每1分钟触发。

示例如下。

```
{
  "uri":"trigger/timer",
  "params":{
    "cron":"* * * * *",
    "cronType":"linux",
    "timezoneID":"Asia/Shanghai"
  }
}
```

- trigger/device/property

  表示设备属性触发。参数描述如下。

参数	类型	描述
productKey	String	产品的Key，设备证书信息之一。创建产品时，生活物联网平台为该产品颁发的全局唯一标识。
deviceName	String	设备的名称，设备证书信息之一。在注册设备时，自定义的或系统生成的设备名称，具备产品维度内的唯一性。
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。

示例如下。

```
{
  "uri":"trigger/device/property",
  "params":{
    "productKey":"test_pk",
    "deviceName":"test_dn",
    "propertyName":"temp",
    "compareType":">",
    "compareValue":30
  }
}
```

- trigger/device/event

  表示设备事件触发。参数描述如下。

参数	类型	描述
productKey	String	产品的Key，设备证书信息之一。创建产品时，生活物联网平台为该产品颁发的全局唯一标识。
deviceName	String	设备的名称，设备证书信息之一。在注册设备时，自定义的或系统生成的设备名称，具备产品维度内的唯一性。
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。

  示例如下。

```
{
  "uri":"trigger/device/event",
  "params":{
    "productKey":"test_pk",
    "deviceName":"test_dn",
    "eventCode":"temp_warning",
    "propertyName":"temp",
    "compareType":">",
    "compareValue":30
  }
}
```

Action节点介绍

目前支持的Action节点包括以下五种。

- action/device/setProperty

  表示设置设备属性。参数描述如下。

参数	类型	描述
iotId	String	设备ID，生活物联网平台为设备颁发的ID，设备的唯一标识符。
propertyName	String	要设置的属性的名称，例如：PowerSwitch。
propertyValue	Object	要设置的属性的值。
delayedExecutionSeconds	Integer	延时执行时间，默认为空，即立即执行，设置了该值后才执行延时操作。单位为秒，最小1s，最大86400s（24小时）。

  示例如下。

```
{
  "uri":"action/device/setProperty",
  "params":{
    "iotId":"xxxx",
    "propertyName":"PowerSwitch",
    "propertyValue":1,
    "delayedExecutionSeconds":10
  }
}
```

- action/device/invokeService

表示调用设备服务。参数描述如下。

参数	类型	描述
iotId	String	设备ID，生活物联网平台为设备颁发的ID，设备的唯一标识符。
serviceName	String	服务名称。
serviceArgs	JSON	服务参数，例如：`{"warningStatus":"off","level":"init"}`。

示例如下。

```
{
  "uri":"action/device/invokeService",
  "params":{
    "iotId":"xxxx",
    "serviceName":"clearWarningStatus",
    "serviceArgs":{
      "warningStatus":"off",
      "level":"init"
    }
  }
}
```

- action/scene/trigger

  表示触发另一个场景时，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。参数描述如下。

参数	类型	描述
sceneId	String	场景ID。

  示例如下。

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

- action/automation/setSwitch

表示启用或停用另一个场景设置。参数描述如下。

参数	类型	描述
automationRuleId	String	要启用或停用的场景ID。
switchStatus	Integer	场景状态：0表示停用，1表示启用。

示例如下。

```
{
  "uri":"action/automation/setSwitch",
  "params":{
    "automationRuleId":"xxxx",
    "switchStatus":0
  }
}
```

- /action/mq/send

  表示向手机推送消息。参数描述如下。

参数	类型	描述
customData	CustomData	推送消息的模型。
customData. message	String	向手机推送消息的内容，最多60个字符。
msgTag	String	推送消息模型，默认值为IlopBusiness_CustomMsg。

  示例如下。

```
"actions": [
  {
    "uri":"action/mq/send",
    "params":{
      "customData":{
        "message":"电源开启了"
      },
      "msgTag":"IlopBusiness_CustomMsg"
    }
  }
]
```

IFTTT规则中Condition节点介绍

目前IFTTT规则支持的Condition节点包括以下两种。

- condition/timeRange

  表示比较当前时间是否在一个区间内。参数描述如下。

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，取值为linux。
timezoneID	String	示例： `Asia/Shanghai` 。

  示例如下。

```
{
  "params":{
    "cron":"0-0 9-12 * * 1,2,3",
    "cronType":"linux",
    "timezoneID":"Asia/Shanghai"
  },
  "uri":"condition/timeRange"
}
```

- condition/device/property

  表示设备属性过滤。参数描述如下。

参数	类型	描述
productKey	String	产品的Key，设备证书信息之一。创建产品时，生活物联网平台为该产品颁发的全局唯一标识。
deviceName	String	设备的名称，设备证书信息之一。在注册设备时，自定义的或系统生成的设备名称，具备产品维度内的唯一性。
propertyName	String	设备属性。
compareType	String	比较类型，如>、<、>=、==、<=、!=、in、like等。
compareValue	Object	比较的值。

  示例如下。

```
{
  "uri":"condition/device/property",
  "params":{
    "productKey":"test_product",
    "deviceName":"test_device",
    "propertyName":"temp",
    "compareType":">",
    "compareValue":30
  }
}
```

CA规则中Condition节点介绍

目前CA规则支持的CaCondition节点包括以下三种。

- condition/timer

  表示定时器。作用和参数与IFTTT规则下的 `trigger/timer` 节点相同。详细介绍请参见本文档“IFTTT规则介绍”。

- condition/timeRange

  表示时间段，只能用于all模式，any模式下将被忽略。

  > **说明** CA规则中 `condition/timeRange` 节点与IFTTT规则中的 `condition/timeRange` 节点作用相同，参数形式不同。

  参数描述如下。

参数	类型	描述

参数	类型	描述
format	String	时间的格式： ○ 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
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，需符合format指定的格式。
endDate	String	结束时间，需符合format指定的格式。 **说明** 如果beginDate大于endDate，将视为跨天。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，用逗号分隔的1~7数字构成的字符串。该参数可为空，表示不重复。例如配置为 1,2,3 ，表示在星期一、星期二、星期三重复。

示例如下。

```
{
  "uri":"condition/timeRange",
  "params":{
    "format":"HH:mm",
    "beginDate":"8:30",
    "endDate":"23:00",
    "repeat":"1,2,3,4,5"
  }
}
```

- condition/device/property

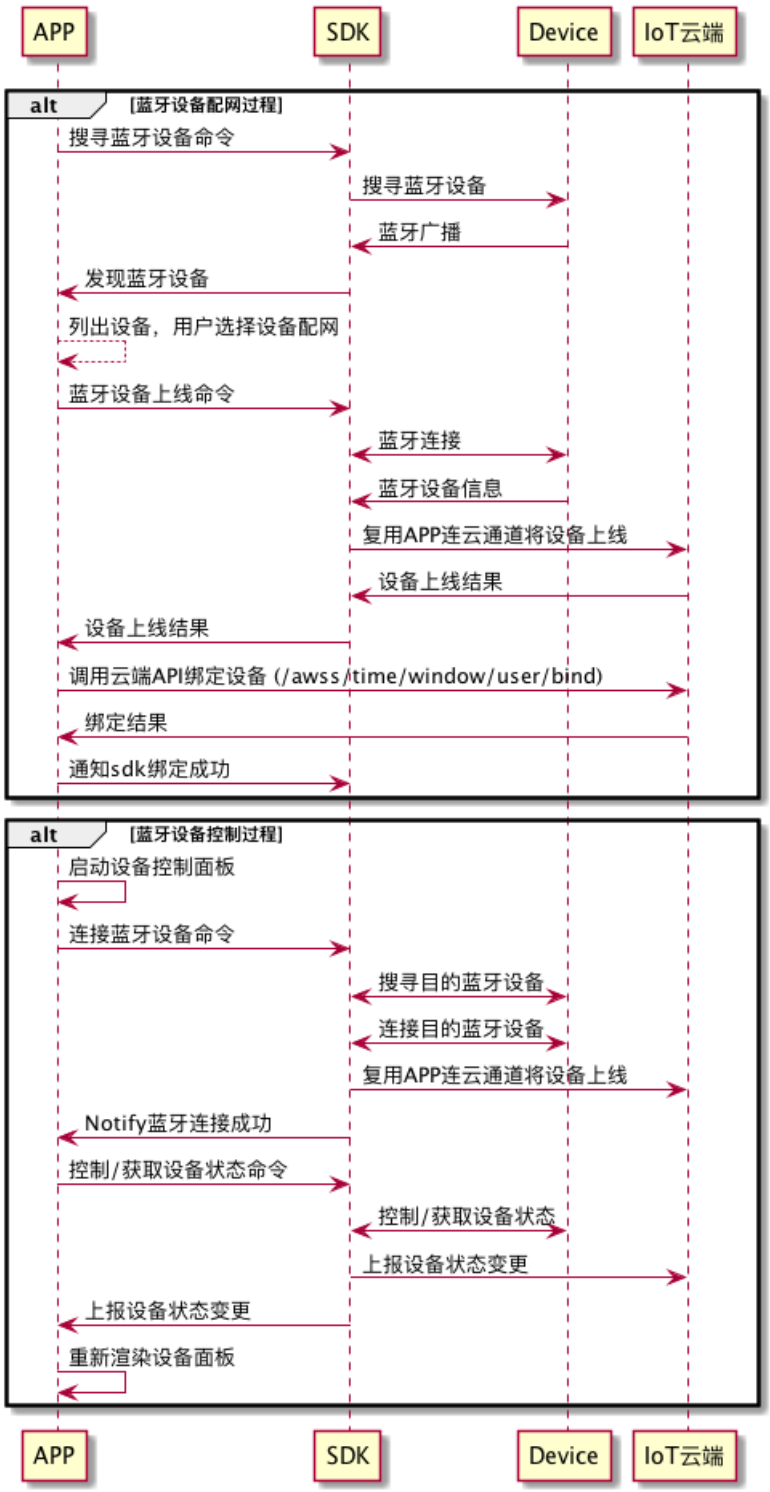
  表示设备属性。作用和参数与IFTTT规则下的 `trigger/device/property` 相同。详细介绍请参见本文档“IFTTT规则介绍”。

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。



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）

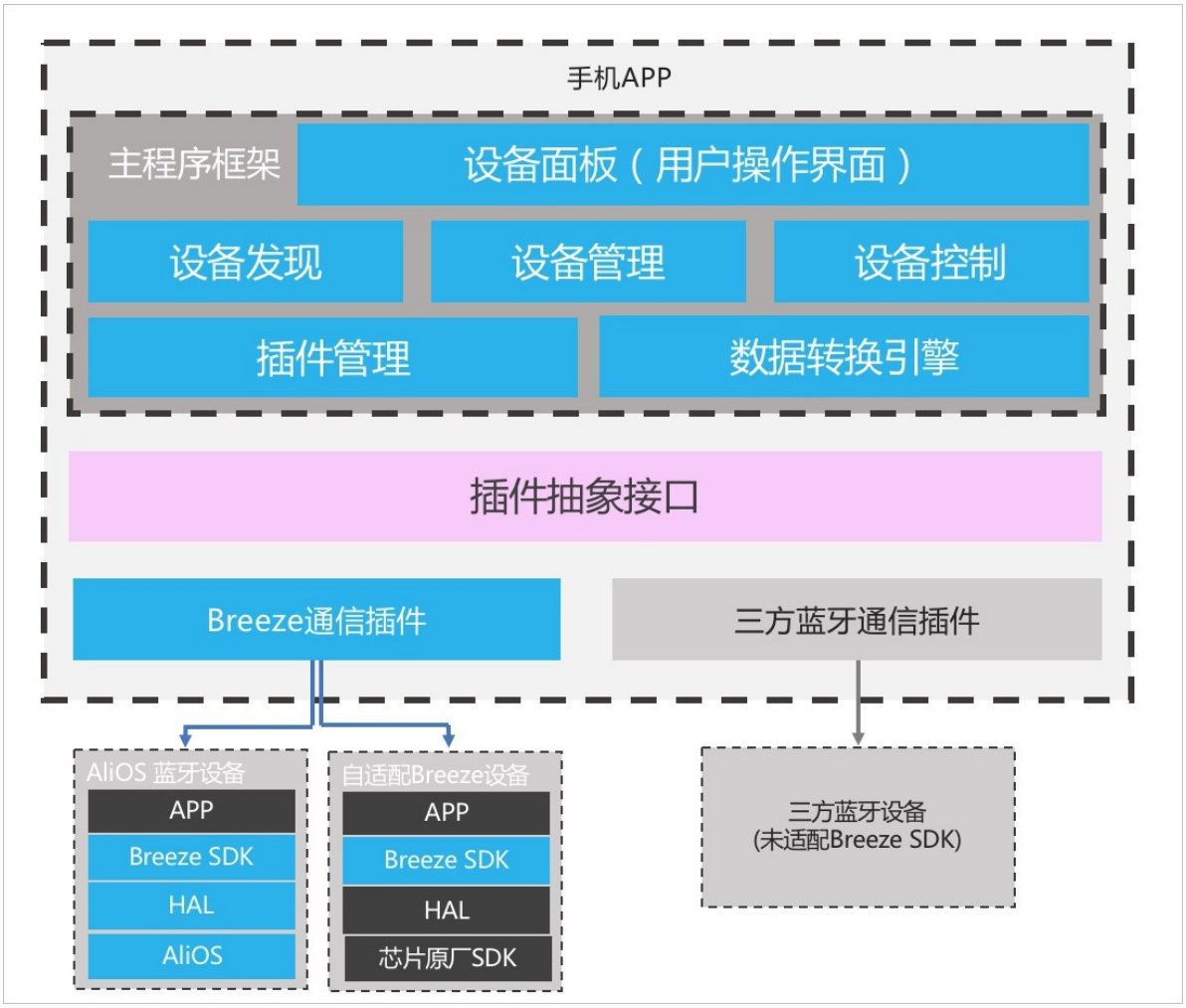
  该规范对蓝牙广播，服务，通信协议等进行了标准化，简称为蓝牙Breeze方案。蓝牙Breeze方案提供了手机端和设备端的SDK，主要包含设备管理，设备发现，加密通信，大数据传输等功能。并对蓝牙芯片和主流手机进行了严格的认证，保证了兼容性和稳定性。

  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，参见第三方蓝牙通信插件适配指南。



  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，请参见第三方蓝牙通信插件适配指南中的相关示例代码。

- 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

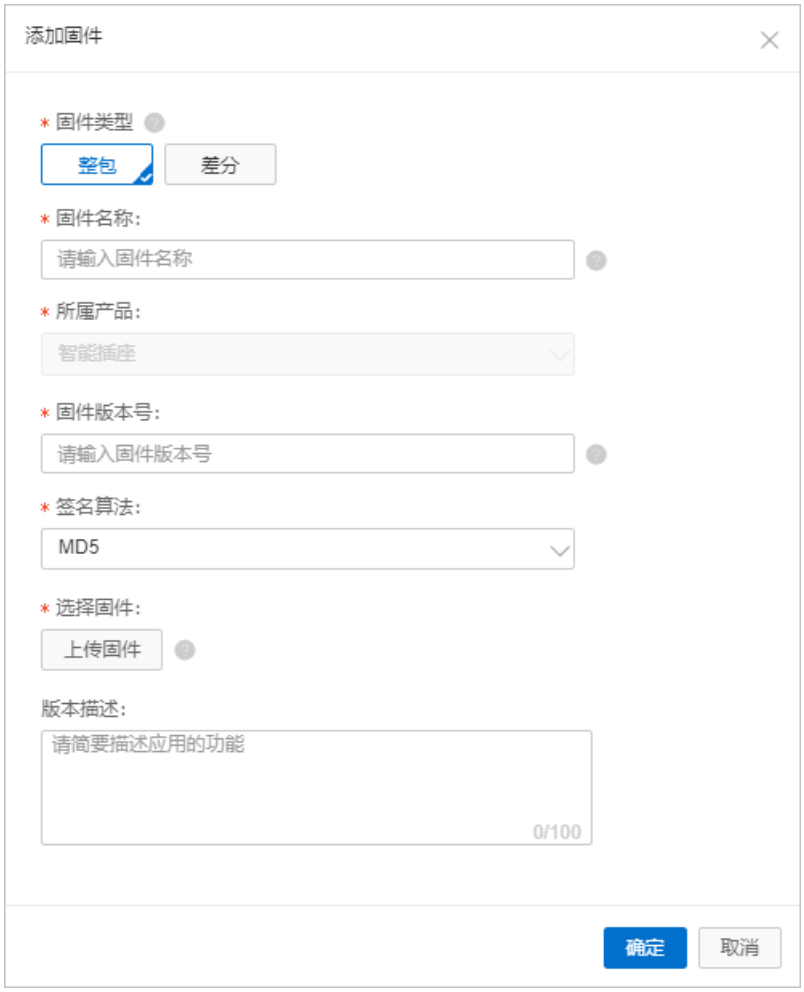
  蓝牙SDK下载勾选蓝牙设备接入框架套餐项，平台会自动生成SDK套餐包，下载集成开发即可。

  - 套餐包介绍：App SDK介绍
  - 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。

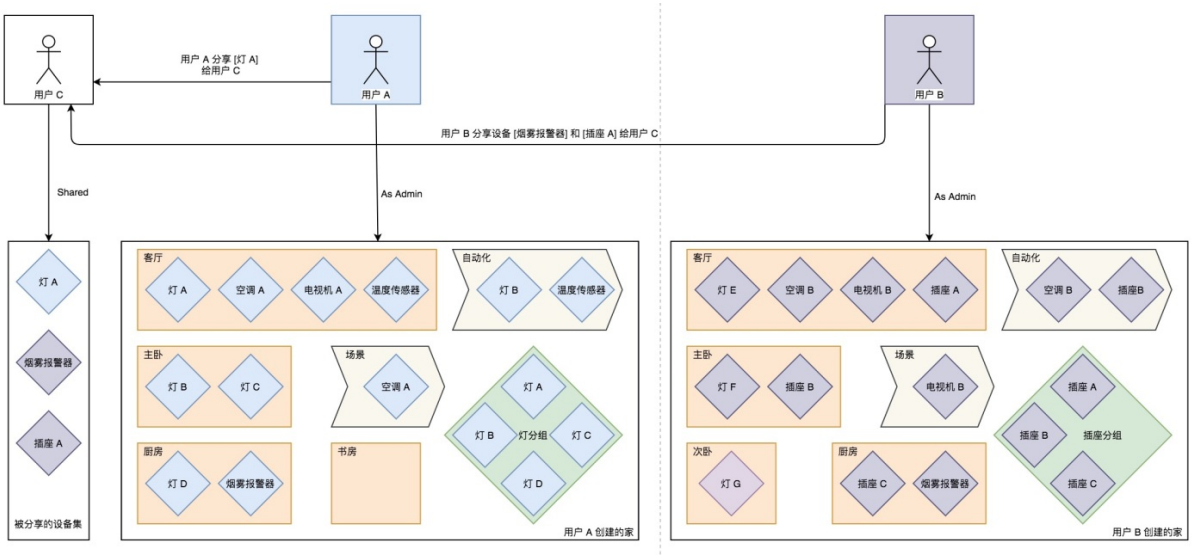

批量升级
* 待升级版本号:
请选择版本号
* APP 确认升级:
是
否
* 升级策略:
静态升级
* 升级范围:
请选择升级范围
* 升级时间:
立即升级
* 固件推送速率:
请输入每分钟推送的设备数
* 升级失败重试间隔:
不重试
设备升级超时时间 (分钟):
请输入超时时间（分钟）
本次批量升级共 0 个设备
确定
取消

7.家空间开发指南

生活物联网平台为自有品牌App提供了家空间功能，可以将App绑定的设备按照家庭、房间、设备组等进行空间分组。

功能概述

家空间功能的空间模型中，人、空间、设备三者之间的拓扑关系如下图所示。



家空间的设备规则

开发家空间服务时，您需要了解以下绑定设备的规则。

类别	规则说明
设备	• 终端用户首次创建家时，App会将原先存在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）。 • 被分享的设备属于用户维度的数据，不属于任何一个家。您可以使用 `/uc/listBindingByAccount` 接口来获取绑定的设备。
设备组	设备组是归属在同一款产品下的设备集合，可用于组控。 • 设备组功能目前仅支持灯和插座两个品类。 • 一个设备组内最少需要存在2个设备，最多存在100个设备。 • 一个设备只可以添加到一个设备组。 • 设备组的属性状态更新会触发组内所有设备的属性状态更新；组内单个设备的属性状态更新不会触发设备组属性状态的更新。 更新设备组状态的接口为 `/living/controlgroup/properties/set` 。 • 设备组属性状态与组内单个设备的属性状态存在不一致的情况。 获取设备组状态的接口为 `/living/controlgroup/properties/get` 。

API调用限制

生活物联网平台提供了家空间服务相关的一组API，为配合家空间的能力，对原有的部分API进行了升级改造，如下表所示。

- 如果您的App使用了家空间服务，请使用支持家空间服务的版本API，否则会返回28806的错误码。详细接口请参见配网服务和场景服务。
- 如果您的App不使用家空间的服务，请使用不支持家空间服务的历史版本API，否则会返回460错误码。详细接口介绍请参见历史版本配网服务和历史版本场景服务。

服务	API path	支持家空间服务的版本	不支持家空间服务的版本
配网服务	/awss/token/user/bind	1.0.8及以上	1.0.7及以下
配网服务	/awss/time/window/user/bind	1.0.8及以上	1.0.7及以下
配网服务	/uc/scanBindByShareQrCode	1.0.7及以上	1.0.6及以下
配网服务	/awss/enrollee/user/bind	1.0.3及以上	1.0.2及以下
配网服务	/awss/ble/user/bind	1.0.3及以上	1.0.2及以下
配网服务	/awss/gprs/user/bind	1.0.3及以上	1.0.2及以下
配网服务	/awss/subdevice/bind	1.0.3及以上	1.0.2及以下
场景服务	/living/scene/create	1.0.1及以上	1.0.0
场景服务	/scene/create	1.0.6及以上	1.0.5及以下
场景服务	/scene/info/get	1.0.6及以上	1.0.5及以下

8.数据统计开发指南

当您开发自有App需要统计数据时（例如插座的电量统计等），请您根据本文档提供的聚合方式来实现。

请您使用聚合数据查询接口，并结合以下聚合统计方式来开发相关业务场景。

- 按小时聚合统计
- 按天聚合统计
- 按月聚合统计
- 统计历史事件

按小时聚合统计

生活物联网平台支持按小时聚合统计属性或服务的数据，对应的表名称为：`ads_ilop_kv_state_hourly`，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`。

按天聚合统计

生活物联网平台支持按天聚合统计属性或服务的数据，对应的表名称为：`ads_ilop_kv_state_daily`，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`。

按月聚合统计

生活物联网平台支持按自然月聚合统计属性或服务的数据，对应的表名称为：`ads_ilop_kv_state_monthly`，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`。

统计历史事件

设备所有上报的历史事件数据的记录集合，对应的表名称为：`r_ilop_event_aggre_di`，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	事件上报的request_id。
iot_id	String	设备唯一身份标识。
event_code	String	事件的Identifier。
data_value	String	设备上报事件原始内容。
client_date	TimeStamp	设备端发送时间。
service_date	TimeStamp	服务端处理时间。

9.Android SDK 手册

9.1. SDK升级

如果您当前使用的App端SDK不是最新版本，建议您根据以下内容将SDK升级至最新版本。

概述

生活物联网平台发布的App端SDK最新版本为API Level 9。App端各版本SDK的区别如下（更多介绍请参见API Level版本介绍）。

类别	API Level 7及以下SDK	API Level 8 SDK	API Level 9 SDK
初始化	每个SDK分别初始化	统一初始化	统一初始化
安全图片	4张（分别对应原中国站与原国际站、原测试版与正式版）	2张（分别对应原中国站与原国际站）	1张（全球适用）
App在中国内地与内地以外地区切换时的操作	切换安全图片，并重启App	切换安全图片，并重启App	无需任何操作

我们给您提供了多种升级方案，请您根据实际情况来选择。

当前SDK集成情况	升级方案
未集成任何版本SDK	请直接使用最新版本的SDK，并对SDK进行初始化。此情况不涉及SDK升级。相关操作请参见下载并集成SDK和SDK初始化。
已集成API Level 8版本SDK	请根据**API Level 8升级SDK方案**来升级SDK，并在后续项目管理中注意以下内容（升级可能给您带来的影响请参见全球激活中心更新公告）。 • 如果您在原中国站与原国际站中都创建了项目，且其中一个项目没有出货或者出货量较少 升级后，建议您以出货量大的项目为主项目（即后续产品、App等在该项目中操作），将另一个项目的产品分享至主项目的App中，便于您日后只需维护一个项目中的App。跨项目分享产品的介绍请参见设置关联产品。 • 如果您在原中国站与原国际站中都创建了项目，且两个项目的出货量相当无法取舍 升级后，您可以通过手动修改安全图片后缀名，并调用接口切换安全图片，从而实现同时管理两个项目中的产品和App。
已集成API Level 7及以下版本SDK	我们提供了两种升级方案供您选择 • API Level 7升级SDK并使用统一初始化方案（推荐） 统一初始化方案对后期增加新功能、开拓海外市场等，具有更大的优势。因此，推荐您升级至最新版本的SDK，并使用统一初始化方案。 • API Level 7升级SDK但不用统一初始化方案（不推荐） 您也可以保留之前的初始化方法，仅更新SDK版本，但此时无法实现全球统一激活。该方案请慎重选择。

API Level 8升级SDK方案

1. 进入自有品牌App的**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 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。更多信息，请参见集成安全图片。

2. 使用统一初始化接口，并完成SDK的初始化。详细请参见SDK初始化。
3. （可选）添加原来SDK中的定制化逻辑，如用户账号的定制化UI等。
4. （可选）设置安全图片后缀名。 当您需要同时管理两个项目下的App时，您还需要根据以下步骤来设置安全图片后缀名。
   i. 复制并重命名安全图片名称，如命名为 `yw_1222_xxxyyy.jpg` 。
   ii. 增加安全图片调用和切换的业务逻辑。

   ```
   IoTSmart.setAuthCode(String authCode);
   //authCode为重命名的安全图片名称后缀名，即示例中的xxxyyy
   //authCode不设置或设置为空时，App默认加载名称为yw_1222_china_production.jpg的安全图片
   ```

   切换安全图片的时机和逻辑需要您自行实现，且切换安全图片后App必须重启才能生效。

API Level 7升级SDK并使用统一初始化方案（推荐）

1. 进入自有品牌App的**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 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。更多信息，请参见集成安全图片。

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SDK中的定制化逻辑，如用户账号的定制化UI等。

API Level 7升级SDK但不用统一初始化方案（不推荐）

> **说明** 该升级方案无法实现全球统一激活，请您慎重选择。

1. 进入自有品牌App的**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 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。更多信息，请参见集成安全图片。

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"
```

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

API Level 8及以上版本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API Level 9，详细请参见SDK升级。

统一初始化接口会根据您下载SDK时勾选的SDK配置项，一次性完成以下SDK的初始化。

- 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。详细操作请参见集成安全图片。

   > **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。详细操作请参见下载并集成SDK。
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
        .setRegionType(IoTSmart.REGION_ALL)
        // setProductEnv是API Level 8专用，API Level 9及以上版本使用IoTSmart.setProductScope来区分App是否操作未发布产品，且不再区分测试版与正式版，统一为正式版
        .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。 生活物联网平台的云端服务为多区域部署，根据您使用的SDK版本以及config.regionType参数的取值，来判断是否需要在初始化时设置国家。

参数取值	API Level 8及以下	API Level 9及以上
REGION_CHINA_ONLY	不需要	不需要

参数取值	API Level 8及以下	API Level 9及以上
REGION_ALL	需要 **说明** 此时如果您没有设置国家，SDK初始化流程会被暂停，且没法使用SDK的任何API。设置国家的操作请参见历史文档。	不需要（此时需在注册App账号时设置国家）

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。

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

初始化前需确保已集成安全图片，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，您需注意以下内容。
   - 使用API Level 9版本SDK时，不涉及切换安全图片，也不再需要重启App。
   - 使用API Level 8及以下版本SDK时，需要切换安全图片，并重启App后才能正确初始化。

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）、波兰文（pl-PL）、荷兰文（nl-NL）等十三种语言。

- 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);
```

获取App当前连接的服务器ID

当自有App可以连接多个业务服务器时，为了App最佳的体验效果，可以根据当前登录服务器来选择业务服务器。此时，您可以通过以下接口获取App当前连接的国家（即登录服务器ID），从而帮助您选择其他业务服务器最快连接的区域。

```
IoTSmart.getShortRegionId()
```

> 说明
>
> - 该接口须升级至0.1.7.1及以上版本才可见。升级代码如下。
>
>   ```
>   com.aliyun.iot.aep.sdk:sdk-framework:0.1.7.1
>   ```
>
> - 该接口必须要在App已登录状态下调用，否则没法获取准确的服务器ID。

平台返回服务器ID的值如下。

- 0：上海
- 1：新加坡
- 3：美国
- 4：德国

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。

9.4. 账号及用户SDK

账号和用户SDK提供了账号能力，包括注册、登录、登出、获取账号、会话管理、人机校验等功能，还提供了UI集成显示、UI定制等能力。

依赖SDK	概述
API通道	提供API通道能力

初始化

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

内置账号

内置账号为生活物联网平台提供的账号能力，您在客户端集成后即可使用。包括以下内容。

- 基础功能

  基础功能包括：注册、登录、退出登录、忘记密码、注销账号等。

  - 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();
```

- 切换OA语言

  Android App切换SDK语言时会同步切换OA的语言，您无需额外开发。切换SDK语言请参见通用SDK。

- 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源码请参见创建自有App。

- 自定义UI

  请参见定制Android App的OA UI。

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。

9.5. 身份认证SDK

提供基于IoT Token的用户身份认证方案，通过集成账号及用户SDK、API通道SDK，生成并管理用户身份凭证，以及鉴权发起API请求的用户身份。

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

初始化前需确保已集成安全图片，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).setIotTokenInvalidListener(IoTTokenInvalidListener listener
  )
  ```

  其中IoTTokenInvalidListener定义如下。

  ```
  public interface IoTTokenInvalidListener {
    void onIoTTokenInvalid();
  }
  ```

  > **说明** API全局只能设置一次，推荐在初始化完成之后，调用该API监听IoT Token失效的情况。当onIotTokenInvalid被触发时，需要提示C端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 ioTCredentialManageError) {
      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

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

初始化前需确保已集成安全图片，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Sch
eme.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,Objec
t> 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
     */
    if (data == null) {
      return;
    }
    JSONObject jsonObject = JSON.parseObject(data.toString());
    //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.pn
g",
        "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
  }
});
```

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

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

初始化前需确保已集成安全图片，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.OnBondBundleInfoGetListe
ner() {
        @Override
        public void onSuccess(BoneDevHelper.BoneBundleInfo boneBundleInfo) {
          BoneDevHelper.RouterInfo info = new BoneDevHelper().handleBundleInfo(MainActivity.this, bone
BundleInfo);
          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");
```

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。

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

初始化前需确保已集成安全图片，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 IMobileRequestListener() {
  @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 IMobileRequestListener() {
  @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 IMobileDownstreamListener() {
  @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.**{*;}
```

9.9. 配网SDK

提供了把Wi-Fi设备配置上家庭路由器以及局域网内已联网设备的发现能力，具体方案包括一键广播配网（P2P）、设备热点配网、蓝牙辅助配网、手机热点配网、设备间相互配网、二维码配网等。本配网SDK主要提供设备发现、设备配网、获取设备端token等能力。

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-biz SDK	提供蓝牙辅助配网支持，依赖breeze-biz SDK

引入方式

- Maven仓库地址

```
// 阿里云仓库地址，包括了阿里云IoT的SDK
maven {
  url "http://maven.aliyun.com/nexus/content/repositories/releases/"
}
```

- Gradle依赖

```
compile ('com.aliyun.alink.linksdk:ilop-devicecenter:1.7.7.1'){
  transitive = true
}
// 当您需要蓝牙辅助配网能力，还需添加以下依赖
// compile ('com.aliyun.alink.linksdk:breeze-biz:1.2.0')
```

- 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.**{*;}
```

初始化

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

设备发现

配网SDK提供了局域网内设备发现的能力，可发现的设备如下。

- 已配网设备
  - Wi-Fi设备
  - 以太网设备
- 进入配网模式的待配网设备
  - 设备热点待配网设备
  - 蓝牙辅助配网待配网设备
  - 通过已配设备发现局域网内的待配设备

设备发现相关API，请参见LocalDeviceMgr。

1. 启动设备发现。

   发现本地的已配网设备、零配发现的待配网设备、符合 `adh_{pk}_{mac}` 格式的设备热点设备或者combo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。如果您需要过滤发现的设备（如过滤已配网的设备、不支持的设备等），请参见本地发现设备列表信息过滤。

   ```
   // 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) {
       // 发现的设备类型
       // 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
     }
   });
   ```

2. 停止设备发现。

   停止发现本地已配网设备和待配网设备。调用该接口会清除已发现设备列表，确保与启动设备发现 `startDiscovery()` 成对调用。

   ```
   // 停止设备发现
   LocalDeviceMgr.getInstance().stopDiscovery();
   ```

设备配网

配网设备可以是本地发现的待配设备，也可以是通过扫码等其他途径获取的待配设备。更多配网方式请参见Android App Native开发配网。设备配置添加相关API说明，参见配网服务。

1. 设置待配网设备的信息。

```
// 启用全球配网时，ProvisionConfigParams的设置只需在配网前调用一次即可，无需每次配网前都调用
ProvisionConfigParams params = new ProvisionConfigParams();
params.enableGlobalCloudToken = true;
ProvisionConfigCenter.getInstance().setProvisionConfiguration(params);
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，蓝牙辅助配网时必配
deviceInfo.id= "xxx";// 设备热点的ID，在发现热点设备返回到App时会携带这个字段
// 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"; // 默认为一键配网
// 如果当前App需要全球使用，且涉及到切换账号的数据中心，配网SDK可以按照以下设置传递数据中心的信息
// getStoredShortRegionId接口由IoTSmart所在的SDK提供
RegionInfo regionInfo = new RegionInfo();
regionInfo.shortRegionId = Integer.parseInt(RegionManager.getStoredShortRegionId());
deviceInfo.regionInfo = regionInfo;
//设置待添加设备的基本信息
AddDeviceBiz.getInstance().setDevice(deviceinfo);
```

可能用到的其他参照信息如下。

- 指定配网方式为linkType：LinkType
- 设备信息相关实体类：DeviceInfo

2. 开始设备配网。

调用 `startAddDevice` 接口进入配网流程。

> **说明** 配网过程需要位置权限和位置服务（GPS），在调用开始配网接口之前，请确保App已获得相关权限。如使用手机热点方式配网，App还需额外获得WRITE_SETTINGS的权限。

```
// 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) {
```

```
    if (prepareType == 1) {
      // 一键配网、蓝牙辅助配网、设备热点配网、二维码配网、手机热点配网会走到prepareType==1的流程
      // 可以交互引导用户输入Wi-Fi的ssid、password信息，获取到ssid、password信息
      // 之后调用toggleProvision接口开始配网，参见第三步“输入账号密码”
    } else if (prepareType == 2) {
      // 手机热点配网时会走到该流程，收到这个回调后，先引导用户开启指定热点，例如ssid为aha，password为12345678
      // 确保热点开启后，可以交互引导用户输入Wi-Fi的ssid、password信息，获取到ssid、password信息
      // 之后调用toggleProvision接口开始配网，参见第三步“输入账号密码”
    }
  }
  @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Wi-Fi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Wi-Fi，
   // 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) {
    // 处理配网结果，如果配网成功后包含token，请使用配网成功带的token做绑定。
  }
});
```

配网回调接口，请参见 IAddDeviceListener。

3. 输入账号密码。

在收到 onProvisionPrepare 回调后，引导App用户输入Wi-Fi的ssid和password，并调用此方法传入ssid、password、timeout信息启动配网。

```
String ssid = "ssid"; // 手机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 password = "xxxxxxxx";
int timeout = 60; //单位秒，目前最短只能设置为60秒
AddDeviceBiz.getInstance().toggleProvision(ssid, password, timeout);
```

4. 停止配网。

```
// 停止配网
AddDeviceBiz.getInstance().stopAddDevice();
```

获取设备绑定Token

Wi-Fi设备或以太网设备绑定时，除了需要productKey、deviceName，还需要设备端的token。以下提供从设备端获取绑定token的方法。

> **说明** 以太网设备还需要先获取到设备的productKey、deviceName，可以使用 startDiscovery 接口去获取。

```
// 获取设备绑定的token
LocalDeviceMgr.getInstance().getDeviceToken(context, productKey, deviceName, 60*1000, 5*1000， new IO
nDeviceTokenGetListener() {
      @Override
      public void onSuccess(String token) {
        // TODO bind
      }
      @Override
      public void onFail(String reason) {
      }
    });
```

在插件中使用配网能力

如果需要在插件中使用配网能力，您需要向BoneMobile容器中注册配网API。请您在插件加载前，调用如下注册代码。

> **说明** 插件配网面板不支持蓝牙辅助配网，且设备热点配网不支持Android10及以上版本。

```
BonePluginRegistry.register("BoneAddDeviceBiz",BoneAddDeviceBiz.class);
BonePluginRegistry.register("BoneLocalDeviceMgr",BoneLocalDeviceMgr.class);
BonePluginRegistry.register("BoneHotspotHelper",BoneHotspotHelper.class);
```

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

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

使用方法

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> extraMa
p) {
    // TODO 处理推送通知
    Log.e("MyMessageReceiver", "Receive notification, title: " + title + ", summary: " + summary + ", extr
aMap: " + 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 + ", ex
traMap:" + extraMap);
  }
  @Override
  protected void onNotificationClickedWithNoAction(Context context, String title, String summary, Stri
ng extraMap) {
    Log.e("MyMessageReceiver", "onNotificationClickedWithNoAction, title: " + title + ", summary: " + su
mmary + ", extraMap:" + extraMap);
  }
  @Override
  protected void onNotificationReceivedInApp(Context context, String title, String summary, Map<Stri
ng, String> extraMap, int openType, String openActivity, String openUrl) {
    Log.e("MyMessageReceiver", "onNotificationReceivedInApp, title: " + title + ", summary: " + summar
y + ", 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androi
d：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>
```

在SDK初始化的代码逻辑里，已经封装了以下动作的业务逻辑，无需您做任何额外的操作。

- 关联移动推送到某账号

  当账号登录时，App会自动关联移动推送到当前账号。参见API服务 `/uc/bindPushChannel` 。

- 取消关联移动推送到某账号

  当账号登出时，App会自动取消关联移动推送到当前账号。参见API服务 `/uc/unbindPushChannel` 。

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。

消息类型说明

移动应用推送支持以下两种类型的消息下发。

- 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。

三方辅助推送通道

当App不在运行状态时，通过三方辅助推送通道可以将消息离线推送到手机。该推送通道无需额外初始化，您只需申请和配置好三方辅助的推送通道即可，详细参见移动应用推送开发指南。

目前支持三方辅助推送通道的手机有：小米、华为、VIVO 、OPPO、FCM。

- 小米、OPPO、FCM的三方推送：在初始化时配置参数，详细请参见SDK初始化。
- 华为、VIVO的三方推送：需要在*AndroidManifest.xml*配置如下相关信息。

> **说明** 测试华为离线推送功能时，须确保测试App的签名与提交至华为推送控制台的App签名保持一致。

```
<!-- huawei push start -->
<meta-data
  android:name="com.huawei.hms.client.appid"
  android:value="your huawei push appid" />
<!-- huawei push end -->
<!-- vivo push start -->
<meta-data
  android:name="com.vivo.push.api_key"
  android:value="your vivo push api_key" />
<meta-data
  android:name="com.vivo.push.app_id"
  android:value="your vivo push app id" />
<!-- vivo push start -->
```

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.**
```

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

初始化前需确保已集成安全图片，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表示具体的数据，失败时是一个AError结构，成功时忽略
```

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

  App上用户主动解绑一台设备或者在设备端reset，云端会主动向App发送通知。App收到推送通知后，SDK内部会自动清除相关缓存数据，且发出解绑通知。具体的解绑通知格式请参见调用示例。

```
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":"0300MSKL03xxxx4Sv4Za4",
   "productKey":"X5xxxxH7",
   "deviceName":"5gJtxDxxxxpisjX",
   "items":{
    "temperature":{
     "time":1510292697471,
     "value":30
    }
   }
 },
 "method":"thing.properties"
}
*/
//当operation为Unbind时，表示该设备已解绑，解绑通知的格式参考如下
//topic: /sys/${pk}/${dn}/app/down/_thing/event/notify
/*
{
  "identifier":"awss.BindNotify",
  "value":{
      "iotId":"apVtLzgkxxxxV000102",
      "identityId":"5063op37bxxxxxe0bfa9d98037",
      "owned":1,
      "productKey":"a2xxxxxyi",
      "deviceName":"IoT_Dev_33",
      "operation":"Unbind"
  }
}
*/
```

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。

依赖 SDK	概述
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（相关操作请参见下载并集成SDK）。

```
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 IDiscoveryFilter() {
      @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。

  i. 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:" + bundl
e);
          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
```

ii. 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").setP
ath("/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。

  i. 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
```

  ii. 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。

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DeviceInfo `BreezeHelper.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.IInquiryNewVersionCallback() {
            @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.IOtaL
istener() {
          @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();
```

9.13. Link Visual视频Media SDK

Link Visual App端SDK提供了音视频播放、语音对讲等功能。

依赖SDK	概述
API通道	提供API通道能力

依赖SDK	概述
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。详细请参见集成安全图片。

依赖引入

```
// 1. 在根目录下的build.gradle中添加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:1.2.13.2-ilop')
```

混淆配置

```
# keep and don't warn Link Visual
-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	✓	✓	✓
循环播放	x	x	x	✓
单帧步进	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
硬解码	✓	✓	✓	✓
数字变焦	✓	✓	✓	✓
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云存录像播放器（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();
...
```

- HLS云存录像播放器（ExoPlayer）推荐使用自研播放器

```
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());
```

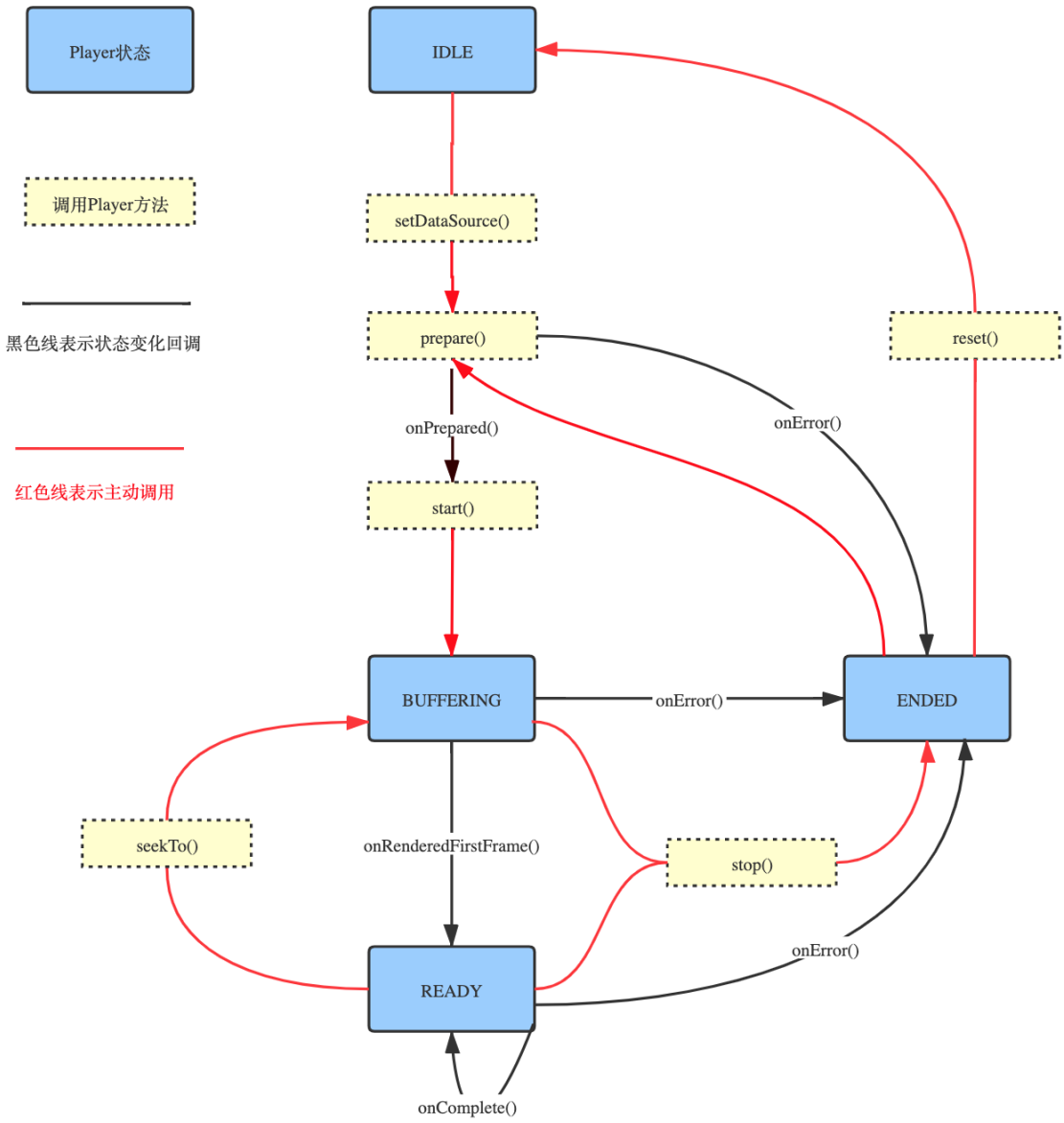
```
    }
});
// 设置状态监听
player.setOnPlayerStateChangedListener(new OnPlayerStateChangedListener() {
    @Override
    public void onPlayerStateChange(int playerState) {
        switch (playerState) {
            case Player.STATE_BUFFERING:
                break;
            case Player.STATE_IDLE:
                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();
```

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

- LinkVisualMedia

  > 说明 LinkVisualMedia用于实现预建连，若无预建连优化直播首帧出图速度需求，可忽略

  - 获取单例方法

    ```
    LinkVisualMedia getInstance();
    ```

  - 初始化方法

    ```
    /**
     * 全局资源初始化方法，SDK内部自动完成初始化，无需再调用
     * @Deprecated
     */
    void init();
    ```

  - 为IPC设备请求播放预建连

    ```
    /**
     * 尝试与IPC设备进行预建连，一旦预建连成功会加快直播的出图速度，需在播放前提前调用
     * @param 已经在线的IPC设备的iotId
     */
    void preConnectByIotId(String iotId);
    ```

- 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（仅适配Link Visual云提供的RTMP地址）
 * @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 Illega
lArgumentException;
```

- 设置IPC直播数据源，该接口已开启加密和强制I帧

```
/**
 * 设置IPC直播数据源，该接口已开启加密和强制I帧
 * @param iotId        IPC设备的iotId
 * @param streamType     流的类型，若有多路码流请关注该参数，C.STREAM_TYPE_MAJOR表示主码流；C
.STREAM_TYPE_MINOR表示辅码流
 */
 void setIPCLiveDataSource(String iotId, int streamType);
```

- 设置IPC直播数据源，该接口已开启加密和强制I帧

```
/**
 * 设置IPC直播数据源，该接口已开启加密和强制I帧
 * @param iotId        IPC设备的iotId
 * @param streamType     流的类型，若有多路码流请关注该参数，C.STREAM_TYPE_MAJOR表示主码流；C
.STREAM_TYPE_MINOR表示辅码流
 * @param cacheDurationInMs 云端缓存的视频长度，若有数据，该数据将会被加速快放，单位ms，该值建议
不超过1个GOP
 *
 void setIPCLiveDataSource(String iotId, int streamType, int cacheDurationInMs);
```

- 设置IPC直播数据源

```
/**
 * 设置IPC直播数据源.
 *
 * @param iotId         设备的iotId
 * @param streamType     流的类型，若有多路码流请关注该参数，C.STREAM_TYPE_MAJOR表示主码流；C.STREAM_TYPE_MINOR表示辅码流
 * @param relayEncrypted   云转是否加密，建议开启
 * @param relayEncryptType 云转加密类型
 * @param forceIFrame     是否需要强制I帧，建议开启
 * @param cacheDurationInMs 云端缓存的视频长度，若有数据，该数据将会被加速快放，单位ms，该值建议不超过1个GOP
 */
 void setIPCLiveDataSource(String iotId, int streamType, boolean relayEncrypted, int relayEncryptType, boolean forceIFrame, int cacheDurationInMs)
```

- 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_STREAM_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 取值范围：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_SCALING_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_FIT_WITH_CROPPING}
 */
void setVideoScalingMode(int videoScalingMode);
```

- 设置软硬解策略

```
/**
 * 设置解码器策略，全局生效
 * 默认使用强制软解
 *
 * @param decoderStrategy
 *     HARDWARE_FIRST - 硬解优先
 *     FORCE_SOFTWARE - 强制软解
 */
void setDecoderStrategy(DecoderStrategy decoderStrategy)
```

- 获取当前播放的软硬解类型

```
/**
 * 获取当前流的解码器类型，软解或硬解
 *
 * @return 若当前未开流，则返回null
 *     HARDWARE - 硬解码
 *     SOFTWARE - 软解码
 */
DecoderType getDecoderType()
```

- 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 playerStoppedDrawingMode)
```

- 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，取值范围：0帧~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);
```

- 设置播放器抗抖动最大缓冲区时长

```
/**
 * 设置播放器抗抖动最大缓冲区时长，默认1000ms，范围200-3000ms
 * @param jitterBufferSizeInMs
 */
void setMaxJitterBufferSizeInMs(int jitterBufferSizeInMs);
```

- 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

> **说明** 该模式为高级模式，一旦开启后，播放器内部将不再进行画面渲染，需要自行设置对YUV数据变化的监听，并及时通过YUV数据获取接口取出暂存的YUV数据帧立刻渲染

```
/**
 * 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 onExternalRenderListener);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 取值范围：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（仅适配Link Visual云提供的RTMP地址）
 * @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 Illega
lArgumentException;
```

- 设置播放源为IPC录像（按录像文件名），该接口已开启加密

```
/**
 * 设置播放源为IPC录像（按录像文件名），该接口已开启加密
 *
 * @param iotId      设备iotId
 * @param fileName    录像文件名
 */
 void setDataSourceByIPCRecordFileName(String iotId, String fileName);
```

- 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
 * 该接口已开启加密
 * see {@link #setDataSourceByIPCRecordTime(String, int, int, boolean, int, long)}
 *
 * @param iotId          设备iotId
 * @param beginTimeInS     录像开始时间，1970年1月1日开始的秒数
 * @param endTimeInS       录像结束时间，1970年1月1日开始的秒数
 * @param seekToPositionInMs 相对于beginTime的播放偏移量，单位为ms
 */
void setDataSourceByIPCRecordTime(String iotId, int beginTimeInS, int endTimeInS, long seekToPos
itionInMs)
/**
 * 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
 * 该接口已开启加密
 * see {@link #setDataSourceByIPCRecordTime(String, int, int, boolean, int, long)}
 *
 * @param iotId          设备iotId
 * @param beginTimeInS     录像开始时间，1970年1月1日开始的秒数
 * @param endTimeInS       录像结束时间，1970年1月1日开始的秒数
 * @param seekToPositionInMs 相对于beginTime的播放偏移量，单位为ms
 * @param recordType       录像类型： 0（表示计划录像）；1（表示报警录像）
 */
void setDataSourceByIPCRecordTime(String iotId, int beginTimeInS, int endTimeInS, long seekToPos
itionInMs, int recordType)
/**
 * 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
 * @param recordType       录像类型： 0（表示计划录像）；1（表示报警录像）
 * 出于安全要求，无特殊情况，请尽量开启加密
 */
void setDataSourceByIPCRecordTime(String iotId, int beginTimeInS, int endTimeInS, boolean encrypt
ed, int encryptType, long seekToPositionInMs, int recordType)
```

- 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);
```

- 逐帧播放

```
/**
 * 逐帧播放，恢复请调用start()
 * @return 是否调用成功
 */
boolean playFrameByFrame();
```

- 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 取值范围：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_SCALING_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_FIT_WITH_CROPPING}
 */
void setVideoScalingMode(int videoScalingMode);
```

- 设置软硬解策略

```
/**
 * 设置解码器策略，全局生效
 * 默认使用强制软解
 *
 * @param decoderStrategy
 *     HARDWARE_FIRST - 硬解优先
 *     FORCE_SOFTWARE - 强制软解
 */
void setDecoderStrategy(DecoderStrategy decoderStrategy)
```

- 获取当前播放的软硬解类型

```
/**
 * 获取当前流的解码器类型，软解或硬解
 *
 * @return 若当前未开流，则返回null
 *     HARDWARE - 硬解码
 *     SOFTWARE - 软解码
 */
DecoderType getDecoderType()
```

- 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 playerStoppedDrawingMode)
```

- 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 onExternalRenderListener);public inte
rface 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，需要设备支持倍速功能
   * 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 取值范围：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);
```

- 逐帧播放

```
  /**
   * 逐帧播放，恢复请调用start()
   * @return 是否调用成功
   */
   boolean playFrameByFrame();
```

- 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表示文件保存成功
 */
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 取值范围：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_SCALING_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_FIT_WITH_CROPPING}
 */
void setVideoScalingMode(int videoScalingMode);
```

- 设置软硬解策略

```
/**
 * 设置解码器策略，全局生效
 * 默认使用强制软解
 *
 * @param decoderStrategy
 *     HARDWARE_FIRST - 硬解优先
 *     FORCE_SOFTWARE - 强制软解
 */
void setDecoderStrategy(DecoderStrategy decoderStrategy)
```

- 获取当前播放的软硬解类型

```
/**
 * 获取当前流的解码器类型，软解或硬解
 *
 * @return 若当前未开流，则返回null
 *     HARDWARE - 硬解码
 *     SOFTWARE - 软解码
 */
DecoderType getDecoderType()
```

- 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 playerStoppedDrawingMode)
```

- 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 onExternalRenderListener);public inte
rface 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 取值范围：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，取值范围：(0,2]，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 取值范围：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_SCALING_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_FIT_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 取值范围：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（若已连接蓝牙耳机或者线控耳机则路由到耳机上）；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 A
udioParams 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)
```

- 设置是否开启对蓝牙耳机和线控耳机的支持

```
/**
 * 设置是否开启对蓝牙和有线耳机外设的支持，开启后优先使用外设进行录放音，默认支持
 * @param supportExternalHeadset 是否支持
 */
void setSupportExternalHeadset(boolean supportExternalHeadset);
```

- 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）
LiveIntercomException.START_LIVE_INTERCOM_REQUEST_FAILED	启动语音对讲失败

错误枚举	描述
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

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。详细操作请参见集成安全图片。

```
//设置图片后缀
ConfigManager.getInstance().setSecGuardImagePostfix("xxxx"); //xxxx为安全图片的后缀名
```

- 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_1513
412765/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.SecurityGuar
dMainPlugin.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(Unkn
own Source)
12-16 16:26:11.431 10486-10486/? W/System.err:    at com.alibaba.wireless.security.open.initialize.a.loadLibr
arySync(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(Act
ivityThread.java:5403)
12-16 16:26:11.431 10486-10486/? W/System.err:    at android.app.ActivityThread.-wrap2(ActivityThread.jav
a)
12-16 16:26:11.431 10486-10486/? W/System.err:    at android.app.ActivityThread$H.handleMessage(Activity
Thread.java:1545)
12-16 16:26:11.431 10486-10486/? W/System.err:    at android.os.Handler.dispatchMessage(Handler.java:102)
12-16 16:26:11.431 10486-10486/? W/System.err:    at android.os.Looper.loop(Looper.java:154)
12-16 16:26:11.431 10486-10486/? W/System.err:    at android.app.ActivityThread.main(ActivityThread.java:6
119)
12-16 16:26:11.431 10486-10486/? W/System.err:    at java.lang.reflect.Method.invoke(Native Method)
12-16 16:26:11.431 10486-10486/? W/System.err:    at com.android.internal.os.ZygoteInit$MethodAndArgsCa
ller.run(ZygoteInit.java:886)
12-16 16:26:11.431 10486-10486/? W/System.err:    at com.android.internal.os.ZygoteInit.main(ZygoteInit.ja
va: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.cloudap
i.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*是否被打包到APK包的*build/outputs/apk/debug*目录下，且确保*lib/armeab*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.m
```

```
ain(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.dispatchMessage(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。
- 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_` 开头的图片大小为0。

```
release {
        minifyEnabled true // 是否混淆
        shrinkResources true //会导致安全图片大小为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,@drawable/yw_1222_china_producti
on"
```

- 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
    }
  }
}
```

10.iOS SDK手册

10.1. SDK升级

如果您当前使用的App端SDK不是最新版本，建议您根据以下内容将SDK升级至最新版本。

概述

生活物联网平台发布的App端SDK最新版本为API Level 9。App端各版本SDK的区别如下（更多介绍请参见API Level版本介绍）。

类别	API Level 7及以下SDK	API Level 8 SDK	API Level 9 SDK
初始化	每个SDK分别初始化	统一初始化	统一初始化
安全图片	4张（分别对应原中国站与原国际站、原测试版与正式版）	2张（分别对应原中国站与原国际站）	1张（全球适用）
App在中国内地与内地以外地区切换时的操作	切换安全图片，并重启App	切换安全图片，并重启App	无需任何操作

我们给您提供了多种升级方案，请您根据实际情况来选择。

当前SDK集成情况	升级方案
未集成任何版本SDK	请直接使用最新版本的SDK，并对SDK进行初始化。此情况不涉及SDK升级。相关操作请参见下载并集成SDK和SDK初始化。
已集成API Level 8版本SDK	请根据**API Level 8升级SDK方案**来升级SDK，并在后续项目管理中注意以下内容（升级可能给您带来的影响请参见全球激活中心更新公告）。 • 如果您在原中国站与原国际站中都创建了项目，且其中一个项目没有出货或者出货量较少 升级后，建议您以出货量大的项目为主项目（即后续产品、App等在该项目中操作），将另一个项目的产品分享至主项目的App中，便于您日后只需维护一个项目中的App。跨项目分享产品的介绍请参见设置关联产品。 • 如果您在原中国站与原国际站中都创建了项目，且两个项目的出货量相当无法取舍 升级后，您可以通过手动修改安全图片后缀名，并调用接口切换安全图片，从而实现同时管理两个项目中的产品和App。
已集成API Level 7及以下版本SDK	我们提供了两种升级方案供您选择 • API Level 7升级SDK并使用统一初始化方案（推荐） 统一初始化方案对后期增加新功能、开拓海外市场等，具有更大的优势。因此，推荐您升级至最新版本的SDK，并使用统一初始化方案。 • API Level 7升级SDK但不用统一初始化方案（不推荐） 您也可以保留之前的初始化方法，仅更新SDK版本，但此时无法实现全球统一激活。该方案请慎重选择。

API Level 8升级SDK方案

1. 进入自有品牌App**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 将下载后的压缩包解压，得到的文件夹中包含安全图片和*Podfile*文件。
2. 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。
3. （可选）添加原来的定制化逻辑，如用户账号的定制化UI等。
4. （可选）设置安全图片后缀名。 当您需要同时管理两个项目下创建的App时，请通过设置不同的安全图片后缀名来实现。
   i. 复制并重命名安全图片名称，如命名为 `yw_1222_xxxyyy.jpg` 。
   ii. 增加安全图片调用和切换的业务逻辑。

```
[[IMSIotSmart sharedInstance] setAuthCode:@"xxxyyy"]
//xxxyyy为重命名的安全图片名称后缀名
//authCode不设置或设置为空时，App默认加载名称为yw_1222_china_production.jpg的安全图片
```

   切换安全图片的时机和逻辑需要您自行实现，且切换安全图片后App必须重启才能生效。

API Level 7升级SDK并使用统一初始化方案（推荐）

1. 进入自有品牌App**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 将下载后的压缩包解压，得到的文件夹中包含安全图片和*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。

API Level 7升级SDK但不用统一初始化方案（不推荐）

> **说明** 该升级方案无法实现全球统一激活，请您慎重选择。

1. 进入自有品牌App**SDK和插件**页面，选择最新的API Level 9，并下载新的SDK套餐项。详细操作请参见下载并集成SDK。 将下载后的压缩包解压，得到的文件夹中包含安全图片和*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。

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

API Level 8及以上版本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API Level 9，详细请参见SDK升级。

统一初始化接口会根据您下载SDK时勾选的SDK配置项，一次性完成以下SDK的初始化。

- 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。详细操作请参见集成安全图片。

   > **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。详细操作请参见下载并集成SDK。
3. 初始化SDK。
   i. 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。

   ii. 设置安全图片。 安全图片名称为： `yw_1222_china_production.jpg` ，您可以通过如下方法设置。

   ```
   [[IMSIotSmart sharedInstance] setAuthCode:@"china_production"];
   ```

   iii. 启动初始化。

```
#import <IMSIotSmart/IMSIotSmart.h>
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  IMSIotSmartConfig *config = [IMSIotSmartConfig new];
  config.regionType = REGION_ALL; //取值范围参见枚举类型 `IMSRegionType`
  [IMSIotSmart sharedInstance].config = config;
  // 设置安全图片。如果不设置，默认使用china_production
  [[IMSIotSmart sharedInstance] setAuthCode:@"china_production"];
  [[IMSIotSmart sharedInstance] application:application didFinishLaunchingWithOptions: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 didReceiveRemoteNotification:userInfo];
}
- (void)application:(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken:(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 didRegisterForRemoteNotificationsWithDeviceToken:deviceToken];
}
- (void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError:(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 didFailToRegisterForRemoteNotificationsWithError:error];
}
```

iv. 设置国家。 生活物联网平台的云端服务为多区域部署，根据您使用的SDK版本以及config.regionType参数的取值，来判断是否需要在初始化时设置国家。

参数取值	API Level 8及以下	API Level 9及以上
REGION_CHINA_ONLY	不需要	不需要
REGION_ALL	需要 **说明** 此时如果您没有设置国家，SDK初始化流程会被暂停，且没法使用SDK的任何API。设置国家的操作请参见历史文档。	不需要（此时需在注册App账号时设置国家）

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。

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

初始化前需确保已集成安全图片，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>
//基于API Level 9及以上版本SDK开发的App，切换国家时不需要重启App，即IMSIotSmart内部不需要实现重启
逻辑
//如果您没有其他业务依赖该逻辑，可忽略needRestartApp参数的配置
[[IMSIotSmart sharedInstance] setCountry:_selectCountry callback:^(BOOL needRestartApp) {


 }];
```

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）、波兰文（pl-PL）、荷兰文（nl-NL）等十三种语言。

- 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。

获取App当前连接的服务器ID

当自有App可以连接多个业务服务器时，为了App最佳的体验效果，可以根据当前登录服务器来选择业务服务器。此时，您可以通过以下接口获取App当前连接的国家（即登录服务器ID），从而帮助您选择其他业务服务器最快连接的区域。

```
[[IMSIotSmart sharedInstance] shortRegionId];
```

> **说明**
> - 该接口须升级至1.2.2及以上版本才可见。升级代码如下。
>
> ```
> pod 'IMSIotSmart', '1.2.2'
> ```
>
> - 该接口必须要在App已登录状态下调用，否则没法获取准确的服务器ID。

平台返回服务器ID的值如下。

- 0：上海
- 1：新加坡
- 3：美国
- 4：德国

SDK API Reference

使用生活物联网平台提供的SDK时，SDK API的相关注释请参见SDK API Reference。

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Oauth 2.0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

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

> **说明** SDK使用Pod集成完成依赖添加后，还需要将ALBBOpenAccountUI.framework中的*xib*目录放置到主工程目录下（每次升级SDK后需要执行该操作）。

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(ALBBOpenAccountUIService);
// 显示手机注册窗口，presentingViewController将通过present方式展示登录界面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方式展示登录界面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(ALBBOpenAccountUIService);
// 显示登录窗口，presentingViewController将通过present方式展示登录界面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"} Callb
ack:^(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:^(NSErr
or *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*文件中自行调整UI即可。

  - 自定义方式（简单集成）

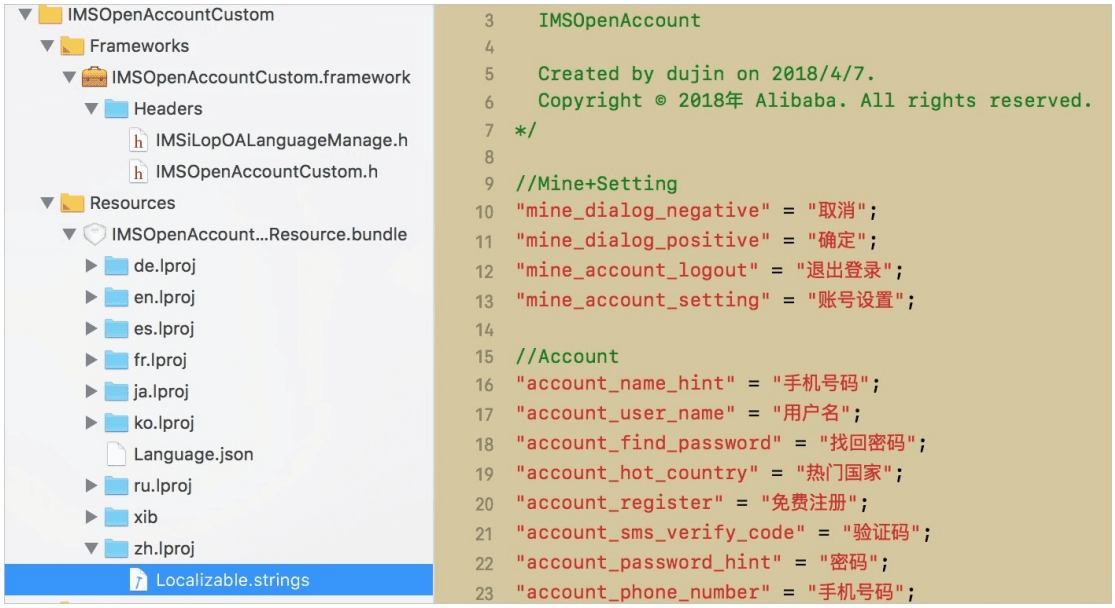
    a. 集成以下依赖文件

    ```
    pod 'IMSOpenAccountCustom', '1.1.5'
    pod 'AlicloudALBBOpenAccount', '3.4.0.46'
    ```

    b. 将IMSOpenAccountCustom的IMSOpenAccountCustomResource.bundle下的xib放到主工程。

c. 设置语言，只需要指定前缀即可。例如，中文设置为 `zh` 。

OA模块可设置的语言种类与SDK相同，详细请参见通用SDK。




```
// 使用默认的翻译资源文件
  [[IMSiLopOALanguageManage shareInstance]
  setOpenAccountModuleLanguageWithLanguagePrefix:@"ko"];
// 使用自己的资源文件
//拷贝IMSOpenAccountCustom下IMSOpenAccountCustomResource.bundle中的.lproj文件，整理一份相同key的多国语言翻译
  [[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，支持的语言种类请参见通用SDK
[[ALBBOpenAccountSDK sharedInstance] setRpcLocale:locale];
//二、修改xib上显示语言，默认显示中文
//1、拷贝ALBBOpenAccountUI下ALBBOpenAccount.bundle中的.lproj文件内容，整理一份相同key的多国语言翻译
//2、设置显示的国际化文件
[[ALBBOpenAccountSDK sharedInstance] setLocale:@"zh.lproj"];
//3、指定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 bundleWithPath: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(ALBBOpenAccountUIService);
[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

  请参见定制iOS App的OA UI。

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Oauth 2.0的AuthCode，并调用ALBBOpenAccountSSOService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))co
mpletionHandler {
  //自有账号登录并通过Oauth 2.0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(ALBBOpenAccountSSOService);
      [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会监听该通知并创建和管理用户身份凭证
    NSString *loginNotificationName = [[IMSAccountService sharedService].sessionProvider accountDi
dLoginSuccessNotificationName];
    [[NSNotificationCenter defaultCenter] postNotificationName:loginNotificationName object:nil];
  } else {
    //处理登录失败
  }
}
@end
```

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

初始化前需确保已集成安全图片，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 para
ms: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, IMSRes
ponse * _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

如果您创建自有App时配置为不支持多端登录（详细介绍请参见创建自有App），当一个账号在多个设备登录时，只有最后登录的终端可以正常访问IoT服务，其他账号在发送API通道请求时会返回认证错误（错误码为401）。此外，如果账号未登录或登录信息过期（或长时间未登录）时，在发送API通道请求时也会返回认证错误。

出现以上两种认证错误时，您只需处理API请求的错误，并提示终端用户重新登录即可。

```
[IMSRequestClient asyncSendRequest:builder.build responseHandler:^(NSError * _Nullable error, IMSRes
ponse * _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。

i. 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, IMSRequestPayload * _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))co
mpletionHandler {
  [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
```

ii. 注册自定义的身份认证。

```
XXCustomAuthentication *iotAuthDelegate = [[XXCustomAuthentication alloc] initWithCredentialMa
nager:IMSCredentialManager.sharedManager];
[IMSRequestClient registerDelegate:iotAuthDelegate forAuthenticationType:IMSAuthenticationType
IoT];
```

- 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, IMSCreden
tial * _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;
  }
}];
```

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

初始化前需确保已集成安全图片，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, IMSRespo
nse * _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
    }
  }
}];
```

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, IMSRespo
nse * _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
    }
  }
}];
```

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

初始化前需确保已集成安全图片，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:NSJSONReadingMutableLeav
es 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:@{} respHandler:^(LKAppExpResp
onse * _Nonnull response) {
    if (![response successed]) {
      IMSLifeLogVerbose(@"解绑长连接推送失败");
    }
  }];
```

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

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

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, NSDictionary *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?platf
orm=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:NSJSONR
eadingAllowFragments 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"];
```

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");
```

10.10. 配网SDK

提供了把Wi-Fi设备配置上家庭路由器以及局域网内已联网设备的发现能力，具体方案包括一键广播配网、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

初始化前需确保已集成安全图片，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Wi-Fi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Wi-Fi设备；1代表ethernet设备（网线连接）；2代表路由器
    // 例如@"ble_subtype_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;
...
// 针对设备端SDK V1.6.0版本以上固件，配网SDK需升级至1.12.0及以上版本，并指定数据中心的信息
model.enableGlobalCloudToken = YES;
model.regionNode = LKRegionNodeCustomID;
model.customRegionID = [[ALBBOpenAccountSDK sharedInstance] getRegionModel].shortRegionId ?:
@"0";
[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
productKey	NSString	是	设备的ProductKey
deviceName	NSString	否	设备名称
productId	NSString	否	待配设备产品ID，蓝牙辅助配网时为必选参数
linkType	assign	否	指定配网方式 ■ ForceAliLinkTypeBroadcast：一键广播配网 ■ ForceAliLinkTypeHotspot：手机热点配网 ■ ForceAliLinkTypeSoftap：设备热点配网 ■ ForceAliLinkTypeQR：摄像头扫码配网 ■ ForceAliLinkTypeBLE：蓝牙辅助配网

- 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...) ");
}
/**
 （可选）手机热点配网状态回调，提示用户关闭热点并切回以前的Wi-Fi
 */
- (void)notifyProvisioningNotice{
  NSLog(@"notifyProvisioningNotice");
}
/*
 （可选）设备热点状态回调
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);
}
/**
（可选）摄像头二维码配网模式相关回调
@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Wi-Fi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Wi-Fi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Wi-Fi
    NSLog(@"请立即切换回开启热点之前的Wi-Fi网络");
  }
}
```

- 配网结果监听

```
- (void)notifyProvisionResult:(IMLCandDeviceModel *)candDeviceModel withProvisionError:(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)");
  }
}];
/**
针对设备端SDK 1.6.0及以上版本的SDK，App配网开启了enableGlobalCloudToken功能时，调用以下接口获取绑
定信息。
      resultInfo 包括：
        - BOOL isBind：是否云端Token已经发起绑定，如果是，则通过iotID、error字段查看绑定结果；如果否，
则查看bindToken字段做后续绑定
        - NSString bindToken：可发起后续绑定的Token
        - NSString iotID：通过云端Token已完成绑定的设备ID
        - NSString pageRouterUrl：通过云端Token已完成绑定的设备面板URL，可忽略
        - NSError error：通过云端Token设备绑定失败
        }
*/
[[IMLLocalDeviceMgr sharedMgr] getDeviceTokenAndBindResult:self.productKey
                                    deviceName:self.deviceName
                                     timeout:30
                                    interval:3
                                   resultBlock:^(NSDictionary *resultInfo, BOOL boolSuccess) {
        NSLog(@"获取设备resultInfo结果：%@，boolSuccess：%d", resultInfo, boolSuccess);
      }];
```

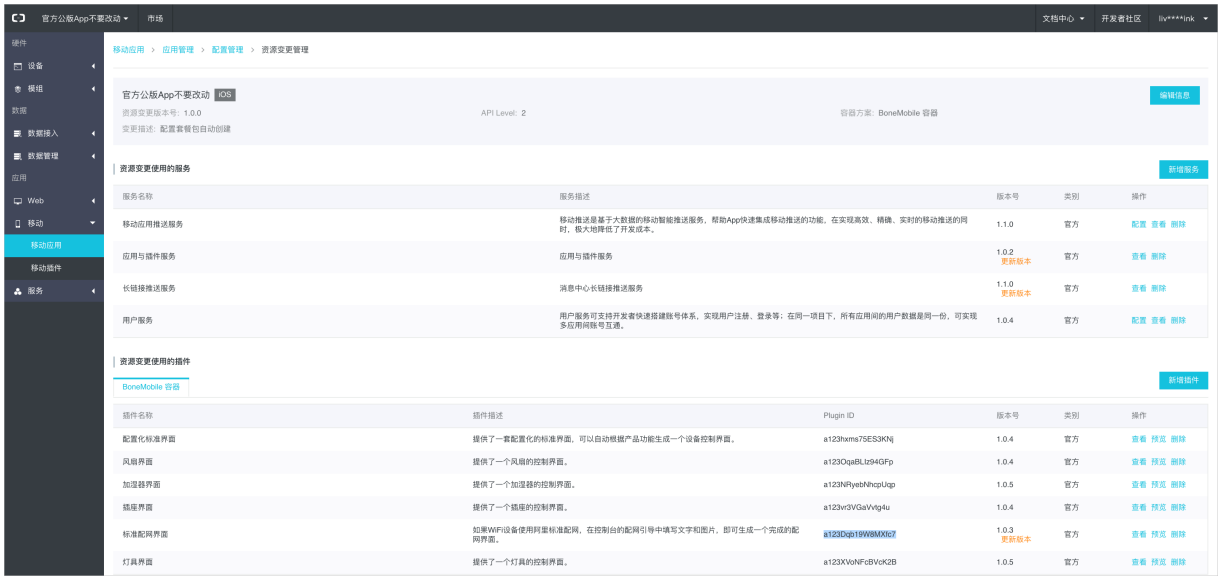
使用配网插件

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。



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。

> **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

初始化前需确保已集成安全图片，初始化的操作请参见SDK初始化。

使用方法

1. 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。
2. 监听推送通知。

```
/*
 * 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"];
  NSLog(@"content = [%@], badge = [%ld], sound = [%@]", content, (long)badge, sound);
  // iOS badge 清0
  application.applicationIconBadgeNumber = 0;
  // 通知打开回执上报
  [[IMSIotSmart sharedInstance] application:application didReceiveRemoteNotification:userInfo];
}
```

在SDK初始化的代码逻辑里，已经封装了以下动作的业务逻辑，无需您做任何额外的操作。

- 关联移动推送到某账号

  当账号登录时，App会自动关联移动推送到当前账号。参见API服务 `/uc/bindPushChannel` 。

- 取消关联移动推送到某账号

  当账号登出时，App会自动取消关联移动推送到当前账号。参见API服务 `/uc/unbindPushChannel` 。

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。

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。

依赖 SDK	概述
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

初始化前需确保已集成安全图片，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 getLocalAuthedDeviceDataList];
//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];
```

- 设备解绑

  App上用户主动解绑一台设备或者在设备端reset，云端会主动向App发送通知。App收到推送通知后，SDK内部会自动清除相关缓存数据，且发出onDeviceUnbind通知。

```
/**
设备被解绑通知，具体参见`IMSThingObserver`
 @param iotId：物的iotId
 @param params：云端推送下来的原始数据
 */
- (void)onDeviceUnbind:(NSString *)iotId params:(NSDictionary *)params;
```

- 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 n
ame]]];
                                        });
                                  } else {
                                    dispatch_async(dispatch_get_main_queue(), ^{
                                        [UiUtils showTip:[NSString stringWithFormat:@"调用服务 %@失败", [service n
ame]]];
                                        });
                                    } }];
```

- 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, NSUIntege
r 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
  i. 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];
  ```

  ii. 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];
  ```

  iii. 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。

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
  i. 预检查事件（LKOTANotificationTypeCheck）
  ii. 下载升级包事件（LKOTANotificationTypeDownload）
  iii. 传输升级包事件（LKOTANotificationTypeTransmit）
  iv. 重启升级事（LKOTANotificationTypeReboot）
  v. OTA 结束（LKOTANotificationTypeFinish）

```
#import
[self.otaBiz startUpgrade:self.iotId alcsOTA:NO  type:LKOTADeviceTypeBle lisener:^(LKOTANotificationTy
pe 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];
```

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

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', '1.5.6'
```

3. 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
- 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	✓	✓	✓

功能	直播播放器	TF卡点播播放器	云端HLS播放器
变速播放	x	✓	✓
画面缩放模式设置	✓	✓	✓
播放器截图	✓	✓	✓
边播边录	✓	✓	✓
提供YUV数据	✓	✓	✓
提供SEI数据	✓	✓	✓

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 | UIViewAutoresizingFlexibleHeight;
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
/*
 注：直播切到手机后台后会自动调用stop方法，如果再次回到直播界面需要重新设置rtmpPath地址，并调用start方法
    rtmpPath具有时效性，如果rtmpPath已过期，需要重新获取rtmpPath地址
    TF卡点播和HLS播放不需要以上操作，请忽略
*/
[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];
```

- 视频截图

```
#import <IMSLinkVisualMedia/IMSLinkVisualMedia.h>
//获取视频截图
UIImage *image = [self.player 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
```

```
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;
/**
直播、点播 共用接口
初次获取画面与分辨率发生变化时回调
注：会触发停止录像
EVENT_RESOLUTION_CHANGE
@see IMSLinkVisualPlayerViewController
@param width 宽
@param height 高
*/
```

```
- (void)linkVisualResolutionChange:(IMSLinkVisualPlayerViewController *_Nullable)player width:(NSInteger)width height:(NSInteger)height;
/*--------------------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

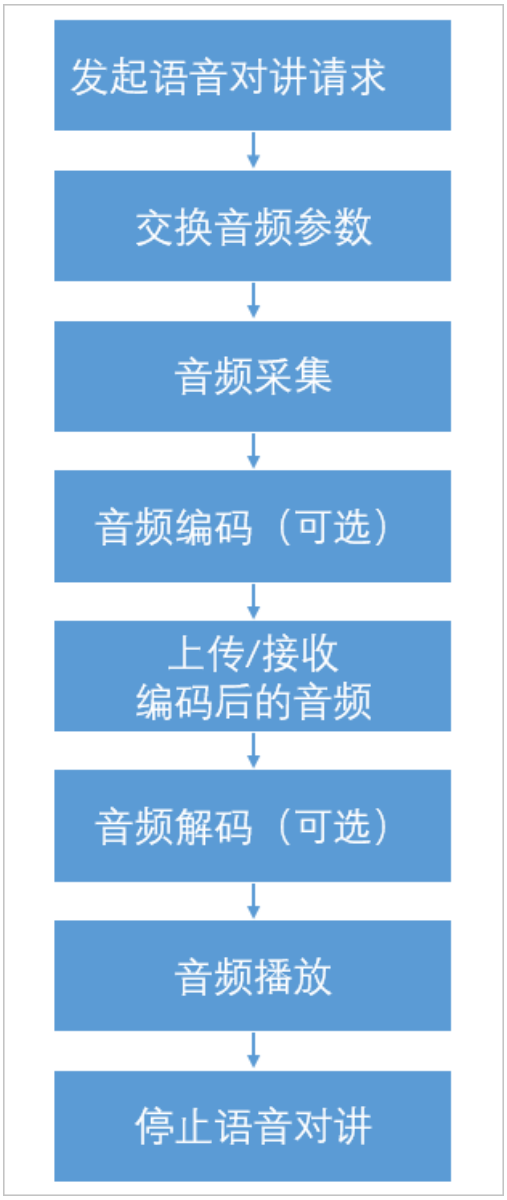
错误主码（PlayerMainError）	描述	子码（解析userInfo中subCode获取）	描述
ErrorSource	数据源相关错误	IMSLinkVisualPlayerErrorConnect	建立连接失败
		IMSLinkVisualPlayerErrorEncrypt	无效的解密密钥
		IMSLinkVisualPlayerErrorUrl	无效的播放地址
		IMSLinkVisualPlayerErrorDataSource	数据源错误或未设置
		IMSLinkVisualPlayerErrorGetURL	获取链接失败
		IMSLinkVisualPlayerErrorStop	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。

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。



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;
```

- 设置对讲数据源

  用于不能通过设备ID发起对讲，需要单独设置数据源

```
/** 设置对讲业务数据源
@param rtmpPath  rtmp地址
@param needEncrypt 是否加密 对讲都为加密，需要填true
@param iv 解密向量，16 byte  如需 base64转码、请查阅文档
@param key 解密密钥，16 byte  如需 base64转码、请查阅文档
NSData* iv = [[NSData alloc] initWithBase64EncodedString:ivString options:NSDataBase64DecodingIgnor
eUnknownCharacters];
@return 是否成功设置数据源
*/
- (BOOL)setDataSource_Intercom:(NSString *_Nullable)rtmpPath needEncrypt:(BOOL)needEncrypt iv:(NS
Data *_Nullable)iv key:(NSData *_Nullable)key;
```

- 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 recordData:(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
```

```
//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

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 'NSInvalidA
rgumentException', reason: '*** -[__NSPlaceholderArray initWithObjects:count:]: attempt to insert nil objec
t 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 - AKOPModuleUpdater.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-a123N7GcC7vD4dFH-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=ZipArchiveBlocksErrorDomain Code=1 "Failed to open output file for writing" UserInfo={NSLocalizedDescription=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=ZipArchiveBlocksErrorDomain 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： {"E7142E1B4BEF6E7D58126924D787701F-release-a123N7GcC7vD4dFH-boneMobile-1.0.4-fdde7a3d0576b493b04b0d5ec83d79c3-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：

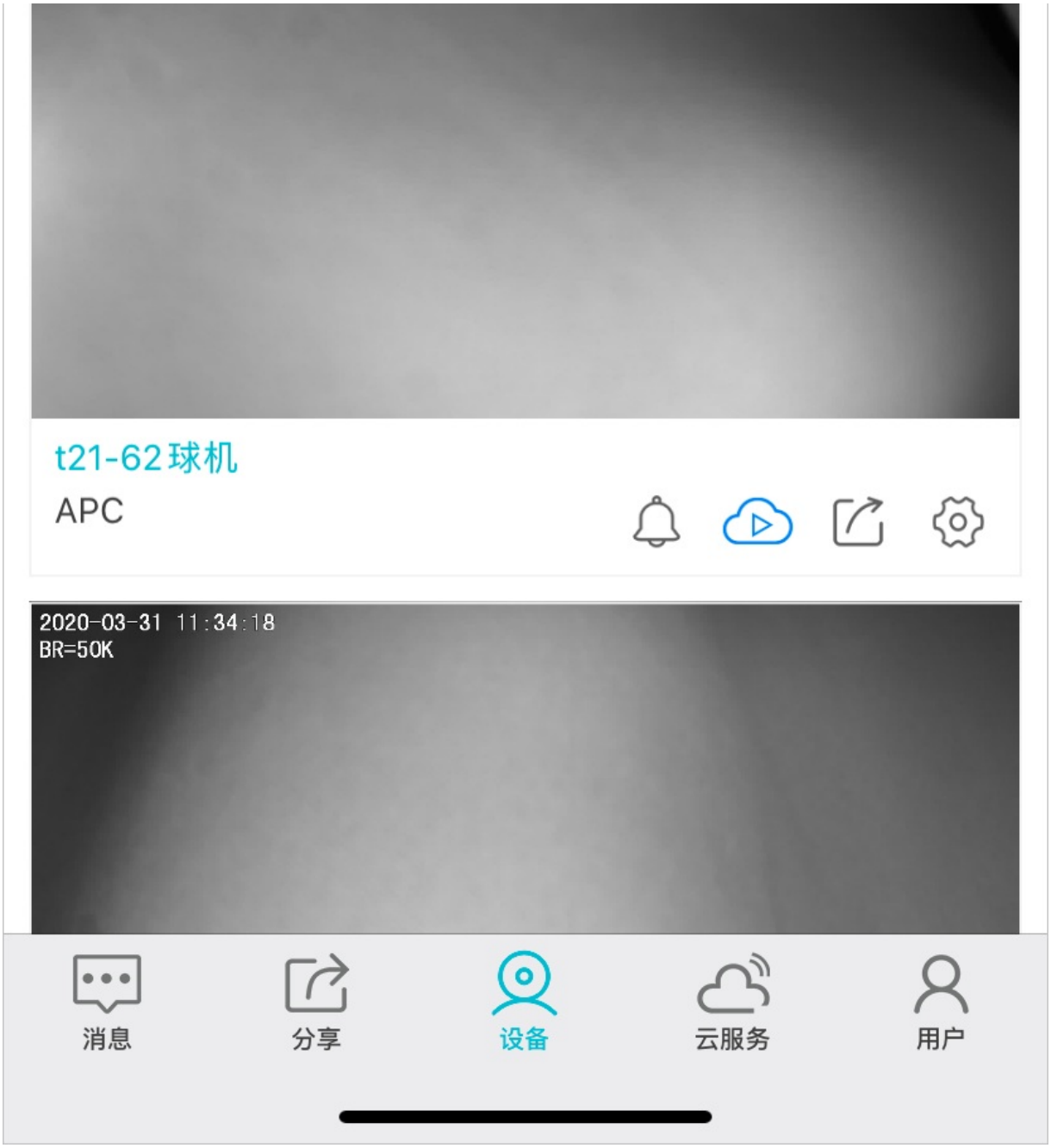
```
[IMSConfiguration initWithHost:@"api.link.aliyun.com" serverEnv:IMSServerRelease];//API通道SDK初始化
[accountSDK setTaeSDKEnvironment:TaeSDKEnvironmentRelease];//账号SDK初始化
```

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。

11.App模板开发指南

11.1. 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，详细请参见服务计费。	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。

11.2. App模板的通用修改

11.2.1. 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。

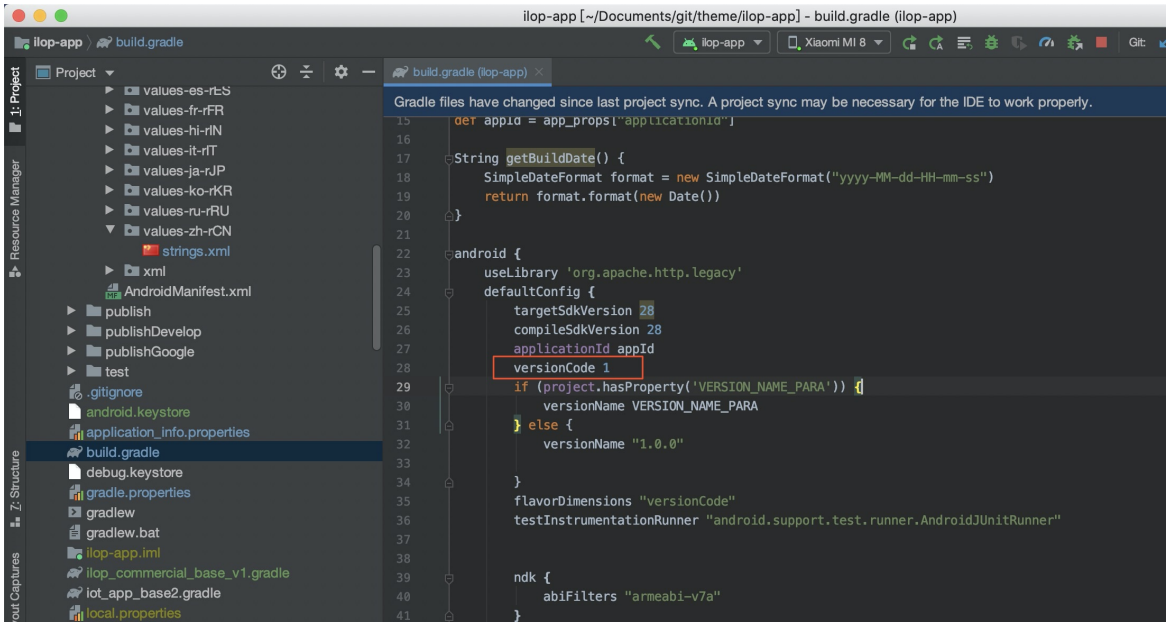
前提条件

- 已完成App模板的源码下载。请参见创建自有App。
- 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。

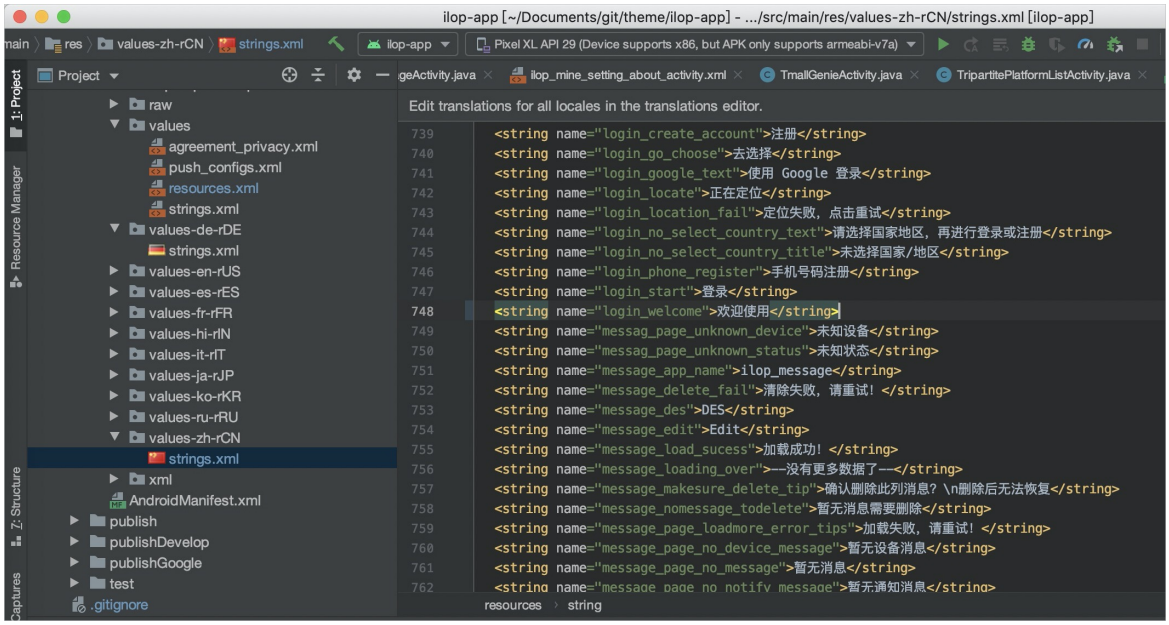
1. 使用Android Studio进入工程的多语言配置文件中。 不同语言的配置文件不同，各语言的路径如下。

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

语言	路径
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。

11.2.2. 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

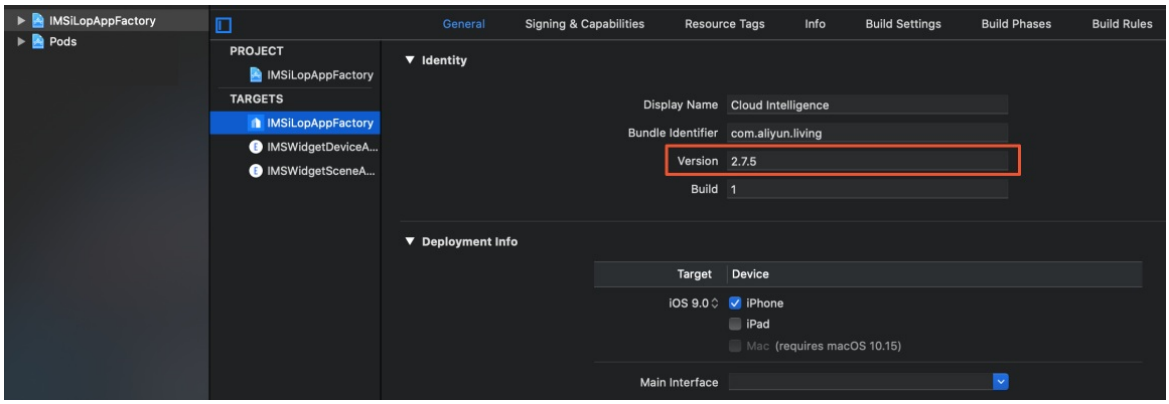
- 已完成App模板的源码下载。请参见创建自有App。
- 已安装iOS App的开发工具xcode。

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。

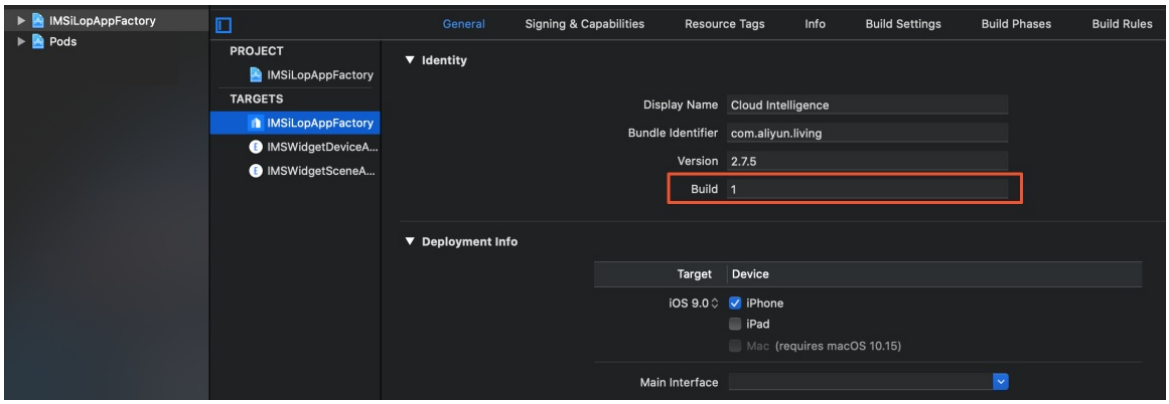
请您根据以下步骤来修改App版本号和Build值。

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。

11.3. 云智能App 2.X系列模板

11.3.1. 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App模板计费介绍。

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

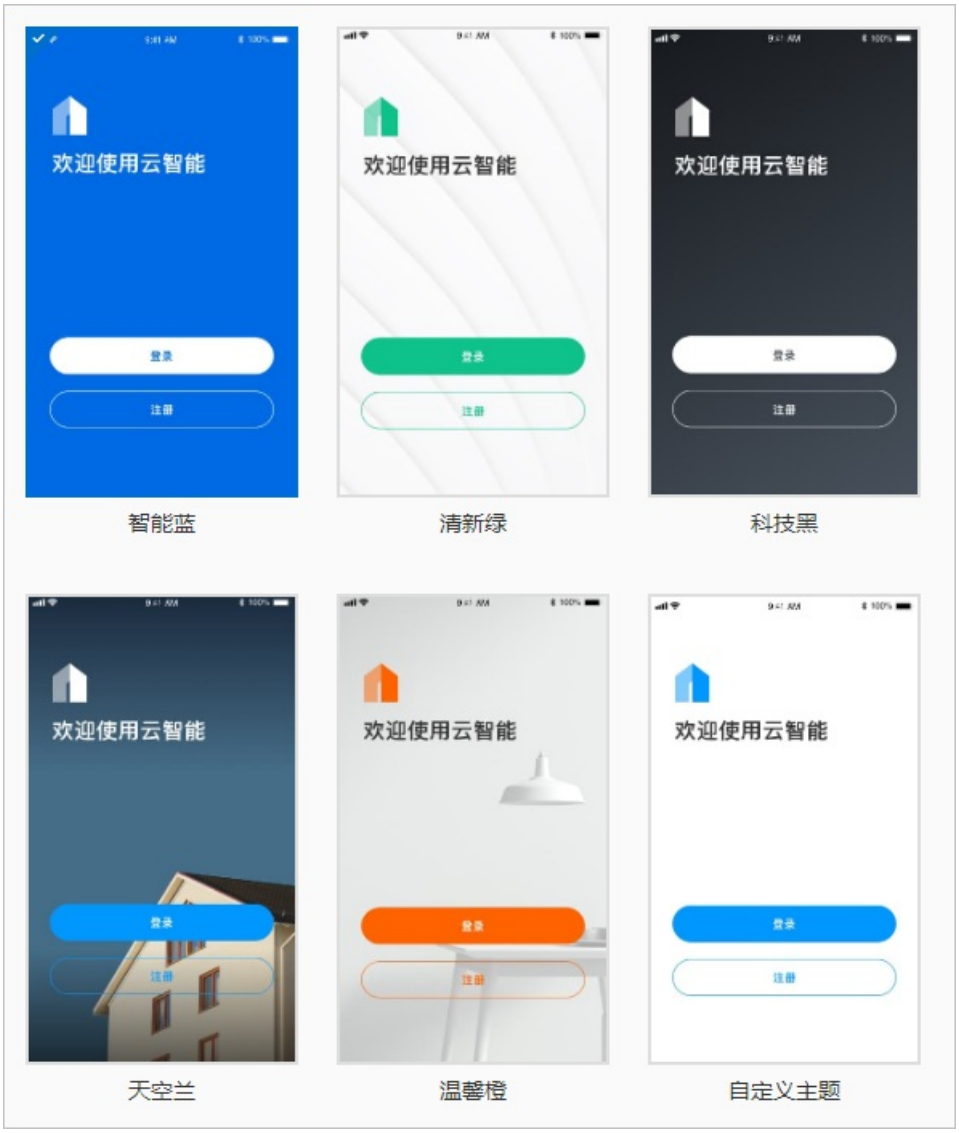
云智能2.X系列模板支持您进行个性化配置，您可以在控制台上配置，也可以在源码中修改。

- 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
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。

11.3.2. 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。请参见创建自有App。
- 已安装iOS App的开发工具xcode。

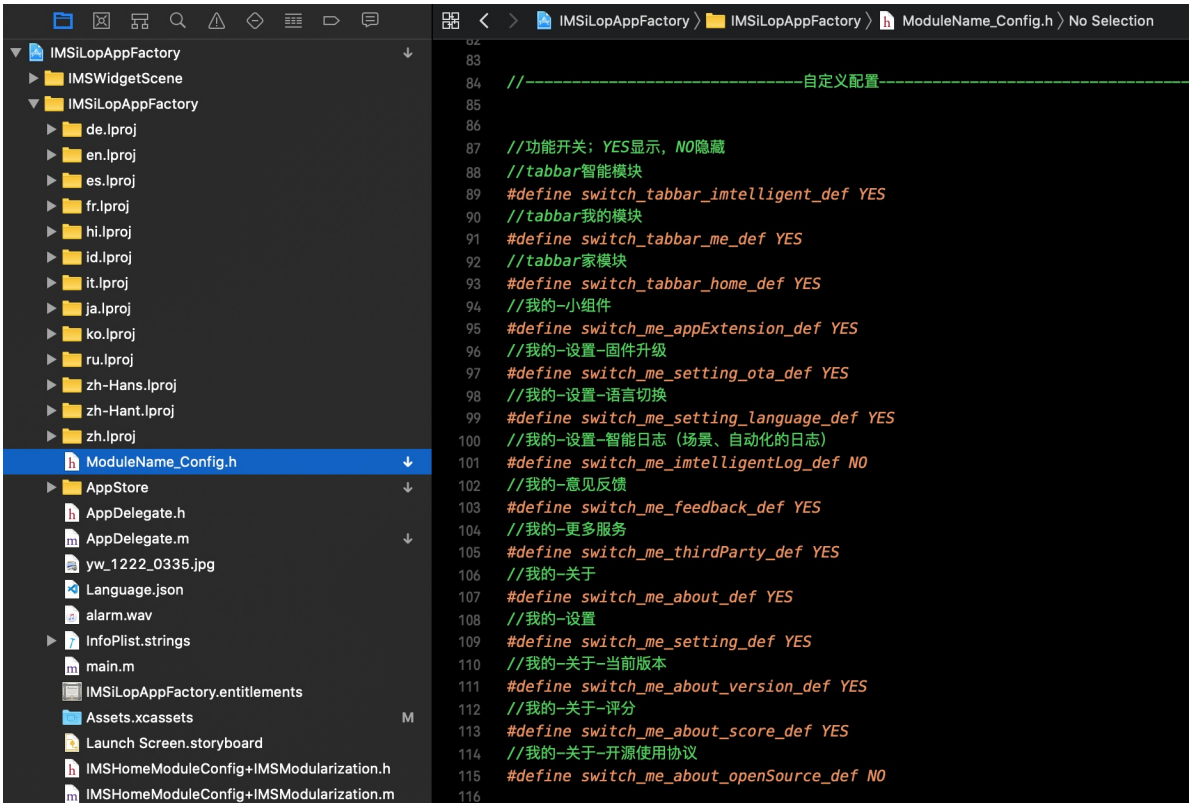
修改说明

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

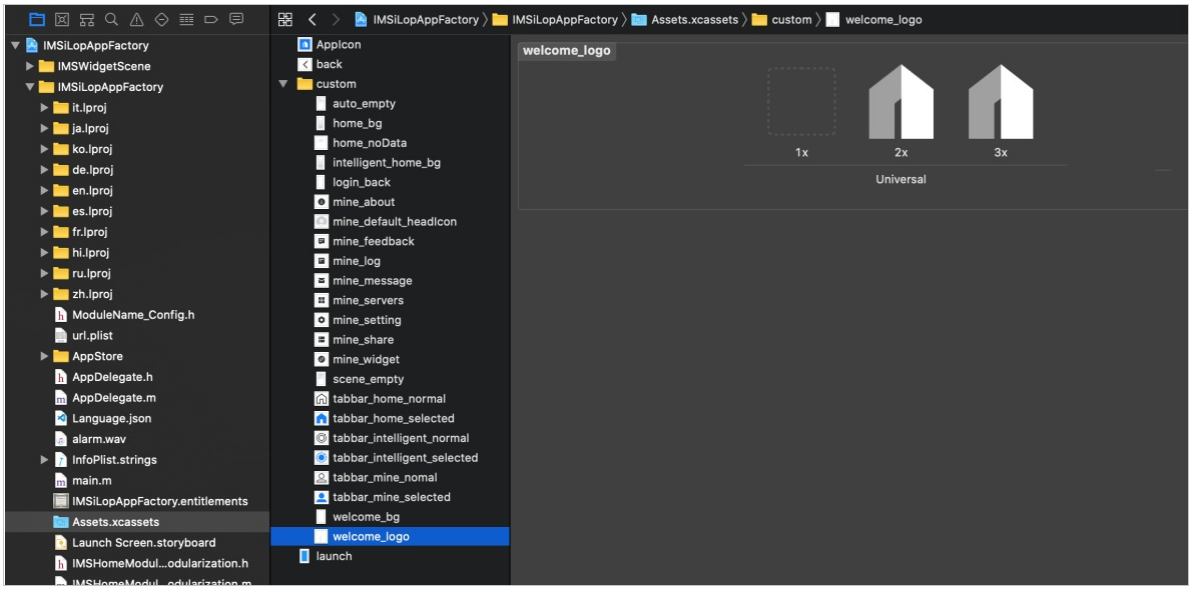
  如果您需要隐藏App中的某菜单，请进入待隐藏模块对应的目录后，修改参数对应的值。如下图所示。



> **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。

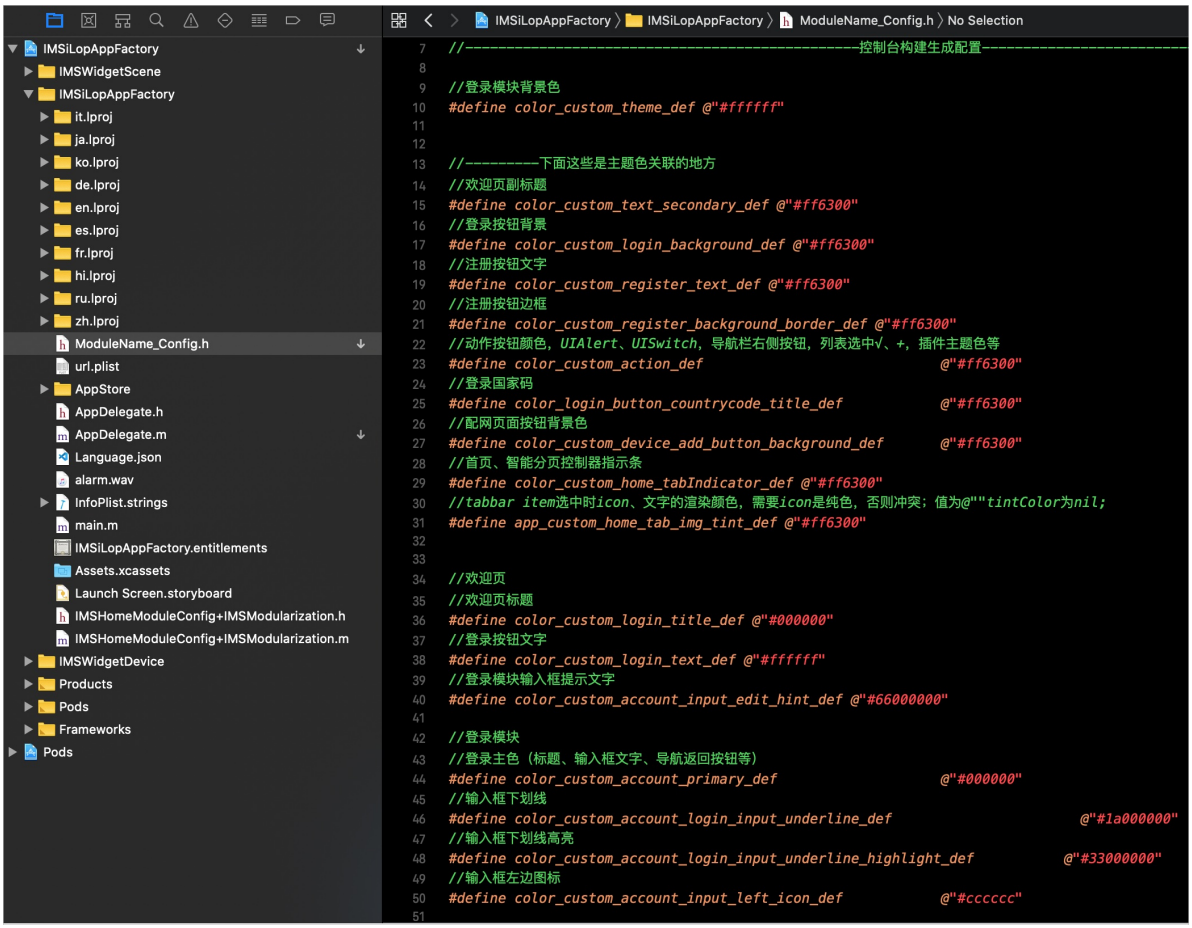


修改App图片时，您需要注意以下内容。

- 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		color_custom_login_background_def

目的	操作目录（或文件）	修改参数的值
修改登录按钮的背景颜色	*./IMSiLopAppFactory/ModuleName_Config.h*	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_imtelligent_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
	是		tabbar_mine_nomal.imageset

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）
	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

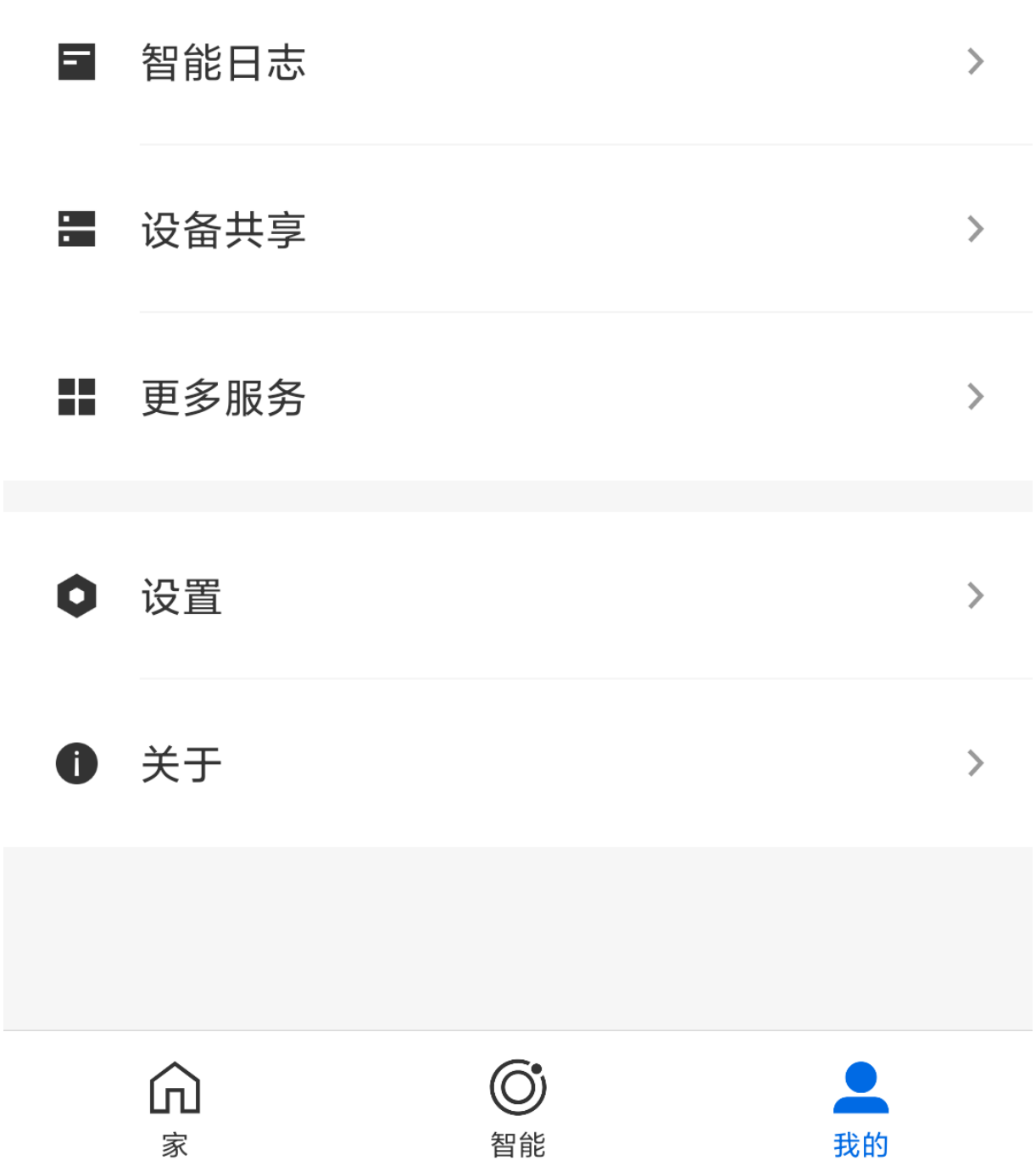
目的	操作父目录	替换子目录下图片（分辨率一致）
修改背景图	*./IMSiLopAppFactory/Assets.xcassets/custom/*	intelligent_home_bg.imageset
修改自动化的空列表背景图		auto_empty.imageset
修改场景的空列表背景图		scene_empty.imageset

- 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。



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

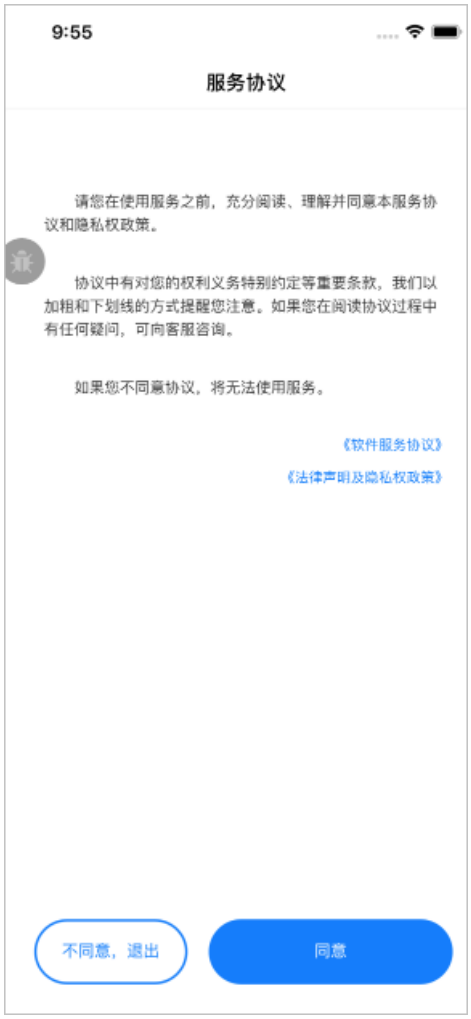
  更多服务中主要展示对接的三方平台，仅对接天猫精灵的源码开源。如果您想对接更多三方平台，可自行对接。对接天猫精灵的详细对接操作请参见自有App接入天猫精灵教程。

> **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` 。

11.3.3. 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。请参见创建自有App。
- 已安装Android App的开发工具，如Android Studio。

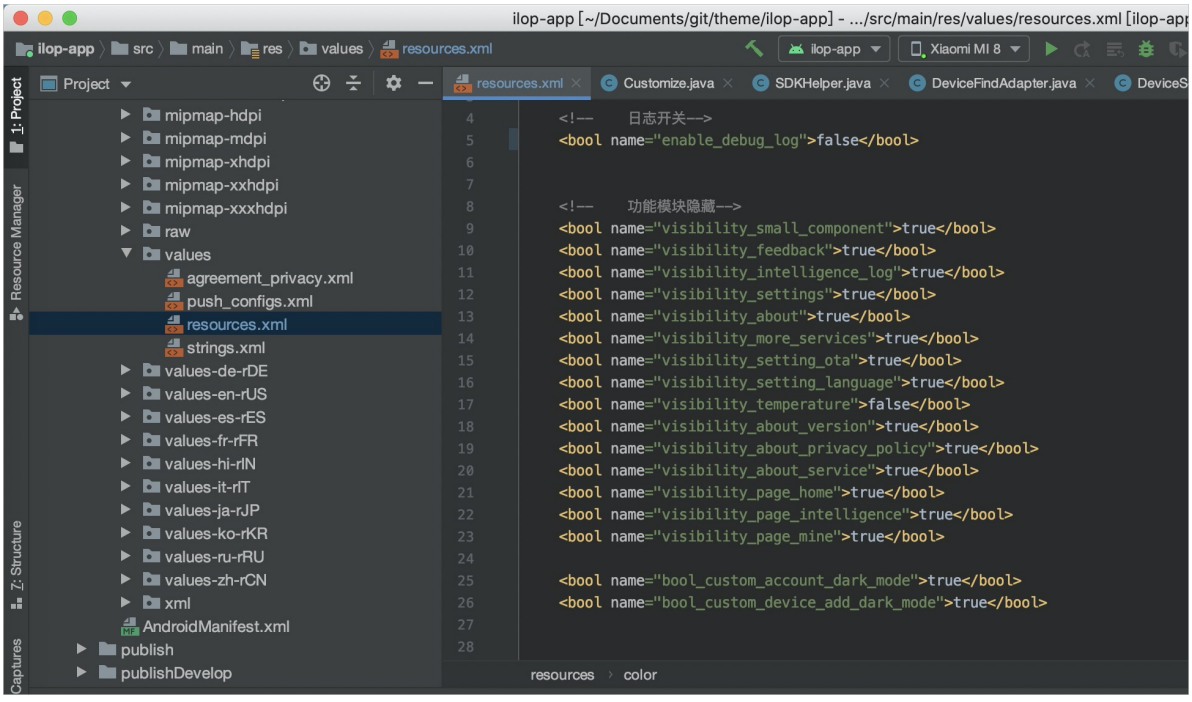
修改说明

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

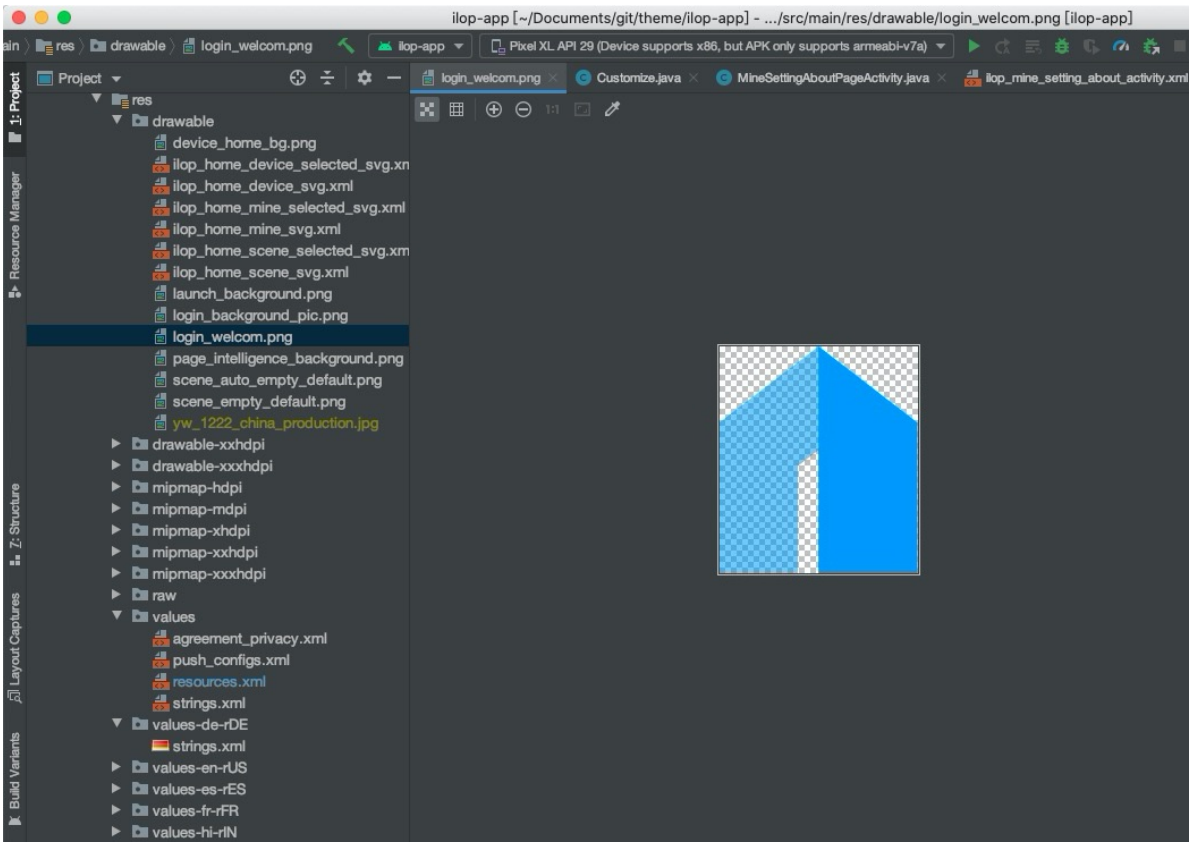
  如果您需要隐藏App中的某菜单，请进入待隐藏模块对应的目录下，修改参数对应的值，如下图所示。



> **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。

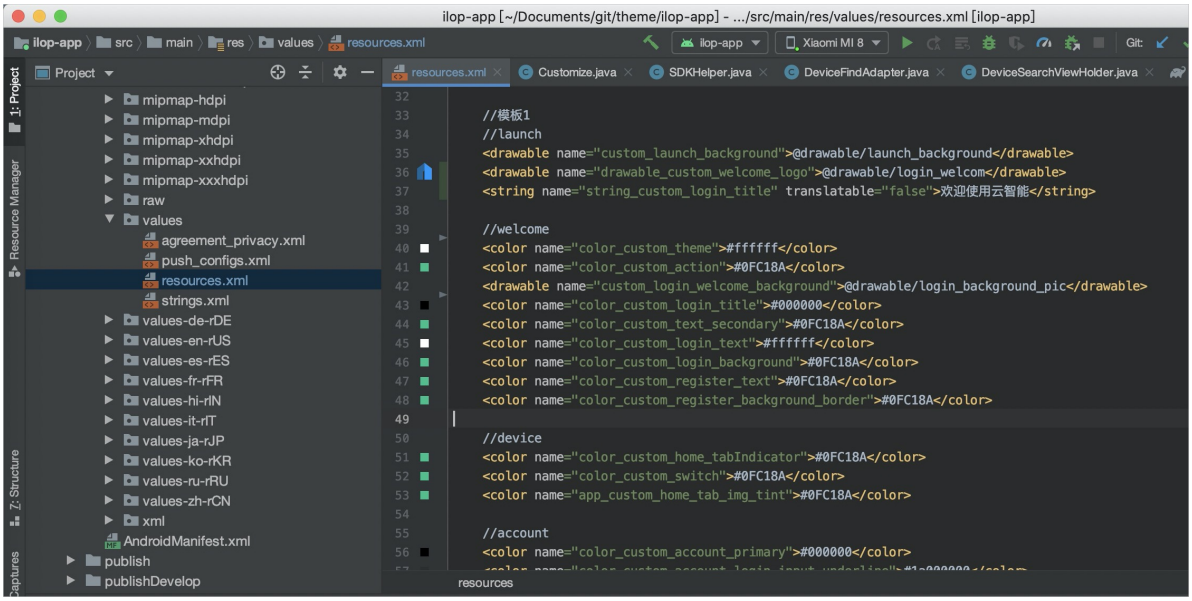


  修改App图片时，需要注意以下内容。

  - 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		color_custom_theme
修改登录按钮的背景颜色		color_custom_login_background

目的	操作目录（或文件）	修改参数的值
修改登录按钮文字的颜色	project-source/src/main/res/values/resources.xml	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

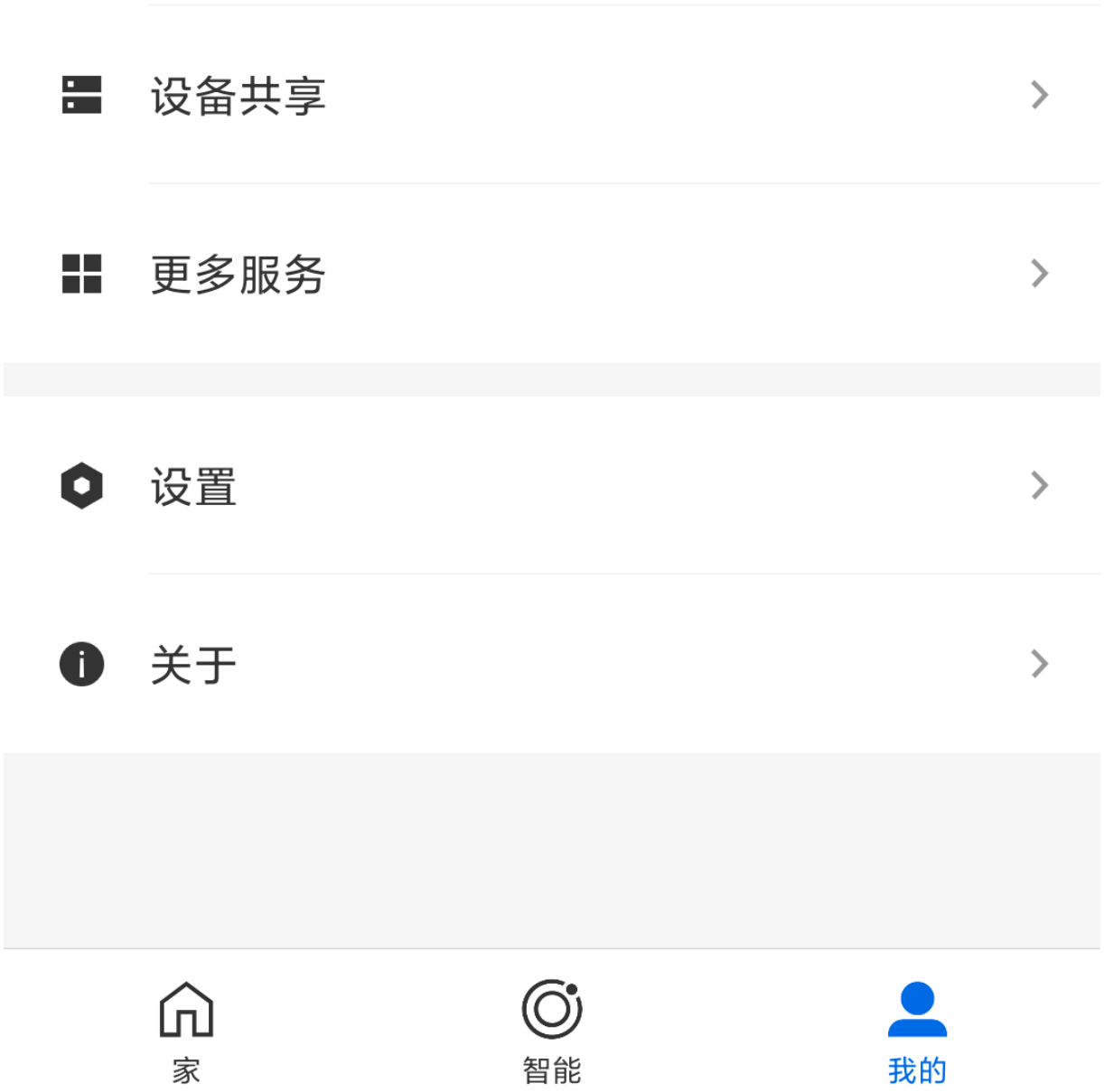
图标	操作目录（或文件）	替换图片（名称须一致）
修改背景图	project-source/src/main/res/drawable-xxxhdpi	scene_empty_default.png
修改自动化的空列表背景图		scene_auto_empty_default.png
修改场景的空列表背景图		scene_empty_default.png

- 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。



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
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

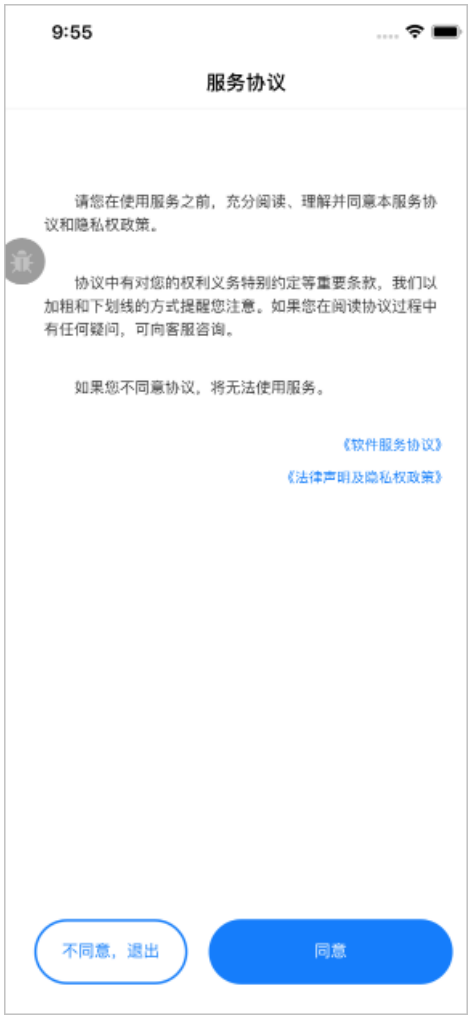
  更多服务中展示了对接的三方平台，仅对接天猫精灵的源码开源。对接天猫精灵的详细对接操作请参见自有App接入天猫精灵教程。您也可以自行对接更多三方平台。

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` 。

11.4. 云智能App 3.X系列模板

11.4.1. 概述

云智能App 3.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3.X系列模板。具体操作，请参见App模板计费介绍。

模板功能介绍

云智能App 3.X系列模板的模板功能与云智能App 2.X系列模板的相同。详细介绍，请参见云智能App 2.X系列模板功能介绍。

个性化配置介绍

云智能App 3.X系列模板对应云智能App（即公版App）的版本号为3.5.5。相比云智能App 2.X系列模板的个性化配置，云智能App 3.X系列模板优化了App的界面和性能，以及增加了以下的可修改源码内容。

- 增加三方账号（谷歌账号、苹果账号）登录自有App。
- 自定义App账号模块、配网模块的状态栏样式。
- 自定义海外用户隐私协议。
- **关于**页面中，设置评分的App。
- **设置**页面中，隐藏首页自动发现设备。
- **更多服务**页面中，显示亚马逊智能音箱、谷歌智能音箱。

除以上变更说明外，更多云智能3.X系列模板个性化配置项介绍，请参见云智能2.X系列模板个性化配置介绍。

11.4.2. iOS 3.X系列模板

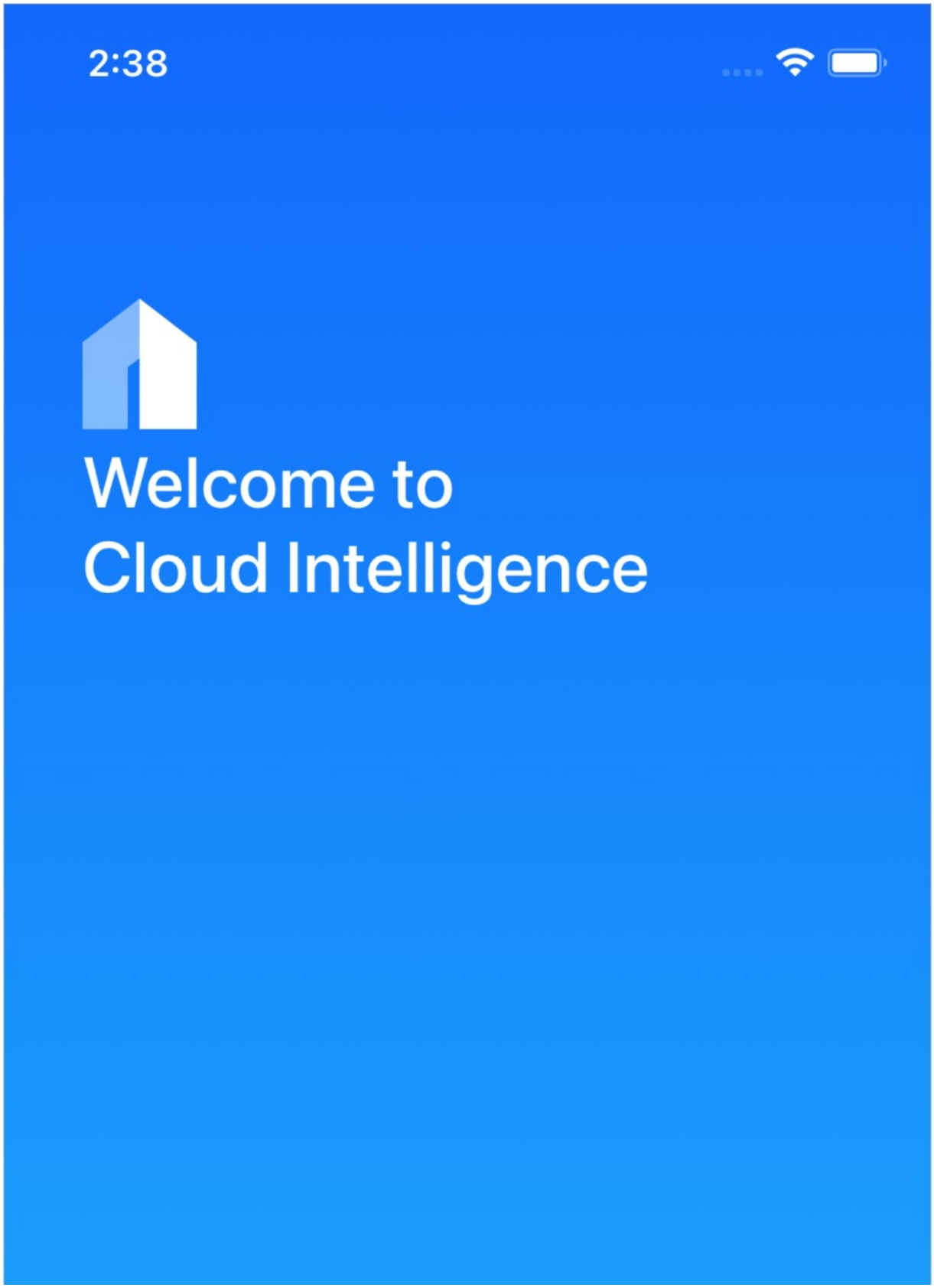
除云智能App 2.X系列模板支持的自定义内容外，云智能App 3.X系列模板还额外支持以下自定义内容。如增加三方账号登录、自定义用户隐私协议、配置页面菜单等。

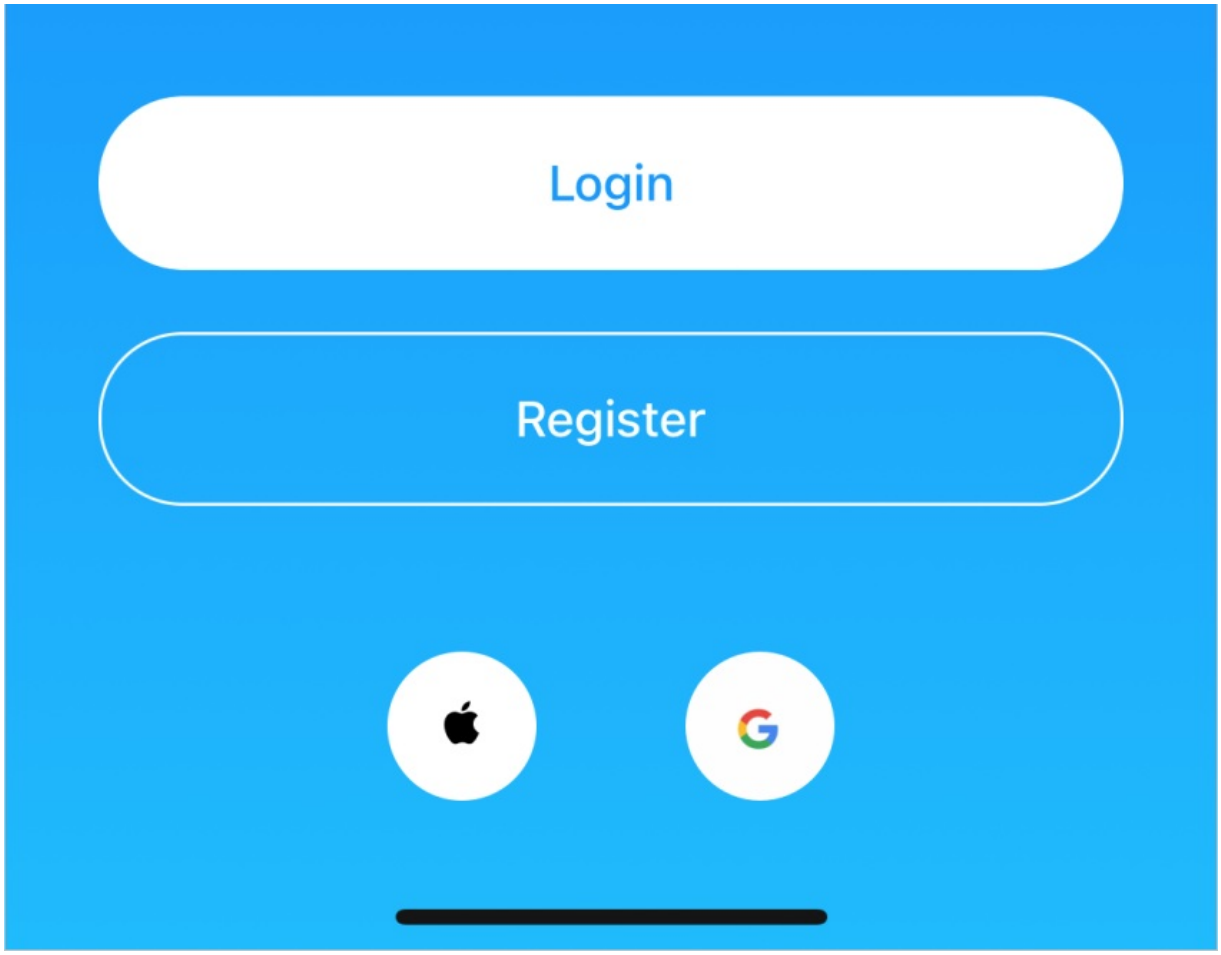
前提条件

- 已完成App模板的源码下载。请参见创建自有App。
- 已安装iOS App的开发工具xcode。

增加三方账号登录

云智能3.X系列App模板支持终端用户使用三方账号快捷登录，如谷歌账号、苹果账号。登录页面如下图所示。





- 增加谷歌账号快捷登录
  i. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
  ii. 修改switch_login_google_def参数的值，打开谷歌显示的开关。
  配置示例如下。

```
switch_login_google_def YES //YES：打开显示开关，NO：关闭显示开关（默认）
```

  > **说明** 使用谷歌登录功能需要开发者去相应平台生成AppKey，并在工程中配置URL Scheme，否则会有闪退。谷歌登录开发引导，请参见谷歌官网内容。

  iii. 至谷歌官网生成AppKey，并在工程中配置URL Scheme。具体操作，请参见谷歌官网内容。
  iv. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
  v. 修改define appkey_login_google_def参数的值。
  配置示例如下。

```
define appkey_login_google_def @"721****818-jqd5l7n****lbg4sih2.apps.googleusercontent.com"
//721****818-jqd5l7n****lbg4sih2.apps.googleusercontent.com为谷歌官网生成的AppKey
```

- 增加苹果账号快捷登录
  i. 在苹果开发者中心开启Sign in with Apple功能。具体操作，请参见苹果官网内容。

ii. 打开工程，在Xcode的**Signing & Capabilities**中开启Sign in with Apple功能。具体操作，请自行查阅网络文档。

iii. 打开*./IMSiLopAppFactory/ModuleName_Config.h*文件。

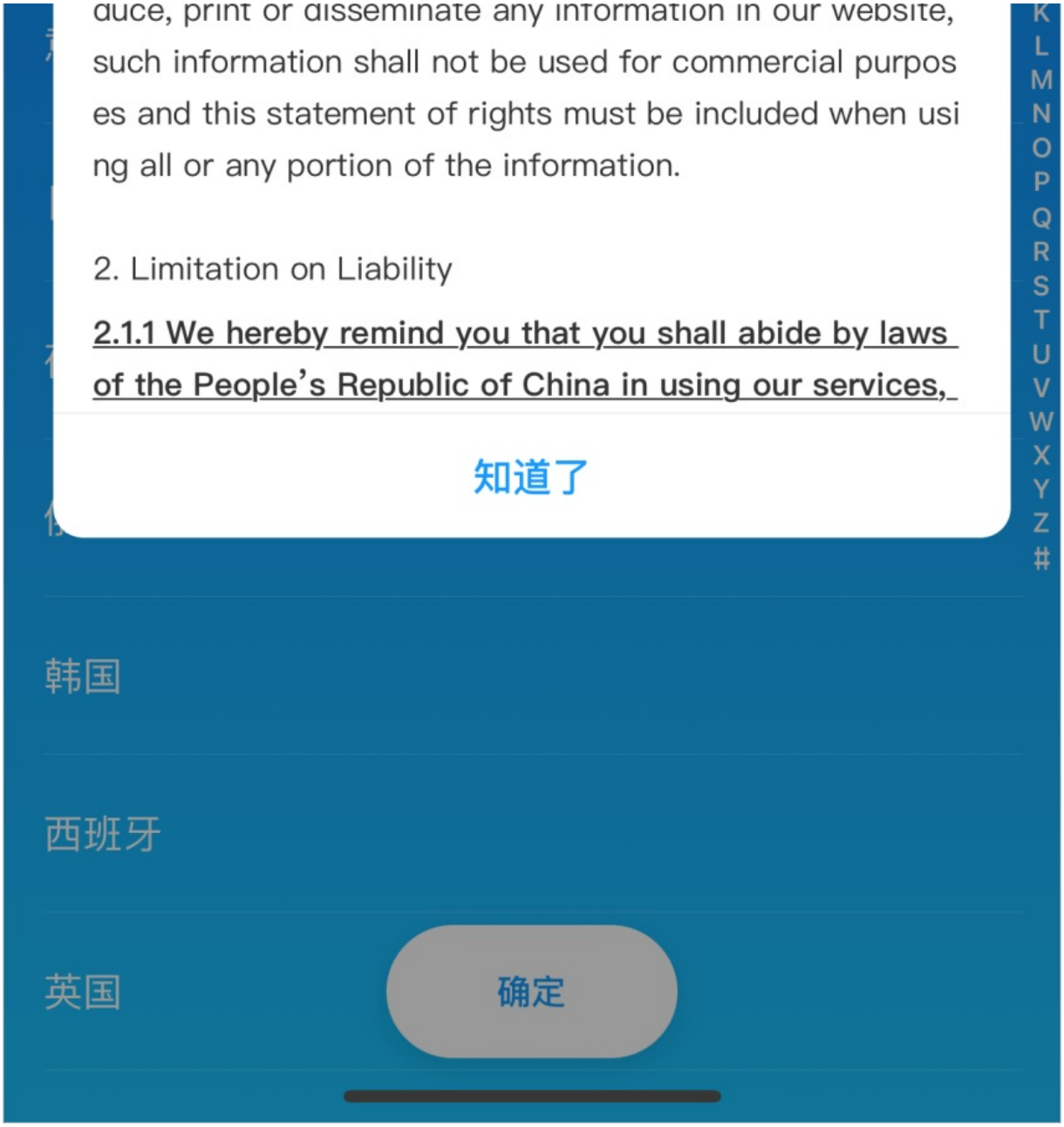
iv. 修改appleId_login_buttonStyle_def参数的值。

配置示例如下。

```
#define appleId_login_buttonStyle_def 0
//0：白底；1：1px黑色边框+白底；2：黑底。详情介绍，请参见苹果官网内容。
```

自定义海外用户隐私协议

在海外用户注册账号时，支持您自定义弹出的用户隐私协议。如下图所示。



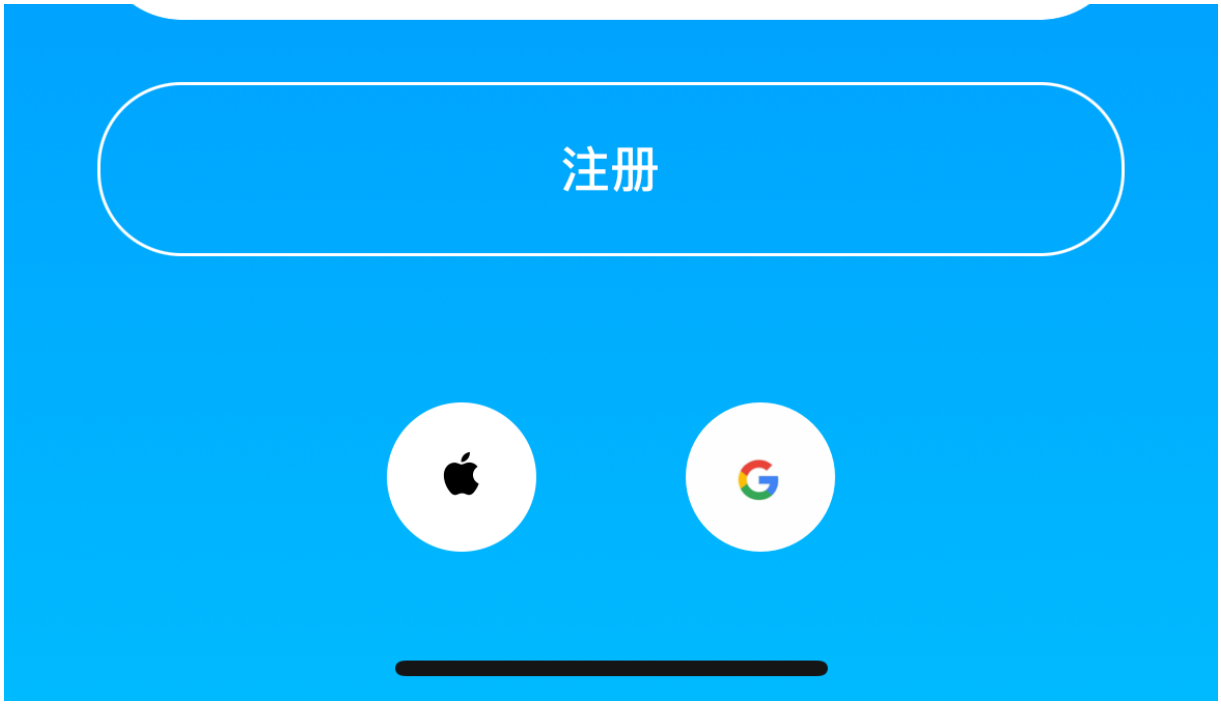
1. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
2. 配置url_account_overseas_privacy_policy_def参数的值。配置示例如下。

```
url_account_overseas_privacy_policy_def @"https://xxxxx"
//该参数不可为空，https://xxxxx为隐私协议的地址
```

自定义App账号模块顶部状态栏样式

支持自定义账号相关页面的背景色和状态栏样式，如下图红色框所示。例如页面设置成白色背景，则将状态栏内容的颜色修改为黑色，从而避免状态栏内容显示不清晰。修改账号页面背景色的参数为color_custom_theme_def。更多介绍，请参见iOS 2.X系列模板。


欢迎使用云智能
登录



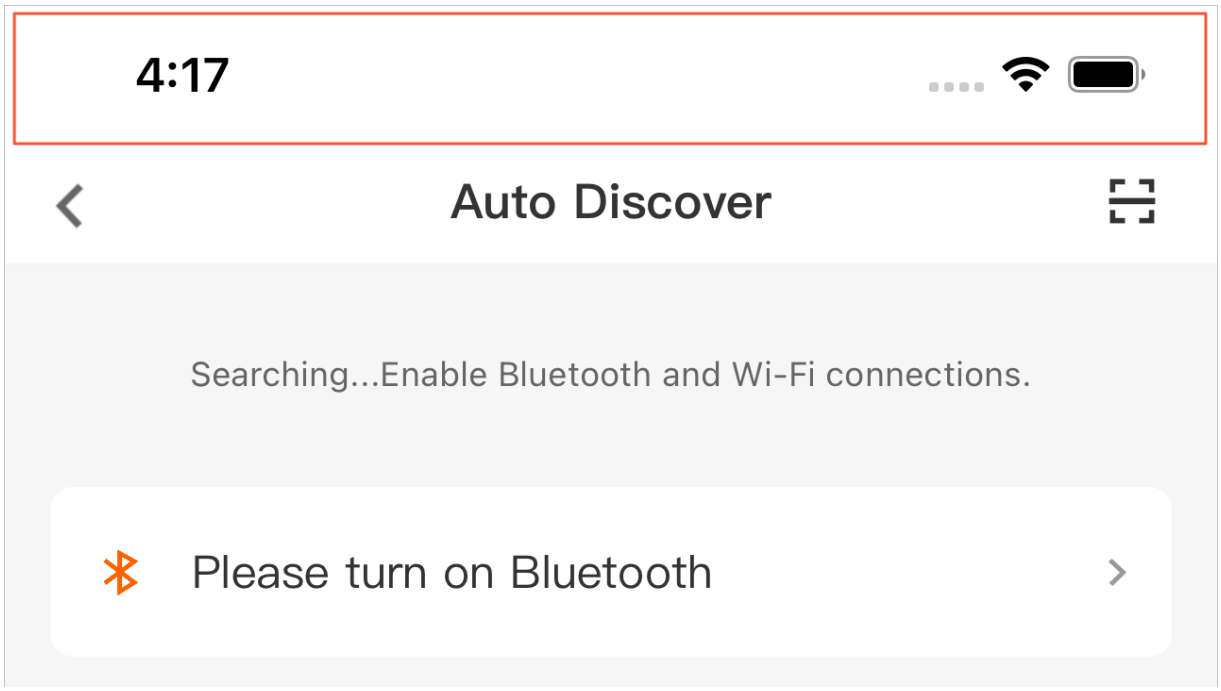
1. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
2. 修改define style_openAccount_statusBar_def参数的值。

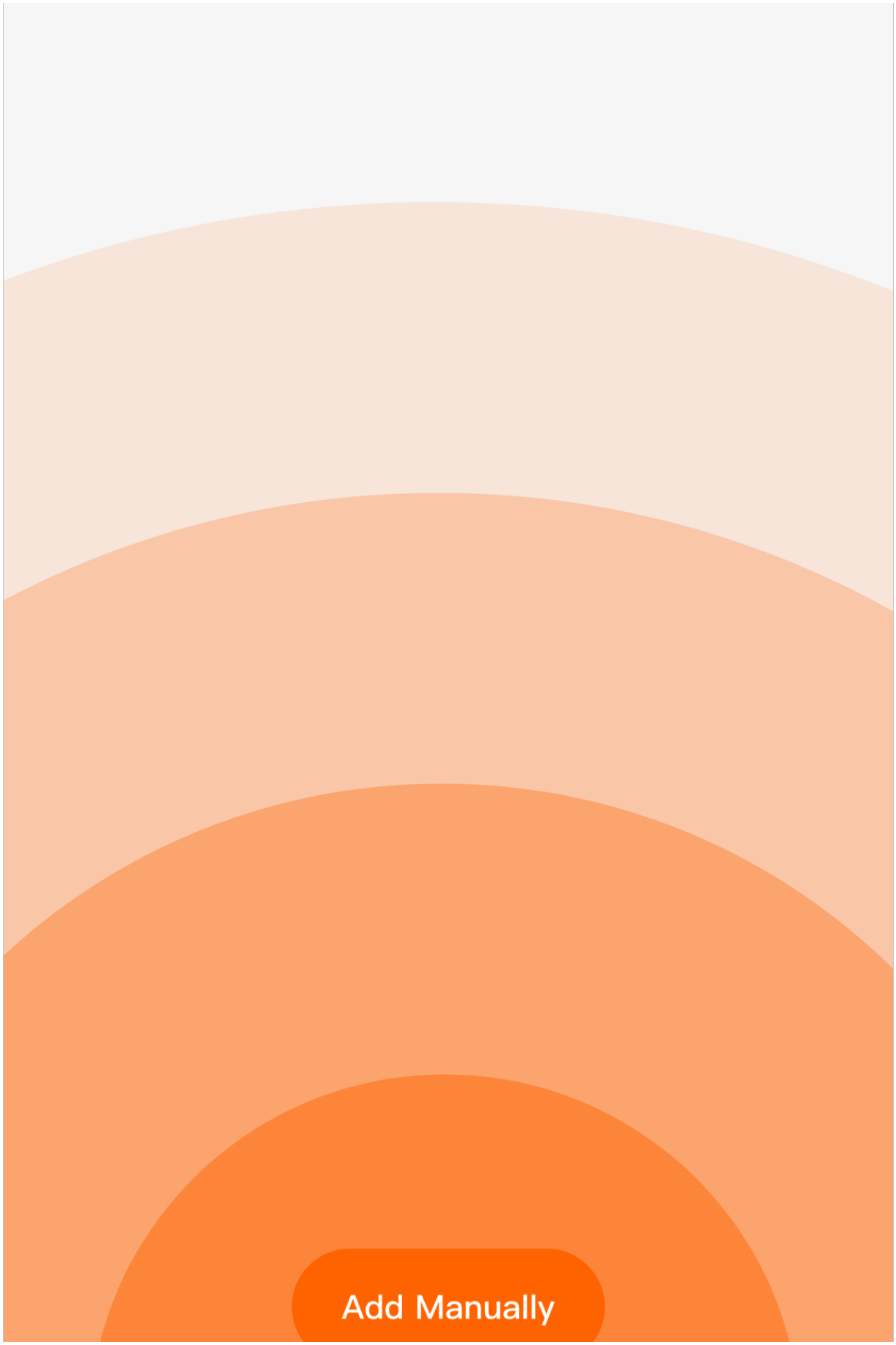
   示例代码如下。

```
#define style_openAccount_statusBar_def 1 //0：黑色字体；1：白色字体（默认）
```

自定义App配网模块顶部状态栏样式

支持自定义配网相关页面的背景色和状态栏样式，如下图红色框所示。例如页面设置成白色背景，则将状态栏内容的颜色修改为黑色，从而避免状态栏内容显示不清晰。修改账号页面背景色的参数为color_custom_device_add_background_def。更多介绍，请参见iOS 2.X系列模板。




Add Manually

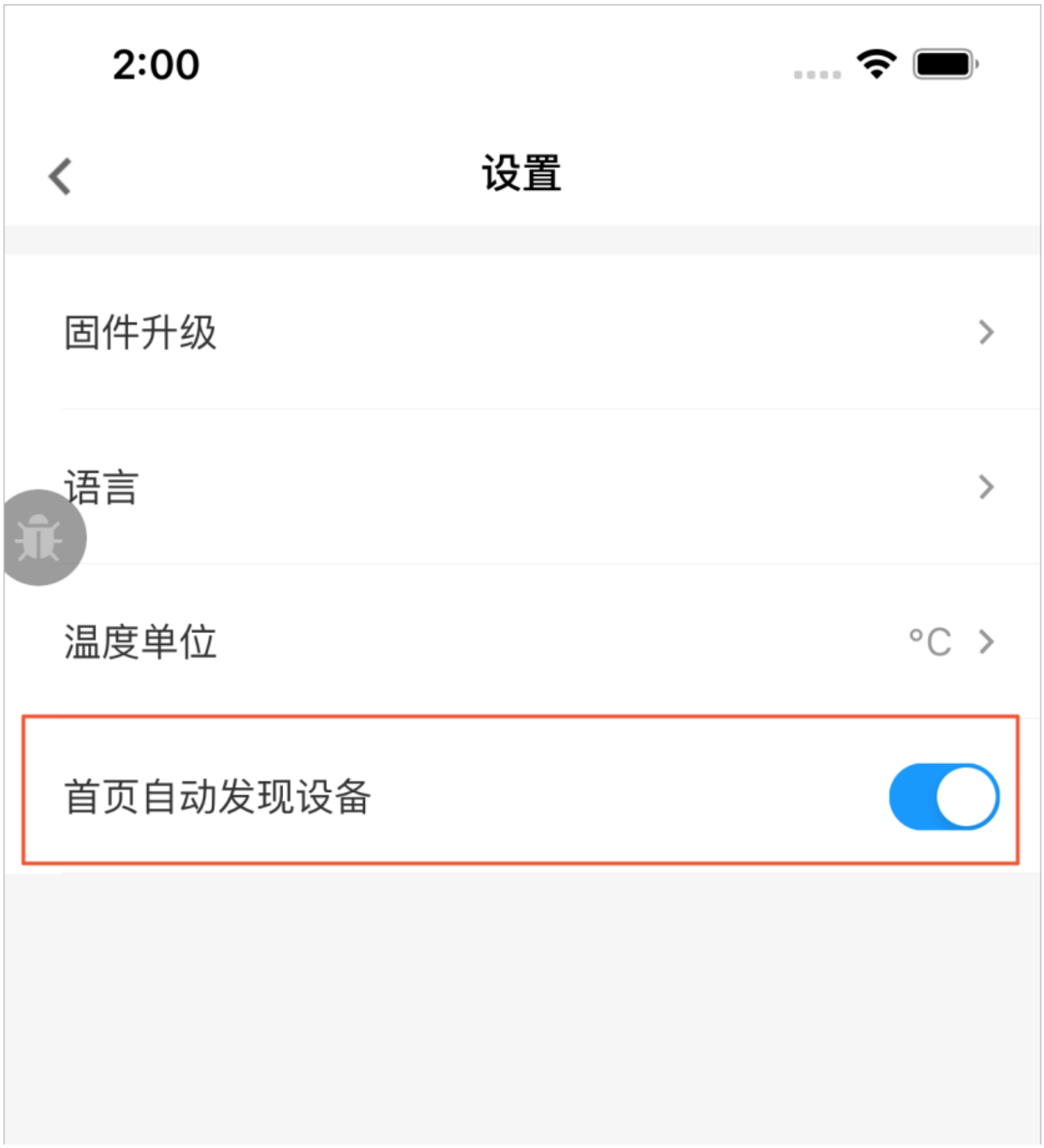
1. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
2. 修改define style_linkNet_statusBar_def参数的值。

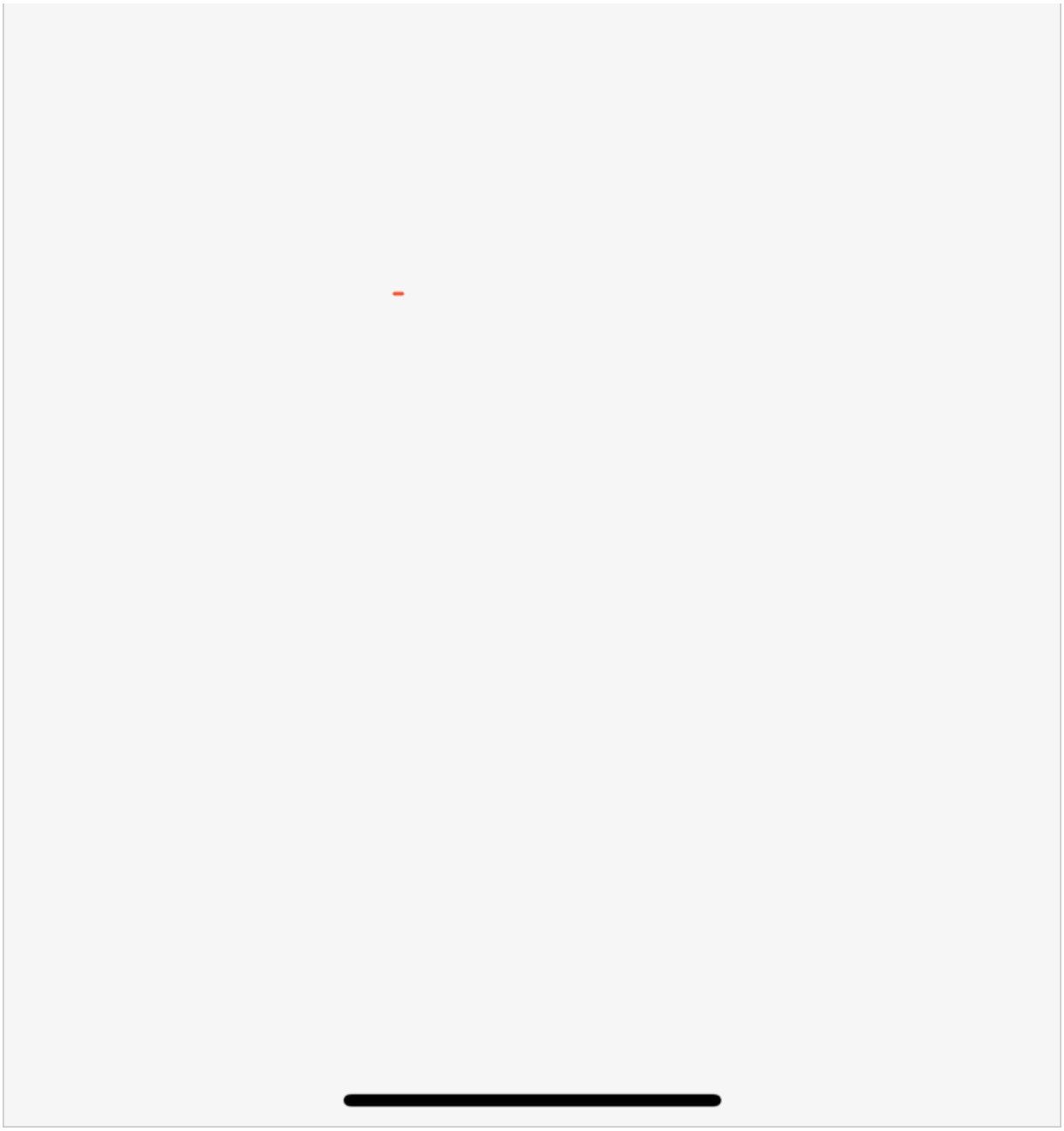
   示例代码如下。

```
#define style_linkNet_statusBar_def 1 //0：白色字体；1：黑色字体（默认）
```

隐藏设置菜单中“首页自动发现设备”

支持您自定义云智能App的**我的** > **设置**页面的**首页自动发送设备**菜单项，如下图所示。





如果您需要隐藏该菜单，请根据以下步骤来操作。

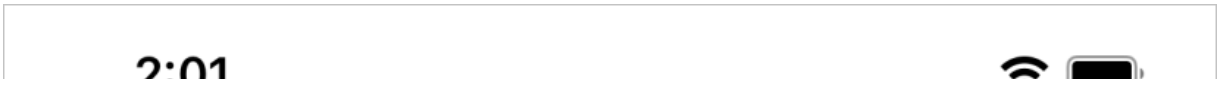
1. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
2. 修改define switch_me_setting_discovery_device_def参数的值。

   示例代码如下。

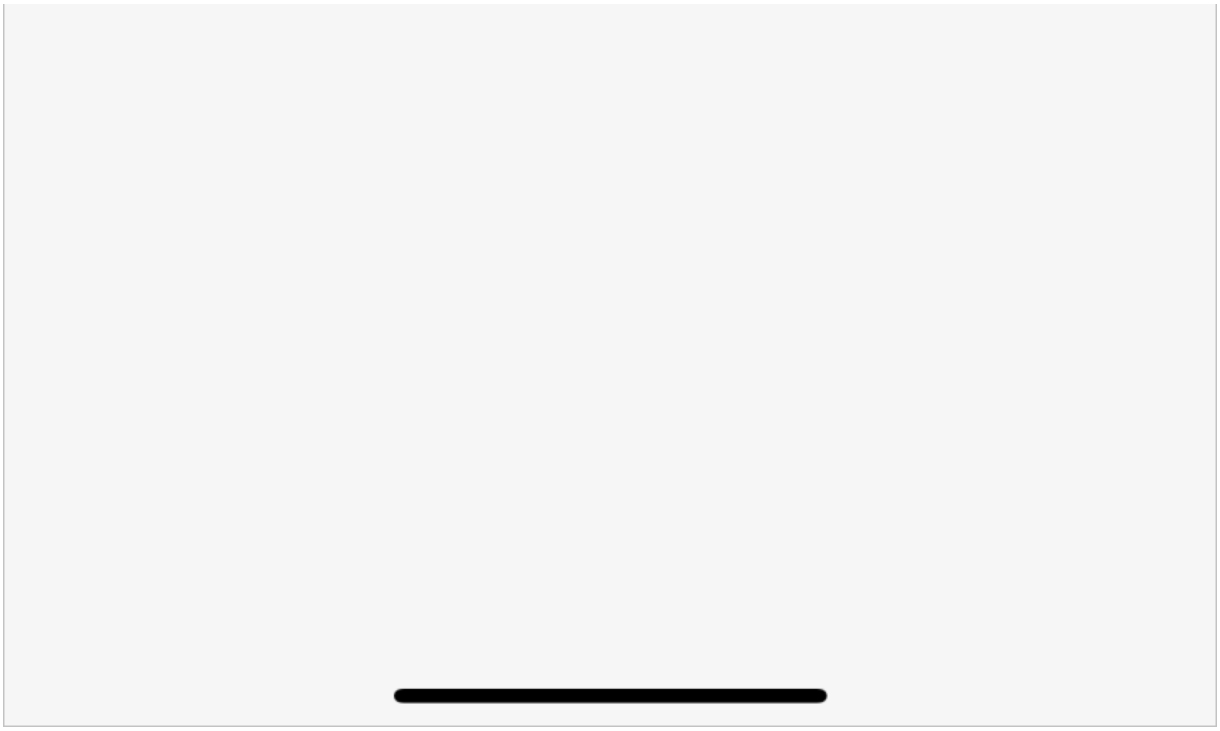
```
#define switch_me_setting_discovery_device_def YES //YES：显示（默认）；NO：隐藏
```

自定义评分的App

在App的**我的** > **关于**页面中，单击**去评分**跳转至App评分页面。您需要修改为自有App的Apple ID，才能对自有App评分。




关于
当前版本
V3.5.5
去评分
隐私权政策
服务协议
开源软件使用协议



1. 前往App Store Connect，获取自有App的Apple ID值。
2. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
3. 修改define switch_me_setting_discovery_device_def参数的值。

   示例代码如下。

```
#define appId_def @"144****649" //144****649为App Store Connect中的Apple ID
```

显示亚马逊、谷歌智能音箱

如果您的App需要对接亚马逊、谷歌的智能音箱，您可以在App的**我的 > 更多服务**页面中，显示相应的智能音箱菜单。


amazon alexa


works with the
Google Assistant

1. 打开工程的*./IMSiLopAppFactory/ModuleName_Config.h*文件。
2. 修改define switch_smartSpeaker_amazon_def参数和define switch_smartSpeaker_google_def参数的值。

   示例代码如下。

```
//亚马逊智能音箱Amazon Alexa
#define switch_smartSpeaker_amazon_def NO //YES：显示；NO：隐藏（默认）
//谷歌智能音箱Google Assistant
#define switch_smartSpeaker_google_def NO //YES：显示；NO：隐藏（默认）
```

11.4.3. Android 3.X系列模板

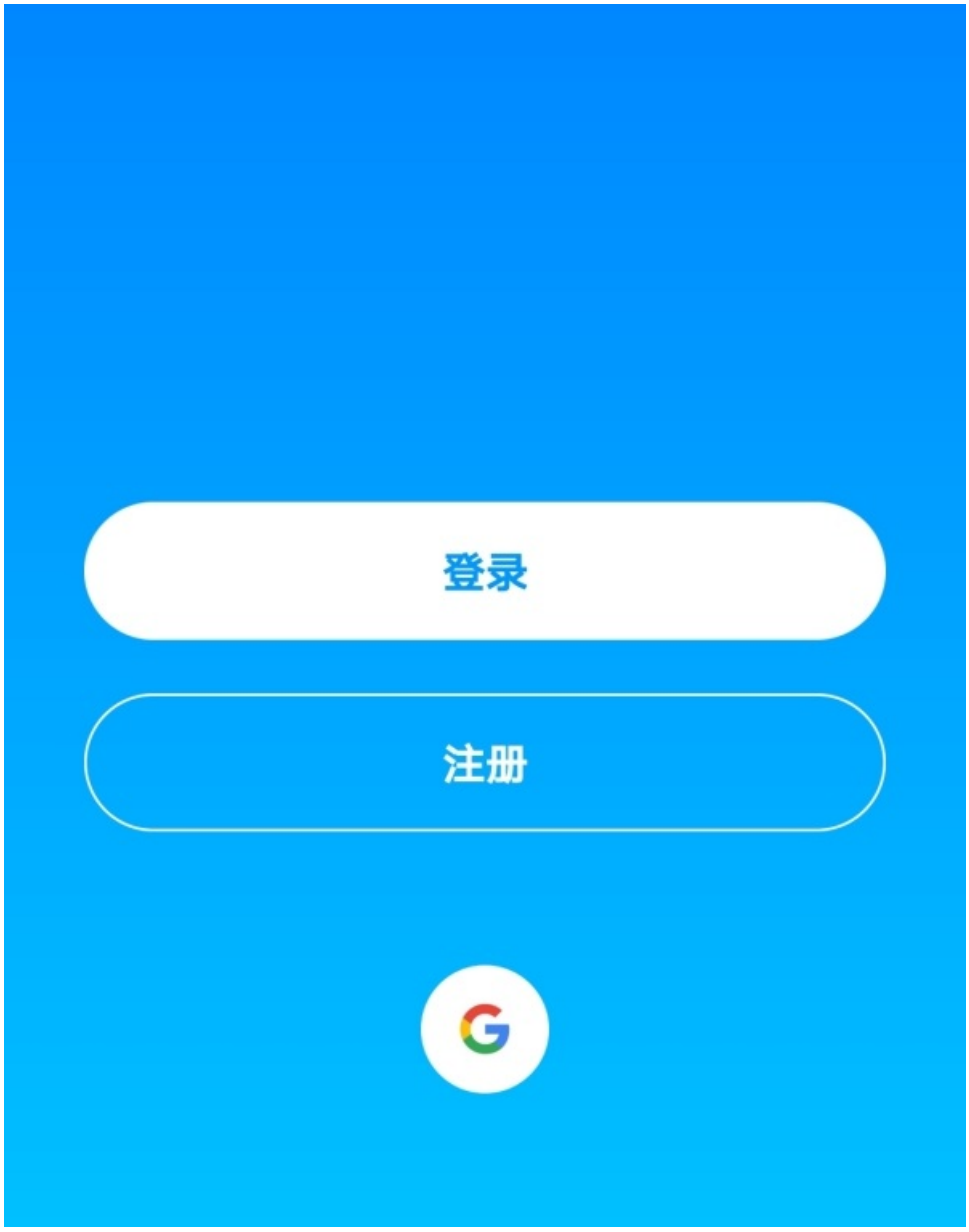
除云智能App 2.X系列模板支持的自定义内容外，云智能App 3.X系列模板还额外支持以下自定义内容。如增加三方账号登录、自定义用户隐私协议、配置页面菜单等。

前提条件

- 已完成App模板的源码下载。请参见创建自有App。
- 已安装Android App的开发工具，如Android Studio。

增加三方账号登录

Android的云智能App 3.X系列模板仅支持谷歌账号进行三方账号快捷登录。登录页如下图所示。



1. 至谷歌官网生成google_id，并在工程中配置*resources*文件。具体操作，请参见谷歌官网内容。
2. 打开工程的*src/main/res/values/resources.xml*文件。
3. 修改visibility_oa_goto_google_switch参数的值，打开谷歌账号登录的显示开关。配置示例如下。

   ```
   <bool name="visibility_oa_goto_google_switch">true</bool>
   //true：打开显示开关；false：关闭显示开关（默认）
   ```

4. 修改define appkey_login_google_def参数的值。

   配置示例如下。

   ```
   <string name="google_id">xxxxxxx</string> //xxxxxxx为谷歌官网生成的google_id
   ```

自定义海外用户隐私协议

在海外用户注册账号时，支持您自定义弹出的用户隐私协议。如下图所示。

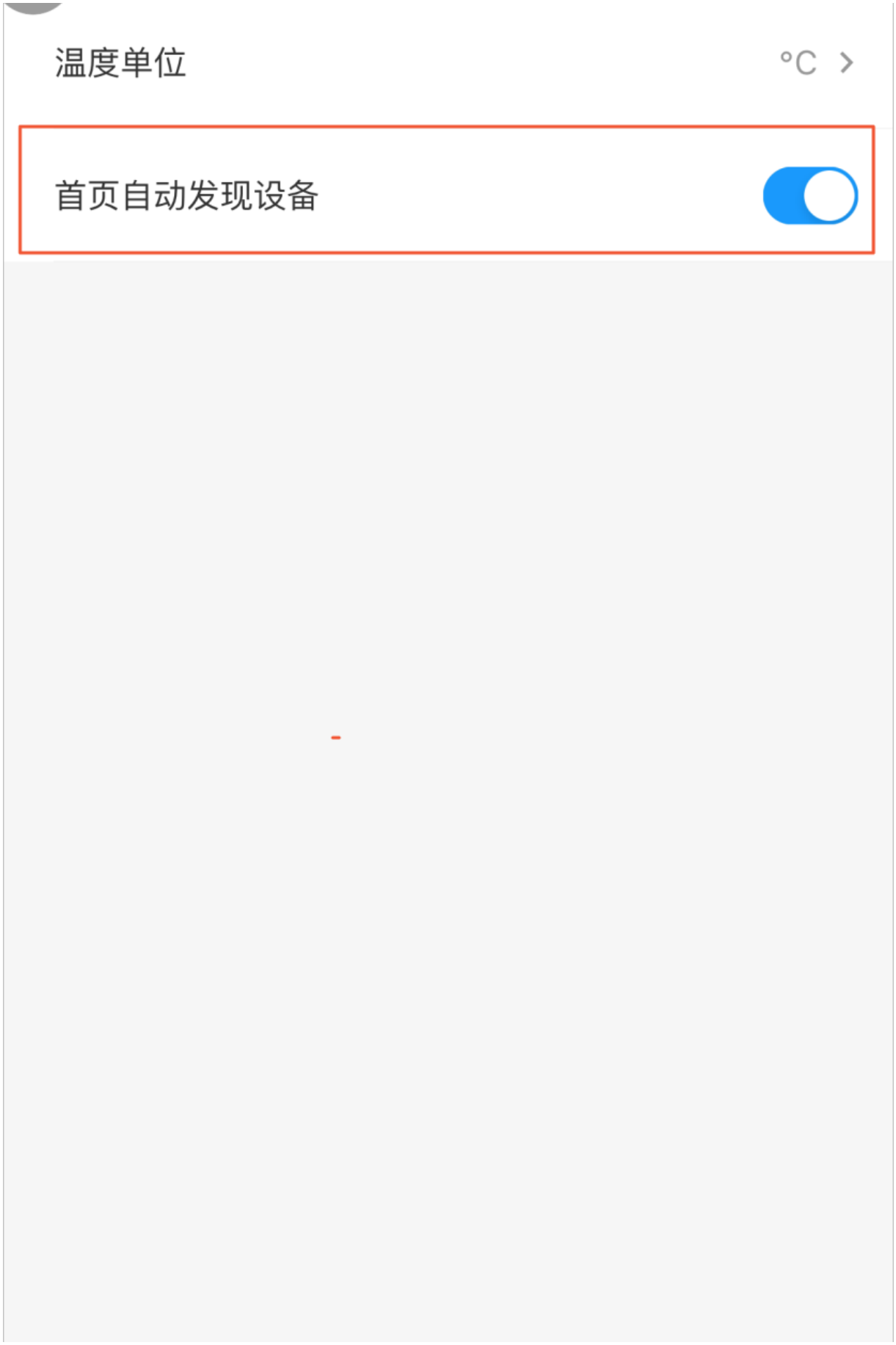

2:23
设备使用国家/地区
请注意，如果选择的国家/地区不正确，可能影响设备的正常使用。
热门国家/地区
中国大陆
服务协议
ments and other information (including but not limited to t
exts, graphs, pictures, photos, audios, videos, icons, color
s, layout and electronic documents) published by us on ou
r website are vested in us. Without our permission, no one
may use the said information (including but not limited to
monitor, reproduce, rebroadcast, display, mirror, upload a
nd download any contents in our website via programs or
devices). Where any person is authorized to browse, repro
duce, print or disseminate any information in our website,
such information shall not be used for commercial purpos
es and this statement of rights must be included when usi
ng all or any portion of the information.
2. Limitation on Liability
2.1.1 We hereby remind you that you shall abide by laws
of the People's Republic of China in using our services,
A B C D E F G H I J K L M N O P Q R S T U V W

1. 打开工程的*src/main/res/values/resources.xml*文件。
2. 配置ilop_privacy_policy_url_oversea参数的值。配置示例如下。

```
<string name="ilop_privacy_policy_url_oversea">https://xxxxx</string>
//该参数不可为空，https://xxxxx为隐私协议的地址
```

隐藏设置菜单中“首页自动发现设备”

支持您自定义云智能App**我的** > **设置**页面的**首页自动发送设备**菜单项，如下图所示。


温度单位
°C
首页自动发现设备

如果您需要隐藏该菜单，请根据以下步骤来操作。

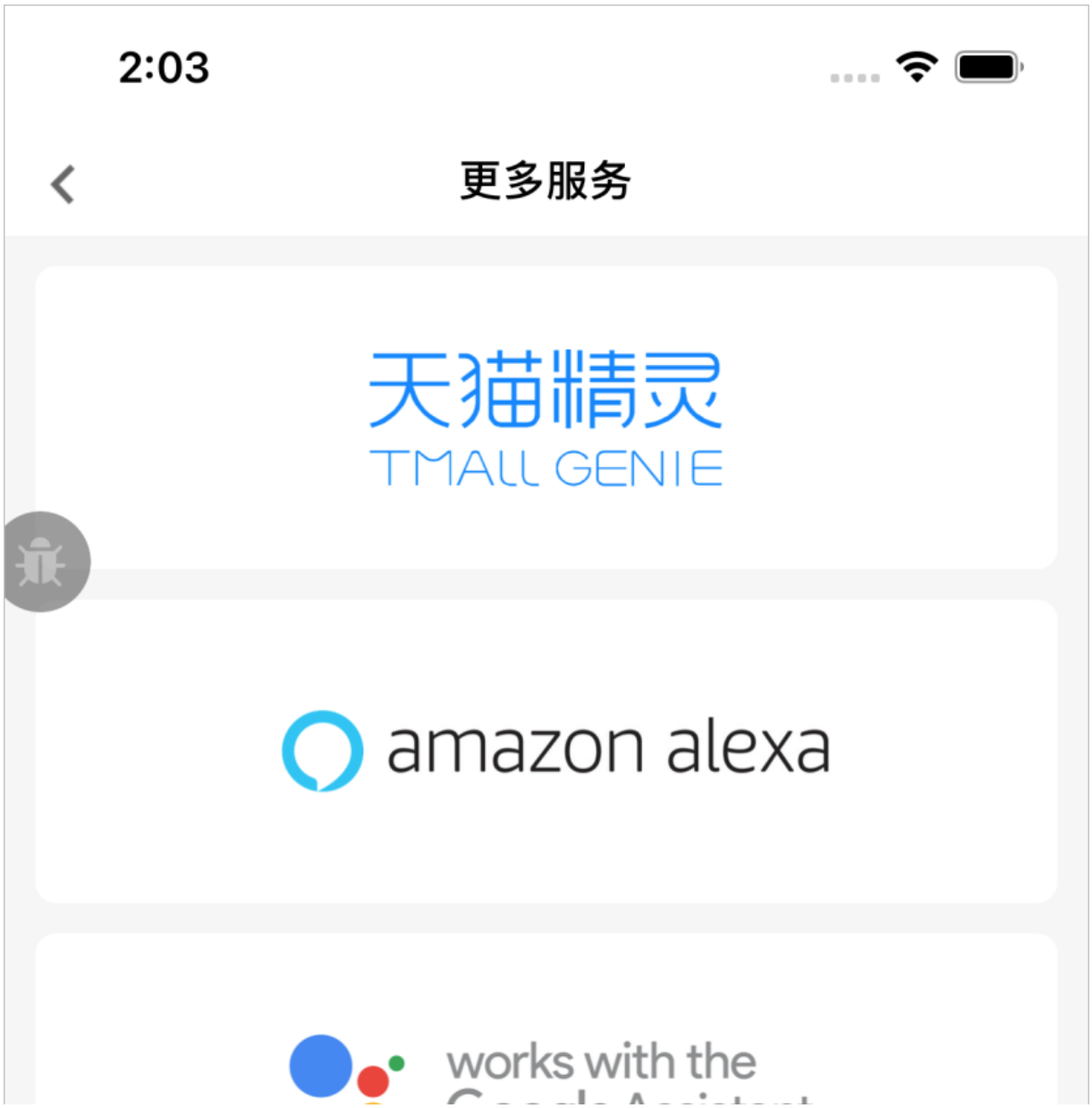
1. 打开工程的*src/main/res/values/resources.xml*文件。
2. 修改visibility_find_switch参数的值。

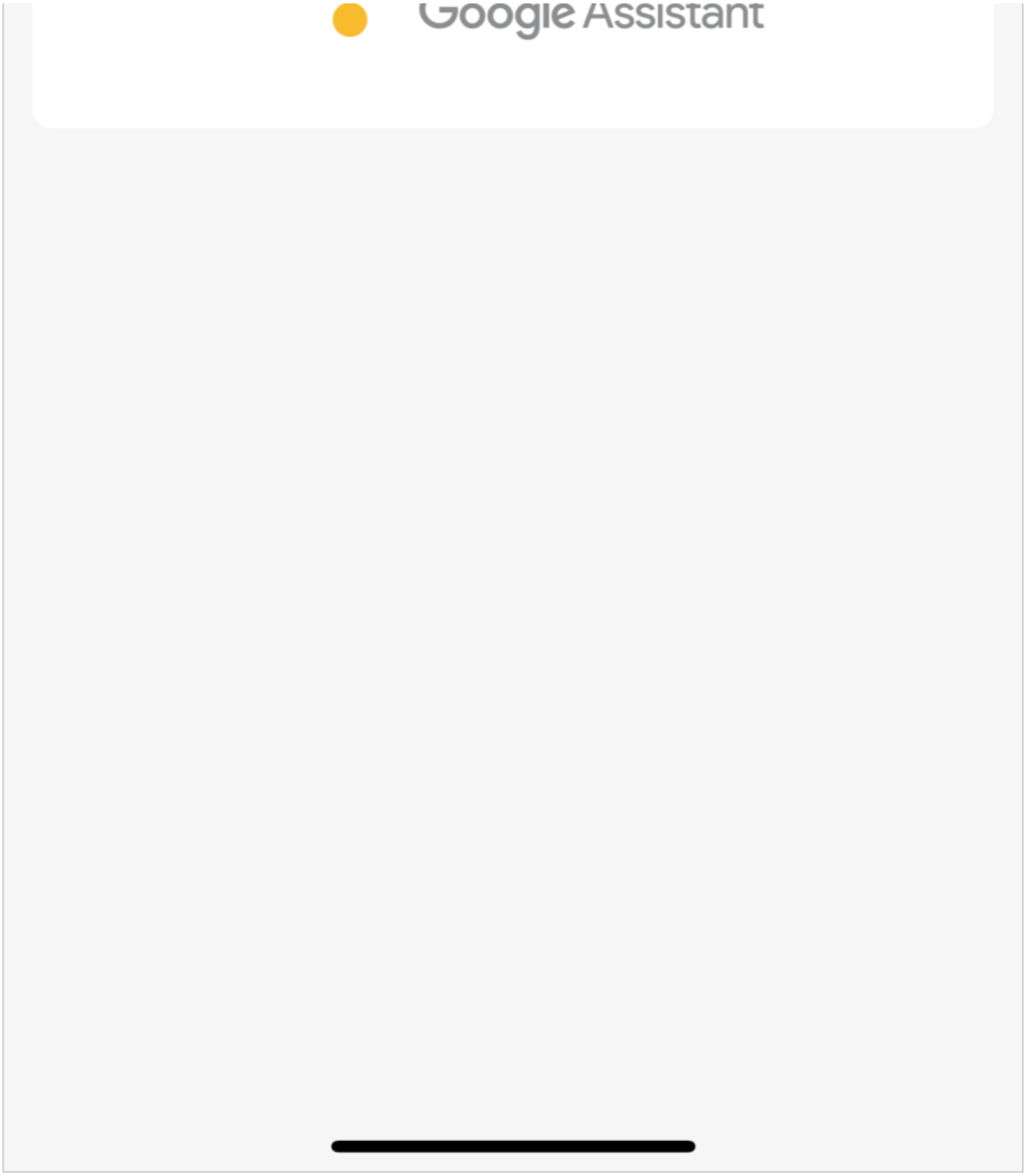
   示例代码如下。

```
<bool name="visibility_find_switch">false</bool> //true：显示菜单；false：隐藏菜单
```

显示亚马逊、谷歌智能音箱

如果您的App需要对接亚马逊、谷歌的智能音箱，您可以在App的**我的 > 更多服务**页面中，显示相应的智能音箱菜单。





1. 打开工程的*src/main/res/values/resources.xml*文件。
2. 修改visibility_page_mine_more_amazonalexa参数和visibility_page_mine_more_google参数的值。

   示例代码如下。

```
//亚马逊智能音箱Amazon Alexa
<bool name="visibility_page_mine_more_amazonalexa">true</bool> //true：显示菜单；false：隐藏菜单（默认）
//谷歌智能音箱Google Assistant
<bool name="visibility_page_mine_more_google">true</bool> //true：显示菜单；false：隐藏菜单（默认）
```

11.5. Demo App模板

11.5.1. 概述

Demo App模板用于帮助您快速上手App的开发，可用于从零搭建您自己的自有品牌App。如果您是初学者，也可以用于熟悉生活物联网平台提供的SDK和API等。

模板功能介绍

生活物联网平台提供了一套免费的便于快速上手的Demo App模板，包括简单的配网、登录、设备控制、个人资料和调试工具等功能。

功能	介绍
配网	• 通过产品列表手动进入配网列表 • 通过本地网络发现的方式配网
登录	设备注册和登录，包括账号注册、账号登录、修改密码等
设备控制	配网绑定后的设备，可以通过App面板进行控制
个人资料	简单的个人中心，包括查看当前登录账号、退出登录
调试	Demo App为您提供了一个调试小工具，可用来切换环境、查看日志等

源码开放程度介绍

除生活物联网平台提供的SDK的源码外，Demo App模板其余所有的源码都对您开放。

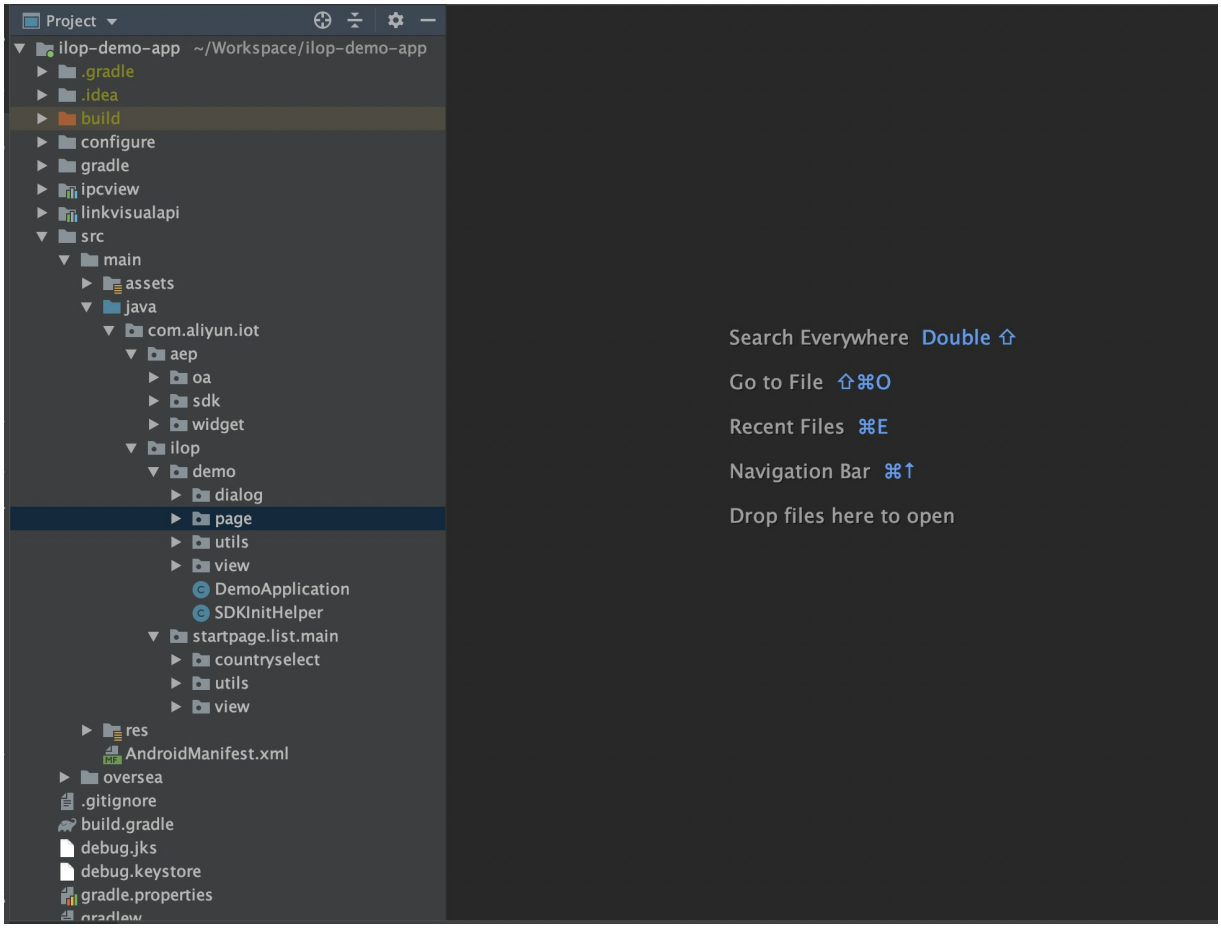
11.5.2. Android Demo App模板

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。详细操作请参见创建自有App。

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，操作请参见集成安全图片。

   > **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。

i. 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')
```

ii. 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");
```

iii. 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);
```

iv. 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。

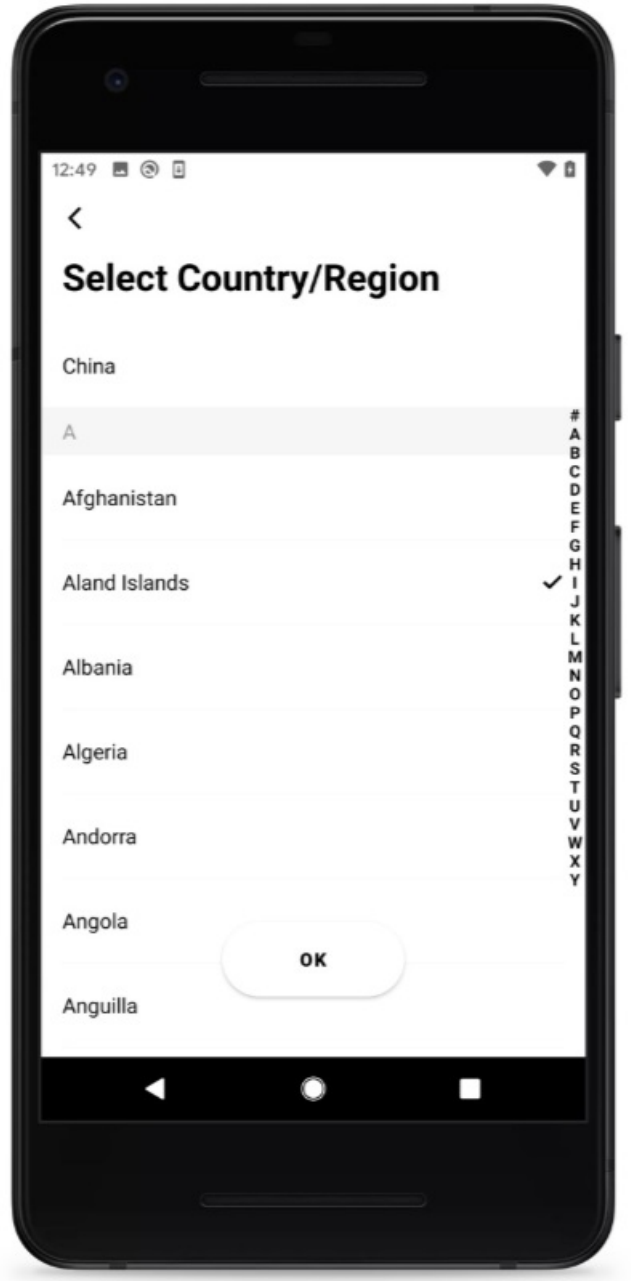
```
./gradlew clean build
```

> **说明**
> - 请配置JDK为1.8及以上版本。
> - 3.0.0版本的打包插件（com.android.tools.build:gradle:3.0.0），需要Android Studio 3.0以上的版本支持，且需要访问Google的仓库，请注意网络环境。

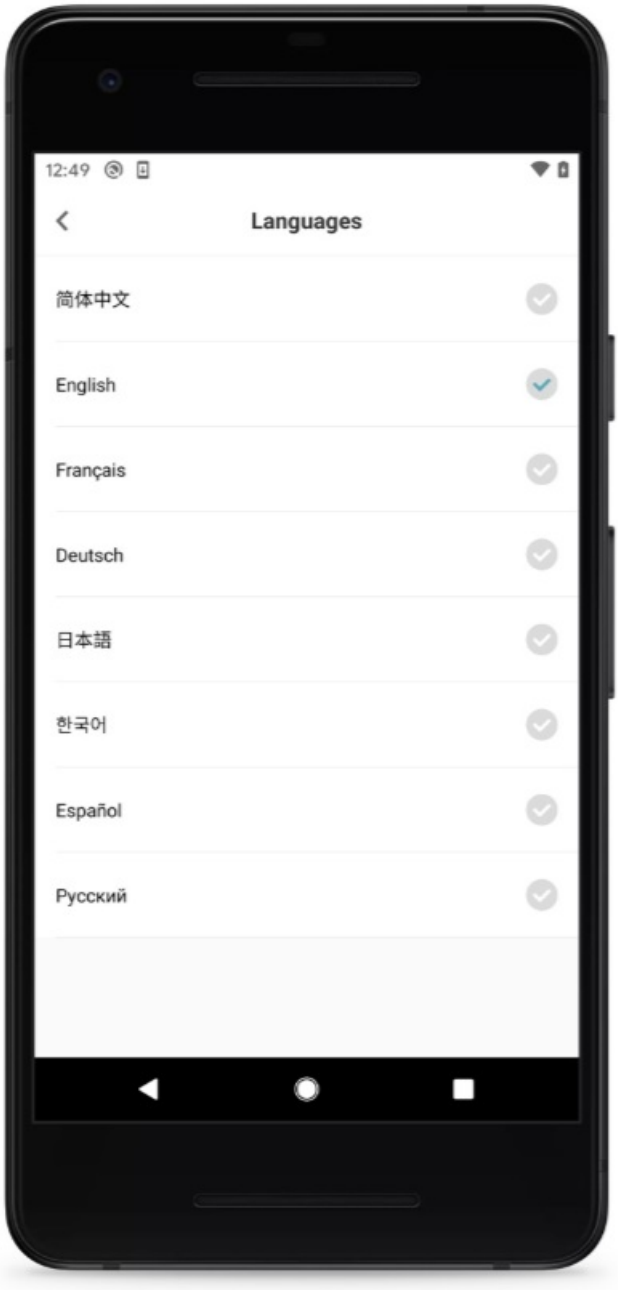
Demo App业务逻辑介绍

默认Demo App配置为官方项目和默认支持的几类设备。

1. 选择国家或地区。 注册账号、三方账号登录前，您须先选择登录的国家或地区。



2. 注册或登录账号。 App启动后会显示如下登录页界面。
   i. 单击**免费注册**，弹出注册方式。
   ii. 选择要注册的国家地点。
   iii. 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。
   iv. 输入收到的校验码，单击**下一步**。
   v. 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

5. 控制虚拟设备。
   i. 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。

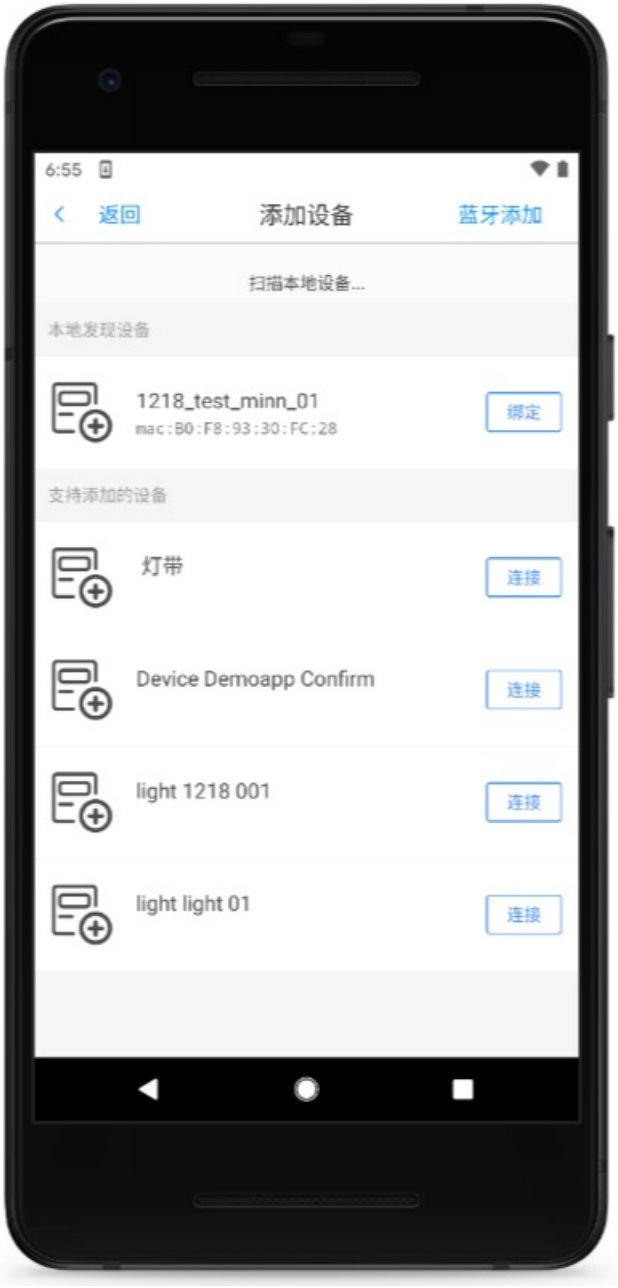
```
/**
 * 将自己项目内的pk填入，可以体验虚拟设备
 */
//String[] pks = new String[]{"a1AzoSi5TMc", "a1B6cFQldpm", "a1XoFUJWkPr", "a1nZ7Kq7AG1", "a1OX20YJSLk"};
String[] pks = new String[]{};
```

ii. 单击任意一个设备，进入相应的设备面板界面。

6. 添加设备。

i. 单击首页右上角加号。

ii. 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.getDeviceName());
}
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。

11.5.3. 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。

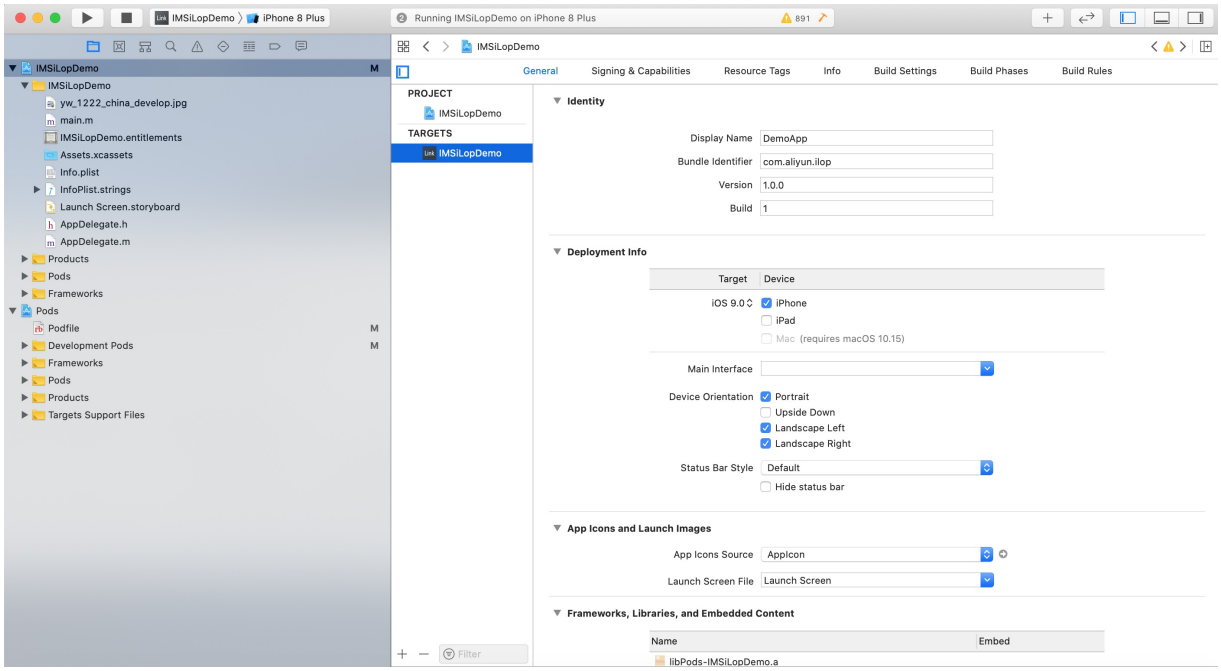
前提条件

- 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。详细操作请参见创建自有App。

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及以上版本（Cocoapods 1.2.0版本您需要在*Profile*文件中，将ToolboxForMac和GTMOAuth2两个SDK的注释去掉）。

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

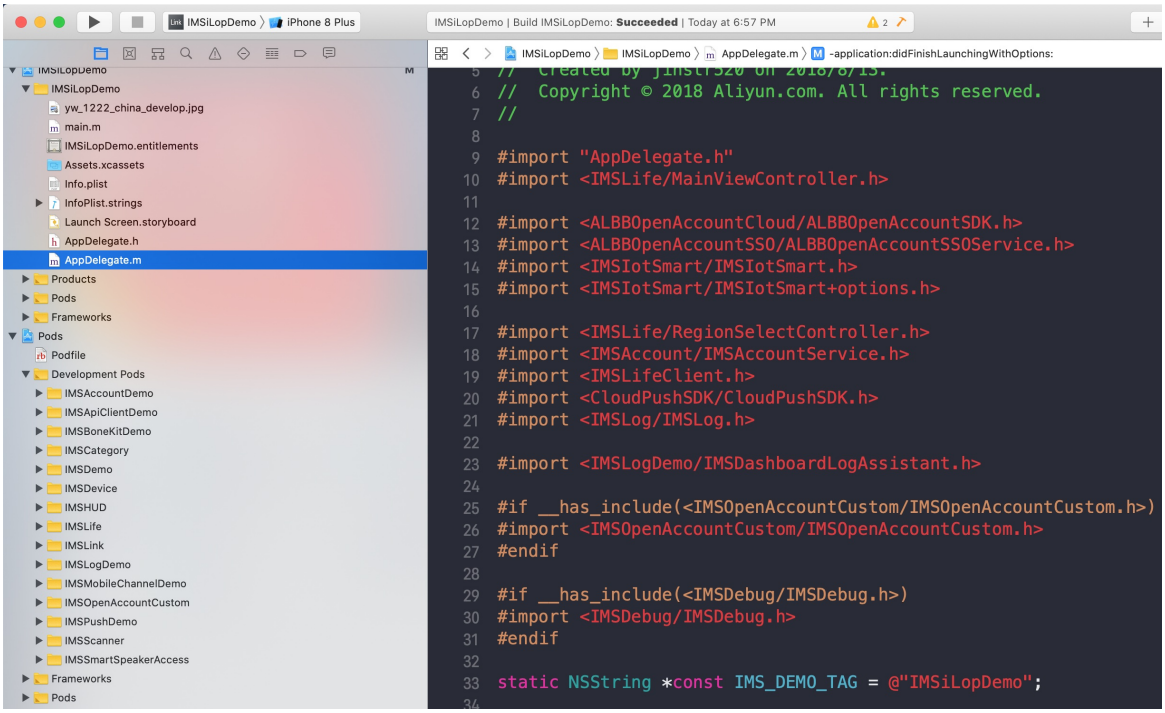
整个Demo App结构上采用CocoaPods管理业务模块，主工程有安全图片，启动页面和AppDelegate实现，如下图所示。

*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，详细请参见创建自有App。
2. 集成安全图片。详细请参见集成安全图片。
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。

Demo App业务逻辑介绍

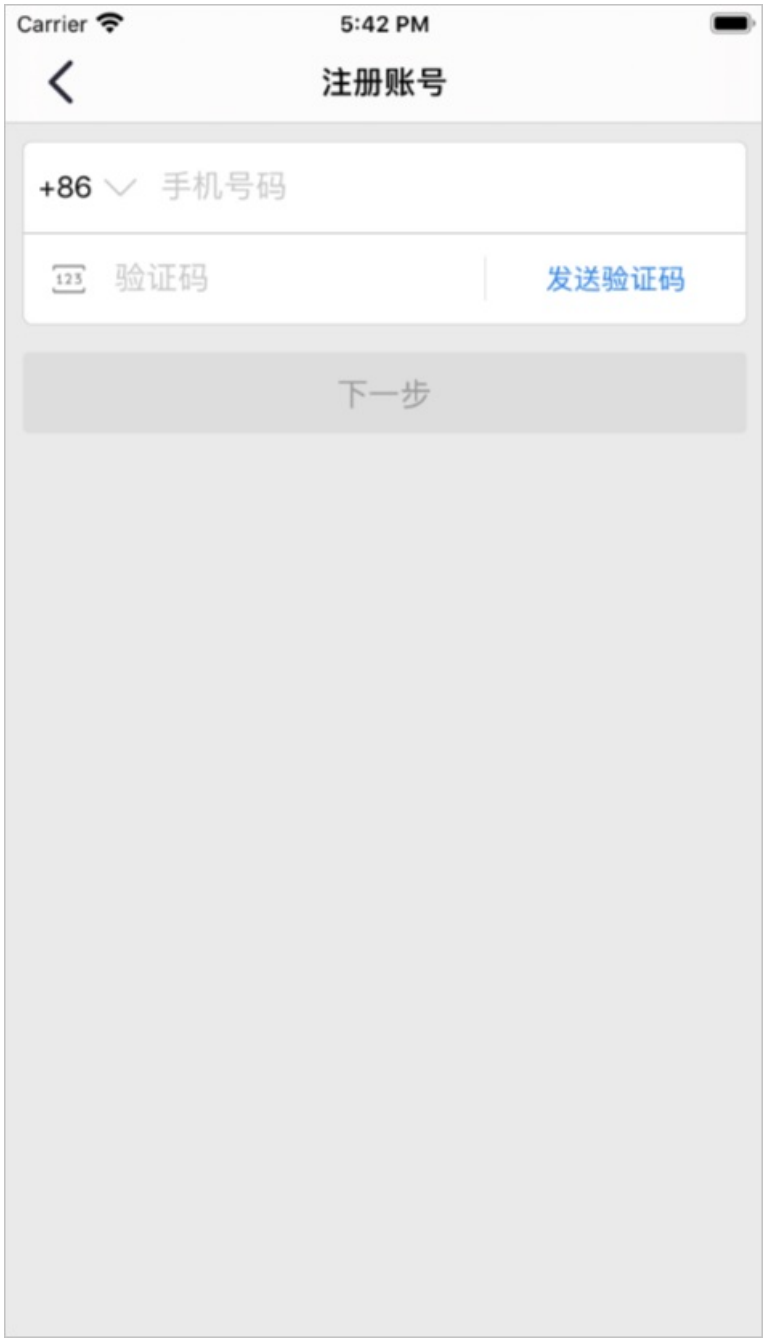
1. 登录自有App。

i. 启动App，选择国家/区域。

ii. 选择一个国家/区域后，进入登录界面。

iii. 单击**免费注册**，进入注册页面。

iv. 输入要注册的手机号码，并单击下方**发送短信校验码**。



v. 在短信校验码区域输入收到的校验码，单击**下一步**。

vi. 设置登录密码，并完成登录。

2. 设置调试界面。

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

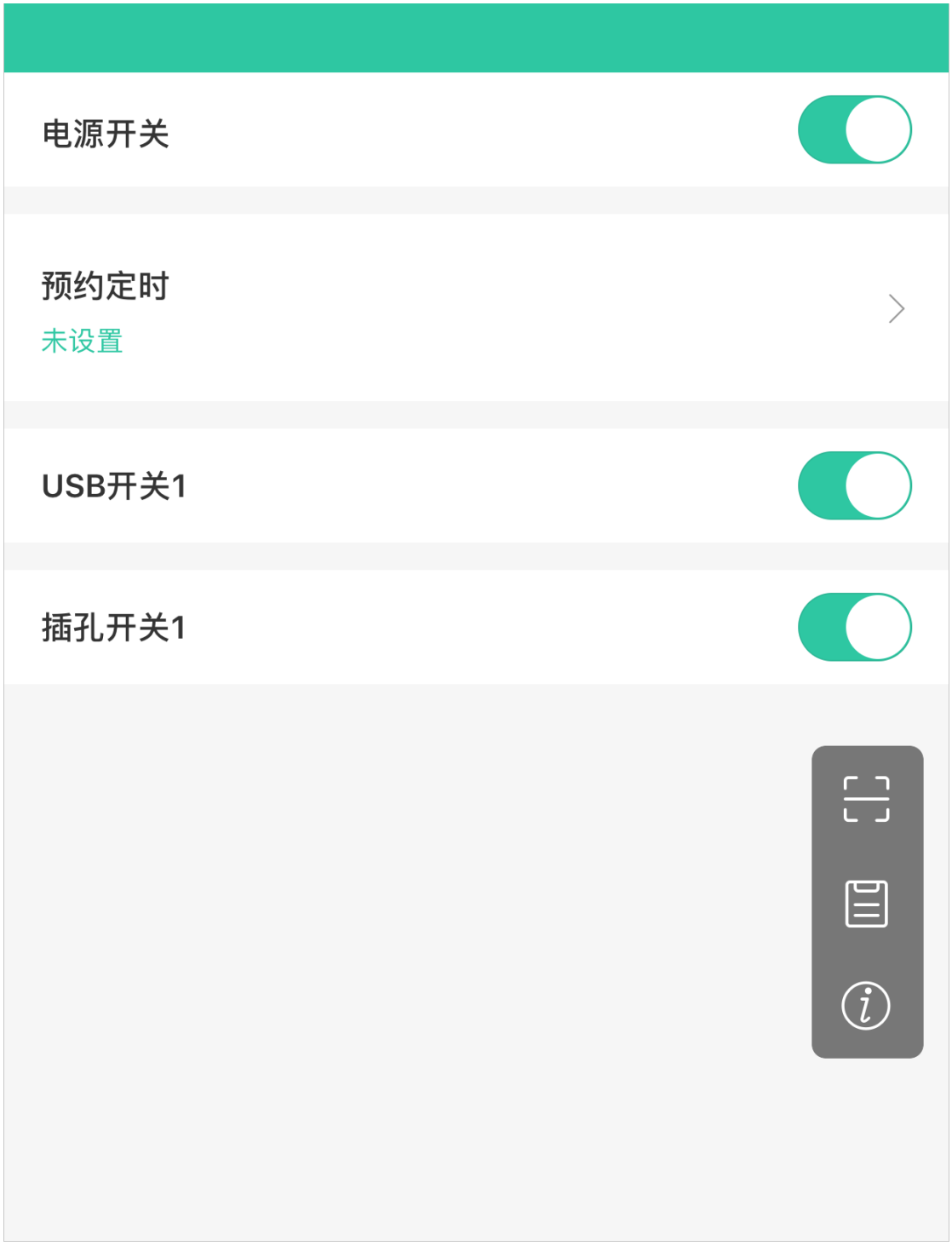
文件名	界面展示名
SDKEntryMobileChannel	长连接通道
SDKEntryAccount	账号和用户
SDKEntryPush	移动应用推送

3. 控制默认的虚拟设备。
   i. 进入主页面，如下图所示。

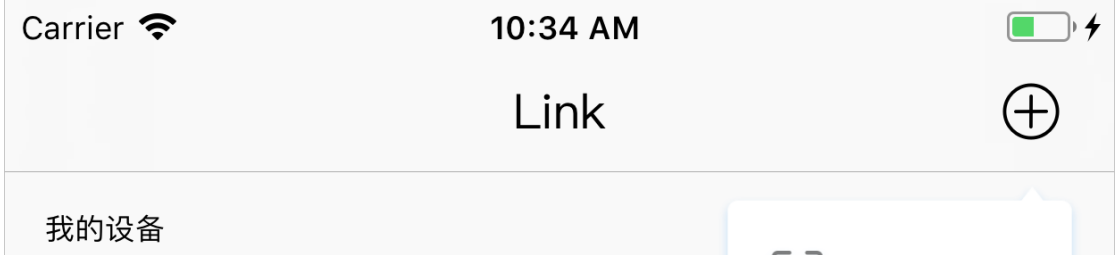
上面显示的4个虚拟设备，并不是默认创建的，需要自己更改代码，在LinkViewController.m里面，将自己项目内创建的产品productKey填入下面的supportVirtualList变量中，再编译运行，才可以创建和体验虚拟设备。

```
- (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 = @[];
}
```

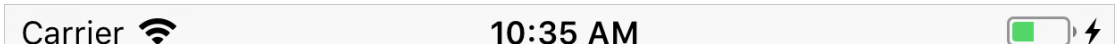
ii. 单击任意一个设备，进入该设备的面板页面。



4. 添加设备。
   i. 单击首页右上角加号，并选择**添加设备**。



ii. 选择添加设备，进入添加设备页面。




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。

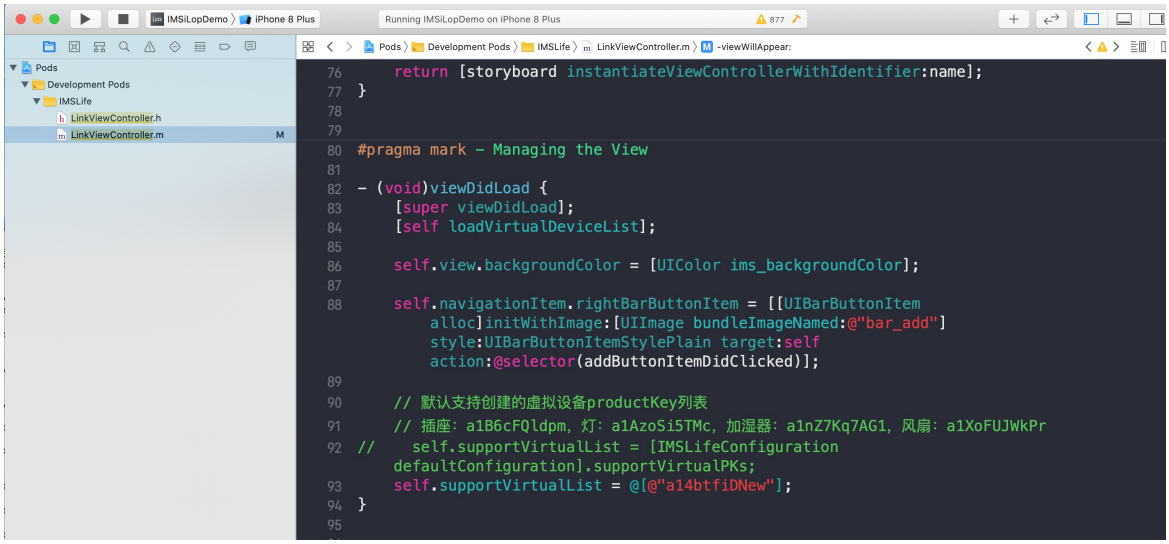
- 通过产品列表的形式进行配置，如上图支持添加的设备。
- 支持零配配网方案的设备，发现到本地设备则会显示出来，如上图本地发现设备。

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。