

目录

开放平台文档中心	1.1
1. 平台概述	1.2
2. 产品开发	1.3
3. APP开发	1.4
3.1 应用管理	1.4.1
3.1.1 新建应用	1.4.1.1
3.1.1.1 普通应用	1.4.1.1.1
3.1.1.2 云对接服务	1.4.1.1.2
3.1.2 应用模式	1.4.1.2
3.1.2.1 开发模式	1.4.1.2.1
3.1.2.2 配置模式	1.4.1.2.2
3.1.3 关联产品	1.4.1.3
3.2 APP SDK	1.4.2
3.2.1 android SDK	1.4.2.1
3.2.2 iOS SDK	1.4.2.2
3.2.3 H5 SDK	1.4.2.3
3.3 APP 开源框架	1.4.3
3.3.1 android APP 开源框架	1.4.3.1
4. 硬件开发	1.5
4.1 硬件接入准备工作	1.5.1
4.2 硬件MCU及源代码	1.5.2
4.2.1 开发板原理图	1.5.2.1
4.2.2 mcu源码框架	1.5.2.2
4.3 硬件通讯模组	1.5.2.3
4.3.1 Clife Agent简介	1.5.2.3.1
4.3.2 通信模组应用指导	1.5.2.3.2
4.3.2.1 Wi-Fi模组	1.5.2.3.2.1
4.3.2.2 蓝牙模组	1.5.2.3.2.1.1
4.3.2.3 NB-IoT模组	1.5.2.3.2.1.2
4.3.2.4 GPRS模组	1.5.2.3.2.1.3
4.3.2.5 Zigbee模组	1.5.2.3.2.1.4
4.3.3 通信模组调试日志获取教程	1.5.2.3.3
4.3.4 常见问题	1.5.2.3.4

5. 云API	1.6
6. 数据服务	1.7
6.1 概况	1.7.1
6.2 数据统计	1.7.2
6.3 数据分析	1.7.3
6.4 实时监控	1.7.4
6.5 设备预警	1.7.5
6.6 书签管理	1.7.6
6.7 运营管理	1.7.7
7. 推送服务	1.8
7.1 Android 推送服务	1.8.1

- [开放平台文档中心](#)
- [文档编写](#)
- [文档目录](#)

开放平台文档中心

文档编写

- 文档统一放在 `/source` 文件夹里，中文文档放在 `/source/zh-cn` 目录下，英文文档放在 `/source/en-us` 目录下(暂不支持英文文档)。
- 增加一个类别需要在对应目录下新建一个文件夹，并在文件夹下新建`package-info`文本文件，说明该文件夹中文档类别和内容，以便日后维护。
- 文档文件为`.md`后缀的`markdown`文件，文件名（及文件夹名）统一规范为英文；为了方便文档管理，中英文文档使用相同的文件夹路径及文件名
- 文档编写格式待定
- 文档编写的过程中用到的图片，放入 `/images` 文件夹中
- 注意：不要提交构建好的文件夹`_book`

文档目录

- 文档目录存放于 `SUMMARY.md` 文件中。根据现有文档类别，见文档目录结构制定如下（后期可能会调整文档目录存放位置，但结构不变）：
 - `/source`
 - `zh-cn`
 - `overview` --平台概述
 - `index.md`
 - `product` --产品开发
 - `index.md`
 - `app` --app开发
 - `application`--应用管理
 - `application.md`
 - `openframework`--开源框架
 - `open_framework.md`
 - `SDK`--SDK
 - `index.md` -- SDK首页
 - `android.md`
 - `ios.md`
 - `H5.md`
 - `device` --硬件开发
 - `cloudAPI` --云API
 - `dataservice` --数据服务
 - `pushservice` --推送服务

- 文件夹下多于1个md文件的类目，均有一个index.md文件作为该类目的目录文件
- 若有修改文件目录结构（或新增新文件夹、文件），在相应目录下建立package-info说明文档
- 添加的新文件夹或文件，需要在 SUMMARY.md 中加入相应目录

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 15:55:51

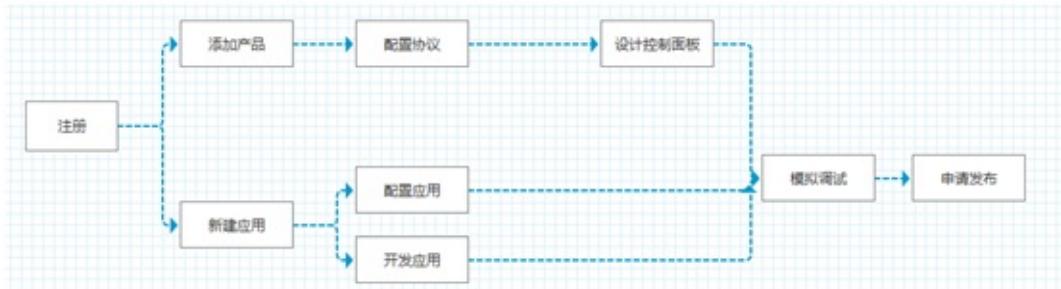
- 平台概述
 - 1.平台介绍
 - 2.接入流程
 - 3.联系我们
 - 4.成为开发者

平台概述

1. 平台介绍

以大数据为媒介，连接一切价值单元与要素，全周期、全链条、全方位，服务于新一代个人与家庭生活场景集群，并以家庭大数据平台为内核，支撑全社会按新秩序运行的IoT平台。CLIFE专注于提供智能云服务及物联网的软硬件解决方案，帮助传统硬件厂商产品升级，快速实现硬件智能化。在硬件实现智能化的过程中，除了硬件外，我们还提供了：智能云平台、手机APP、联网模块的智能化服务，每个领域都有专业的团队。在cliffe的支持下，厂商仅需要关注自身硬件，cliffe提供智能平台的接入、数据分析、手机APP开源，以最小的投入实现硬件产品最高价值。

2. 接入流程



3. 联系我们

深圳和而泰智能控制股份有限公司

地址：[公司总部] 深圳市南山区高新园南区科技南十路6号深圳航天科技研究院D座10楼

电话：0755-86119291

传真：0755-2627137

地址：[工业园] 深圳市光明新区公明街道田寮社区田寮路和而泰工业园

电话：0755-26727188

股票代码：002402



4. 成为开发者

用户在cliffe开放平台申请成为平台开发者，开发者可以在平台上创建及管理账户下的设备和应用。账户可分为个人账户和公司账户，在进入个人中心之后可进行维护。平台更开设了账号体系，账户可指定创建子账户，同一账号下的不同子账号可以进行分权限管理平台上的设备、应用。

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间：2017-11-14 15:32:03

- 产品开发
 - 1. 创建应用
 - 2. 协议配置
 - 3. H5控制页面
 - 4. 应用开发
 - 配置APP
 - APP开发
 - 5. 创建产品
 - 5.1 第一步，填写产品信息
 - 5.2 第二步，选择技术方案
 - 5.3 第三步，完善产品信息
 - 5.4 协议配置
 - 5.5 H5页面配置
 - 5.6 模拟调试
 - 6. 拓展功能
 - 6.1 接入方案
 - 6.2 联网配置
 - 6.3 固件升级
 - 6.4 发布产品

产品开发

1. 创建应用

开启设备智能化的第一步便是在平台上创建产品。产品创建成功，该产品可以在clife平台注册生成，自动生成APP ID 创建产品引导

2. 协议配置

协议配置 协议配置指的是配置产品的功能，通过不同的数据类型进行定义表示。例如最简单的设备“开关”，具备功能为：开启、关闭，将其抽象为一个协议为布尔型，0表示关闭，1表示开启。协议配置是一个产品最重要的属性，因此产品智能化的第一步，就是明确产品功能，并逐一在平台上进行创建。产品的APP控制界面将可以根据所配置的协议自动生成。如何创建功能点详见：协议配置

3. H5控制页面

提供多种App控制界面模板供选择，也可使用自由配置模板，即可以自己来搭建、设计自己的产品App控制界面，可以最快速的实现APP设备控制界面，真正实现APP开发0投入。如何配置H5控制页面详见：H5控制面板设计

4. 应用开发

配置APP

平台提供APP的快速开发功能，针对app的色调、字体、功能上有了标准化代码，开发者直接在平台上进行配置，节省开发成本。

APP开发

Clife开放APP源代码供开发者下载，开发者可在代码基础上进行修改，可修改为更符合自身需求的app，节约开发人员精力。如何开发app详见：APP开发

5. 创建产品

5.1 第一步，填写产品信息

在产品管理中，点击新建应用，选择产品的分类，输入产品名称和型号。注意：产品的分类决定了后续的协议配置和APP模块提供，请谨慎选择。如果分类中没有你需要的选项，请联系我们。

The screenshot shows a form titled '添加产品'. It includes dropdown menus for '分类' (Category) set to '冰箱-1' and '单门冰箱-1001' (Single-door refrigerator-1001). Below these are input fields for '产品名称' (Product Name) and '产品型号' (Product Model), both with placeholder text '必填项，输入在20字符以内' (Required, input within 20 characters). A note at the top right says: '如果找不到你需要添加的产品大类和产品小类，请联系管理员tanggf@szhittech.com，来信请附上需要添加的产品详情' (If you can't find the product category or sub-category you need to add, please contact the administrator tanggf@szhittech.com, attach the product details in the email).

5.2 第二步，选择技术方案

根据产品模组选择技术方案，不同的技术方案将有不同的联网模式



5.3 第三步，完善产品信息

创建产品成功，系统会根据产品类别等信息自动生成唯一的产品ID和设备编码，可以对该产品进行信息完善。.jpg)

5.4 协议配置

根据产品功能点进行协议的配置，协议中填写字节，参数类型，设定参数，添加保存即为一条协议。
注意：产品协议位数必须为16的倍数。

The screenshot shows the 'Protocol Configuration' interface. At the top, there are tabs for 'Control Data', 'Run Data', 'Fault Data', and 'Configure Data'. On the right, there are buttons for 'Export Protocol' and 'Add Custom Data'. A note says 'You can increase the byte length by 13 to reach the closest configuration condition.' Below is a table with two rows:

	数据名称	数据标识	数据单位	数据类型	数据长度	数据属性	保留字	操作
/	开关	onoff	字节	字符串	1	1-开, 2-关	否	编辑 删除
/	功能变更	updateFlag	字节	十六进制	1		否	

Below the table is a 'Edit Data' dialog box:

- 数据单位:** 字节 (selected)
- 数据名称:** 字节中文名, *必填项
- 数据标识:** 字母或数字, *必填项
- 数据长度:** 正整数, *必填项
- 参数类型:** 字符型 数值型 浮点型
- 参数设定:** (empty)
- + 添加参数**
- 保留字:** 是 否
- 数据备注:** 选项项 (empty)
- 保存** **取消**

5.4.1 标准化协议模板

根据选择的分类推荐标准化协议模板，可根据需求选择需要的协议内容，省去配置成本

The screenshot shows the 'Protocol Template Management' interface. It has a sidebar with 'Select Data' and several filter categories:

- 选择数据:**
 - 全部数据
 - 控制数据
 - 冷藏室温度设定
 - 冷冻室温度设定
 - 工作模式设定
 - 重锁设定
 - 冷藏室开关设定
 - 冷冻室开关设定
 - 除霜设定
 - 运行数据
 - 环境温度
 - PCBA环境温度
 - 冷藏室温度
 - 冷藏室补偿温度
 - 冷藏室盘管温度
 - 冷冻室温度
 - 冷冻室补偿温度
 - 冷冻室盘管温度
 - 压机线包温度

保存 **取消**

5.5 H5 页面配置

产品实现智能化，需要由终端来进行控制。开发者可直接上传H5文件或者进入自助设计的页面，根据

新建HTML5模板

模板名称：

版本号： 请输入点号隔开的数字，例如“1.0.1”

启用设计 (须将本句不小于45的Chrome浏览器支持) 进入源版设计工具进行H5页面的DIY

上传H5文件 目前平台的JS SDK仅对内部用户进行开放，外部用户敬请期待，文件大小不超过10MB

说 明：

需求选择控件进行设计。

5.6 模拟调试

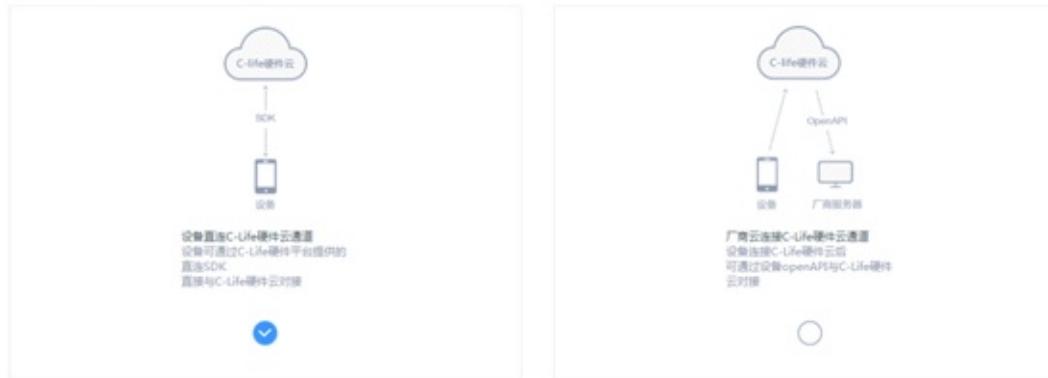
从设备调试页面进入，需要关联账号与设备mac地址，针对性的进行调试。开发者可以通过模拟调试来进行设备协议的调试，通过查看上行下行数据，以校验设备协议的配置是否正确。

6. 拓展功能

6.1 接入方案

接入平台的方案分别有设备直连C-Life硬件云通道和厂商云连接C-Life硬件云通道，开发者可根据自身需求，选择合适的接入方案，这决定了设备联网的数据通道。

请选择方案



6.2 联网配置

用户在操作APP绑定硬件时可能对绑定方式和流程不太熟悉，在本页面配置操作指引，提升APP用户体验。

联网配置

若您的产品支持wifi或蓝牙配置能力，请设置配置时的用户指引，以便用户正确绑定，例图如下



绑定指引

1. 长按开关键几秒
2. wifi指示灯开始闪烁，设备进入绑定状态，如果没有，请重复步骤1
3. 点击绑定设备按钮

 不设置 不设置任何联网指引

6.3 固件升级

用于管理设备固件，可针对单个或批量进行固件升级

新增固件

版本类型 :	请选择
固件类型 :	请选择
总版本号	只能包含字母,数字,下划线,短横线
内部版本	请输入正整数
外部版本	只能包含字母,数字,下划线,短横线
版本名称	只能包含字母、数字、符号，24~
升级附件	选择文件
升级要求	普通
发布内容	

[指定升级](#) [全部升级](#) [指定mac升级](#)

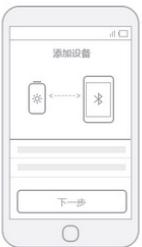
[确定](#) [取消](#)

6.4发布产品

完成产品的创建流程，即可申请发布到C家，在C家APP上可以对设备进行管理

发布到C家


C家
[申请发布](#)

 1.选择添加设备方式
 2.wifi/蓝牙配对
 3.查看设备详情
 4.进入设备控制界面

间： 2017-11-14 15:35:00

1.1 android SDK

1.2 iOS SDK

1.3 H5 SDK

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-10-30 13:49:37

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-10-30 13:49:37

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-10-30 13:49:37

- Android SDK 概述
 - 1.SDK功能简介
 - 2.相关名词定义
 - 2.1.applId和appSecret
 - 2.2.硬件模组
 - 2.3.productId
 - 2.4.deviceId
 - 3.SDK 快速开发，相关第三方库支持
 - 3.1.RxJava 函数式编程
 - 3.2.RxBus 事件传递总线
 - 3.3.retrofit+okhttp 网络库的支持
 - 3.4.ActiveAndroid 数据库支持
 - 3.5.X5内核 浏览服务的支持
 - 3.6.第三方社交平台服务的支持
 - 4.集成准备
 - 4.1.注册开放平台账号
 - 4.2.下载SDK终端DEMO 请前往下载中心下载最新SDK包。
 - 4.3.在Android Studio上集成SDK，配置项目根目录build.gradle如下：
 - 4.4.引用SDK到工程中
 - 4.5.配置AndroidManifest.xml 添加权限
- SDK集成流程
 - 1.SDK集成流程简介
 - 2.SDK初始化
 - 3.授权登录和用户模块
 - 3.1.非私有服务器授权登录
 - 3.2.云云对接用户授权登录
 - 3.3.退出登录
 - 3.4.判断登录状态
 - 3.5.异地登录
 - 3.6.获取用户信息
 - 3.7.修改密码
 - 4.设备绑定模块（添加设备）
 - 5.设备管理
 - 5.1.设备model说明
 - 5.2.获取设备列表
 - 5.3.删除设备
 - 5.4.设备分享
 - 6.设备控制
 - 7.其他接口
 - 7.1.意见反馈
 - 7.2.消息模块
- 第三方平台服务的集成（登录和分享）
- 通用的业务接口
- 全局返回码

- H5+Native混合框架
 - 1.H5开发框架
 - 2.Android和H5通讯流程图
 - 3.H5设备控制集成流程
 - 3.1.上传H5开发包
 - 3.2.初始化webView和H5交互接口
 - 3.3.加载H5插件
 - 3.4.监听H5插件加载成功与失败
 - 3.5.上传设备数据给H5
 - 3.6.退出释放资源

Android SDK 概述

1.SDK功能简介

clife开放平台（以下简称开放平台）设备接入的SDK封装了clife对外开放的服务接口，以及手机与智能硬件通讯接口。包括用户模块，设备绑定模块，设备控制模块和其他的开放平台接口。开发者不需要关注这些模块的具体内部逻辑，只需要根据自己的业务需求编写界面和调用SDK接口就可以完成APP的快速开发。下面是SDK的基础架构图：



2.相关名词定义

2.1.appId和appSecret

开放平台app的应用标识和密钥。开发者在开放平台接入设备创建APP的时候，后台自动生成appId和appSecret，用于初始化SDK。

2.2.硬件模组

这里的硬件模组是指的WIFI模组。在开放平台创建WIFI产品的时候需要指定设备的WIFI模组。开发者需要在APP中先注册WIFI模组才可以进行设备绑定。

2.3.productId

产品唯一标识。开发平台创建产品，自动分配。

2.4.deviceId

设备的唯一标识。设备绑定成功，自动分配。

3.SDK 快速开发，相关第三方库支持

3.1.RxJava 函数式编程

开放平台SDK集成了RxJava，并且使用RxJava封装了Android 6.0+的动态权限申请接口和网络请求库等。

3.2.RxBus 事件传递总线

开放平台SDK提供了RxBus事件总线的支持，用于事件的发布和订阅，实现数据的传递。

RxBus事件的发布：

```
//发布 HetCodeConstants.Login.LOGIN_SUCCESS 事件  
RxManage.getInstance().post(HetCodeConstants.Login.LOGIN_SUCCESS, object);
```

RxBus事件的订阅：

```
RxManage.getInstance().register(HetCodeConstants.Login.LOGIN_SUCCESS, o -> {  
    //订阅 HetCodeConstants.Login.LOGIN_SUCCESS 事件  
});
```

RxBus事件的取消订阅：

```
RxManage.getInstance().unregister(HetCodeConstants.Login.LOGIN_SUCCESS);
```

3.3.retrofit+okhttp 网络库的支持

开放平台SDK集成了retrofit+okhttp的网络库，用于请求服务器数据。

3.4.ActiveAndroid 数据库支持

开放平台SDK集成了 ActiveAndroid 第三方的轻量级的数据库,方便轻量级数据的存取。

3.5.X5内核 浏览服务的支持

开放平台SDK集成了X5内核的浏览服务，来提高移动端webview的加载性能和兼容性。

使用实例：

```
<com.tencent.smstt.sdk.WebView
    id="@+id/forum_context"
    layout_width="fill_parent"
    layout_height="fill_parent"
    paddingLeft="5dp"
    paddingRight="5dp" />
```

注意：将源码和XML里的系统包和类替换为SDK里的包和类，如：

android.webkit.WebChromeClient 替换成 com.tencent.smstt.sdk.WebChromeClient 。

3.6.第三方社交平台服务的支持

SDK集成了第三方社交平台服务库，支持微信、qq和新浪微博的分享和登录。详细使用请查看 [第三方平台服务的集成（登录和分享）](#)

4.集成准备

4.1.注册开放平台账号

通过<https://open.clife.cn/#/home>注册一个开发者账号。登录到开放平台创建应用完善详细资料。此部分请参考《clife开发平台使用手册》。创建产品之后创建APP获取到后台分配的appId和appSecret。

4.2.下载SDK终端DEMO 请前往下载中心下载最新SDK包。

4.3.在Android Studio上集成SDK，配置项目根目录build.gradle如下：

```
allprojects {
    repositories {
        jcenter()
        //clife对外仓库
        maven { url "https://oss.sonatype.org/content/repositories/snapshots/" }
    }
}

dependencies {
```

```
//clifessdk库
compile 'com.github.szhittech:HetOpenSdk:1.0.7-SNAPSHOT'
}
```

4.4.引用SDK到工程中

```
dependencies {
    compile fileTree(include: ['*.jar'], dir: 'libs')
    testCompile 'junit:junit:4.12'
    compile project(':HetOpenLib')
    compile 'com.android.support:appcompat-v7:23.0.1'
    compile 'com.android.support:support-v4:23.0.1'
    compile 'com.android.support:design:23.1.1'
    compile 'com.facebook.shimmer:shimmer:0.1.0@aar'
    compile 'com.facebook.fresco:fresco:0.8.1+'
    compile 'com.google.code.gson:gson:2.5'
    compile 'org.greenrobot:eventbus:3.0.0'
    compile 'com.readystatesoftware.systembartint:systembartint:1.0.3'
    compile 'com.github.szhittech:hetrycyclersdk:1.0.9-SNAPSHOT'
    //乐鑫信息科技(esptouchmodule) 模组ID: 7
    compile 'com.github.szhittech:esptouchmodule:1.0.1-SNAPSHOT'
    //clifeAP绑定(hetapmodule) 模组ID: 28
    compile 'com.github.szhittech:hetapmodule:1.0.1-SNAPSHOT'
    //clifessmartlink绑定(在庆科基础上修改 hetsmartlink) 模组ID: 10
    compile 'com.github.szhittech:hetsmartlink:1.0.1-SNAPSHOT'
    //科中龙(realtekmodule) 模组ID: 4
    compile 'com.github.szhittech:realtekmodule:1.0.1-SNAPSHOT'
    //新力维_NL6621底层库(xlwmodule) 模组ID: 6
    compile 'com.github.szhittech:xlwmodule:1.0.1-SNAPSHOT'
    //双驰达(sctechmodule) 模组ID: 15
    compile 'com.github.szhittech:sctechmodule:1.0.1-SNAPSHOT'
    //信驰达_MTK7681底层库(elianmodule) 模组ID: 5
    compile 'com.github.szhittech:elianmodule:1.0.1-SNAPSHOT'
    //Marvell(marvellmodule) 模组ID: v1=8, v2=27
    compile 'com.github.szhittech:marvellmodule:1.0.1-SNAPSHOT'
    //博通(cooeemodule) 模组ID: 20
    compile 'com.github.szhittech:cooeemodule:1.0.1-SNAPSHOT'
    //二维码扫描
    compile 'com.google.zxing:core:3.3.0'
    compile 'cn.bingoogolapple:bga-qrcodecore:1.1.8@aar'
    compile 'cn.bingoogolapple:bga-zxing:1.1.8@aar'
}
```

引用SDK，根据自己硬件的模组选择对应的库。

4.5.配置AndroidManifest.xml 添加权限

请将下面权限配置代码复制到 AndroidManifest.xml 文件中：

```
<uses-permission          name="android.permission.INTERNET" />
<uses-permission          name="android.permission.READ_PHONE_STATE" />
<uses-permission          name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission          name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission          name="android.permission.WRITE_SETTINGS" />
<uses-permission          name="android.permission.VIBRATE" />
```

```

<uses-permission name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission name="android.permission.DISABLE_KEYGUARD" />
<uses-permission name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission name="android.permission.CAMERA" />
<uses-permission name="android.permission.WAKE_LOCK" />

```

权限说明：

权限	用途
INTERNET	允许应用可以访问网络
READ_PHONE_STATE	允许应用访问手机状态
ACCESS_NETWORK_STATE	允许应用获取网络信息状态，如当前的网络连接是否有效
RECEIVE_BOOT_COMPLETED	系统广播权限
WRITE_SETTINGS	允许应用读写系统设置项
VIBRATE	允许应用震动
WRITE_EXTERNAL_STORAGE	允许应用写入外部存储
DISABLE_KEYGUARD	允许程序禁用键盘锁
ACCESS_COARSE_LOCATION	允许程序访问位置信息
ACCESS_WIFI_STATE	允许程序访问Wi-Fi网络状态信息
CAMERA	允许程序打开相机
WAKE_LOCK	允许应用在手机屏幕关闭后后台进程仍然运行

4.6.Android6.0系统文件读写权限设置

Android 6.0+新增了运行时权限动态检测，敏感权限必须要动态申请。开发者可以使用SDK提供的RxPermissions来动态申请权限。

```

if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
    RxPermissions.getInstance(AppDelegate.getContext())
        .request(Manifest.permission.READ_PHONE_STATE,
                 Manifest.permission.WRITE_EXTERNAL_STORAGE)
    )
    .subscribe(grant -> {
        if (!grant) {//申请权限成功与否
            finish();
        }
    });
}

```

SDK集成流程

1.SDK集成流程简介

- 第一步：SDK初始化(SDK的日志信息开关、环境设置、app主题的配置信息)
 第二步：授权登录和用户模块(授权登录、获取用户信息、修改密码)
 第三步：设备绑定(WIFI设备的AP绑定、smartLink绑定和蓝牙的扫描绑定)
 第四步：设备管理(获取设备列表、删除设备、设备分享)
 第五步：设备控制(wifi和蓝牙设备的控制)

2.SDK初始化

第一步：在application里面初始化SDK。设置SDK的日志信息开关、环境设置、app主题的配置信息等

```
/*
 * 配置SDK
 *
 * @param appId
 * @param appSecret
 */
private void configApplication(String appId, String appSecret) {
    ConfigModel configModel = new ConfigModel();
    configModel.setLog(true); //是否开启log信息
    configModel.setHost(HetCodeConstants.TYPE_PRODUCE_HOST); //环境设置
    configModel.setH5UIconfig(UIJsonConfig.getInstance(this).getJsonString(UIJsonConfig.fileName, this));
    //配置第三方登录
    mLoginDelegate = new HetSdkThirdDelegate.Builder(this)
        .registerQQ(UIJsonConfig.getTencentAppID())
        .registerWeixin(UIJsonConfig.getWechatAppID(), UIJsonConfig.getWechatAppSecret())
        .registerSinaWeibo(UIJsonConfig.getSinaAppID(), UIJsonConfig.getSinaAppSecret(),
        this.mSinaRedirectURL)
        .create();
    HetSdk.getInstance().init(this, appId, appSecret, configModel);
}
```

- 1、appId、appSecret可以在开放平台创建的应用的应用详情里查看。
- 2、HetSdkThirdDelegate 配置第三方社交平台（微信、QQ、新浪微博登录和分享），需要的开发者自行配置，不需要的可以不要。关于第三方登录的集成请参考 **3.3 (SDK第三方登录的集成)**。
- 3、configModel.setH5UIconfig 配置授权登录页面主题样式；通过参数定义的JSON字符串来进行配置，例如demoAPP是通过assets/h5UIConfig.json这个文件来组装JSON字符串的。

接口调用请求说明

SDK初始化接口 HetSdk.getInstance().init ()

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
context	是	Context	上下文常量
secret	是	string	用户密匙
configModel	是	ConfigModel	初始配置

configModel说明

字段名称	字段类型	字段说明
isLog	boolean	是否开启调试信息
host	int	网络环境设置: 0x01: 正式 0x02: 预发布 0x03:内网 0x04: 测试环境
H5UIconfig	String	APP初始配置

H5UIconfig配置说明

SDK的授权登录页面样式可以通过JSON参数来配置，包括是否需要第三方登录，登录页面的样式等。可以参考SDK的DEMO工程通过assets/h5UIConfig.json的配置，配置详情：

```
{
  "app_id": "your_app_id",
  "app_secret": "your_app_app_secret",
  "navBackgroundColor": "FF3285FF",
  "navTitleColor": "FFFFFF",
  "logoshow": true,
  "loginBtnBackgroundColor": "FFFFFF",
  "loginBtnBorderColor": "FF5BA532",
  "loginBtnFontColor": "FF000000",
  "weiboLogin": true,
  "weixinLogin": true,
  "qqLogin": true,
  "copyRight": "",
  "privacyPolicy": "",
  "SMSTempalte": "",
  "tencent_app_id": "your_tencent_app_id",
  "wechat_app_id": "your_wechat_app_id",
  "wechat_app_secret": "your_wechat_app_secret",
  "sina_app_id": "your_sina_app_id",
  "sina_app_secret": "your_sina_app_secret"
}
```

字段说明：

字段名称	字段类型	字段说明
app_id	String	APP标识(开放平台创建应用申请获得)
app_secret	String	APP密钥(开放平台创建应用申请获得)
navBackgroundColor	String	授权登录标题栏的背景色 (例如： FF3285FF)
navTitleColor	String	授权登录标题栏的文字颜色 (例如： FFFFFFFF)
logoshow	boolean	授权登录页面的安全登录的图标是否显示 (false, true)
loginBtnBackgroundColor	String	授权登录页面登录按钮背景颜色 (例如： FFFFFFFF)
loginBtnBorderColor	String	授权登录页面登录按钮背景边框颜色 (例如： FF5BA532)

loginBtnFontColor	String	授权登录页面登录按钮文字颜色 (例如: FF000000)
weiboLogin	boolean	授权登录页面, 是否显示微博登录 (false, true)
weixinLogin	boolean	授权登录页面, 是否显示微信登录 (false, true)
qqLogin	boolean	授权登录页面, 是否显示QQ登录 (false, true)
tencent_app_id	String	QQ登录的应用标识 (腾讯开放平台创建应用申请获得)
wechat_app_id	String	微信登录的应用标识 (微信开放平台创建应用申请获得)
wechat_app_secret	String	微信登录的应用密钥 (微信开放平台创建应用申请获得)
sina_app_id	String	新浪微博登录的应用标识 (新浪开放平台创建应用申请获得)
sina_app_secret	String	新浪微博的应用密钥 (新浪开放平台创建应用申请获得)
copyRight	String	保留字段
privacyPolicy	String	保留字段
SMSTemplat	String	保留字段

备注: 颜色值都使用argb的格式, 前2位表示透明度, 后6位是rgb颜色值。

第二步: 在application注册模组

```
/**
 * 模组注册
 */
private void registerModule() {
    try {
        ModuleManager.getInstance().registerModule(HeTApModuleImpl.class, getApplicationContext());//clifeAP绑定
        ModuleManager.getInstance().registerModule(HeTSmartlinkImpl.class, getApplicationContext());//clifessmartlink绑定
        ModuleManager.getInstance().registerModule(RealtekModuleImpl.class, getApplicationContext());//科中龙(realtekmodule)
        ModuleManager.getInstance().registerModule(XlwModuleImpl.class, getApplicationContext());//新力维_NL6621底层库
        ModuleManager.getInstance().registerModule(SctechModuleImpl.class, getApplicationContext());//双驰达(sctechmodule)
        ModuleManager.getInstance().registerModule(ElianModuleImpl.class, getApplicationContext());//信驰达_MTK7681底层库
        ModuleManager.getInstance().registerModule(MarvellV1WiFiImpl.class, getApplicationContext());//Marvell(marvellmodule)
        ModuleManager.getInstance().registerModule(MarvellV2WiFiImpl.class, getApplicationContext());//Marvell(marvellmodule)
        ModuleManager.getInstance().registerModule(EsptouchModuleImpl.class, getApplicationContext());//乐鑫信息科技(esptouchmodule)
        ModuleManager.getInstance().registerModule(CooeeModuleImpl.class, getApplicationContext());//博通(cooee module)
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```
}
```

模组注册是按需注册，根据APP需要支持的设备模组类型来选择性注册。

3.授权登录和用户模块

3.1.非私有服务器授权登录

HetNewAuthApi.getInstance().authorize() 跳转到授权登录页面。

```
/**
 * 授权登录
 *
 * @param authCallback 回调
 * @param specifyTitle 标题栏文字
 * @param navigationBarTextColor 标题栏文字颜色
 * @param navBackground 标题栏背景颜色
 */
public void authorize(Context mContext, AuthCallback authCallback, String specifyTitle, int
navigationBarTextColor, int navBackground) {
    ...
}
protected void auth() {

    HetNewAuthApi.getInstance().authorize(activity, new AuthCallback() {
        @Override
        public void onSuccess(int code, String msg) {
            //登录成功 do something
        }
        @Override
        public void onFailed(int code, String msg) {
            //登录失败 do something
        }
    }, "授权登录", Color.parseColor("#ff3285ff"), Color.parseColor("#FFFFFF"));
}
```

登录成功之后，SDK还会抛出HetCodeConstants.Login.LOGIN_SUCCESS事件。开发者也可以订阅这个事件来监听登录状态。

```
RxManage.getInstance().register(HetCodeConstants.Login.LOGIN_SUCCESS, o -> {
    //登录成功 刷新界面
});
```

授权登录页面：



为了您的设备使用安全，请先进
行手机号验证

请输入手机号

中国+86>

请输入验证码

获取验证码

提交

已有账号采用[密码验证](#)

第三方登录



3.2.云云对接用户授权登录

为了适应不同的业务需求，同时也考虑平台的安全问题SDK也提供了云云对接用户授权验证接口，该流程请参考文档[C-Life开放平台验证码三方授权流程](#)。

3.3.退出登录

退出登录SDK接口

```
HetSdk.getInstance().logout();
```

退出登录之后记得发出通知APP相应的页面跳转到首页，并刷新到未登录的状态。例如：

```
public void logout() {
    new AlertDialog.Builder(activity)
        .setMessage(activity.getString(R.string.confirm_logout))
        .setPositiveButton(activity.getString(R.string.logout_sure), (dialog, which) -> {
            dialog.dismiss();
            HetSdk.getInstance().logout();
        })
        .setNegativeButton(activity.getString(R.string.logout_cancel), null)
        .show();
}
```

退出登录成功，SDK会抛出HetCodeConstants.Login.LOGINOUT_SUCCESS事件。开发者可以订阅这个事件来监听退出登录。如：接收退出登录的事件，刷新页面

```
RxManage.getInstance().register(HetCodeConstants.Login.LOGINOUT_SUCCESS, o -> {
    //退出登录刷新页面
});
```

3.4.判断登录状态

判断登录状态的接口

```
HetSdk.getInstance().isAuthLogin();
```

3.5.异地登录

开放平台的账号只能在一台设备上登录。当同一个账号同时在2台设备上登录时，服务器会把前面登录成功的设备踢下线。被踢下线设备的SDK会退出登录，并且抛出

HetCodeConstants.Login.EC_LOGINOUT的RxBus事件，通知账号在其他设备登录。

开发者可以订阅这个事件，处理异地登录。例：

```
RxManage.getInstance().register(HetCodeConstants.Login.EC_LOGINOUTT, s -> {
```

```
//账号在其他设备登录，此时HetSdk.getInstance().isAuthLogin() 为false，跳转页面刷新到  
未登录状态。  
.....  
});
```

3.6.获取用户信息

HetUserApi.getInstance().getUserMess()获取用户信息

```
/**  
 * 获取用户信息  
 * @param iCallback 回调  
 */  
public void getUserMess(final IHetCallback iCallback) {  
    .....
```

调用实例：

```
public void getUserInfo() {  
    //账号登录之后才可以获取到用户信息，否则获取不到  
    if (!HetSdk.getInstance().isAuthLogin()) return;  
    HetUserApi.getInstance().getUserMess(new IHetCallback() {  
        @Override  
        public void onSuccess(int code, String msg) {  
            //获取用户信息成功  
            Type type = new TypeToken<UserInfoBean>() {  
                }.getType();  
            //users 包含账号的所有用户信息  
            UserInfoBean users = GsonUtil.getGsonInstance().fromJson(msg, type);  
        }  
        @Override  
        public void onFailed(int code, String msg) {  
            //获取用户信息失败  
        }  
    });  
}
```

接口返回UserInfoBean的用户详细参数说明：

参数名称	字段类型	参数说明
userId	String	用户ID
userName	String	用户名称
email	String	用户邮箱
sex	number	性别（1-男， 2-女）
birthday	String	生日（yyyy-MM-dd）
weight	number	体重（克）
height	number	身高（厘米）
avatar	String	头像URL

city	String	城市名
------	--------	-----

3.7.修改密码

调用 HetNewAuthApi.getInstance().alterPassword() 跳转到修改密码的页面。

```
/**
 * 修改密码
 * @param mContext 上下文
 * @param authCallback 回调
 * @param phone 手机号
 * @param specifyTitle 标题栏文字
 * @param navigationBarTextColor 标题栏文字颜色
 * @param navBackground 标题栏背景颜色
 * @throws Exception
 */
public void alterPassword(Context mContext, AuthCallback authCallback, String phone, String
specifyTitle, int navigationBarTextColor, int navBackground) throws Exception{
    .....
}
```

通过用户的手机号来修改密码，调用实例：

```
public void editPwd(String phone) {
    if (!HetSdk.getInstance().isAuthLogin()) return;
    HetNewAuthApi.getInstance().alterPassword(activity, new AuthCallback() {
        @Override
        public void onSuccess(int code, String msg) {
            //修改密码成功
        }
        @Override
        public void onFailed(int code, String msg) {
            //修改密码失败
        }
    }, phone, "修改密码", Color.parseColor("#ff3285ff"), Color.parseColor("#FFFFFF"));
}
```

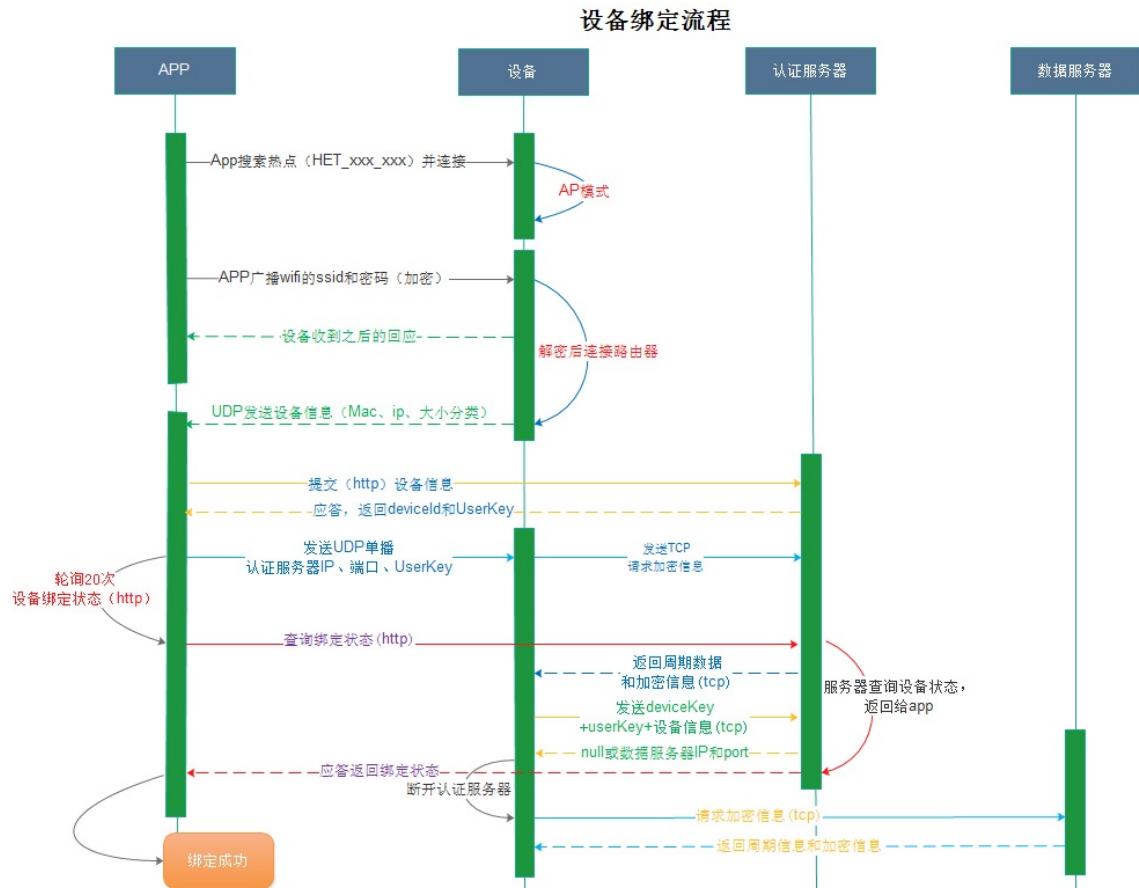
修改密码页面：



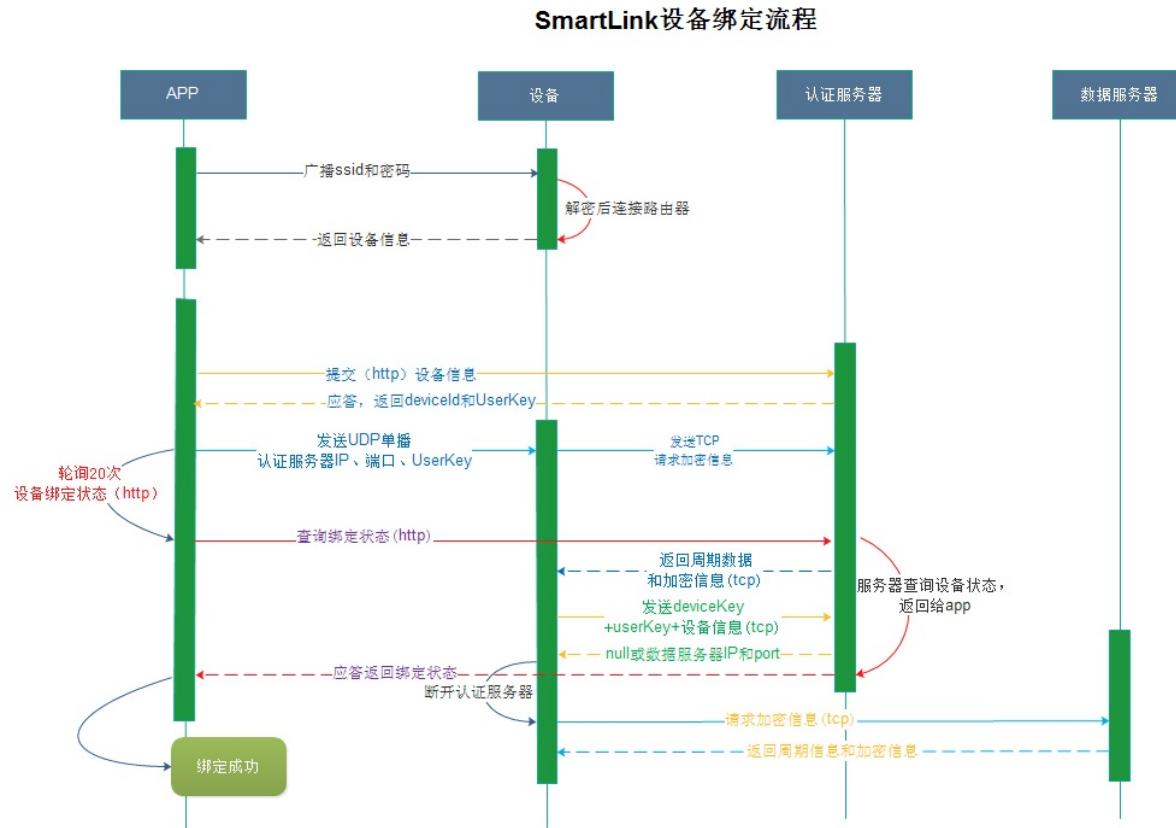
4.设备绑定模块（添加设备）

开放平台封装了wifi设备和蓝牙设备绑定接口，包括手机与服务器、手机与智能设备的通讯接口。开发者只要获取到设备的产品ID就可以调用SDK的绑定接口进行设备绑定，不需要关注复杂的绑定流程。

WIFI设备AP绑定流程图如下：



WIFI设备SmartLink绑定流程图如下：



开放平台SDK提供了三种方式来获取产品ID。

第一种：获取设备大类和设备小类（设备小类信息包含产品ID，也就是productId字段），具体分成2个步骤。

1. 获取设备大类型

`HetDeviceListApi.getInstance().getDownList()` 获取APP支持绑定的设备大类型。

调用实例：

```

HetDeviceListApi.getInstance().getDownList(new IHetCallback() {
    @Override
    public void onSuccess(int code, String s) {
        if (code == 0) {
            Type type = new TypeToken<List<DeviceTypeModel>>() {
                .getType();
            } // 获取设备大类成功 刷新UI
            List<DeviceTypeModel> models = GsonUtil.getGsonInstance().fromJson(list, t
            ype);
            .....
        }
    }
    @Override
    public void onFailure(int code, String msg) {
        // 获取设备大类列表失败
        .....
    }
});
```

2.获取设备小类型

HetDeviceListApi.getInstance().getSubTypeListProduct() 获取APP支持绑定的设备大类型。

通过选择的设备大类，查询大类下的小类列表。

```
HetDeviceListApi.getInstance().getSubTypeListProduct(new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
        if (code == 0) {
            Type type = new TypeToken<List<DeviceSubModel>>() {
                .getType();
            };
            List<DeviceSubModel> models = GsonUtil.getGsonInstance().fromJson(list, type);
        }
    }
    @Override
    public void onFailed(int code, String msg) {
        //获取设备小类列表失败
        .....
    }
}, deviceType); //deviceType 是大类类型
```

第二种：扫描开放平台创建的产品二维码来获取。

扫描到的结果调用SDK的HetQrCodeApi的dealQrCode方法获取产品信息。产品信息包含了产品ID

```
//二维码规则
private void parseQrCodeVersion(String url) {
    String param = Uri.parse(url).getQueryParameter("param");
    if (TextUtils.isEmpty(param)) {
        tips(getResources().getString(R.string.qr_code_error));
    } else {
        HetQrCodeApi.getInstance().dealQrCode(new IHetCallback() {
            @Override
            public void onSuccess(int code, String msg) {
                DeviceProductBean deviceProductBean = new Gson().fromJson(msg, DeviceProductBean.class);
                //获取到产品信息之后，按照第三步和第四步的流程来绑定
            }
            @Override
            public void onFailed(int code, String msg) {
                //获取到产品信息失败
                finish();
            }
        }, url);
    }
}
```

第三种：在开放平台后台直接查看产品ID，详情请查《clife开发平台使用手册》。

通过项目需求选择合适的方式来获取产品ID。然后根据设备类型选择SDK的绑定接口。具体分为WIFI绑定和蓝牙绑定2种。通过设备小类(moduleType字段)判断是WIFI设备还是蓝牙设备，进入相应的绑定设备绑定流程。如：

```
int type = deviceSubModel.getModuleType(); // type == 1标识WIFI type == 2标识蓝牙 type == 9
标识AP模式
```

下面对这2种绑定做具体说明

WIFI设备绑定： SHetWifiBindApi.getInstance().startBind() 启动绑定。

```
/*
 * 开始绑定设备
 *
 * @param ssid wifi名称
 * @param wifiPassword wifi密码
 * @param productId 产品ID
 * @param iWifiBind 绑定接口回调
 */
public void startBind(Activity activity, String ssid, String wifiPassword, String productId,
    IWifiBind iWifiBind) {
    .....
}

使用实例：
HetWifiBindApi.getInstance().startBind(this, sSidInfoBean.getSsid(), sSidInfoBean.getPass(
), "" + currentDevice.getProductId(), new IWifiBind() {
    @Override
    public void onStatus(HetWifiBindState hetWifiBindState, String msg) {
    }
    @Override
    public void onFailed(int errId, String msg) {
        //绑定失败 显示失败界面
        showBindFail();
    }
    @Override
    public void onSuccess(DeviceModel deviceModel) {
        //绑定成功 显示成功界面
        showBindSuccess();
    }
    @Override
    public void onProgress(int type, int value) {
        //扫描绑定的进度
        if(HetDeviceConstans.SCAN_PROGESS == type) {//扫描的进度
            setTextProgress(value);
        }else if(HetDeviceConstans.BIND_PROGESS == type){//绑定的进度
            showBindDialog();
        }
    }
});
```

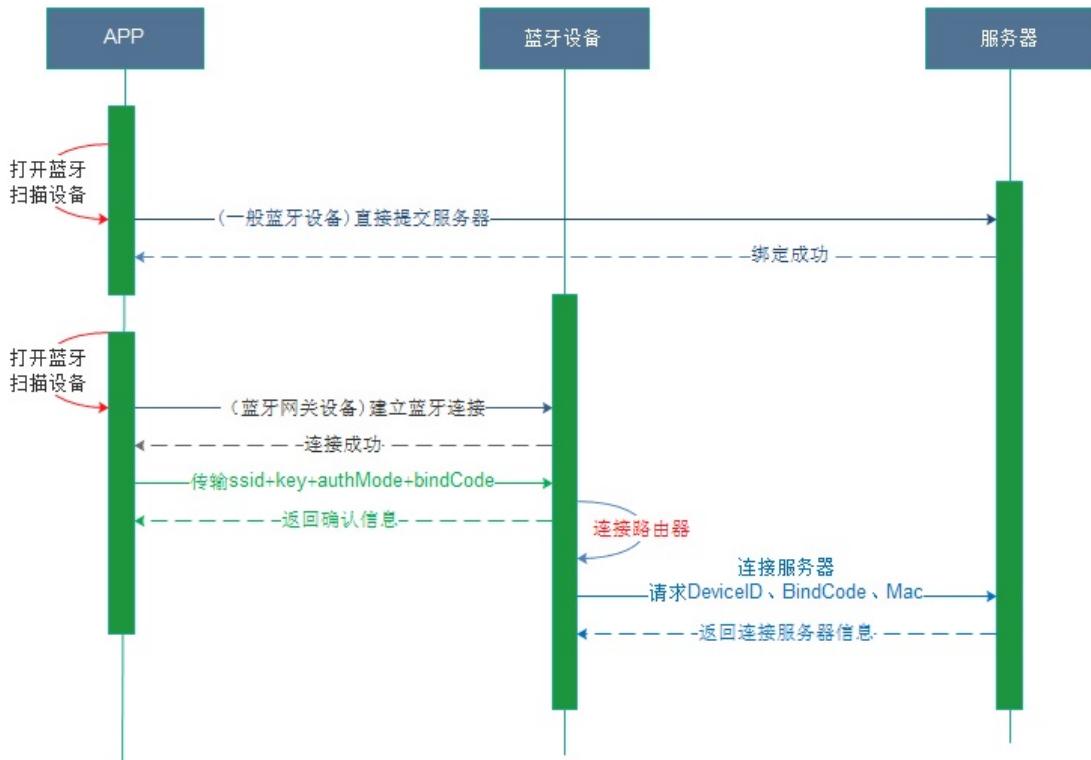
BLE蓝牙设备绑定： HetCommonBleBindApi.getInstance().startBind() 启动蓝牙设备扫描绑定。整个过程有2个步骤：

第一步：扫描搜索周围设备；

第二步：选择扫描到的某个设备绑定到服务器；

蓝牙网关设备绑定方式跟前面的方式有些小区别，两种方式的蓝牙设备绑定流程图如下：

蓝牙设备绑定



具体的接口调用说明：

```

HetCommonBleBindApi.getInstance().startBind(this, "" + deviceProductBean1.getProductId(),
new ICommonBleBind() {
    @Override
    public void onScanDevices(int code, String devices) {
        if (code == 0) {
            Type type = new TypeToken<List<DeviceProductBean>>() {
                .getType();
            List<DeviceProductBean> scanDevices = GsonUtil.getGsonInstance().fromJ
son(devices, type);
            //获取到扫描到的蓝牙设备，显示扫描的设备列表。选择某一个设备之后调用 bindToSe
rver绑定到服务器
            showScanSucess(scanDevices);
        }
    }
    @Override
    public void onFailed(int errId, String msg) {
        //绑定失败 刷新界面
        hideDialog();
        showBindFail();
    }
    @Override
    public void onSuccess(DeviceModel deviceModel) {
        //绑定成功 刷新界面
        showBindSuccess();
    }
    @Override
    public void onProgress(int type, int value) {
        if (HetDeviceConstans.SCAN_PROGESS == type) { //扫描进度
    }
}

```

```

        setTextProgress(value);
    } else if (HetDeviceConstans.BIND_PROGRESS == type) {//绑定进度
        showBindDialog();
    }
}
});

//扫描到设备列表，选择其中一个设备，绑定到服务器
HetCommonBleBindApi.getInstance().bindToServer(deviceProductBean);

```

绑定无法绑定？这里给出设备无法绑定的几种检查方法：

设备是否置为绑定模式，是否在绑定的有效时间内

是否正确输入wifi密码,请确认手机是否能正常连接网络

是扫描不到设备还是绑定不了设备,扫描失败会有对应提示是扫描不到设备，还是绑定不了设备

设备是否已在CLife开放平台注册，并按照要求将大小类信息写入设备中

APP端服务是否开启（udpservice）

5.设备管理

5.1.设备model说明

SDK所有的设备devicemodel，参数说明

字段名称	字段类型	字段说明
deviceId	String	设备标识
macAddress	String	MAC地址
deviceBrandId	number	设备品牌标识
deviceBrandName	String	设备品牌名称
deviceTypeId	number	设备大分类标识
deviceTypeName	String	设备大分类名称
deviceSubtypeId	number	设备子分标识
deviceSubtypeName	String	设备子分类名称
deviceName	String	设备名称
roomId	String	房间标识
roomName	String	房间名称
authUserId	String	授权设备用户标识
bindTime	String	绑定时间
onlineStatus	number	在线状态（1-正常，2-异常）
share	number	设备分享（1-是，2-否，3-扫描分享）
controlType	number	控制类型（1-原生，2-插件，3-H5插件）

userKey	String	MAC与设备ID生成的KEY
productId	number	设备型号标识
productName	String	设备型号名称
productCode	String	设备型号编码
productIcon	String	设备型号图标
moduleId	number	模块标识
moduleType	number	模块类型 (1-WiFi, 2-蓝牙, 9-AP模式)
moduleName	String	模块名称
radiocastName	String	设备广播名
deviceCode	String	设备编码

5.2. 获取设备列表

HetDeviceListApi.getInstance().getBindList()获取设备列表，设备按照归属来划分有2种：

第一种是绑定设备。用户绑定的设备，这类设备用户拥有这台设备的所有权限。

第二种是分享设备。别人分享的设备，这类设备用户拥有控制权限，但是不可以再分享给其他人。

可以根据设备deviceModel的share字段来判断是否是分享的设备。

```

HetDeviceListApi.getInstance().getBindList(new IHetCallback() {
    @Override
    public void onSuccess ( int code, String s){
        if (code == 0) {
            if (!TextUtils.isEmpty(s)) {
                Type type = new TypeToken<List<DeviceModel>>() {
                    .getType();
                List<DeviceModel> models = GsonUtil.getGsonInstance().fromJson(list, t
ype);
                //获取设备列表成功列表
            }
        }
    }
    @Override
    public void onFailed ( int code, String msg){
        //获取列表失败
    }
}

```

5.3. 删除设备

SDK删除设备列表中的设备有2中情况：

第一种：绑定设备。调用HetDeviceManagerApi.getInstance().unBind()解除绑定。实例：

```

HetDeviceManagerApi.getInstance().unBind(deviceModel, new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
        //解绑成功
    }
}

```

```

@Override
public void onFailed(int code, String msg) {
    //解绑失败
}
});

```

第二种：分享设备。调用HetDeviceShareApi.getInstance().deviceDel()解绑分享关系。实例：

```

HetDeviceShareApi.getInstance().deviceDel(new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
        iHetCallback.onSuccess(code, msg);
    }
    @Override
    public void onFailed(int code, String msg) {
        iHetCallback.onFailed(code, msg);
    }
}, deviceId, null);

```

传参是设备ID和用户UserId。UserId传null表示被分享者解除分享关系，传userId表示设备拥有者回收这个用户的设备控制授权。

5.4.设备分享

5.4.1.获取设备授权的用户列表

调用HetDeviceShareApi.getInstance().getDeviceAuthUser()获取设备授权的用户列表，调用实例：

```

HetDeviceShareApi.getInstance().getDeviceAuthUser(new IHetCallback() {
    @Override
    public void onSuccess(int code, String s) {
        if (code == 0) {
            Type type = new TypeToken<List<DeviceAuthUserModel>>() {
                }.getType();
            List<DeviceAuthUserModel> deviceAuthUsers= GsonUtil.getGsonInstance().from
Json(s, type);
            //获取到设备授权的用户列表
            .....
        }
    }
    @Override
    public void onFailed(int code, String msg) {
        //获取设备授权的用户列表失败
    }
}, deviceId);

```

DeviceAuthUserModel 的字段说明：

参数名称	字段类型	参数说明
userId	String	用户ID
userName	String	用户名称

avatar	String	用户名称
authTime	String	授权时间

5.4.2. 用户设备授权删除

调用HetDeviceShareApi.getInstance().deviceDel()解除分享关系。

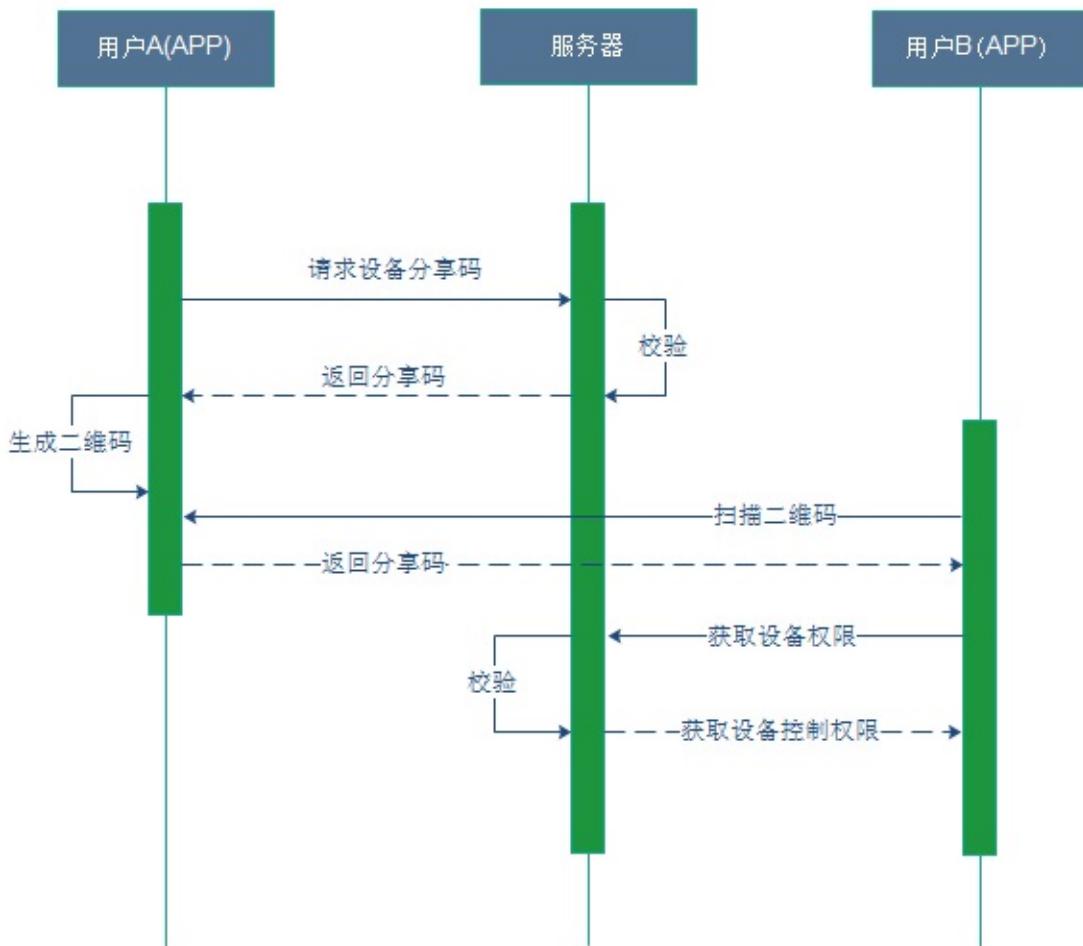
```
HetDeviceShareApi.getInstance().deviceDel(new IHetCallback() {
    @Override
    public void onSuccess(int code, String s) {
        if (code == 0) {
            //删除成功,刷新获取设备授权用户列表
            .....
        }
    }
    @Override
    public void onFailed(int code, String msg) {
        //删除失败
    }
}, deviceId, userId);
```

5.4.3. 分享设备

设备分享有面对面二维码分享和第三方平台（微信、QQ等）分享两种方式。绑定设备才能分享，分享设备不能再分享给其他用户。

第一种：面对面分享，通过deviceId（要分享的设备的标识）获取分享码，分享的流程图如下：

设备分享(应用内)



具体的接口调用说明：

```

/**
 * @param shareType 5是面对面分享 6第三方分享
 */
HetDeviceShareApi.getInstance().getShareCode(new IHetCallback() {
    @Override
    public void onSuccess(int code, String s) {
        if (code == 0) {
            Type treeType = new TypeToken<ShareCodeModel>() {
            }.getType();
            ShareCodeModel codeModel = GsonUtil.getGsonInstance().fromJson(s, treeType
        );
        //分享邀请码获取成功
    }
}
@Override
public void onFailed(int code, String msg) {
    //分享邀请码获取失败
}
}, deviceId, shareType);
  
```

ShareCodeModel字段说明:

参数名称	字段类型	参数说明
shareCode	String	设备分享码（面对面分享）
h5Url	String	H5 页面地址（第三方分享）

将分享码用二维码的形式展示。被分享的用户，通过二维码扫描，调用 HetDeviceShareApi.getInstance().authShareDevice()完成设备分享。

```

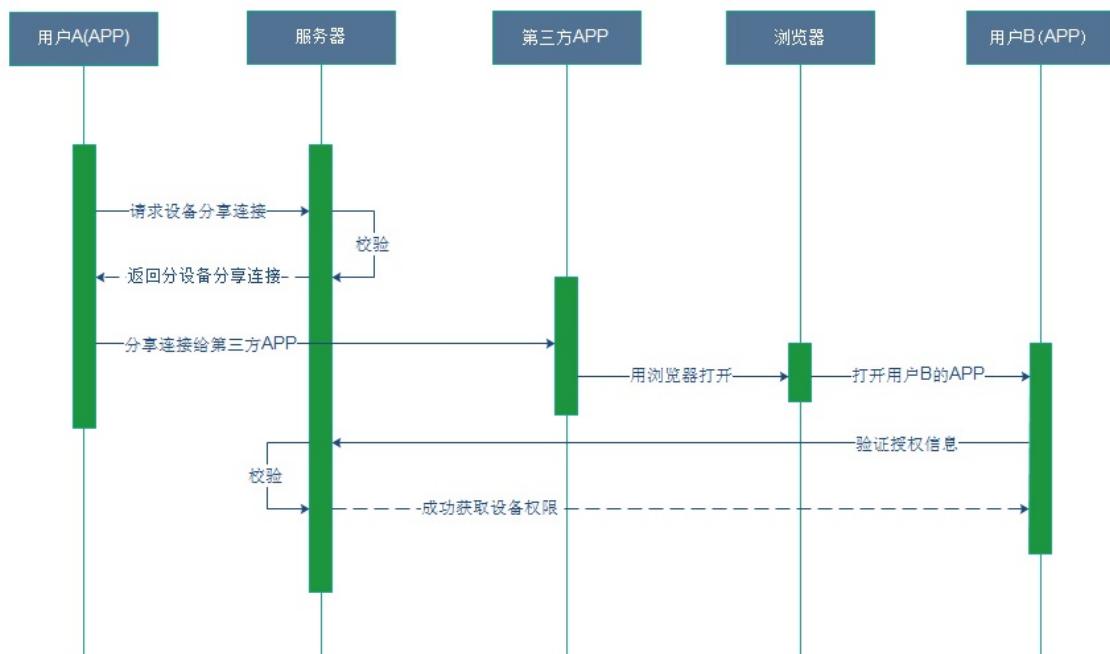
HetDeviceShareApi.getInstance().authShareDevice(new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
        ToastUtil.showToast(mContext, "设备分享成功");
        RxManage.getInstance().post(HetShareEvent.HET_EVENT_MAIN_SHARE_SUCCEE,null);
    }
    @Override
    public void onFailed(int code, String msg) {
        ToastUtil.showToast(mContext, "设备分享失败");
    }
}, code, "5");//5是面对面分享 6是远程分享
    
```

设备分享成功，抛出RxBus事件，监听事件刷新设备列表。

第二种：第三方分享（通过QQ、微信分享设备）

利用第三方社交平台，可以快速的实现设备分享。有利于实现产品的快速推广。远程的第三方分享一定要集成第三方分享服务。详细集成实例请参考下面 第三方平台服务的集成（登录和分享）的集成。分享的流程图如下：

设备分享（第三方应用分享）



具体的流程有3个步骤：

第一步：获取分享URL地址

HetDeviceShareApi.getInstance().getShareCode(IHetCallback callback, String devicId, String shareType) 参数 shareType(5是面对面分享 6远程分享) 标识分享类型，返回远程分享设备的分享地址。

```
HetDeviceShareApi.getInstance().getShareCode(new IHetCallback() {
    @Override
    public void onSuccess(int code, String s) {
        if (code == 0) {
            Type treeType = new TypeToken<ShareCodeModel>() {
                .getType();
            ShareCodeModel codeModel = GsonUtil.getGsonInstance().fromJson(s, treeType
        );
        //分享邀请码获取成功
        String shareUrl = codeModel.getH5Url();
        //调用第三分享接口把这个网页地址分享到第三方平台
        .....
    }
    @Override
    public void onFailed(int code, String msg) {
        //分享邀请码获取失败
    }
}, deviceId, "6");
```

第二步：分享者分享URL到第三方平台

SDK提供了第三方分享的接口(暂时只支持微信，QQ，新浪微博)。分享者把URL分享地址通过第三方平台发送给分享者，邀请被分享者控制设备。

1.集成第三方服务

详细的集成流程请查看 [第三方平台服务的集成（登录和分享）](#)

2.初始化SKD第三方分享接口

```
HetSdkThirdDelegate mShareManager = HetSdkThirdDelegate.getInstance();
CommonShareProxy mShareProxy = new CommonShareProxy(this);
mShareManager.setShareOperate(new CommonShareOperate(mContext));
mICommonShareListener = new ICommonShareListener() {
    @Override
    public void onStartShare(CommonSharePlatform sharePlatform) {
        CommonShareWebpage webpage = new CommonShareWebpage(sharePlatform);
        webpage.setUiListener(this);
        webpage.setTitle("设备分享");
        webpage.setDescription("设备分享，极速体验");
        webpage.setAppName(getString(R.string.app_name));
        webpage.setTargetUrl(shareUrl);
        webpage.setWebpageUrl(shareUrl);
        webpage.setBm(BitmapFactory.decodeResource(mContext.getResources(), R.mipmap.i
con_share));
        webpage.setSharePlatform(sharePlatform);
        mShareManager.shareWebpage(webpage);
    }
    @Override
```

```

    public void onShareSuccess(CommonSharePlatform sharePlatform, String msg) {
        UserMessShareActivity.this.runOnUiThread(() -> {
            ToastUtil.showToast(mContext, "分享成功");
        });
    }
    @Override
    public void onShareFialure(CommonSharePlatform sharePlatform, String msg) {
        UserMessShareActivity.this.runOnUiThread(() -> {
            ToastUtil.showToast(mContext, "分享失败");
        });
    }
    @Override
    public void onShareCancel(CommonSharePlatform sharePlatform, String msg) {
        UserMessShareActivity.this.runOnUiThread(() -> {
            ToastUtil.showToast(mContext, "分享取消");
        });
    }
}

```

3.添加回调

```

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (mShareManager != null && mShareProxy != null) {
        mShareProxy.onActivityResult(requestCode, resultCode, data);
    }
}

```

4.开始分享

```

mICommonShareListener.onStartShare(CommonSharePlatform.WeixinFriend); //微信分享
mICommonShareListener.onStartShare(CommonSharePlatform.QQ_Friend); //QQ分享
分享类型有5种
public enum CommonSharePlatform {
    WeixinFriend, //微信好友
    WeixinFriendCircle, //微信朋友圈
    QQ_Friend, //QQ好友
    QQ_Zone, //QQ空间
    SinaWeibo; //新浪微博
    ...
}

```

5.退出释放资源

```

@Override
public void onDestroy() {
    super.onDestroy();
    if (mShareManager != null) {
        mShareManager.releaseResource();
        mShareManager = null;
    }
}

```

第三步:被分享者接收设备控制邀请，完成设备的分享。

被分享者通过第三方平台（微信或QQ）收到URL，在浏览器打开URL。浏览器请求打开用户APP（通过scheme协议），用户APP获取到shareCode调用
HetDeviceShareApi.getInstance().authShareDevice() 接收设备控制邀请。

```
HetDeviceShareApi.getInstance().authShareDevice(new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
        RxManage.getInstance().post(HetShareEvent.HET_EVENT_MAIN_SHARE_SUCCEE, null);
    }

    @Override
    public void onFailed(int code, String msg) {
    }
}, shareCode, "6");
```

浏览器打开用户APP的scheme协议，需要在 AndroidManifest.xml 中加上一下代码

```
<activity
    name="your_open_activity"
    screenOrientation="portrait">

    <intent-filter>
        <action      name="android.intent.action.VIEW" />

        <category     name="android.intent.category.DEFAULT" />
        <category     name="android.intent.category.BROWSABLE" />

        <data        scheme="hetopenplatform" />
    </intent-filter>

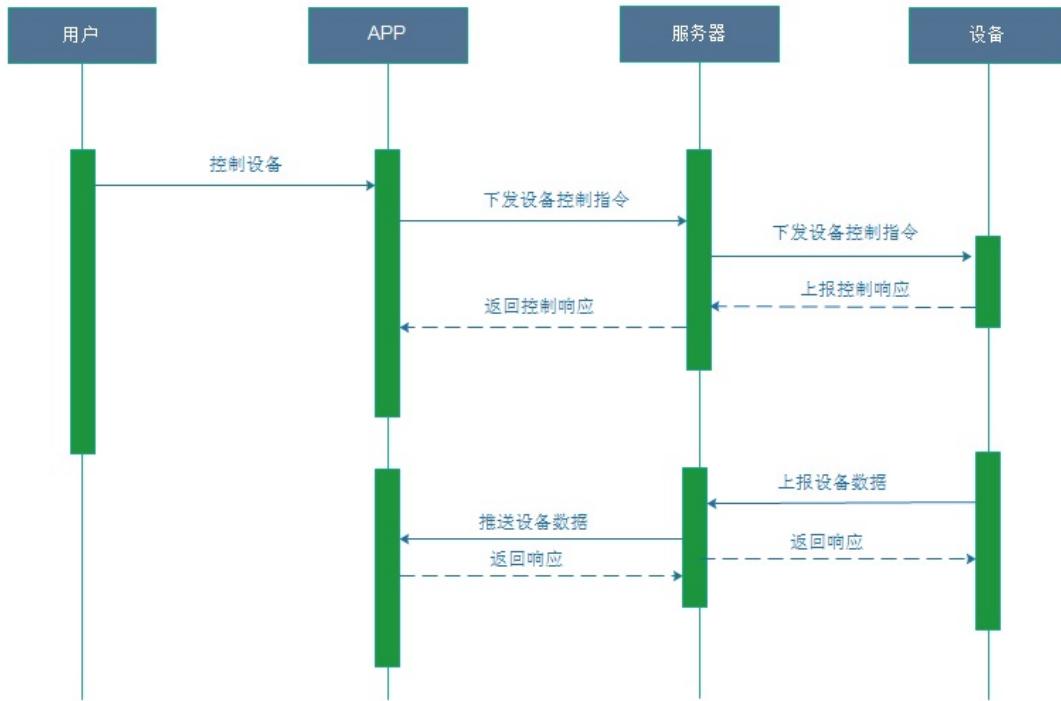
</activity>
```

注意：该Activity是浏览器请求打开app第一个请求打开的界面。

6.设备控制

设备有WIFI设备控制和蓝牙设备控制两种，wifi设备控制流程图示如下：

设备控制流程



SDK提供统一的数据流接口。接收设备数据和监听设备状态都是通过IWifiDeviceData这个接口来完成，发送数据调用HetDeviceWifiControlApi.getInstance().setDataToDevice()。

IWifiDeviceData 的接口说明：

```

public interface IWifiDeviceData {
    /**
     * 获取控制数据
     * @param jsonData
     */
    void onGetConfigData(String jsonData);

    /**
     * 获取运行数据
     * @param jsonData
     */
    void onGetRunData(String jsonData);

    /**
     * 获取异常数据
     * @param jsonData
     */
    void onGetErrorHandlerData(String jsonData);

    /**
     * 设备状态
     * @param onlineStatus 在线状态 (1-正常, 2-异常不在线)
     */
    void onDeviceStatues(int onlineStatus);

    /**
    
```

```

    * 收取数据异常
    * @param code 错误码
    * @param msg 错误信息
    * 1000 无法连接服务器
    * 1001 设备不在线
    */
void onDataErrorHandler(int code, String msg);
}

```

WIFI设备控制具体可以分成3个步骤：

第一步：设置接收设备数据的监听

```

HetWifiDeviceControlApi.getInstance().start(deviceModel.getDeviceId(), deviceModel.getMacAddress());
private IWifiDeviceData iWifiDeviceData = new IWifiDeviceData() {
    @Override
    public void onGetConfigData(String jsonData) {
        //获取到设备上报控制数据 根据开放平台配置的协议解析成相应的数据
        System.out.println("onGetConfigData: " + jsonData);
        LedConfigModel configModel = GsonUtil.getInstance().toObject(jsonData, LedConfigModel.class);
        if (ledConfigModel != null) {
            ledConfigModel = configModel;
        }
    }
    @Override
    public void onGetRunData(String jsonData) {
        //获取到设备上报运行数据 根据开放平台配置的协议解析成相应的数据
        System.out.println("onGetRunData: " + jsonData);
        LedRunDataModel runDataModel = GsonUtil.getInstance().toObject(jsonData, LedRunDataModel.class);
        if (runDataModel != null) {
            ledRunDataModel = runDataModel;
        }
    }
    @Override
    public void onGetErrorHandler(String jsonData) {
        //获取到设备上报故障数据 根据开放平台配置的协议解析成相应的数据
        System.out.println("onGetErrorHandler: " + jsonData);
    }
    @Override
    public void onDeviceStatuses(int onlineStatus) {
    }
    @Override
    public void onDataErrorHandler(int code, String msg) {
        System.out.println("onDataErrorHandler: " + msg + " code " + code);
    }
};

```

可以根据IWifiDeviceData 的回调接口接收设备数据渲染UI。

第二步：控制设备

调用HetDeviceWifiControlApi.getInstance().setDataToDevice()发送控制数据到服务器。调用的时候把需要发送的控制数据组装好。

```

HetDeviceWifiControlApi.getInstance().setDataToDevice(new IHetCallback() {
    @Override
    public void onSuccess(int code, String msg) {
    }
    @Override
    public void onFailure(int code, String msg) {
    }
}, deviceModel.getDeviceId(), GsonUtil.getInstance().toJson(ledConfigModel));

```

第三步：释放资源

停止使用设备控制功能的时候释放资源，例如退出控制界面时，实例：

```

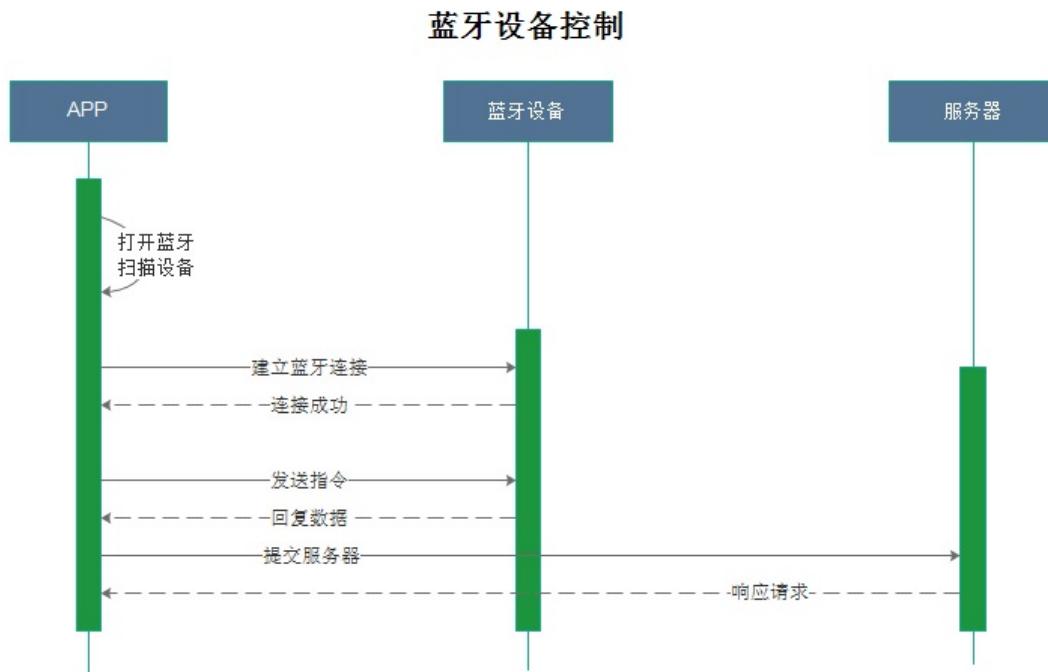
@Override
protected void onDestroy() {
    super.onDestroy();
    HetWifiDeviceControlApi.getInstance().stop();
}

```

注意：设备控制的时候会有一个updateFlag字段。这个字段标识是改变了那几个控制字段。

第二种 蓝牙设备控制：

蓝牙设备控制主要是通过手机和蓝牙设备先建立连接，然后根据定义的协议进行数据交互，具体的交互流程图如下：



具体的蓝牙控制分成5个步骤：

第一步：初始化HetCommonBleControlApi

```

HetCommonBleControlApi.getInstance().init(this);

```

第二步：手机建立蓝牙连接,设置数据的收发回调

```

private void connect() {
    HetCommonBleControlApi.getInstance().connect(macAddress,connectCallback);
}
private final IConnectCallback connectCallback = new IConnectCallback() {
    @Override
    public void onConnectSuccess(BluetoothGatt gatt, int status) {
        conDevice.setText("连接成功");
        HookManager.getInstance().enableHook(true, "fuck@academy");
        HookManager.getInstance().addHook(hookCallBack);
    }
    @Override
    public void onConnectFailure(final BleException exception) {
        conDevice.setText("连接失败");
    }
    @Override
    public void onDisconnect(String mac) {
        conDevice.setText("连接断开");
    }
}

```

IHookCallBack 监听发送数据和接收数据的回调。开发者可以在此监听app发送的数据和收到的设备数据。

```

private IHookCallBack hookCallBack = new IHookCallBack() {
    @Override
    public void onWrite(byte[] bytes) {
        showInfo("Send:" + HexUtil.encodeHexStr(bytes));
    }
    @Override
    public void onRead(byte[] bytes) {
        showInfo("Read:" + HexUtil.encodeHexStr(bytes));
    }
    @Override
    public void onReceived(byte[] bytes) {
        showInfo("Rec:" + HexUtil.encodeHexStr(bytes));
    }
};

```

备注:connect方法是允许重复调用的。对于已经连接的成功的连接，在此调用connect方法，SDK会直接返回连接成功不会再次去连接设备。

第三步：写数据

SDK对开放平台蓝牙设备提供写数据的标准接口 HetCommonBleControlApi.getInstance().write(), 调用实例：

```

HetCommonBleControlApi.getInstance().write(macAddress,CmdIndexConstant.HET_COMMAND_BIND_AP
P,writeCallback);

private IBleCallback<HetOpenPlatformCmdInfo> writeCallback = new IBleCallback<HetOpenPlat
ormCmdInfo>() {
    @Override
    public void onSuccess(HetOpenPlatformCmdInfo cmdInfo, int type) {

```

```

        //byte[] bytes = (byte[]) cmdInfo.getReceivePacket();
        //showInfo(HexUtil.encodeHexStr(bytes));
    }

    @Override
    public void onFailure(BleException exception) {
        showInfo(exception.getDescription());
    }
};

```

第一个参数是设备mac。

第二个参数是开放平台蓝牙协议标准接口类型。可以参照一下：

CmdIndexConstant.HET_COMMAND_BIND_APP 绑定APP

CmdIndexConstant.HET_COMMAND_GET_TIME_APP 获取设备时间

CmdIndexConstant.HET_COMMAND_SET_TIME_APP 设置设备时间

CmdIndexConstant.HET_COMMAND_GET_HISTORY_DATA_APP 获取设备历史数据

CmdIndexConstant.HET_COMMAND_CLEAR_HISTORY_DATA_APP 清楚设备历史数据

CmdIndexConstant.HET_COMMAND_GET_REAL_TIME_DATA_APP 获取真实的设备数据

CmdIndexConstant.HET_COMMAND_CONFIG_DATA_APP 下发控制协议

第三个参数是写数据成功与否的监听回调。这里开发者可以自行做重发处理。

第四步：读数据

SDK对开放平台蓝牙设备提供写数据的标准接口 HetCommonBleControlApi.getInstance().read()。

SDK中提供了标准的获取设备信息的接口，调用实例：

```

HetCommonBleControlApi.getInstance().read(macAddress,CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_SYSTEM_ID,readCallback);

private IBleCallback<HetOpenPlatformCmdInfo> readCallback = new IBleCallback<HetOpenPlatformCmdInfo>() {
    @Override
    public void onSuccess(HetOpenPlatformCmdInfo cmdInfo, int type) {
        byte[] bytes = (byte[]) cmdInfo.getReceivePacket();
        String msg = "Read:" + ConvertUtil.convertHexToString(HexUtil.encodeHexStr(bytes));
    }
    //成功读取到设备数据
    .....
}

@Override
public void onFailure(BleException exception) {
    //读取设备数据失败
    .....
}
};

```

第一个参数是设备mac。

第二个参数是开放平台蓝牙协议标准接口类型。可以参照一下：

CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_SYSTEM_ID System Id

CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_FIRMWARE_REVISION Firmware

Revision

CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_HARDWARE_REVISION Hardware Revision
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_SOFTWARE_REVISION Software Revision
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_SERIAL_NUMBER Serial Number
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_MANUFACTURE_NAME Manufacture Name ()
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_MODEL_NUMBER Model Number
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_PNP_ID PnP ID
 CmdIndexConstant.DeviceInfoConstant.HET_COMMAND_BATTERY Battery Level
 第三个参数是读数据成功与否的监听回调。

第五步：释放资源(断开连接)

```
HetCommonBleControlApi.getInstance().disConnect(macAddress);
```

7.其他接口

7.1.意见反馈

调用HetFeedbackApi.getInstance().addFeedback()提交意见反馈

```
/**  
 * 意见反馈  
 *  
 * @param iCallback the callback  
 * @param contact 联系方式 可以传null匿名提交  
 * @param content 反馈内容  
 */  
public void addFeedback(final IHetCallback iCallback, String contact, String content) {  
    FeedbackDeal.addFeedback(iCallback, contact, content);  
}
```

7.2.消息模块

SDK提供了操作消息的接口HetMessageApi

```
/**  
 * 刷新列表  
 *  
 * @param callback 回调  
 * @param messageId 消息ID  
 * @param messageType 消息类型  
 * @param pageRows 每页数据大小  
 */  
public void refreshList(IHetCallback callback, String messageId, String messageType, String pageRows) {  
    ....  
}
```

```

    /**
     * 加载更多
     *
     * @param callback 回调
     * @param messageId 消息ID
     * @param messageType 消息类型
     * @param pageRows 每页数据大小
     */
    public void loadList(IHetCallback callback, String messageId, String messageType, String p
ageRows) {
        ....
    }
    /**
     * 删除消息
     *
     * @param callback 回调
     * @param messageId 消息ID
     */
    public void deleteMessage(IHetCallback callback, String messageId) {
        ....
    }
    /**
     * 消息标记为已读
     *
     * @param callback 回调
     * @param messageId 消息ID
     */
    public void readMessage(IHetCallback callback, String messageId) {
        ....
    }
    /**
     * 消息标记为已读
     *
     * @param callback 回调
     * @param messageId 消息ID
     */
    public void updateMsg(IHetCallback callback, String messageId) {
        ....
    }
}

```

根据项目的需求来调用相关接口。

调用HetMessageApi.getInstance().refreshList()获取消息列表，调用传参说明：

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	访问凭证
timestamp	是	number	时间戳
messageId	是	number	消息标识，只有上拉时传值，下拉时不能传值
messageType	否	number	0-系统消息；1-添加好友；2-邀请控制设备；3-查看帖子评论；5-运营互动
selType	否	number	查询类型。按照人查询消息时不传值，按照app查询时，必传1

pageRows	否	number	每页数据大小
pageIndex	否	number	加载第几页

返回json结果：

```
{
  "data": {
    "pager": {"totalRows":0,
              "pageRows":1,
              "pageIndex":1,
              "paged":false,
              "defaultPageRows":20,
              "currPageRows":0,
              "pageStartRow":0,
              "hasPrevPage":false,
              "hasNextPage":false,
              "totalPages":0,
              "pageEndRow":0},
    "list": [{"messageId":1,
              "title":"特特特特",
              "description":"大声答答",
              "businessParam":"11111",
              "sender":1,
              "icon":"http://www.test.com",
              "createTime":1434014367000,
              "messageType":1,
              "status":1,
              "level2":3,
              "content":"http://200.200.200.50/clife_app/page/topic-view.html?type=2&id=927",
              "readed":0,
              "readonly":0,
              "summary":null,
              "pictureUrl":null
            }]
  },
  "code": 0
}
```

返回的结果字段说明：

字段名称	字段类型	字段说明
messageld	number	消息标识
title	string	标题
description	string	描述
businessParam	string	业务参数的值(系统推送消息对应消息详情URL(businessParam为空时不要跳转); 添加好友消息对应用户ID, 控制设备消息对应设备ID, 查看帖子评论对应帖子详情URL。)
sender	number	发送者ID

icon	string	图标URL
messageType	number	消息类型: 0-系统消息; 1-添加好友; 2-邀请控制设备; 3-查看帖子评论; 5-运营互动; 其他后续补充
createTime	number	时间戳
status	number	(系统消息的时候如果操作类标识)系统消息下的二级分类: 1-无正文; 2-文本H5; 3-外链; 4-设备
content	String	表示设备信息时候建议接口调用时传json格式值)系统消息内容
readed	number	消息是否已读 (0-已读 1-未读)
readonly	number	消息是否只读 (0-只读类 1-操作类)
summary	String	简要描述
pictureUrl	String	简图路径

第三方平台服务的集成（登录和分享）

第三方登录和分享的集成，SDK目前只支持三种方式，也是目前比较主流的第三方平台。包括微信、QQ、和新浪微博。

具体过程分4个步骤：

第一步：在集成之前需要在微信开放平台、腾讯开放平台、新浪开放平台创建应用，获取到相应的appId和appSecret。

第二步：在Application里面配置第三方登录SDK。

```
//配置第三方登录
mLoginDelegate = new HetSdkThirdDelegate.Builder(this)
    .registerQQ("your_qq_app_id")
    .registerWeixin("your_weixin_app_id", "your_weixin_app_secret")
    .registerSinaWeibo("your_sina_app_id", "your_sina_app_secret", "your_sina_redirect_url")
    .create();
```

注意：your_sina_redirect_url是新浪微博用于OAuth authorize页面回调的url。

第三步：配置清单文件AndroidManifest.xml

```
<!-- =====第三方登录分享开始 ===== -->
<activity
    name="com.het.open.lib.wb.WBEntryActivity"
    configChanges="keyboardHidden|orientation"
    launchMode="singleTask"
    screenOrientation="portrait"
    windowSoftInputMode="stateAlwaysHidden">
    <intent-filter>
        <action      name="com.sina.weibo.sdk.action.ACTION_SDK_REQ_ACTIVITY" />
        <category    name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
<activity
```

```

        name=".wxapi.WXEntryActivity"
        configChanges="keyboardHidden|orientation"
        exported="true"
        screenOrientation="portrait"
        theme="@android:style/Theme.Translucent.NoTitleBar" />
<activity
        name="com.tencent.connect.common.AssistActivity"
        screenOrientation="portrait"
        theme="@android:style/Theme.Translucent.NoTitleBar" />
<activity
        name="com.tencent.tauth.AuthActivity"
        launchMode="singleTask"
        noHistory="true"
        screenOrientation="portrait">
    <intent-filter>
        <action      name="android.intent.action.VIEW" />
        <category    name="android.intent.category.DEFAULT" />
        <category    name="android.intent.category.BROWSABLE" />
        <!-- 这里要把1106346235替换成自己在腾讯开放平台注册的appId -->
        <data        scheme="tencent1106346235" />
    </intent-filter>
</activity>
<!-- sinaweibo -->
<activity
        name="com.sina.weibo.sdk.component.WeiboSdkBrowser"
        configChanges="keyboardHidden|orientation"
        exported="false"
        screenOrientation="portrait"
        windowSoftInputMode="adjustResize" />
<service
        name="com.sina.weibo.sdk.net.DownloadService"
        exported="false" />
<!-- =====第三方登录分享结束===== -->

```

并添加相应的权限

```

<uses-permission      name="android.permission.INTERNET"></uses-permission>
<uses-permission      name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission      name="android.permission.WRITE_APN_SETTINGS"></uses-permission>
<uses-permission      name="android.permission.CHANGE_WIFI_STATE"></uses-permission>

```

第四步：设置微信登录的回调页面。

在项目包名目录下添加一个wxapi目录在wxapi里面新建一个WXEntryActivity页面，如：

```

public class WXEntryActivity extends Activity implements IWXAPEventHandler {
    private IWXAPE api;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Logc.e("Weixin", "WXEntryActivity....onCreate", false);
        api = WXAPIFactory.createWXAPI(this, UIJsonConfig.getWechatAppID(), true);
        api.registerApp(UIJsonConfig.getWechatAppID());
        api.handleIntent(getIntent(), this);
    }
    @Override
    public void onReq(BaseReq arg0) {
        Logc.e("WXEntryActivity....onReq", false);
    }
}

```

```

    }
    @Override
    public void onResp(BaseResp resp) {
        Logc.e("WXEntryActivity....onResp", false);
        if (resp instanceof SendAuth.Resp) {
            WeiXinLogin.getInstance().onResp(this, (SendAuth.Resp) resp);
            return;
        }
        this.finish();
    }
}

```

注意：wxapi和WXEntryActivity的位置和名字都不能改变，否则不能回调到app中来。例如：DEMO APP的包名是com.het.sdk.demo，那WXEntryActivity的完整名称就是com.het.sdk.demo.wxapi.WXEntryActivity。

新浪微博分享回调SDK已经集成，com.het.open.lib.wb.WBEntryActivity，开发者不需要关注。

通用的业务接口

开发平台其他的业务接口可以调用 HetHttpApi 来发起请求

```

/**
 * post请求
 *
 * @param url      请求地址
 * @param params   请求参数
 * @param callback 请求结果回调
 */
public void hetPost(@NotNull String url, TreeMap params, IHetCallback callback){
    .....
}

/**
 * get请求
 *
 * @param url      请求地址
 * @param callback 请求结果回调
 */
public void hetGet(@NotNull String url, IHetCallback callback){
    .....
}

/**
 * get请求
 *
 * @param url      请求地址
 * @param params   请求参数
 * @param callback 请求结果回调
 */
public void hetGet(@NotNull String url, TreeMap params, IHetCallback callback){
    .....
}

```

带参数的GET请求调用实例：

```

/**
 * 带参数的通用get请求
 *
 * @param url      请求地址
 * @param params   请求参数
 */
private void commonGet(url,param1) {
    TreeMap<String, String> params = new TreeMap<String, String>();
    params.put("param1", param1);
    HetHttpApi.getInstance().hetGet(url, params, new IHetCallback() {
        @Override
        public void onSuccess(int code, String msg) {
            //成功
        }

        @Override
        public void onFailed(int code, String msg) {
            //失败
        }
    });
}

```

全局返回码

全局返回码说明

每次调用接口时，可能获得正确或错误的返回码，可以根据返回码信息调试接口，排查错误。

全局返回码说明如下：

返回码	说明
0	请求成功
100010100	缺少授权信息
100010101	AccessToken错误或已过期
100010102	RefreshToken错误或已过期
100010103	AppId不合法
100010104	timestamp过期
100010105	签名错误
100010200	失败
100010201	缺少参数
100010202	参数错误
100010203	必须使用https
100010208	产品不存在
100021000	帐号已注册

100021001	帐号未注册
100021007	帐号已邀请
100021008	邀请你的用户已解绑该设备
100021010	邀请已被接受
100021301	验证码错误
100021302	随机码错误
100021303	您的访问太过频繁, 请15分钟之后再尝试
100021304	重新获取验证码成功
100021401	用户不存在
100021500	密码错误
100021603	数据不存在
100022000	设备不存在
100022001	设备未绑定
100022002	设备已绑定
100022003	设备解绑失败
100022004	MAC地址已绑定另一种设备
100022005	设备控制JSON错误
100022006	设备不在线
100022008	不能邀请自己控制
100022011	设备已授权
100022012	待更换MAC与原MAC相同
100022013	appId与产品未做关联
100022014	待绑定MAC未进行服务注册
106000021	应用无权限查看该设备信息
106000026	产品不存在
106000031	应用包名错误
106000036	openId错误
106000037	手机号码错误
106000041	帐号错误, 请使用开放平台账号登录

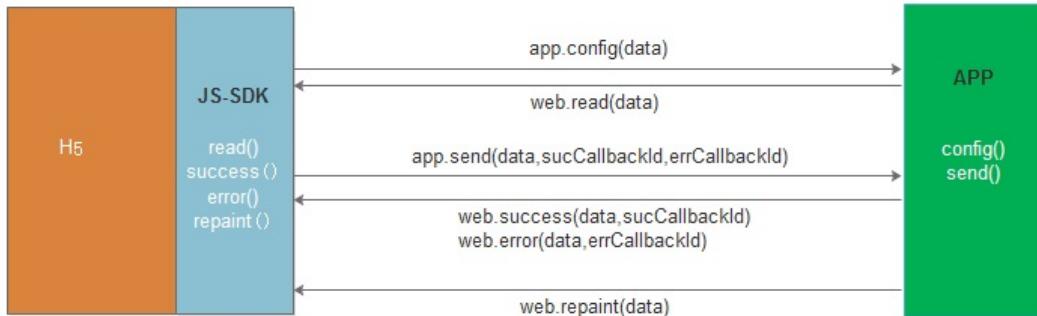
H5+Native混合框架

为了适应APP不断添加新的设备和动态更新, clife平台结合APP开发一套动态的插件更新框架。基于这套框架可以实现app功能的快速开发迭代, 减少产品的上线周期。

1.H5开发框架

请参考 [基于React的JS-SDK框架](#)

2.Android和H5通讯流程图



3.H5设备控制集成流程

SDK封装了H5插件下载和原生与H5通讯接口，开发者可以轻松实现H5插件的设备控制。

3.1.上传H5开发包

开发者需要在开放平台上传完整的H5开发包待审核，审核通过之后才可以正常使用。

3.2.初始化webView和H5交互接口

3.2.1.初始化webView

SDK采用了X5内核的浏览服务，添加方式有2种：

第一种：通过xml方式创建布局

```

<com.tencent.smtt.sdk.WebView
    id="@+id/device_h5_web"
    layout_width="match_parent"
    layout_height="match_parent"
/>

```

```

WebView webView = (WebView) findViewById(R.id.device_h5_web);
  
```

第二种：代码动态创建

```
WebView webView = new WebView(mContext);
```

注意：将源码和XML里的系统包和类替换为SDK里的包和类，如：
android.webkit.WebChromeClient 替换成 com.tencent.smtt.sdk.WebChromeClient 。

3.2.2 初始化H5交互接口

```
HtmlFiveManager htmlFiveManager = new HtmlFiveManager(activity, webView, iAppJavaScriptsInterface);
```

HtmlFiveManager是封装了H5与Android原生的交互接口，通过IAppJavaScriptsInterface来暴露H5接口给原生和从原生获取数据。

```
IAppJavaScriptsInterface iAppJavaScriptsInterface = new IAppJavaScriptsInterface() {
    @Override
    public void send(String data, final IMethodCallBack methodCallBack) {
        //H5 het.send()调用的原生接口 这里可以实现设备发送控制命令
    }

    @Override
    public String getModeJson() {
        //H5 het.ready() 获取的原生信息
        return null;
    }

    @Override
    public void onWebViewCreate() {
        //界面加载完成时回调
    }

    @Override
    public void tips(String str) {
        //H5 het.toast()调用的原生接口
    }

    @Override
    public void setTitle(String title) {
        //H5 het.setTitle()调用的原生接口
    }

    @Override
    public void onLoadH5Failed(int errCode, String errMsg) {
        //界面加载失败时回调
    }

    @Override
    public void h5SendDataToNative(int i, String s, String s1, IH5CallBack ih5CallBack) {
        //H5 发送数据到 App 端
    }

    @Override
    public void h5GetDataFromNative(int i, String s, IH5CallBack ih5CallBack) {
        //H5 从 App 端获取数据
    }
};
```

3.3.加载H5插件

```
HetH5Api.getInstance().getH5ControlPlug(context,deviceBean);
```

SDK会加载最新的H5插件（下载和检查更新）。加载成功会抛出
HetH5PlugEvent.HET_EVENT_H5_PLUG_GET_LOCAL_URL_SUCCESS事件，加载失败会抛出
HetH5PlugEvent.HET_EVENT_H5_PLUG_GET_LOCAL_URL_FAILED事件。

3.4. 监听H5插件加载成功与失败

```
RxManage.getInstance().register(HetH5PlugEvent.HET_EVENT_H5_PLUG_GET_LOCAL_URL_SUCCESS +  
model.getProductId(), o -> {  
    if (htmlFiveManager != null) {  
        String localPath = (String) o;  
        String path = Uri.fromFile(new File(localPath)).toString();  
        path += "/index.html";  
        htmlFiveManager.loadUrl(path);  
    }  
});
```

H5插件加载成功，调用htmlFiveManager.loadUrl(path); 加载H5页面展示UI。

3.5. 上传设备数据给H5

上传运行数据：

```
htmlFiveManager.updateRunData(json);
```

上传控制数据：

```
htmlFiveManager.updateConfigData(json);
```

上传异常数据：

```
htmlFiveManager.updateErrorData(json);
```

3.6. 退出释放资源

```
@Override  
public void onDestroy() {  
    if (webView != null) {  
        webView.removeJavascriptInterface("bindJavaScript");  
        if (webView.getSettings() != null) {  
            webView.getSettings().setJavaScriptEnabled(false);  
        }  
        webView.loadDataWithBaseUrl(null, "", "text/html", "utf-8", null);  
        webView.clearHistory();  
        ((ViewGroup) webView.getParent()).removeView(mDevice_h5_web);  
        webView.destroy();  
        webView = null;  
    }  
}
```

```
    }  
}
```

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 16:46:24

- iOS SDK 概述
 - 目录
 - 一、设备接入SDK概述
 - 1. SDK功能简介
 - 2. SDK的总体框架
 - 3. 相关名词定义
 - 二、初始化
 - 1. 注册开放平台账号
 - 2. 下载C-Life iOS SDK文件，并且配置工程
 - 3. 导入硬件模组对应的SDK
 - 4. 注册使用SDK
 - 三、授权登录
 - 1. 授权登录
 - 2. 取消授权登录，退出当前账号
 - 3. 获取用户信息
 - 4. 修改密码
 - 5. 异地登录、accessToken过期通知
 - 四、第三方登录
 - 1. 申请各平台的App key和App secret
 - 2. 导入SDK
 - 3. 项目配置
 - 4. 接入HETOpenSDK 第三方登录
 - 五、设备绑定
 - 1. 绑定概述
 - 2. 获取设备信息
 - 3. WiFi设备绑定方式介绍：
 - 4. 绑定
 - 4.2 wifi设备的AP绑定
 - 4.3 蓝牙设备的绑定
 - 六、WIFI设备控制
 - 1、获取绑定设备列表
 - 2、监听设备状态
 - 3、设备控制
 - 4、设备管理
 - 七、蓝牙设备控制
 - 1. 设备控制流程
 - 2. 控制和监听设备
 - 3. 设备升级
 - 八、iOS 设备分享
 - 1、分享流程
 - 2、面对面分享
 - 3、第三方应用分享
 - 4、接口说明
 - 九、其他接口

- 1.意见反馈

iOS SDK 概述

目录

- [一、设备接入SDK概述](#)
- [二、SDK初始化](#)
- [三、SDK授权](#)
- [四、SDK第三方登录](#)
- [五、SDK设备扫描绑定](#)
- [六、SDKWIFI设备的控制](#)
- [七、SDK蓝牙设备的控制](#)
- [八、SDK设备的分享](#)
- [九、其他接口](#)

一、设备接入SDK概述

1. SDK功能简介

clife开放平台（以下简称开放平台）设备接入的SDK封装了clife对外开放的服务接口，以及手机与智能硬件通讯接口。包括用户模块，设备绑定模块，设备控制模块和其他的开放平台接口。开发者不需要关注这些模块的具体内部逻辑，只需要根据自己的业务需求编写界面和调用SDK接口就可以完成APP的快速开发。

2. SDK的总体框架



3. 相关名词定义

3.1 大循环

智能设备通过路由器或直接接入互联网以实现用户的远程监测与控制，我们称为大循环。

3.2 productId

设备产品号，设备在开放平台管理系统录入设备的时候，系统会根据设备录入的设备大类、设备小类、客户代码、DeviceKey、设备编码生成一个productId，可在开放平台管理系统上看到。

3.3 deviceId

设备号，当一个设备通过设备绑定的接口初次接入开放平台时，开放平台会自动根据productId以及设备的mac地址为此设备注册一个deviceId，此deviceId全网唯一，用于通过开放平台进行设备的操作。

二、初始化

1. 注册开放平台账号

在[C-Life开发平台](#)注册开发者账号，创建应用完善详细资料。此部分请参考《C-Life开发平台使用手册》。

欢迎注册C-Life开放平台

账号

设置密码

确认密码

立即注册

点击「立即注册」按钮将视为您同意<C-Life服务协议>

2. 下载C-Life iOS SDK文件，并且配置工程

2.1 确认本机安装的cocoapods能正常工作

```
pod --help
```

2.2 编辑Podfile文件

```
pod 'HETOpenSDK', '2.0.0'
```

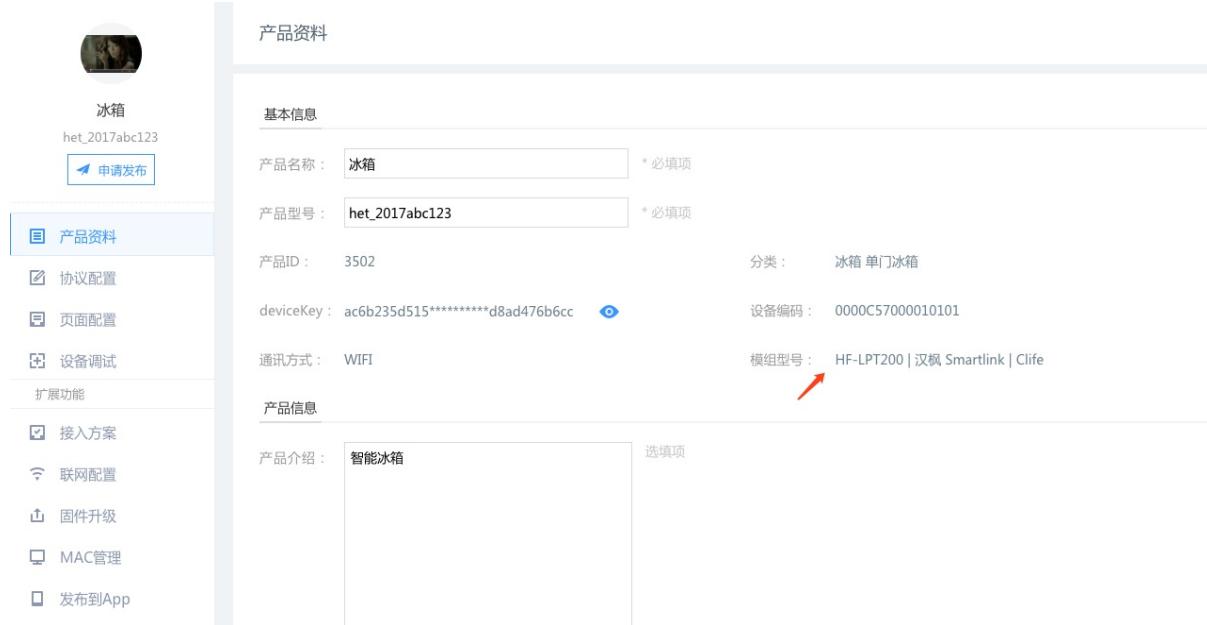
2.3 安装

以下两种方式任选一种就可以：

```
pod install  
pod update
```

3. 导入硬件模组对应的SDK

3.1 根据产品类型找到对应的芯片模组名称，如下：



3.2 在podfile中导入对应的sdk，并且安装，对应类表如下：

```

source 'https://github.com/C-Life/HETSDKSpecs.git'
source 'https://github.com/CocoaPods/Specs.git'

##支持真机调试和模拟器调试
# 汉枫-WiFi模组
pod 'HETPublicSDK_WiFiModule/HF_LPT100_V3','1.0.0'
# 乐鑫-WiFi模组
pod 'HETPublicSDK_WiFiModule/ESP8266','1.0.0'
# COOEE-WiFi模组
pod 'HETPublicSDK_WiFiModule/COOEE','1.0.0'
# MarvellV2-WiFi模组
pod 'HETPublicSDK_WiFiModule/Marvell_MW300_V2','1.0.0'

##支持真机调试
# TI-WiFi模组
pod 'HETPublicSDK_WiFiModule/TI_CC3200R2',      '1.0.0'
# 科中龙-WiFi模组
pod 'HETPublicSDK_WiFiModule/Realtek8711AF',      '1.0.0'
# 信驰达-WiFi模组
pod 'HETPublicSDK_WiFiModule/MTK7681',      '1.0.0'
# 信驰达-WiFi模组
pod 'HETPublicSDK_WiFiModule/MTK7687',      '1.0.0'
# 新力维-WiFi模组
pod 'HETPublicSDK_WiFiModule/NL6621',      '1.0.0'

```

备注：在使用了Wifi模组后，就不再支持模拟器调试。

4. 注册使用SDK

4.1 在AppDelegate中如下地方添加，注册使用SDK，打开Log

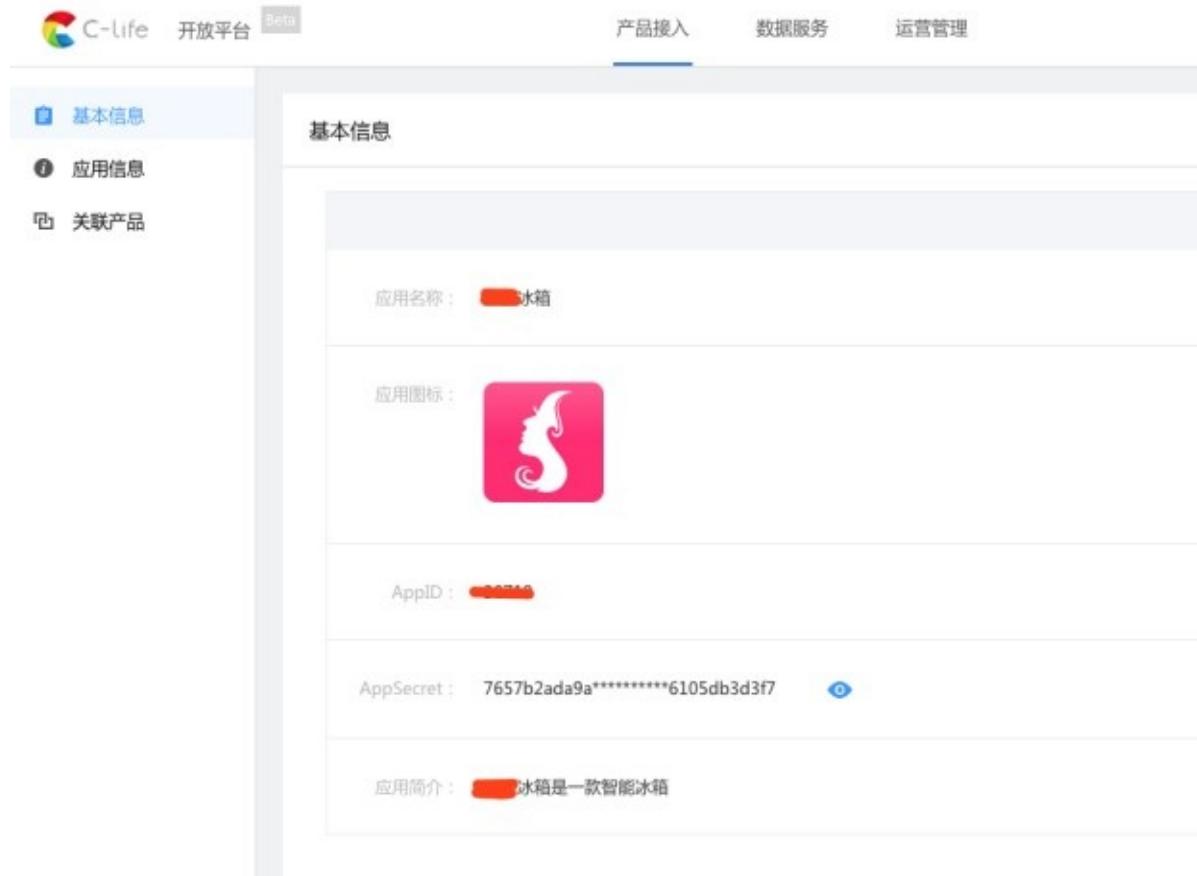
```

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

```

```
// 1.注册HET开发平台SDK
[HETOpenSDK registerAppId:@"yourAppId" appSecret:@"yourAppSecret"];
[HETOpenSDK openLog:YES];
return YES;
}
```

yourAppId、**yourAppSecret** 的值是在“应用创建”时生成的 **AppID**、**AppSecret**。在如下图查看：



注意:如果网络请求出现AppID不合法,请检查Xcode工程里面的BundleId和appId, 必须跟在开放平台创建应用时填的BundleId和AppID保持一致。

三、授权登录

参考 `HETAuthorize` 类里面方法,调用 `authorizeWithCompleted` 接口会弹出授权登录的界面, 登录成功后接口返回openId (授权用户唯一标识) 可用于与自己平台的账户体系关联。

1. 授权登录

【示例代码】

```
HETAuthorize *auth = [[HETAuthorize alloc] init];
self.auth = auth;
if (![self.auth isAuthenticated]) {
    [self.auth authorizeWithCompleted:^(NSString *openId, NSError *error) {
    }];
}
```



2. 取消授权登录，退出当前账号

【示例代码】

```
// 在授权登录成功的情况才执行操作
if ([self.auth isAuthenticated]) {
    [self.auth unauthorize];
}
```

3. 获取用户信息

【示例代码】

```
WEAKSELF
[HETAuthorize getUserInformationSuccess:^(id responseObject) {
```

```

} failure:^(NSError *error) {
    NSLog(@"error ==%@", error);
}];

```

接口返回的结果数据

```

{
    "code":0,
    "data":{
        "userId": "d09f572c60ffced144d6fc55a6881b9",
        "userName": "葫芦娃",
        "email":"",
        "phone":"",
        "sex": 1,
        "birthday": "2014-12-31",
        "weight": 48000,
        "height": 163,
        "avatar": "",
        "city": "深圳"
    }
}

```

字段名称	字段类型	字段说明
userName	string	用户名
phone	string	用户手机
email	string	用户邮箱
sex	number	性别 (1-男, 2-女)
birthday	string	生日 (yyyy-MM-dd)
weight	number	体重 (克)
height	number	身高 (厘米)
avatar	string	头像URL
city	string	城市名

4. 修改密码

【示例代码】

```

HETAuthorize *auth = [[HETAuthorize alloc] init];
[auth changePasswordSuccess:^(id responseObject) {

} failure:^(NSError *error) {

}];

```



5. 异地登录、accessToken过期通知

开放平台的账号只能在一台设备上面登录。当有账号在另一台设备登录时，SDK会抛出一个HETLoginOffNotification消息。开发者可以在首页监听这个消息，处理异地登录的逻辑。例：

【示例代码】

```
[[NSNotificationCenter defaultCenter] addObserver:self  
selector:@selector(XXX) name:HETLoginOffNotification object: nil];
```

四、第三方登录

1. 申请各平台的App key和App secret

注意：app bundleId跟各平台注册的时候一致。

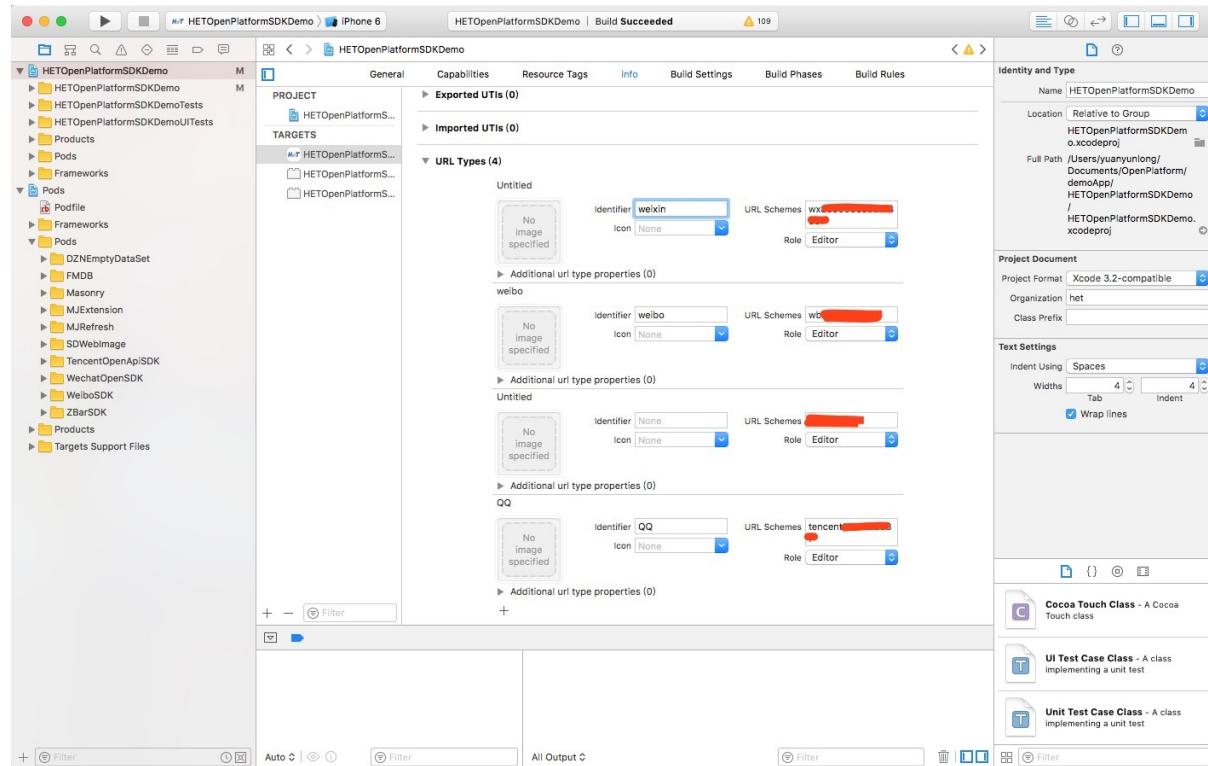
微信接入地址

2. 导入SDK

```
pod 'WechatOpenSDK', '1.7.7'
pod 'WeiboSDK', '3.1.3'
pod 'TencentOpenApiSDK', '2.9.5'
```

3. 项目配置

3.1 URLScheme 配置



3.2 针对iOS9+, 添加白名单

在Info.plist文件中加入 LSApplicationQueriesSchemes

▼ LSApplicationQueriesSchemes	◆ Array	(30 items)
Item 0	String	mqqapi
Item 1	String	mqq
Item 2	String	mqqOpensdkSSoLogin
Item 3	String	mqqconnect
Item 4	String	mqqopensdkdataline
Item 5	String	mqqopensdkgrouptribeshare
Item 6	✚ - String	mqqopensdkfriend
Item 7	String	mqqopensdkapi
Item 8	String	mqqopensdkapiV2
Item 9	String	mqqopensdkapiV3
Item 10	String	mqzoneopensdk
Item 11	String	wtloginmqq
Item 12	String	wtloginmqq2
Item 13	String	mqqwpa
Item 14	String	mqzone
Item 15	String	mqzonev2
Item 16	String	mqzoneshare
Item 17	String	wtloginqzone
Item 18	String	mqzonewx
Item 19	String	mqzoneopensdkapiV2
Item 20	String	mqzoneopensdkapiV9
Item 21	String	mqzoneopensdkapi
Item 22	String	mqzoneopensdk
Item 23	String	weixin
Item 24	String	wechat
Item 25	String	sinaweibohd
Item 26	String	sinaweibo
Item 27	String	sinaweibosso
Item 28	String	weibosdk
Item 29	String	weibosdk2.5

```

<key>LSApplicationQueriesSchemes</key>
<array>
    <!-- 微信 URL Scheme 白名单-->
    <string>wechart</string>
    <string>weixin</string>

    <!-- 新浪微博 URL Scheme 白名单-->
    <string>sinaweibohd</string>
    <string>sinaweibo</string>
    <string>sinaweibosso</string>
    <string>weibosdk</string>
    <string>weibosdk2.5</string>

    <!-- QQ、Qzone URL Scheme 白名单-->
    <string>mqqapi</string>
    <string>mqq</string>
    <string>mqqOpensdkSSoLogin</string>
    <string>mqqconnect</string>
    <string>mqqopensdkdataline</string>
    <string>mqqopensdkgrouptribeshare</string>
    <string>mqqopensdkfriend</string>
    <string>mqqopensdkapi</string>
    <string>mqqopensdkapiV2</string>
    <string>mqqopensdkapiV3</string>
    <string>mqzoneopensdk</string>
    <string>wtloginmqq</string>

```

```

<string>wtloginmqqq2</string>
<string>mqqwpac</string>
<string>mqzone</string>
<string>mqzonenv2</string>
<string>mqzoneshare</string>
<string>wtloginqzone</string>
<string>mqzonewx</string>
<string>mqzoneopensdkapiV2</string>
<string>mqzoneopensdkapi19</string>
<string>mqzoneopensdkapi</string>
<string>mqzoneopensdk</string>

</array>

```

3.4 针对iOS9默认使用https,现在先还原成http请求方式。

在Info.plist中添加NSAppTransportSecurity类型Dictionary。

在NSAppTransportSecurity下添加NSAllowsArbitraryLoads类型Boolean,值设为YES

第一步：在plist中添加NSAppTransportSecurity项，此项为NSDictionary

第二步：在NSAppTransportSecurity下添加 NSAllowsArbitraryLoads类型为Boolean, value为YES

4. 接入HETOpenSDK 第三方登录

4.1 在appdelegate.m中，添加代码

第一步：注入appkey

【示例代码】

```

[HETOpenSDK setPlatform:HETAuthPlatformType_QQ appKey:QQ_APP_ID appSecret:nil redirectURL:nil];
[HETOpenSDK setPlatform:HETAuthPlatformType_Weibo appKey:WB_APP_KEY appSecret:nil redirectURL:nil];
[HETOpenSDK setPlatform:HETAuthPlatformType_Wechat appKey:WX_APP_KEY appSecret:WX_APP_SECRET redirectURL:nil];

```

第二步：添加请求方法

【示例代码】

```

- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    BOOL result = [HETOpenSDK application:application openURL:url sourceApplication:sourceApplication annotation:annotation];
    return result;
}

- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
{
    BOOL result = [HETOpenSDK handleOpenURL:url];
    return result;
}

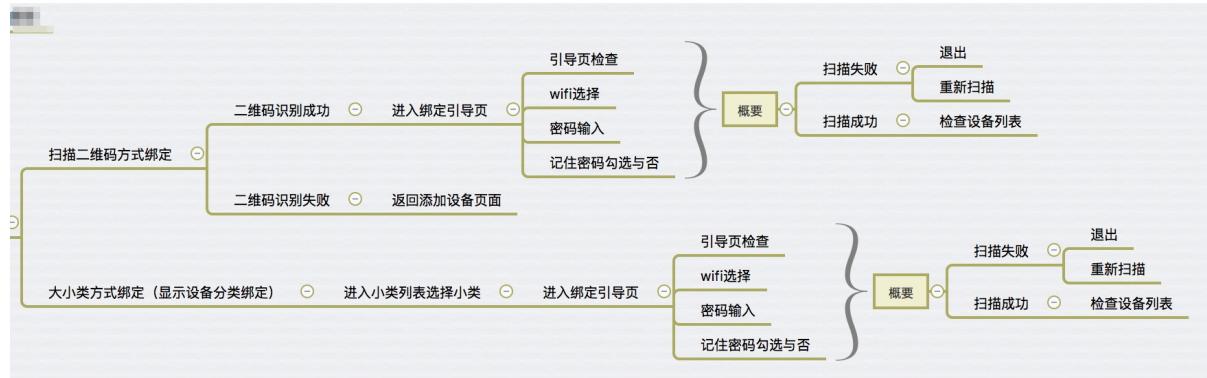
```

}

五、设备绑定

1. 绑定概述

1.1 绑定流程



1.2 设备分类

从设备层级上分为 **设备大类** 和 **设备小类**。例如，冰箱是大类，冰箱下有Clife智能冰箱，即小类。

从设备类型上分为 **蓝牙设备** 和 **wifi设备**，当我们拿到 **设备的信息** 的时候，就能区别设备是wifi设备还是蓝牙设备。

wifi设备绑定方式为 **smartLink绑定** 或者 **AP绑定**；蓝牙设备绑定方式为 **蓝牙绑定**。

smartLink、**AP**和**蓝牙绑定**凡是根据 **moduleType** 来区分，如下图所示：

moduleType	绑定类型
1	wifi设备 SmartLink绑定
2	蓝牙设备
9	wifi设备 AP绑定

蓝牙设备暂时只有一种绑定方式（蓝牙绑定）

2. 获取设备信息

2.1 扫描二维码获取设备信息

2.1.1 二维码命名规则

二维码命名规则：

```
http://open.clife.net/v1/web/open/product?param={"a":产品id}
```

2.1.2 获取产品ID

扫一扫内容：

```
urlStr: http://open.clife.net/v1/web/open/product?param={"a":3531}
```

"a":3531 3531 即是 产品ID

2.1.3 根据产品ID获取设备信息

获取产品信息，区分设备绑定类型。

【示例代码】

```
WEAKSELF
[HETDeviceRequestBusiness fetchDeviceInfoWithProductId:productId success:^(id responseObject) {

    if ([[responseObject allKeys] containsObject:@"data"]) {
        NSDictionary *dataDict = [responseObject valueForKey:@"data"];

        HETDevice *device = [HETDevice mj_objectWithKeyValues:dataDict];
        // wifi绑定
        if ([device.moduleType integerValue] == 1
            || [device.moduleType integerValue] == 9) {
            }
        // 蓝牙绑定
        if ([device.moduleType integerValue] == 2) {
            }
    }
} failure:^(NSError *error) {

}];
```

返回结果

正确的Json返回结果：

```
{
  "code": 0,
  "data": {
    "productId": 6,
    "deviceSubtypeId": 3,
    "deviceSubtypeName": "多门冰箱",
    "productIcon": "http://200.200.200.50/v1/device/icon",
    "productName": "通用冰箱",
    "productCode": "CC-1003",
    "moduleId": 2,
    "moduleName": "TI CC3200R2",
    "moduleType": 1,
    "deviceTypeId": 1,
    "radiocastName": "HET_",
    "deviceCode": "00000199000000301"
  }
}
```

字段名称	字段类型	字段说明
productId	int	产品ID
deviceSubtypeId	int	设备子分类ID
deviceSubtypeName	string	设备子分类名称
productIcon	string	产品LOGO
productName	string	设备接入秘钥
productCode	string	产品型号
moduleId	int	芯片模块ID
moduleName	string	芯片模块名称
moduleType	int	模块类型 (1-WiFi, 2-蓝牙, 3-音频, 4-GSM, 5-红外)
deviceTypeId	int	设备分类ID
radiocastName	string	设备广播名
deviceTypeName	string	设备分类名称
deviceTypeName	string	设备分类名称
deviceCode	string	设备编码

- 到此，在上图中已经获取到 **productId**、**moduleType**、**deviceTypeId**、**deviceSubtypeId** 可以进行设备绑定。

2.2 通过大类小类获取设备信息

2.2.1 获取设备大类列表

在 **HETDeviceRequestBusiness** 查询设备信息获取相关接口

【示例代码】

```
[HETDeviceRequestBusiness fetchDeviceTypeListSuccess:^(id responseObject)
{
    } failure:^(NSError *error) {
    }];
}];
```

返回结果

正确的Json返回结果：

```
{
    "data": [
        {
            "deviceTypeId": 1,
            "deviceTypeName": "冰箱",
            "deviceTypeIcon": null
        }
    ],
    "code": 0
}
```

字段名称	字段类型	字段说明
deviceTypeId	number	设备大分类标识
deviceTypeName	string	设备大分类名称
deviceTypeIcon	string	设备图标

2.2.2 通过大类ID，获取设备小类

【示例代码】

```
[HETDeviceRequestBusiness fetchDeviceProductListWithDeviceTypeId:
[NSString stringWithFormat:@"%@",deviceTypeId] success:^(id responseObject) {
    } failure:^(NSError *error) {
    }];
}];
```

返回结果

正确的Json返回结果：

```
{
  "data": [
    {
      "deviceTypeId": 1,
      "deviceSubtypeId": 1,
      "deviceSubtypeName": "冰箱",
      "productId": 1,
      "productName": "惠而浦-KST",
      "productCode": "kst-001",
      "productIcon": "http://200.200.200.50/v1/device/icon/1.png",
      "moduleId": 3,
      "moduleType": 1,
      "moduleName": "汉枫",
      "remark": "备注",
      "radiocastName": null,
      "ssid": null,
      "ssidPassword": null,
      "deviceCode": "0000C3AA00010105"
    },
    "code": 0
  }
}
```

字段名称	字段类型	字段说明
deviceTypeId	number	设备大分类标识
deviceSubtypeId	number	设备子分类标识
deviceSubtypeName	string	设备子分类名称
productId	number	设备型号标识
productName	string	设备型号名称
productCode	string	设备型号编码
productIcon	string	设备型号图标
moduleId	number	模块标识
moduleType	number	模块类型 (1-WiFi, 2-蓝牙, 3-音频, 4-GSM, 5-红外)
moduleName	string	模块名称
remark	string	备注
radiocastName	string	设备广播名
ssid	string	【2017-05-18新增】ssid
ssidPassword	string	【2017-05-18新增】ssid密码
deviceCode	string	【2016-06-03新增】设备编码

到此，在上图中已经获取到 productId、moduleType、deviceTypeId、deviceSubtypeId 可以进行设备绑定。

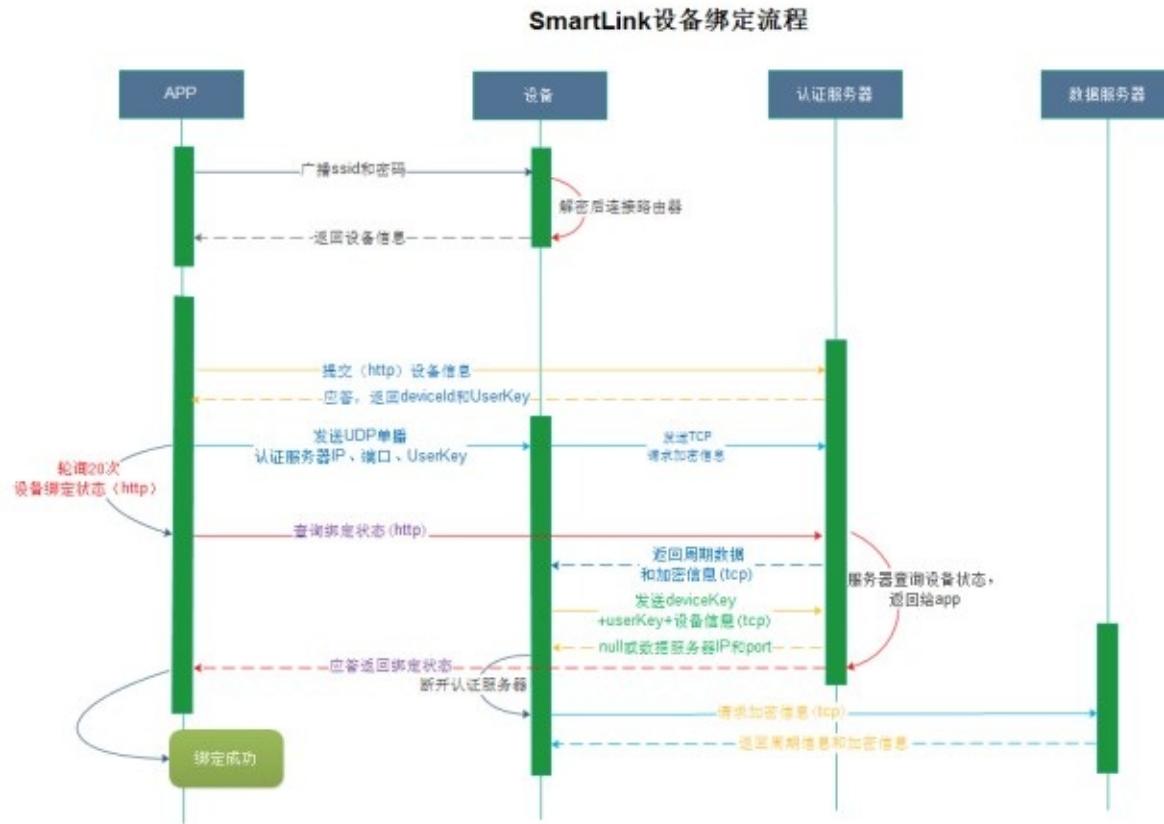
3. WiFi设备绑定方式介绍：

3.1 smartLink绑定

在开始配置前，设备要先进入配置模式，然后APP发送要配置的路由器ssid和密码，开启扫描设备服务将扫描到的设备进行绑定，获取绑定结果。

第一步：获取路由器ssid和密码

第二步：传入参数产品ID **productId**，路由器ssid 和 密码，启动绑定流程



3.2 AP绑定

在开始配置前，设备进入配置模式后，会产生一个Wifi热点。手机连接设备热点，将发送要配置的路由器ssid和密码给设备，然后APP将配置信息给设备，之后设备自行于服务器绑定，APP想服务器查询绑定状态。

使用C-life提供的模组固件，设备产生的Wifi热点以“HET-xxx”开头，没有密码。其他厂商提供的模组，SoftAP热点名称由各自厂商指定。

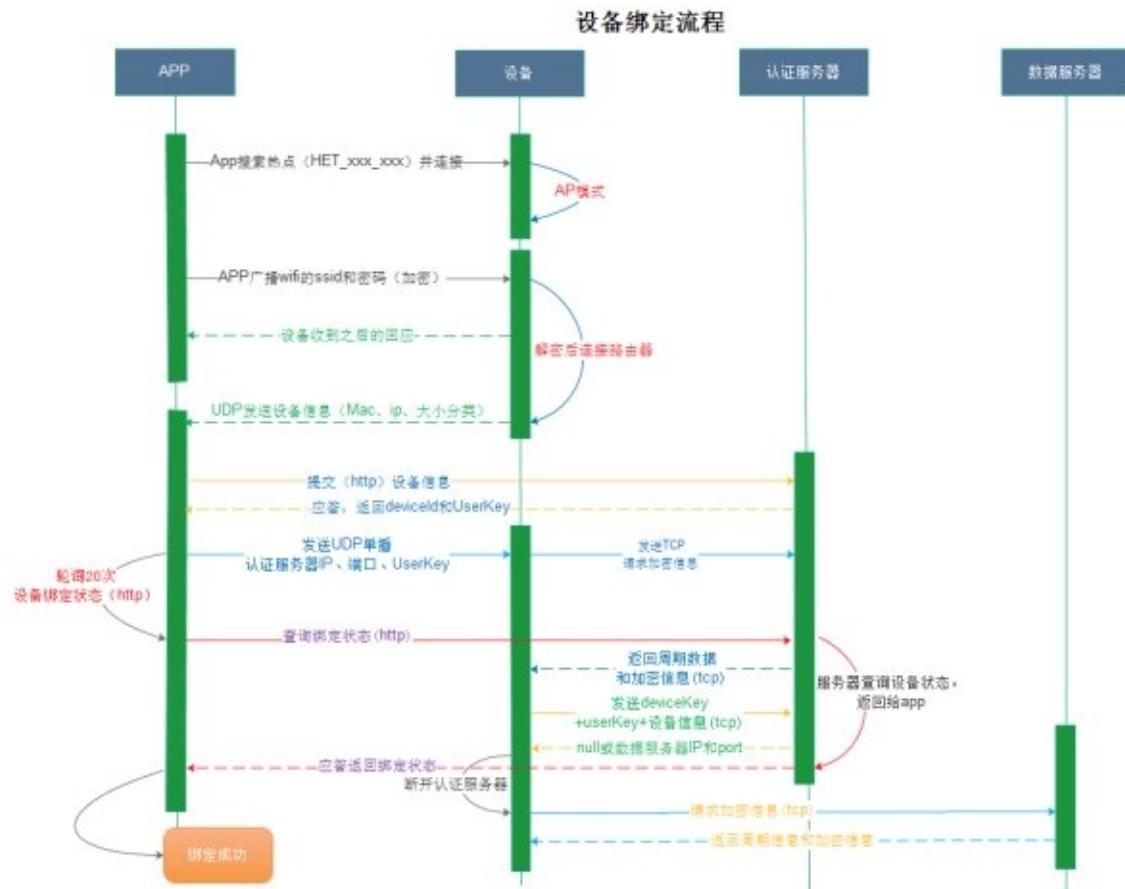
AP绑定的交互流程：

第一步：获取路由器ssid和密码

第二步：手机连接路由器热点

第三步：手机切换设备热点

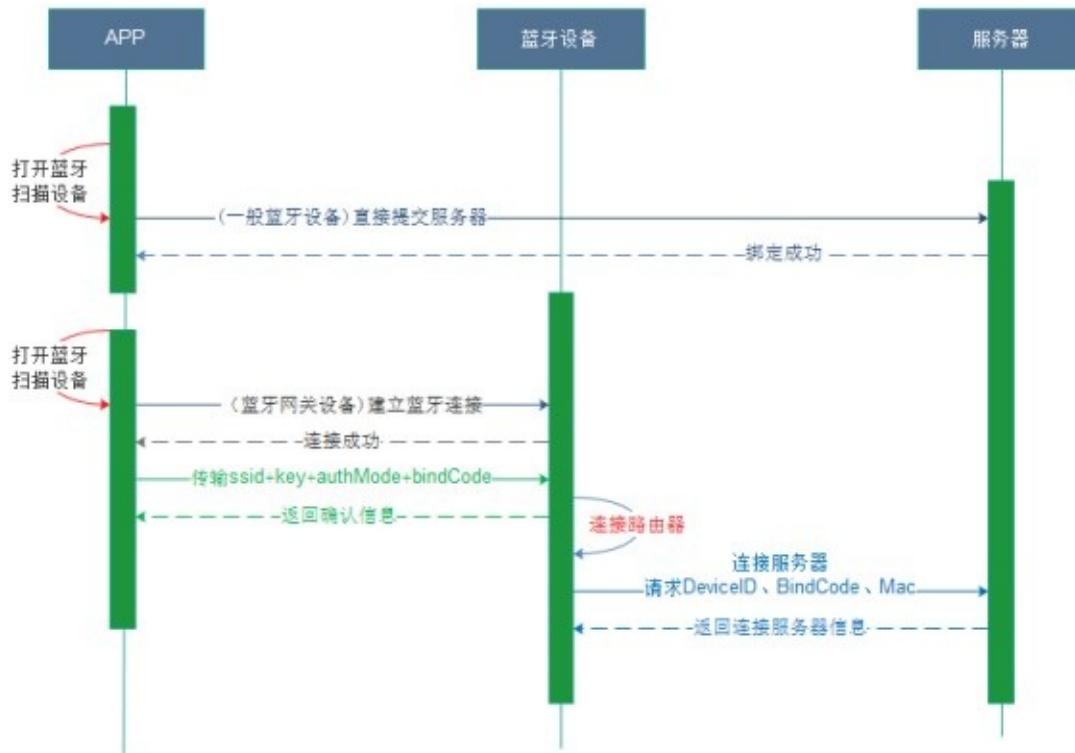
第四步：传入参数 产品ID **productId**、设备大类ID、设备小类ID、路由器ssid 和 密码，启动绑定流程



3.3 蓝牙设备绑定流程

第一步：传入参数 产品ID **productId**、设备大类ID、设备小类ID，初始化 第二步：启动绑定流程

蓝牙设备绑定



4. 绑定

4.1 wifi设备的smartLink绑定

4.1.1 通过 HETWIFIBindBusiness 获取路由器ssid

【示例代码】

```
NSString *macAddr = [[HETWIFIBindBusiness sharedInstance] fetchmacSSIDInfo];
```

4.1.2 传入参数，通过 HETWIFIBindBusiness 调用接口，启动绑定

【示例代码】

```
[[HETWIFIBindBusiness sharedInstance]
startSmartLinkBindDeviceWithProductId:[NSString stringWithFormat:@"%@",self.device.productId]
withSSID:self.ssid withPassWord:self.password withTimeOut:timeOut
bindHandler:^(HETWiFiDeviceBindState state, HETDevice *deviceObj
, NSError *error) {
    NSLog(@"HETWiFiDeviceBindState: %ld", state);

    if (error) {
        // 扫描失败
    }else{
        // 扫描成功
    }
}];
```

```

    }
};


```

4.2 wifi设备的AP绑定

4.2.1 校验用户是否连接设备

设备AP热点命名规则: `radiocastName_deviceTypeId_deviceSubtypeId`

当判断用户已经成功连接设备AP热点，即可进入绑定流程。

4.2.2 通过 `HETWIFIBindBusiness` 调用接口， 启动绑定

【示例代码】

```

NSString *productId = [NSString stringWithFormat:@"%@",self.device.productId];
NSString *typeId = [NSString stringWithFormat:@"%@",self.device.deviceTypeId];
NSString *subType = [NSString stringWithFormat:@"%@",self.device.deviceSubtypeId];

[[HETWIFIBindBusiness sharedInstance] startAPBindDeviceWithProductId:productId
                                                       withDeviceTypeId:typeId
                                                       withDeviceSubtypeId:subType
                                                       withSSID:self.ssid
                                                       withPassWord:self.password
                                                       withTimeOut:timeOut
bindHandler:^(HETWiFiDeviceBindState state, HETDevice *deviceObj, NSError *error) {
    OPLog(@"HETWiFiDeviceBindState: %ld", state);

    if (error) {

        }else{
            [weakSelf doSomeThingWithState:state deviceObj:deviceObj];
        }
    }];

```

4.3 蓝牙设备的绑定

蓝牙的扫描绑定主要看**HETBLEBusiness**相关接口

4.3.1 初始化**HETBLEBusiness**对象， 启动绑定流程

【示例代码】

```

//初始化蓝牙设备的业务类，需要设备的productId, deviceTypeId, deviceSubtypeId
self.bleBusiness = [[HETBLEBusiness alloc] initWithProductId:self.productId.integerValue
deviceTypeId:self.deviceTypeId.integerValue
deviceSubtypeId:self.deviceSubtypeId.integerValue];

```

开始扫描蓝牙设备，扫描到的蓝牙设备，用tableView显示出来，给用户选择。

【示例代码】

```

WEAKSELF
[self.bleBusiness scanForPeripheralsWithTimeOut:timeOut name:nil mac:nil scanForPeripheral
sBlock:^(NSArray<CBPeripheral *> *peripherals, NSError *error) {
if (error) {
    // 停止扫描
    return;
}
if (peripherals) {
    // 返回一个设备数组
    [peripherals enumerateObjectsUsingBlock:^(id _Nonnull obj, NSUInteger idx, BOOL * _Nonnull stop) {
        // 过滤重复的设备，并且刷新蓝牙设备列表
    }];
    return;
}
}];

```

4.3.2 绑定设备

选择需要绑定的设备，启动绑定流程。

【示例代码】

```

[self.bleBusiness bindBleDeviceWithPeripheral:cbp macAddress:nil completionHandler:^(NSString *deviceId, NSError *error) {
[weakself.bleBusiness disconnectWithPeripheral:cbp];
if(error) {
    // 填写绑定失败的代码
}
else
{
    // 填写绑定成功的代码
}
}];

```

六、WIFI设备控制

参考 `HETDeviceControlBusiness` 类里面方法，实现设备控制和运行状态的监听。

参考 `HETDeviceRequestBusiness` 类里面的方法，获取设备的信息。

控制设备的流程如下：

第一步：获取已绑定的设备列表，获取设备信息（`HETDevice`）。

第二步：根据获取的设备信息，监听设备状态，控制设备。

1、获取绑定设备列表

绑定成功后，用户可以获取绑定成功的设备列表，获取到设备列表拿到设备的`HETDevice`设备信息才可以控制设备

【示例代码】

```
[HETDeviceRequestBusiness fetchAllBindDeviceSuccess:^(NSArray<HETDevice *> *deviceArray) {
    NSLog(@"responseObject ==%@",deviceArray);
} failure:^(NSError *error) {
    NSLog(@"error ==%@",error);
}];
```

字段名称	字段类型	字段说明
deviceid	string	设备标识
macAddress	string	MAC地址
deviceBrandId	number	设备品牌标识
deviceBrandName	string	设备品牌名称
deviceTypeId	number	设备大分类标识
deviceTypeName	string	设备大分类名称
deviceSubtypeId	number	设备子分标识
deviceSubtypeName	string	设备子分类名称
deviceName	string	设备名称
roomId	string	房间标识
roomName	string	房间名称
authUserId	string	授权设备用户标识
bindTime	string	绑定时间
onlineStatus	number	在线状态 (1-正常, 2-异常)
share	number	设备分享 (1-是, 2-否, 3-扫描分享) 【2015-11-11新增状态 (3)】
controlType	number	控制类型 (1-原生, 2-插件, 3-H5插件)
userKey	string	MAC与设备ID生成的KEY
productId	number	设备型号标识
productName	string	设备型号名称
productCode	string	设备型号编码
productIcon	string	设备型号图标
moduleId	number	模块标识
moduleType	number	模块类型 (1-WiFi, 2-蓝牙, 3-音频, 4-GSM, 5-红外, 6-直连, 8-zigbee, 9-ap模式)
moduleName	string	模块名称
radiocastName	string	设备广播名
deviceCode	string	【2016-06-03新增】 设备编码
guideUrl	string	【2017-09-27新增】 引导页URL

2、监听设备状态

2.1、初始化

初始化HETDeviceControlBusiness的实例对象，传递需要监听的设备信息作为参数，监听block的回调信息，做相应的业务逻辑。对于运行数据、控制数据、错误数据的内容，请参考具体设备的配置协议内容。

【示例代码】

```
- (HETDeviceControlBusiness *)controlBusiness
{
    if (!_controlBusiness) {
        WEAKSELF
        _controlBusiness = [[HETDeviceControlBusiness alloc] initWithHetDeviceModel:self.device deviceRunData:^(id responseObject) {
            // 监听设备运行数据, responseObject请具体参考协议配置。
            OPLog(@"deviceRunData:@%@ ",responseObject);

        } deviceCfgData:^(id responseObject) {
            // 监听设备控制数据
            OPLog(@"deviceCfgData:@%@ ",responseObject);

        } deviceErrorData:^(id responseObject) {
            // 监听设备错误数据
            OPLog(@"deviceErrorData:@%@ ",responseObject);

        } deviceState:^(HETWiFiDeviceState state) {
            // 监听设备在线状态数据
            OPLog(@"deviceState:@%ld ",(long)state); //deviceState:2

        }];
    }
    return _controlBusiness;
}
```

2.1、启动监听服务

【示例代码】

```
- (void)viewWillAppear:(BOOL)animated
{
    [self.controlBusiness start];
}
```

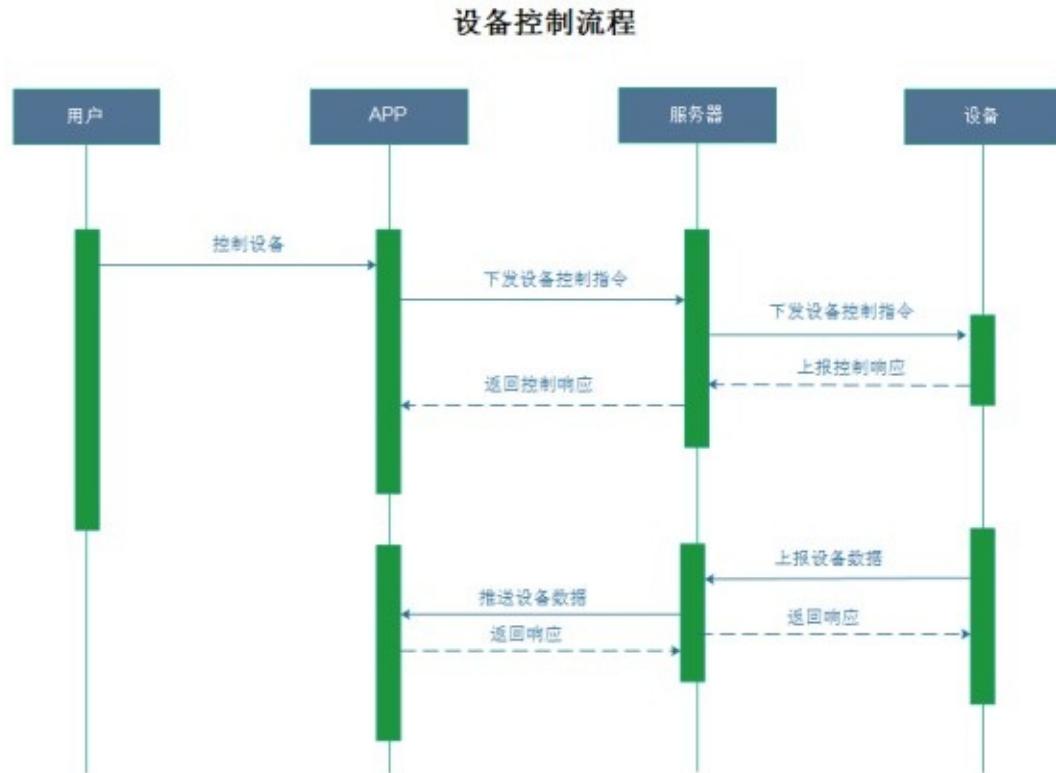
2.3、停止监听服务

【示例代码】

```
- (void)viewWillDisappear:(BOOL)animated
{
    [self.controlBusiness stop];
}
```

3、设备控制

设备控制流程如下：



【示例代码】

```

- (void)configDataSetWithColorTemp:(NSDictionary *)dict{
    NSData * jsonData = [NSJSONSerialization dataWithJSONObject:dict options:Nil prettyPrinted:error:nil];
    NSString * jsonStr = [[NSString alloc] initWithData:jsonData encoding:NSUTF8StringEncoding];
    [self.controlBusiness deviceControlRequestWithJson: jsonStr withSuccessBlock:^(id responseObject) {
        NSLog(@"responseObject = %@",responseObject);
    } withFailBlock:^(NSError *error) {
        NSLog(@"error = %@",error);
    }];
}
    
```

关于updateflag

这个修改标记位是为了做统计和配置下发的时候设备执行相应的功能。下发数据必须传递updateflag标志。

例如，空气净化器（广磊K180）配置信息协议：

紫外线(1)、负离子(2)、臭氧(3)、儿童锁(4)、开关(5)、WiFi(6)、过滤网(7)、模式(8)、定时(9)、风量(10)上面一共上10个功能，那么updateFlag就2个字节，没超过8个功能为1个字节，超过8个为2个字节，超过16个为3个字节，以此类推。

打开负离子，2个字节，每一个bit的值为下：

0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0

4、设备管理

4.1、解绑设备

设备删除有2中情况,需要自己根据设备分享类型 (device.share) 来区分：第一种：设备是用户自己绑定的设备。调用 `unbindDeviceWithDeviceId: success: failure:` 来解除绑定。

【示例代码】

```
[HETDeviceRequestBusiness unbindDeviceWithDeviceId:device.deviceId success:^(id responseObject) {
    } failure:^(NSError *error) {
}];
```

第二种：设备是别人分享过来的设备。调用 `HetDeviceShareApi.getInstance().deviceDel()` 方法来解绑分享关系。

【示例代码】

```
[HETDeviceShareBusiness deviceAuthDelWithDeviceId:device.deviceId userId:@"" success:^(id responseObject) {
    } failure:^(NSError *error) {
}];
```

4.2、修改设备信息

修改设备信息，用户可以修改设备的名称

【示例代码】

```
NSString *deviceId=self.hetDeviceModel.deviceId;
[HETDeviceRequestBusiness updateDeviceInfoWithDeviceId:deviceId
deviceName:@"123fsdg" roomId:@"12" success:^(id responseObject) {
    } failure:^(NSError *error) {
}];
```

七、蓝牙设备控制

蓝牙设备控制，参考 `HETBLEBusiness` 类里面的方法和实现。

1. 设备控制流程

第一步、获取已绑定的设备列表，从中获取某个设备信息 `HETDevice`。

第二步、根据获取的设备信息，监听设备状态，控制设备。



1.1 获取绑定设备列表

【示例代码】

```

[HETDeviceRequestBusiness fetchAllBindDeviceSuccess:^(NSArray<HETDevice *> *deviceArray) {
    NSLog(@"responseObject ==%@",deviceArray);

} failure:^(NSError *error) {
    NSLog(@"error ==%@",error);

}];

```

2. 控制和监听设备

2.1 初始化

【示例代码】

```

- (void)setDevice:(HETDevice *)device
{
    _device = device;
    // 设备控制需要的设备信息
    _macAddress = device.macAddress;
    _deviceType = device.deviceTypeId.integerValue;
    _deviceSubType = device.deviceSubtypeId.integerValue;
    _productId = device.productId.integerValue;
    _deviceId = device.deviceId;
}

-(void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];

    if(!_bleBusiness)
    {
        _bleBusiness=[[HETBLEBusiness alloc]init]initWithProductId:self.productId
            deviceId:self.deviceType deviceSubtypeId:self.deviceSubType];
    }
}

```

2.2 监听获取设备状态

【示例代码】

```

WEAKSELF;
[_bleBusiness fetchStatusDataWithPeripheral:self.blePeripheral
macAddress:self.macAddress deviceId:self.deviceId
completionHandler:^(CBPeripheral *currentPeripheral,NSDictionary *dic, NSError *error) {
    STRONGSELF;
    strongSelf.blePeripheral=currentPeripheral;
    NSLog(@"状态数据:@%@,%@",dic,error);
    if(dic)
    {
        uint8_t state ;
        UInt8 color;
        NSString *colorStr=dic[@"LED"];
        NSString *stateStr=dic[@"LIGHT"];
        color=colorStr.intValue;//@"MIST":@"0",@"LIGHT":@"1",@"LED"
        state=stateStr.intValue;
        strongSelf->ledColor = color%9;
        strongSelf->ledState = state;
        if (state == 3) {
            [strongSelf changeLED_color:0];//灯是关掉的
        }else if(state == 1) {
            [strongSelf changeLED_color:color];
        }
    }
}];

```

2.3 控制设备

【示例代码】

```
[_bleBusiness deviceControlRequestWithPeripheral:self.blePeripheral
macAddress:self.macAddress
sendDic:@{"LED":@(ledColor %9)}
completionHandler:^(CBPeripheral *currentPeripheral, NSError *error) {
    STRONGSELF;
    strongSelf.blePeripheral=currentPeripheral;
    NSLog(@"数据发送回调:%@",error);
}];
```

3. 设备升级

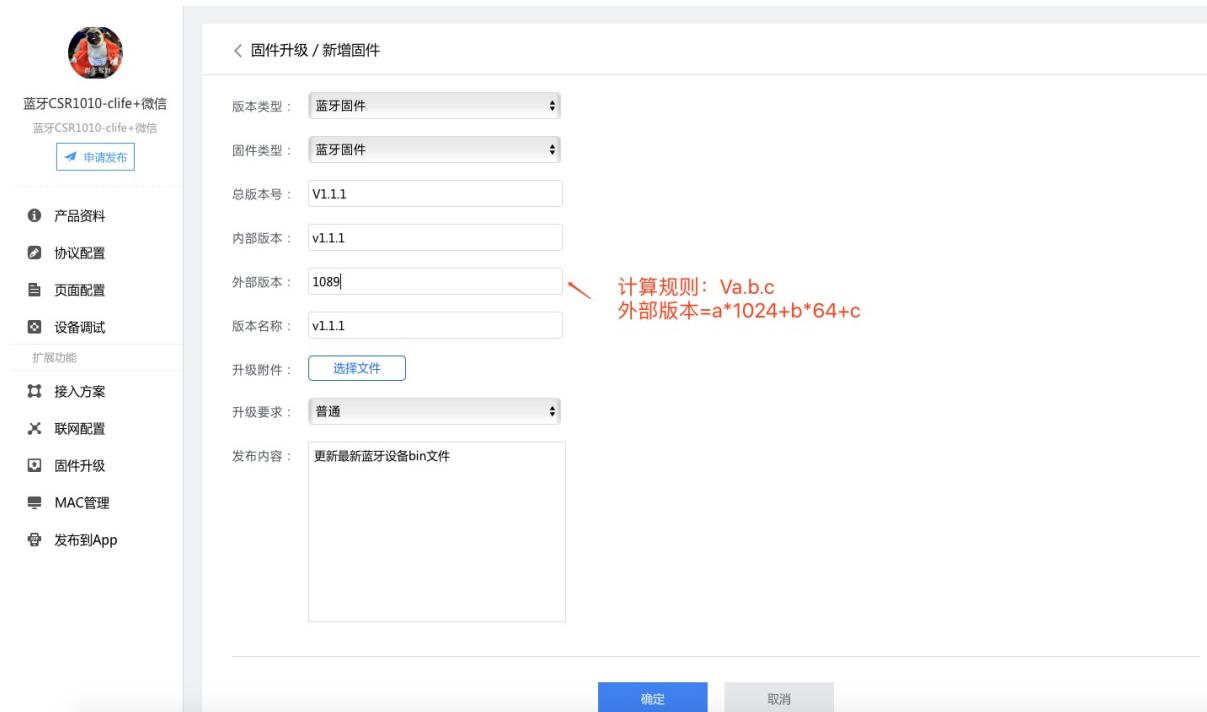
- 从平台上传最新硬件版本
- 从服务器获取设备最新版本
- 下发最新版本给蓝牙设备

3.1 上传包文件

第一步：登录开发平台，进去产品页面

版本标识	版本类型	模块版本类型	产品名称	总版本号	内部版本	外部版本	发布时间	发布内容	操作
2258	WiFi	Bootloader	蓝牙CSR1010-...	0	0	1.0.0	2017-07-13 18...	设备绑定生成...	编辑 删除
2024	PCB	控制板固件	蓝牙CSR1010-...	V_1.1.2	1058	V1.1.2	2017-04-18 11...	测试用例，固件...	编辑 删除
2030	PCB	控制板固件	蓝牙CSR1010-...	0	1024	1.0.0	2017-04-20 16...	设备绑定生成...	编辑 删除
2031	PCB	控制板固件	蓝牙CSR1010-...	0	0	1.0.0	2017-04-20 17...	设备绑定生成...	编辑 删除
2126	PCB	控制板固件	蓝牙CSR1010-...	0	1088	1.2.0	2017-05-23 19...	设备绑定生成...	编辑 删除
2352	PCB	控制板固件	蓝牙CSR1010-...	0	1056	1.1.0	2017-08-24 14...	设备绑定生成...	编辑 删除
2425	PCB	控制板固件	蓝牙CSR1010-...	v1.2.1	1089	v1.2.1	2017-09-18 10...	升级测试	编辑 删除

第二步：填写最新版本和选择包文件并且上传



3.2 app检查固件版本，是否存在固件更新

【示例代码】

```
// 获取最新版本
[HETDeviceUpgradeBusiness deviceUpgradeCheckWithDeviceId:self.deviceId success:^(HETDevice
VersionModel * deviceVersionModel) {
    //         deviceVersionId = 2024;
    //         filePath = "http://200.200.200.58:8981/group1/M00/0D/83/yMjI01j4drWAeBL-
AAB2rNrQug170.bin";
    //         newDeviceVersion = "V1.1.2";
    //         oldDeviceVersion = "1.0.0";
    //         releaseNote = "\U6d4b\U8bd5\U7528\U4f8b\Uff0c\U56fa\U4ef6\U5347\U7ea7";
    //         status = 1;
    if(deviceVersionModel.newDeviceVersion&&![deviceVersionModel.newDeviceVersion isEqualToString:deviceVersionModel.oldDeviceVersion])//有新固件
    {
        // 填写固件升级相关代码
    }else{
        // 填写没有新固件升级提示
    }
} failure:^(NSError *error) {
    NSLog(@"获取硬件版本信息错误:%@",error);
}];
```

3.3 固件升级，下发最新版本给蓝牙设备

【示例代码】

```
[_bleBusiness mcuUpgrade:self.blePeripheral macAddress:self.macAddress deviceVersionModel:
deviceVersionModel progress:^(float progress) {
    //升级进度
    hud.progress=progress;
```

```

} completionHandler:^(CBPeripheral *currentPeripheral,NSError *error) {
    if(error)
    {
        // 填写固件升级失败的处理
    }
    else
    {
        // 填写固件升级成功的处理
    }
}];

```

八、iOS 设备分享

C-Life设备分享分为面对面分享和第三方应用分享,分享相关接口请参考 [HETDeviceShareBusiness](#)

1、分享流程

- 面对面分享：

A用户打开APP设备面对面分享产生一个分享二维码， B用户打开APP的扫一扫，直接获取设备的控制权限。

- 第三方应用分享：

A用户打开APP设备第三方应用分享（微信，QQ），例如分享到微信好友， B用户识别微信中的二维码，打开分享网页，尝试打开APP成功即获取设备的控制权限，失败就提示用户B下载APP。

注意：

1、URL scheme

- 第三方分享需要APP提供URL scheme，方便web页面打开APP，并且传递分享码给APP。

2、分享码有效期

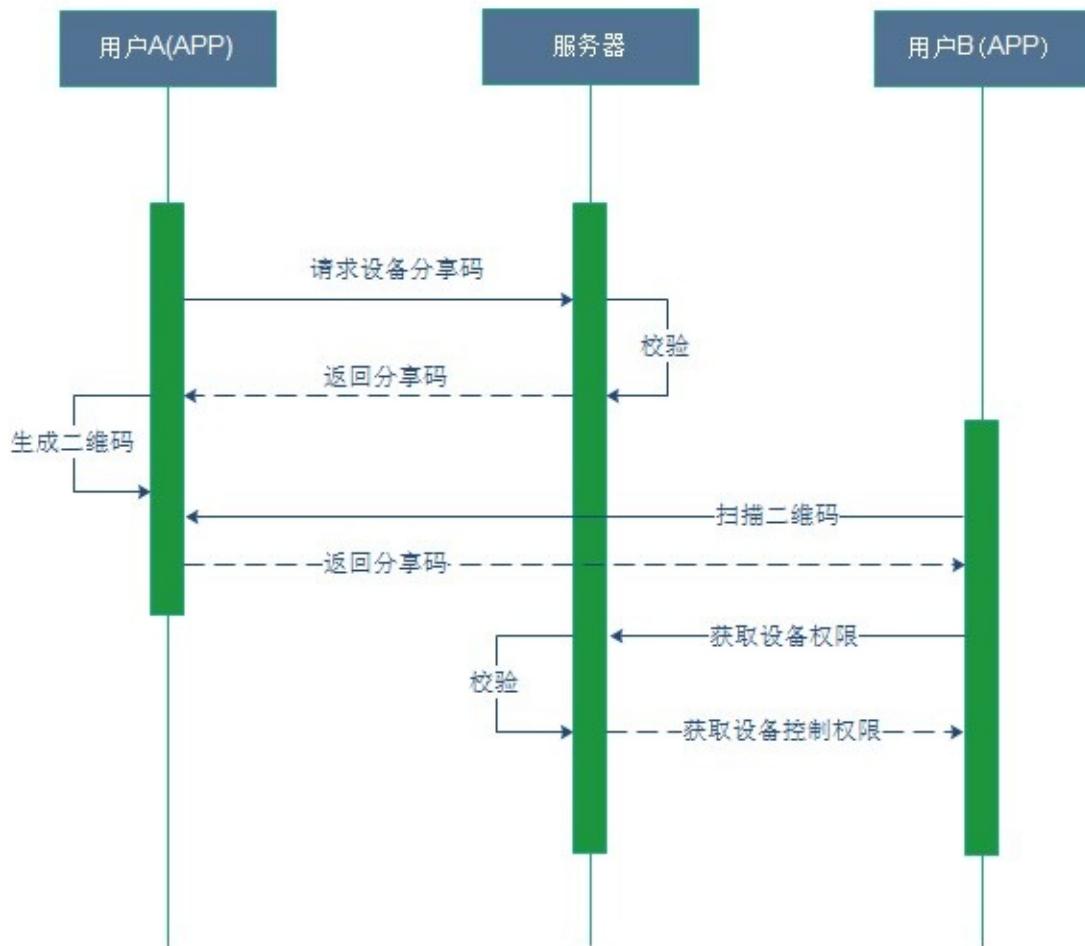
- 面对面分享码有效期为10分钟
- 第三方分享码有效期为一个小时

2、面对面分享

第一步：请求分享码，并生成分享二维码

第二步：验证分享码，获取设备权限

设备分享(应用内)



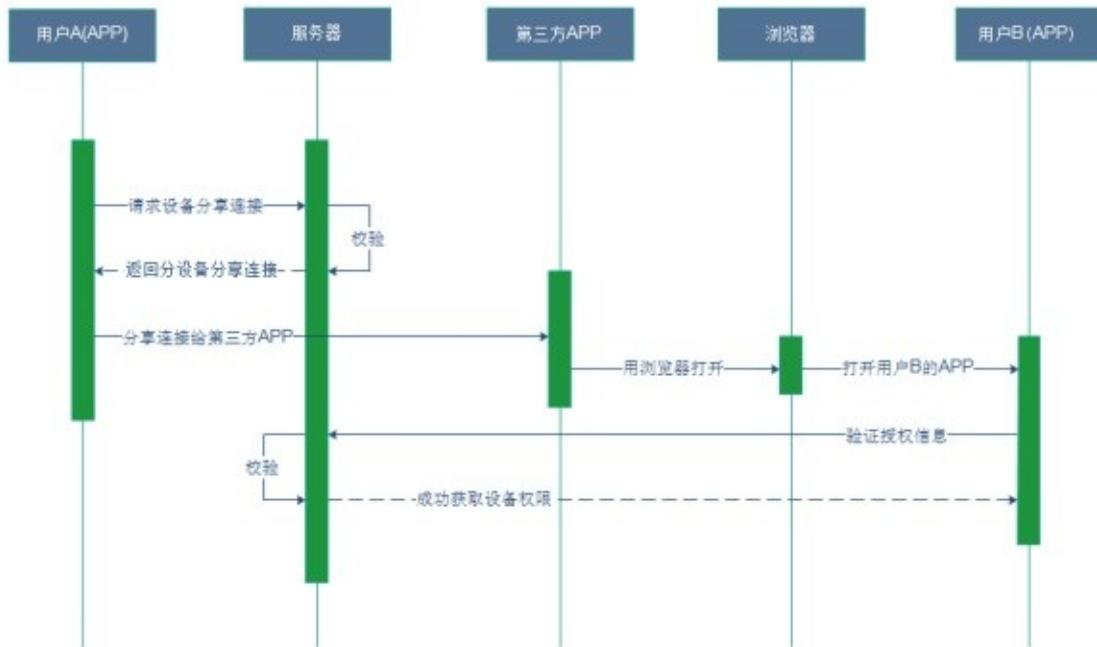
3、第三方应用分享

第一步：请求分享连接，分享到第三方应用

第二步：第三方应用打开连接，web页面尝试打开APP

第三步：验证分享码，获取设备权限

设备分享（第三方应用分享）



4、接口说明

1、分享

1.1、获取设备分享码

【示例代码】

```

weakSelf
[HETDeviceShareBusiness getShareCodeWithDeviceId:self.deviceId shareType:HETDeviceShareTyp
e_ThirthShare success:^(id responseObject) {

    OPLog(@"responseObject == %@",responseObject);
    NSString *h5Url = [responseObject valueForKey:@"h5Url"];

} failure:^(NSError *error) {
    OPLog(@"error == %@",error);
}];
  
```

参数说明

参数名称	是否必须	字段类型	参数说明
deviceId	是	NSString	设备ID
shareType	是	HETDeviceShareType	分享类型

1.2、获取设备权限

1.2.1 本地扫描二维码获取设备权限。

【示例代码】

```
[HETDeviceShareBusiness authShareDeviceWithShareCode:shareCode shareType:HETDeviceShareType_FaceToFaceShare success:^(id responseObject) {
    } failure:^(NSError *error) {
}];
```

1.2.2 微信、微博、QQ分享，通过浏览器打开APP获取设备控制权限。

【示例代码】

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation
{
    // 将URL转成字符串
    NSString * urlString = url.scheme;
    if ([urlString isEqualToString:@"hetopenplatform"]) {
        NSString * shareCode = [[url.host componentsSeparatedByString:@"="] lastObject];
        [HETDeviceShareBusiness authShareDeviceWithShareCode:shareCode shareType:HETDeviceShareType_ThirdShare success:^(id responseObject) {
            OPLog(@"responseObject == %@", responseObject);
            [HETCommonHelp showHudAutoHidenWithMessage:@"获取设备权限成功"];
            [[NSNotificationCenter defaultCenter] postNotificationName:BindDeviceSuccess object:nil];
        } failure:^(NSError *error) {
            OPLog(@"error == %@", error);
            [HETCommonHelp showHudAutoHidenWithMessage:[error.userInfo valueForKey:@"NSLocalizedDescription"]];
        }];
        return YES;
    }else{
        BOOL result = [HETOpenSDK application:application openURL:url sourceApplication:sourceApplication annotation:annotation];
        return result;
    }
}
```

参数说明

参数名称	是否必须	字段类型	参数说明
shareCode	是	NSString	设备分享码
shareType	是	HETDeviceShareType	分享类型

2. 获取设备授权的用户列表

【示例代码】

```
[HETDeviceShareBusiness deviceGetAuthUserWithDeviceId:self.deviceId success:^(id responseObject) {
```

```

bject) {
    OPLog(@"responseObject == %@", responseObject);
} failure:^(NSError *error) {
    OPLog(@"error == %@", error);
}];

```

3.用户设备授权删除

【示例代码】

```

WEAKSELF
[HETDeviceShareBusiness deviceAuthDelWithDeviceId:self.deviceId userId:userId success:^(id
responseObject) {

} failure:^(NSError *error) {

}];

```

九、其他接口

其他接口指业务性接口，如意见反馈、常见问题、隐私政策、版本声明等

使用方法：用户选择对应的requestUrl作参数，调用sdk提供的通用接口

通用接口如下：

```

/**
 * 普通网络请求
 *
 * @param method      HTTP网络请求方法
 * @param requestUrl 网络请求的URL
 * @param params      请求参数
 * @param needSign   是否需要签名
 * @param success     网络请求成功的回调
 * @param failure     网络请求失败的回调
 */
+(void)startRequestWithHTTPMethod:(HETRequestMethod)method
                           withRequestUrl:(NSString *)requestUrl
                           processParams:(NSDictionary *)params
                           needSign:(BOOL)needSign
                           BlockWithSuccess:(successBlock)success
                           failure:(failureBlock)failure

```

requestUrl说明

requestUrl	参数说明
/v1/feedback/addFeedback	意见反馈
暂未开放	常见问题
暂未开放	隐私政策
暂未开放	版本声明

1.意见反馈

【示例代码】

```
WEAKSELF
[HETDeviceRequestBusiness startRequestWithHTTPMethod:HETRequestMethodPost withRequestUrl:@
"/v1/feedback/addFeedback" processParams:params needSign:NO BlockWithSuccess:^(id response
Object) {
    [HETCommonHelp hideHudFromView:weakSelf.view];
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(0.5f * NSEC_PER_SEC)), dispa
tch_get_main_queue(), ^{
        [HETCommonHelp showHudAutoHidenWithMessage:@"提交成功，谢谢您的反馈"];
    });
    dispatch_after(dispatch_time(DISPATCH_TIME_NOW, (int64_t)(1.5f * NSEC_PER_SEC)), dispa
tch_get_main_queue(), ^{
        [weakSelf.navigationController popViewControllerAnimated: YES];
    });
} failure:^(NSError *error) {
    [HETCommonHelp hideHudFromView:weakSelf.view];
    [HETCommonHelp showHudAutoHidenWithMessage:@"提交失败，请检测网络连接"];
}];
```

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时
间： 2017-11-15 10:44:06

- [hetsdk说明文档](#)
 - [目录](#)
 - [项目开发](#)
 - [项目发布](#)

hetsdk说明文档

目录

- [项目开发及发布规范](#)
 - [项目开发](#)
 - [项目发布](#)

项目开发

项目开发目录(推荐)

```
+ page      // app设备html文件
  + index.html
+ src       // 工程源文件，代码编写在此目录进行
  + css
  + js
+ static    // 构建完成的工程文件，请勿直接编辑
  + css
  + img
  + js
+ vm        // 用于模拟app环境的虚拟器
```

项目开发注意细节

1. 使用sdk开发时请登记repaint方法的回调函数，否则将无法获取app推送过来的数据。
2. 向app发送数据时请携带updateFlag参数，并正确计算，否则可能导致下发指令失败。
3. sdk约定app推送过来的数据type=0为控制数据，type=1为运行数据。

项目发布

项目文件发布

1. 项目发布目录结构
 - page // app设备html文件
 - index.html(必须)
 - static // 构建完成的工程文件，请勿直接编辑

- CSS
- img
- js

2. 文件后缀 为便于sdk识别,项目文件需打包放在一个zix压缩包内,如test.zix

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-10-30 13:49:37

- 3.3 APP 开源框架

3.3 APP 开源框架

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 16:09:31

Android开源框架使用指南

Not Only Demo简介

Not Only Demo 以开放平台sdk和主流的APP开发框架为基础搭建一套可以帮助开发者快速开发APP的脚手架.在此基础上开发者可以一天开发一款智能硬件控制APP.

Not Only Demo下载地址

下载地址 https://github.com/C-Life/clife_open_android_sdk 本项目在Android Studio进行开发。

Not Only Demo项目功能模块说明

目录结构

项目包括的主要UI ##### 1.账号授权页面和APP侧边栏框架



为了您的设备使用安全，请先进
行手机号验证

请输入手机号

中国+86>

请输入验证码

获取验证码

提交

已有账号采用 [密码验证](#)

第三方登录

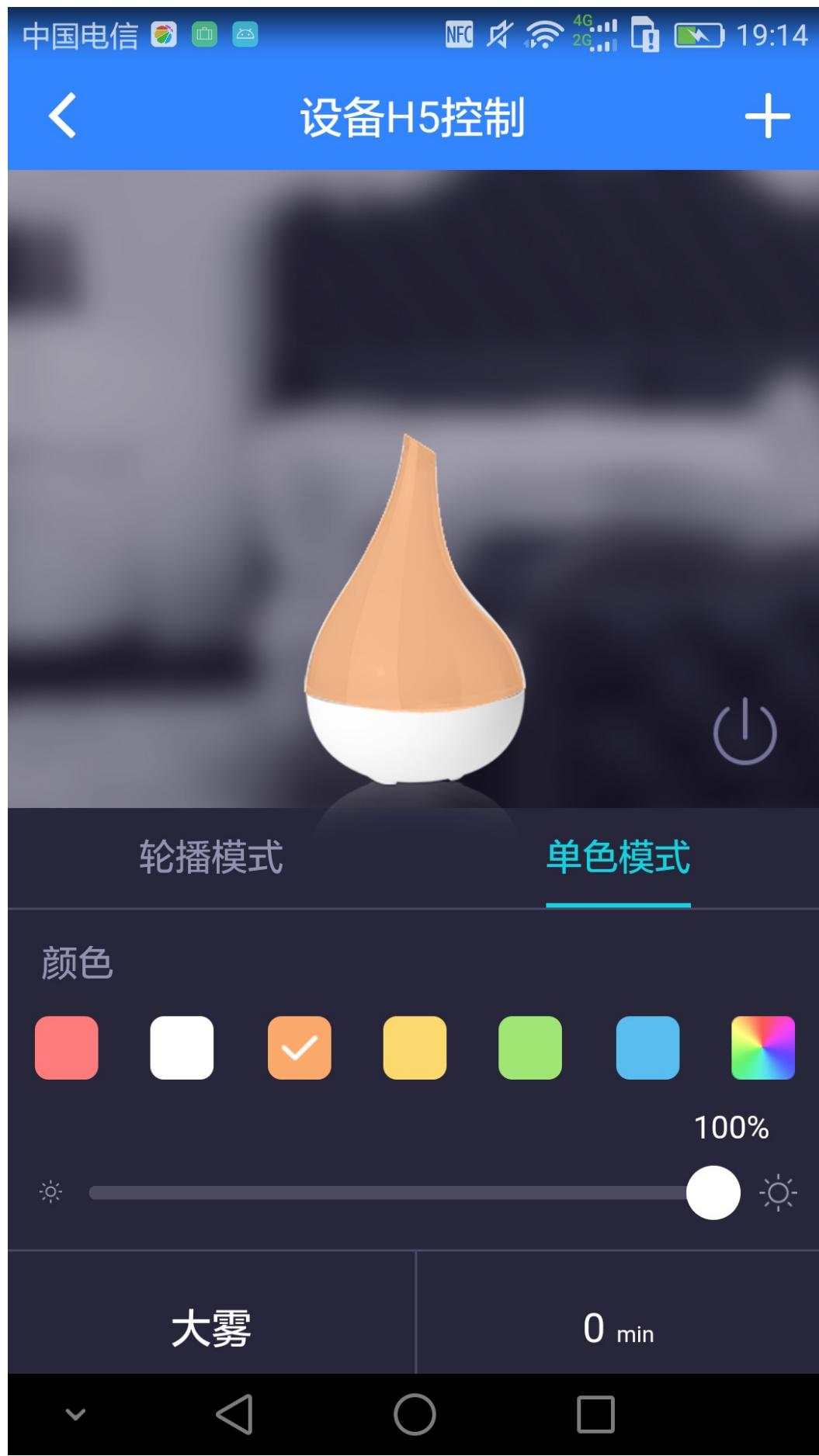


[提交代表同意 C-Life云平台提供的服务 声明](#)



2.我的设备列表页面，H5设备控制页面，原生设备控制页面







3.设备类型选择页面，设备扫描页面，设备绑定页面



设备分类绑定



冰箱



洗衣机



空调



取暖设备



加湿器



惠而浦冰箱

BCD-583WDGBI



惠而浦冰箱

BCD-583WDGBI



惠而浦冰箱

BCD-583WDGBI









常见失败原因

1. 设备是否处于绑定状态
2. WIFI密码是否输入正确
3. 设备型号是否选择正确
4. APP是否适配此WIFI模组

主要包名下面的内容介绍

1.com.het.sdk.demo.base

该包名的类主要对Activity,Fragement和基于MVP架构的P和V进行简单的封装，应用开发可以基于这些类进行扩展。

2.com.het.sdk.demo.adapter

该包名的类主要对列表的ViewHolder和BaseAdapter进行封装，应用开发只要实现对应的item layout和注入数据列表就可以快速列表类页面。

3.com.het.sdk.demo.event

该包名的类主要对应用所用的消息事件和常量进行定义，然后RxBus进行应用。

4.com.het.sdk.demo.manager

该包名的类主要对Activity,Fragement和基于MVP架构的P和V进行简单的封装，应用开发可以基于这些类进行扩展。

5.com.het.sdk.demo.model

该包名主要对网络请求json和页面DTO进行数据定义，然后数据在页面中传输和逻辑使用。

6.com.het.sdk.demo.push

该包名下的类主要对百度和极光推送进行集成而形成的类，用户如果需要就可以直接套用，修改一下推送初始化信息即可完成推送集成。

7.com.het.sdk.demo.ui.activity

该包名下面主要是Activity页面有关的类，具体就不详细介绍，提取几个重要的讲一下。

1.APP侧菜单框架： com.het.sdk.demo.ui.activity.sidebarlayout

基于侧边栏的APP框架，主要包括SidebarMainActivity.java,闪屏页面SplashActivity.java，在结合NavigationDrawerFragment.java来实现侧边导航框架。

2.APP底部Tab框架： com.het.sdk.demo.ui.activity.tablayout

基于底部Tab APP框架，主要包括承载Fragment的TabMainActivity.java，底部选择指示器TabIndicator.java，自定义item视图TabIconView.java和样例SimpleFragment，用户可以基于这几个组件可以非常简单的开发基于底部导航应用。

3.设备绑定： com.het.sdk.demo.ui.activity.bind

此包名下包含了设备大小类型选择DeviceSubTypeListActivity.java，蓝牙绑定页面BleBindActivity.java,二维码扫描页面QrScanActivity.java,WIFI信息输入页面WifiBindActivity.java,设备扫描绑定页面SmartLinkConfigActivity.java，开发者可以修改页面的style或者按照这些页面的逻辑自定义UI就可以快速开发。

5.设备控制: com.het.sdk.demo.ui.activity.device

该包下有蓝牙设备控制页面BleCommonControlActivity.java, Android元素LED控制页面ControlLedActivity.java和H5加载页面H5ControlLedActivity.java,这个几个页面包含了不同通讯方式的设备控制通讯开发样例。

6.消息中心: com.het.sdk.demo.ui.activity.message

该包名下的类主要包含推送消息列表展示, 开发者可以结合开发平台的消息推送服务可以实现运营消息的推送。

7.设备分享: com.het.sdk.demo.ui.activity.share

该包名主要包含设备分享的业务逻辑, 用户可以实现应用内扫描分享, 也可以把设备分享的链接发送到微信, QQ,微博等第三方社交平台推广应用。

9.com.het.sdk.demo.utils

该包名下的类主要常用工具, 方便业务开发共用。

9.com.het.sdk.demo.widget

该包名下的类主要常用控件进行封装, 比如Button, EditText,ToggleButton,Dialog等, 用户可以方便去快速选择自己想要的, 或者继承进行扩展。

集成指南

这个具体浏览Android sdk的集成文档。

常见问题

关于账号

1.手机号码+手机验证码登录生成账号。

2.验证码登录进去以后, 可以通过调用设置密码接口设置密码, 后期可以通过验证码或者账号密码登录。

关于设备绑定不上

1.产品列表找不到对应的产品? 请检查开放平台APP是否关联具体的硬件产品, 可以参考产品接入文档说明。

2.产品绑定不上? 设备绑定目前支持SmartLink绑定和AP绑定, 在进行设备绑定时, 必须让设备处于绑定模式。

3.密码是否正确? 确认输入路由网关的密码。

关于设备控制不了

- 1.检查网络是否异常
- 2.开发调试打开调试信息，查看log日志信息。

最后

上面所有问题都可以直接咨询clife开放平台技术支持，我们讲竭尽全力为你们服务。

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 14:29:18

1.1 Clife Agent简介

1.2 通信模组应用指导

1.3 通信模组调试日记获取教程

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

- 4.1 硬件接入准备工作

4.1 硬件接入准备工作

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 16:09:31

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-10 17:47:51

- 和而泰云端通讯MCU程序詳解
 - 通讯协议詳解
 - 1、通讯数据格式
 - 2、数据包格式
 - 一、此文档包含 3 个文档，分别为
 - 二、函数 API 说明
 - 2.1 void Het_DriveWifi_WifilInit(pfUartSend_pf_uart_send,pfUartDecode_pf_uart_decode,pfWifiReset_pf_wifi_reset)
 - 2.2 void Het_DriveWifi_SystickISR (void)
 - 2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t_het_data)
 - 2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t_flag)
 - 2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t_flag)
 - 2.6 het_uint8_t Het_DriveWifi_GetWifiStatus (void)
 - 2.7 het_bool Het_DriveWifi_GetAppSynStatus (void)
 - 2.8 enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType_type,het_uint8_t*_pbuf,het_uint8_t_len)
 - 2.9 enum_WResult Het_DriveWifi_WifiProcess (void)
 - 三、操作流程
 - 四、实例代码
- 和而泰云端通讯MCU程序詳解
 - 通讯协议詳解
 - 1、通讯数据格式
 - 2、数据包格式
 - 一、此文档包含 3 个文档，分别为
 - 二、函数 API 说明
 - 2.1 void Het_DriveWifi_WifilInit(pfUartSend_pf_uart_send,pfUartDecode_pf_uart_decode,pfWifiReset_pf_wifi_reset)
 - 2.2 void Het_DriveWifi_SystickISR (void)
 - 2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t_het_data)
 - 2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t_flag)
 - 2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t_flag)
 - 2.6 het_uint8_t Het_DriveWifi_GetWifiStatus (void)
 - 2.7 het_bool Het_DriveWifi_GetAppSynStatus (void)
 - 2.8 enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType_type,het_uint8_t*_pbuf,het_uint8_t_len)
 - 2.9 enum_WResult Het_DriveWifi_WifiProcess (void)
 - 三、操作流程
 - 四、实例代码

<<<<< HEAD

和而泰云端通讯MCU程序詳解

通讯协议详解

1、通讯数据格式

通讯方式 UART	异步通讯
起始位	1bit
数据位	8bit
校验位	无
停止位	1bit
波特率	9600bps--115200bps (初始为 9600) 可以设置

2、数据包格式

起始标志	数据长度	协议版本	wifi状态	数据帧序	保留	注【1】 数据类型	数据内容	校验码
0x5A	14+N	0x1X 注 2	见注 3	设备主动发出: 0x00000000 -0xffffffff f, 服务器: 0x10000000 APP:0x2000 0000 滚动 增加	0x000 0	Data	N < 513	帧头 12 字节 +数据部分 【N】+校验 2 字节. 注【4】 (CRC16-0x84 08)
1 字节	2 字节	1 字节	1 字节	4 字节	2 字节	2 字节	N	2 字节

说明

- 1> 起始标志：固定为5A;
- 2> 数据长度：从数据长度到整包数据结束所占用的字节数;
- 3> 协议版本：根据用户需要接入的平台选择版本信息，具体如下：

版本信息	BIT3	BIT2	BIT1	BIT0
	00 - CLife 绑定		绑定方式	数据格式
	01 - 微信绑定		1-AP	1-JSON
	02 - 京东绑定		0-SMARTLINK	0-字节流

4> wifi状态：wifi模块的连接信息，发送是填0，接收时具体解析如下：

wifi 状态	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
WIFI 升级	服务器	路由器	绑定状态	WIFI 信号强度				
1-升级中	1-已连接	1-已连接	1-绑定成功	0 - 10 对应 0%--100%				
0-正常	0-未连接	0-未连接	0-未绑定					

WIFI 信号强度			WIFI 信号强度 5 格显示	WIFI 信号强度 4 格显示(推荐)	WIFI 信号强度 3 格显示	WIFI 信号强度 1 格显示
0	0%	≥-95 dBm	不显示	不显示	不显示	不显示
1	10%	≥-88 dBm	不显示	不显示	不显示	不显示
2	20%	≥-81 dBm	显示 1 格	显示 1 格	显示 1 格	显示 1 格
3	30%	≥-74 dBm	显示 2 格	显示 2 格	显示 2 格	显示 1 格
4	40%	≥-67 dBm	显示 3 格	显示 3 格	显示 2 格	显示 1 格
5	50%	≥-60 dBm	显示 4 格	显示 4 格	显示 3 格	显示 1 格
6	60%	≥-53 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
7	70%	≥-46 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
8	80%	≥-39 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
9	90%	≥-32 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
10	100%	≥-25 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格

5> 数据帧序：设备主发0x00000000-0xffffffff;服务器主发0X10000000-0x1fffffff;APP主发0X20000000-0x2fffffff;滚动增加,应答时原样返回; 6> 保留：固定0x0000; 7> 数据类型：

数据类型	数据说明
0x0108	心跳包
0x0208	心跳包应答
0x0150	绑定命令
0x0250	绑定命令应答
0x0104	控制数据
0x0204	控制数据应答
0x0105	运行数据
0x0205	运行数据应答
0x0107	配置数据
0x0207	配置数据应答
0x0406	时间同步请求
0x0206	时间同步请求应答

8> 数据内容：用户自定义数据区，数据长度必须是16倍数，长度小于512字节；9> 校验码：校验码的计算方式采用(CRC16-0x8408)的方式，具体代码如下：

```

79  /*************************************************************************/
80  * 名称: het_crc_check
81  * 功能: 计算指定数据区域的CRC校验值
82  * 输入: 缓存, 长度, 长度, 校验值
83  * 输出: 校验值
84  * 说明:
85  ****
86  static void DriveWifi_CrcCheck(het_uint8_t *_pbuf,het_uint16_t _len,het_uint16_t *_pcrc)
87  {
88      het_uint16_t i,j;
89      het_uint16_t crc = 0xFFFF;
90      for(i=0; i<_len; i++)
91      {
92          crc ^= *_pbuf++;
93          for(j = 0;j < 8;j++)
94          {
95              if(crc & 0x01)
96              {
97                  crc = (crc >> 1) ^ 0x8408;
98              }
99              else
100             {
101                 crc >>= 0x01;
102             }
103         }
104     *_pcrc = ~crc;
105 }
106
107

```

10>

一、此文档包含 3 个文档，分别为

1. DriveWifi.c – wifi绑定和通信模块驱动，不可更改文件。 2. DriveWifi.h – wifi驱动头文件,不可更改文件。 3. WiFiConfig.h – wifi模块配置信息，根据不同的设备而修改。

二、函数 API 说明

2.1 void Het_DriveWifi_Wifilnit(pfUartSend _pf_uart_send,pfUartDecode _pf_uart_decode,pfWifiReset _pf_wifi_reset)

<1> Description: wifi 模组初始函数，用于注册用户函数。

<2> Arguments: `_pf_uart_send` – 用户串口发送字符串函数，函数名可自定义，函数原型必须为：
`void fun(het_uint8_t pbuf,het_uint16_t len)`。
`_pf_uart_decode` – 用户命令解码函数，函数名可自定义，函数原型必须为：
`void fun(het_uint16_t cmd,het_uint8_t pbuf,het_uint16_t len)`。
`_pf_wifi_reset` – 用户wifi 模组复位函数,函数名可自定义，函数原型必须为：
`void fun(het_uint8_t flag)`。

<3> Return:NONE

2.2 void Het_DriveWifi_SystickISR (void)

<1> Description: 此函数作用是给程序提供 10ms 时钟,必须放在 10ms 定时器里

<2> Arguments: NONE

<3> Return: NONE

2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t _het_data)

<1> Description:WIFI 模组串口中断接收函数,必须放在串口中断函数里面

<2> Arguments: data – 串口接收到的字节

<3> Return: NONE

2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t _flag)

<1> Description: WIFI绑定触发条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能绑定操作

<3> Return: NONE

2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t _flag)

<1> Description: WIFI 进入产测条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能产测操作

<3> Return: NONE

2.6 het_uint8_t Het_DriveWifi_GetWifiStatus (void)

<1> Description: 获取 WiFi 连接状态函数

<2> Arguments: NONE

<3> Return: WiFi 状态值

2.7 het_bool Het_DriveWifi_GetAppSynStatus (void)

<1> Description: 获取 APP 同步状态函数

<2> Arguments: NONE

<3> Return: APP 同步值

2.8 enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType _type,het_uint8_t *_pbuf,het_uint8_t _len)

<1> Description: 发送用户私有数据函数,

<2> Arguments: type – 用户发送的数据所处类型, 如CMD_TYPE_CTRL 表示为控制命令数据, CMD_TYPE_STATUS 表示为状态命令数据 Pbuf – 用户发送数据缓存的首地址 Len – 用户发送数据的长度

<3> Return:发送的状态, WR_OK 表示发送成功

2.9 enum_WResult Het_DriveWifi_WifiProcess (void)

<1> Description:wifi 绑定和数据交互处理,此函数放在主循环里面

<2> Arguments:NONE

<3> Return:返回当前程序的状态。特别注意的是, 当返回状态等于 WR_WAIT_SEND_CTRL_CMD 或者 WR_TIMER_SEND_STATUS_CMD 时, 程序正处于 塞状态,等待用户输入设备控制和状态数据。

三、操作流程

1、修改 WifiConfig.h 文件里面的如下配置信息：

<1>DEVICE_TYPE- 设备类型 (该信息在开放平台上创建产品后获得)

<2>DEVICE_KEY - 设备秘钥 (该信息在开放平台上创建产品后获得)

<3>MCU设备基本信息, 如SOFT_CONTROL_BOARD_VERSION等 (一般情况不用改) 此三项开发前申请。2、把 Het_DriveWifi_SystickISR 函数放在 10ms 定时器里。3、把 Het_DriveWifi_UsartRecvISR 函数放在 wifi 串口中断接收服务函数里。4、初始化 wifi 模块, 调用 DriveWifi.c 文件里面 Het_DriveWifi_WifiInit 函数。5、在主循环调用 Het_DriveWifi_WifiProcess 函数, Het_DriveWifi_WifiProcess 函数会返回 wifi 模组当前状态, 当返回的当前状态等于 WR_WAIT_SEND_CTRL_CMD 时, wifi 模组正等待设备调用Het_DriveWifi_WifiDataSend 函数发送当前控制参数; 当返回的当前状态等于WR_TIMER_SEND_STATUS_CMD 时, wifi 模组正在等待设备调用 Het_DriveWifi_WifiDataSend 函数发送当前状态参数; 但是返回的当前状态等于 WR_TIMER_SEND_ERROR_CMD 时, wifi模组正在等待设备调用Het_DriveWifi_WifiDataSend函数发送当前故障信息。当用户需要主动上传数据时, 也可以调用 Het_DriveWifi_WifiDataSend 函数发送数据。

四、实例代码

```
//-----
//操作头文件WifiConfig.h对wifi通讯进行设备
#define MOUDLE_TYPE           MODULE_WIFI          /*choose the type of module*/
#define BIND_TYPE              SMARTLINK_MODE     /*choose the type of bind*/
#define BIND_PLATFORM          WECHAT             /*connect the platform*/
#define HET_FACTORY_TEST_EN    1                  /*1:enable  0:disable*/
#define HET_UPGRADE_FW_EN      1                  /*1:enable  0:disable*/
#define HET_FILE_UPLOAD_EN     1                  /*1:enable  0:disable*/
#if BIND_PLATFORM == WECHAT
#define DEVICE_TYPE_WECHAT    "gh_d04b5c778911" /*微信绑定相关信息*/

```

```

#define DEVICE_ID_WECHAT           "clife_7879_"
#define DEVICE_TYPE_LEN_WECHAT     15
#define DEVICE_ID_LEN_WECHAT       11
#endif
#define DEVICE_TYPE 0x00,0x00,0xC5,0x85,0x00,0x5A,0x01,0x01 //设备编码
#define DEVICE_KEY "5aa5fba81a864beb859f0fd8234d32dc"      //设备devicekey
#define DEVICE_INFO_LEN             48                         //设备信息
#define SOFT_CONTROL_BOARD_VERSION  0x01                      /*ctrl board software version*/
#define SOFT_DISPLAY_BOARD_VERSION   0x01                      /*display board software version*/
#define SOFT_DRIVE_BOARD_VERSION    0x01                      /*drive board software version*/

#define SOFT_OTHER_VERSION          0x01                      /*other board software version*/

#define HARD_CONTROL_BOARD_VERSION  0x01                      /*ctrl board hardware version*/
#define HARD_DISPLAY_BOARD_VERSION   0x01                      /*ctrl board hardware version*/
#define HARD_DRIVE_BOARD_VERSION    0x01                      /*ctrl board hardware version*/
#define HARD_OTHER_VERSION          0x01                      /*ctrl board hardware version*/

#define DEVICE_MAIN_TYPE_H          0X00                      /*device main type high byte*/
#define DEVICE_MAIN_TYPE_L          0x4B                      /*device main type low byte*/
#define DEVICE_SUB_TYPE             0x03                      /*device sub type*/
#define DEVICE_FUN_TYPE              0x02                      /*device function type */

#define DEVICE_PROJECT_NUMBER        "Z029-A16033A"      /*project number*/
#define WIFI_CONTROL_LEN            48                        /*控制数据长度*/
#define WIFI_STATUS_LEN             48                        /*运行数据长度*/
#define WIFI_ERROR_LEN              16                        /*故障数据长度*/
//-----
-----
```

```

//main.c //用户实现3个函数 /** 名称: WifiPro_UartRxPro 功能: 处理WIFI下发给MCU的指令 输入:
命令, 缓存指针, 缓存长度 输出: 说明: */ void WifiPro_UartRxPro(unsigned short _cmd,unsigned
char *_buf,unsigned short _data_len) {
if(_cmd== CMD_0104_CTRL) //处理服务器下发的控制命令 { //用户代码, 用于解析下发的控制命令
WifiPro_ControlHandle(_buf,_data_len); //MCU端响应控制指令
WifiPro_SendMachineControlCommand();
Het_DriveWifi_WifiDataSend(CMD_0204_CTRL_ACK,UartTxBuf,16); } else if(_cmd==
CMD_0107_CONFIG) //接收服务器下发的硬件配置命令 { //用户代码, 用于解析下发的配置命令
```

```

//MCU端响应配置指令
Het_DriveWifi_WifiDataSend(CMD_0207_CONFIG_ACK,UartTxBuf,0);
}
else if(_cmd == CMD_0206_GET_TIME_ACK)                                //接受服务器下发的时间信息
{
;
}
else if(_cmd == CMD_0120_UPGRADE_START)                                //收到升级起始帧, 开启升
级程序
{
;
}
```

```

else if(_cmd == CMD_0122_UPGRADE_RECV_PACKET)           //接收到升级的数据
{
;
}
else if(_cmd == CMD_0225_UPGRADE_END_ACK)               //升级成功
{
;
}

}

/** 名称: WifiPro_WifiReset 功能: WIFI复位状态设定 输入: 复位状态 输出: 说明: */ void
WifiPro_WifiReset(het_uint8_t _status) { if(_status) { WIFI_RESET_LOW; } else {
WIFI_RESET_HIGH; } } /** 名称: WifiPro_ProtocolPro 功能: WIFI模块与MCU之间的通信处理 输入: 输出: 说明: 每10ms处理一次 */ void WifiPro_ProtocolPro(void) { enum_WResult run_wifi;

//该部分处理WIFI模块下发的数据
run_wifi = Het_DriveWifi_WifiProcess();
switch(run_wifi)
{
    case WR_WAIT_SEND_CTRL_CMD:                                //该部分是第一次连接服务器
时设备端主动向服务器上传一次的控制状态信息
        WifiPro_SendMachineControlCommand();                      //MCU端响应WIFI模块的控制指令
        break;

    case WR_TIMER_SEND_STATUS_CMD:                             //每5S发送一次状态信息
        WifiPro_SendMachineStateCommand();
        Het_DriveWifi_WifiDataSend(CMD_0105_STATUS,UartTxBuf,WIFI_STATE_LEN);
        break;

    case WR_TIMER_SEND_ERROR_CMD:                            //发送错误信息
        WifiPro_SendMachineErrorCommand();
        Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
        break;

    case WR_WAIT_SEND_CTRL_REQUEST_ACK_CMD:                  //WIFI模块请求发送控制数据
        WifiPro_SendMachineControlCommand();
        Het_DriveWifi_WifiDataSend(CMD_0304_STATUS_REQUEST_ACK,UartTxBuf,WIFI_CONTROL_LEN)
        ;
        break;

    case WR_WAIT_SEND_STATUS_REQUEST_ACK_CMD:                //WIFI模块请求发送状态信息
        WifiPro_SendMachineStateCommand();
        Het_DriveWifi_WifiDataSend(CMD_0305_STATUS_REQUEST_ACK,UartTxBuf,WIFI_STATE_LEN);
        break;

    case WR_WIFI_TEST_OK:                                    //WIFI模块告诉MCU测试OK
        break;
    case WR_WIFI_TEST_ERR:                                  //WIFI模块告诉MCU测试出错
        break;

    default:
        break;
}
}

```

```

//该部分处理MCU主动上传的数据
if(UpgradeAp.UpControlStatusF == UPGRADE_FALSE) //保证不在升级状态
{
    //这个标志位很重要，主动上传控制命令是为了保证控制状态的实时性，服务器可以立刻获取到当前设备的
    //控制状态
    if(WifiPro.LocalControlF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
    {
        WifiPro.LocalControlF = WIFIPRO_FALSE; //单片机主动发送控制状态信息
        WifiPro_SendMachineControlCommand();
        Het_DriveWifi_WifiDataSend(CMD_0104_CTRL,UartTxBuf,WIFI_CONTROL_LEN);
    }
    if(WifiPro.LocalErrorF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
    {
        WifiPro.LocalErrorF = WIFIPRO_FALSE; //单片机主动发送错误信息
        WifiPro_SendMachineErrorCommand();
        Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
    }
}
}

```

{ /*

- Function Name: main
- Description : 主函数
- Arguments :
- Return Value: void **/

```

int main(void) { Het_DriveWifi_WifiInit(WifiPro_WifiUartSend,WifiPro_UartRxPro,WifiPro_WifiReset);
//wifi模块化程序初始化
while (1) { ///////////////////////////////// //WIFI程序处理区
WifiPro_ProtocolPro(); //WIFI信息处理 ///////////////////////////////// //应用程序
处理区 UserApplicationHandle(); } }

=====

```

和而泰云端通讯MCU程序详解

通讯协议详解

1、通讯数据格式

通讯方式 UART	异步通讯
起始位	1bit
数据位	8bit
校验位	无
停止位	1bit
波特率	9600bps--115200bps (初始为 9600) 可以设置

2、数据包格式

起始标志	数据长度	协议版本	wifi状态	数据帧序	保留	注【1】数据类型	数据内容	校验码
0x5A	14+N	0x1X 注 2	见注 3	设备主动发出: 0x00000000 -0x0fffffff f, 服务器: 0x10000000 APP:0x2000 0000 滚动 增加	0x0000 0	Data	N < 513	帧头 12 字节 +数据部分 【N】+校验 2 字节. 注【4】 (CRC16-0x84 08)
1 字节	2 字节	1 字节	1 字节	4 字节	2 字节	2 字节	N	2 字节

说明

1> 起始标志：固定为5A; 2> 数据长度：从数据长度到整包数据结束所占用的字节数; 3> 协议版本：根据用户需要接入的平台选择版本信息，具体如下：

版本信息	BIT3	BIT2	BIT1	BIT0
	00 - CLife 绑定	绑定方式	数据格式	
	01 - 微信绑定	1-AP	1-JSON	
	02 - 京东绑定	0-SMARTLINK	0-字节流	

4> wifi状态：wifi模块的连接信息，发送是填0，接收时具体解析如下：

wifi 状态	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
	WIFI 升级	服务器	路由器	绑定状态	WIFI 信号强度			
1-升级中	1-已连接	1-已连接	1-已连接	1-绑定成功	0 - 10 对应 0%--100%			
0-正常	0-未连接	0-未连接	0-未连接	0-未绑定				

WIFI 信号强度			WIFI 信号强度 5 格显示	WIFI 信号强度 4 格显示(推荐)	WIFI 信号强度 3 格显示	WIFI 信号强度 1 格显示
0	0%	≥-95 dBm	不显示	不显示	不显示	不显示
1	10%	≥-88 dBm	不显示	不显示	不显示	不显示
2	20%	≥-81 dBm	显示 1 格	显示 1 格	显示 1 格	显示 1 格
3	30%	≥-74 dBm	显示 2 格	显示 2 格	显示 2 格	显示 1 格
4	40%	≥-67 dBm	显示 3 格	显示 3 格	显示 2 格	显示 1 格
5	50%	≥-60 dBm	显示 4 格	显示 4 格	显示 3 格	显示 1 格
6	60%	≥-53 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
7	70%	≥-46 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
8	80%	≥-39 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
9	90%	≥-32 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格
10	100%	≥-25 dBm	显示 5 格	显示 4 格	显示 3 格	显示 1 格

5> 数据帧序：设备主发0x00000000-0x0fffffff;服务器主发0X10000000-0x1fffffff;APP主发0X20000000-0x2fffffff;滚动增加,应答时原样返回; 6> 保留：固定0x0000; 7> 数据类型：

数据类型	数据说明
0x0108	心跳包
0x0208	心跳包应答
0x0150	绑定命令
0x0250	绑定命令应答
0x0104	控制数据
0x0204	控制数据应答
0x0105	运行数据
0x0205	运行数据应答
0x0107	配置数据
0x0207	配置数据应答
0x0406	时间同步请求
0x0206	时间同步请求应答
0x015A	厂测命令
0x025A	厂测命令应答

8> 数据内容：用户自定义数据区，数据长度必须是16倍数，长度小于512字节; 9> 校验码：校验码的计算方式采用(CRC16-0x8408)的方式，具体代码如下：

```

79 //*****
80 //名称: het_crc_check
81 //功能: 计算指定数据区域的CRC校验值
82 //输入: 缓存, 长度, 长度, 校验值
83 //输出: 校验值
84 //说明:
85 *****
86 static void DriveWifi_CrcCheck(het_uint8_t *pbuf,het_uint16_t _len,het_uint16_t *_pcrc)
87 {
88     het_uint16_t i,j;
89     het_uint16_t crc = 0xFFFF;
90     for(i=0; i<_len; i++)
91     {
92         crc ^= *pbuf++;
93         for(j = 0;j < 8;j++)
94         {
95             if(crc & 0x01)
96             {
97                 crc = (crc >> 1) ^ 0x8408;
98             }
99             else
100            {
101                 crc >>= 0x01;
102             }
103         }
104     }
105     *_pcrc = ~crc;
106 }
```

10> 在联网的情况下发送的命令除0x0105外都会立即应答，如在200 毫秒内无应答，则需重发，最多5次，在没有联网的情况下只有心跳命令、绑定命令、厂测命令有应答。

一、此文档包含 3 个文档，分别为

1.DriveWifi.c – wifi绑定和通信模块驱动，不可更改文件。 2.DriveWifi.h – wifi驱动头文件,不可更改文件。 3.WifiConfig.h – wifi模块配置信息，根据不同的设备而修改。

二、函数 API 说明

2.1 void Het_DriveWifi_Wifilnit(pfUartSend _pf_uart_send,pfUartDecode _pf_uart_decode,pfWifiReset _pf_wifi_reset)

<1> Description: wifi 模组初始函数，用于注册用户函数。

<2> Arguments: `_pf_uart_send` – 用户串口发送字符串函数，函数名可自定义，函数原型必须为：
`void fun(het_uint8_t pbuf,het_uint16_t len)`。
`_pf_uart_decode` – 用户命令解码函数，函数名可自定义，函数原型必须为：
`void fun(het_uint16_t cmd,het_uint8_t pbuf,het_uint16_t len)`。
`_pf_wifi_reset` – 用户wifi 模组复位函数,函数名可自定义，函数原型必须为：
`void fun(het_uint8_t flag)`。

<3> Return: NONE

2.2 void Het_DriveWifi_SystickISR (void)

<1> Description: 此函数作用是给程序提供 10ms 时钟,必须放在 10ms 定时器里

<2> Arguments: NONE

<3> Return: NONE

2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t _het_data)

<1> Description: WIFI 模组串口中断接收函数,必须放在串口中断函数里面

<2> Arguments: data – 串口接收到的字节

<3> Return: NONE

2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t _flag)

<1> Description: WIFI绑定触发条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能绑定操作

<3> Return: NONE

2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t _flag)

<1> Description: WIFI 进入产测条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能产测操作

<3> Return: NONE

2.6 **het_uint8_t Het_DriveWifi_GetWifiStatus (void)**

<1> Description: 获取 WiFi 连接状态函数

<2> Arguments: NONE

<3> Return: WiFi 状态值

2.7 **het_bool Het_DriveWifi_GetAppSynStatus (void)**

<1> Description: 获取 APP 同步状态函数

<2> Arguments: NONE

<3> Return: APP 同步值

2.8 **enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType _type,het_uint8_t *_pbuf,het_uint8_t _len)**

<1> Description: 发送用户私有数据函数,

<2> Arguments: type – 用户发送的数据所处类型, 如CMD_TYPE_CTRL 表示为控制命令数据, CMD_TYPE_STATUS 表示为状态命令数据 Pbuf – 用户发送数据缓存的首地址 Len – 用户发送数据的长度

<3> Return: 发送的状态, WR_OK 表示发送成功

2.9 **enum_WResult Het_DriveWifi_WifiProcess (void)**

<1> Description: wifi 绑定和数据交互处理, 此函数放在主循环里面

<2> Arguments: NONE

<3> Return: 返回当前程序的状态。特别注意的是, 当返回状态等于 WR_WAIT_SEND_CTRL_CMD 或者 WR_TIMER_SEND_STATUS_CMD 时, 程序正处于塞状态, 等待用户输入设备控制和状态数据。

三、操作流程

1、修改 WifiConfig.h 文件里面的如下配置信息：

<1> DEVICE_TYPE - 设备类型 (该信息在开放平台上创建产品后获得)

<2> DEVICE_KEY - 设备秘钥 (该信息在开放平台上创建产品后获得)

<3> MCU设备基本信息, 如SOFT_CONTROL_BOARD_VERSION等 (一般情况不用改) 此三项开发前申请。2、把 Het_DriveWifi_SystickISR 函数放在 10ms 定时器里。3、把 Het_DriveWifi_UsartRecvISR 函数放在 wifi 串口中断接收服务函数里。4、初始化 wifi 模块, 调用 DriveWifi.c 文件里面 Het_DriveWifi_WifiInit 函数。5、在主循环调用 Het_DriveWifi_WifiProcess 函数。

数，`Het_DriveWifi_WifiProcess` 函数会返回 wifi 模组当前状态，当返回的当前状态等于 `WR_WAIT_SEND_CTRL_CMD` 时， wifi 模组正等待设备调用 `Het_DriveWifi_WifiDataSend` 函数发送当前控制参数；当返回的当前状态等于 `WR_TIMER_SEND_STATUS_CMD` 时， wifi 模组正在等待设备调用 `Het_DriveWifi_WifiDataSend` 函数发送当前状态参数；但是返回的当前状态等于 `WR_TIMER_SEND_ERROR_CMD` 时， wifi 模组正在等待设备调用 `Het_DriveWifi_WifiDataSend` 函数发送当前故障信息。当用户需要主动上传数据时，也可以调用 `Het_DriveWifi_WifiDataSend` 函数发送数据。

四、实例代码

```

//-----
-----操作头文件WiFiConfig.h对wifi通讯进行设备
#define MOUDLE_TYPE           MODULE_WIFI          /*choose the type of module*/
#define BIND_TYPE              SMARTLINK_MODE    /*choose the type of bind*/
#define BIND_PLATFORM           WECHAT            /*connect the platform*/
#define HET_FACTORY_TEST_EN     1                  /*1:enable  0:disable*/
#define HET_UPGRADE_FW_EN       1                  /*1:enable  0:disable*/
#define HET_FILE_UPLOAD_EN      1                  /*1:enable  0:disable*/
#if BIND_PLATFORM == WECHAT
#define DEVICE_TYPE_WECHAT     "gh_d04b5c778911"
#define DEVICE_ID_WECHAT        "clife_7879_"
#define DEVICE_TYPE_LEN_WECHAT 15
#define DEVICE_ID_LEN_WECHAT   11
#endif
#define DEVICE_TYPE 0x00,0x00,0xC5,0x85,0x00,0x5A,0x01,0x01 //设备编码
#define DEVICE_KEY "5aa5fba81a864beb859f0fd8234d32dc" //设备devicekey
#define DEVICE_INFO_LEN         48                //设备信息
#define SOFT_CONTROL_BOARD_VERSION 0x01          /*ctrl board software version*/
#define SOFT_DISPLAY_BOARD_VERSION 0x01          /*display board software version*/
#define SOFT_DRIVE_BOARD_VERSION 0x01            /*drive board software version*/

#define SOFT_OTHER_VERSION       0x01            /*other board software version*/

#define HARD_CONTROL_BOARD_VERSION 0x01          /*ctrl board hardware version*/
#define HARD_DISPLAY_BOARD_VERSION 0x01          /*ctrl board hardware version*/
#define HARD_DRIVE_BOARD_VERSION 0x01            /*ctrl board hardware version*/
#define HARD_OTHER_VERSION       0x01            /*ctrl board hardware version*/

#define DEVICE_MAIN_TYPE_H       0X00            /*device main type high byte*/
#define DEVICE_MAIN_TYPE_L       0x4B            /*device main type low byte*/
#define DEVICE_SUB_TYPE          0x03            /*device sub type*/
#define DEVICE_FUN_TYPE          0x02            /*device function type */

#define DEVICE_PROJECT_NUMBER    "Z029-A16033A" /*project number*/
#define WIFI_CONTROL_LEN         48               /*控制数据长度*/
#define WIFI_STATUS_LEN          48               /*运行数据长度*/
#define WIFI_ERROR_LEN           16               /*故障数据长度*/
//-----
-----
```

```

//main.c //用户实现3个函数 /* 名称: WifiPro_UartRxPro 功能: 处理WIFI下发给MCU的指令 输入:
命令, 缓存指针, 缓存长度 输出: 说明: */ void WifiPro_UartRxPro(unsigned short _cmd,unsigned
char *_buf,unsigned short _data_len) {
if(_cmd== CMD_0104_CTRL) //处理服务器下发的控制命令 { //用户代码, 用于解析下发的控制命令
WifiPro_ControlHandle(_buf,_data_len); //MCU端响应控制指令
WifiPro_SendMachineControlCommand();
Het_DriveWifi_WifiDataSend(CMD_0204_CTRL_ACK,UartTxBuf,16); } else if(_cmd==
CMD_0107_CONFIG) //接收服务器下发的硬件配置命令 { //用户代码, 用于解析下发的配置命令

    //MCU端响应配置指令
    Het_DriveWifi_WifiDataSend(CMD_0207_CONFIG_ACK,UartTxBuf,0);
}
else if(_cmd == CMD_0206_GET_TIME_ACK)                                //接受服务器下发的时间信息
{
;
}
else if(_cmd == CMD_0120_UPGRADE_START)                               //收到升级起始帧, 开启升
级程序
{
;
}
else if(_cmd == CMD_0122_UPGRADE_RECV_PACKET)                         //接收到升级的数据
{
;
}
else if(_cmd == CMD_0225_UPGRADE_END_ACK)                            //升级成功
{
;
}

}

/** 名称: WifiPro_WifiReset 功能: WIFI复位状态设定 输入: 复位状态 输出: 说明: */ void
WifiPro_WifiReset(het_uint8_t _status) { if(_status) { WIFI_RESET_LOW; } else {
WIFI_RESET_HIGH; } } /** 名称: WifiPro_ProtocolPro 功能: WIFI模块与MCU之间的通信处理 输入:
输出: 说明: 每10ms处理一次 */ void WifiPro_ProtocolPro(void) { enum_WResult run_wifi;

    //该部分处理WIFI模块下发的数据
    run_wifi = Het_DriveWifi_WifiProcess();
    switch(run_wifi)
    {
        case WR_WAIT_SEND_CTRL_CMD:                                         //该部分是第一次连接服务器
时设备端主动向服务器上传一次的控制状态信息
            WifiPro_SendMachineControlCommand();                           //MCU端响应WIFI模块的控制指
令
            Het_DriveWifi_WifiDataSend(CMD_0104_CTRL,UartTxBuf,WIFI_CONTROL_LEN);
            break;

        case WR_TIMER_SEND_STATUS_CMD:                                     //每5S发送一次状态信息
            WifiPro_SendMachineStateCommand();
            Het_DriveWifi_WifiDataSend(CMD_0105_STATUS,UartTxBuf,WIFI_STATE_LEN);
            break;

        case WR_TIMER_SEND_ERROR_CMD:                                    //发送错误信息
            WifiPro_SendMachineErrorCommand();
    }
}

```

```

        Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
        break;

    case WR_WAIT_SEND_CTRL_REQUEST_ACK_CMD: //WIFI模块请求发送控制数据
        WifiPro_SendMachineControlCommand();
        Het_DriveWifi_WifiDataSend(CMD_0304_STATUS_REQUEST_ACK,UartTxBuf,WIFI_CONTROL_LEN);
    ;
        break;

    case WR_WAIT_SEND_STATUS_REQUEST_ACK_CMD: //WIFI模块请求发送状态信息
        WifiPro_SendMachineStateCommand();
        Het_DriveWifi_WifiDataSend(CMD_0305_STATUS_REQUEST_ACK,UartTxBuf,WIFI_STATE_LEN);
        break;

    case WR_WIFI_TEST_OK: //WIFI模块告诉MCU测试OK
        break;
    case WR_WIFI_TEST_ERR: //WIFI模块告诉MCU测试出错
        break;

    default:
        break;
}
//该部分处理MCU主动上传的数据
if(UpgradeAp.UpControlStatusF == UPGRADE_FALSE) //保证不在升级状态
{
    //这个标志位很重要，主动上传控制命令是为了保证控制状态的实时性，服务器可以立刻获取到当前设备的控制状态
    if(WifiPro.LocalControlF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
    {
        WifiPro.LocalControlF = WIFIPRO_FALSE; //单片机主动发送控制状态信息
        WifiPro_SendMachineControlCommand();
        Het_DriveWifi_WifiDataSend(CMD_0104_CTRL,UartTxBuf,WIFI_CONTROL_LEN);
    }
    if(WifiPro.LocalErrorF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
    {
        WifiPro.LocalErrorF = WIFIPRO_FALSE; //单片机主动发送错误信息
        WifiPro_SendMachineErrorCommand();
        Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
    }
}
}
}
/*

```

- Function Name: main
- Description : 主函数
- Arguments :
- Return Value: void **/

```

int main(void) { Het_DriveWifi_WifiInit(WifiPro_WifiUartSend,WifiPro_UartRxPro,WifiPro_WifiReset);
//wifi模块化程序初始化
while (1) { ///////////////////////////////// //WIFI程序处理区
WifiPro_ProtocolPro(); //WIFI信息处理 ////////////////////////////// //应用程序
处理区 UserApplicationHandle(); } }

```

||||| 600b0dfbb830044fc4487bc06b1ead1ed04f4330

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 15:42:15

Clife Agent通讯模组使用教程 Clife Agent主要的作用是数据转发，是设备数据、Clife云、应用端（APP）的数据交互桥梁。

为了适应不同通讯模组接入Clife云，我们提供了多个模组的烧写方法以及日志打印的方法，以确保 GAgent固件在模组里正常运行。

注：合作厂商在进行模块烧写操作时请依次按照如下步骤进行，否则会产生不可控的错误** Clife Agent详细介绍 HF-LPB100串口烧写说明 HF-LPT120串口烧写说明 Realtek8710AF串口烧写说明 ESP8266串口烧写说明 广和通G510模组烧写说明 通讯模组调试日志获取教程

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-10 17:47:51

1.1 Wi-Fi简介

1.2 Wi-Fi固件支持

1.3 Wi-Fi模组SDK下载

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

- MCU与WiFi协议模块化软件使用说明
 - 一、此文档包含 3 个文档，分别为
 - 二、函数 API 说明
 - 2.1 void Het_DriveWifi_Wifilnit(pfUartSend _pf_uart_send,pfUartDecode _pf_uart_decode,pfWifiReset _pf_wifi_reset)
 - 2.2 void Het_DriveWifi_SystickISR (void)
 - 2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t het_data)
 - 2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t flag)
 - 2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t flag)
 - 2.6 het_uint8_t Het_DriveWifi_GetWifiStatus (void)
 - 2.7 het_bool Het_DriveWifi_GetAppSynStatus (void)
 - 2.8 enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType _type,het_uint8_t *_pbuf,het_uint8_t _len)
 - 2.9 enum_WResult Het_DriveWifi_WifiProcess (void)
 - 三、操作流程
 - 四、实例代码

MCU与WiFi协议模块化软件使用说明

一、此文档包含 3 个文档，分别为

1.DriveWifi.c – wifi绑定和通信模块驱动，不可更改文件。 2.DriveWifi.h – wifi驱动头文件,不可更改文件。 3.WifiConfig.h – wifi模块配置信息，根据不同的设备而修改。

二、函数 API 说明

2.1 void Het_DriveWifi_Wifilnit(pfUartSend _pf_uart_send,pfUartDecode _pf_uart_decode,pfWifiReset _pf_wifi_reset)

<1> Description:wifi 模组初始函数，用于注册用户函数。

<2> Arguments: _pf_uart_send – 用户串口发送字符串函数，函数名可自定义，函数原型必须为：
void fun(het_uint8_t pbuf,het_uint16_t len)。 _pf_uart_decode – 用户命令解码函数，函数名可自定
义，函数原型必须为： void fun(het_uint16_t cmd,het_uint8_t pbuf,het_uint16_t len)。
_pf_wifi_reset– 用户wifi 模组复位函数,函数名可自定义，函数原型必须为: void fun(het_uint8_t flag)。

<3> Return:NONE

2.2 void Het_DriveWifi_SystickISR (void)

<1> Description:此函数作用是给程序提供 10ms 时钟,必须放在 10ms 定时器里

<2> Arguments:NONE

<3> Return: NONE

2.3 void Het_DriveWifi_UsartRecvISR (het_uint8_t _het_data)

<1> Description: WIFI 模组串口中断接收函数,必须放在串口中断函数里面

<2> Arguments: data – 串口接收到的字节

<3> Return: NONE

2.4 void Het_DriveWifi_WifiModuleBindCmd (het_uint8_t _flag)

<1> Description: WIFI绑定触发条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能绑定操作

<3> Return: NONE

2.5 void Het_DriveWifi_WifiModuleTestCmd (het_uint8_t _flag)

<1> Description: WIFI 进入产测条件

<2> Arguments: flag – 如果 flag 大于 0,表示使能产测操作

<3> Return: NONE

2.6 het_uint8_t Het_DriveWifi_GetWifiStatus (void)

<1> Description: 获取 WiFi 连接状态函数

<2> Arguments: NONE

<3> Return: WiFi 状态值

2.7 het_bool Het_DriveWifi_GetAppSynStatus (void)

<1> Description: 获取 APP 同步状态函数

<2> Arguments: NONE

<3> Return: APP 同步值

2.8 enum_WResult Het_DriveWifi_WifiDataSend (enum_CMDType _type,het_uint8_t *_pbuff,het_uint8_t _len)

<1> Description: 发送用户私有数据函数,

<2> Arguments: type – 用户发送的数据所处类型, 如CMD_TYPE_CTRL 表示为控制命令数据, CMD_TYPE_STATUS 表示为状态命令数据 Pbuf – 用户发送数据缓存的首地址 Len – 用户发送数据的长度

<3> Return:发送的状态, WR_OK 表示发送成功

2.9 enum_WResult Het_DriveWifi_WifiProcess (void)

<1> Description:wifi 绑定和数据交互处理,此函数放在主循环里面

<2> Arguments:NONE

<3> Return:返回当前程序的状态。特别注意的是, 当返回状态等于 WR_WAIT_SEND_CTRL_CMD 或者 WR_TIMER_SEND_STATUS_CMD 时, 程序正处于 塞状态,等待用户输入设备控制和状态数据。

三、操作流程

1、修改 WiFiConfig.h 文件里面的如下配置信息：

<1>DEVICE_TYPE- 设备类型 (该信息在开放平台上创建产品后获得)

<2>DEVICE_KEY - 设备秘钥 (该信息在开放平台上创建产品后获得)

<3>MCU设备基本信息, 如SOFT_CONTROL_BOARD_VERSION等 (一般情况不用改) 此三项开发前申请。2、把 Het_DriveWifi_SystickISR 函数放在 10ms 定时器里。3、把 Het_DriveWifi_UsartRecvISR 函数放在 wifi 串口中断接收服务函数里。4、初始化 wifi 模块, 调用 DriveWifi.c 文件里面 Het_DriveWifi_WifiInit 函数。5、在主循环调用 Het_DriveWifi_WifiProcess 函数, Het_DriveWifi_WifiProcess 函数会返回 wifi 模组当前状态, 当返回的当前状态等于 WR_WAIT_SEND_CTRL_CMD 时, wifi 模组正等待设备调用 Het_DriveWifi_WifiDataSend 函数发送当前控制参数; 当返回的当前状态等于WR_TIMER_SEND_STATUS_CMD 时, wifi 模组正在等待设备调用 Het_DriveWifi_WifiDataSend 函数发送当前状态参数; 但是返回的当前状态等于 WR_TIMER_SEND_ERROR_CMD 时, wifi 模组正在等待设备调用 Het_DriveWifi_WifiDataSend 函数发送当前故障信息。当用户需要主动上传数据时, 也可以调用 Het_DriveWifi_WifiDataSend 函数发送数据。

四、实例代码

```
//-----
-----  
//操作头文件WiFiConfig.h对wifi通讯进行设备  
#define MOUDLE_TYPE           MODULE_WIFI          /*choose the type of module*/  
#define BIND_TYPE              SMARTLINK_MODE    /*choose the type of bind*/  
#define BIND_PLATFORM          WECHAT            /*connect the platform*/  
#define HET_FACTORY_TEST_EN    1                 /*1:enable  0:disable*/  
#define HET_UPGRADE_FW_EN      1                 /*1:enable  0:disable*/  
#define HET_FILE_UPLOAD_EN     1                 /*1:enable  0:disable*/  
#if BIND_PLATFORM == WECHAT  
#define DEVICE_TYPE_WECHAT    "gh_d04b5c778911"  
#define DEVICE_ID_WECHAT      "clife_7879_"  
#define DEVICE_TYPE_LEN_WECHAT 15                /*1:enable  0:disable*/  
#define DEVICE_ID_LEN_WECHAT   11  
#endif  
#define DEVICE_TYPE 0x00,0x00,0xC5,0x85,0x00,0x5A,0x01,0x01 //设备编码
```

```

#define DEVICE_KEY "5aa5fba81a864beb859f0fd8234d32dc"           //设备devicekey
#define DEVICE_INFO_LEN          48                           //设备信息
#define SOFT_CONTROL_BOARD_VERSION 0x01                      /*ctrl board software version*/
#define SOFT_DISPLAY_BOARD_VERSION 0x01                      /*display board software version*/
#define SOFT_DRIVE_BOARD_VERSION 0x01                        /*drive board software version*/

#define SOFT_OTHER_VERSION        0x01                      /*other board software version*/

#define HARD_CONTROL_BOARD_VERSION 0x01                      /*ctrl board hardware version*/
#define HARD_DISPLAY_BOARD_VERSION 0x01                      /*ctrl board hardware version*/
#define HARD_DRIVE_BOARD_VERSION 0x01                        /*ctrl board hardware version*/

#define HARD_OTHER_VERSION        0x01                      /*ctrl board hardware version*/

#define DEVICE_MAIN_TYPE_H       0X00                      /*device main type high byte*/
#define DEVICE_MAIN_TYPE_L       0x4B                      /*device main type low byte*/

#define DEVICE_SUB_TYPE          0x03                      /*device sub type*/
#define DEVICE_FUN_TYPE          0x02                      /*device function type */

#define DEVICE_PROJECT_NUMBER    "Z029-A16033A"           /*project number*/
#define WIFI_CONTROL_LEN         48                           /*控制数据长度*/
#define WIFI_STATUS_LEN          48                           /*运行数据长度*/
#define WIFI_ERROR_LEN           16                           /*故障数据长度*/
//-----
-----
```

```

//main.c //用户实现3个函数 /** 名称: WiFiPro_UartRxPro 功能: 处理WIFI下发给MCU的指令 输入:
命令, 缓存指针, 缓存长度 输出: 说明: */ void WiFiPro_UartRxPro(unsigned short _cmd,unsigned
char *_buf,unsigned short _data_len) {
if(_cmd== CMD_0104_CTRL) //处理服务器下发的控制命令 { //用户代码, 用于解析下发的控制命令
WiFiPro_ControlHandle(_buf,_data_len); //MCU端响应控制指令
WiFiPro_SendMachineControlCommand();
Het_DriveWifi_WifiDataSend(CMD_0204_CTRL_ACK,UartTxBuf,16); } else if(_cmd==
CMD_0107_CONFIG) //接收服务器下发的硬件配置命令 { //用户代码, 用于解析下发的配置命令
```

```

    //MCU端响应配置指令
    Het_DriveWifi_WifiDataSend(CMD_0207_CONFIG_ACK,UartTxBuf,0);
}
else if(_cmd == CMD_0206_GET_TIME_ACK)                                //接受服务器下发的时间信息
{
;
}
else if(_cmd == CMD_0120_UPGRADE_START)                                 //收到升级起始帧, 开启升
级程序
{
;
}
else if(_cmd == CMD_0122_UPGRADE_RECV_PACKET)                         //接收到升级的数据
{
;
}
else if(_cmd == CMD_0225_UPGRADE_END_ACK)                            //升级成功
```

```

{
    ;
}

/** 名称: WifiPro_WifiReset 功能: WIFI复位状态设定 输入: 复位状态 输出: 说明: */ void
WifiPro_WifiReset(het_uint8_t _status) { if(_status) { WIFI_RESET_LOW; } else {
WIFI_RESET_HIGH; } } /** 名称: WifiPro_ProtocolPro 功能: WIFI模块与MCU之间的通信处理 输入: 输出: 说明: 每10ms处理一次 */ void WifiPro_ProtocolPro(void) { enum_WResult run_wifi;

//该部分处理WIFI模块下发的数据
run_wifi = Het_DriveWifi_WifiProcess();
switch(run_wifi)
{
    case WR_WAIT_SEND_CTRL_CMD:                                //该部分是第一次连接服务器
时设备端主动向服务器上传一次的控制状体信息
        WifiPro_SendMachineControlCommand();                      //MCU端响应WIFI模块的控制指令
    break;

    case WR_TIMER_SEND_STATUS_CMD:                             //每5S发送一次状态信息
        WifiPro_SendMachineStateCommand();
        Het_DriveWifi_WifiDataSend(CMD_0104_CTRL,UartTxBuf,WIFI_CONTROL_LEN);
    break;

    case WR_TIMER_SEND_ERROR_CMD:                            //发送错误信息
        WifiPro_SendMachineErrorCommand();
        Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
    break;

    case WR_WAIT_SEND_CTRL_REQUEST_ACK_CMD:                 //WIFI模块请求发送控制数据
        WifiPro_SendMachineControlCommand();
        Het_DriveWifi_WifiDataSend(CMD_0304_STATUS_REQUEST_ACK,UartTxBuf,WIFI_CONTROL_LEN);
    break;

    case WR_WAIT_SEND_STATUS_REQUEST_ACK_CMD:                //WIFI模块请求发送状态信息
        WifiPro_SendMachineStateCommand();
        Het_DriveWifi_WifiDataSend(CMD_0305_STATUS_REQUEST_ACK,UartTxBuf,WIFI_STATE_LEN);
    break;

    case WR_WIFI_TEST_OK:                                    //WIFI模块告诉MCU测试OK
        break;
    case WR_WIFI_TEST_ERR:                                  //WIFI模块告诉MCU测试出错
        break;

    default:
        break;
}
//该部分处理MCU主动上传的数据
if(UpgradeAp.UpControlStatusF == UPGRADE_FALSE)           //保证不在升级状态
{
    //这个标志位很重要，主动上传控制命令是为了保证控制状态的实时性，服务器可以立刻获取到当前设备的
    //控制状态
}

```

```

if(WifiPro.LocalControlF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
{
    WifiPro.LocalControlF = WIFIPRO_FALSE;                                //
    WifiPro_SendMachineControlCommand();                                     //单片机主动发送控制状态信息
    Het_DriveWifi_WifiDataSend(CMD_0104_CTRL,UartTxBuf,WIFI_CONTROL_LEN);
}
if(WifiPro.LocalErrorF == WIFIPRO_TRUE && WifiPro.Tick10msCnt == WIFI_SEND_IDLE)
{
    WifiPro.LocalErrorF = WIFIPRO_FALSE;                                //
    WifiPro_SendMachineErrorCommand();                                    //单片机主动发送错误信息
    Het_DriveWifi_WifiDataSend(CMD_010E_ERROR,UartTxBuf,WIFI_ERROR_LEN);
}
}

```

{ /*

- Function Name: main
- Description : 主函数
- Arguments :
- Return Value: void **/

```

int main(void) { Het_DriveWifi_WifiInit(WifiPro_WifiUartSend,WifiPro_UartRxPro,WifiPro_WifiReset);
//wifi模块化程序初始化
while (1) { ///////////////////////////////// //WIFI程序处理区
WifiPro_ProtocolPro(); //WIFI信息处理 ///////////////////////////////// //应用程序
处理区 UserApplicationHandle(); } }

```

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

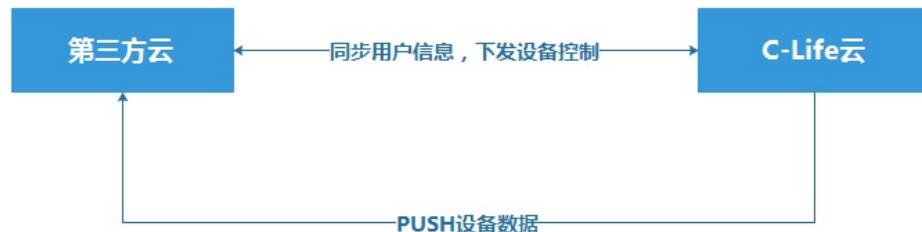
Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 19:50:21

- C-Life 开放平台云接口
 - 一、接入方式
 - 二、API调用流程
 - 三、PUSH设备数据接入说明
 - 接入方案配置
 - 查询推送失败的设备数据
 - 四、变量说明
 - 4.1 全局返回码
 - 4.2 分页信息
 - 4.3 全局变量说明
 - 五、访问凭证接口
 - 1、获取接口访问凭证
 - 六、用户相关接口
 - 1、获取用户基本信息
 - 2、获取用户列表
 - 3、获取用户设备信息
 - 七、设备相关接口
 - 1、下发设备控制数据
 - 2、获取设备最新控制数据
 - 3、获取设备最新运行数据
 - 4、获取设备最新故障信息
 - 5、批量下发控制命令
 - 6、获取控制命令执行结果
 - 7、广播下发控制命令
 - 八、PUSH设备数据接口
 - 1、查询推送失败的设备数据
 - 推送数据格式说明
 - 九、第三方授权相关接口
 - 1、第三方云请求CLife云验证授权码

C-Life 开放平台云接口

一、接入方式



如上图所示，C-Life云支持第三方云通过API同步用户信息、设备信息，下发控制设备和主动PUSH设备数据；

API：开发者可以根据C-Life云的API来封装HTTPS请求进行调用。

PUSH设备数据：PUSH设备数据是C-Life云提供给第三方云的一种推送服务，让第三方云及时收集所属产品的设备数据，为后续的产品研发提供最可靠的数据支持。

二、API调用流程

1. 开发者申请appId和appSecret

appId作为服务端向开发者发放的授权标识，调用所有接口时的必传参数，用于后台对接口调用的权限管理和调用频次限制。

appSecret是服务端分配给开发者的密钥，主要用于请求签名加密之用，证明开发者的合法身份。

2. 开发者在平台绑定调用接口的IP白名单

绑定开发者调用平台接口的服务器IP

3. 开发者调用TOKEN接口从平台拿到accessToken

accessToken是全局唯一接口调用凭据，调用各接口时都需使用accessToken，开发者需要进行妥善保存。accessToken的有效期目前为2个小时，需定时刷新，重复获取将导致上次获取的accessToken失效。

4. 开发者使用accessToken参数调用其他业务接口

5. 开发者提交timestamp参数的值为当前北京时间戳

6. API签名机制

步骤：

1. 将请求方法+请求地址+所有参数按照参数名的字母顺序升序排序后拼接+appSecret
2. 将第1步的结果MD5后生成签名参数sign。

说明：

1. 请求方法为GET或POST，注意大写。
2. 请求地址为客户端请求完整地址，如：<http://open.api.clife.cn/v1/data/get>
3. 参数拼接，按照参数名的字母顺序升序排序后进行拼接，如：accessToken=xxx&appId=xxx&name1=value1&name2=value2&
4. 组成完整的拼接，如：GET<http://open.api.clife.cn/v1/data/get>accessToken=xxx&appId=xxx&name1=value1&name2=value2&appSecret
5. 生成MD5值，如：74B024F42F3075C4C06E4E8F22CA7A5F

三、PUSH设备数据接入说明

PUSH设备数据是C-Life云提供给第三方云的一种推送服务，让第三方云及时收集所属产品的设备数据，为后续的产品研发提供最可靠的数据支持。

当C-Life云推送服务发送设备数据给第三方云服务接口时，会产生一个POST请求，第三方云服务接口需要在响应包(GET)中返回特定JSON结构，来对该推送进行响应。

C-Life云推送服务在将设备数据发给第三方云的推送服务器地址(开放平台产品接入的接入方案处配置)后，C-Life云推送服务在三秒内收不到响应会断掉连接，并且重新发起请求，总共重试三次。关于重试的设备数据排重，使用msgId进行排重。

假如第三方云服务器无法保证在三秒内处理并回复，必须做出下述回复，这样C-Life云推送服务才不会对此作任何处理，并且不会发起重试，否则，将出现第三方云服务接口被锁定的情况。详见下面说明：

- 1、第三方云服务接口验证阶段 (html包体没设备数据) 时，直接回复{"code":0,data:token字符串}。
- 2、第三方云服务接口接收推送服务 (html包体有设备数据) 时，直接回复{"code":0,data:msgId字符串}。

接入方案配置

第三方云可以到C-Life开放平台的产品接入中的接入方案处进行配置。

流程如下：

- 1)选择第三方云连接C-Life硬件云通道(设备连接第三方云服务器后，可通过设备openAPI与C-Life硬件云对接)
- 2)填写URL和Token
URL：第三方云服务接口的唯一标识，供C-Life云推送服务给第三方云推送数据用的。（现在仅支持http方式）
Token：第三方云服务接口在和C-Life云推送服务的凭证，用来验证厂商服务接口的合法性。

请注意：

第三方云在接入方案配置前，请确保第三方云服务接口能够正常运行。当接入方案配置完成时，C-Life云即向第三方云服务接口URL发送一个空请求，第三方云服务接口此时就能返回规定的JSON字符串响应({ "code":0,data:Token字符串}，Token字符串即为页面上所填写的Token内容)，否则C-Life云会发起重试机制。

查询推送失败的设备数据

第三方云发现推送设备数据有缺失的话，可以调用此接口获取产品的推送失败的设备数据。C-Life云会保留30天的推送失败的设备数据。

推送数据失败有下述原因：

- 1)推送时第三方云服务接口出现问题；
- 2)第三方云服务接口没有返回规定的JSON字符串响应；({ "code":0,data:msgId字符串}，msgId为推送设备数据的唯一标识，第三方云服务接口原样返回即可)
- 3)第三方云服务处理业务超时没返回响应；(推送请求超时时间为三秒)
- 4)第三方云服务接口返回的msgId与和C-Life云推送服务发送的msgId不一致。

四、变量说明

4.1 全局返回码

全局返回码说明

每次调用接口时，可能获得正确或错误的返回码，可以根据返回码信息调试接口，排查错误。

全局返回码说明如下：

返回码	说明
0	请求成功
100010100	缺少授权信息
100010101	AccessToken错误或已过期
100010103	AppId不合法
100010104	timestamp过期
100010105	签名错误
100010111	timestamp不合法
100010200	失败
100010201	缺少参数
100010202	参数错误

4.2 分页信息

分页JSON格式：

```
"pager" : {
    "pageIndex": 1,
    "pageRows": 20,
    "totalRows": 45,
    "totalPages": 3,
    "defaultPageRows": 20,
    "currPageRows": 20,
    "pageStartRow": 0,
    "pageEndRow": 19,
    "hasPrevPage": false,
    "hasNextPage": true
}
```

分页信息说明：

字段名称	基础 信息	字段类 型	字段说明

pageIndex	是	number	请求的页（从1开始），不设则默认为1
pageRows	是	number	请求的每页行数，不设则默认为defaultPageRows
totalRows	否	string	总行数
totalPages	否	number	总页数
defaultPageRows	是	number	默认每页行数：20
currPageRows	否	number	当前页的实际行数
pageStartRow	是	number	当前页的起始行（从0开始，有可能超出总行数）
pageEndRow	是	number	当前页的结束行（从0开始，有可能超出总行数）
hasPrevPage	否	boolean	是否有前一页
hasNextPage	否	boolean	是否有后一页

4.3 全局变量说明

字段	说明
deiveld	加密后的设备标识
openId	加密后的开放账号标识

五、访问凭证接口

1、获取接口访问凭证

接口调用请求说明

```
http请求方式: POST
https://open.api.clife.cn/v1/cloud/token
```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
appSecret	是	string	凭证密钥
timestamp	是	number	时间戳

返回结果

正确的Json返回结果：

```
{
  "data": {
    "accessToken": "677c11e05aac4c85b6442ba99c6653a1",
    "expiresIn": 28800
  },
  "code": 0
}
```

字段名称	字段类型	字段说明
accessToken	string	接口调用凭证
expiresIn	string	accessToken接口调用凭证超时时间, 单位 (秒)

六、用户相关接口

1、获取用户基本信息

接口调用请求说明

http请求方式: POST
<https://open.api.clife.cn/v1/cloud/user/get>

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
openId	是	string	openId (APP SDK里OAuth授权后返回的openId)

返回结果

正确的Json返回结果:

```
{
  "code":0,
  "data":{
    "userNmae": "葫芦娃",
    "sex": 1,
    "birthday": "2014-12-31",
    "weight": 48000,
    "height": 163,
    "avatar": "",
    "city": "深圳"
    "regTime": "2014-12-31 00:00:00"
  }
}
```

字段名称	字段类型	字段说明
userName	string	用户名
sex	number	性别 (1-男, 2-女)
birthday	string	生日 (yyyy-MM-dd)
weight	number	体重 (克)
height	number	身高 (厘米)
avatar	string	头像URL
city	string	用户所在城市
regTime	string	用户注册时间

2、获取用户列表

接口调用请求说明

```
http请求方式: POST
https://open.api.clife.cn/v1/cloud/user/list
```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
pageRows	否	number	每页显示的行数 (默认为20,最大为1000)
pageIndex	否	number	当前页

返回结果

正确的Json返回结果：

```
{
    "code": 0,
    "data": {
        "list": [
            "OPENID1",
            "OPENID2"
        ],
        "pager": {
            "totalRows": 2,
            "pageRows": 10,
        }
    }
}
```

```

        "pageIndex":1,
        "paged":false,
        "pageStartRow":0,
        "pageEndRow":9,
        "hasPrevPage":false,
        "hasNextPage":false,
        "defaultPageRows":20,
        "totalPages":1,
        "currPageRows":2
    }
}
}
}

```

字段名称	字段类型	字段说明
list	数组	开放账号标识列表

3、获取用户设备信息

接口调用请求说明

```

http请求方式: POST
https://open.api.clife.cn/v1/cloud/user/device/list

```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
openId	是	string	openId

返回结果

正确的Json返回结果：

```

{
    "data": [
        {
            "deviceId": "501D275D6CD840F39FF862CC9AE3ABBA",
            "macAddress": "5c313e08fc09",
            "deviceName": "CC13653",
            "bindTime": "2015-06-11 06:00:03",
            "onlineStatus":1,
            "deviceCode": "0000C3AA00010105"
        }],
    "code": 0
}

```

字段名称	字段类型	字段说明
deviceId	string	设备标识
macAddress	string	MAC地址
deviceName	string	设备名称
bindTime	string	绑定时间
onlineStatus	number	在线状态 (1-正常, 2-异常)
deviceCode	string	设备编码

七、设备相关接口

1、下发设备控制数据

接口调用请求说明

```
http请求方式: POST
https://open.api.clife.cn/v1/cloud/device/config/single/set
```

ps:需要通知CLIFE云添加应用标识白名单

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	访问凭证
timestamp	是	number	时间戳
deviceId	是	string	设备标识
json	是	string	设备控制JSON
sign	是	string	签名

返回结果

正确的Json返回结果:

```
{"code":0}
```

2、获取设备最新控制数据

接口调用请求说明

http请求方式: GET
<https://open.api.clife.cn/v1/cloud/device/config/get>

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	访问凭证
timestamp	是	number	时间戳
deviceIds	是	string	设备标识（多个设备用英文逗号隔开，最多支持20个）

返回结果

正确的Json返回结果：(ps:data里包括设备ID及其控制数据)

```
{
    "code": 0,
    "data": [
        {
            "75a06180045a9f4e8228f330bb76f5fa": {
                "Airlevel": 1,
                "mode": 2,
                "AddLiquid": 0,
                "wash": 1
                "sound": 2,
                "updateFlag": "0000"
            }
        },
        {
            "2937f353e51177cec466067f3f430ea7": {
                "fanGear": 8,
                "childlock": 1,
                "onoff": 1,
                "uvsw": 2
                "mode": 0
            }
        }
    ]
}
```

3、获取设备最新运行数据

接口调用请求说明

http请求方式: GET
<https://open.api.clife.cn/v1/cloud/device/data/get>

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	访问凭证
timestamp	是	number	时间戳
deviceIds	是	string	设备标识（多个设备用英文逗号隔开，最多支持20个）

返回结果

正确的Json返回结果：(ps:data里包括设备ID及其运行数据)

```
{
  "code": 0,
  "data": [
    {
      "75a06180045a9f4e8228f330bb76f5fa": {
        "Deodorant": 3,
        "Ammoniahigh": 0,
        "Ammonialow": 0
      }
    },
    {
      "2937f353e51177cec466067f3f430ea7": {
        "RemainTime": 255,
        "powerOn": 1,
        "airQty": 3,
        "ambientTemp": 168,
        "anionSw": 2,
        "wifiSignal": 118,
        "ozoneSw": 2
      }
    },
    {
      "ffdab511175cd210ff53bb50be985817": {
        "FreezeWorkStatus": 1,
        "RefrDefrostCycle": "0",
        "dataTimeStamp": 1503252476902,
        "FreezeDefrostCy": "0"
      }
    }
  ]
}
```

4、获取设备最新故障信息

接口调用请求说明

```
http请求方式: GET
https://open.api.clife.cn/v1/cloud/device/data/getErrorData
```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	访问凭证
timestamp	是	number	时间戳
deviceIds	是	string	设备标识 (多个设备用英文逗号隔开, 最多支持20个)

返回结果

正确的Json返回结果: (ps:data里包括设备ID及其故障数据)

```
{
  "code": 0,
  "data": [
    {
      "75a06180045a9f4e8228f330bb76f5fa": {
        "FanError2": 0,
        "AmmoniaSensorError": 0,
        "SolenoidValveError2": 0,
        "AtomizerError": 0,
        "FanError": 0
      }
    },
    {
      "2937f353e51177cec466067f3f430ea7": {
        "AirQtyErr": 0,
        "dataTimeStamp": 1499295434552,
        "PM25SnrErr": 0,
        "LeakCurrentErr": 0
      }
    },
    {
      "fffdab511175cd210ff53bb50be985817": {
        "WiFiFlashErr": 0,
        "dataTimeStamp": 1503252319761,
        "reserved10": "",
        "FreezeCoilHeatErr": 0,
        "FreezeFanErr": 0,
        "CompErr": 0,
        "RefrCoilTempErr": 0
      }
    }
  ]
}
```

5、批量下发控制命令

接口调用请求说明

```
http请求方式: POST
https://open.api.clife.cn/v1/cloud/device/config/set

ps:需要通知CLIFE云添加应用标识白名单
```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
json	是	string	设备控制JSON
deviceIds	是	string	设备标识列表(deviceId之间以逗号间隔)

返回结果

正确的Json返回结果：

```
返回流水批次号

{
    "code": 0,
    "data": "152017032818000000025"
}
```

6、获取控制命令执行结果

接口调用请求说明

```
http请求方式: POST
https://open.api.clife.cn/v1/cloud/device/config/getPushReprot
```

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
batchSendNo	是	string	流水批次号

返回结果

正确的Json返回结果：

```
如果没有data节点，表明处理结果还没生成。
```

```
{
    "batch_send_no": "152017032818000000025",
    "online_num": 3,
    "offline_num": 2,
```

```

"illegal_num":1,
"online_success_num":2,
"online_fail_num":1,
"detailResult":{
    "batchSendOnlineResult":{
        "send_success":[
            "EE91A91EEE12BF98EB4E4040C438FC9D","EE91A91EEE12BF98EB4E4040C438FC1D"
        ],
        "send_fail":[
            "EE91A91EEE12BF98EB4E4040C438FC2D"
        ]
    },
    "offline_info":[
        "EE91A91EEE12BF98EB4E4040C438FC3D","EE91A91EEE12BF98EB4E4040C438FC4D"
    ],
    "illegal_info":[
        "EE91A91EEE12BF98EB4E4040C438FC5D"
    ]
}
}
}

```

字段名称	字段类型	字段说明
batch_send_no	string	流水批次号
online_num	int	在线数
offline_num	int	离线数
online_success_num	int	在线设备推送成功数
online_fail_num	int	在线设备推送失败数
illegal_num	int	无效设备数(广播接口无此属性)
detailResult		推送结果详情(广播接口无此属性)
batchSendOnlineResult		在线推送详情
send_success	数组	推送成功MAC列表
send_fail	数组	推送失败MAC列表
offline_info	数组	离线MAC列表
illegal_info	数组	无效MAC列表

7、广播下发控制命令

接口调用请求说明

http请求方式: POST
<https://open.api.clife.cn/v1/cloud/device/config/broadcast>

参数说明

参数名称	是否	字段类	参数说明

参数名称	必须	型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
json	是	string	设备控制JSON
productId	是	int	产品标识

返回结果

正确的Json返回结果：

返回流水批次号

```
{
  "code": 0,
  "data": "152017032818000000025"
}
```

八、PUSH设备数据接口

1、查询推送失败的设备数据

接口调用请求说明

http请求方式：POST
<https://open.api.clife.cn/v1/cloud/devicepush/getPushFailData>

参数说明

参数名称	是否必须	字段类型	参数说明
appId	是	string	应用标识
accessToken	是	string	接口调用凭证
timestamp	是	number	时间戳
productId	是	number	产品标识
quaryDate	否	string	查询天数（默认为7天，最多30天）
pageRows	否	number	每页显示的行数（默认为20）
pageIndex	否	number	当前页

返回结果

正确的Json返回结果：

```

    "code":0,
    "data":{
        "list":[{
            "productId":58,
            "content":"{...}"
        }],
        "pager":{
            "totalRows":2,
            "pageRows":10,
            "pageIndex":1,
            "paged":false,
            "pageStartRow":0,
            "pageEndRow":9,
            "hasPrevPage":false,
            "hasNextPage":false,
            "defaultPageRows":20,
            "totalPages":1,
            "currPageRows":2
        }
    }
}

```

字段名称	字段类型	字段说明
productId	int	产品ID
content	string	失败消息内容，格式详情请看下面说明

推送数据格式说明

推送服务的POST数据示例如下：

```

{
    "msgId":"000ad405-e7c4-44a7-84fd-f55f1d653efd",
    "list":[{
        "timestamp":1490858478589,
        "dataType":3,
        "data":{
            "authUserId":"7F74CC9DFEC3047685DAD6E48F142C4B",
            "breathRate":0,
            "activityEnergy":0,
            "source":3,
            "hasAnybody":0,
            "heartRate":0,
            "userId":"7F74CC9DFEC3047685DAD6E48F142C4B",
            "linkStatus":1,
            "timeZone":480,
            "turnOverTimes":0,
            "dataTime":1490858473434,
            "batteryPower":39
        },
        "mac":"405EE110006B"
    },
    {
        "timestamp":1490858478589,
        "dataType":3,

```

```

    "timestamp":1490858478589,
    "dataType":3,
    "data":{
        "authUserId":"7F74CC9DFEC3047685DAD6E48F142C4B",
        "breathRate":0,
        "activityEnergy":1,
        "source":3,
        "hasAnybody":0,
        "heartRate":0,
        "userId":"7F74CC9DFEC3047685DAD6E48F142C4B",
        "linkStatus":1,
        "timeZone":480,
        "turnOverTimes":0,
        "dataTime":1490858473434,
        "batteryPower":39
    },
    "mac":"405EE110006B"
}
}

```

字段名称	字段类型	字段说明
msgId	string	消息标识
list	数组	设备数据列表
mac	string	设备MAC
dataType	int	数据类型 2-控制数据 3-运行数据 4-故障数据
timestamp	date	时间戳，数据生成时间
data		设备数据区
authUserId	string	设备协议属性
breathRate	int	设备协议属性
...	...	设备协议属性

九、第三方授权相关接口

1、第三方云请求CLife云验证授权码

接口调用请求说明

http请求方式: POST
<https://open.api.clife.cn/v1/cloud/user/checkAuth>

参数说明

参数名称	是否必须	字段类型	参数说明

appSecret	是	string	应用密钥AppSecret，在C-Life开放平台提交应用审核通过后获得
authorizationCode	是	string	第一步得到的授权码，用于获取随机码
accessToken	是	string	访问凭证
timestamp	是	number	时间戳

返回结果

正确的Json返回结果：

```
{
  "code":0,
  "data":
  {
    "randomCode":"b7573dbadfe84ba2b3659d1e49f8bf08"
  }
}
```

字段名称	字段类型	字段说明
randomCode	string	随机码

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间：2017-11-10 17:47:51

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-10-30 13:49:37

- 7. 推送服务

7. 推送服务

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 16:09:31

- **开放平台Android APP消息推送集成指南**
 - 概述
 - **快速集成极光推送**
 - 1.极光推送集成准备
 - 2.APP集成极光推送
 - **快速集成百度推送**
 - 1.百度推送集成准备
 - 2.APP集成百度推送

开放平台Android APP消息推送集成指南

概述

开放平台APP消息推送集成了极光推送和百度推送功能，封装了推送接口。替换申请的相关KEY值和相关配置就可以直接使用，简单快捷。节省开发者集成过程的繁琐细节。本文档基于开放平台APP消息推送集成流程，详细介绍集成百度推送和极光推送流程和注意事项。

快速集成极光推送

1.极光推送集成准备

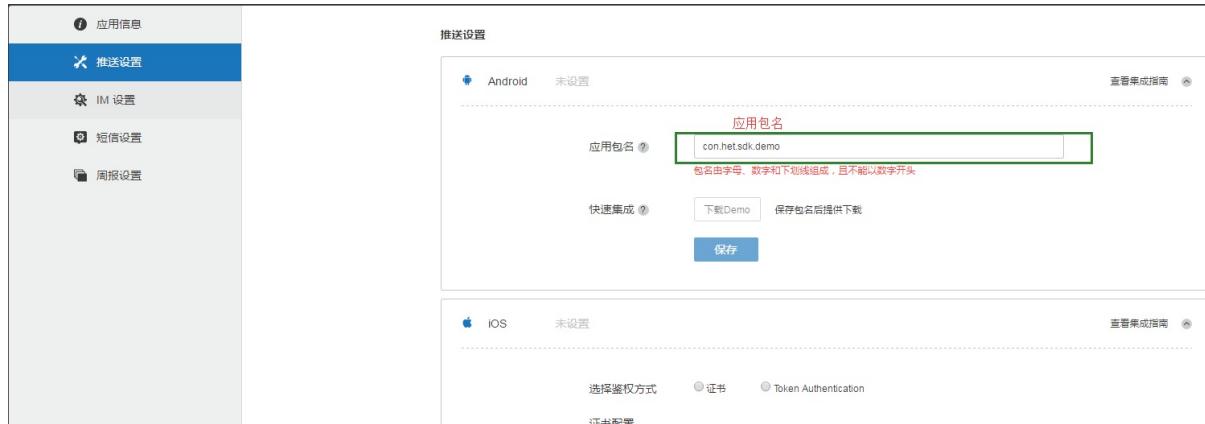
1.1.申请极光开发者账号，创建应用。

The screenshot shows the 'Application Information' section of the JPush developer console. On the left, there's a sidebar with tabs: 'Application Information' (selected), 'Push Settings', 'IM Settings', 'SMS Settings', and 'Report Settings'. The main area displays the following details for the application 'heti测试App':

- Application Name:** heti测试App
- Application Icon:** A placeholder icon.
- AppKey:** b870a5ac7f2ab2442598b68b (highlighted with a red box)
- Master Secret:** A button with 'View' and 'Reset Master Secret' options.
- Creation Time:** 2017-11-13 16:32:04
- Last Modified Time:** 2017-11-13 16:32:24
- Server Location:** Beijing Data Center

 Below this, there's a 'Push Settings' section with a note: 'Use push service or IM business before completing push settings.' and a 'Finish Push Settings' button.

1.1.2.设置Android推送



1.1.3. 查看对应的AppKey和Master Secret



2.APP集成极光推送

目前极光推送SDK只支持Android 2.3或以上版本的手机系统。富媒体信息流功能则需Android3.0或以上版本的系统。

2.1. 在 module 的 gradle 中添加依赖

```
android {
    .....
    defaultConfig {
        applicationId "com.het.sdk.demo" //JPush上注册的包名和工程包名必须要一致
        .....

        ndk {
            //选择要添加的对应cpu类型的.so库。
            abiFilters 'armeabi', 'armeabi-v7a', 'arm64-v8a'
            // 还可以添加 'x86', 'x86_64', 'mips', 'mips64'
        }

        manifestPlaceholders = [
            JPUSH_PKGNAME : "com.het.sdk.demo", //填包名
            JPUSH_APPKEY : "你的appkey", //JPush上注册的包名对应的appkey.
            JPUSH_CHANNEL : "developer-default", //暂时填写默认值即可.
        ]
        .....
    }
    .....
}
```

```

}

dependencies {
    .....

    compile 'cn.jiguang.sdk:jpush:3.0.5' // 此处以JPush 3.0.5 版本为例。
    compile 'cn.jiguang.sdk:jcore:1.1.2' // 此处以JCore 1.1.2 版本为例。
    .....
}

```

注：如果在添加以上 abiFilter 配置之后 android Studio 出现以下提示：

```
NDK integration is deprecated in the current plugin. Consider trying the new experimental plugin
```

则在 Project 根目录的 gradle.properties 文件中添加：

```
android.useDeprecatedNdk=true
```

2.2.AndroidManifest的替换变量

```

<?xml version="1.0" encoding="utf-8"?>
<manifest      android="http://schemas.android.com/apk/res/android"
    package="您应用的包名">

    <!-- Required -->
    <permission
        name="您应用的包名.permission.JPUSH_MESSAGE"
        protectionLevel="signature" />

    <uses-permission          name="android.permission.RECEIVE_USER_PRESENT" />
    <uses-permission          name="android.permission.INTERNET" />
    <uses-permission          name="android.permission.WAKE_LOCK" />
    <uses-permission          name="android.permission.READ_PHONE_STATE" />
    <uses-permission          name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission          name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission          name="android.permission.VIBRATE" />
    <uses-permission          name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
    <uses-permission          name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission          name="android.permission.WRITE_SETTINGS" />
    <uses-permission          name="android.permission.ACCESS_WIFI_STATE" />

    <!-- Optional. Required for location feature -->
    <uses-permission          name="android.permission.SYSTEM_ALERT_WINDOW" /> <!-- 用于开启
debug 版本的应用在6.0 系统上 层叠窗口权限 -->
    <uses-permission          name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission          name="android.permission.CHANGE_WIFI_STATE" />
    <uses-permission          name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission          name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
    <uses-permission          name="android.permission.CHANGE_NETWORK_STATE" />
    <uses-permission          name="android.permission.GET_TASKS" />

    <application
        icon="@drawable/ic_launcher"
        label="@string/app_name"

```

```

        name="Your Application Name">

        <!-- =====极光推送开始===== -->
<!-- Required SDK 核心功能 -->
<!-- option since 2.0.5 可配置PushService, DaemonService,PushReceiver,AlarmReceiver的an
droid:process参数 将JPush相关组件设置为一个独立进程 -->
<!-- 如: android:process=":remote" -->
<service
        name="cn.jpush.android.service.PushService"
        enabled="true"
        exported="false">
<intent-filter>
        <action          name="cn.jpush.android.intent.REGISTER" />
        <action          name="cn.jpush.android.intent.REPORT" />
        <action          name="cn.jpush.android.intent.PushService" />
        <action          name="cn.jpush.android.intent.PUSH_TIME" />
</intent-filter>
</service>

<!-- since 1.8.0 option 可选项。用于同一设备中不同应用的JPush服务相互拉起的功能。 -->
<!-- 若不启用该功能可删除该组件，将不拉起其他应用也不能被其他应用拉起 -->
<!-- <service -->
<!-- android:name="cn.jpush.android.service.DaemonService" -->
<!-- android:enabled="true" -->
<!-- android:exported="true" -->
<!-- tools:ignore="ExportedService" -->
<!-- <intent-filter> -->
<!-- <action android:name="cn.jpush.android.intent.DaemonService" /> -->

<!-- <category android:name="您应用的包名" /> -->
<!-- </intent-filter> -->
<!-- </service> -->

<!-- Required -->
<receiver
        name="cn.jpush.android.service.PushReceiver"
        enabled="true"
        ignore="ExportedReceiver">
<intent-filter          priority="1000">
        <action          name="cn.jpush.android.intent.NOTIFICATION RECEIVED_PROXY" />

        <category          name="您应用的包名" />
</intent-filter>
<intent-filter>
        <action          name="android.intent.action.USER_PRESENT" />
        <action          name="android.net.conn.CONNECTIVITY_CHANGE" />
</intent-filter>
<!-- Optional -->
<intent-filter>
        <action          name="android.intent.action.PACKAGE_ADDED" />
        <action          name="android.intent.action.PACKAGE_REMOVED" />

        <data          scheme="package" />
</intent-filter>
</receiver>
<!-- Required SDK核心功能 -->
<activity
        name="cn.jpush.android.ui.PushActivity"

```

```

        configChanges="orientation|keyboardHidden"
        exported="false">
<intent-filter>
    <action      name="cn.jpush.android.ui.PushActivity" />

    <category     name="android.intent.category.DEFAULT" />
    <category     name="您应用的包名" />
</intent-filter>
</activity>
<!-- Required SDK核心功能 -->
<service
        name="cn.jpush.android.service.DownloadService"
        enabled="true"
        exported="false" />
<!-- Required SDK核心功能 -->
<receiver      name="cn.jpush.android.service.AlarmReceiver" />

<!-- User defined. 用户自定义的广播接收器 -->
<receiver
        name="自定义 Receiver"
        enabled="true"
        ignore="ExportedReceiver">
    <intent-filter>

        <!-- Required 用户注册SDK的intent -->
        <action      name="cn.jpush.android.intent.REGISTRATION" />
        <!-- Required 用户接收SDK消息的intent -->
        <action      name="cn.jpush.android.intent.MESSAGE_RECEIVED" />
        <!-- Required 用户接收SDK通知栏信息的intent -->
        <action      name="cn.jpush.android.intent.NOTIFICATION_RECEIVED" />
        <!-- Required 用户打开自定义通知栏的intent -->
        <action      name="cn.jpush.android.intent.NOTIFICATION_OPENED" />
        <!-- Optional 用户接受Rich Push Javascript 回调函数的intent -->
        <action      name="cn.jpush.android.intent.ACTION_RICH PUSH_CALLBACK" />
        <!-- 接收网络变化 连接/断开 since 1.6.3 -->
        <action      name="cn.jpush.android.intent.CONNECTION" />

        <category     name="您应用的包名" />
    </intent-filter>
</receiver>

<!-- Required. For publish channel feature -->
<!-- JPUSH_CHANNEL 是为了方便开发者统计APK分发渠道。 -->
<!-- 例如： -->
<!-- 发到 Google Play 的APK可以设置为 google-play; -->
<!-- 发到其他市场的 APK 可以设置为 xxx-market。 -->
<!-- 目前这个渠道统计功能的报表还未开放。 -->
<meta-data
        name="JPUSH_CHANNEL"
        value="developer-default" />
<!-- Required. AppKey copied from Portal -->
<meta-data
        name="JPUSH_APPKEY"
        value="您应用的Appkey" />

<!-- =====极光推送结束===== -->

</application>
</manifest>

```

2.3.自定义广播接收器

通过广播可以获取到RegistrationId，表示集成成功了。

```
public class JPushReceiver extends BroadcastReceiver {

    private static final String TAG = "JPush";

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();
        Log.e(TAG, "[JPushReceiver] onReceive - " + intent.getAction() + ", extras: " + p
rintBundle(bundle));

        if (JPushInterface.ACTION_REGISTRATION_ID.equals(intent.getAction())) {
            String regId = bundle.getString(JPushInterface.EXTRA_REGISTRATION_ID);
            ACache.get(context).put(JPushInterface.EXTRA_REGISTRATION_ID, regId);
            Log.d(TAG, "[JPushReceiver] 接收Registration Id : " + regId);
            //send the Registration Id to your server...

        } else if (JPushInterface.ACTION_MESSAGE_RECEIVED.equals(intent.getAction())) {
            processCustomMessage(context, bundle);
        } else if (JPushInterface.ACTION_NOTIFICATION_RECEIVED.equals(intent.getAction()))
        {
            Log.d(TAG, "[JPushReceiver] 接收到推送下来的通知");
            int notificationId = bundle.getInt(JPushInterface.EXTRA_NOTIFICATION_ID);
            Log.d(TAG, "[JPushReceiver] 接收到推送下来的通知的ID: " + notificationId);
        } else {
            .....
        }
    }

    .....
}
```

2.4.启动极光推送服务

```
//初始化极光推送服务 百度推送不能在application里面初始化 极光推送可以
HetePushManager.getInstance().init(new JpushService(this));
```

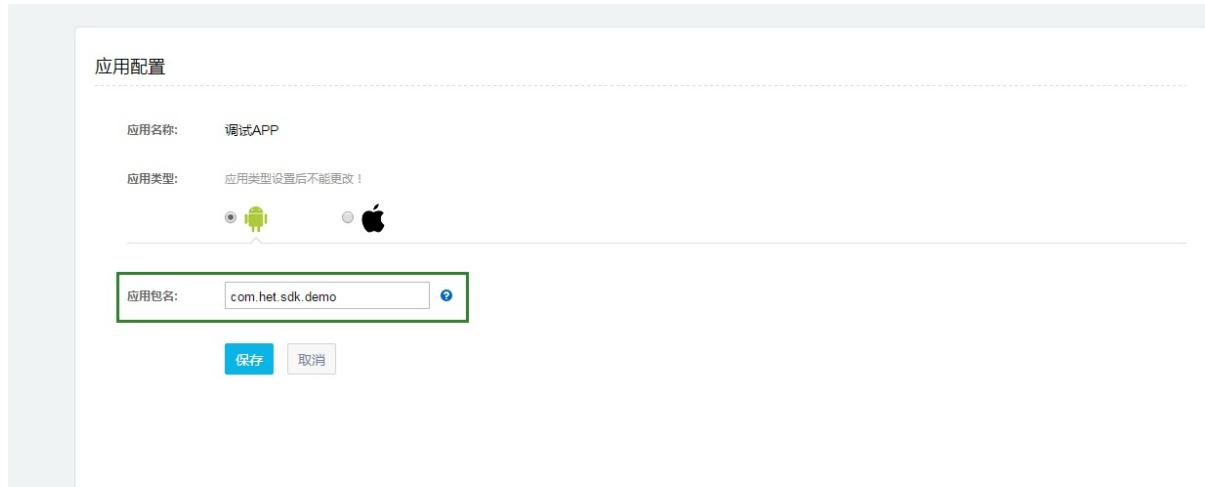
2.5.停止极光推送服务

```
HetePushManager.getInstance().stopPush();
```

快速集成百度推送

1.百度推送集成准备

1.1.申请百度开发者账号，创建应用



1.2.Android 推送配置，获取API KEY和SECRET KEY

应用名称	调试APP	APP ID	10359454
应用包名	com.het.sdk.demo	API KEY	UHILEYi6bX9yV9mGZjnvyzKL
创建时间	2017-11-14 11:17:47	SECRET KEY	T1v9xxFxzjCxOPeN1hwqtIH9GBz1xfo
更新时间	2017-11-14 11:17:47	二维码	

配置流程

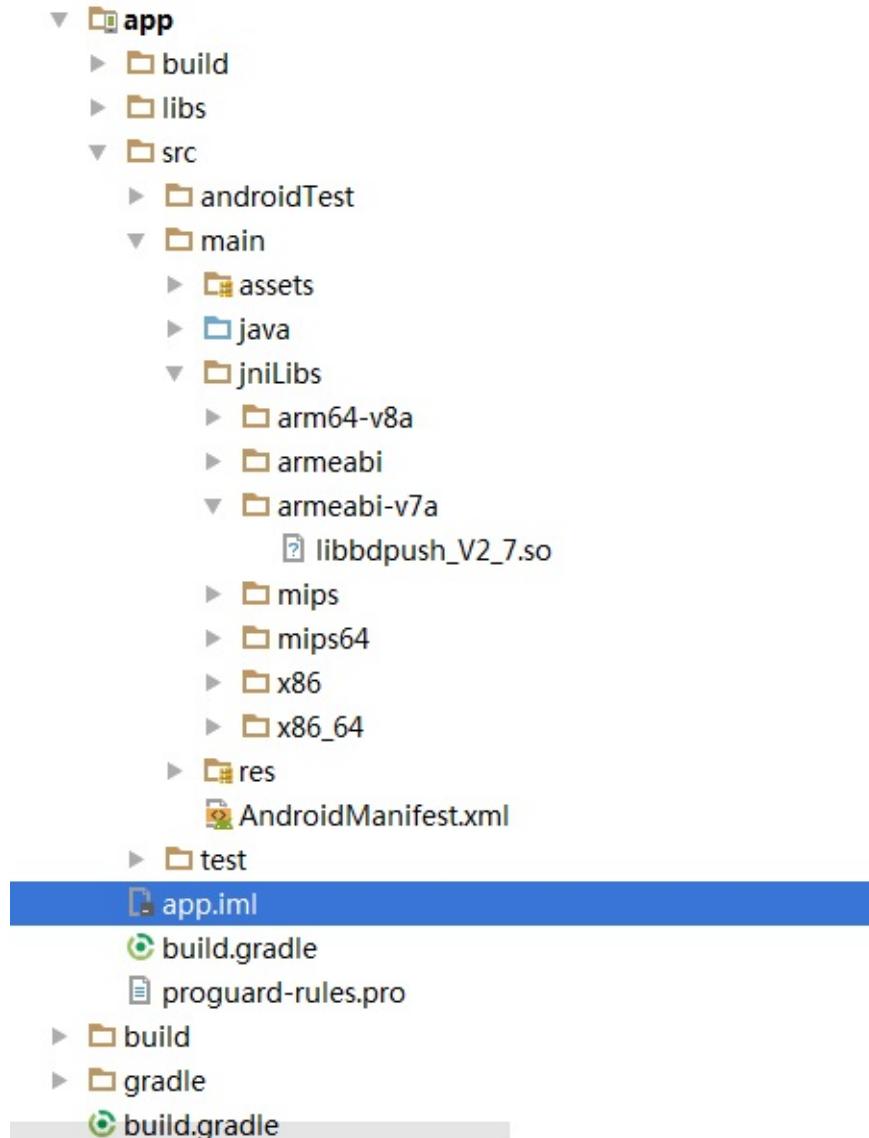
- 1 创建应用
- 2 下载SDK
- 3 应用程序嵌入百度云推送SDK安装包，参考[开发文档](#)
- 4 APP安装用于测试的手机上，获取测试机channel ID，进行验证测试
- 5 确定配置完成，可以发布应用，进行消息推送

2.APP集成百度推送

2.1.下载最新的Android SDK压缩包并解压，在新建工程或已有工程中增加百度云推送功能。注意：如果您的Android工程使用的是Android API level 21及以上的版本，您的通知图标背景必须是透明的，否则在Android5.0及以上的机器上通知图标可能会变成白色的方块。

2.2.导入jar包和so库文件（Android Studio 开发环境）

在工程中“src/main”目录中新建名为jniLibs的目录。将解压后的libs文件夹中所有文件拷贝到您的工程的jniLibs文件夹中如果您的工程中没有其他的.so文件，建议只拷贝armeabi文件夹。如果您的工程中还使用了其他的.so文件，只需要拷贝对应目录下的.so文件即可。



2.3.配置AndroidManifest文件

在当前工程的AndroidManifest.xml文件中，添加权限和声明信息：

```

<!-- Push service 运行需要的权限 -->
<uses-permission      name="android.permission.INTERNET" />
<uses-permission      name="android.permission.READ_PHONE_STATE" />
<uses-permission      name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission      name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission      name="android.permission.WRITE_SETTINGS" />
<uses-permission      name="android.permission.VIBRATE" />
<uses-permission      name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission      name="android.permission.DISABLE_KEYGUARD" />
<uses-permission      name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission      name="android.permission.ACCESS_WIFI_STATE" />
<!-- 富媒体需要声明的权限 -->
<uses-permission      name="android.permission.ACCESS_DOWNLOAD_MANAGER"/>
<uses-permission      name="android.permission.DOWNLOAD_WITHOUT_NOTIFICATION" />
<uses-permission      name="android.permission.EXPAND_STATUS_BAR" />

<!-- 适配Android N系统必需的ContentProvider写权限声明，写权限包含应用包名-->

```

```

<uses-permission           name="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName"
" />
<permission
    name="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName"
    protectionLevel="signature">
</permission>

<!-- push应用定义消息receiver声明 -->
<receiver          name="YourPackageName.BaiduPushReceiver">
    <intent-filter>

        <!-- 接收push消息 -->
        <action      name="com.baidu.android.pushservice.action.MESSAGE" />
        <!-- 接收bind,unbind,fetch,delete等反馈消息 -->
        <action      name="com.baidu.android.pushservice.action.RECEIVE" />
        <action      name="com.baidu.android.pushservice.action.notification.CLICK" />
    </intent-filter>
</receiver>

<!-- push service start -->
<!-- 用于接收系统消息以保证PushService正常运行 -->
<receiver          name="com.baidu.android.pushservice.PushServiceReceiver"
    process=":bdservice_v1" >
    <intent-filter>
        <action      name="android.intent.action.BOOT_COMPLETED" />
        <action      name="android.net.conn.CONNECTIVITY_CHANGE" />
        <action      name="com.baidu.android.pushservice.action.notification.SHOW" />
        <action      name="com.baidu.android.pushservice.action.media.CLICK" />
        <!-- 以下四项为可选的action声明, 可大大提高service存活率和消息到达速度 -->
        <action      name="android.intent.action.MEDIA_MOUNTED" />
        <action      name="android.intent.action.USER_PRESENT" />
        <action      name="android.intent.action.ACTION_POWER_CONNECTED" />
        <action      name="android.intent.action.ACTION_POWER_DISCONNECTED" />
    </intent-filter>
</receiver>
<!-- Push服务接收客户端发送的各种请求-->
<receiver          name="com.baidu.android.pushservice.RegistrationReceiver"
    process=":bdservice_v1" >
    <intent-filter>
        <action      name="com.baidu.android.pushservice.action.METHOD" />
        <action      name="com.baidu.android.pushservice.action.BIND_SYNC" />
    </intent-filter>
    <intent-filter>
        <action      name="android.intent.action.PACKAGE_REMOVED" />
        <data        scheme="package" />
    </intent-filter>
</receiver>
<service           name="com.baidu.android.pushservice.PushService"          exported="true"
    process=":bdservice_v1" >
    <intent-filter >
        <action      name="com.baidu.android.pushservice.action.PUSH_SERVICE" />
    </intent-filter>
</service>

<!-- 4.4版本新增的CommandService声明, 提升小米和魅族手机上的实际推送到达率 -->
<service           name="com.baidu.android.pushservice.CommandService"
    exported="true" />

<!-- 适配Android N系统必需的ContentProvider声明, 写权限包含应用包名-->
<provider

```

```

    android:name="com.baidu.android.pushservice.PushInfoProvider"
    android:authorities="YourPackageName.bdpush"
    android:writePermission="baidu.push.permission.WRITE_PUSHINFOPROVIDER.YourPackageName"
    android:protectionLevel = "signature"
    android:exported="true" />

    <!-- push结束 -->

```

2.4. 创建百度推送广播接收器

```

public class BaiduPushReceiver extends PushMessageReceiver {

    public static final String TAG = BaiduPushReceiver.class
        .getSimpleName();
    public static String Baidu_ChannelId;

    @Override
    public void onBind(Context context, int errorCode, String appid,
                      String userId, String channelId, String requestId) {
        String responseString = "onBind errorCode=" + errorCode + " appid="
            + appid + " userId=" + userId + " channelId=" + channelId
            + " requestId=" + requestId;
        Log.d(TAG, responseString);

        if (errorCode == 0) {
            // 绑定成功
            Log.d(TAG, "绑定成功");
            Baidu_ChannelId = channelId;
        }
    }
}

```

2.5. 启动百度推送服务

```

//初始化百度推送服务 不能在application 初始化
//在主 Activity 的 OnCreate 方法中，调用接口 startWork， 其中 loginValue 是 apiKey。 (注意：不
//要在 Application 的 onCreate 里去做 startWork 的操作，否则可能造成应用循环重启的问题，
//将严重影响应用的功能和性能。)
HetPushManager.getInstance().init(new BdPushService(this));

```

2.6. 停止百度推送服务

```
HetPushManager.getInstance().stop();
```

2.7. 错误码说明

error_code	描述
0	绑定成功
10001	当前网络不可用，请检查网络

10002	服务不可用，连接server失败
10003	服务不可用，503错误
10101	应用集成方式错误，请检查各项声明和权限
20001	未知错误
30600	服务内部错误
30601	非法函数请求，请检查您的请求内容
30602	请求参数错误，请检查您的参数
30603	非法构造请求，服务端验证失败
30605	请求的数据在服务端不存在
30608	绑定关系不存在或未找到
30609	一个百度账户绑定设备超出个数限制(多台设备登录同一个百度账户)
30612	百度账户绑定应用时被禁止，需要白名单授权

Copyright ©2017 clife - 深圳和而泰家居在线网络科技有限公司 粤ICP备14057188号-1 该文件修订时间： 2017-11-14 15:42:15