Rocke Xu

MXCHIP Co., Ltd

August 2015

MiCO - IoT 智能设备_APP 开发向导

(V1.0)

摘要（Abstract）

本文档主要介绍 MiCO 物联智能（IoT）设备开发流程，详细阐述设备接入 FogCloud 的过程，及接入后，如何实现“APP”对“设备”的控制。目的是让开发者对 MiCO 物联智能（IoT）设备开发有初步认识和理解，帮助开发者利用 MiCO 物联网操作系统，手机 APP SDK 和 FogCloud 庆科云进行物联网应用开发。

文档状态（Status of This Document）

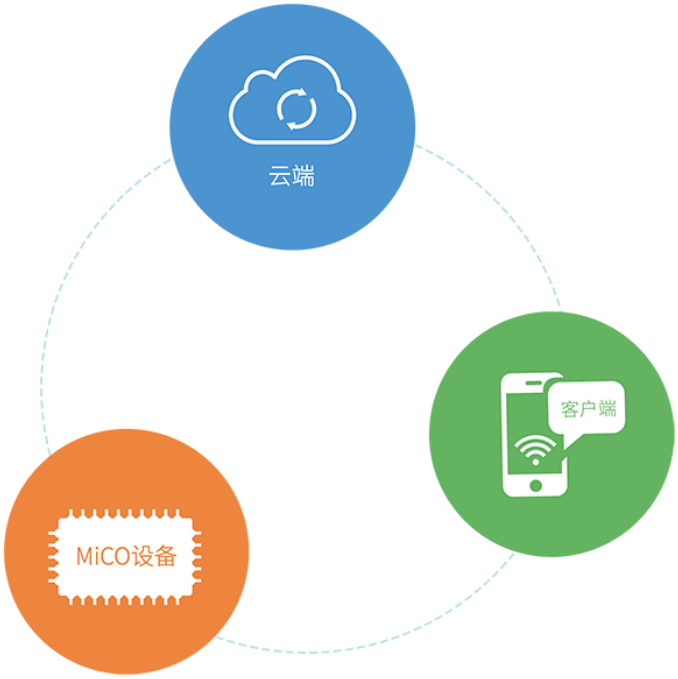
对 MiCO-IoT 物联网开发者公开

目 录

1. 概述

1.1. 简介

上海庆科为开发者提供一整套可用于智能设备开发的基础组件和实现方案，庆科物联系统组成，如下图：



庆科物联系统组成

运行基于 MiCO 开发的嵌入式设备应用程序，实现了互联网接入的智能设备 - MiCO 设备，以下简称“设备“;

可接入互联网的控制终端，通常是运行在移动设备上的有完整用户界面的应用程序–用户控制终端，如手机 APP 客户端，以下简称“APP”；

由庆科信息技术有限公司开发的为 MiCO 设备和控制终端提供接入和数据

交换服务，及其相关的云计算功能的网络服务集合- “FogCloud”。

本文主要介绍 MiCO 物联智能（IoT）设备 APP 的开发步骤及相关功能实现原理，让开发者能轻松上手，快速进行更深入的二次开发工作，尽快实现产品化生产。

1.2. 名词解释

token：资源操作凭证，由“FogCloud”分配。向“FogCloud 端请求时提交，如“user_token”，“dev_token”；

Ssid：手机 APP 所连接的 WIFI 的名称；

APPID：FogCloud 上注册的 app 的唯一标识；

（云）：此接口是与云进行通讯；

（设备）：此接口是与设备进行通讯。

1.3. 兼容性

支持 Android 4.0，iOS8.0 及以上系统

2. 准备工作

2.1. APP 管理

要使用云服务，需要先建立你的智能硬件产品及对应 APP。智能硬件产品 FAE 或者固件工程师会在开发固件时候完成。APP 需要前端开发者自己创建。

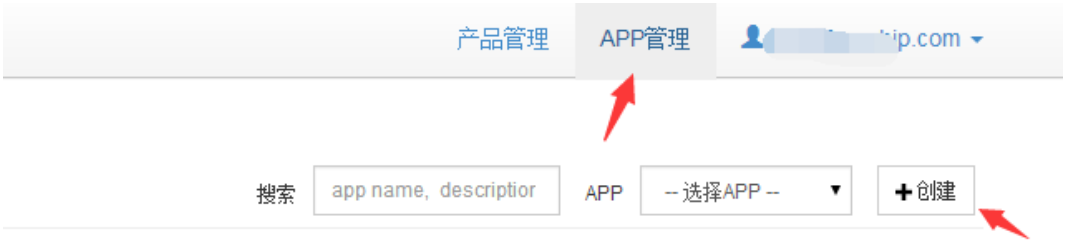
2.1.1. 注册开发者

首先打开地址 http://easylink.io，在右上角点击注册按钮，会打开开发者注册的界面，如下图，

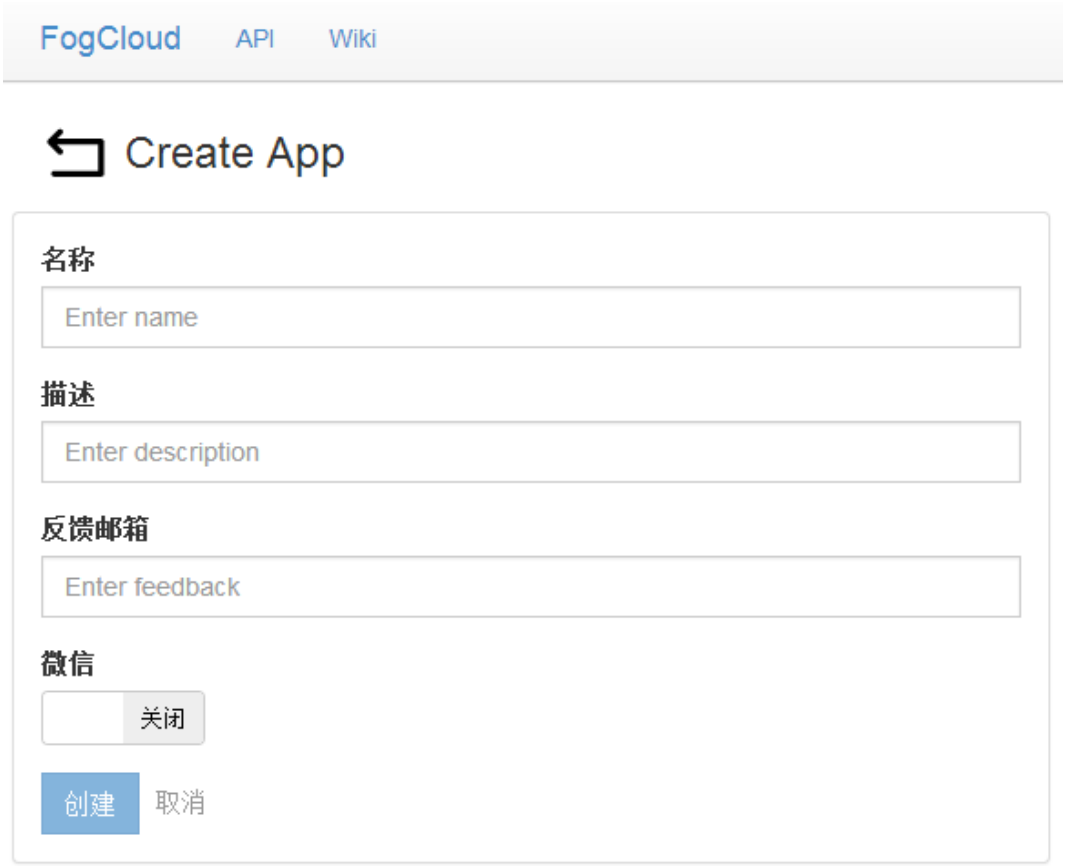
开发者通过邮箱注册开发者账号并登录系统。

2.1.2. **创建 APP**

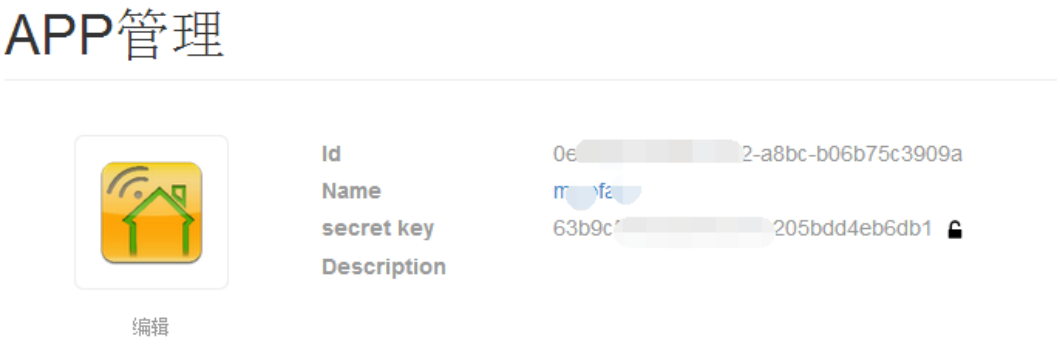
开发者登录成功之后，在主页点击右上角的 App 管理进入对应的控制台，如下图在这里你可以创建并管理你的 App，



点击创建，打开如下界面，填写信息后点击创建，即完成 APP 的创建。如下图

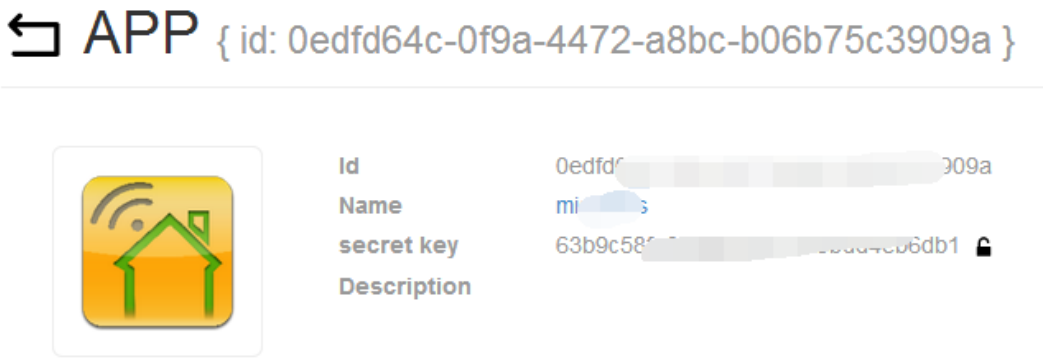


填写创建信息



APP 管理列表

点击 APP 名字后会打开 APP 管理的界面如下图，其中 ID 和 secret key 会在以后的开发中用到。



2.2. **用户管理**

用户管理分为邮箱用户和手机用户两种，其中手机用户注册需要用到手机验证码，手机验证短信目前仅支持云通讯，需要现在云通讯注册开发者账号（需要企业用户），然后再绑定到 Fogcloud 中，邮箱注册则可以直接使用 Fogcloud 的免费服务。

2.2.1. **手机验证**

目前云通讯的短信验证服务需要添加短信模版，而短信模版的要求是充值达到 500 的企业用户，如下图

2.2.1.1. **注册云通讯**

可参考云通讯的官方教程进行开发者注册：

http://docs.yuntongxun.com/index.php/新手指引。

2.2.1.2. **创建应用**

注册成功后登录云通讯平台，点击管理→创建应用，

创建新的应用，把所有功能都勾上（目前只用到短信验证码），创建成功后会在应用列表中看到已经创建的应用，如下图。

2.2.1.3. **上线设备**

点击上图的上线按钮，会提醒开发者实名认证，需要提供手持身份证的照片，提交后代云通讯官方审核通过（一般 2 小时之内）。

审核成功后上线设备，并创建短信模版。

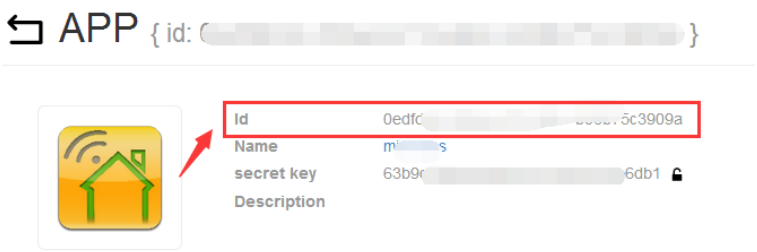
2.2.1.4. **绑定云通讯与 FogCloud**

完成以上操作后，将云通讯的信息与 Fogcloud 绑定，打开以下地址并登录 http://api.easylink.io/docs/#/!/apps-v2/update_put，

在顶部的 Sign 中输入专用签名“**942673c09ce9585c877db1e59331b967, 1440754176**”，APPID 不需要选择。

完成以下信息的填写，必填项为：

1）app_id：如下



2）name：app 的名称

3）sms_app_id：云通讯的应用管理→应用列表→相应应用的 APPID

4）sms_temp_id：短信模版的 ID

5）sms_accout_sid：云通讯开发者的 SID

6）sms_auth_token：云通讯开发者的 Token

如下图填写相关信息后点击“Try it out”，完成绑定，如果返回 success，说明绑定成功。

2.2.2. **邮箱验证**

邮箱注册使用的是 Fogcloud 的免费服务，相对比较简单，不需要这么多流程

3. 开发指南

3.1. Hello

与云端的通讯均通过 HTTP 的方式发送请求，请求代码参考如下

3.1.1. Android 开发

```
JSONObject postParam = new JSONObject();
try {
	postParam.put("login_id", "888888@qq.com");
	postParam.put("password", "111111");
} catch (JSONException e) {
	e.printStackTrace();
}
// 加载json数据
HttpUtils http = new HttpUtils();
http.configTimeout(15000);
RequestParams params = new RequestParams();
params.addHeader("Content-Type", "application/json");
params.addHeader("X-Application-Id", APP_ID);
params.addHeader("X-Request-Sign", REQUESTSIGN);
//params.addHeader("Authorization", "token " + userToken);
try {
	params.setBodyEntity(new StringEntity(postParam.toString(), "UTF-8"));
} catch (UnsupportedEncodingException e) {
	e.printStackTrace();
}
// 使用HttpUtils提交数据到服务器（post方法）
http.send(HttpRequest.HttpMethod.POST,
"http://api.easylink.io/v1/user/login.json", params,
new RequestCallBack<String>() {
	@Override
	public void onLoading(long total, long current,
	boolean isUploading) {
	}
	// 加载成功，开始解析
	@Override
```

```
        public void onSuccess(ResponseInfo<String> responseInfo) {
        }
        @Override
        public void onStart() {
        }
        // 加载失败，开始解析
        @Override
        public void onFailure(HttpException error, String msg) {
        }
});
```

3.1.2. iOS 开发

待补充

3.2. 用户注册（云）

用户注册 Fogcloud 账号，分为邮箱注册和手机注册，完成 2.2 用户管理的部分后，可以使用此功能。

3.2.1. 手机注册

通过手机注册用户，需要获取验证码并验证码才算真实身份，然后才能完成注册。

3.2.1.1. 获取验证码

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v2/users/sms_verification_code

参数说明：

参数	是否必须	说明
username	是	用户名
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回获取成功：

```
{"result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 500, "message": " Internal Server Error: 短信服务器错误:112307-【短信】模板短信模板 ID 为空"}}
```

3.2.1.2. **验证注册**

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v2/users

参数说明：

参数	是否必须	说明
username	是	用户名
password	是	账号密码
verification_code	是	收到的短信验证码
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回用户的 token：

```
{ "token": "4ced38c1-XXXX-XXXX-XXXX-2d2738696c87"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.2.2. **邮箱注册**

通过邮箱注册的用户，也需要获取验证码并验证码才算真实身份，然后才能完成注册。

3.2.2.1. **获取验证码**

接口调用请求说明：

HTTP 请求方式：POST http://easylink.io/v2/users/email_verification_code

参数说明：

参数	是否必须	说明
username	是	用户名
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回获取成功：

```
{"result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": "Forbidden: username is illegal"}}
```

3.2.2.2. **验证注册**

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v2/users

参数说明：

参数	是否必须	说明
username	是	用户名
password	是	账号密码
verification_code	是	邮箱收到的验证码
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp>

		APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回用户的 token：

```
{ "token": "4ced38c1-XXXX-XXXX-XXXX-2d2738696c87"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.3. **用户登录（云）**

已经注册成功的用户 ,可以进行登录 ,每次用户登录都会返回此用户的 token，设备应保留此 token 用于进行后续操作。

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v2/users/login

参数说明：

参数	是否必须	说明
username	是	用户名
password	是	账号密码
Headers 中的内容		
Content-Type	是	application/json

X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回用户的 token：

```
{ " user_token": "4ced38c1-XXXX-XXXX-XXXX-2d2738696c87"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.4. **修改密码（云）**

用户需要修改密码时候可以调用以下接口：

接口调用请求说明：

HTTP 请求方式：PUT

http://easylink.io/v2/users/password

参数说明：

参数	是否必须	说明
username	是	用户名

password	是	账号密码
new_password	是	账号新密码
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回修改成功：

```
{ "result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.5. 重置密码（云）

忘记密码的用户可以调用此接口来进行密码重置，同样必须先获取验证码，验证通过后才能设置新的密码

3.5.1. **获取验证码**

接口调用请求说明：

HTTP 请求方式：POST 邮箱验证码：http://easylink.io/v2/users/email_verification_code 短信验证码：http://easylink.io/v2/users/sms_verification_code

参数说明：

参数	是否必须	说明
username	是	用户名
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回成功获取：

```
{ "result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.5.2. **重置密码**

重置密码时候需要验证 3.5.1 中获取到的验证码，无论是手机验证码还是邮箱验证码。

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v2/users/password/reset

参数说明：

参数	是否必须	说明
username	是	用户名
password	是	账号新密码
verification_code	是	邮箱或者手机获取的验证码
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret +

		timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回修改成功：

```
{ "result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 400, "message": " Bad Request: fail to verify "}}
```

3.6. **设备联网（设备）**

新设备需要连上云端才能注册和被控制，所以需要将 WIFI 的名称和密码发给设备，让设备能连上路由器，从而连上云端。发送的数据包（SSID/密码）已广播形式发送，我们称之为 EasyLink。

设备会收到 APP 发送过来的数据包，解析后自动根据 SSID 和密码连上路由器，并自动连接上 APP，给 APP 发送设备的基本信息。

3.6.1. **Android 系统配网**

首先需要导入三个包：

1、 easylink_ftc_out.jar

2、 jetty-8.1.15.jar

3、 dd-plist.jar

配网操作分为三步：

1）获取 ssid

```
private EasyLinkWifiManager mWifiManager = null;
private Context ctx = null;
ctx = MainActivity.this;
mWifiManager = new EasyLinkWifiManager(ctx);
String Ssid = mWifiManager.getCurrentSSID();
```

2）发送数据包（SSID，password）

```
public EasyLinkAPI elapi;
elapi.startFTC(wifissid,    wifipsw, new FTCListener() {
        @Override
        public void onFTCfinished(String ip,
                String jsonString) {
            Log.d("FTCEnd", ip + " " + jsonString);
            //需要自行关闭
            elapi.stopEasyLink();
        }
        @Override
        public void isSmallMTU(int MTU) {
        }
    });
```

3）得到设备 ip

如上代码，onFTCfinished 中返回的 ip 即设备的 IP。

3.6.2. **iOS 系统配网**

首先需要导入库：

待补充

配网操作分为三步：

1）获取 ssid

2）发送数据包（SSID，password）

3）得到设备 ip

如上代码，onFTCfinished 中返回的 ip 即设备的 IP。

3.7. **设备激活（设备）**

APP 拿到设备的 ip 后，通过 HTTP 的方式连接上设备，给设备发送激活请求

接口调用请求说明：

HTTP 请求方式：POST http://设备的 IP:8000/dev-activate

参数说明：

参数	是否必须	说明
login_id	是	设备用户，默认”admin”
dev_passwd	是	设备密码：默认“1234”
user_token	是	激活密钥：<md5(Fogcloud 用户登录得到的 token + timestamp)> 如：0f08007b1b775b216d7740ca14daa8e8
Headers 中的内容		
Content-Type	是	application/json

返回说明：

正常情况下成功注册返回设备的 deviceid：

```
{ "device_id": "af2b33be/c8934645dd0a" }
```

3.8. 绑定设备（云）

拿到 deviceid 后说明设备在云端注册成功，此时 APP 只需再将用户和设备在云端绑定即可：

接口调用请求说明：

HTTP 请求方式：POST

http://api.easylink.io/v1/key/authorize

参数说明：

参数	是否必须	说明
active_token	是	3.6 设备激活时候发送给设备的“user_token”
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret +

		timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回设备的 key，但是此 key 暂时并不会用到：

```
{"user-device-key": "e8e2f038-XXXX-XXXX-XXXX-3d38d562958a"}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": " Forbidden: active_token is not found!"}}
```

3.9. 获取设备列表（云）

想获取此用户名下有哪些可控制的设备时候，可以调用此接口，将会返回此用户当前所有设备的列表。

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v1/device/devices

参数说明：

Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id

		（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回设备列表的 json 数组：


```
[ {
      "id": "d64fXXXX/c8XXXXXX13c",
      "serial": null,
      "bssid": " c8XXXXXX13c",
      "created": "2015-08-29 17:12:09",
      "alias": "我的",
      "online": "1",
      "power_time": null,
      "ip": "223.166.139.208",
      "ssid": null,
      "online_time": "2015-08-29 17:00:42",
      "offline_time": "2015-08-28 18:40:57"
   },{
      "id": "d64fXXXX/c8XXXXXX13c",
      "serial": null,
      "bssid": " c8XXXXXX13c",
      "created": "2015-08-29 17:12:09",
      "alias": "我的",
      "online": "1",
      "power_time": null,
```

```
    "ip": "223.166.139.208",
    "ssid": null,
    "online_time": "2015-08-29 17:00:42",
    "offline_time": "2015-08-28 18:40:57"
}]
```


关键参数说明

参数	说明
Id	设备的 deviceID
Bssid	设备的 mac 地址
Alias	设备的名称
Online	设备是否在线：1 在线 0 不在线

错误时，返回错误原因如：


```
{"error": { "code": 403, "message": " Forbidden: active_token is not found!"}}
```


3.10.**修改设备信息（云）**

通过 3.8 获取设备列表，可以拿到设备的 deviceid，以后对设备的任何操作均需要设备的 deviceid。

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v1/device/modify

参数说明：

参数	是否必须	说明

device_id	是	设备的 deviceid
alias	是	设备的新名字
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下返回修改成功：

```
{ "result": "success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": " Forbidden: "}}
```

3.11.**授权设备（云）**

如果对于设备 A 而言，我的 owner，我便可以将设备 A，授权给别人来控制，这时候需要知道设备的 deviceid 和对方的账号（手机号或者邮箱），然后调用此接口：

接口调用请求说明：

HTTP 请求方式：POST http://easylink.io/v1/key/user/authorize

参数说明：

参数	是否必须	说明
login_id	是	被授权方的用户名（手机号/邮箱）
owner_type	是	授予什么权限（owner/share） Owner：可以控制设备，并可以把设备再次授权给别人 Share：可以控制设备
id	是	设备的 deviceid
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注

		意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功返回设备的 token：

```
{ "result": " d2a2abdd-XXXX-XXXX-XXXX-ad3e83f008be "}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": " Forbidden: "}}
```

3.12. **取消授权（云）**

我如果是设备 A 的 owner，我不想让某人（甲）再使用设备 A 了，我同样需要知道设备的 deviceid 和甲的账号（手机号或者邮箱），然后调用此接口：

接口调用请求说明：

HTTP 请求方式：POST

http://easylink.io/v1/device/user/delete

参数说明：

参数	是否必须	说明
device_id	是	设备的 deviceid

user_id	是	被授权方的用户名（手机号/邮箱）
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回设备的 key：

```
{ "result": " success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": " Forbidden: "}}
```

3.13.**解绑设备（云）**

假如我不想控制设备 A 了，我可以删除他：

接口调用请求说明：

HTTP 请求方式：POST http://easylink.io/v1/device/delete

参数说明：

参数	是否必须	说明
device_id	是	设备的 deviceid
Headers 中的内容		
Content-Type	是	application/json
X-Application-Id	是	Fogcloud 中创建 APP 时获得的应用唯一 id（APPID）
Authorization	是	"token " + userToken： 这里 userToken 为用户登录后返回的 token，注意 token 后面有个空格
X-Request-Sign	是	APP 请求签名 <md5(APP_Secret + timestamp), timestamp> APP_Secret 为 Fogcloud 中创建 APP 时获得的应用 secret key 如：0f08007b1b775b216d7740ca14daa8e8,1440830056

返回说明：

正常情况下成功注册返回设备的 key：

```
{ "result": " success"}
```

错误时，返回错误原因如：

```
{"error": { "code": 403, "message": " Forbidden: "}}
```

3.14.控制设备（MQTT）

好了，重点来了，目前 APP 控制设备都是使用的 IBM 开发的一个即时通讯协议 MQTT(该协议支持所有平台),来与设备进行通讯 ,比如获取设备的状态，发送指令控制设备等。

3.14.1. Android 系统 MQTT

Android 部分首先需要导入一个包：mqttservice2.jar，

接下来连接和控制设备主要分三步

1）建立连接并订阅

```
private static MqttServiceAPI mapi;
private Context ctx;
ctx = MainActivity.this;
mapi = new MqttServiceAPI(ctx);

String host = "api.easylink.io";
String port = "1883";
String userName = "admin";
String passWord = "admin";
String topic = "d6XXX17c/c8934XXXX13c/out/read";
String clientID = "v1-app-ro0222er99ss";
/**
* @param host 如api.easylink.io
 * @param port 如1883
 * @param userName 用户名：如admin
 * @param password 密码：如admin
 * @param Client-ID:v1_app_[MAC] //版本号_app_USERTOKEN(前12位必须小写)
```

```
 * @param topics
 */
mapi.startMqttService(host, port, userName, passWord, clientID,
          topic, new MqttServiceListener() {
        @Override
        public void onMqttReceiver(String msgType,
              String messages) {
         Log.d("---" + msgType + "---", messages);
        }
});
```

注：此接口的所有参数不能为空

2）发送指令

```
String topic = "b674706a/c8934691/in.json";
String command = "{}";
/**
     * @param topic 发送指令的通道
     * @param command 发送的指令
     * @param qos 服务质量：0
     * @param retained 是否在服务保存消息：false
     */
mapi.publishCommand(topic, command, 0, false);
```

3）断开连接

```
mapi.stopMqttService();
```

3.14.2. iOS 系统 MQTT

待补充

3.14.3. 返回说明

StartMqttService的时候就已经开启了消息接受的通道，即实现MqttServiceListener的接口，返回的消息均为String格式的字符串，其中

msgType为消息类型，目前有两种类型，status和payload，

1） status返回的是服务器的状态目前的状态类型有

a) Connected （连接成功）

b) Connect Exception （连接异常/失败）

c) Stop （断开连接）

d) Lost （连接丢失）

e) ReSubscribe success （再次连接后订阅成功）

f) Subscribe success （添加订阅成功）

g) UnSubscribe success （删除订阅成功）

h) Publish success （指令发送成功）

2） payload为云端或者mqtt服务器返回的字符串类型消息，如

“{"topic":"d643517c/c8934691333c/out/read","payload":{ "17": 30, "19": 35 }}”

4. 总结

上文介绍的APP开发主要是基于“MiCO系统”和上海庆科云 “FogCloud”开发的基本流程。

MiCO智能设备开发核心有两部分：

1）如何让设备接入FogCloud（EasyLink配网部分）。

2）如何通过FogCloud进行数据交互（MQTT协议部分）。

有关：设备SDK开发，FogCloud云开发，和APP SDK开发，更详细内容请参阅：

a. 庆科云开发者网站：http://easylink.io/#/ 的wiki中心。

b. MiCO开发者网站：http://mico.io/的wiki中心。

c. 开发者常见问题可在MiCO开发者网站MiCO论坛中提出。

另外，APP开发的所有资料和demo可以见以下地址：里面包含一些开源的项目，并且会及时更新最新的库和资料。

http://git.oschina.net/bringmehome/MiCOSDK

（完）