

MiCO - IoT 智能设备_APP 开发向导

APICloud 版 (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。

版权声明 (Copyright Notice)

Copyright (c) 2015 MWG Trust and the persons identified as the document authors. All rights reserved.

目 录

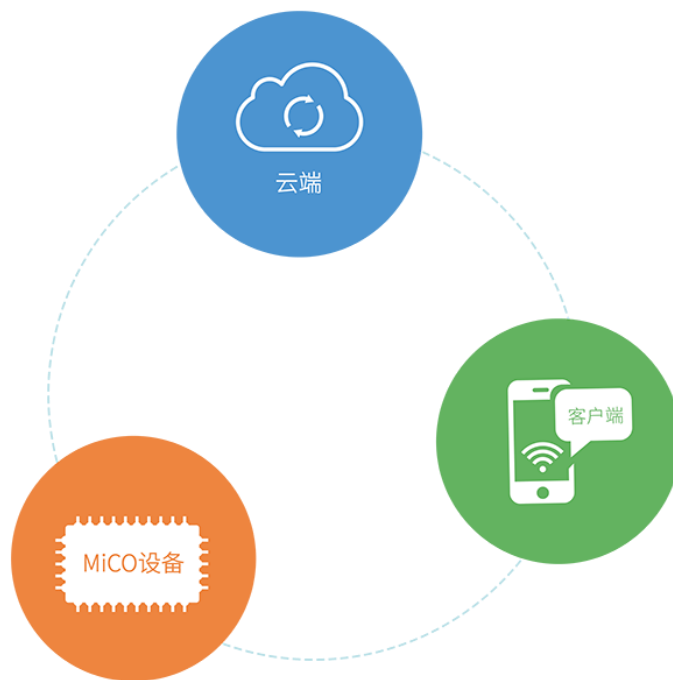
MiCO - IoT 智能设备_APP 开发向导	1 -
1. 概述	4 -
1.1. 简介	4 -
1.2. 名词解释	5 -
1.3. 兼容性	5 -
2. 准备工作	6 -
2.1. APP 管理.....	6 -
2.1.1. 注册开发者.....	6 -
2.1.2. 创建 APP	7 -
2.2. APICloud 引导	8 -
2.2.1. 注册并登录.....	8 -
2.2.2. 登录 IDE	9 -
2.2.3. 同步项目.....	9 -
2.2.4. 应用包结构.....	11 -
2.3. 环境搭建	11 -
3. 开发指南	12 -
3.1. 开发引导.....	12 -
3.2. 用户注册	14 -
3.2.1. 获取验证码.....	16 -
3.2.2. 验证验证码.....	17 -
3.2.3. 设置初始密码完成注册.....	17 -

3.3. 用户登录	- 18 -
3.4. 重置密码	- 19 -
3.4.1. 短信验证	- 19 -
3.4.2. 重置密码	- 19 -
3.5. 修改密码	- 20 -
3.6. 设备联网	- 21 -
3.6.1. 添加模块	- 21 -
3.6.2. 开始配网	- 23 -
3.6.3. 结束配网	- 24 -
3.7. 绑定设备	- 24 -
3.8. 获取设备列表	- 26 -
3.9. 修改设备信息	- 27 -
3.10. 获取 owner 设备列表	- 28 -
3.11. 授权设备	- 29 -
3.12. 获取此设备下的所有用户	- 30 -
3.13. 取消授权	- 31 -
3.14. 解绑设备	- 32 -
3.15. 控制设备 (MQTT)	- 32 -
3.15.1. 打开 MQTT	- 32 -
3.15.2. 发送指令	- 34 -
3.15.3. 关闭 MQTT	- 34 -
4. 总结	- 35 -

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；

APP_ID : FogCloud 上注册的 app 的唯一标识；

APP_NAME : FogCloud 上注册的 app 的名称；

1.3. 兼容性

支持 Android 4.0 , iOS8.0 及以上系统

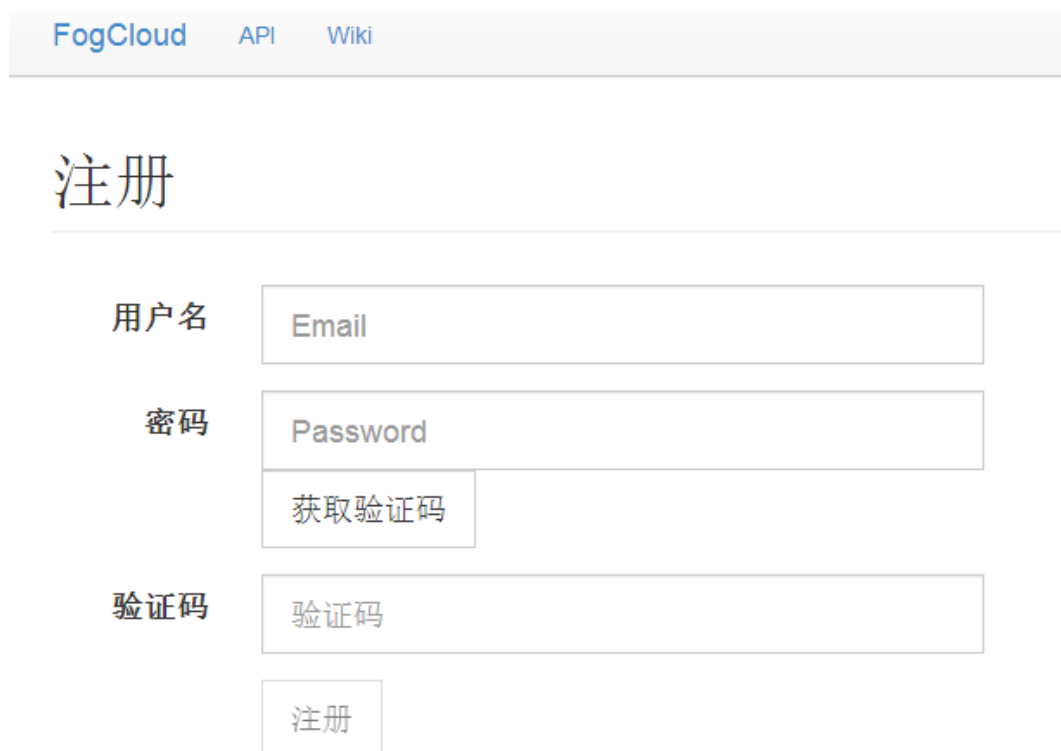
2. 准备工作

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，



FogCloud API Wiki

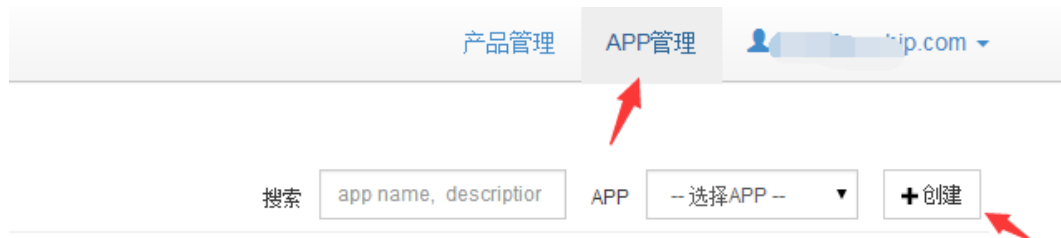
注册

用户名	<input type="text" value="Email"/>
密码	<input type="password" value="Password"/> <input type="button" value="获取验证码"/>
验证码	<input type="text" value="验证码"/>
	<input type="button" value="注册"/>

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。如下图所示

The screenshot shows the 'Create App' form in the FogCloud dashboard. The form has a title 'Create App' with a back arrow icon. It contains three input fields: '名称' (Name) with placeholder 'Enter name', '描述' (Description) with placeholder 'Enter description', and '反馈邮箱' (Feedback Email) with placeholder 'Enter feedback'. Below these fields is a '微信' (WeChat) section with a '关闭' (Close) button. At the bottom of the form are two buttons: '创建' (Create) and '取消' (Cancel).

填写创建信息

APP管理



编辑

Id	0e[REDACTED]2-a8bc-b06b75c3909a
Name	mi[REDACTED]
secret key	63b9c[REDACTED]205bdd4eb6db1 🔒
Description	

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。

🔙 APP { id: 0edfd64c-0f9a-4472-a8bc-b06b75c3909a }



Id	0edfd[REDACTED]909a
Name	mi[REDACTED]
secret key	63b9c58[REDACTED]205bdd4eb6db1 🔒
Description	

2.2. APICloud 引导

下载(“<http://www.apicloud.com/dev>”)并安装 APICloud IDE 开发环境 ,IDE 当前支持 Windows 系统。

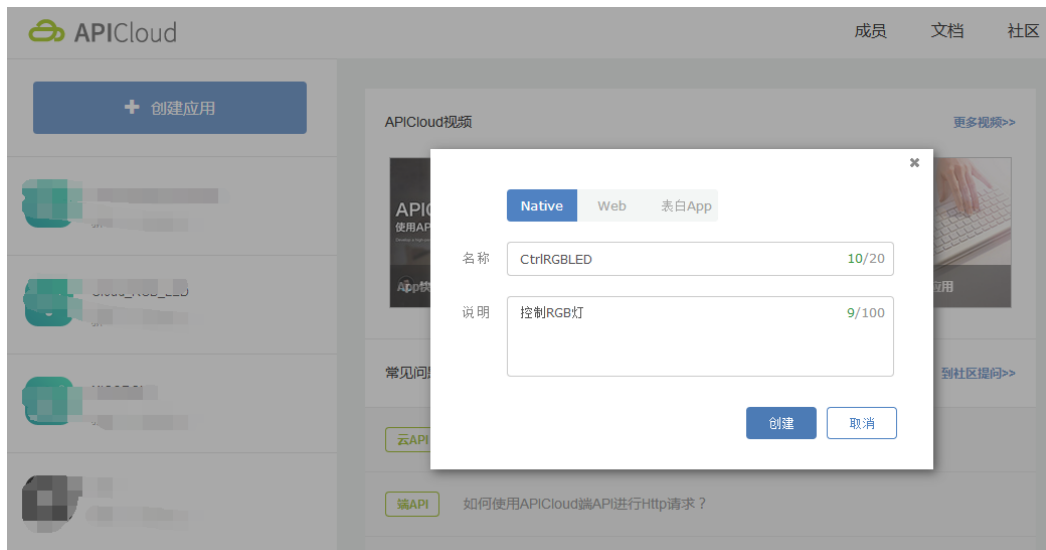
注：具体可以参考

<http://docs.apicloud.com/APICloud/creating-first-app#1>

2.2.1. 注册并登录

打开 <http://www.apicloud.com/>，注册并登录。

打开 <http://www.apicloud.com/console> 点击左上角“创建应用”，如图：选择“Native”，填写“名称”及“说明”，创建应用。



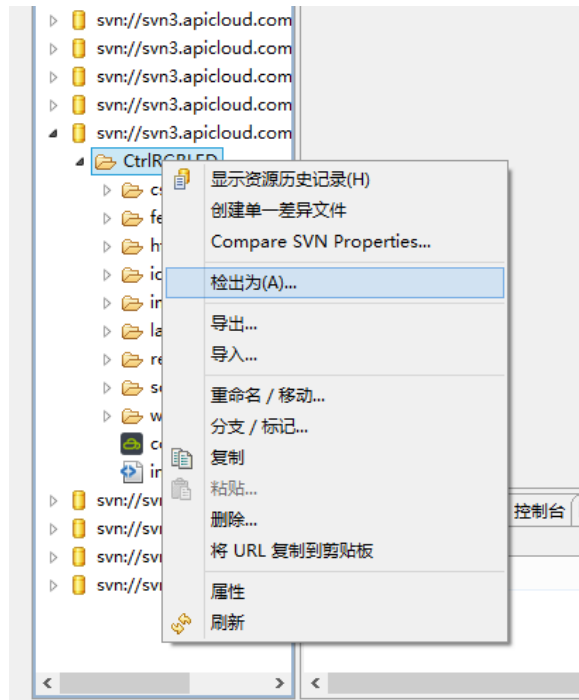
2.2.2. 登录 IDE

用以上注册的 APICloud 账号登录 IDE



2.2.3. 同步项目

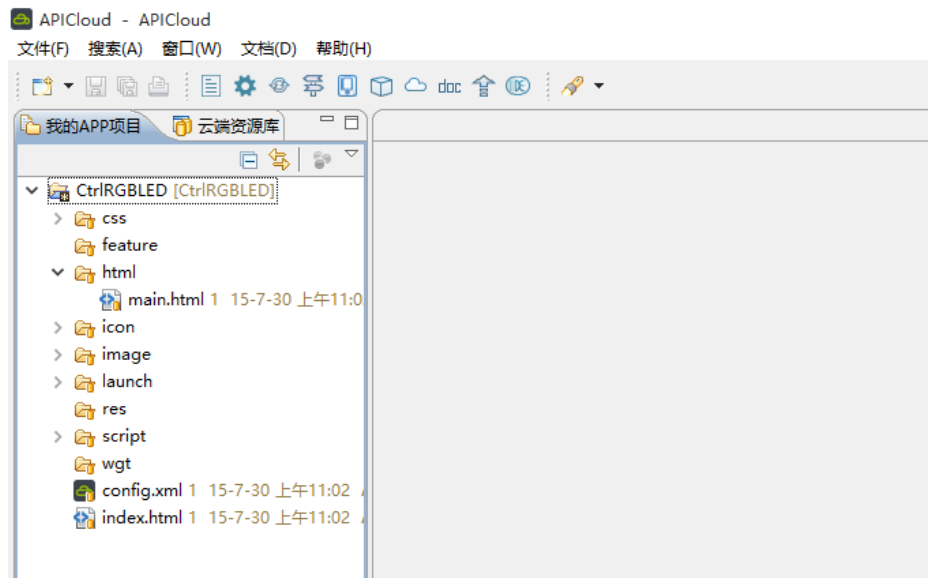
- 1) 登录后,左侧选择“云端资源库”,根据 APICloud 创建的应用 ID 选择 SVN 项目。
- 2) 选择项目,右键“检出为”



3) 点击“完成”，完成应用同步。



2.2.4. 应用包结构



“config.xml” 是配置文件，“index.html” 是启动页面，“icon” 为图标文件目录，“launch” 为启动图片目录（更多介绍详见 Widget 包结构说明文档）。

2.3. 环境搭建

将 git 中的 mxClean 项目下载到本地，拷贝项目中的以下 5 个 js 到工程的 script 目录：

- 1) av-0.5.0.min.js
- 2) config.js
- 3) mico-ac-0.2.js
- 4) jquery-2.1.3.js
- 5) jquery.md5.js

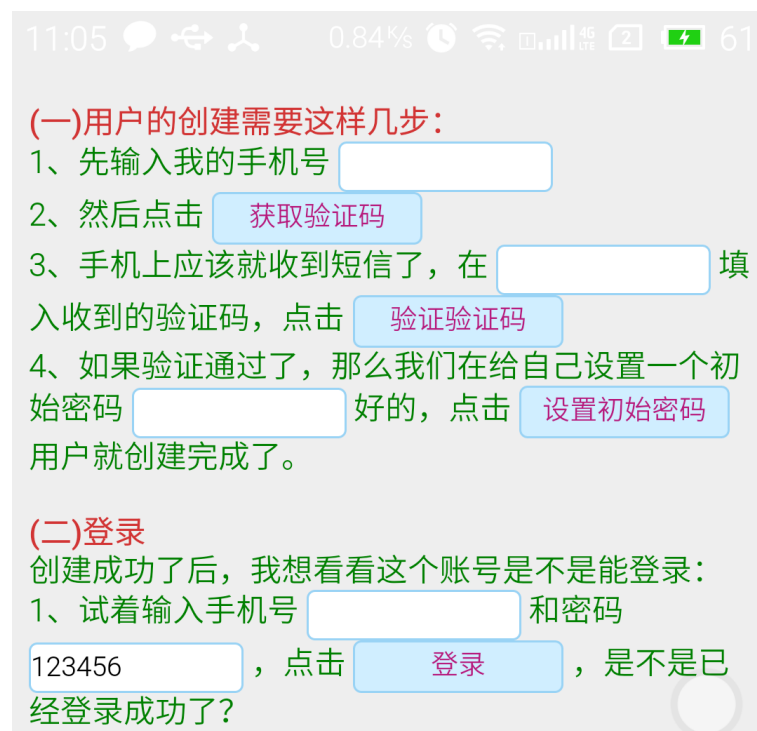
将 config.js 里的 APP_ID ,APP_NAME 改为 2.1.2 创建的 APP 的 id 和 name。

3. 开发指南

APICloud 开发不依赖于客户端系统，均使用 HTML，js 来实现逻辑，所以一套代码可以同时使用在 Android 系统和 iOS 系统下运行。以下的所有内容，均不涉及原生开发。

3.1. 开发引导

整个系统分为用户管理、设备管理、设备控制三个部分，测试 demo 中将这三个部分分为三个页面，以故事的形式贯穿了整个 APP 的开发流程。Demo 界面截图如下：



(三)密码

如果忘记密码了，

1、首先我要在创建用户的第一步，输入手机号，验证短信验证码，

2、验证通过后，在这里输入新密码

点击 **重置密码**，好的，密码重置成功。

3、如果我没有忘记密码，只是想修改下，我先输入老秘密

点击 **修改密码**，嗯嗯，不错，密码也修改好了。

好了，我要去玩设备了

用户管理

返回用户管理

首先我得先有个设备

(一)添加设备需要这样几步：

1、点击 **获取WIFI名称** 拿到WIFI的名称为

2、然后输入WIFI的密码 点击

开始配网，现在APP在发数据包（SSID和WIFI密码）

3、设备收到WIFI密码后，会自动连上路由器，并回连到手机，返回设备的IP地址，我们可以点击

绑定设备，这样设备和我就关联起来了。

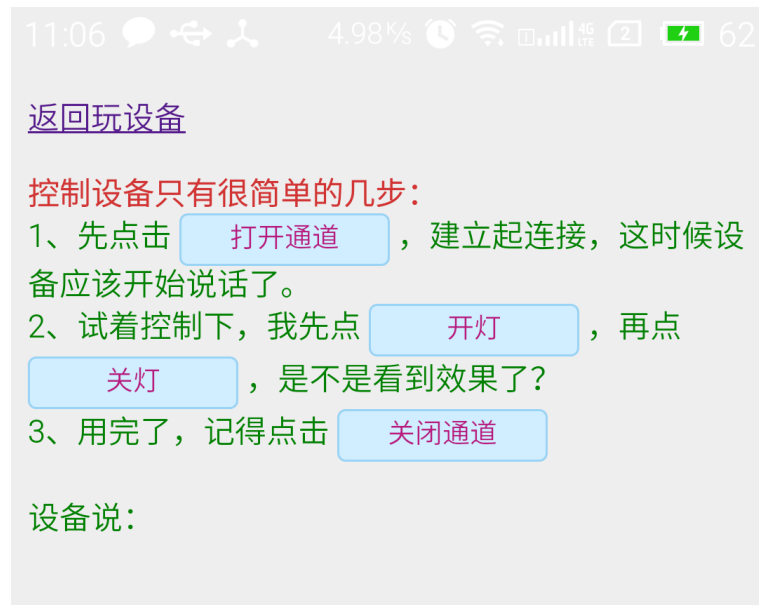
4、好了，赶紧点击 **停止配网**，否则网络要阻塞了。

这样，我们就有设备了。接下来看看怎么玩他。

(二)设备的管理功能非常多：

- 1、我们先点击 **获取设备列表**，看看我手上有多少设备了。
- 2、好像名字不好听，我输入 然后点击 **修改设备名称**，哈哈，好名字。
- 3、隔壁老王经常来我家，我想显摆下，我输入老王手机号 ，然后点击 **授权设备**，嗯，他也能用我家的冰箱了。
- 4、虽然我能控制很多设备，但是有些是别人的，我看看我自己到底有哪些家电，点击 **获取有权限设备**，哦，原来他们都是我的。
- 5、那么现在哪些人能用我的空调呢？我点击 **获取设备的用户**，哇，这么多人啊。
- 6、我不想让小明用，我输入他手机号 ，点击 **取消授权**。哼
- 7、我也不想用小明家的灯，我点击。**解绑设备**

设备管理

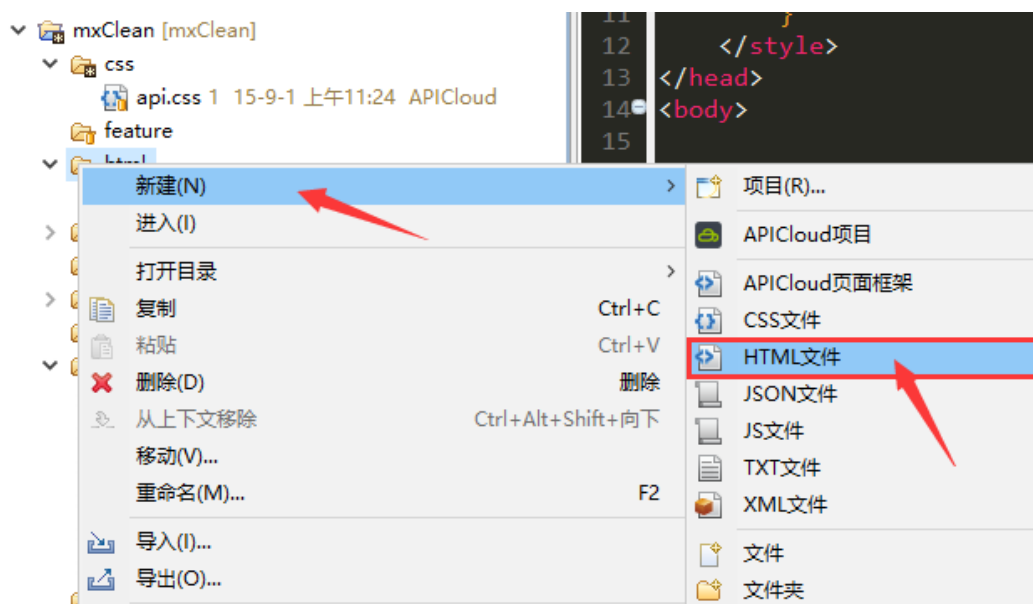


设备控制

开发中可以参考 demo，结合自己的实际情况进行开发。下面开始→

3.2. 用户注册

在 html 文件夹内，新建 HTML 文件，起名为 userManage.html，如下图



打开 userManage.html 文件，在 “</head>” 之前添加 js 文件，如下：

```
<script type="text/javascript" src="../../script/config.js"></script>
<script type="text/javascript" src="../../script/av-0.5.0.min.js"></script>
<script type="text/javascript" src="../../script/mico-user-0.1.js"></script>
<script type="text/javascript" src="../../script/jquery-2.1.3.js"></script>
```

打开根目录下的 index.html，将 `apiready = function() {};` 内的代码改为以下内容，

```
api.openWin({
  name : 'userManage.html',
  url : 'html/userManage.html',
  slidBackEnabled : false,
  reload : true,
  bgColor : '#F0F0F0',
  animation : {
    type : 'none'
  }
});
```

移动端的用户注册基本都是手机注册，所以这里只讲手机用户的注册（邮箱注册的也支持，需要可以联系技术支持），需要获取验证码，验证通过才能完成注册。

3.2.1. 获取验证码

打开 userManage.html 文件，在 `<script type="text/javascript">`
`</script>` 内添加获取短信验证码的函数 `getSmsCode()`，需要传入几个参数：

参数	是否必须	说明
mobilePhoneNumber	是	手机号
template	是	模版名称 "micodeveloper"，如果没有另外申请模版，请不用改
username	是	对收短信人的称呼，可以是尊称或者昵称
name	是	注册项目的名称，或者是 APP 的名称
appname	是	APP 的名称
ttn	是	失效时间，不超过 10 分钟

代码片段

```
function getSMS() {  
    var phone = '18623452345';  
    var template = "micodeveloper";  
    var username = "哥哥";  
    var name = 'MiCO总动员';  
    var appname = 'MiCO开发者';  
    var ttl = 10;  
    $micojs.getSmsCode(phone, template, username, name, appname,  
    ttl, function(ret, err) {  
        alert(JSON.stringify(ret));  
    });  
}
```

发送成功后会返回 `{"status": "success"}`;

手机收到的短信效果为：

Hi 哥哥，欢迎注册 MiCO 总动员，亲可以通过验证码:495629，进行注册。但是呢，这个验证码会在 10 分钟后自行失效。请尽快使用，么么哒！【MiCO 开发者】

3.2.2. 验证验证码

收到验证码后当然要验证了 验证前在 script 中添加函数 `checkSmsCode()`，

需要传入的参数：

参数	是否必须	说明
phone	是	手机号
Smscode	是	手机收到的验证码

代码片段

```
var phone = '18623452345';
var smscode = '336765';
$micojs.checkSmsCode(phone, smscode, function(ret, err) {
    alert(ret);
});
```

验证成功后会返回用户的大部分信息，json 格式如下

```
{
  "mobilePhoneNumber": "18623452345",
  "mobilePhoneVerified": true,
  "username": "18623452345",
  "objectId": "55e5XXXdddb2e44a9b89e3df",
  "createdAt": "2015-09-01T09:06:15.179Z",
  "updatedAt": "2015-09-01T09:06:15.179Z"
}
```

3.2.3. 设置初始密码完成注册

接下来就剩设置初始密码了，初始密码的函数 `setInitPwd()`，需要传入的

参数：

参数	是否必须	说明
phone	是	手机号
password	是	手机密码

代码片段

```
var phone = '18623452345';
var password = 'mxchip123456';
$micojs.setInitPwd(phone, password, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

验证成功后会返回用户的大部分信息，json 格式如下

```
{
  "login_id": "18623452345",
  "token": "ec8aed67-XXXX-XXXX-XXXX-27a5148b353c",
  "message": "注册成功!"
}
```

3.3. 用户登录

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大部分信息，并保留在 localStorage 里。

登录使用的函数 `loginWithPhone ()`，需要传入的参数：

参数	是否必须	说明
phone	是	手机号
password	是	登录密码

代码片段

```
var phone = '18623452345';
var password = 'mxchip123456';
$micojs.loginWithPhone (phone, password, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

验证成功后会返回用户的大部分信息，json 格式如下

```
{
  "mobilePhoneNumber": " 18623452345",
  "userToken": " ec8aed67-XXXX-XXXX-XXXX-27a5148b353c",
  "appname": "micofans",
  "username": " 18623452345",
  "emailVerified": false,
  "authData": null,
  "mobilePhoneVerified": true,
  "objectId": "55e5XXXdddb2e44a9b89e3df",
  "createdAt": "2015-09-01T09:06:15.179Z",
  "updatedAt": "2015-09-01T09:06:56.777Z"
}
```

如果密码不正确

```
{
  "code": 210,
  "message": "The username and password mismatch."
}
```

3.4. 重置密码

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

3.4.1. 短信验证

忘记密码后，重置密码流程类似注册账号，先获取验证码，再验证验证码，验证通过后，设置新密码。

获取验证码方法同 3.1.1 获取验证码，获取后再进行 3.1.2 验证验证码，验证成功后调用重置密码的接口，请看下一节。

3.4.2. 重置密码

重置密码的前提是执行完 3.3.1 短信验证，然后调用接口 `resetPassword()`，

需要传入的参数：

参数	是否必须	说明
password	是	新密码

代码片段

```
var password = 'mxchip123456';
$micojs.resetPassword (password, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

验证成功后会返回用户的大部分信息，json 格式如下

```
{
  "mobilePhoneNumber": " 18623452345",
  "userToken": " ec8aed67-XXXX-XXXX-XXXX-27a5148b353c",
  "appname": "micofans",
  "username": " 18623452345",
  "emailVerified": false,
  "authData": null,
  "mobilePhoneVerified": true,
  "objectId": "55e5XXXXddb2e44a9b89e3df",
  "createdAt": "2015-09-01T09:06:15.179Z",
  "updatedAt": "2015-09-01T09:06:56.777Z"
}
```

3.5. 修改密码

如果还能记住原来的密码 ,只是想换个新密码 ,可调用接口 `updatePassword`

`()` , 需要传入的参数 :

参数	是否必须	说明
oldpsw	是	旧密码
password	是	新密码

代码片段

```
var oldpsw = '888888';
var password = 'mxchip123456';
$micojs.updatePassword(oldpsw, password, function(ret, err) {
```

```
if (ret)
    alert(JSON.stringify(ret));
else
    alert(JSON.stringify(err));
});
```

验证成功后会返回更新的时间，json 格式如下

```
{
  "updatedAt": "2015-09-01T10:13:59.400Z",
  "objectId": "55e56a87ddb2e44a9b89e3df"
}
```

3.6. 设备联网

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）以广播形式发送，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包括设备的 IP。

EasyLink 的过程需要原生的支持，仅仅通过 HTML5 或者 js 是无法实现的，所有我们需要引入自定义模块，具体操作如下。

3.6.1. 添加模块

打开 <https://git.oschina.net/bringmehome/MiCOSDK.git>，下载以下三个包的文件，每个文件内有两个子文件夹，分别对应了 Android 和 iOS 的模块。



打开 <http://www.apicloud.com/module-custom> 此页面 , 点击模块-->自定义模块-->上传自定义模块 , 分别选择 Android 和 iOS 版本的模块上传 , 模块名和 zip 包的文件名一致。版本为 0.0.1。



上传成功后点击加号 , 应用此模块



模块添加成功后界面如下 , 以下三个模块都需要添加 :



3.6.2. 开始配网

开始 EasyLink 的时候需要先获取到 WIFI 的 SSID，所以需要先调用接口

`getssid ()`，不需要传入参数，代码片段

```
$micojs.getssid(function(ret, err) {  
    if (ret)  
        alert(ret);  
});
```

验证成功后会返回 **WIFI 的名称 (SSID, 字符串)**。

拿到上面的名称后，可以使用名称以及手动输入 WIFI 的密码，再进行

EasyLink 配网，需要调用接口 `startEasyLink ()`，需要传入的参数：

参数	是否必须	说明
Ssid	是	WIFI 名称，上面已获取到
password	是	WIFI 密码

代码片段

```
var ssid = $("#wifiSsid").val();
var psw = $("#password").val();
$micojs.startEasyLink(ssid, psw, function(ret, err) {
    if (ret) {
        alert("ip = " + ret);
    }
});
```

可以通过抓包工具 Wireshark 监听到是否在发包，效果如下

Capturing from WLAN [Wireshark 1.12.4 (v1.12.4-0-gb4861da from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
39528	1470.90292	192.168.9.107	255.255.255.255	UDP	410	Source port: 52674 Destination port: 50000
39529	1470.90378	192.168.9.107	255.255.255.255	UDP	610	Source port: 52674 Destination port: 50000
39530	1470.90508	192.168.9.107	255.255.255.255	UDP	866	Source port: 52674 Destination port: 50000
39531	1470.90633	192.168.9.107	255.255.255.255	UDP	1122	Source port: 52674 Destination port: 50000
39532	1470.92486	192.168.9.107	255.255.255.255	UDP	1325	Source port: 52674 Destination port: 50000
39533	1470.92745	192.168.9.107	255.255.255.255	UDP	354	Source port: 52674 Destination port: 50000
39534	1470.96105	192.168.9.107	255.255.255.255	UDP	610	Source port: 52674 Destination port: 50000
39535	1470.97860	192.168.9.107	255.255.255.255	UDP	866	Source port: 52674 Destination port: 50000
39536	1470.98477	192.168.9.107	255.255.255.255	UDP	1122	Source port: 52674 Destination port: 50000
39537	1470.98640	192.168.9.107	255.255.255.255	UDP	1326	Source port: 52674 Destination port: 50000
39538	1470.99396	192.168.9.107	255.255.255.255	UDP	354	Source port: 52674 Destination port: 50000
39539	1470.99495	192.168.9.107	255.255.255.255	UDP	589	Source port: 52674 Destination port: 50000
39540	1470.99609	192.168.9.107	255.255.255.255	UDP	917	Source port: 52674 Destination port: 50000
39541	1470.99946	192.168.9.107	255.255.255.255	UDP	1075	Source port: 52674 Destination port: 50000
39542	1471.04274	192.168.9.107	255.255.255.255	UDP	1327	Source port: 52674 Destination port: 50000
39543	1471.04992	192.168.9.107	255.255.255.255	UDP	466	Source port: 52674 Destination port: 50000
39544	1471.06899	192.168.9.107	255.255.255.255	UDP	746	Source port: 52674 Destination port: 50000
39545	1471.08378	192.168.9.107	255.255.255.255	UDP	816	Source port: 52674 Destination port: 50000
39546	1471.08893	192.168.9.107	255.255.255.255	UDP	1176	Source port: 52674 Destination port: 50000
39547	1471.09041	192.168.9.107	255.255.255.255	UDP	1328	Source port: 52674 Destination port: 50000
39548	1471.09794	192.168.9.107	255.255.255.255	UDP	1492	Source port: 52674 Destination port: 50000
39549	1471.09974	192.168.9.107	255.255.255.255	UDP	1493	Source port: 52674 Destination port: 50000
39550	1471.10117	192.168.9.107	255.255.255.255	UDP	1494	Source port: 52674 Destination port: 50000

此接口返回设备的 ip 地址（字符串）。

3.6.3. 结束配网

拿到设备返回的 IP 地址后 就可以结束配网了 调用接口 `stopEasyLink()` ,

不需要传入参数，代码片段

```
$micojs.stopEasyLink();
```

也无返回值。

3.7. 绑定设备

绑定设备分两步：

第一步，先 APP 主动连接设备（因为设备连上 WIFI 后会自动开启一个 TCP 服务，APP 通过之前获取到的 IP 连上设备），发送激活设备的请求，调用接口 `bindingDev()`，需要传入的参数：

参数	是否必须	说明
Devip	是	设备的 IP 地址，3.5.2 开始配网时获取
Port	是	设备 tcp 服务的端口，一般为 8000
bindKey	是	绑定设备需要用到的 key， <md5(Devip + userToken)>

代码片段

```
function bindingDev() {  
    var userToken = $micojs.getUserToken();  
    var bindKey = $.md5(devip + userToken);  
    $micojs.bindingDev(devip, "8000", bindKey,  
        function(ret, err) {  
            alert(JSON.stringify(ret));  
            if (ret) {  
                var devid = ret.device_id;  
                bindtcloud(devid);  
            }  
        });  
}
```

验证成功后会返回设备的 deviceid，

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

获取了 deviceid 后，说明设备已经在云端注册成功，接下来要再到云端去把当前用户和当前设备绑定起来，调用接口 `bindDevCloud()`，需要传入的参数：

参数	是否必须	说明
----	------	----

userToken	是	当前登录用户的 token
bindKey	是	3.6 中发给设备的 bindKey

代码片段

```
$micojs.bindDevCloud(userToken, bindKey,
    function(ret, err) {
        alert(JSON.stringify(ret));
    });
```

验证成功后会返回设备的 deviceid ,

```
{
  "user-device-key":"2c7700b2-xxxx-xxxx-xxxxc735a714cd"
}
```

3.8. 获取设备列表

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

`getDevList()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken

代码片段

```
var userToken = $micojs.getUserToken();
$micojs.getDevList(userToken, function(ret) {
    alert(JSON.stringify(ret));
});
```

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"
  }]
}

```

3.9. 修改设备信息

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。

比如修改设备的名称，需要调用接口 `editDevName()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken
Devname	是	设备的新名字
Devid	是	设备的 deviceid 通过 3.7 获取设备列表后能得到此 id

代码片段

```

var devname = "设备的新名字";
var devid = "d64f517c/c8934691818b";
$micojs.editDevName(userToken, devname, devid, function(ret, err)
{

```

```
    alert(JSON.stringify(ret));
  });
```

正常情况下成功注册返回设备列表的 json 数组：

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

3.10. 获取 owner 设备列表

设备的所有权限分为 owner 和 share，

Owner：我是设备的拥有者，我不但可以控制设备，还可以把此设备授权给别人用；

Share：我是设备的使用者，我只能使用设备。

调用此接口 `getAuthDevList()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken

代码片段

```
var userToken = $micojs.getUserToken();
$micojs.getAuthDevList(userToken, function(ret, err) {
  alert(JSON.stringify(ret));
});
```

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"
  }
]

```

3.11. 授权设备

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

`authDev()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken
phone	是	被授权者的手机号（需注册过）
userType	是	被授权者的用户类型 share 或者 owner
Devid	是	设备的 deviceid 通过 3.7 获取设备列表后能得到此 id

代码片段

```

var phone = "13122223333";
var userType = "share";

```

```
var devid = "d64f517c/c8934691818b";
$micojs.authDev(userToken, phone, userType, devid,
  function(ret, err) {
    alert(JSON.stringify(ret));
  });
```

正常情况下成功注册返回设备列表的 json 数组：

```
{
  "result": "2c7700b2-xxxx-xxxx-xxxxc735a714cd"
}
```

3.12. 获取此设备下的所有用户

先看看此设备又那些人能使用，调用此接口 `queryDevUser ()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken
Devid	是	设备的 deviceid 通过 3.7 获取设备列表后能得到此 id

代码片段

```
var devid = "d64f517c/c8934691818b";
$micojs.queryDevUser(userToken, devid, function(ret, err) {
  alert(JSON.stringify(ret));
});
```

正常情况下返回所有的用户信息：

```
{
  {
    "id": "82af2473-XXXX-XXXX-XXXX-ae99f49df5f9",
    "username": "15178222205",
    "role": "share"
  },
  {
    "id": "ca9f9583-d80e-4db2-b27b-0bc968b48649",
    "username": "13212341234",
  }
}
```

```
    "role": "owner"
  }
}]}
```

3.13.取消授权

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）然后调用此接口 `removeOneUser()`，需要传入的参数：

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken
userID	是	3.11 获取设备下所有用户的时候，得到手机号的同时也得到了 id，这个 id 就是 userID
Devid	是	设备的 deviceid 通过 3.7 获取设备列表后能得到此 id

代码片段

```
var userID = "82af2473-XXXX-XXXX-XXXX-ae99f49df5f9";
var devid = "d64f517c/c8934691818b";
$micojs.removeOneUser(userToken, userID, devid, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

正常情况下返回取消授权成功：

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

3.14.解绑设备

假如我不想控制设备 A 了 ,我可以删除他 ,调用此接口 `unBindingDev ()` ,

需要传入的参数 :

参数	是否必须	说明
userToken	是	用户登陆后返回的 userToken
Devid	是	设备的 deviceid 通过 3.7 获取设备列表后能得到此 id

代码片段

```
var devid = "d64f517c/c8934691818b";
$micojs.unBindingDev(userToken, devid, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

正常情况下返回解绑成功 :

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

3.15.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。

3.15.1. 打开 MQTT

如果要进行 MQTT 通讯 , 首先是要建立一个通道 , 调用接口

`startMqttService ()` , 需要传入的参数 :

参数	是否必须	说明
host	是	一百年不变的域名 api.easylink.io
username	是	admin
password	是	admin
topic	是	需要监听的通道名称： 如：d64f517c/c8934691813c/out/#
clientID	是	<i>v1-app-[token 版本号-app-userToken(前 12 位)]</i> userToken 就是用户登陆后返回的 userToken

代码片段

```
var userToken = $micojs.getUserToken();
var device_id = "d64f517c/c8934691813c";
var host = "api.easylink.io";
var username = "admin";
var password = "admin";
//clientID, 需要按照此标准来定义:v1-app-token 版本号-app-userToken(前12位)
var clientID = "v1-app-" + userToken.substring(0, 12);
var topic = device_id + '/out/#';
$micojs.startMqttService(host, username, password, topic,
clientID, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

正常情况下返回设备上报的数据，数据是 json，内容不定，如：

```
{
  "9": 3021
}
```

执行 stopMqttService 之前，此监听通道都会一直打开。

3.15.2. 发送指令

如果是想发送指令送给设备，需要调用此接口 `publishCommand()`，需要

传入的参数：

参数	是否必须	说明
topic	是	需要发送指令的通道名称： 如：d64f517c/c8934691813c/in/write
Command	是	控制的指令： 如{ "power" :1}

代码片段

```
var device_id = "d64f517c/c8934691813c";
var command;
if (1 == type) {
    command = '{"4":true}';
} else if (0 == type) {
    command = '{"4":false}';
}
var topic = device_id + "/in/write";
$micojs.publishCommand(topic, command, function(ret, err) {
    alert(JSON.stringify(ret));
});
```

正常情况下返回操作成功，如：

```
{
  "status": true
}
```

3.15.3. 关闭 MQTT

使用好指令控制，需要关闭通道的话调用接口 `stopMqttService()`，不需

要传入参数，代码片段

```
$micojs.stopMqttService();
```

也没有返回值。

4. 总结

上文介绍的APP开发是使用APICloud工具，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)