MiCO Documentation Working Group (MDWG) Eshen Wang

Track Number: 0003CN MXCHIP Co., Ltd

Version: 1.2 2016.1.14

Category: Technical Manual

MiCOKit 固件开发手册

摘要（Abstract）

本文档主要介绍 MiCOKit 固件开发方法，使用户对 MiCO 智能硬件开发有个初步认识，方便 MiCO-IoT 物联网开发者使用 MiCOKit 开发板及 FogCloud 进行物联网应用开发。

适用对象（Status of This Document）

本文档适用于初级 MiCOKit 开发者，对所有 MiCO-IoT 物联网智能硬件开发者公开。

获取更多帮助（More Help）

MiCO 开发团队向您推荐：MiCO 开发者学习网站：http://mico.io/（至开发者中心），获取更多最新资料。

手机微信“扫一扫”关注：“MiCO 总动员”公众号，获取 MiCO 团队小伙伴最新活动信息。

登录上海庆科官方网站：http://mxchip.com/，获取公司最新产品信息。

目 录

1. 概述

MiCOKit 固件基于庆科 MiCO 物联网操作系统，快速接入 Wi-Fi 网络；并通过庆科的 FogCloud 云完成设备和手机 APP 之间的交互，从而实现物联网应用的各种功能。

本文档主要介绍 MiCOKit 固件开发方法以及所涉及到的相关内容，主要包括 MiCOKit SDK、MiCO 系统、FogCloud 云接入、以及固件 App 框架，帮助开发者深入理解 MiCOKit 的固件开发流程，以便进行更深入的二次开发工作。

2. MiCOKit SDK

上海庆科为 MiCOKit 开发者提供一套完整的固件开发工具包，帮助固件开发者学习并快速开发自己的 IOT 应用。下载 MiCOKit SDK 请登录 MiCO 开发者网站：mico.io 的 WiKi 中心下载(http://mico.io/wiki/doku.php?id=micokit_sdk)。

2.1. 固件 SDK 使用

（1）安装开发环境，主要包括：

a. 安装 IAR workbench for ARM on Windows(7.30.1 及以上);

b. 安装 ST-LINK 或 J-LINK 驱动；

c. 安装 USB 虚拟串口驱动（FTDI）;

d. 安装 secureCRT 串口调试工具；

详细方法请登录 mico.io 参考：http://mico.io/wiki/doku.php?id=prepare_pc

（2）在 MiCOKit 发布中心页面，下载最新 MiCOKit SDK，内含 MiCOKit 固件示例工程源代码；

（3）打开 MiCOKit SDK 固件工程，编译代码，下载到 MiCOKit 开发板中；

固件工程在 Projects 目录下， 按照 MCU 类型分类，应用程序源代码在 Demos\micokit 目录下，按照应用功能分类。

详细操作方法请登录 mico.io 参考：http://mico.io/wiki/doku.php?id=debug2.4.0

（4）按开发板上的 Reset 键，运行程序，可通过串口 LOG 查看固件运行情况，并使用 MiCOKit 配套的手机 APP 测试相关功能。

2.2. 实例工程说明

（1）以 MiCOKit-3165 为例，工程路径： Projects\STM32F4xx。如图 2.1 所示。

图 2.1 工程路径

（2）使用 IAR 打开 Projects\STM32F4xx\micokit\Cloud_RGB_LED.eww 工程环境，选择 MiCOKit-3165，如图 2.2 所示：

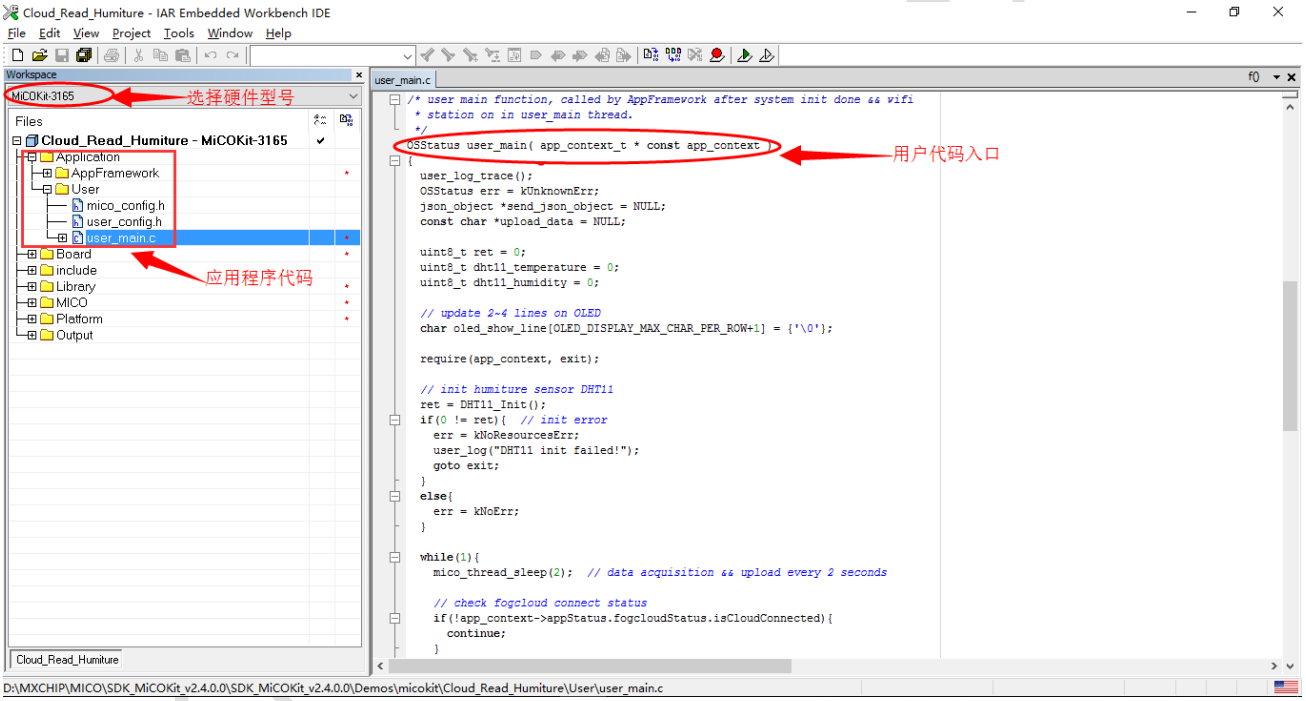

图 2.2 工程环境

（4）查看（修改）用户代码：

工程中 Application 目录下为应用程序相关的代码。

将 MiCOKit 演示程序固件修改为开发者自己的产品（修改固件产品 id/key）：

(a)登录 FogCloud 网站，创建自己的产品，如图 2.3 所示。

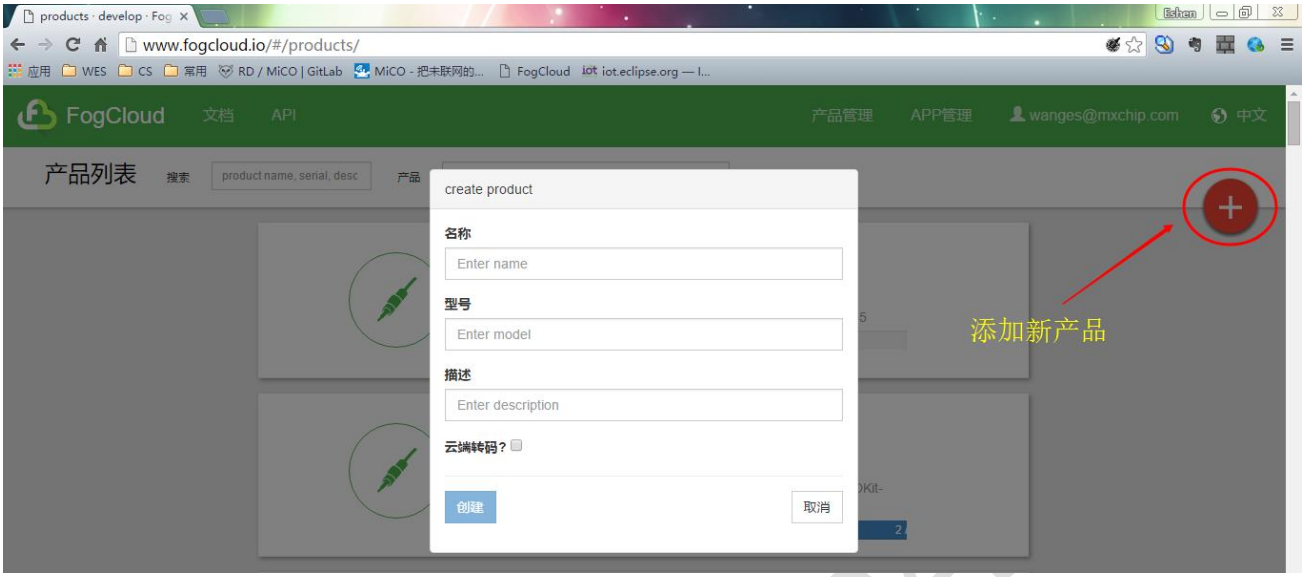

图 2.3 创建产品

(b)获取产品 ID/Key:

图 2.4 获取产品 ID 和 Key

(c)替换示例工程中的 ID/Key：

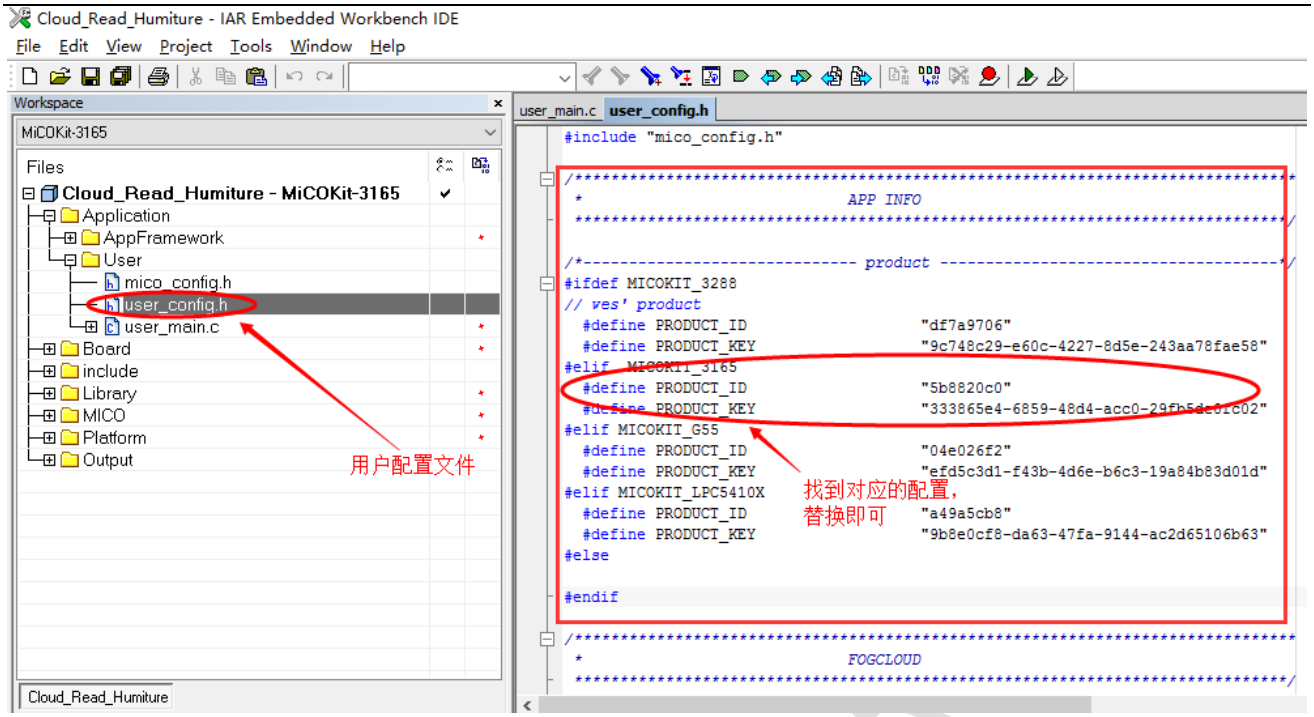

图 2.5 替换产品 ID 和 Key

（5）编译、调试、下载。

详细的编译，调试，下载方法请登录 mico.io 参考：http://mico.io/wiki/doku.php?id=debug2.4.0

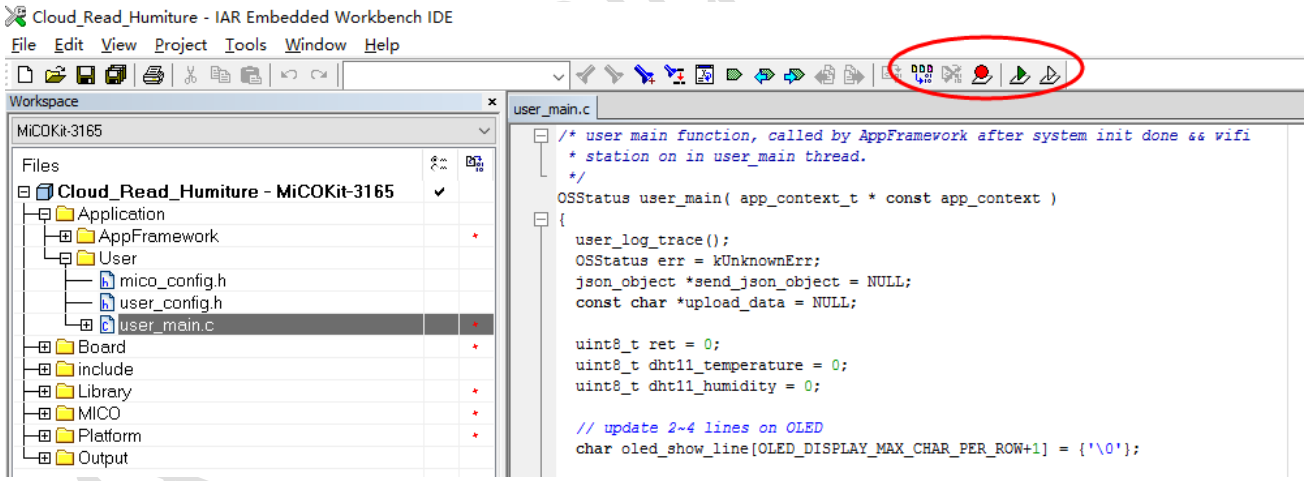

图 2.6 编译调试与下载

（6）连接好 USB，PC 上配置好串口，查看设备运行 LOG：

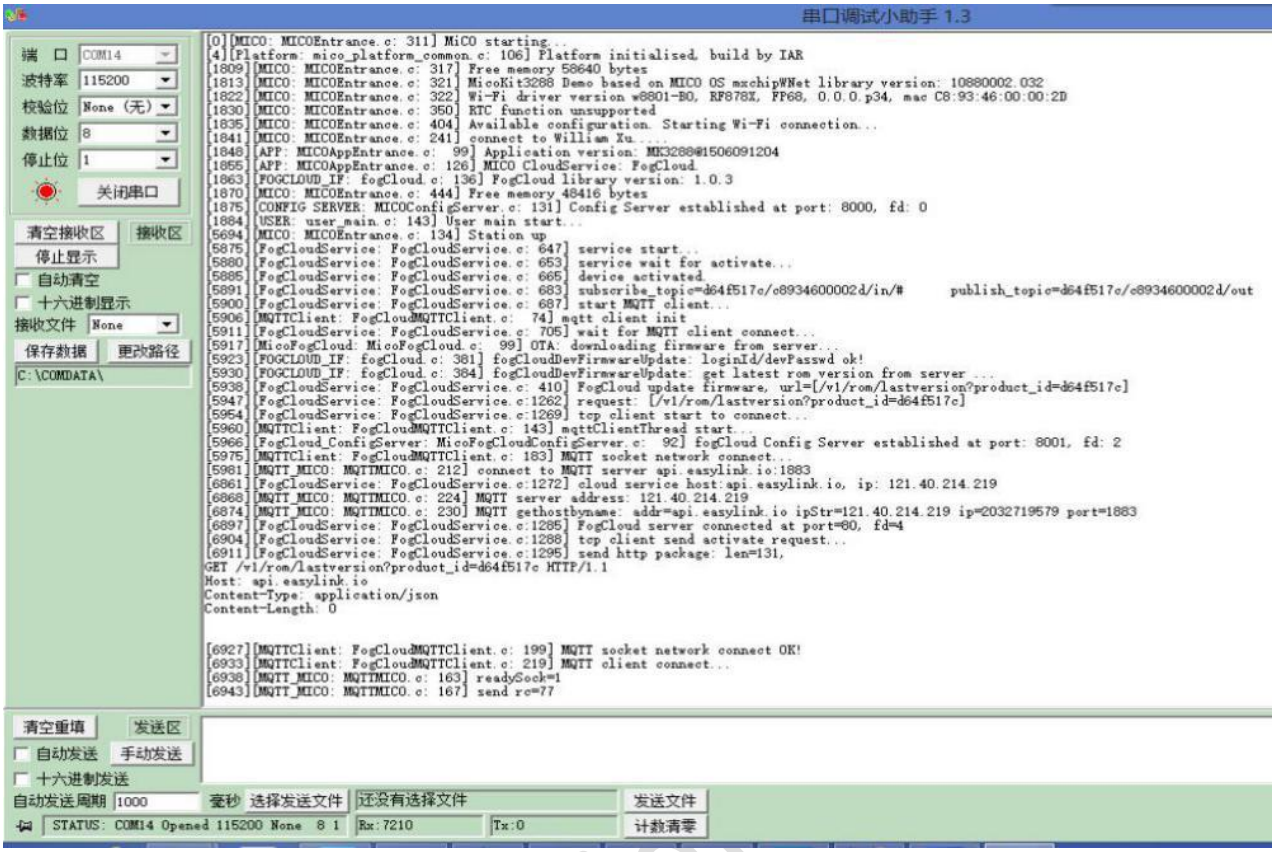

图 2.7 串口 log 输出

（7）使用手机 APP 联合设备进行控制调试（详细操作方法见《MiCOKit 用户手册》）。

3. MiCO

MiCO(Micro-controller based Internet Connectivity Operating System 是一个面向智能硬件优化设计的、运行在微控制器上的、高度可移植的操作系统和中间件开发平台。MiCO 作为独立的系统，拥有开放架构，还包括了底层芯片驱动、无线网络协议、射频控制技术、安全、应用框架等，能够帮助 IoT 设备开发者降低软件开发难度，快速形成可以量产的产品。

MiCO 内含一个面向 IoT 设备的实时操作系统内核，特别适合运行在能量受限的微控制设备上。此外，MiCO 还包含了网络通信协议栈，安全算法和协议，硬件抽象层，编程工具等开发 IoT 必不可少的软件功能包，MiCO 系统内核框架结构如下图所示。

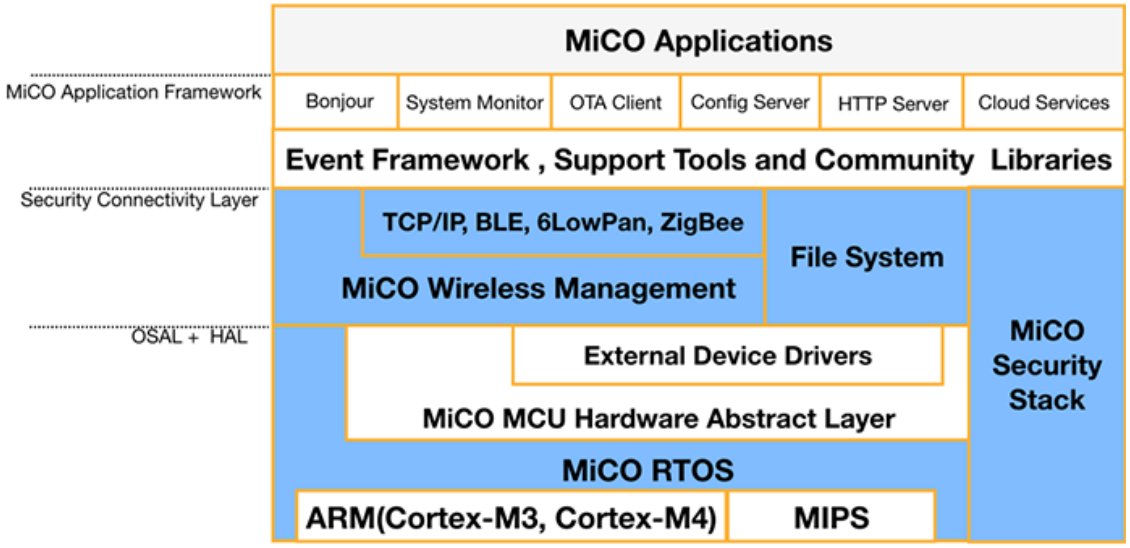

图 3.1 MiCO 系统框架

MiCO 提供 MCU 平台的抽象化，使得基于 MiCO 的应用程序开发不需要关心 MCU 具体件功能的实现，通过 MiCO 中提供的各种编程组件快速构建 IoT 设备中的软件，配合 MiCOKit 开发套件实现快速产品原型开发。

MiCO 的关键特性

- 支持多种网络协议栈，并持续增加中：Bluetooth low energy，以太网，Wi-Fi，ZigBee，6LoWPAN；
- 全自动高效功耗管理；
- 支持多种 MCU 体系结构：Cortex-M 系列，MIPS 等，提供 MCU 平台级的抽象化，标准外设驱动接口；
- 完整的网络安全算法和协议栈，支持常用的传输层安全协议 TLS；
- 方便易用的应用程序框架，使 MiCO 应用安全地直达云端；
- 支持多种 MCU 常用的开发调试环境，可以进行硬件仿真。提供命令行接口和标准调试信息输出，方便实现运行中的调试；
- 提供完整的设备量产技术，如引导程序，量产烧录，远程网络升级服务，生产测试程序等；

- 基于 MiCO 系统开发的 IoT 设备软件已经成功地运行在大量的商品上，是一个被证明了的，安全，稳定，可靠的软件平台。

  MiCO 详细资料请查看 MiCO 开发者网站：http://mico.io/wiki/doku.php?id=homepage

在 MiCOKit SDK 中，MiCO 相关的源代码如下图：

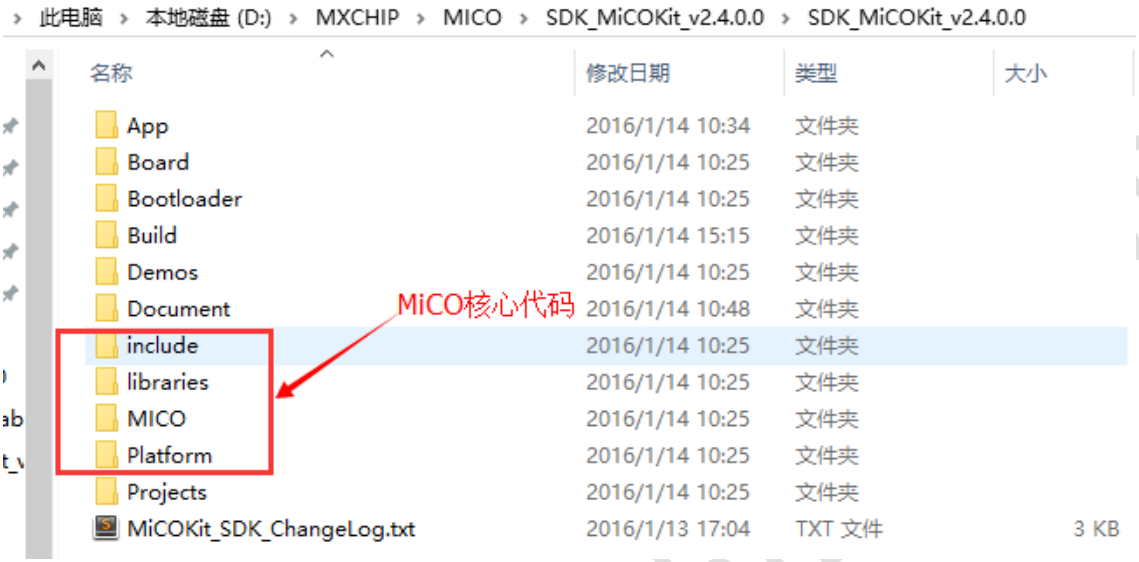

图 3.2 MiCO 核心代码文件

4. FogCloud

FogCloud 是上海庆科专门为智能硬件提供后台支持的云服务平台，实现了硬件产品与智能手机及云端服务互连，为多种行业提供云端解决方案。目前庆科云可以提供包括设备云端互联、MiCO 软件 OTA 升级、阿里智能云、微信等第三方公有云接入等支持服务。MiCOKit 接入 FogCloud 工作架构如图 4.1 所示。

图 4.1 MiCOKit 接入 FogCloud 工作架构图

设备接入 FogCloud 流程如图 4.2 所示：

（1）创建新产品，获取产品 ID 和 KEY；

（2）设备使用产品 ID/KEY 向 FogCloud 激活，获取设备 ID 和 KEY；

（3）设备使用设备 ID/KEY 连接 FogCloud 消息服务器，收发消息；

（4）设备从云端注销（删除设备）。

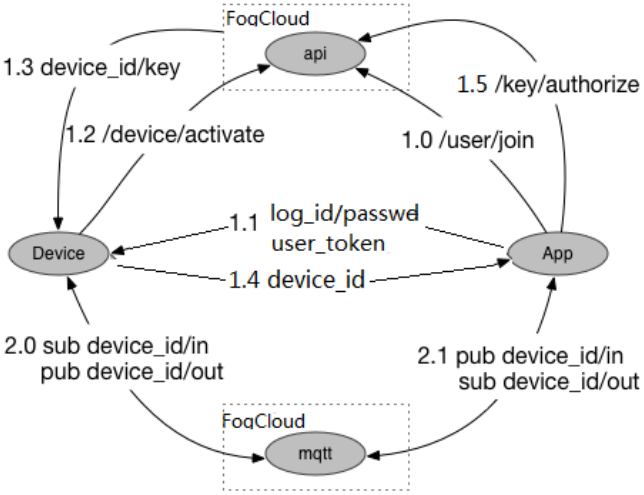

图 4.2 设备接入 FogCloud 流程

FogCloud 详细接入流程请参考 FogCloud 文档: http://www.fogcloud.io/documents/

5. AppFramework

MiCOKit 基于 MiCO 操作系统，集成 FogCloud 云接入功能，给开发者提供了一套简单易用的固件应用程序框架，用户只需要修改用户相关的功能代码即可完成相应的功能，无需关心设备网络连接、云接入等内容，大大节约了固件开发者的开发工作。

5.1. 固件结构

MiCOKit 固件基于 MiCO 系统 ,开发者在 MiCO 应用程序入口函数(application_start)中添加自己的应用代码，通过调用 MiCO API 即可完成应用所需的功能。

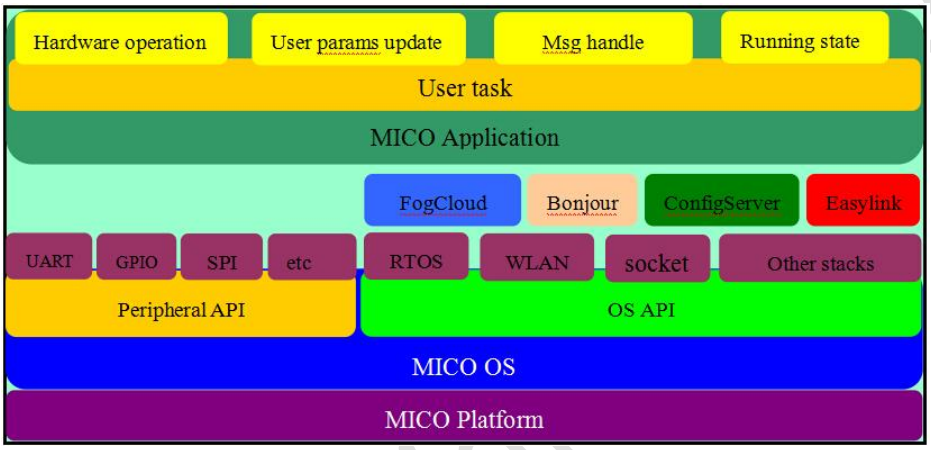

图 5.1 MiCOKit 固件结构

5.2. 固件流程图

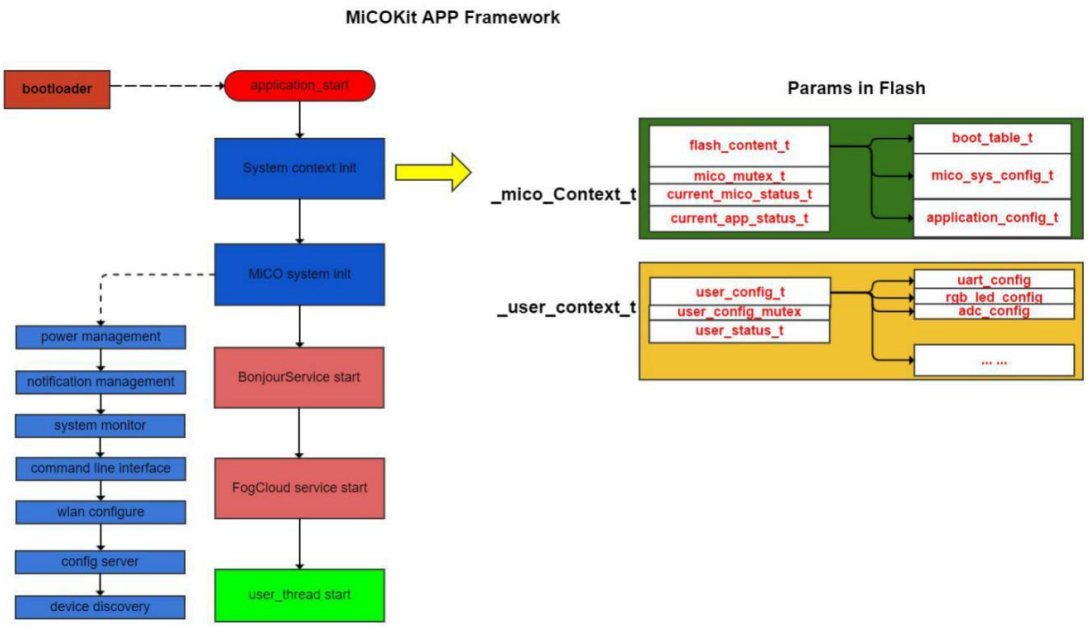

图 5.2 MiCOKit APP Framework

6. 设备网络配置

MiCOKit 网络配置主要使用手机 APP 等客户端通过 Easylink 协议给设备发送要连接的 Wi-Fi 网络的 SSID 和密码，让设备可以连接指定的 Wi-Fi 网络。

6.1. Wi-Fi 网络配置

当 MiCOKit 进入配置模式时（通常为短按 MiCOKit 上的 Easylink 按键，或者没有配置过的模块初始状态会进入配置模式），即可使用手机 APP 通过 Easylink 配网协议给设备发送要连接的 Wi-Fi 网络的 SSID 和密码，设备接收到消息数据后，通过 Easylink 协议解析出 SSID 和密码，即可连接指定的 Wi-Fi 网络。

设备连接 Wi-Fi 成功，则开启 Bonjour 服务，向 APP 广播设备信息，表示 Wi-Fi 网络配置成功；如果设备连接 Wi-Fi 失败，则重新进入配置模式，准备重新接收 SSID 和密码。**注意：**

（1）Easylink 协议已集成到 MiCO 系统中，固件开发者只需要调用相关 API 即可开启配网功能；

（2）APP 开发者可通过 Easylink 接口库 API 完成 Easylink 功能，可登录 mico.io 中 APP 开发相关参考：http://mico.io/wiki/doku.php?id=micoappdownload

7. 云连接配置

设备 Wi-Fi 连接成功后，开启 FogCloud Congifg Server(TCP Server)，用户 APP 作为 TCP client 可与之建立连接。APP 使用 mDNS 协议发现设备，并与之建立 TCP 连接，之后通过 HTTP 协议与设备进行数据交互，完成设备的激活、授权、重置等请求，使得设备接入 FogCloud 云端并和 APP 用户绑定。

通信数据包采用 JSON 格式，具体通信方式如下：

7.1. 设备接入 FogCloud 云流程

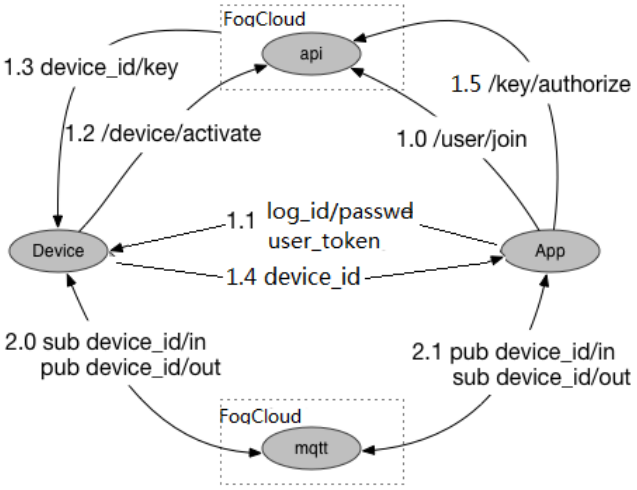

图 7.1 设备接入 FogCloud 云流程

（1）设备首次接入 FogCloud 云之前，需要先激活设备；APP 向设备发送激活请求，使得设备向 FogCloud 云端激活，成功后返回设备 ID 给 APP，完成设备激活和绑定；如上图 1 中的 1.1，1.2，1.3，1.4，1.5；

（2）设备激活成功后，重启后自动连接 FogCloud 云消息服务器（上图 1 中 2.0）；

（3）之后其他 APP 要绑定设备，只需要向设备发送授权请求（如果发送激活请求，则设备实际执行授权），设备向云端请求 APP 授权，返回设备 ID 给 APP（过程同步骤(1)）；

（4）激活（或授权）完成后，APP 通过设备 ID 向 FogCloud 云端查询设备连接状态，并可与设备间进行云端消息收发（上图 1 中的 2.1）。

7.2. 设备配置接口（供参考）

按照以上 7.1 中描述的 FogCloud 接入流程，MiCOKit 演示实例使用如下方法完成接入配置操作，开发者也可以自行选择其他实现方法完成相同的功能。

7.2.1 查询设备状态请求

- App 发送给设备：

表 7.1 查询设备状态，App 发送数据

数据包结构	内容	说明
Header	Host	mDNS 发现设备，获得设备 IP
	Port	8001
	URL	/dev-state
data	login_id	设备登录名（默认 admin）
	dev_passwd	设备登录密码（默认 12345678）
	user_token	用户 token

- 设备响应：

表 7.2 查询设备状态，设备响应数据

数据包结构	内容	说明
Header	status	200 OK
	Content-Type	application/json
data	isActivated	激活状态(true/false)
	isConnected	云连接状态(true/false)
	version	ROM 版本字符串(如：v0.x.x)

- 实例：

- App 发送：

```
POST /dev-state HTTP/1.1
Host: 192.168.31.180:8001
Content-Length: 74
Cache-Control: no-cache

{"login_id":"admin","dev_passwd":"12345678","user_token":"11111111"}
```

- 设备返回：

```
{ "isActivated": true, "isConnected": true, "version": "v0.2.3" }
```

7.2.2 激活请求

- APP 向设备发送：

表 7.3 激活请求，App 发送数据

数据包结构	内容	说明
Header	Host	mDNS 发现设备，获得设备 IP
	Port	8001
	URL	/dev-activate
data	login_id	设备登录名（激活后保存）
	dev_passwd	设置设备登录密码（激活后保存）
	user_token	用户 token

- 设备响应：

表 7.4 激活请求，设备响应数据

数据包格式	内容	说明
Header	status	200 OK
	Content-Type	application/json
data	device_id	设备激活后获得的唯一 ID

- 实例：

- APP 发送：

```
POST /dev-activate HTTP/1.1
Host: 192.168.31.180:8001
Content-Length: 74
Cache-Control: no-cache
{"login_id":"admin","dev_passwd":"12345678","user_token":"11111111"}
```

- 设备返回：

{ "device_id": "af2b33be/c8934645dd0a" }

注意：激活成功后，设备将会保存用户设置的用户名和密码，后续请求会验证该用户名和密码。

7.2.3 用户授权请求

- APP 向设备发送:

表 7.5 用户授权请求，APP 发送数据

数据包结构	内容	说明
Header	Host	mDNS 发现设备，获得设备 IP
	Port	8001
	URL	/dev-authorize
data	login_id	设备登录名
	dev_passwd	设备登录密码
	user_token	用户 token

- 设备响应

表 7.6 用户授权请求，设备响应数据

数据包结构	内容	说明
Header	status	200 OK
	Content-Type	application/json
data	device_id	设备的唯一 ID

- 实例：

- APP 发送：

POST /dev-authorize HTTP/1.1
Host: 192.168.31.180:8001
Content-Length: 74
Cache-Control: no-cache

{"login_id":"admin","dev_passwd":"12345678","user_token":"22222222"}

- 设备返回：

{ "device_id": "af2b33be/c8934645dd0a" }

7.2.4 设备注销请求

该方法使得设备从云端注销，下次再使用需要重新激活。

- APP 向设备发送：

表 7.7 设备注销请求，APP 发送数据

数据结构	内容	说明
Header	Host	mDNS 发现设备，获得设备 IP
	Port	8001
	URL	/dev-cloud_reset
data	login_id	设备登录名
	dev_passwd	设备登录密码
	user_token	用户 token

- 设备返回：

表 7.8 设备注销请求，设备返回数据

数据结构	内容	说明
Header	status	200 OK
data	error	{ “error” : <err_code>}

- 实例：

- APP 发送：

```
POST /dev-cloud_reset HTTP/1.1
Host: 192.168.31.180:8001
Content-Length: 74
Cache-Control: no-cache

{"login_id":"admin","dev_passwd":"12345678","user_token":"11111111"}
```

- 设备返回：

成功，无数据实体；

以上操作失败时，返回状态码 500，消息实体返回详细错误码：

{ “error” : <err_code>}

7.2.5 状态码

200: 执行成功

500: 执行失败

详细错误码请参考：MiCOKit 最新 SDK 中，头文件 Common.h 中 MiCO 错误码。

7.3. 云数据通信

MiCOKit 和云端及用户端通过 FogCloud 服务器进行消息通信。通信协议采用 MQTT 协议 ,设备端和 APP 端作为 MQTT 客户端，FogCloud 作为 MQTT 服务器完成消息通信。设备、APP 以及服务器间进行消息通信示意如图 7.2。

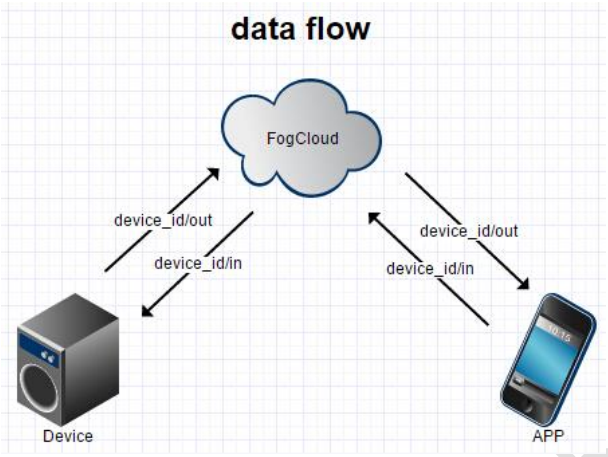

图 7.2 设备、APP 及服务器间消息通信

7.3.1 数据通道

APP 和 MiCO 设备之间通过 FogCloud 云的不同的 MQTT 数据通道进行数据交互。

表 7.9 设备数据通道

类型	通道	消息流向
设备输入	<device_id>/in/<session_id>	APP ==> dev
设备输出	<device_id>/out/<session_id>	Dev ==> APP

其中，<session_id>表示请求的来源（由 APP 决定），设备根据此 session_id 回复请求方，不设置则表示广播该消息。

7.3.2 消息体格式

为了方便在 FogCloud 上存储消息数据，APP 和设备之间消息体的数据格式采用 JSON 格式（目前仅支持单层级的 JSON 数据），使用一个 key-value 对表示一个数据点的名称和值。

JSON 单层级 Key-Value 结构：

```
{
    “data1”：<value1>,
    “data2”：<value3>,
    “data3”：<value3>,
  ...
}
```

8. 版本更新说明

日期	版本	更新内容	作者
2015-7-29	V1.0	1. 初始版本	Eshen Wang
2015-9-8	V1.1	1. 格式修订	Jenny Liu
2016-1-14	V1.2	1. 合并所有固件开发相关内容	Eshen Wang