MicoKit APP 和设备云数据通信协议

wanges@mxchip.com

v0.1.1

2015.3.13

MXCHIP

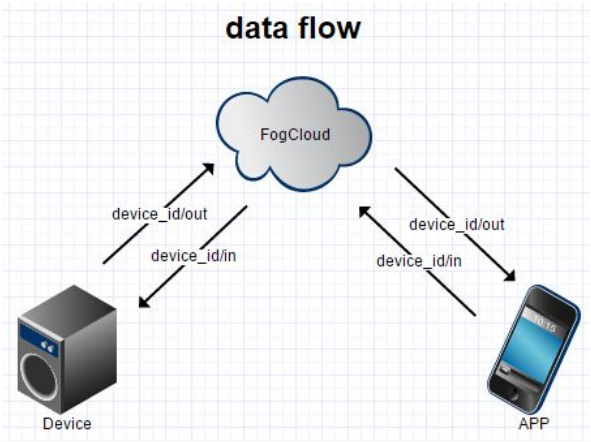
一、概述

本文档主要定义 MICO 设备及相应的 APP 连接 FogCloud 后，APP 与设备之间的产品功能数据的封装、交互协议，主要是 MQTT 数据包的交互。

APP 端和设备端根据约定的 MQTT 数据通道收发不同类型的消息；同时根据数据协议进行封装、解析 MQTT 数据包中的字段，提取属性值，完成相应的操作。用户或者设备的数据包封装采用 JSON 格式，不同产品可灵活的添加不同的属性，定制不同的产品功能。

二、通信协议

1. 通信模型



2. 数据通道

APP 和设备之间通过不同的 MQTT 通道完成数据的读取、写入。

消息通道定义以设备为中心，“in”表示设备输入，“out”表示设备输出，

(1) 消息通道定义：

<device_id>/<direction>/<attribute>/<session_id>

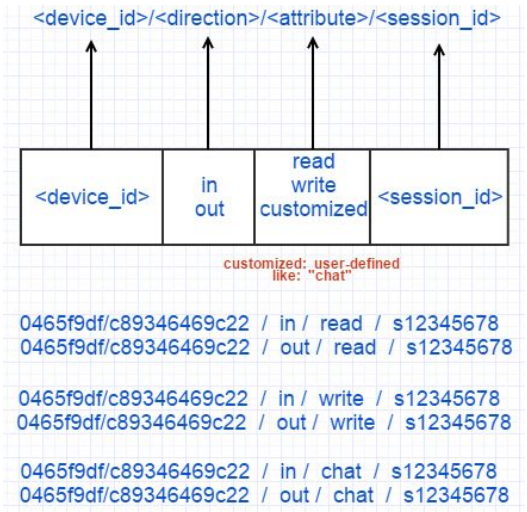
其中：

<device_id>表示设备在服务器上的唯一 ID，如： <product_id>/<mac>；

<direction>表示消息是输入到设备，还是从设备输出， 如： in、out；

<attribute>表示当前请求的属性，如：read、write、chat；

<session_id>表示当前请求的会话 ID，由 APP 随机产生，设备根据此 ID 确定消息响应的对象；若不带 session_id，则设备广播回复。



例如：

APP 向通道发送消息： <device_id>/**in**/read/<session_id>；

ID 为<device_id>的设备收到该消息，为读属性消息，发送对象为<session_id>，

设备响应通道： <device_id>/**out**/read/<session_id>；

(2) 设备标准属性通道：

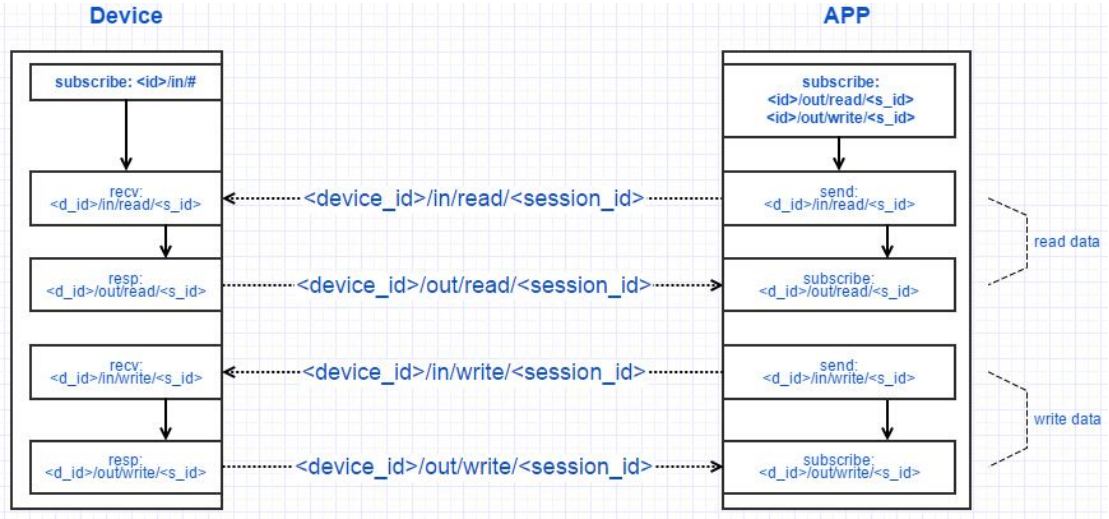
APP 向设备读取请求通道： <device_id>/in/read/<session_id>

APP 向设备写入数据通道： <device_id>/in/write/<session_id>

设备响应 APP 读数据请求通道： <device_id>/out/read/<session_id>

设备响应 APP 写数据请求通道： <device_id>/out/write/<session_id>

通道数据流程示例，如下图所示：



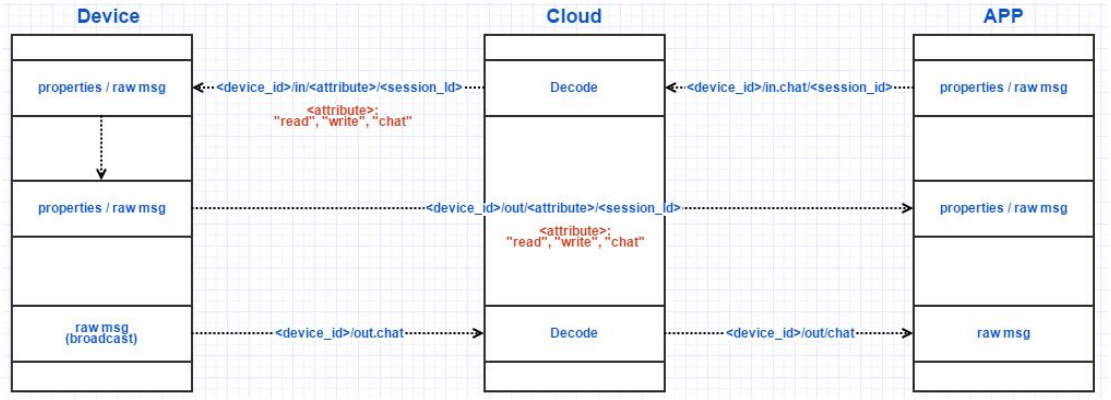
(2) 用户自定义属性通道（使用云端编解码）：

APP 向设备发送消息： <device_id>/in.xxx/<session_id>

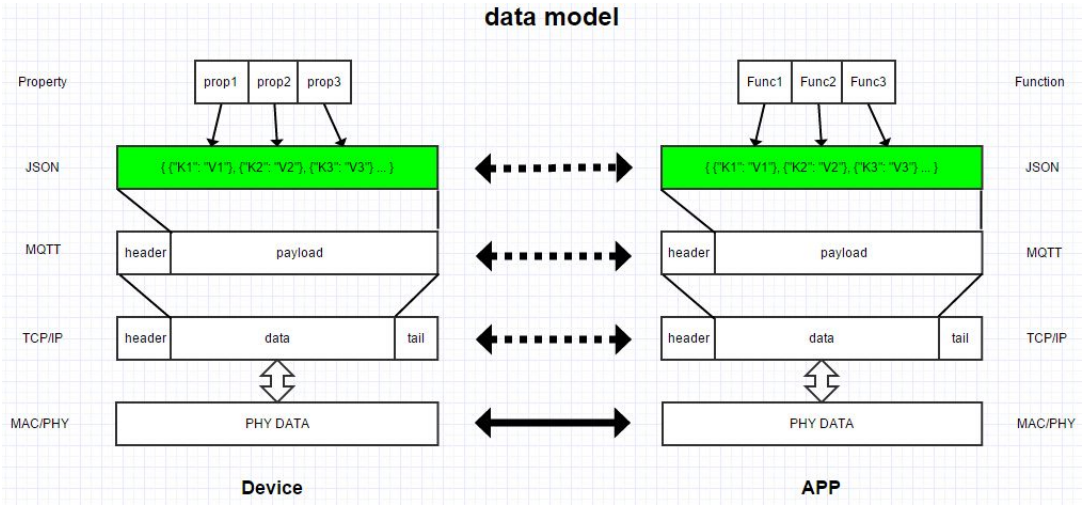
设备向 APP 广播消息： <device_id>/out.xxx

其中”in.xxx”和”out.xxx”表示用户自定义协议子通道，如：”in.chat”及”out.chat”，表示用户自定义的 APP 和设备聊天的协议。云端会拦截这一类型的通道的消息，进行云端解码后再转发给设备相应的标准通道(in/out)。

通道数据流程如下图所示：



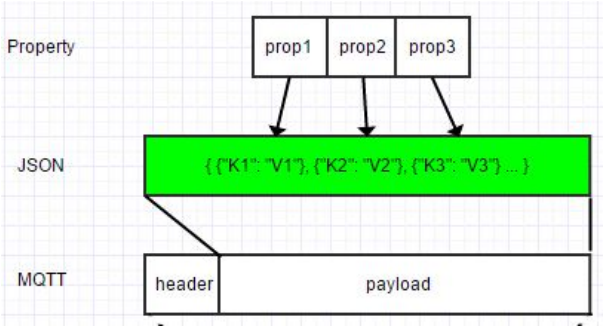
3. 数据封装



三、数据封装

1. 数据帧

设备和 FogCloud、APP 与 FogCloud 之间均采用 MQTT 协议收发消息，APP 或者设备的 JSON 数据包填充到 MQTT 的 Payload 部分进行发送。



2. 封装

这里主要定义 JSON 数据包的封装。将设备不同的模块的各个属性封装到一个 JSON 数据包中，JSON 数据包中的字段都只有一个层级，设备的每一个属性对应 JSON 的一个 K-V 值。

(1) JSON 基本数据包格式如下：

```
{
  "property1": "value1",          // 属性 1
  "property2": "value2",          // 属性 2
  "property3": "value3",          // 属性 3
 }
```

(2)JSON 命名规则：

JSON 数据包命名规则如下：

(a) key 值表示设备属性，value 表示属性值；

(b) key 字符串命名采用短横线分隔，将子属性进行分类，如：

“module-property-subprop”

(3)设备响应状态：

(a) 状态 key 字符串:

“status-write” ： 写入操作返回状态

(b) 状态码：

200： 成功

201： 失败

202： 写入无效的属性

(4)实例

例如：设备上有一个 LED 灯，一个温湿度传感器，两个模块；

每个模块具有不同的属性：

LED：

开关（switch）；

HSB 颜色参数（hues，saturation，brightness）

温湿度传感器：

温度值（temperature）；

湿度值（humidity）

JSON 数据封装成如下格式：

```
{
   "led-hsb-h": 360,              // LED 灯色相值(0~360)
   "led-hsb-s": 100,              // LED 灯饱和度值(0~100)
   "led-hsb-b": 100,              // LED 灯亮度值(0~100)
   "led-sw": 0,                   // LED 灯开关量(0/1)
   "ht-t": 20,                    // 温湿度传感器的温度值(0~100)
   "ht-h": 60                     // 温湿度传感器的湿度值(0~100)
}
```

(a)APP 向设备查询 LED 状态，温湿度值，发送请求消息格式如下：

发送通道： <device_id>/in/read/<session_id>

```
{
   "led-hsb-h": 0,              // 请求 LED 灯色相值
   "led-hsb-s": 0,              // 请求 LED 灯饱和度值
   "led-hsb-b": 0,              // 请求 LED 灯亮度值
   "led-sw": 0,                 // 请求 LED 灯开关量
   "ht-t": 0,                   // 请求温湿度传感器的温度值
   "ht-h": 0                    // 请求温湿度传感器的湿度值
}
```

设备端响应：

发送通道： <device_id>/out/read/<session_id>

```
{
   "led-hsb-h": 360,              // 请求 LED 灯色相值
   "led-hsb-s": 100,              // 请求 LED 灯饱和度值
   "led-hsb-b": 100,              // 请求 LED 灯亮度值
   "led-sw": 0,                   // 请求 LED 灯开关量
   "ht-t": 20,                    // 请求温湿度传感器的温度值
   "ht-h": 60                     // 请求温湿度传感器的湿度值
}
```

(b)APP 向设备设置 LED 灯颜色，发送写入数据格式如下：

发送通道： <device_id>/in/write/<session_id>

```
{
   "led-hsb-h": 120,              // 设置 LED 灯色相值
   "led-hsb-s": 90,               // 设置 LED 灯饱和度值
   "led-hsb-b": 50,               // 设置 LED 灯亮度值
   "led-sw": 1,                   // 设置 LED 灯开
}
```

设备端响应：

发送通道：<device_id>/out/write/<session_id>

```
{
  "status-write": 200,              // 设备数据写入成功(状态码 200)
}
```

(c)设备端主动上传（广播）数据：

发送通道： <device_id>/out/write

```
{
  "led-hsb-h": 120,                 // 上传 LED 灯色相值
  "led-hsb-s": 90,                  // 上传 LED 灯饱和度值
  "led-hsb-b": 50,                  // 上传 LED 灯亮度值
  "led-sw": 1,                      // 上传 LED 灯开关量
  "ht-t": 20,                       // 上传温湿度传感器的温度值
  "ht-h": 60                        // 上传温湿度传感器的湿度值
}
```