**MeOcer系统接口说明书**

——MQTT

|  |  |  |  |
| --- | --- | --- | --- |
| 文档版本号： | V1.1 | 文档编号： |  |
| **文档密级：** | 公开 | **归属部门/项目：** | 研发 |
| **产品名：** | MeOcer | **子系统名：** | MQTT API |
| **编写人：** | 石伟丞 | **编写日期：** | 2017-02-10 |

成都微智慧科技 版权所有

内部资料 注意保密

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修订人 | 修订日期 | 修订描述 |
| V 1.0 | 石伟丞 | 2017-01-10 | 初稿 |
| V 1.1 | 石伟丞 | 2017-02-10 | 修改message接口中命令参数命名   1. funcName改为cmdName 2. funcCode改为cmdCode 3. parameters改为options |
|  |  |  |  |
|  |  |  |  |

文档说明

1文档简介

本文档主要描述物联网云平台MeOcer提供的应用接口。MeOcer支持多种通讯协议HTTP, Socket.io, Websocket, MQTT, CoAP, AMQP, and XMPP。但本文档只详细阐述MeBox将会调用到的MQTT协议。

2文档读者

本文档主要面向以下读者：

* MeBox产品的研发人员
* MeHome产品的研发人员
* 测试人员
* 产品经理
* 市场运营人员
* 管理人员

3专业术语

目录

[——MQTT 1](#_Toc471990580)

[1. MQTT协议简介 5](#_Toc471990581)

[1.1. 网络连接 Network Connection 5](#_Toc471990582)

[1.2. 客户端 Client 5](#_Toc471990583)

[1.3. 服务端 Server 5](#_Toc471990584)

[1.4. 订阅 Subscription 5](#_Toc471990585)

[1.5. 发布 Publish 6](#_Toc471990586)

[2. MeOcer通讯协议简介 6](#_Toc471990587)

[3. API功能说明 7](#_Toc471990588)

[3.1. message 7](#_Toc471990589)

[3.1.1. Request 7](#_Toc471990590)

[3.1.2. Response 8](#_Toc471990591)

[3.2. Update 8](#_Toc471990592)

[3.2.1. Request 8](#_Toc471990593)

[3.2.2. Response 8](#_Toc471990594)

[3.3. whoami 9](#_Toc471990595)

[3.3.1. Request 9](#_Toc471990596)

[3.3.2. Response 9](#_Toc471990597)

[3.4. resetToken 10](#_Toc471990598)

[3.4.1. Request 10](#_Toc471990599)

[3.4.2. Response 10](#_Toc471990600)

[3.5. generateAndStoreToken 11](#_Toc471990601)

[3.5.1. Request 11](#_Toc471990602)

[3.5.2. Response 11](#_Toc471990603)

[3.6. getPublicKey 12](#_Toc471990604)

[3.6.1. Request 12](#_Toc471990605)

[3.6.2. Response 12](#_Toc471990606)

# MQTT协议简介

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）是IBM开发的一个即时通讯协议，有可能成为物联网的重要组成部分。它是一个基于发布/订阅模式的，简单和轻量级的消息传递协议，专为受限设备和低带宽，高延迟或不可靠的网络而设计。设计原则是尽量减少网络带宽和设备资源的要求，同时也试图确保可靠性和一定程度的交付保证。这些原则为“机器对机器（M2M）”或“物联网（IoT）”领域中，设备间的互联互通提供极好的通讯保障。

## 网络连接 Network Connection

MQTT使用的底层传输协议基础设施。

* 客户端使用它连接服务端。
* 它提供有序的、可靠的、双向字节流传输。

应用消息 Application Message MQTT协议通过网络传输应用数据。应用消息通过MQTT传输时，它们有关联的服务质量（QoS）和主题（Topic）。

## 客户端 Client

使用MQTT的程序或设备。客户端总是通过网络连接到服务端。它可以

* 发布应用消息给其它相关的客户端。
* 订阅以请求接受相关的应用消息。
* 取消订阅以移除接受应用消息的请求。
* 从服务端断开连接。

## 服务端 Server

一个程序或设备，作为发送消息的客户端和请求订阅的客户端之间的中介。服务端

* 接受来自客户端的网络连接。
* 接受客户端发布的应用消息。
* 处理客户端的订阅和取消订阅请求。
* 转发应用消息给符合条件的已订阅客户端。

## 订阅 Subscription

订阅包含一个主题过滤器（Topic Filter）和一个最大的服务质量（QoS）等级。订阅与单个会话（Session）关联。会话可以包含多于一个的订阅。会话的每个订阅都有一个不同的主题过滤器。

## 发布 Publish

从客户端向服务端或者服务端向客户端传输一个带主题过滤器（Topic Filter）和一个最大的服务质量（QoS）等级的消息。

# MeOcer通讯协议简介

MeOcer会为每一个网内实体（Client）分配一个唯一的身份ID（UUID）和认证令牌（TOKEN），通过UUID和TOKEN客户端Client可以连接到MeOcer的M2M网络中，连接建立后便可以向MeOcer或者网络内对等通讯实体发送消息，或者接收对等实体发送的消息。

MQTT Client基本通讯流程：

1. Connect：使用username（UUID）和password（TOKEN）连接服务器（MeOcer）建立通讯会话。
2. Subscribe：以Client的UUID作为topic订阅消息，接收（Receive）来自MeOcer网络的消息。
3. Publish：发布一个带topic的消息，向MeOcer网络发送（Send）消息。MeOcer支持的topic如下（具体意义详见，第3章:API功能说明）：
   1. message
   2. update
   3. whoami
   4. resetToken
   5. getPublicKey
   6. generateAndStoreToken

MQTT Client与MeOcer之间只有2个通讯接口：Publish与Subscribe，不同的主题（Topic）对应不同效载荷（MQTT Payload）。在MeOcer中数据交换都是异步的，信息的交互实质都是实体之间的消息的发送，Request与Response只是相对的概念，（Request/Response）通过callbackId对应起来。

MeOcer

Client

Send：{ callbackId，requestMessage}

Receive：{ callbackId，responseMessage}

图2：Client-to-MeOcer

MQTT Client与MQTT Client间的通讯是通过MeOcer桥接的，其实质还是MQTT Client与MeOcer通讯。Client间的信息交换（Request/Response）也是通过callbackId对应起来。如图2所示：

MeOcer

ClientA

ClientB

Send：{ callbackId，requestMessage}

Receive：{ callbackId，requestMessage}

Send：{ callbackId，reponseMessage}

Receive：{ callbackId，responseMessage}

图2：Client-to-Client

# API功能说明

由于MQTT Client与MeOcer的上行通讯接口只有Publish，因此其核心是Topic和对应的Payload。因此我们的API接口将根据Topic来划分，Payload为一个JSON数据对象。

## message

Client-to-Client消息传递，向某个或者某些Client发送消息。MQTT Client之间的通讯过程在MeOcer中被定义为相互之间的远程过程调用（RPC），即其通讯的实质是要调用对方的某个功能函数，并获得相应的结果。

### Request

* MQTT Topic: message
* MQTT Payload:

{

“devices”:[“<Other\_Client\_UUID>”], /\*必选，目标Client 的UUID，可以为多个\*/

“topic”:”RPC\_CALL”, /\*必选，通讯主题，**固定值RPC\_CALL\***/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*可选，消息回调UUID，如果无需回复则不填 \*/

“payload”:{ /\*必选，RPC有效载荷\*/

“cmdName”:”<NAME>”, /\*可选，功能函数名\*/

“cmdCode”:”<XXXXXX>”, /\*必选，功能码\*/

“parameters”:{} /\*必选，RPC输入参数，JSON对象\*/

}

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload：

{

“topic” : “message”,

“data” : {

“devices”:[“<Other\_Client\_UUID>”], /\*必选，目标Client 的UUID，可以为多个\*/

“topic”:”RPC\_BACK”, /\*必选，通讯主题，**固定值RPC\_BACK**\*/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*必填，消息回调UUID \*/

“payload”:{ /\*必选，RPC有效载荷\*/

“retCode”:”<XXXXXX>”, /\*必选，错误代码\*/

“discription”:”<DISCRIPTION> ”, /\*必选，错误信息\*/

“data”:{} /\*必选，RPC返回数据，JSON对象\*/

}

}

}

## Update

Client-to-MeOcer消息传递，更新云端Client自身或者子设备的属性。

### Request

* MQTT Topic: update
* MQTT Payload:{

“uuid” : ”<Client\_UUID>”, /\*必选，Client自身的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*可选，消息回调UUID，如果无需回复则不填 \*/

… /\*其他需要修改或者添加的属性\*/

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload:
  + 变更通知： /\*如果Client的属性发生变更，那么MeOcer会主动发送config事件，通知Client\*/

{

“topic” : “config”,

“data” : {…}，/\*Client对象，包含全部属性\*/

}

* + 结果通知：
    - 成功：

{

“topic” : “update”,

“data” : {}，

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

* + - 失败：

MQTT Payload:{

topic” : “error”,

“data” : {“message” : “error message”},

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

## whoami

Client-to-MeOcer消息传递，查询云端Client自身的属性。

### Request

* MQTT Topic: whoami
* MQTT Payload:{

“uuid” : ” <Client\_UUID>”, /\*必选，Client自身的UUID\*/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*必选，消息回调UUID \*/

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload:{

“topic” : “whoami”,

“data” : { /\*JSON对象，Client的所有属性\*/

“uuid” : ” <Client\_UUID>”，

… /\*其他属性\*/

}

}

## resetToken

Client-to-MeOcer消息传递，更新Client的认证令牌。

### Request

* MQTT Topic: resetToken
* MQTT Payload:{

“uuid” : ” <Client\_UUID>”, /\*必选，Client自身的UUID\*/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*必选，消息回调UUID \*/

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload:
  + 变更通知： /\*如果Client的属性发生变更，那么MeOcer会主动发送config事件，通知Client\*/

{

“topic” : “config”,

“data” : {…}，/\*Client对象，包含全部属性\*/

}

* + 结果通知：
    - 成功：

{

“topic” : “update”,

“data” : {

“uuid” : “<Client\_UUID>”,

“token” : “NEW\_TOKEN” /\*新的token\*/

}，

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

* + - 失败：

MQTT Payload:{

topic” : “error”,

“data” : {“message” : “error message”},

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

## generateAndStoreToken

Client-to-MeOcer消息传递，新增Client的认证令牌。

### Request

* MQTT Topic: generateAndStoreToken
* MQTT Payload:{

“uuid” : ” <Client\_UUID>”, /\*必选，Client自身的UUID\*/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*必选，消息回调UUID \*/

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload:
  + 变更通知： /\*如果Client的属性发生变更，那么MeOcer会主动发送config事件，通知Client\*/

{

“topic” : “config”,

“data” : {…}，/\*Client对象，包含全部属性\*/

}

* + 结果通知：
    - 成功：

{

“topic” : “update”,

“data” : {

“uuid” : “<Client\_UUID>”,

“token” : “NEW\_TOKEN” /\*新的token\*/

}，

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

* + - 失败：

MQTT Payload:{

topic” : “error”,

“data” : {“message” : “error message”},

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

## getPublicKey

Client-to-MeOcer消息传递，获取公钥。

### Request

* MQTT Topic: getPublicKey
* MQTT Payload:{

“uuid” : ” <Client\_UUID>”, /\*必选，Client自身的UUID\*/

“fromUuid”:” <Client\_UUID>”, /\*可选，源Client的UUID\*/

“callbackId”:”<Callback\_UUID>”, /\*必选，消息回调UUID \*/

}

### Response

* MQTT Topic: “<Client\_UUID>” /\*Client 自身的UUID\*/
* MQTT Payload:
  + 结果通知：
    - 成功：

{

“topic” : “update”,

“data” : {

"publicKey" : “<PUBLIC\_KEY >”

}，

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}

* + - 失败：

MQTT Payload:{

topic” : “error”,

“data” : {“message” : “error message”},

“\_request” : { … } /\*JSON对象，Request payload的数据\*/

}