阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

# AIoT物联网开发实战(上)

快速上手MQTT协议、设备上云、消息发送/接收实战

藏经阁

阿里云开发者电子书系列


钉钉扫一扫加入
AIoT 物联网平台群

阿里云开发者“藏经阁”
海量免费电子书下载

# 作者简介

## 作者简介

苏堤嘉木，《 IoT 物联网技术 》专栏作者之一，从事云计算、大数据和 IoT 物联网领域技术布道。

## 推荐语

本电子书由阿里云开发者社区基于《 IoT 物联网技术 》专栏中的技术文章整理成电子书，方便广大 IoT 物联网开发者学习和掌握基于阿里云 IoT 物联网平台技术的开发，加速 IoT 企业业务落地。

# 目录

# 设备接入

# MQTT 协议与 IoT 物联网平台

作者 | 苏堤嘉木

## 一、MQTT 协议介绍

### 1. MQTT 协议

MQTT（消息队列遥测传输） 是基于 TCP/IP 协议栈而构建的支持在各方之间异步通信的消息协议。MQTT 在空间和时间上将消息发送者与接收者分离，因此可以在不可靠的网络环境中进行扩展。虽然叫做消息队列遥测传输，但它与消息队列毫无关系，而是使用了发布和订阅(Pub/Sub)的模型。

MQTT 是一种轻量级的、灵活的网络协议，致力于为 IoT 开发人员实现适当的平衡：

- 这个轻量级协议可在严重受限的设备硬件和高延迟/带宽有限的网络上实现。
- 它的灵活性使得为 IoT 设备和服务的多样化应用场景提供支持成为可能。


### 2. MQTT Client 库

MQTT Client 库在很多语言中都有实现，包括 Embedded C、C、Java、JavaScript、Python、C++、C#、Go、iOS、Android 等。

Eclipse Paho 的 MQTT 库下载地址：
https://www.eclipse.org/paho/downloads.php

MQTT Clients

| Client | Official Release | Unstable | GitHub |
|---|---|---|---|
| Java | 1.1.1 - Maven Central | 1.1.2-SNAPSHOT - Eclipse | https://github.com/eclipse/paho.mqtt.java |
| Python | 1.3.0 - Pypi (Pip) | *Build from develop branch* | https://github.com/eclipse/paho.mqtt.python |
| JavaScript | 1.0.3 - Eclipse | 1.0.4-SNAPSHOT - *Build from develop branch* | https://github.com/eclipse/paho.mqtt.javascript |
| Golang | 1.1.0 - Github repo tag v1.1.0 | `go get github.com/eclipse/paho.mqtt.golang` | https://github.com/eclipse/paho.mqtt.golang |
| C | 1.3.0 - Win32 / Win64 / Unix / Mac | *Build from master branch* | https://github.com/eclipse/paho.mqtt.c |
| C++ | 1.0.0 - *Build from source* | *Build from master branch* | https://github.com/eclipse/paho.mqtt.cpp |
| Rust | *Comming soon* | *Build from develop branch* | https://github.com/eclipse/paho.mqtt.rust |
| .Net (C#) | 4.3.0 - NuGet | *Build from master branch* | https://github.com/eclipse/paho.mqtt.m2mqtt |
| Android Service | 1.1.1 - Eclipse | 1.1.2-SNAPSHOT - Eclipse | https://github.com/eclipse/paho.mqtt.android |
| Embedded C/C++ | 1.1.0 - *Build from source* / Arduino | *Build from master branch* | https://github.com/eclipse/paho.mqtt.embedded-c |

## 3. MQTT 报文


固定报头 Fixed header

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| byte 1 | MQTT控制报文的类型 | | | | 用于指定控制报文类型的标志位 | | | |
| byte 2,3,4,5 | 剩余长度，最大4个字节 | | | | | | | |

控制报文类型:

| 名字 | 值 | 报文流动方向 | 描述 |
| --- | --- | --- | --- |
| Reserved | 0 | 禁止 | 保留 |
| CONNECT | 1 | Client -> Broker | device连接IoT平台 |
| CONNACK | 2 | Broker -> Client | IoT平台确认连接结果 |
| PUBLISH | 3 | 双向 | 发布消息 |
| PUBACK | 4 | 双向 | QoS=1消息发布收到确认 |
| ~~PUBREC~~ | ~~5~~ | ~~双向~~ | ~~IoT不支持~~ |
| ~~PUBREL~~ | ~~6~~ | ~~双向~~ | ~~IoT不支持~~ |
| ~~PUBCOMP~~ | ~~7~~ | ~~双向~~ | ~~IoT不支持~~ |
| SUBSCRIBE | 8 | Client -> Broker | device订阅IoT平台Topic |
| SUBACK | 9 | Broker -> Client | IoT平台确认订阅结果 |
| UNSUBSCRIBE | 10 | Client -> Broker | device取消订阅IoT平台Topic |
| UNSUBACK | 11 | Broker -> Client | IoT平台确认取消订阅结果 |
| PINGREQ | 12 | Client -> Broker | device发送心跳请求到IoT平台 |
| PINGRESP | 13 | Broker -> Client | IoT平台响应device心跳 |
| DISCONNECT | 14 | Client -> Broker | device断开IoT平台连接 |
| Reserved | 15 | 禁止 | 保留 |

控制报文类型标志位:

| 控制报文 | 固定报头标志 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| --- | --- | --- | --- | --- | --- |
| PUBLISH | MQTT 3.1.1使用 | DUP | QoS | QoS | RETAIN |

剩余长度:

| 字节数 | 最小值 | 最大值 |
| --- | --- | --- |
| 1 | 0 (0x00) | 127 (0x7F) |
| 2 | 128 (0x80, 0x01) | 16,383 (0xFF, 0x7F) |
| 3 | 16,384 (0x80, 0x80, 0x01) | 2,097,151 (0xFF, 0xFF, 0x7F) |
| 4 | 2,097,152 (0x80, 0x80, 0x80, 0x01) | 268,435,455 (0xFF, 0xFF, 0xFF, 0x7F) |

**注：阿里云IoT的单个payload最大256K**

### 可变报头 Variable header

某些 MQTT 控制报文包含一个可变报头部分。它在固定报头和负载之间。可变报头的内容根据报文类型的不同而不同。可变报头的报文标识符（Packet Identifier）字段存在于在多个类型的报文里。

报文标识符字节 Packet Identifier bytes:

| Bit | 7 - 0 |
| --- | --- |
| byte 1 | 报文标识符 MSB |
| byte 2 | 报文标识符 LSB |

| 控制报文 | 报文标识符 |
| --- | --- |
| PUBLISH | 需要（如果QoS =1，2） |
| PUBACK | 需要 |
| PUBREC | 需要 |
| PUBREL | 需要 |
| PUBCOMP | 需要 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |
| UNSUBACK | 需要 |

**有效载荷 Payload**

以下 MQTT 控制报文在报文的最后部分包含一个有效载荷。对于 PUBLISH 来说有效载荷就是业务消息。

| 控制报文 | 有效载荷 |
| --- | --- |
| CONNECT | 需要 |
| PUBLISH | 可选 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |

## 二、与阿里云 IoT 平台建立连接


### 1. CONNECT

阿里云 IoT 物联网平台的 MQTT 协议不支持 will 消息，CONNECT 消息内容参数如下：

| 参数 | 说明 |
| --- | --- |
| cleanSession | 此标志指定连接是否是持久性的。<br>0为持久会话，QoS=1消息不会丢失；<br>1为非持久会话，清理离线消息。 |
| clientId | 客户端标识符 |
| username | 代理的身份验证和授权凭证。 |
| password | 代理的身份验证和授权凭证。 |
| keepAlive | 心跳时间,**IoT平台约定心跳范围 30s~1200s** |

其中 clientId，username，password 由设备三元组(productKey,deviceName,deviceSecret)按照规则生成，具体规则如下：

| | | |
| --- | --- | --- |
| **clientId** | **id**+"\|securemode=3,signmethod=hmacsha1,timestamp="+timestamp+"\|" | **id**：表示客户端ID，64字符内。其中\|\|内为扩展参数<br>securemode：安全模式;2为TLS加密，3为非加密<br>signmethod：签名算法类型。<br>timestamp：当前时间毫秒值。 |
| **username** | deviceName+"&"+productKey | |
| **password** | sign_hmac(deviceSecret,content) | **sign_hmac**为clientId中的signmethod算法类型<br>**content**为如下拼接字符串：<br>"**clientId**${id}**deviceName**${deviceName}**productKey**${productKey}**timestamp**${timestamp}" |

官方文档：https://help.aliyun.com/document_detail/73742.html

**设备端代码示例(Nodejs 版) client.js**

```
    /**
    "dependencies": { "mqtt": "2.18.8" }
    */
const crypto = require('crypto');
const mqtt = require('mqtt');

//设备身份三元组+区域 const deviceConfig = {
        productKey: "替换",
        deviceName: "替换",
        deviceSecret: "替换",
        regionId: "cn-shanghai"
};

//根据三元组生成 mqtt 连接参数
const options = initMqttOptions(deviceConfig);
const url = `tcp://${deviceConfig.productKey}.iot-as-mqtt.${deviceConfig.regionId}.aliyuncs.
com:1883`;

//2.建立连接
const client = mqtt.connect(url, options);

client.on('packetsend', function (packet){
      console.log('send '+packet.cmd+' packet =>',packet)
})

client.on('packetreceive', function (packet){
      console.log('receive '+packet.cmd+' packet =>',packet)
})

//IoT 平台 mqtt 连接参数初始化
function initMqttOptions(deviceConfig) {
        const params = {
            productKey: deviceConfig.productKey,
            deviceName: deviceConfig.deviceName,
            timestamp: Date.now(),
            clientId: Math.random().toString(36).substr(2),
        }
```

```
        //CONNECT 参数
        const options = {
            keepalive: 60, //60s
            clean: false, //cleanSession 保持持久会话
            protocolVersion: 4 //MQTT v3.1.1
        }
        //1.生成 clientId，username，password
        options.password = signHmacSha1(params, deviceConfig.deviceSecret);
        options.clientId = `${params.clientId}|securemode=3,signmethod=hmacsha1,timesta
mp=${params.timestamp}|`;
        options.username = `${params.deviceName}&${params.productKey}`;

        return options;
}

/*
      生成基于 HmacSha1 的 password
      参考文档：https://help.aliyun.com/document_detail/73742.html?#h2-url-1
    */
function signHmacSha1(params, deviceSecret) {
        let keys = Object.keys(params).sort();
        // 按字典序排序
        keys = keys.sort();
        const list = [];
        keys.map((key) => {
            list.push(`${key}${params[key]}`);
        });
        const contentStr = list.join('');
        return crypto.createHmac('sha1', deviceSecret)
                .update(contentStr)
                .digest('hex');
}
```

## 2. CONNACK

```
    receive connack packet => Packet {
      cmd: 'connack',
      retain: false,
      qos: 0,
      dup: false,
      length: 2,
      topic: null,
```

```
  payload: null,
  sessionPresent: false,
  returnCode: 0 }
```

### 3. PINGRESP

```
send pingreq packet => { cmd: 'pingreq' }
```

### 4. PINGRESP

```
receive pingresp packet => Packet {
  cmd: 'pingresp',
  retain: false,
  qos: 0,
  dup: false,
  length: 0,
  topic: null,
  payload: null }
```

### 5. DISCONNECT


## 三、发布数据

### 1. PUBLISH

```
//3.属性数据上报 const topic = `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/property/post`;

setInterval(function() {
    //发布数据到 topic
    client.publish(topic, getPostData(),{qos:1});
```

```
}, 5 * 1000);

function getPostData() {
        const payloadJson = {
            id: Date.now(),
            params: {
                temperature: Math.floor((Math.random() * 20) + 10),
                humidity: Math.floor((Math.random() * 20) + 60)
            },
            method: "thing.event.property.post"
        }
        console.log("===postData\n topic=" + topic)
        console.log(payloadJson)

        return JSON.stringify(payloadJson)
}
    send publish packet => { cmd: 'publish',
      topic: '/sys/a1hQSwFledE/eud1jXfEgCsAiP2eId9Q/thing/event/property/post',
      payload: '{"id":1543896481106,"params":{"temperature":23,"humidity":73},"method":"thing.
event.property.post"}',
      qos: 1,
      retain: false,
      messageId: 38850,
      dup: false }
```

### 2. PUBACK

```
    receive puback packet => Packet {
      cmd: 'puback',
      retain: false,
      qos: 0,
      dup: false,
      length: 2,
      topic: null,
      payload: null,
      messageId: 38850 }
```

## 四、接收数据

### 1. SUBSCRIBE

```
    //4.订阅主题,接收指令 const subTopic = `/${deviceConfig.productKey}/${deviceConfig.devi
ceName}/control`;

client.subscribe(subTopic)
client.on('message', function(topic, message) {
        console.log("topic " + topic)
        console.log("message " + message)
})
    SUBSCRIBE 消息体
    send subscribe packet => { cmd: 'subscribe',
      subscriptions:
       [ { topic: '/a1hQSwFledE/eud1jXfEgCsAiP2eId9Q/control', qos: 0 } ],
      qos: 1,
      retain: false,
      dup: false,
      messageId: 38851 }
```

### 2. SUBACK

SUBACK 消息体:

```
    receive suback packet => Packet {
      cmd: 'suback',
      retain: false,
      qos: 0,
      dup: false,
      length: 3,
      topic: null,
      payload: null,
      granted: [ 128 ],
      messageId: 38851 }
```

### 3. UNSUBSCRIBE

```
    send unsubscribe packet => { cmd: 'unsubscribe',
      qos: 1,
      messageId: 34323,
      unsubscriptions: [ '/a1hQSwFledE/eud1jXfEgCsAiP2eId9Q/control' ] }
```

### 4. UNSUBACK

```
receive unsuback packet => Packet {
  cmd: 'unsuback',
  retain: false,
  qos: 0,
  dup: false,
  length: 2,
  topic: null,
  payload: null,
  messageId: 34323 }
```

## 五、服务质量 QoS

| 服务质量<br>Quality of Service | 描述 | 阿里云IoT |
| --- | --- | --- |
| QoS=0 | 最多一次的传输，可能会收不到消息 | 支持 |
| QoS=1 | 至少一次的传输，一定会收到消息，可能重复 | 支持 |
| QoS=2 | 有且仅有一次的传输 | 不支持 |


## 六、设备掉线重连

设备与阿里云 IoT 的订阅关系在云端保持，除非设备主动 unsubscribe，否则订阅关系不清理。设备重连后，依然保持之前的订阅关系，不需要重复订阅。

## 七、传输层安全 TLS v1.2

设备和 IoT 平台之间的链路可以通过 TLS v1.2 加密。

如果使用 TLS 加密，需要下载根证书。

CONNECT 参数中 clientId 的 securemode=2

https://help.aliyun.com/document_detail/73742.html

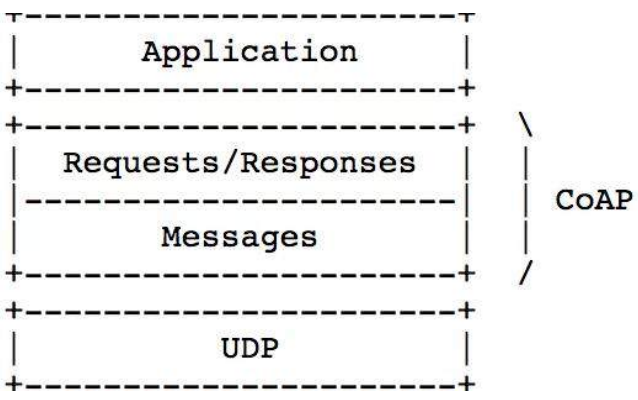
# CoAP 协议详解

作者 | IoT 物联网技术


http://coap.technology

CoAP 是受限制的应用协议(Constrained Application Protocol)的缩写。在 IoT 物联网场景，为了让小设备可以接入互联网，CoAP 协议被设计出来。CoAP 是一种应用层协议，它运行于 UDP 协议之上而不是像 HTTP 那样运行于 TCP 之上。CoAP 协议非常小巧，最小的数据包仅为 4 字节。

CoAP 是一种面向网络的协议，采用了与 HTTP 类似的特征，核心内容为资源抽象、REST 式交互以及可扩展的头选项等。为了克服 HTTP 对于受限环境的劣势，CoAP 既考虑到数据报长度的最优化，又考虑到提供可靠通信。一方面，CoAP 提供 URI，REST 式的方法如 GET，POST，PUT 和 DELETE，以及可以独立定义的头选项提供的可扩展性。另一方面，CoAP 基于轻量级的 UDP 协议，并且允许 IP 多播。为了弥补 UDP 传输的不可靠性，CoAP 定义了带有重传机制的事务处理机制。并且提供资源发现机制，并带有资源描述。



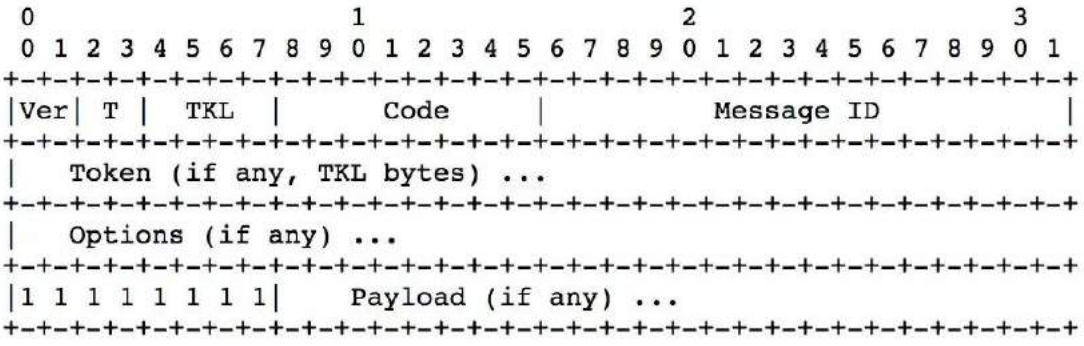
**协议特点：**

- 基于消息模型
- 请求/响应模型
- 双向通信
- 轻量、低功耗
- 支持可靠传输
- 支持 IP 多播
- 非长连接通信，支持受限设备
- 支持观察模式
- 支持异步通信

**协议内容：**

CoAP 是一个完整的二进制应用层协议，消息格式紧凑，默认运行在 UDP 上。

**CoAP 报文：**




- 【Ver】版本编号。
- 【T】报文类型，CoAP 协议定了 4 种不同形式的报文，CON 报文，NON 报文，ACK 报文和 RST 报文。
- 【TKL】CoAP 标识符长度。CoAP 协议中具有两种功能相似的标识符，一种 Message ID(报文编号)，一种为 Token(标识符)。其中每个报文均包含消息编号，但是标识符对于报文来说是非必须的。

- 【Code】功能码/响应码。Code 在 CoAP 请求报文和响应报文中具有不同的表现形式，Code 占一个字节，它被分成了两部分，前 3 位一部分，后 5 位一部分，为了方便描述它被写成了 c.dd 结构。其中 0.XX 表示 CoAP 请求的某种方法，而 2.XX、4.XX 或 5.XX 则表示 CoAP 响应的某种具体表现。
- 【Message ID】报文编号。
- 【Token】标识符具体内容，通过 TKL 指定 Token 长度。
- 【Option】报文选项，通过报文选项可设定 CoAP 主机，CoAP URI，CoAP 请求参数和负载媒体类型等等。
- 【1111 1111B】CoAP 报文和具体负载之间的分隔符。

**请求方法：**

- 0.01 GET：获取资源
- 0.02 POST：创建资源
- 0.03 PUT：更新资源
- 0.04 DELETE：删除资源

**响应码：**

Success 2.xx

这一类型的状态码，代表请求已成功被服务器接收、理解、并接受。

- 2.01 Created
- 2.02 Deleted
- 2.03 Valid
- 2.04 Changed
- 2.05 Content

Client Error 4.xx：

这类的状态码代表了客户端看起来可能发生了错误，妨碍了服务器的处理。

- 4.00 Bad Request
- 4.01 Unauthorized
- 4.02 Bad Option
- 4.03 Forbidden
- 4.04 Not Found
- 4.05 Method Not Allowed
- 4.06 Not Acceptable
- 4.12 Precondition Failed
- 4.13 Request Entity Too Large
- 4.15 Unsupported Content-Format

Server Error 5.xx:

这类状态码代表了服务器在处理请求的过程中有错误或者异常状态发生,也有可能是服务器的软硬件资源无法完成对请求的处理。

- 5.00 Internal Server Error
- 5.01 Not Implemented
- 5.02 Bad Gateway
- 5.03 Service Unavailable

**媒体类型:**

| Media type | Encoding | ID |
|---|---|---|
| text/plain; charset=utf-8 | - | 0 |
| application/link-format | - | 40 |
| application/xml | - | 41 |
| application/octet-stream | - | 42 |
| application/exi | - | 47 |
| application/json | - | 50 |

**工作模式：**

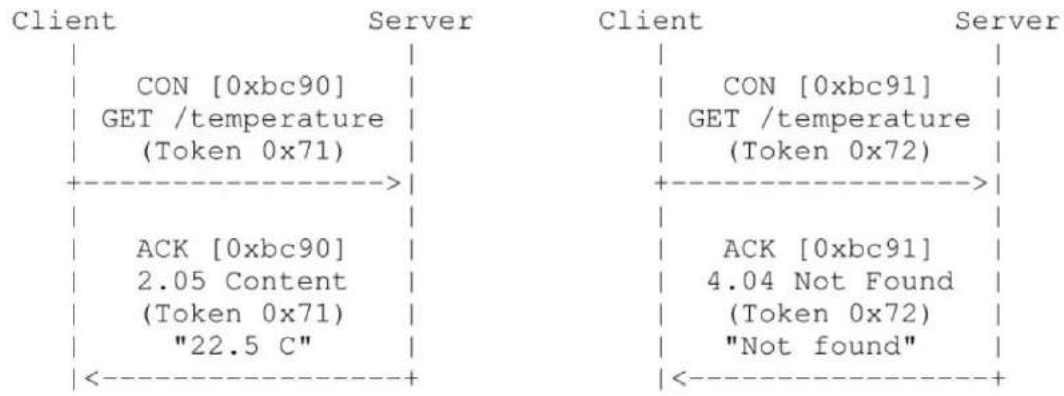
CoAP 参考了很多 HTTP 的设计思路，同时也根据受限资源限制设备的具体情况改良了诸多的设计细节，增加了很多实用的功能。

消息类型

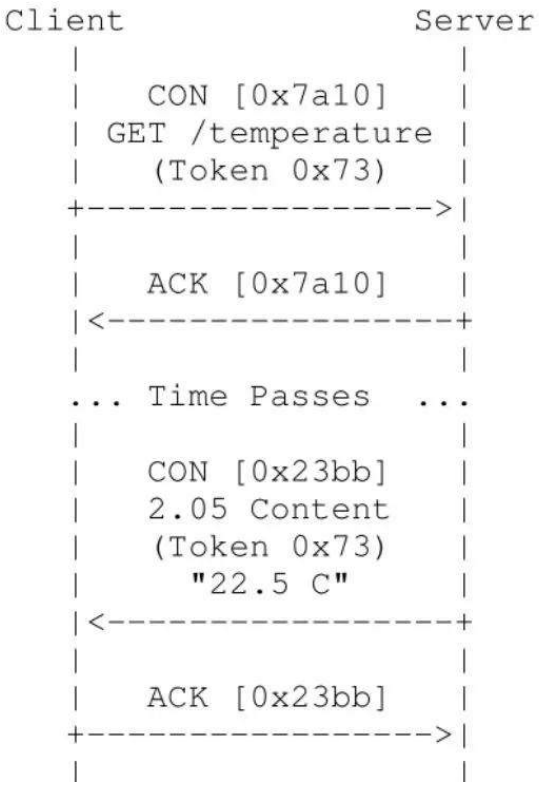
- CON：需要被确认的请求，如果 CON 请求被发送，那么对方必须做出响应。
- NON：不需要被确认的请求，如果 NON 请求被发送，那么对方不必做出回应。
- ACK：应答消息，接受到 CON 消息的响应。
- RST：复位消息，当接收者接收到的消息包含一个错误，接收者解析消息或者不再关心发送者发送的内容，那么复位消息将会被发送。

**请求/响应模型：**

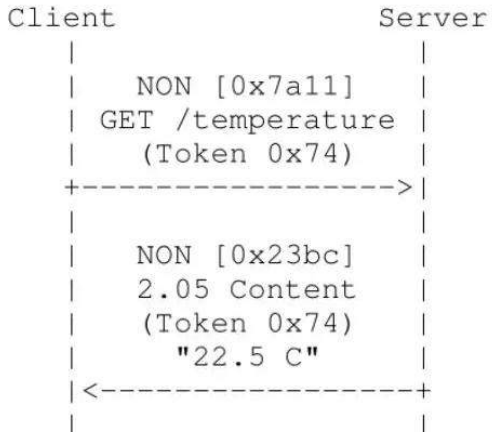
1.携带模式




2.分离模式




3.非确认模式




**CoAP 协议的 URI:**

在 HTTP 的世界中，正式 RESTFul 协议由于其简单性和适用性，在 WEB 应用中越来越受欢迎，这样的道理同样适用于 CoAP。一个 CoAP 资源可以被一个 URI 所描述，例如一个设备可以测量温度，那么这个温度传感器的 URI 被描述为：CoAP://machine.address:5683/sensors/temperature。

**CoAP 的默认 UDP 端口号为 5683**

**HTTP 和 CoAP 对比：**

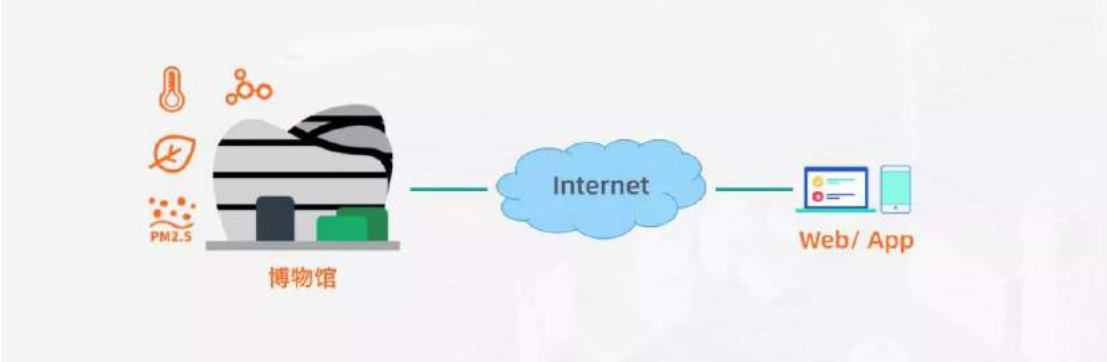
- HTTP 代表超文本传输协议，CoAP 代表约束应用协议；
- HTTP 协议的传输层采用了 TCP，CoAP 协议的传输层使用 UDP；
- CoAP 协议是 HTTP 协议的简化版；
- CoAP 协议和 HTTP 协议一样使用请求/响应模型，拥有相同的方法；
- CoAP 开销更低，并支持多播；
- CoAP 专为资源构成应用而设计，如：IoT/WSN/M2M 等...

**CoAP 和 MQTT 对比：**

- MQTT 协议使用发布/订阅模型，CoAP 协议使用请求/响应模型；
- MQTT 是长连接，CoAP 协议是无连接；
- MQTT 通过中间代理传递消息的多对多协议，CoAP 协议是 Server 和 Client 之间消息传递的单对单协议；
- MQTT 不支持带有类型或者其它帮助 Clients 理解的标签消息，CoAP 内置内容协商和发现支持，允许设备彼此窥测以找到交换数据的方式。

# IoT 设备上云方案详解

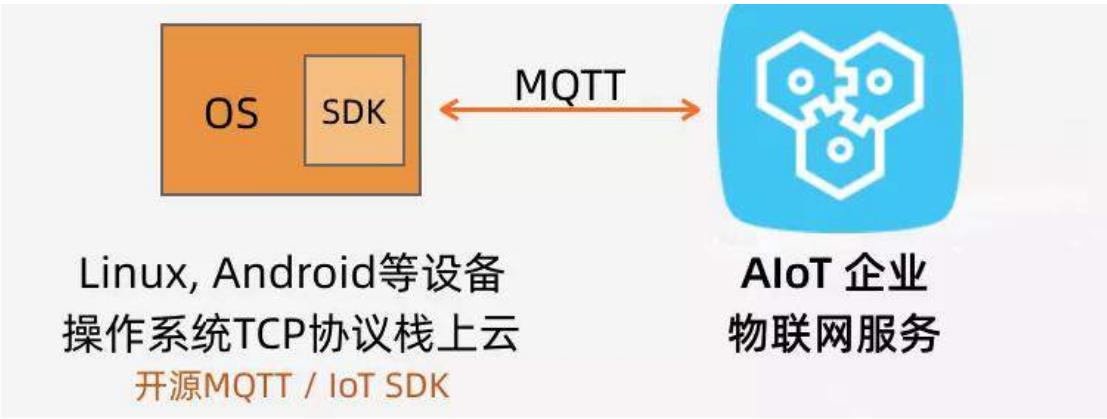
作者 | 苏堤嘉木




随着传感器和通信技术的不断发展，物联网行业方兴未艾，业务链路涉及**数据采集**，**通信连接**，**数据存储**，**数据可视化**，**洞察**，**行动决策**。但，在实施过程中，碎片化的设备端通信连接难题往往就阻碍了项目落地进程。

今天，本文总结不同设备场景的连接上云方案，供大家参考。

## 一、资源丰富类设备




高性能硬件的发展，很多智能设备带有完整的 Linux、Android、Arduino 等操作系统，在操作系统层面，解决了不同通信模块的差异，硬件端的应用程序**只需要集成云平台的 IoTSDK**，或者**集成开源 MQTT SDK** 即可和云端建立长连接通信链路。

## 二、资源受限类设备

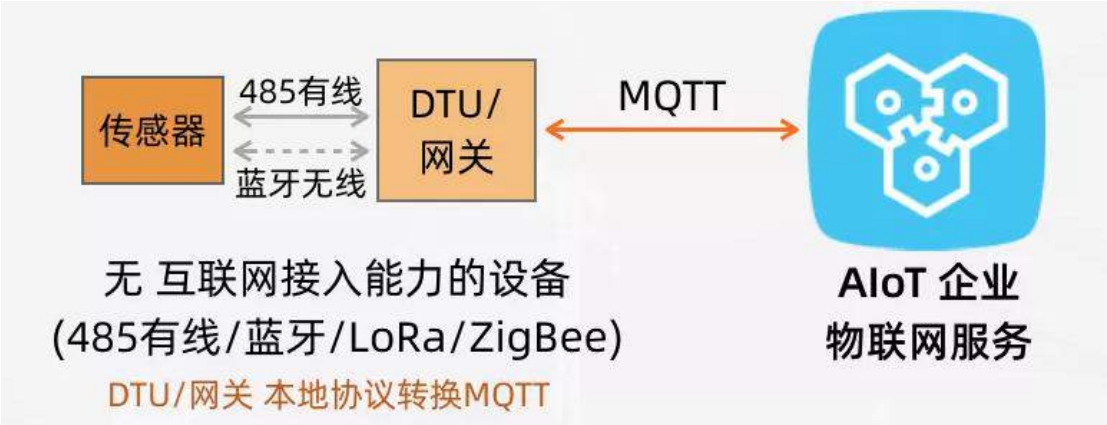

物联网场景中有很大占比设备是资源受限的，运行 ROTS 系统，甚至无操作系统，采用 MCU+通信模组的方式，实现设备数据远程采集。

市面上蜂窝模组(NB-IoT/2G/3G/4G)供应商较多，比如**移远通信**、**芯讯通**、**合宙**、**有方科技**、**广和通**、**日海智能**、**高新兴**等，而各家的 AT 指令也各不相同，为设备端应用程序开发带来了很大难度。

根据模组集成度不同又细分一下几种场景方案：

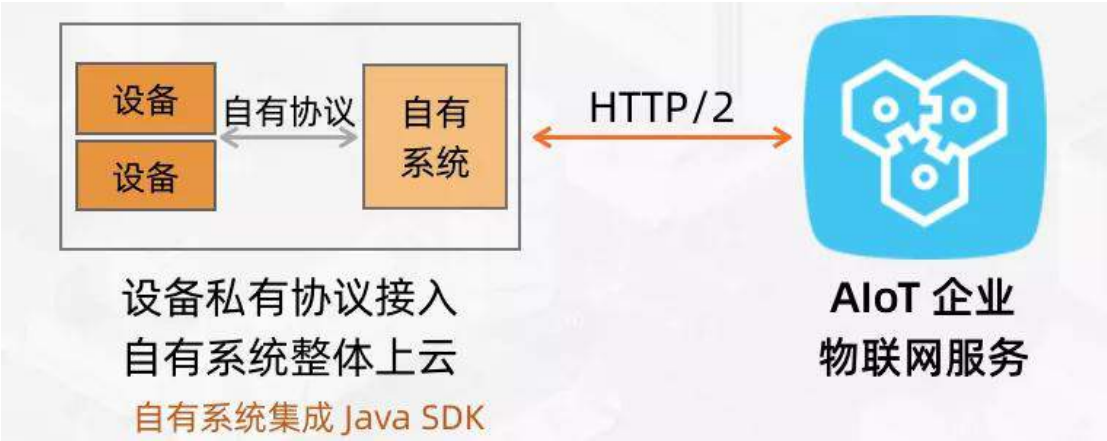
| 模组AT指令集成度 | MCU上应用程序开发 |
| --- | --- |
| 模组集成云平台 **IoT AT**指令 (比如 ALICONN) | 开发简单，仅需对接IoT平台的AT指令 |
| 模组支持 **MQTT**协议 AT指令 | 开发难度中等，需要对接IoT平台业务规则 |
| 模组支持 **TCP**协议 AT指令 | 开发难度高，需要实现MQTT协议和IoT平台业务规则 |

## 三、本地通信类设备




物联网场景中还有大量设备仅具有本地局域通信能力，比如蓝牙设备，ZigBee 设备，LoRa 设备，Modbus 设备，而不具有互联网接入协议栈支持，此时需要**借助 DTU/网关设备**，代理子设备把本地协议转换成 MQTT 协议，从而实现数据采集上云。

## 四、本地系统整体上云




在工业，商业综合体等场景中，本地往往有一套成熟的系统，实现了设备数据的集中采集，由于集团业务统一管理的诉求，需要把各地数据采集上云。

面对这种场景，可以通过自有系统集成**泛化 SDK 通过 HTTP/2 协议**，在不改造设备前提下，高效的实现海量数据快速上云。

# 电信 NB-IoT 无缝对接阿里云 IoT

作者 | 苏堤嘉木

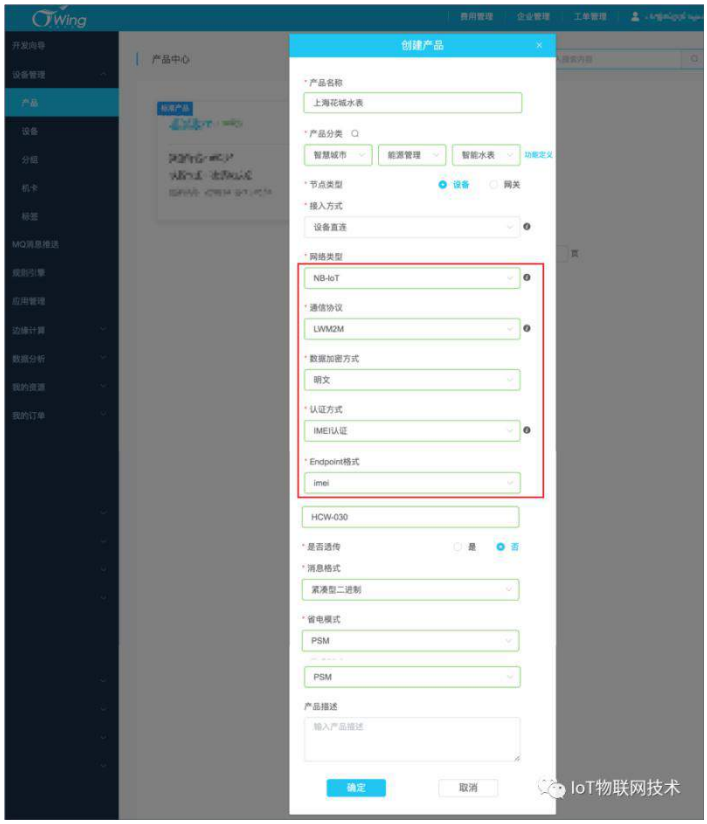
众所周知，中国电信的 **NB-IoT 设备必须直连电信 CTWing 平台**，无法直接在阿里云 IoT，AWS IoT，腾讯云 IoT 管理中国电信的 NB-IoT 设备。

**IoT 开发者苦于运营商平台久已**，阿里云终于出手了！今天我们就给大家介绍如何通过阿里云 IoT 企业物联网实例来管理中国电信的 NB-IoT 设备。

## 一、电信 CTWing 开发

### 1. 创建产品

登陆电信 CTWing 控制台，创建产品：**上海花城水表**，选择**智能水表**，其他配置信息如下图：



产品创建成功后，查看**服务定义**，可以看到属性值：**用水量**，标识为:water_consumption 如下图：


## 2. 注册设备

在**产品详情-设备管理**页面添加 NB-IoT 设备，这里我们输入 NB-IoT 水表设备的 IMEI，如下图：


## 3. 注册应用

在**应用管理**页面添加应用:**上海花城水表管理**，获取到应用的 AppKey 和 AppSecret，如下图：


## 二、 阿里云 IoT 平台开发

### 4. 开通企业实例-连接型

为了对接电信 CTWing 物联网平台，我们按需开通**连接型**企业物联网实例，如下图：


### 5. 创建产品

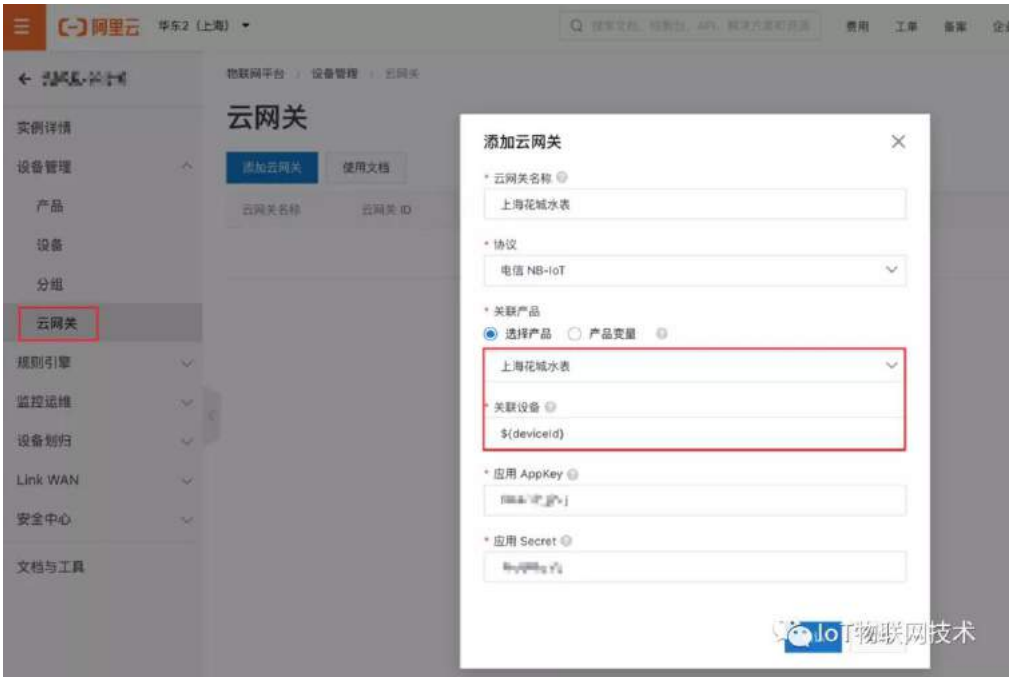
在连接型实例中，新建产品：**上海花城水表**，选择**智能水表**，，其中数据格式选择：**透传/自定义**，数据校验级别选择：**弱校验**，如下图：


在产品**功能定义**中添加物模型的属性：**用水量**，标识为:water_consumption 如下图：


在产品**数据解析**中我们需要编写数据解析脚本，把电信 CTWing 平台的数据格式转换成阿里云 IoT 企业实例的物模型格式，如下图：


## 6. 创建云网关

在设备管理 中创建**云网关**，用来解析电信平台流转过来的设备数据，关联前面创建的产品**上海花城水表**，用电信 CTWing 平台的**${deviceId}**来自动注册设备，如下图：



等待几分钟，云网关创建完成后，我们获取到网关 URL，如下图：


## 三、电信 IoT 配置云网关

### 7. 配置云网关

我们回到电信 CTWing 控制台，在产品详情的**订阅管理**中，配置阿里云 IoT 的云网关 URL，如下图：


## 四、联机运行

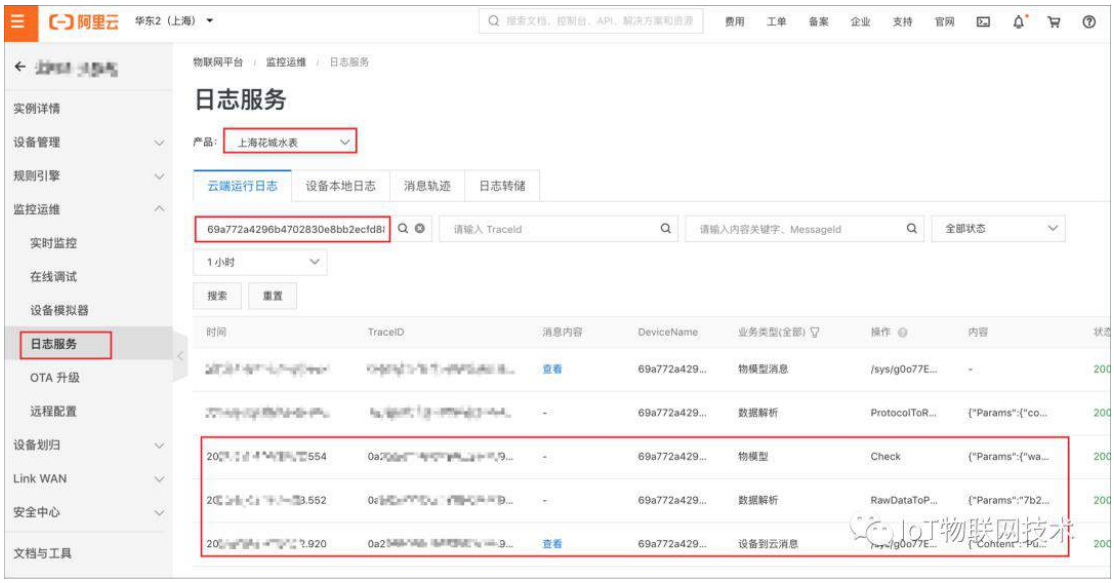
### 8. NB-IoT 设备启动

我们启动手上的 NB-IoT 智能水表，即可在电信 CTWing 控制台看到水表上报的数据，如下图：


## 9. 阿里云 IoT 数据

我们在**企业物联网平台**控制的**设备详情**，可以看到有一台新的在线设备，**物模型数据**的**运行状态**可以看到实时的**用水量**值，如下图：


在**日志服务**也可以看到电信 CTWing 平台流转过来的 NB-IoT 设备数据在阿里云 IoT 企业物联网平台物**模型解析**的完整过程，如下图：




NB-IoT 设备从电信 CTWing 平台流转过来的**完整数据报文**，如下图：


至此，我们完成了**电信 NB-IoT 设备接入阿里云 IoT 平台**，充分享受阿里云的海量存储，大数据计算能力，拓展 IoT 业务的无限可能。

# LoRaWAN 设备接入实战

作者 | 苏堤嘉木

随着 IoT 物联网的高速发展，**低功耗**，**远距离**，**抗干扰**的低功耗广域网快速崛起，LoRa 与 NB-IoT 就是其中的典型代表。

今天，我们给大家介绍 LoRaWAN 传感设备如何接入阿里云 IoT 企业物联网实例中。

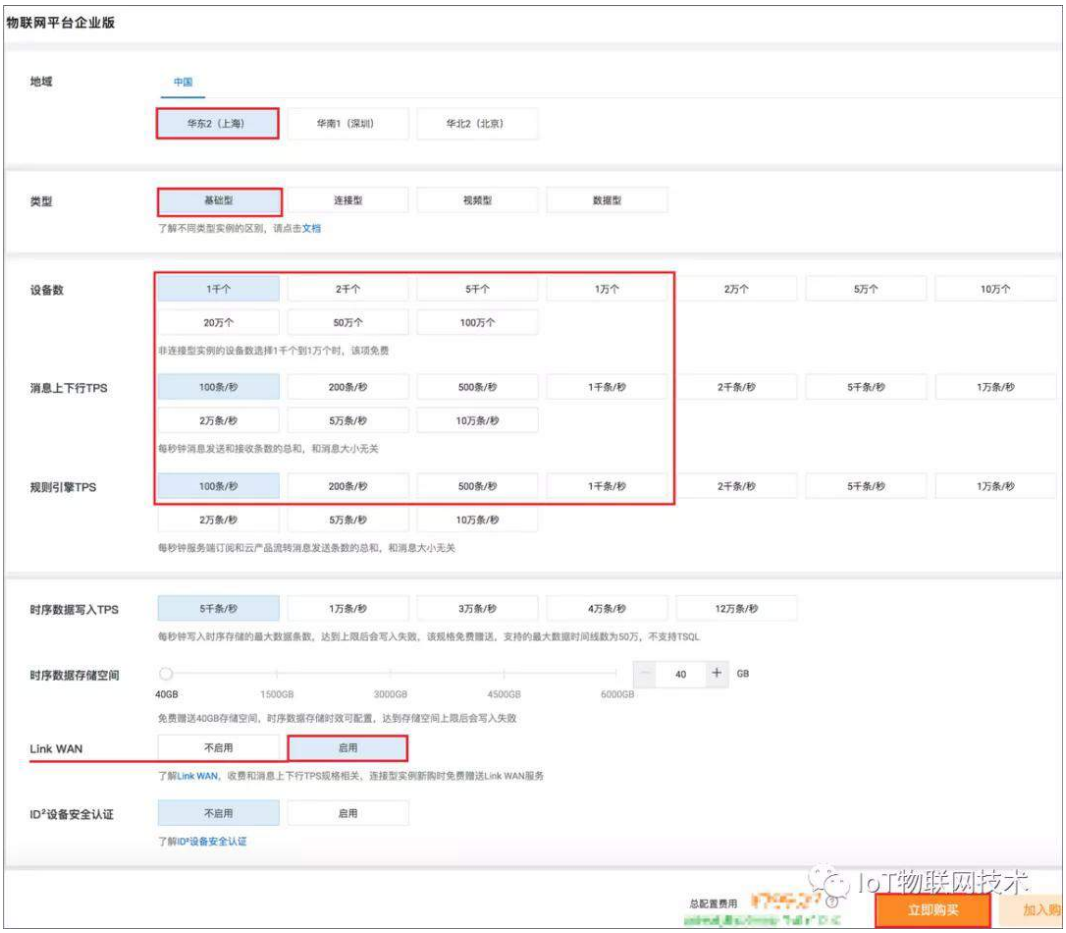
## 一、整体架构方案

阿里云企业物联网平台开启 LinkWAN 服务后，即可实现 LoRaWAN 设备的上云。我们以 LoRaWAN 液位传感器为例，完整方案如下：


## 二、阿里云 IoT 平台开发

### 1. 开通企业实例-LinkWAN 服务

首先，我们按需开通**基础型**企业物联网实例，开启 LinkWAN 服务，如下图：

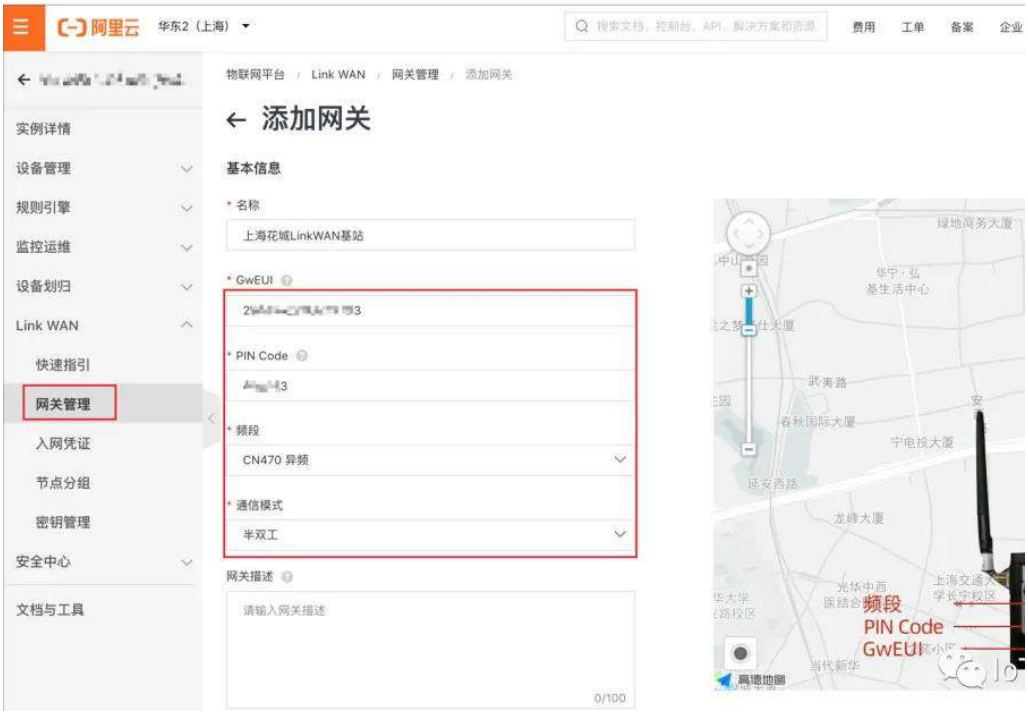



稍等几分钟，等待实例创建完成，可以查看完整实例信息，如下图：


## 2. 添加 LinkWAN 网关

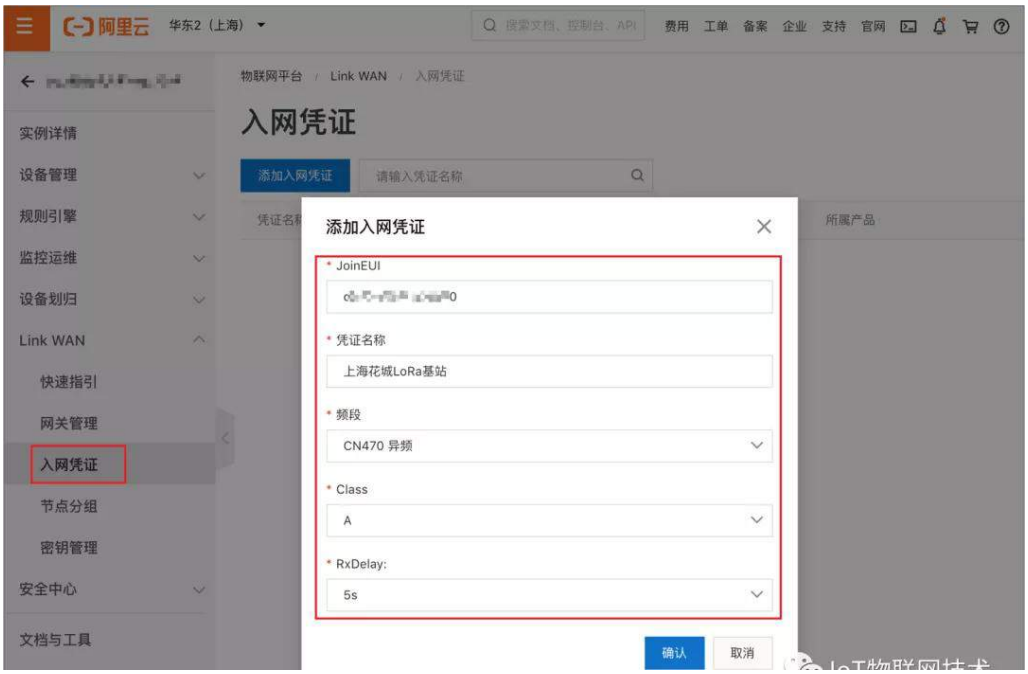
在 Link WAN 的网关管理页面，添加网关，录入手中 LinkWAN 网关设备的

GwEUI、PIN Code 和**频段**、**通信模式**，如下图：




### 3. 创建入网凭证

在 Link WAN 的**入网凭证**页面，添加入网凭证信息，如下图：




## 4. 创建产品

在**设备管理**的产品页面，创建 **LoRa 液位传感器**产品，选择 LoRaWAN 连网模式，并选择前面创建的 LoRa 入网凭证，如下图：

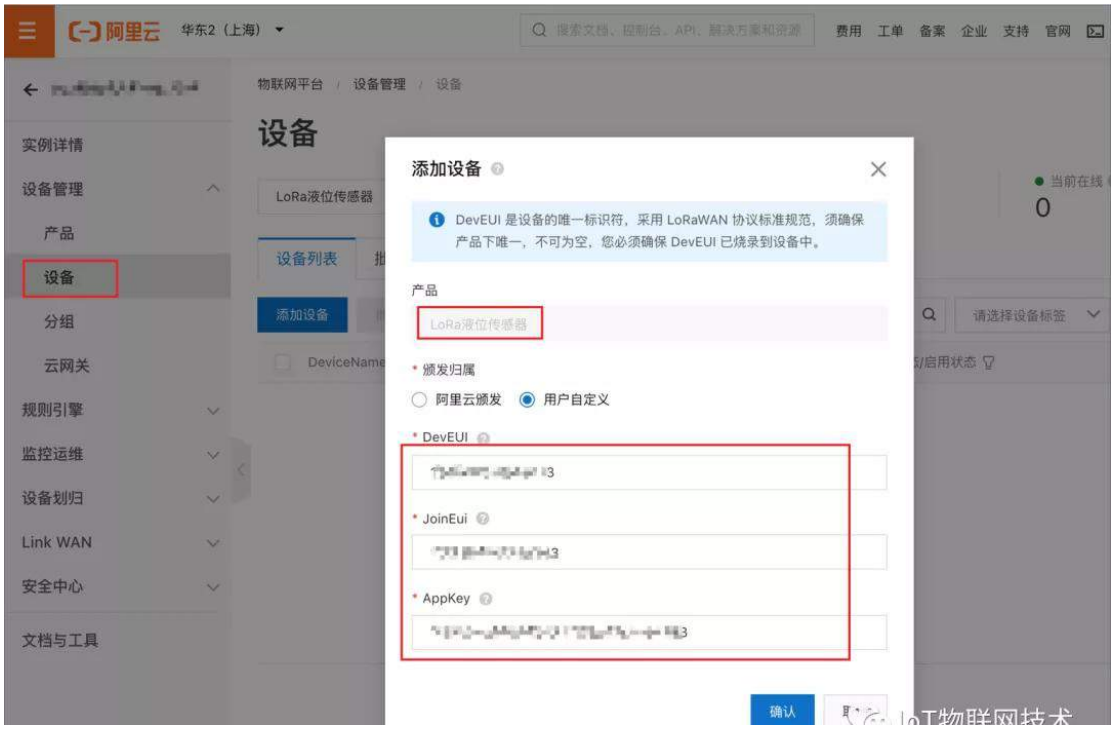
创建产品完成后，我们在 LinkWAN 的**入网凭证**列表可以看到 LoRa 产品和入网凭证的绑定关系，如下图：


在**产品详情**页面的**功能定义**，我们添加物模型-属性：当前液位，标识符:water_level，如下图：


在**产品详情**页面的**数据解析**，我们按照设备说明书编写数据解析脚本，如下图：


## 5. 注册设备

在**设备管理**的设备页面，添加 **LoRa 液位传感器**具体设备，录入手上 LoRa 设备的 DevEUI，JoinEui 和 AppKey 信息，如下图：



## 三、联机运行

启动 LinkWAN 网关，LoRa 液位传感器设备上电运行后，在企业物联网实例控制台，设备详情，我们可以看到 LoRa 设备处于在线状态，已经实时上报的**当前液位值**，如下图：

# 微信小程序 MQTT 模拟器

作者 | 苏堤嘉木


## 一、IoT 设备模拟器 小程序

微信小程序 **MQTT 模拟器**可以实现无代码开发，模拟设备接入阿里云 IoT 物联网平台设备连接，自定义 Topic 通信，物模型协议通信的完整过程，支持多个 region 接入。


点击原文进入小程序

## 二、创建产品和注册设备

①我们进入物联网平台，公共实例，创建如下产品：


②在产品详情页面，**功能定义**下添加**属性和事件**，如下图：

| 类型 | 名称 | 标识符 | 参数 |
| --- | --- | --- | --- |
| 属性 | 消息 | msg | |
| 事件 | 电量不足 | lowBattery | currentBattery |

③基于当前产品，**注册设备**，并获取**身份三元组**，用于设备连接时的身份认证，如下图：

# 三、MQTT 模拟器 使用

## 1. 设备上线

①打开微信，扫描进入小程序。

②输入**设备三元组**。

③选择设备所在的接入 region。

④点击**设备上线**。

⑤进入物联网平台控制台，设备详情，设备为**在线**状态。


## 2. 自定义 Topic 数据上报

①模拟设备发布传感器数据到/{pk}/{dn}/user/update 主题。

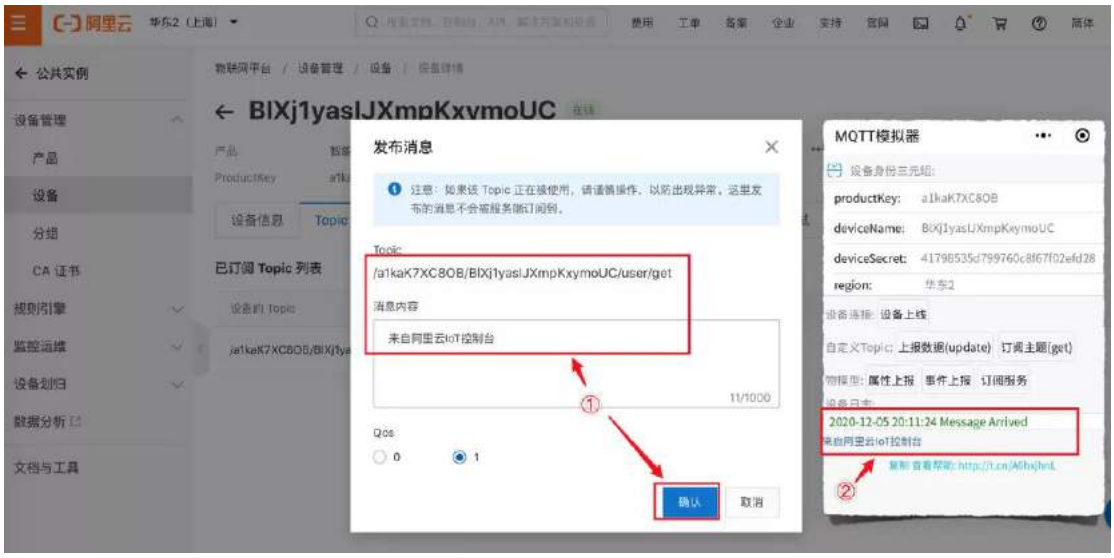
②在物联网平台控制台，日志服务验证数据上报：


### 3. 设备接收自定义 Topic 控制指令

① 模拟设备主动订阅主题 /{pk}/{dn}/user/get。

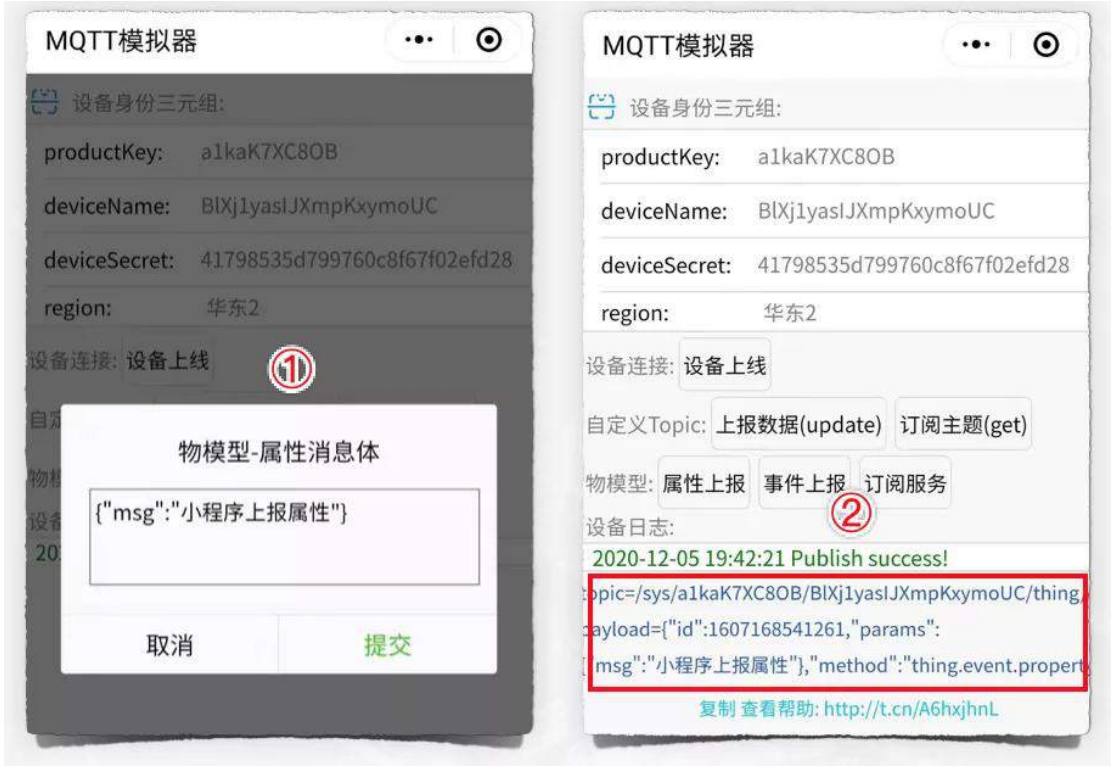
② 在物联网平台控制台，设备详情的 Topic 列表可以看到一条订阅记录。


③ 点击**发布消息**，输入内容，**确认**发布后，小程序会**实时展示收到的消息**。




### 4. 物模型–属性上报

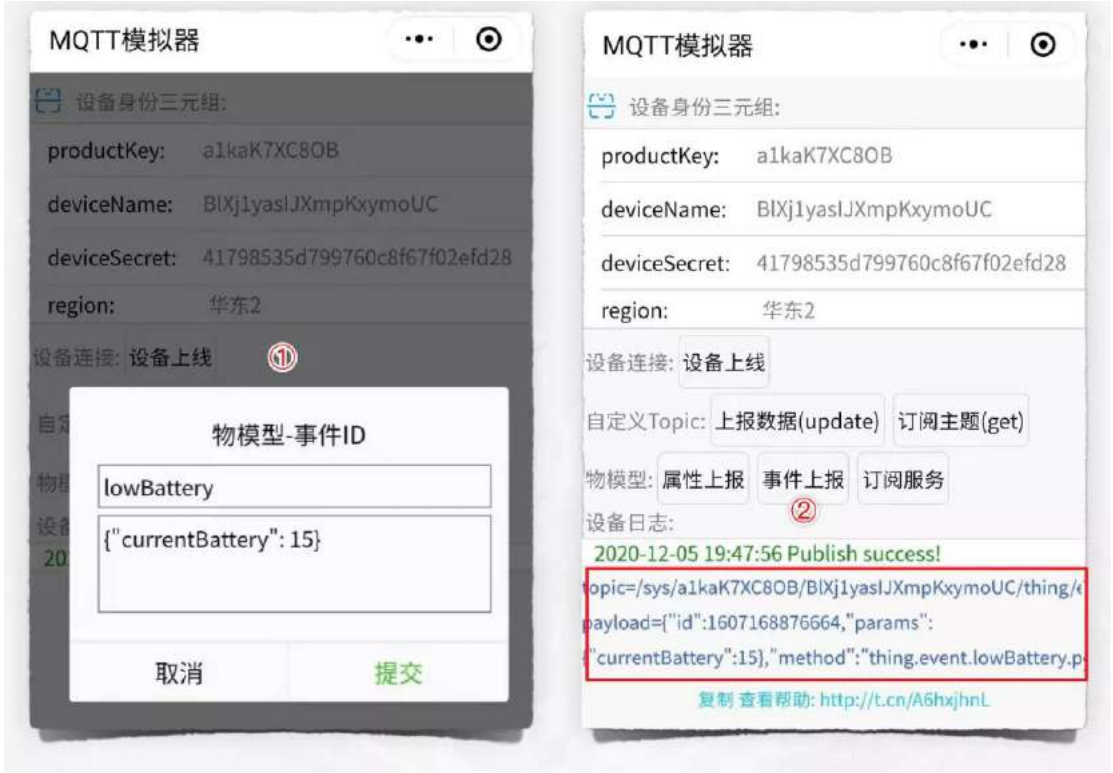
①点击**属性上报**，输入 JSON 结构的属性值。




②物联网平台控制台，**设备详情>物模型数据>运行状态**，查看设备上报属性的内容。


## 5. 物模型–事件上报

①点击事件上报，输入事件 Id 和 JSON 结构的事件参数。



②物联网平台控制台，**设备详情>物模型数据>事件管理**，查看设备上报的事件**标识符**合**输出参数**内容。


## 四、设备三元组 Chrome 插件

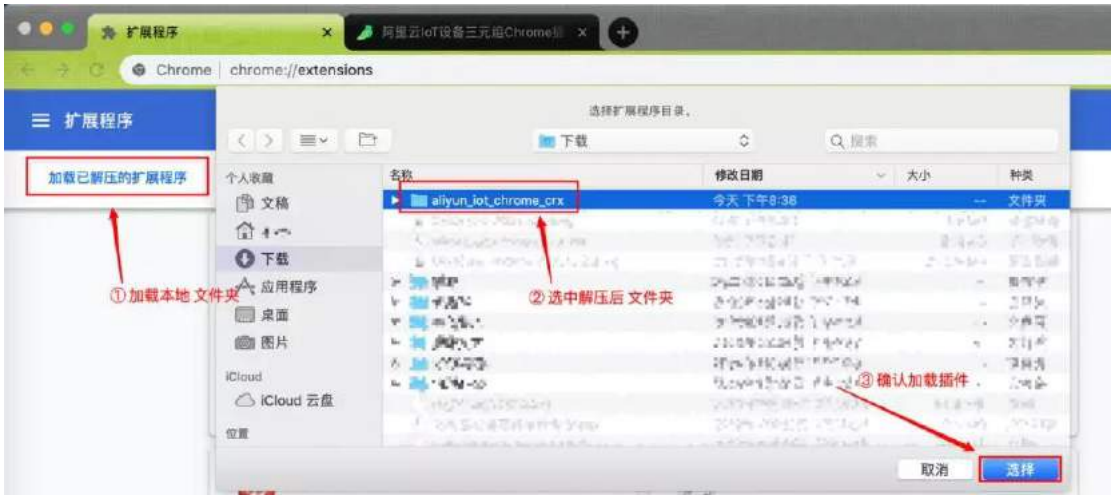
为了方便在小程序里输入设备身份二维码，推荐大家安装 IoT 设备身份三元组转化二维码的 Chrome 插件。点击阅读原文获取插件安装包。

### 1. 插件安装

① 打开 Chrome 浏览器，导航栏输入: chrome://extensions/

② 扩展程序设置页面右上角，开启 **开发者模式**。


③ 点击左上角**加载已解压的扩展程序**按钮，选中本地扩展程序文件夹，然后点击**选择**按钮。




④ IoT 设备身份管理插件安装完成，如下图：


## 2. 插件使用方式

①进入物联网平台控制台，进入设备详情，点击**查看 DeviceSecret**。

②在设备证书弹框，点击**一键复制**。

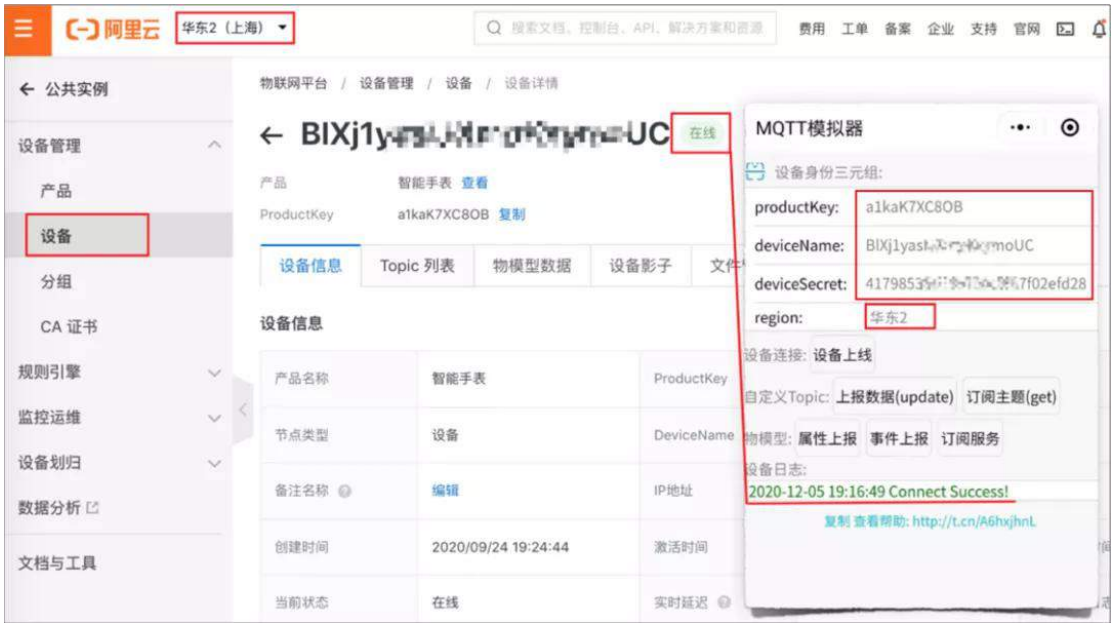
③打开浏览器右上角 **IoT 插件**，粘贴设备信息，生成二维码。


④打开微信，扫描进入小程序。

⑤点击小程序左上角，扫描设备身份二维码，自动填入三元组。

⑥选择设备所在的接入 region。

⑦点击**设备上线**。



# IoT 设备免烧录三元组，开机即时注册

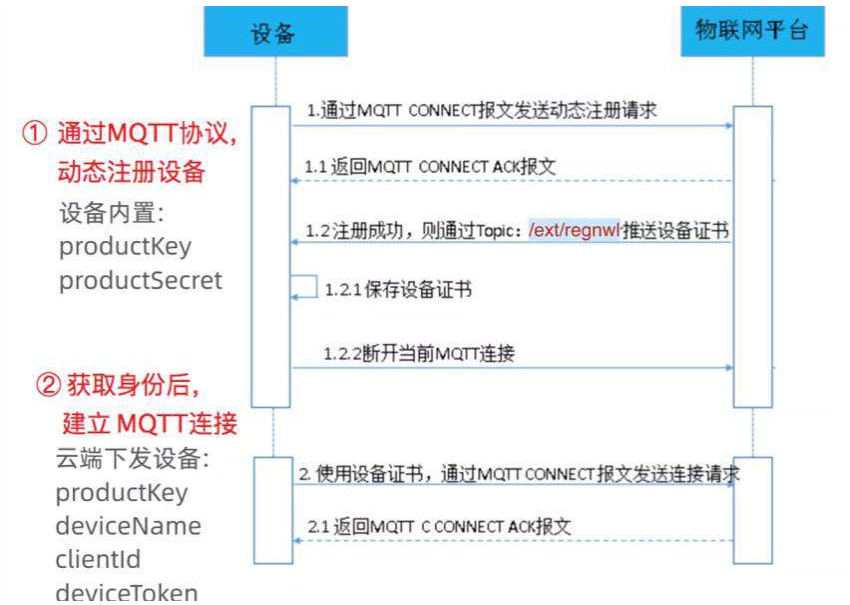
作者 | 孙承旭


## 一、背 景

物联网场景中，设备产线烧录不同三元组成本很高，Android 设备更是无法独立烧录三元组，IoT 存量设备迁移更是无法预置身份三元组，面对这种场景，IoT 物联网平台提供了无需预注册三元组，在设备运行时通过 MQTT 动态注册，获取认证信息，再发起设备业务连接的方案。

## 二、流 程 图

设备动态注册三元组流程如下：

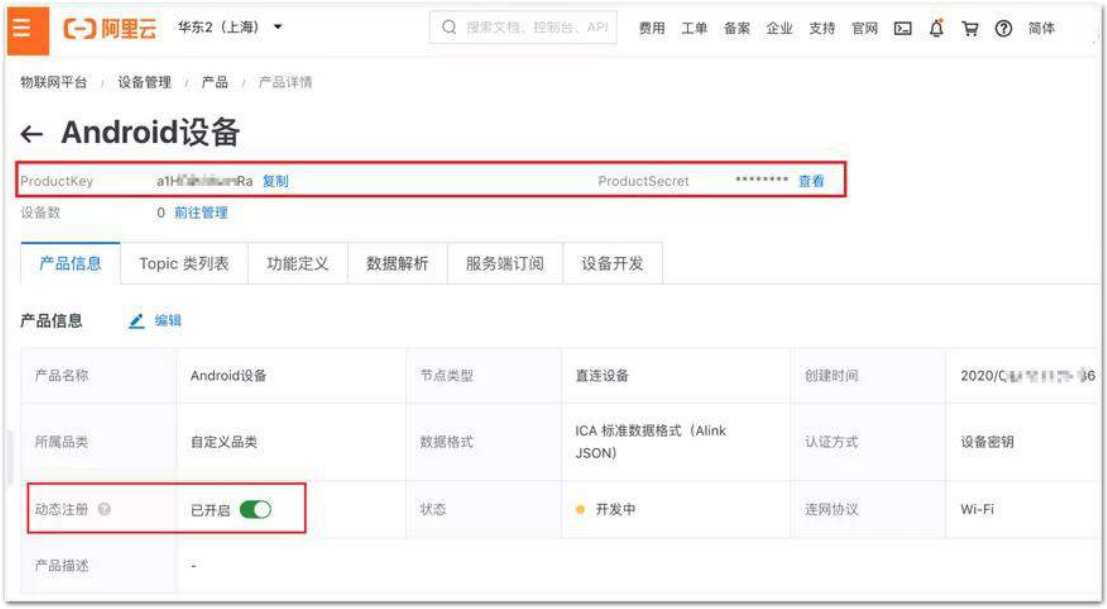


# 三、开发实战

## 1. 创建产品

我们进入 IoT 物联网平台控制台，创建一个新产品：Android 设备。

进入产品详情，获取 productKey 和 productSecret 。开启**动态注册**功能，如下图：




接下来，我们无需按常规流程，**预先注册设备**，而是直接开发设备端程序。

## 2. 设备端开发

我们以 Node.js 代码演示设备动态注册完整过程。

### 动态注册

设备发送 CONNECT 报文，报文中包含动态注册参数，请求建立连接。

- MQTT 连接域名：公共实例的连接域名为`${productKey}.iot-as-mqtt.${regionId}.aliyuncs.com:1883`
- MQTT 动态注册的 CONNECT 报文参数和取值结构如下：

```
mqttClientId: clientId+"|securemode=2,authType=regnwl,random=xxxx,signmethod=xxx|"
mqttUserName: deviceName+"&"+productKey
mqttPassword: sign_hmac(productSecret,content)
```

<table>
<tr><td>mqttClientId</td><td>组成结构：<br>clientId+"|securemode=2,authType=regnwl,random=xxxx,signmethod=xxx|"<br><table><tr><th>参数</th><th>说明</th></tr><tr><td>clientId</td><td>客户端ID,长度在64字符内。</td></tr><tr><td>securemode</td><td>必须设置为2 。</td></tr><tr><td>authType</td><td>认证参数：<br>▪ regnwl：一型一密免预注册认证方式，返回DeviceToken、ClientID。</td></tr><tr><td>random</td><td>随机数。您自定义随机数。</td></tr><tr><td>signMethod</td><td>签名算法。目前支持hmacmd5、hmacsha1、hmacsha256。</td></tr></table></td></tr>
<tr><td>mqttUserName</td><td>组成结构：deviceName+"&"+productKey</td></tr>
<tr><td>mqttPassword</td><td>使用秘钥 productSecret 对拼接字符串<br>("deviceName"+device1+"productKey"+al123456789+"random"+123456)<br>按 mqttClientId 中的 signMethod 签名</td></tr>
</table>

免预注册认证方式，设备注册成功后，物联网平台使用 Topic：**/ext/regnwl**，返回 **ClientID、DeviceToken**。

IoT 物联网平台推送的设备身份消息 Payload 格式如下：

```
{
"productKey" : "xxx",
"deviceName" : "xxx",
"clientId" : "xxx",
"deviceToken" : "xxx"
}
```

动态注册示例代码：

```
function doDeviceRegister() {
// 1.产品信息
const productInfo = {
```

```
    productKey: "产品 productKey",
    productSecret: "产品 productSecret",
    regionId: "cn-shanghai"
    }
    // 2.程序读取的设备唯一标识，比如 MAC，Serial Number 等
    productInfo.deviceName = Math.random().toString(36).substr(2)
    // 3.生成设备动态注册的参数
     var options = getRegisterOptions(productInfo, trustedCA);
    // 4.发起动态注册，获取设备连接 clientId 和 deviceToken
    var registerClient = mqtt.connect(options);

     registerClient.on('message', function(topic, message) {
    // 5. 解析注册结果
    if ('/ext/regnwl' == topic) {
    // 6.断开注册连接
    registerClient.end();
    // 7.发起设备 MQTT 连接
    //deviceOnline(JSON.parse(message),"cn-shanghai")
    }
     })
}
```

生成设备动态注册的参数，示例代码和参考文档：

https://help.aliyun.com/document_detail/132111.html

```
    function getRegisterOptions(productInfo, rootCA) {
    var random = Date.now();
    var content = {
    deviceName: productInfo.deviceName,
    productKey: productInfo.productKey,
    random: random
    }
    var options = {}
    options.clientId = Date.now() + "|securemode=2,authType=regnwl,random=" + random
+ ",signmethod=hmacsha1|"
    options.username = productInfo.deviceName + "&" + productInfo.productKey
    options.password = signHmacSha1(content, productInfo.productSecret)
    options.port = 1883;
```

```
    options.host = `${productInfo.productKey}.iot-as-mqtt.${productInfo.regionId}.aliyuncs.com`;
    options.protocol = 'mqtts';
    options.ca = rootCA
    options.keepalive = 120
    return options;
    }
```

动态注册成功后，产品下会设备数量会更新：


**设备上线和上报数据**

设备端收到并保存 ClientID 和 DeviceToken 的组合后，需要断开当前 MQTT 连接，重新发起设备直连 IoT 平台的请求的 CONNECT 参数如下：

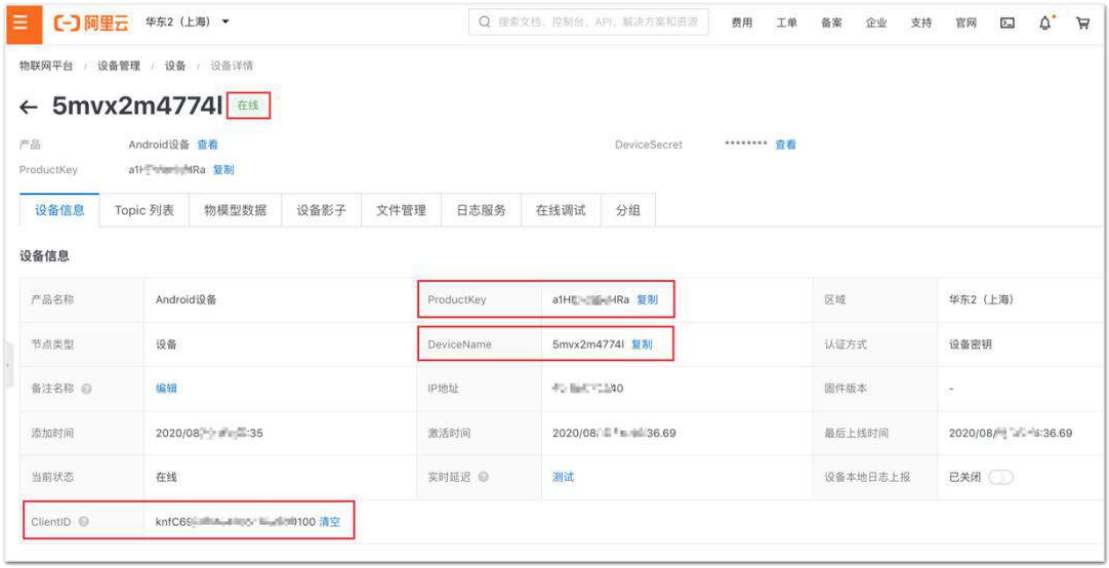
```
mqttClientId: clientId+"|securemode=-2,authType=connwl|"
mqttUsername: deviceName+"&"+productKey
mqttPassword: deviceToken
```

- mqttClientId ： 设备动态注册时获得的 ClientID 拼接固定字符串。
- mqttUserName ： 组成结构为 deviceName+"&"+productKey
- mqttPassword ： 设备动态注册时获得的 DeviceToken

设备建立 MQTT 连接和上报数据，示例代码：

```
function deviceOnline(opts,regionId) {
// 设备 MQTT 连接参数
var options = {}
options.clientId = opts.clientId + "|securemode=-2,authType=connwl|"
options.username = opts.deviceName + "&" + opts.productKey
options.password = opts.deviceToken
options.port = 1883
options.host = `${opts.productKey}.iot-as-mqtt.${regionId}.aliyuncs.com`
options.protocol = 'mqtt'
options.keepalive = 120
// 设备建立 MQTT 连接
var deviceClient = mqtt.connect(options);
// 上报业务数据
deviceClient.publish(`/${opts.productKey}/${opts.deviceName}/user/update`, "sdk client
" + Date.now(), { qos: 1 });
}
```

启动设备端程序后，我们可以在控制台看到设备在线，并展示 ClientID 信息，如下：



# IoT 存量设备零改造迁移上云

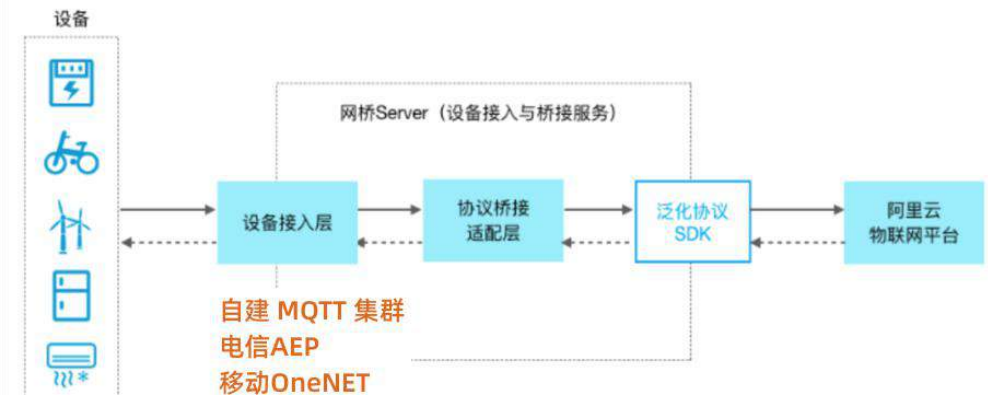
作者 | 苏堤嘉木

在物联网实际项目中，有些设备采用私有协议接入了**本地设备管理系统**，有些 NB-IoT 设备被迫接入了**电信 AEP** 平台，有些设备接入了**移动 OneNET** 平台。但甲方客户的整体业务都部署在阿里云上，我们如何实现整体业务上云呢?

阿里云 IoT 物联网平台提供了**泛化协议 SDK 接入的方案**，在 IoT **设备零改造**的前提下，帮助企业快速构建云上桥接服务，通过网桥实现 IoT 终端设备与阿里云 IoT 物联网平台的**双向数据通信**。

## 一、技术架构

泛化协议 SDK 是协议自适应的框架，用以构建与阿里云物联网平台进行高效双向通信的桥接服务。



**适用场景**

泛化协议 SDK 面向的目标场景包括：

- 设备无 Internet 联网能力。
- 设备采用私有协议。
- 存量设备不修改固件逻辑

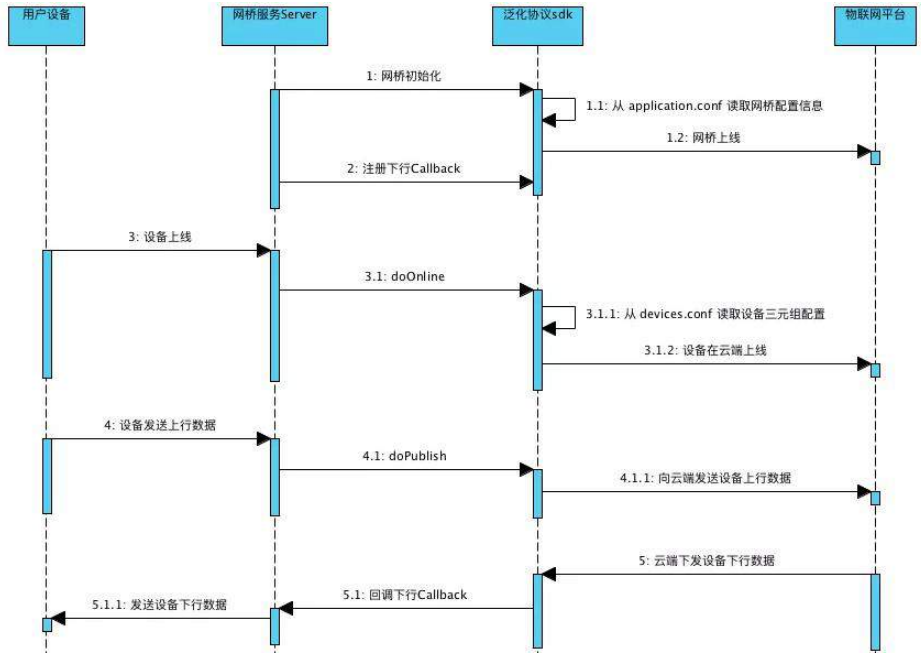
**适用场景**

泛化协议 SDK 使得网桥 Server 具备与物联网平台进行通信的能力。

- 提供基于配置文件的静态配置管理能力。
- 提供设备连接管理能力。
- 提供上行通信能力。
- 提供下行通信能力。

## 二、接入流程

使用泛化协议 SDK，桥接设备与物联网平台的整体流程图，如下：



# 三、开发实战

### 泛化 SDK 依赖

泛化协议仅支持 Java 开发语言，添加泛化协议 SDK 的项目 Maven 依赖，如下：

```
    <dependency>
 <groupId>com.aliyun.openservices</groupId>
 <artifactId>iot-as-bridge-sdk-core</artifactId>
 <version>2.1.3</version>
</dependency>
```

### 初始化 SDK

您需要创建一个 BridgeBootstrap 对象实例，并调用 bootstrap 方法。泛化协议 SDK 初始化工作完成后，读取网桥信息，并向云端发起网桥设备上线请求等。

此外，可以在调用 bootstrap 方法的同时，向泛化协议 SDK 注册一个 DownlinkChannelHandler 回调，用于接收云端下行消息。

```
    BridgeBootstrap bridgeBootstrap = new BridgeBootstrap();
bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
  @Override
  public boolean pushToDevice(Session session, String topic, byte[] payload) {
     // 云端下行控制指令
     String content = new String(bytes);
     log.info("Get DownLink message, session:{}, {}, {}", session, topic, content);
     return true;
  }

  @Override
  public boolean broadcast(String topic, byte[] payload) {
     return false;
  }
});
```

**配置泛化网桥身份**

网桥配置默认使用配置文件方式。默认从 Java 工程默认资源文件路径（一般是 src/main/resources/）下的 application.conf 中读取配置文件。

```
    # Server endpoint
http2Endpoint = "https://你的 ProductKey.iot-as-http2.cn-shanghai.aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge"

# Gateway device info, productKey & deviceName & deviceSecret
productKey = ${bridge-ProductKey}
deviceName = ${bridge-DeviceName}
deviceSecret = ${bridge-DeviceSecret}
```

**配置网桥下设备身份**

配置设备原始身份标识符和设备证书信息的映射关系。默认使用配置文件方式，默认从 Java 工程的默认资源文件路径（一般是 src/main/resources/）下的 devices.conf 中读取配置文件。

```
    ${device-original-Identity} {
 productKey : ${device-ProductKey}
 deviceName : ${device-DeviceName}
 deviceSecret : ${device-DeviceSceret}
}
```

**设备上线**

设备上线时，需要传 Session。下行消息回调时，会把 Session 回调给网桥。Session 中包含设备的原始身份标识符字段，以便网桥判断消息属于哪个设备。

```
    UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
//创建 Session
Object channel = new Object();
Session session = Session.newInstance(originalIdentity, channel);
//设备上线
```

```
    boolean success = uplinkHandler.doOnline(session, originalIdentity);
if (success) {
   // 设备上线成功，网桥接受后续设备通信请求。
} else {
   // 设备上线失败，网桥可以拒绝后续设备通信请求，如断开连接。
}
```


**设备通过网桥上报数据**

网桥使用泛化协议 SDK 代理设备上报消息，代码如下：

```
    // originalIdentity 设备上报数据
DeviceIdentity deviceIdentity = ConfigFactory.getDeviceConfigManager().getDeviceIdentity(or
iginalIdentity);
ProtocolMessage protocolMessage = new ProtocolMessage();
protocolMessage.setPayload(payload);
protocolMessage.setQos(0);
protocolMessage.setTopic(String.format(TOPIC_TEMPLATE_USER_DEFINE, deviceIdentity.
getProductKey(), deviceIdentity.getDeviceName()));
// 网桥代理上报
uplinkChannelHandler.doPublishAsync(originalIdentity, protocolMessage);
```


pump8393941

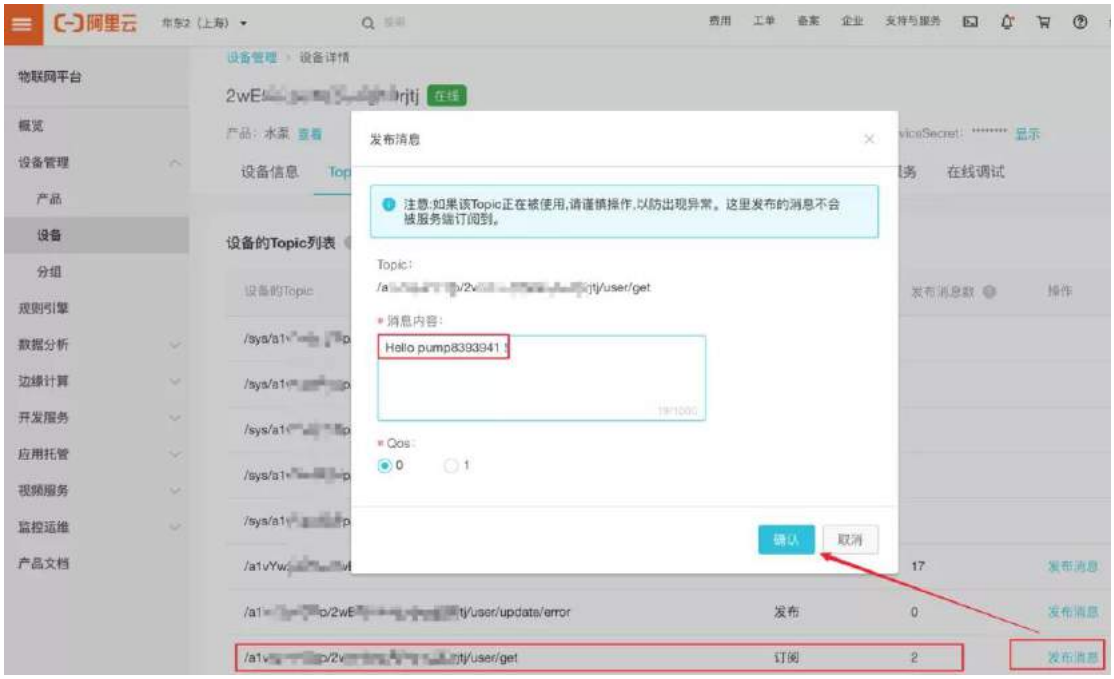

pump6785763

**设备接收云端指令**

云端可以调用 Pub API 给设备下发控制指令，网桥监听和处理云端消息的代码如下：

```
    private static ExecutorService executorService  = new ThreadPoolExecutor(
  Runtime.getRuntime().availableProcessors(),
  Runtime.getRuntime().availableProcessors() * 2,
  60, TimeUnit.SECONDS,
  new LinkedBlockingQueue<>(1000),
  new ThreadFactoryBuilder().setDaemon(true).setNameFormat("bridge-downlink-handle
-%d").build(),
  new ThreadPoolExecutor.AbortPolicy());

public static void main(String args[]) {
  //Use application.conf & devices.conf by default
  bridgeBootstrap = new BridgeBootstrap();
  bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
    @Override
    public boolean pushToDevice(Session session, String topic, byte[] payload) {
      //get message from cloud
      //get downlink message from cloud
      executorService.submit(() -> handleDownLinkMessage(session, topic, payload));
      return true;
    }
    @Override
    public boolean broadcast(String s, byte[] bytes) {
      return false;
    }
```

```
        });
}

private static void handleDownLinkMessage(Session session, String topic, byte[] payload) {
    String content = new String(payload);
    log.info("Get DownLink message, session:{}, topic:{}, content:{}", session, topic, content);
    Object channel = session.getChannel();
    String originalIdentity = session.getOriginalIdentity();
    //for example, you can send the message to device via channel, it depends on you specifi
c server implementation
}
```





**设备下线**

当设备从网桥断开后，可以调用下线接口，告知云端：

```
    upLinkHandler.doOffline(originalIdentity);
```

# 基于函数计算实现 IoT 设备动态注册

作者 | 苏堤嘉木


## 一、技术方案

1. 设备激活，携带 deviceId(设备唯一标识)发起 HTTPS 请求到函数计算 FC。
2. 函数计算 FC 调用 IoT 平台的 RegisterDevice 接口，传递 productKey 和 deviceId。
   - 成功，返回设备三元组。
   - 失败，调用 IoT 平台 QueryDeviceDetail 接口，传递 productKey 和 deviceId，获取设备三元组。
3. 设备获取到三元组后，通过 MQTT 和阿里云 IoT 平台建立连接。

**依赖 API**

**RegisterDevice** https://help.aliyun.com/document_detail/69470.html
**QueryDeviceDetail** https://help.aliyun.com/document_detail/69594.html

## 二、函数计算实现

创建函数计算函数，并配置 HTTPS 触发器：


Nodejs 版本实现：

```
    const getRawBody = require('raw-body');
const co = require('co');
const RPCClient = require('@alicloud/pop-core').RPCClient;

const options = {
        accessKey: "云账号的 AK",
        accessKeySecret: "云账号的 AK Secret"
};

//1.创建 iot pop server client
const iotClient = new RPCClient({
        accessKeyId: options.accessKey,
        secretAccessKey: options.accessKeySecret,
        endpoint: options.endpoint || 'https://iot.cn-shanghai.aliyuncs.com',
        apiVersion: options.apiVersion || '2018-01-20'}
);

const productKey = '产品的 productKey';

module.exports.handler = function(req, resp, context) {

        getRawBody(req, function(err, body) {

            body = JSON.parse(decodeURIComponent(body.toString()));
```

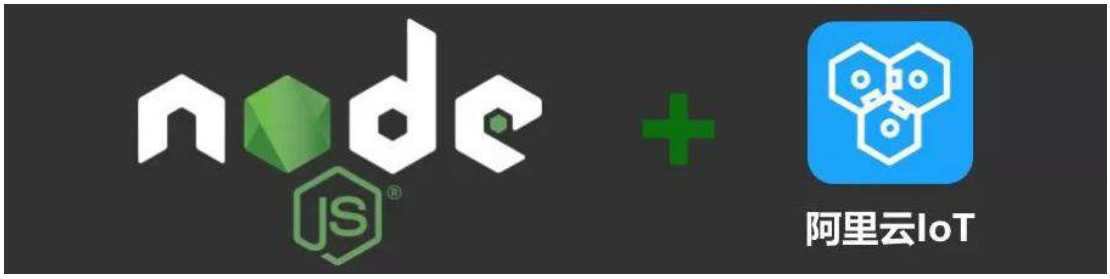
```
        const deviceId = body.deviceId;

        co(function*() {

            try {
                //创建设备
                const registerResponse = yield iotClient.request(
                'RegisterDevice', {
                    productKey: productKey,
                    deviceName: deviceId
            });

                resp.send(JSON.stringify({
                success: true,
                data: registerResponse.Data
            }));
             } catch (err) {
                //设备已存在，查询设备详情
                const queryResponse = yield iotClient.request(
              'QueryDeviceDetail', {
                        productKey: productKey,
                        deviceName: deviceId
            });
                const returnJson = { success: true }
                returnJson.data = {
                    DeviceName: queryResponse.Data.DeviceName,
                    ProductKey: queryResponse.Data.ProductKey,
                    DeviceSecret: queryResponse.Data.DeviceSecret,
                    IotId: queryResponse.Data.IotId
            }
                resp.send(JSON.stringify(returnJson));
             }

        });

    });

};
```

# 三、设备调用 FC

设备通过 HTTPS POST JSON 触发函数计算，获取三元组。

# Nodejs 版 mqtt 接入阿里云 IoT

作者 | 苏堤嘉木




## 一、准备工作

### 1. 注册阿里云账号

使用个人淘宝账号或手机号，开通阿里云账号，并通过支付宝实名认证。

### 2. 免费开通 IoT 物联网套件

产品官网：https://www.aliyun.com/product/iot

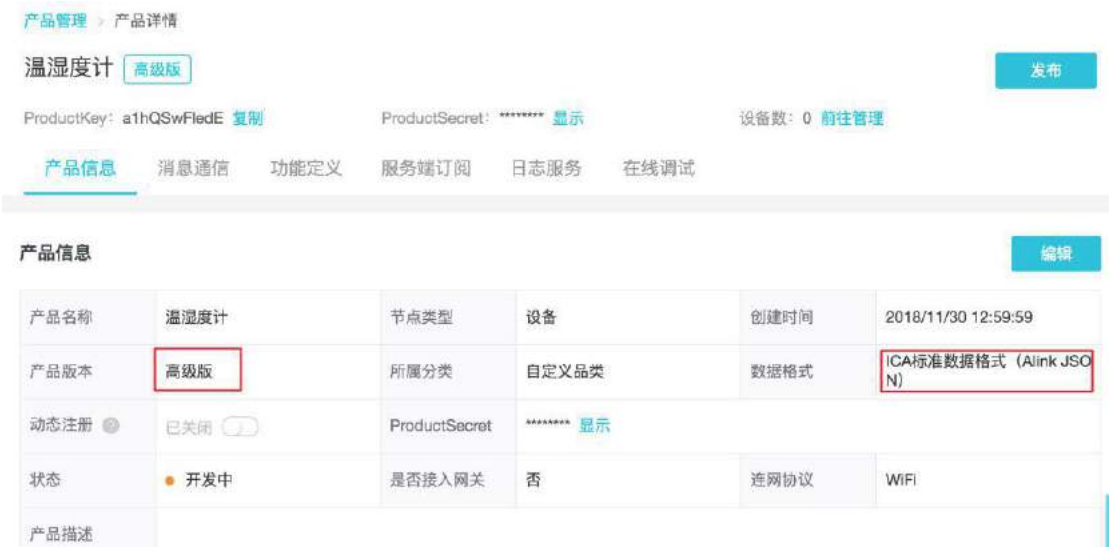
### 3. 软件环境

Nodejs 安装 https://nodejs.org

## 二、开发步骤

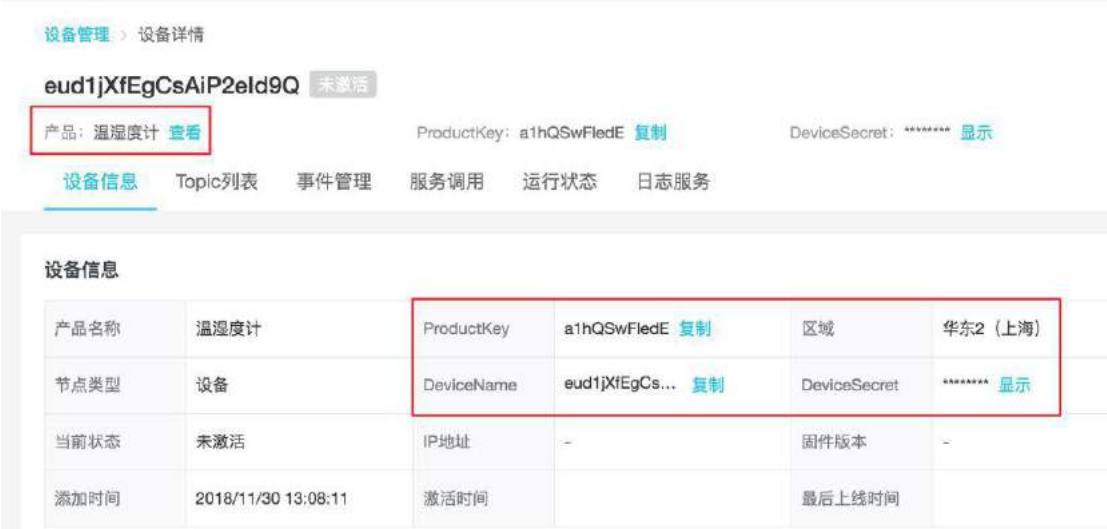
### 1. IoT 云端开发

1)创建高级版产品




2)功能定义，添加产品属性

3）注册设备，获得身份三元组




## 2. 设备端开发

我们用 nodejs 程序来模拟设备，建立连接，上报数据。

1）创建设备端项目

创建文件夹 iot-demo
创建 2 个文件 package.json 和 device.js

2）package.json 文件

添加阿里云 IoT 套件 sdk 依赖：

```
{
  "name": "aliyun-iot-demo",
  "dependencies": {
    "mqtt": "2.18.8"
  }
}
```

npm 下载安装 IoT 套件 SDK：

```
$ npm install
```

3) device.js 应用程序代码

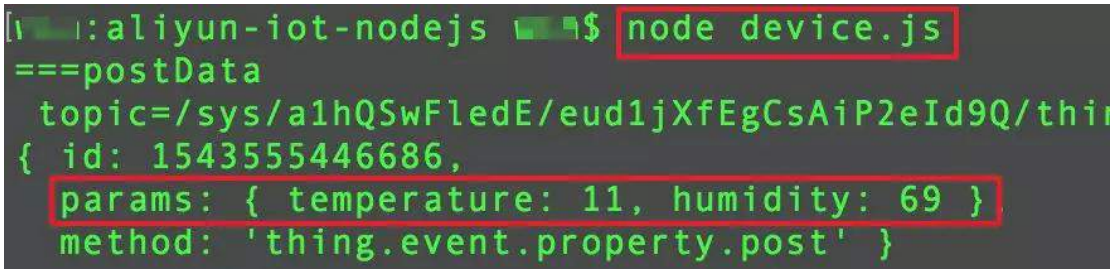
```
/**
"dependencies": { "mqtt": "2.18.8" }
*/
const crypto = require('crypto');
const mqtt = require('mqtt');

//设备身份三元组+区域
const deviceConfig = {
        productKey: "替换产品",
        deviceName: "替换设备",
        deviceSecret: "替换密码",
        regionId: "cn-shanghai"
};

const params = {
        productKey: deviceConfig.productKey,
        deviceName: deviceConfig.deviceName,
        timestamp: Date.now(),
        clientId: Math.random().toString(36).substr(2)
}

//CONNECT 参数
const options = {
        keepalive:60, //60s
        clean:false, //cleanSession 保持持久会话
        protocolVersion: 4 //MQTT v3.1.1
}

//1.生成 clientId，username，password
options.password = signHmacSha1(params, deviceConfig.deviceSecret);
options.clientId = `${params.clientId}|securemode=3,signmethod=hmacsha1,timestamp=${p
arams.timestamp}|`;
options.username = `${params.deviceName}&${params.productKey}`;
```

```
    const url = `tcp://${deviceConfig.productKey}.iot-as-mqtt.${deviceConfig.regionId}.aliyu
ncs.com:1883`;

//2.建立连接
const client = mqtt.connect(url,options);

//3.属性数据上报
const topic = `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/pro
perty/post`;

setInterval(function() {
        //发布数据到 topic
        client.publish(topic, getPostData());
}, 5 * 1000);

//模拟数据
function getPostData(){
        const payloadJson = {
            id: Date.now(),
            params: {
                temperature: Math.floor((Math.random() * 20) + 10),
                humidity: Math.floor((Math.random() * 20) + 60)
            },
            method: "thing.event.property.post"
        }

        console.log("===postData\n topic=" + topic)
        console.log(payloadJson)

        return JSON.stringify(payloadJson);
}

/*
      生成基于 HmacSha1 的 password
      参考文档：https://help.aliyun.com/document_detail/73742.html?#h2-url-1
    */function signHmacSha1(params, deviceSecret) {

        let keys = Object.keys(params).sort();
        // 按字典序排序
```

```
        keys = keys.sort();
        const list = [];
        keys.map((key) => {
            list.push(`${key}${params[key]}`);
        });
        const contentStr = list.join('');
        return crypto.createHmac('sha1', deviceSecret).update(contentStr).digest('hex');
}
```

## 三、设备运行

### 1. 设备启动




### 2. IoT 控制台查看设备运行状态


至此，完成了物联网设备接入阿里云 IoT 物联网云平台的开发实践。

# C#设备接入 IoT 物联网平台

作者 | 苏堤嘉木


## 一、准备工作

### 1. 注册阿里云账号

使用淘宝账号或手机号，开通阿里云账号，并通过实名认证(可以用支付宝认证)。

### 2. 免费开通 IoT 物联网套件

产品官网： https://www.aliyun.com/product/iot

### 3. 软件开发环境

- 语言 C#
- 工具 Visual Studio IDE

## 二、IoT 平台云端开发

### 1. 创建基础版产品

产品信息：


消息通信 Topic：


### 2. 注册设备

获取设备身份三元组，ProductKey，DeviceName，DeviceSecret：


## 三、设备端开发

### 1. IoT 平台接入 password 签名算法文件

签名规则参考： https://www.yuque.com/cloud-dev/iot-tech/mebm5g

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography;
namespace iotxsdkmqttnet {
    public class IotSignUtils {
        public static string sign(Dictionary<string, string> param,
                                  string deviceSecret, string signMethod) {
            string[] sortedKey = param.Keys.ToArray();
            Array.Sort(sortedKey);

            StringBuilder builder = new StringBuilder();
            foreach(var i in sortedKey){
                builder.Append(i).Append(param[i]);
            }

            byte[] key = Encoding.UTF8.GetBytes(deviceSecret);
            byte[] signContent = Encoding.UTF8.GetBytes(builder.ToString());
            //这里根据signMethod动态调整，本例子硬编码了： 'hmacmd5'
            var hmac = new HMACMD5(key);
            byte[] hashBytes = hmac.ComputeHash(signContent);

            StringBuilder signBuilder = new StringBuilder();
            foreach (byte b in hashBytes)
                signBuilder.AppendFormat("{0:x2}", b);

            return signBuilder.ToString();
        }
    }
}
```

## 2. 接入 IoT 平台 C#版本的 MQTT 库

C#的 mqtt 库：

https://www.eclipse.org/paho/clients/dotnet/

Visual Studio 工程：

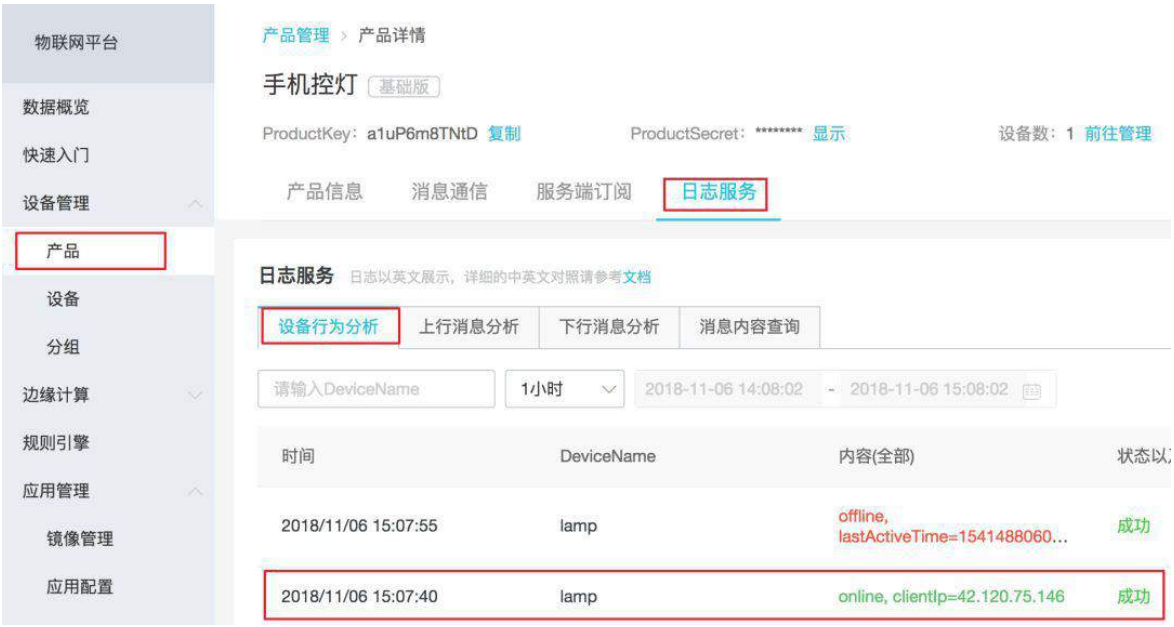
https://www.eclipse.org/downloads/download.php?file=/paho/1.2-milestones/m2mqtt/M2Mqtt_4.2.0.0.zip

## 3. 设备端应用程序

```
ce iotMqttDemo {
ss MainClass {
 static string ProductKey = "******";
 static string DeviceName = "******";
 static string DeviceSecret = "******";
 static string RegionId = "cn-shanghai";

 static string PubTopic = "/" + ProductKey + "/" + DeviceName + "/update";
 static string SubTopic = "/" + ProductKey + "/" + DeviceName + "/get";

 public static void Main(string[] args)
 {
     IPHostEntry host = Dns.GetHostEntry(Dns.GetHostName());
     string clientId = host.AddressList.FirstOrDefault(
         ip => ip.AddressFamily == System.Net.Sockets.AddressFamily.InterNetwork).ToString();
     string t = Convert.ToString(DateTimeOffset.Now.ToUnixTimeMilliseconds());
     string signmethod = "hmacmd5";

     Dictionary<string, string> dict = new Dictionary<string, string>();
     dict.Add("productKey", ProductKey);
     dict.Add("deviceName", DeviceName);
     dict.Add("clientId", clientId);
     dict.Add("timestamp", t);

     string mqttUserName = DeviceName + "&" + ProductKey;
     string mqttPassword = IotSignUtils.sign(dict, DeviceSecret, signmethod);
     string mqttClientId = clientId + "|securemode=3,signmethod="+signmethod+",timestamp=" + t + "|";

     string targetServer = "tcp://" + ProductKey + ".iot-as-mqtt." + RegionId + ".aliyuncs.com";

     ConnectMqtt(targetServer, mqttClientId, mqttUserName, mqttPassword);
 }

 static void ConnectMqtt(string targetServer, string mqttClientId, string mqttUserName, string mqttPass
     MqttClient client = new MqttClient(targetServer);
     client.ProtocolVersion = MqttProtocolVersion.Version_3_1_1;

     client.Connect(mqttClientId, mqttUserName, mqttPassword, false, 60);
     client.MqttMsgPublishReceived += Client_MqttMsgPublishReceived;

     //发布消息
     String content = "{'content':'msg from :" + mqttClientId + ", 你好, 这里是.NET设备}";
     var id = client.Publish(PubTopic, Encoding.ASCII.GetBytes(content));
     //订阅消息
     client.Subscribe(new string[] { SubTopic }, new byte[] { 0 });
 }

 static void Client_MqttMsgPublishReceived(object sender, MqttMsgPublishEventArgs e)
 {
     // handle message received
     string topic = e.Topic;
     string message = Encoding.ASCII.GetString(e.Message);
 }
```

## 四、运行结果

云端看到设备上线记录，数据上报记录。




在设备和云端的这条 mqtt 双向数据通道上，就可以实现业务数据通信了。

至此，完成了基于 C#语言开发的传感器设备接入阿里云 IoT 物联网云平台的开发实践。

# IoT 设备用 HTTPS 协议接入物联网平台

作者 | 苏堤嘉木


在物联网开发过程中，有时候我们现有的 IoT 设备是基于 **HTTPS 协议栈**，那么我们能否接入 IoT 物联网平台呢？答案是肯定的。本文就通过开发实战给大家讲解如何使用 HTTPS 接入 IoT 物联网平台**华东 2(上海)区域**。

HTTPS 接入的官网文档 ：https://help.aliyun.com/document_detail/58034.html

**HTTPS 接入主要有两个步骤：**

- 设备身份认证，获取数据传输的 token
- 通过 HTTPS 上报传感器设备采集的数据

# 一、设备身份认证

设备身份认证服务器接入点：iot-as-http.cn-shanghai.aliyuncs.com

*表 1. 参数说明*

| 参数 | 说明 |
| --- | --- |
| Method | 请求方法。支持POST方法。 |
| URL | `/auth`，URL地址，只支持HTTPS。 |
| Host | endpoint地址：`iot-as-http.cn-shanghai.aliyuncs.com` |
| Content-Type | 设备发送给物联网平台的上行数据的编码格式。目前只支持 application/json |
| body | 设备认证信息。JSON数据格式。具体信息，请参见下表body参数。 |

*表 2. body参数*

| 字段名称 | 是否必需 | 说明 |
| --- | --- | --- |
| productKey | 是 | 设备所属产品的Key。从物联网平台的控制台获取。 |
| deviceName | 是 | 设备名称。从物联网平台的控制台获取。 |
| clientId | 是 | 客户端ID。长度为64字符内，建议以MAC地址或SN码作为clientId。 |
| timestamp | 否 | 时间戳。校验时间戳15分钟内的请求有效。 |
| sign | 是 | 签名。<br>签名计算格式为 `hmacmd5(DeviceSecret,content)` 。<br>其中，**content**为将所有提交给服务器的参数（除**version**、**sign**和**signmethod**外），按照英文字母升序，依次拼接排序（无拼接符号）的结果。<br>签名示例：<br>假设 `clientId = 12345` ， `deviceName = device` ， `productKey = pk` ， `timestamp = 789` ， `signmethod=hmacsha1` ， `deviceSecret=secret` ，那么签名计算为 `hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();` 。最后二进制数据转十六进制字符串，大小写不敏感。 |
| signmethod | 否 | 算法类型，支持hmacmd5和hmacsha1。<br>若不传入此参数，则默认为hmacmd5。 |
| version | 否 | 版本号。若不传入此参数，则默认default。 |

认证请求示例：

```
    POST /auth HTTP/1.1
Host: iot-as-http.cn-shanghai.aliyuncs.com
Content-Type: application/json
body: {
  "version": "default",
  "clientId": "mylight1000002",
  "signmethod": "hmacsha1",
  "sign": "4870141D4067227128CBB4377906C3731CAC221C",
  "productKey": "ZG1EvTEa7NN",
  "deviceName": "NlwaSPXsCpTQuh8FxBGH",
  "timestamp": "1501668289957"
}返回示例:
```

返回结果示例:

```
    {
 "code": 0,//业务状态码
 "message": "success",//业务信息
 "info": {
  "token":  "6944e5bfb92e4d4ea3918d1eda3942f6"
 }
}
```

## 二、设备数据上报

数据上报 IoT 平台接入点： iot-as-http.cn-shanghai.aliyuncs.com/topic/${topic}

| 参数 | 说明 |
| --- | --- |
| Method | 请求方法。支持POST方法。 |
| URL | `/topic/${topic}` 。其中，变量 `${topic}` 需替换为当前设备对应的Topic。只支持HTTPS。 |
| Host | endpoint地址：`iot-as-http.cn-shanghai.aliyuncs.com` 。 |
| password | 放在Header中的参数，取值为调用设备认证接口auth返回的token值。 |
| body | 发往${topic}的数据内容，格式为二进制byte[]，UTF-8编码。 |

请求示例：

```
POST /topic/a1GFjLP3xxC/device123/pub
Host: iot-as-http.cn-shanghai.aliyuncs.com
password:${token}
Content-Type: application/octet-stream
body: ${your_data}
```

返回示例：

```
{
 "code": 0,//业务状态码
 "message": "success",//业务信息
 "info": {
  "messageId": 892687627916247040
 }
}
```

## 三、使用 HTTPS 上报数据实战

### 1. 创建产品

## 2. 功能定义

添加产品属性定义：

| 属性名 | 标识符 | 数据类型 | 范围 |
| --- | --- | --- | --- |
| 温度 | temperature | float | -50~100 |
| 湿度 | humidity | float | 0~100 |

产品功能定义：


## 3. 注册设备，获得身份三元组

设备详情:


### 4. 设备模拟代码

```
    var rp = require('request-promise');
const crypto = require('crypto');

const deviceConfig = {
  productKey: "替换 productKey",
  deviceName: "替换 deviceName",
  deviceSecret: "替换 deviceSecret"
}

const topic = `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/prop
erty/post`;

//1.获取身份 token
rp(getAuthOptions(deviceConfig))
  .then(function(parsedBody) {
    console.log('Auth Info :'+JSON.stringify(parsedBody))
    //2.上报物模型数据
    pubData(topic, parsedBody.info.token, getPostData())
  })
  .catch(function(err) {
    console.log('Auth err :'+JSON.stringify(err))
  });
```

```
    //生成 Auth 认证的参数
function getAuthOptions(deviceConfig) {

  const params = {
    productKey: deviceConfig.productKey,
    deviceName: deviceConfig.deviceName,
    timestamp: Date.now(),
    clientId: Math.random().toString(36).substr(2),
  }

  //生成 clientId，username，password
  var password = signHmacSha1(params, deviceConfig.deviceSecret);

  var options = {
    method: 'POST',
    uri: 'https://iot-as-http.cn-shanghai.aliyuncs.com/auth',
    body: {
      "version": "default",
      "clientId": params.clientId,
      "signmethod": "hmacsha1",
      "sign": password,
      "productKey": deviceConfig.productKey,
      "deviceName": deviceConfig.deviceName,
      "timestamp": params.timestamp
    },
    json: true
  };

  return options;
}

//数据上报
function pubData(topic, token, data) {

  const options = {
    method: 'POST',
    uri: 'https://iot-as-http.cn-shanghai.aliyuncs.com/topic' + topic,
    body: data,
    headers: {
```

```
            password: token,
        'Content-Type': 'application/octet-stream'
      }
  }

  rp(options)
      .then(function(parsedBody) {
          console.log('publish success :' + parsedBody)
      })
      .catch(function(err) {
          console.log('publish err ' + JSON.stringify(err))
      });

}
//模拟物模型数据
function getPostData() {
  var payloadJson = {
      id: Date.now(),
      params: {
          humidity: Math.floor((Math.random() * 20) + 60),
          temperature: Math.floor((Math.random() * 20) + 10)
      },
      method: "thing.event.property.post"
  }

  console.log("===postData\n topic=" + topic)
  console.log(payloadJson)

  return JSON.stringify(payloadJson);
}
//HmacSha1 sign
function signHmacSha1(params, deviceSecret) {

  let keys = Object.keys(params).sort();
  // 按字典序排序
  keys = keys.sort();
  const list = [];
  keys.map((key) => {
      list.push(`${key}${params[key]}`);
```

```
  });
  const contentStr = list.join('');
  return crypto.createHmac('sha1', deviceSecret).update(contentStr).digest('hex');
}
```

### 5. 运行效果

# 网关与子设备上云开发实战

作者 | 苏堤嘉木

在 IoT 物联网场景中，对我们的终端设备本身**无连接互联网能力**时，那么数据如何上云呢?

IoT 物联网平台支持设备 MQTT 直连，也支持的设备挂载到网关上，作为网关的子设备，由网关代理接入 IoT 物联网平台。这样只需要网关建立一条 MQTT 长连接通道，所有子设备可以**复用网关的 MQTT 通道**，高效传输数据到云端。

这时候网关设备除了自身作为 IoT 网关设备(拥有身份三元组)与 IoT 物联网平台建立 **MQTT 连接**，**收发数据**，还要负责**子设备管理**，包括:

- 网关添加子设备网络拓扑关系
- 子设备复用网关 mqtt 连接通道上线
- 网关把子设备数据上报到云端
- 网关接收指令，并转发给子设备
- 网关上报子设备下线
- 网关删除子设备网络拓扑关系

网关和子设备通信的协议由本地网络决定，可以是 http、mqtt、ZigBee、Modbus、BLE、OPC-UA 等，这部分逻辑由网关实现。整体架构如下：


我们以 Java 版本 LinkKit SDK 为例进行开发实战讲解。

## 一、创建网关产品

创建网关产品时，需要选择节点类型：网关，即指可以挂载子设备的直连设备。网关需要管理子设备、维持与子设备的拓扑关系，并将该拓扑关系同步到云端。

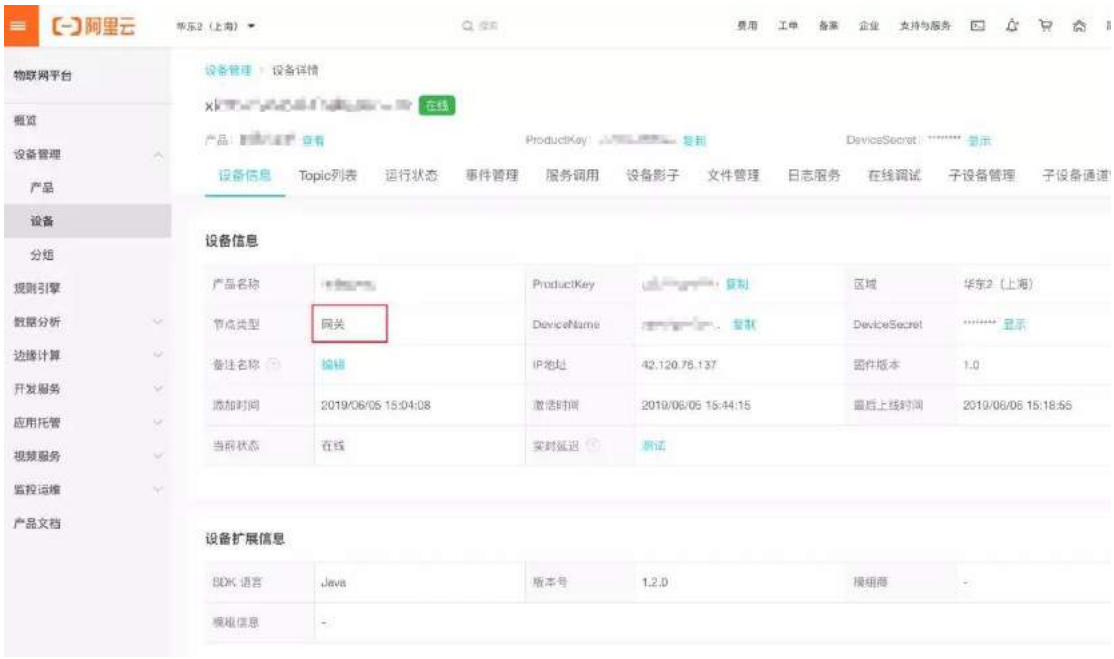

基于此网关产品注册一台网关设备，并获得身份认证三元组。


## 二、网关设备上线

使用网关设备三元组身份，建立端到云上的 MQTT 长连接，逻辑如下：

```
LinkKitInitParams params = new LinkKitInitParams();

DeviceInfo gatewayInfo = new DeviceInfo();
gatewayInfo.productKey = gateway.productKey;
gatewayInfo.deviceName = gateway.deviceName;
gatewayInfo.deviceSecret = gateway.deviceSecret;

params.deviceInfo = gatewayInfo;
LinkKit.getInstance().init(params, ILinkKitConnectListener)
```

此时，我们在控制台查看网关设备状态为：**在线**



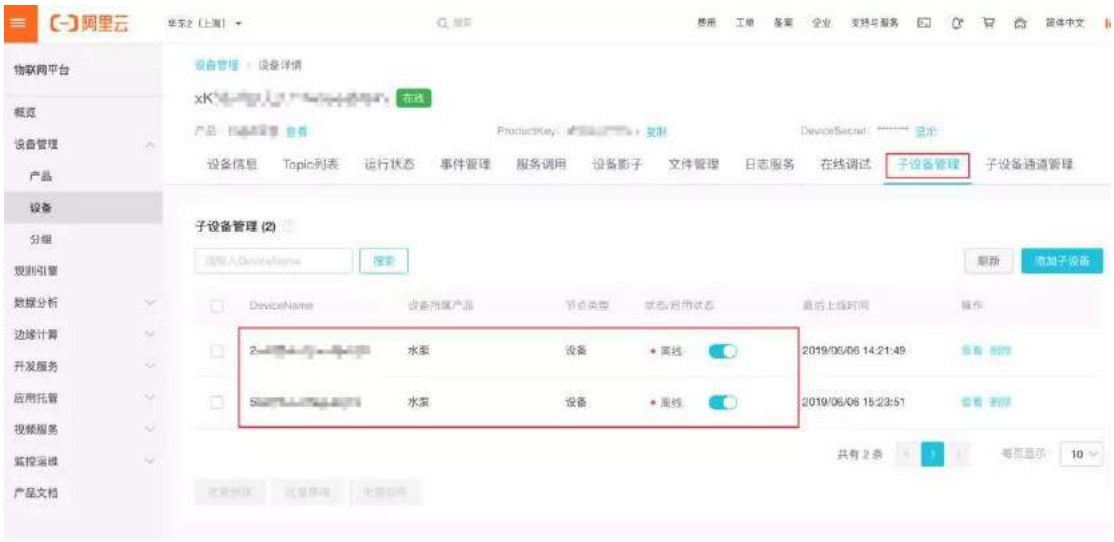

## 三、添加网络拓扑关系

接下来，我们在 IoT 物联网控制台，创建水泵产品，并注册设备，获取身份三元组。


当**子设备水泵**通过本地协议接入**网关**后，我们需要同步网络拓扑关系到云端，也就是添加子设备到这个网络里。代码逻辑如下：

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewayAddSubDevice(
  deviceInfo, //子设备身份
  SubDeviceConnectListener)
```

添加成功后，我们在控制台网关设备详情，可以看到关联的子设备列表，如下：



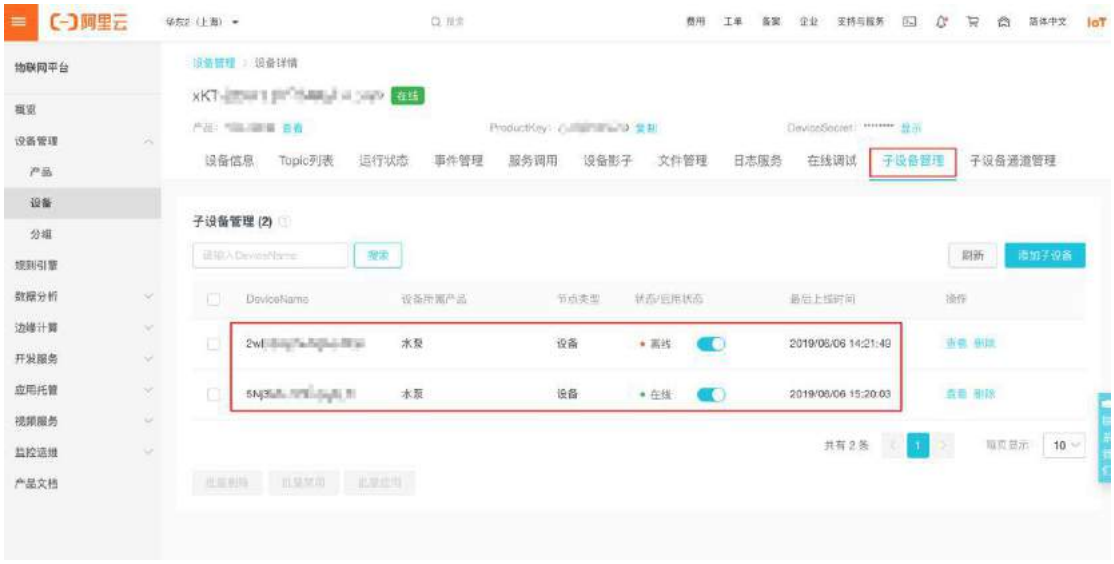

## 四、子设备上线

添加网络拓扑关系后，我们需要把子设备在本地网络的状态同步到云端。

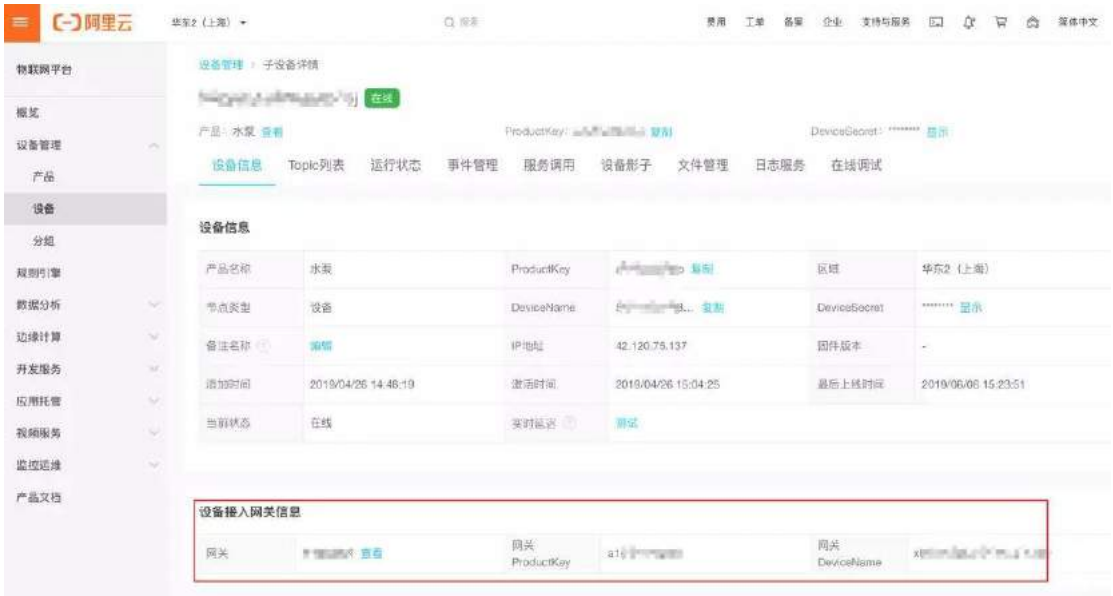
子设备上线代码逻辑如下：

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewaySubDeviceLogin(
  deviceInfo, //子设备身份
  ISubDeviceActionListener)
```

子设备成功上线后，我们在网关设备详情，可以查看到子设备状态为: **在线**




在设备详情页面，可以查看到接入的**网关信息**，如下图:



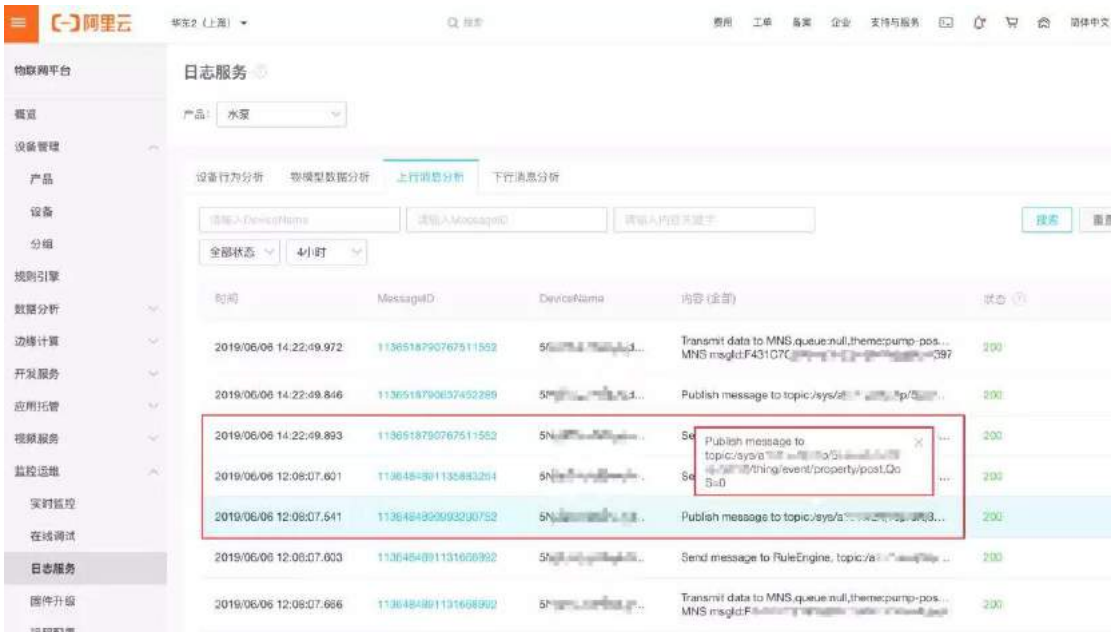

## 五、子设备上报数据

子设备在线的情况下，可以发布数据到 IoT 物联网平台。

子设备上报数据代码逻辑如下:

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewaySubDevicePublish(
  topic, //子设备 topic
  data, //数据
  deviceInfo, //子设备身份
  ISubDeviceActionListener)
```

日志服务查看子设备上报数据的日志如下：




## 六、子设备订阅主题

子设备同样也可以接收云端指令，首先要订阅对应的 Topic，代码逻辑如下：

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewaySubDeviceSubscribe(
  topic, //子设备订阅 Topic
```

```
        deviceInfo, //子设备身份

    ISubDeviceActionListener)
```

## 七、子设备下线

子设备和本地网关断开连接时，我们需要把离线状态同步到云端，代码逻辑如下：

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewaySubDeviceLogout(
    deviceInfo, //子设备身份
    ISubDeviceActionListener)
```

## 八、子设备网络拓扑删除

子设备完全从本地网络移除是，我们需要删除设备的网络拓扑关系，代码逻辑如下：

```
    DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
LinkKit.getInstance().getGateway().gatewayDeleteSubDevice(
    deviceInfo, //子设备身份
    ISubDeviceRemoveListener)
```

至此，我们掌握了网关和子设备的开发过程。如果网关设备采用 C，或者 Python 开发，背后交互逻辑相同，具体可以参考阿里云 IoT 云产品开发文档。

# 设备用 X.509 证书接入实战(一)

作者 | 苏堤嘉木

IoT 物联网平台支持使用私有数字证书进行设备接入身份认证。使用私有数字证书，需要完成如下操作：

- 在物联网平台注册 CA 证书；
- 将数字设备证书与设备身份相绑定。

**限制说明**

- 仅 **MQTT 协议**直连的设备可使用私有 CA 证书。
- 目前仅**华东 2（上海）**地域支持使用私有 CA 证书。
- 使用私有 CA 证书时，只支持 **RSA 算法**签名的设备证书。
- 一个阿里云账号最多可注册 **10 个私有 CA 证书**。

本次开发实战，我们介绍如何在物联网平台完成私有 CA 证书注册。

## 一、制作私有 CA 证书

我们在 Mac 电脑上使用 OpenSSL 工具制作私有 CA 证书。

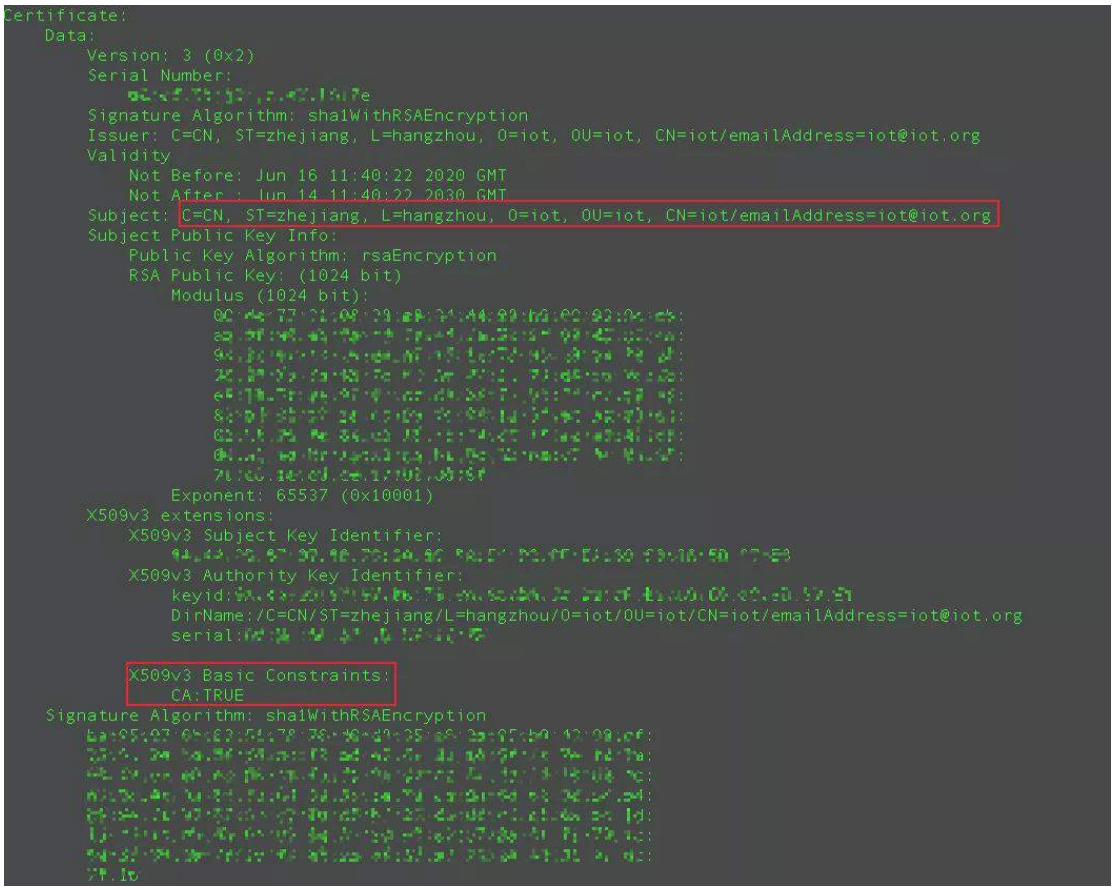
查看 openssl 版本：

```
    openssl version -a
OpenSSL 0.9.8zh 14 Jan 2016
built on: Nov 19 2017
platform: darwin64-x86_64-llvm
options:  bn(64,64) md2(int) rc4(ptr,char) des(idx,cisc,16,int) blowfish(idx)
compiler: -arch x86_64 -fmessage-length=0 -pipe -Wno-trigraphs -fpascal-strings -fasm
-blocks -O3 -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIAN -DMD3
2_REG_T=int -DOPENSSL_NO_IDEA -DOPENSSL_PIC -DOPENSSL_THREADS -DZLI
B -mmacosx-version-min=10.6
OPENSSLDIR: "/System/Library/OpenSSL"
```

生成私有 CA 和公钥 key 命令如下：

```
# 生成私有 CA 和 key，有效期 10 年
openssl req -new -x509 -days 3650 -newkey rsa:2048 -keyout myIoTCARoot.key -out myI
oTCARoot.crt -subj \
"/C=CN/ST=Shanghai/L=Shanghai/O=IoT/OU=iot/CN=x.iot.cn"
# 查看 CA 证书
openssl x509 -noout -text -in myIoTCARoot.crt
```

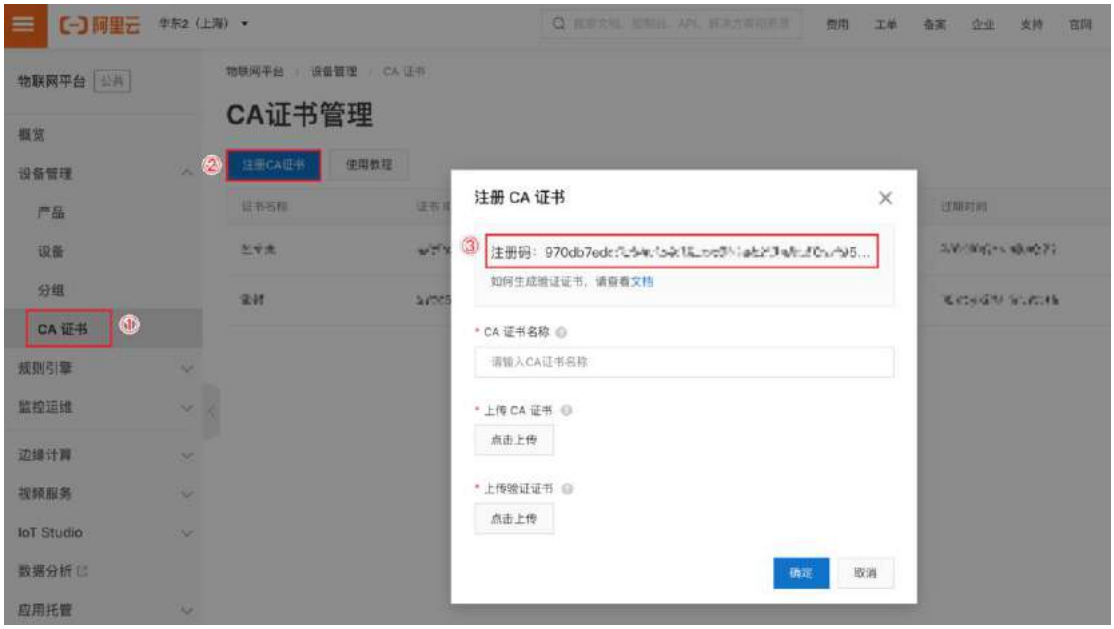
私有 CA 证书内容：




## 二、制作验证证书

### 1. 查看私有证书注册码

- 登录 IoT 物联网平台控制台。
- 在左侧导航栏，选择**设备管理 > CA 证书**。
- 在 CA 证书管理页，单击**注册 CA 证书**。
- 在注册 CA 证书对话框中，获取**注册码**。




### 2. 验证证书制作

我们同样以 OpenSSL 为例，制作验证证书，操作步骤如下：

- 生成验证证书 Key：

```
# 生成验证证书
openssl genrsa -out verificationCert.key 2048
```

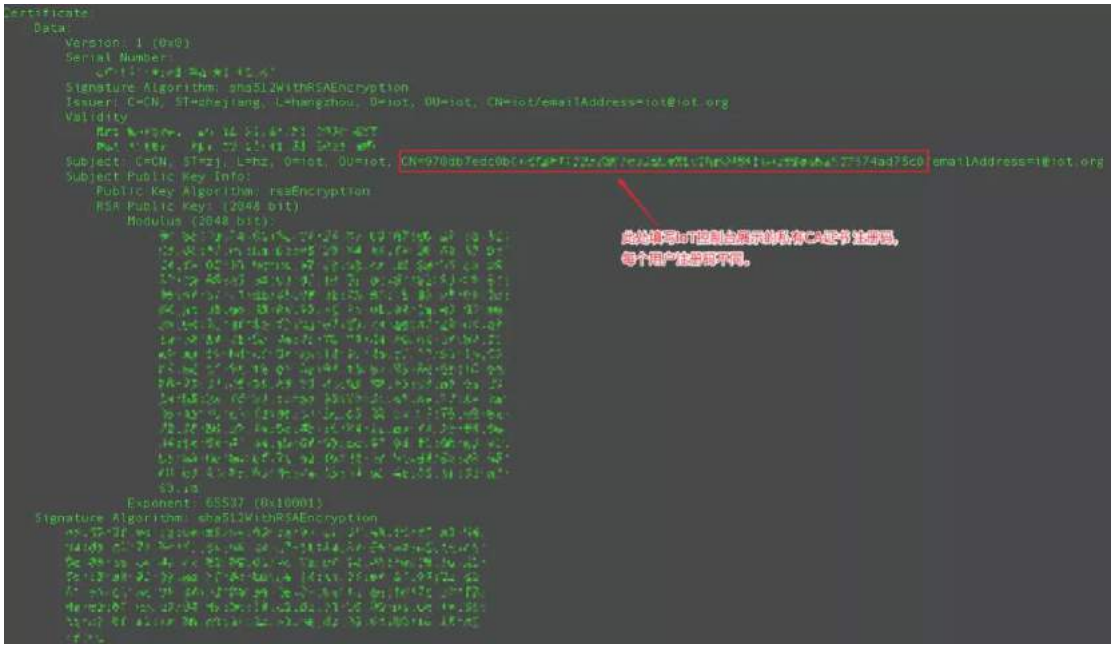
- 生成验证证书 CSR，其中 Common Name 需填入 IoT 控制台获取的私有 CA 证书注册码：

```
# 生成验证证书 CSR
openssl req -new -key verificationCert.key -out verificationCert.csr -subj \
"/C=CN/ST=Shanghai/L=Shanghai/O=IoT/OU=iot/CN=***这里是注册码***"
```

- 使用由私有 CA 证书私钥签名的 CSR 创建验证证书：

```
    # 用私有 CA 和 key 签发验证证书
openssl x509 -req -in verificationCert.csr -CA myIoTCARoot.crt -CAkey myIoTCARoot.ke
y -CAcreateserial -out verificationCert.crt -days 365 -sha512
# 查看验证证书内容
openssl x509 -noout -text -in verificationCert.crt
```

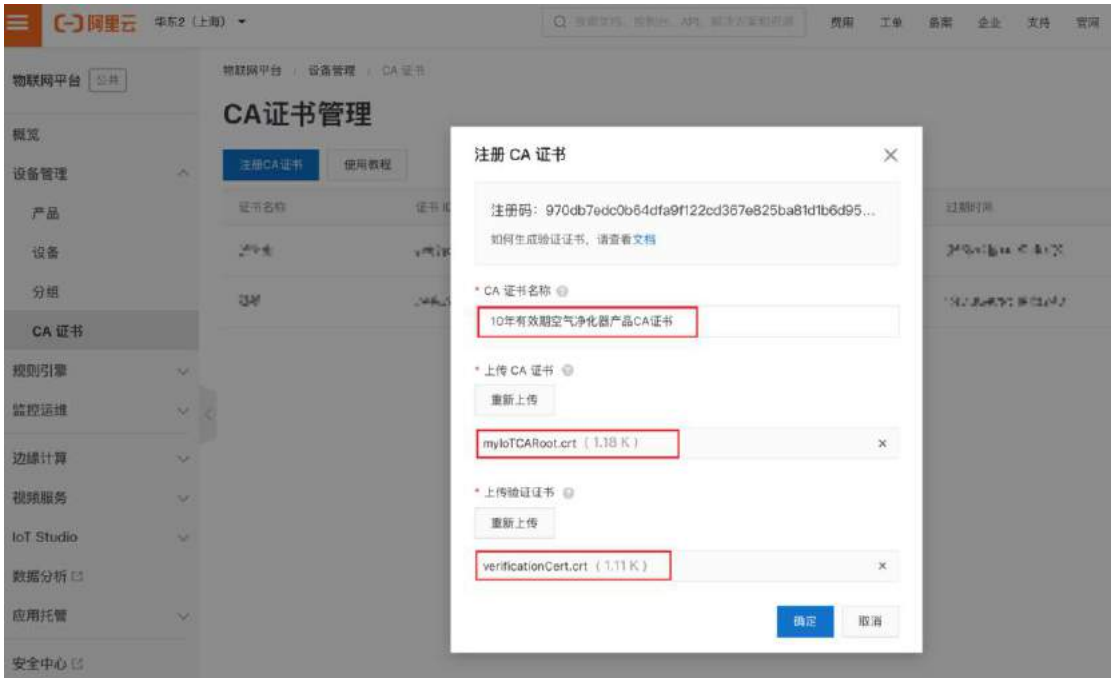
查看验证证书：




## 三、上传并验证私有 CA 证书

当我们准备完成私有 CA 证书和对应验证证书，就可以在 IoT 物联网平台的控制台上传证书了。

上传证书页面如下：




验证通过后，在私有 CA 证书详情页面，可以查看证书信息，参考如下：


至此，我们完成了私有 CA 证书的制作和在 IoT 物联网平台的注册。后续，我们会介绍如何用此 CA 证书来签发设备证书，从而使设备以 X.509 证书认证方式接入 IoT 物联网平台。

# 设备用 X.509 证书接入实战(二)

作者 | 苏堤嘉木

本次开发实战，我们介绍设备绑定 X.509 证书和激活的过程。

## 一、创建产品和注册设备

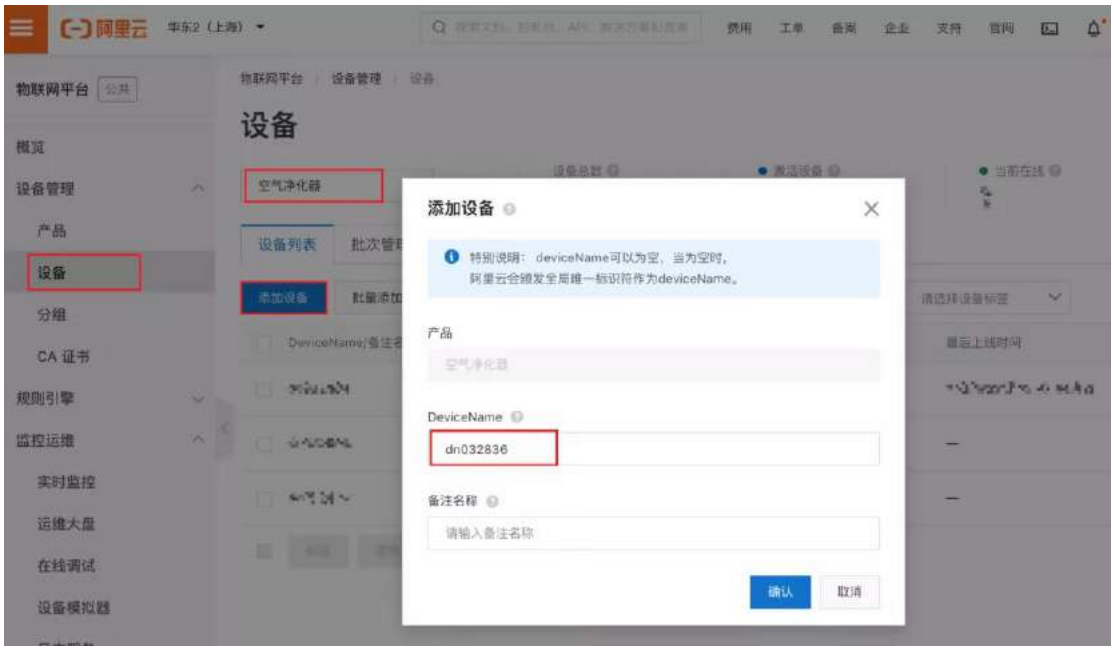
### 1. 创建产品

- 登录 IoT 物联网平台**控制台**。
- 在左侧导航栏，选择**设备管理 > 产品**。
- 在产品管理页，单击**创建产品**。
- 选择直连设备，认证方式选择 **X.509 证书**，使用私有 CA 证书勾选**是**。

产品创建完成后，如下图：


## 2. 注册设备

基于已创建的空气净化器产品，添加设备，输入设备 deviceName。




注册完成后，设备处于**未激活**状态，其中 **X.509 证书为空**。

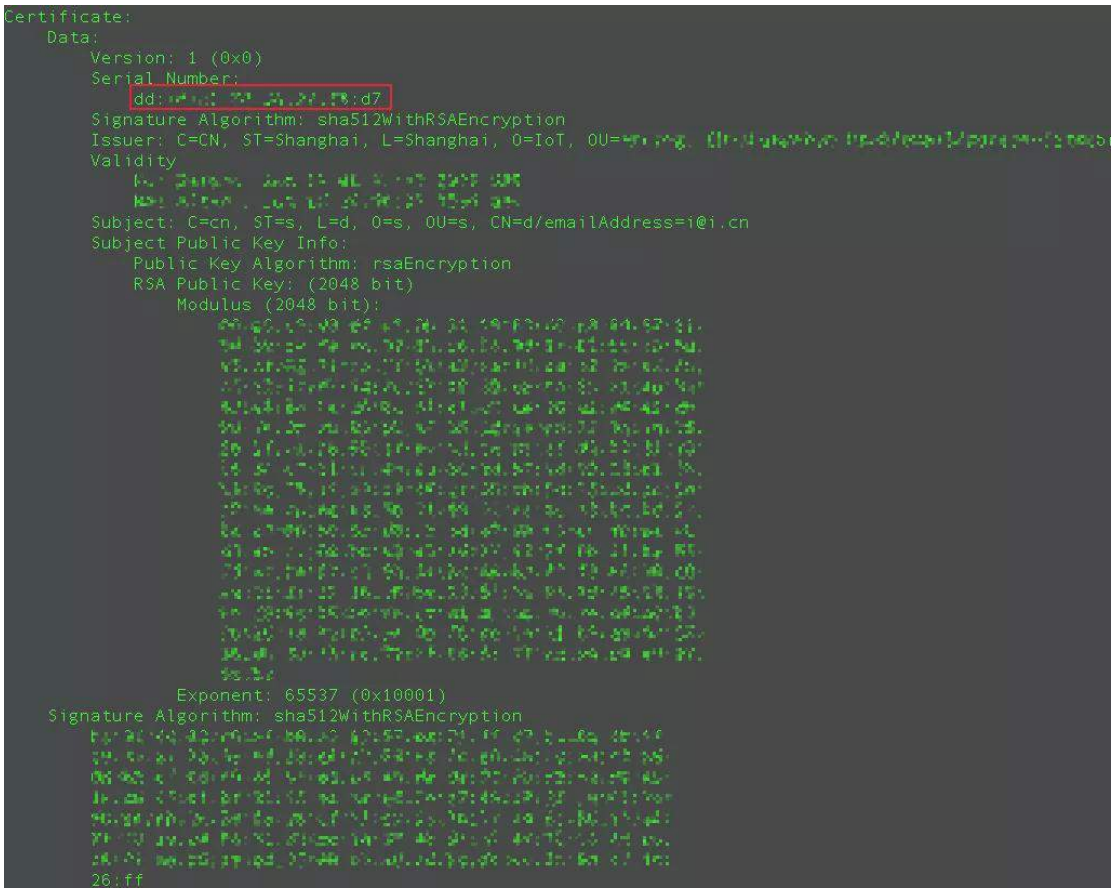

## 二、签发设备证书

我们使用已经在 IoT 物联网平台注册的私有 CA 证书，来签发设备证书。

openssl 的操作指令如下，其中 CN 可以填写 deviceName。

```
    # 生成 pem 的私有 key
openssl genrsa -out device-1.key 2048
# 生成设备证书 CSR
openssl req -new -key device-1.key -out device-1.csr -newkey  rsa:2048 -subj \
"/C=CN/ST=Shanghai/L=Shanghai/O=IoT/OU=iot/CN=dn3023842"
# -set_serial 指定序列号

# 用私有 CA 签发设备证书 CRT
openssl x509 -req -in device-1.csr -CA myIoTCARoot.crt -CAkey myIoTCARoot.key -CAcr
eateserial -out device-1.crt -days 3650 -sha512
# 查看设备证书 SN
openssl x509 -noout -text -in device-1.crt
```

证书生成后，我们可以查看设备证书的 SN 码，如下图。

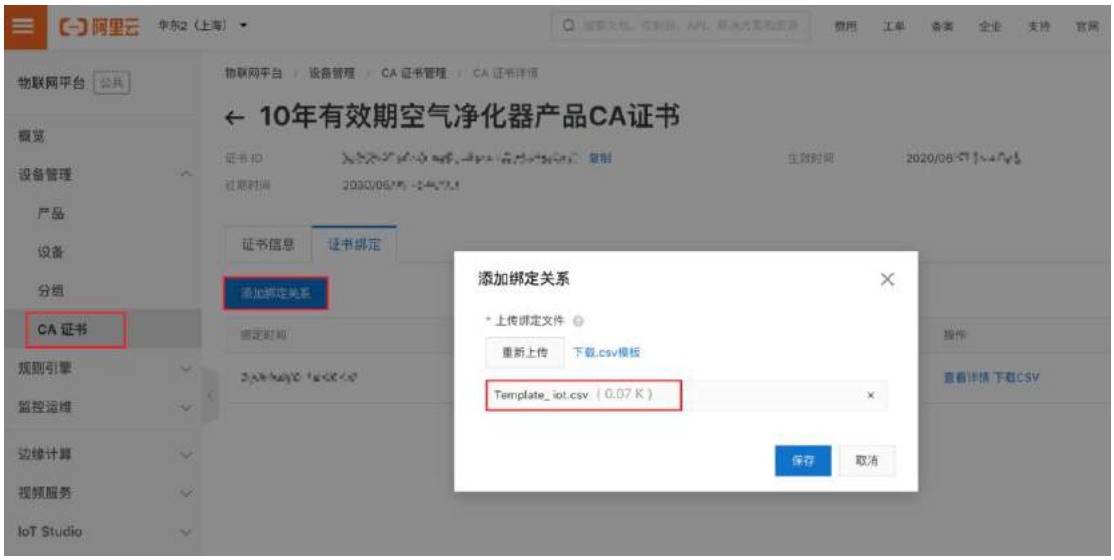



## 三、绑定设备证书

IoT 物联网平台需要用户上传设备 ProductKey,DeviceName 和设备证书 SN 的映射关系的 csv 文件，内容格式如下：

```
ProductKey,DeviceName,CertSN
a15GiUoOzAL,dn032836,dd3dc180a02768d7
```

IoT 物联网平台控制台证书详情页面，上传 CSV 文件，完成绑定。




绑定成功后，我们查看绑定结果，如下图：


## 四、设备激活

我们以 Java 设备为例，介绍设备以私有证书接入过程。

- Java 原生代码只能使用 PKCS#8 格式，我们需要用 OpenSSL 来进行转换，命令如下：

```
    # 转换格式 PKCS#8
openssl pkcs8 -topk8 -inform PEM -in device-1.key -out device-1_pkcs8.key -nocrypt
```

- 使用 TLS 方式( securemode=2 )将设备接入物联网平台，需使用物联网平台根证书。
- 使用 iot_root.crt、设备证书、设备证书私钥来构造 SSLSocketFactory 实例。

```
    protected SSLSocketFactory createSSLSocket() throws Exception {

    // 物联网平台根证书，可以从官网文档中下载 https://help.aliyun.com/document_detail/73742.html
    // 设备 X.509 证书，可以从控制台设备信息中下载。
    // 用来验证 IoT 平台的 CA 证书
    InputStream in = IotMqttClientWithAuthByX509.class.getResourceAsStream("/iot_root.crt");
    CertificateFactory cf = CertificateFactory.getInstance("X.509");
    Certificate ca = cf.generateCertificate(in);
    in.close();
    KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
    keyStore.load(null, null);
    keyStore.setCertificateEntry("ca", ca);
    TrustManagerFactory tmf = TrustManagerFactory.getInstance(TrustManagerFactory.getDefaultAlgorithm());
    tmf.init(keyStore);

    // 传入设备证书、证书证书私钥
    InputStream certIn = IotMqttClientWithAuthByX509.class.getResourceAsStream(certPath);
    CertificateFactory certCf = CertificateFactory.getInstance("X.509");
    Certificate certCa = certCf.generateCertificate(certIn);
    certIn.close();
    KeyStore ks = KeyStore.getInstance(KeyStore.getDefaultType());
    ks.load(null, null);
    ks.setCertificateEntry("certificate", certCa);

    PrivateKey privateKey = getPrivateKey(privateKeyPath);
    ks.setKeyEntry("private-key", privateKey, privateKeyPassword.toCharArray(), new Certificate[] { certCa });
    KeyManagerFactory kmf = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
    kmf.init(ks, privateKeyPassword.toCharArray());

    // 构造 socketFactory
```

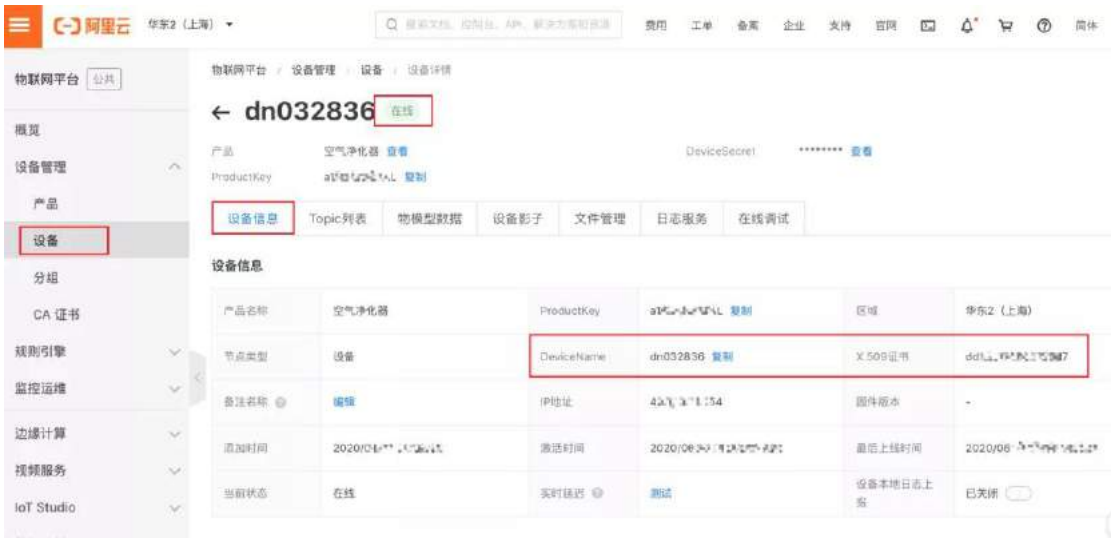
```
            SSLContext context = SSLContext.getInstance("TLSV1.2");
        context.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);
        SSLSocketFactory socketFactory = context.getSocketFactory();
        return socketFactory;
    }
```

设备发起 MQTT 的 CONNECT，此时 username 和 password 无需设置：

```
    // 接入域名
String broker = "ssl://x509.itls." + regionId + ".aliyuncs.com:1883";
// MQTT 的 clientId
String mqttClientId = System.currentTimeMillis()+"|securemode=2|";
MemoryPersistence persistence = new MemoryPersistence();
mqttClient = new MqttClient(serverURL, mqttClientId, persistence);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setMqttVersion(4);// MQTT 3.1.1
// 使用 TLS，需要下载根证书 root.crt，mqttClientId 中设置 securemode=2。
connOpts.setSocketFactory(createSSLSocket());
connOpts.setCleanSession(false);
connOpts.setAutomaticReconnect(true);
connOpts.setKeepAliveInterval(300);
// 设置 connect 回调
mqttClient.setCallback(new MqttCallback() {
    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception {
        // 只处理 X.509 认证返回信息
        if ("/ext/auth/identity/response".equals(topic)) {
            JSONObject json = JSONObject.parseObject(new String(message.getPayload(), Stan
dardCharsets.UTF_8));
            // 获取到设备的 productKey 和 deviceName
            String productKey = json.getString("productKey");
            String deviceName = json.getString("deviceName");
        } else {
            // 处理其他下行消息，强烈建议另起线程处理，以免回调堵塞。
        }
    }
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
```

```
        }
    @Override
    public void connectionLost(Throwable cause) {
    }
});
mqttClient.connect(connOpts);
```

设备启动，联网后，我们可以在 IoT 物联网平台控制台查看设备当前状态为在线，X.509 证书栏为绑定设备证书的序列号，如下图。




至此，我们完成了设备与设备证书绑定，从而使设备以 X.509 证书认证方式接入 IoT 物联网平台。

# 消息处理

# 深度解读 IoT 消息洪峰怎么扛

作者 | IoT 物联网技术

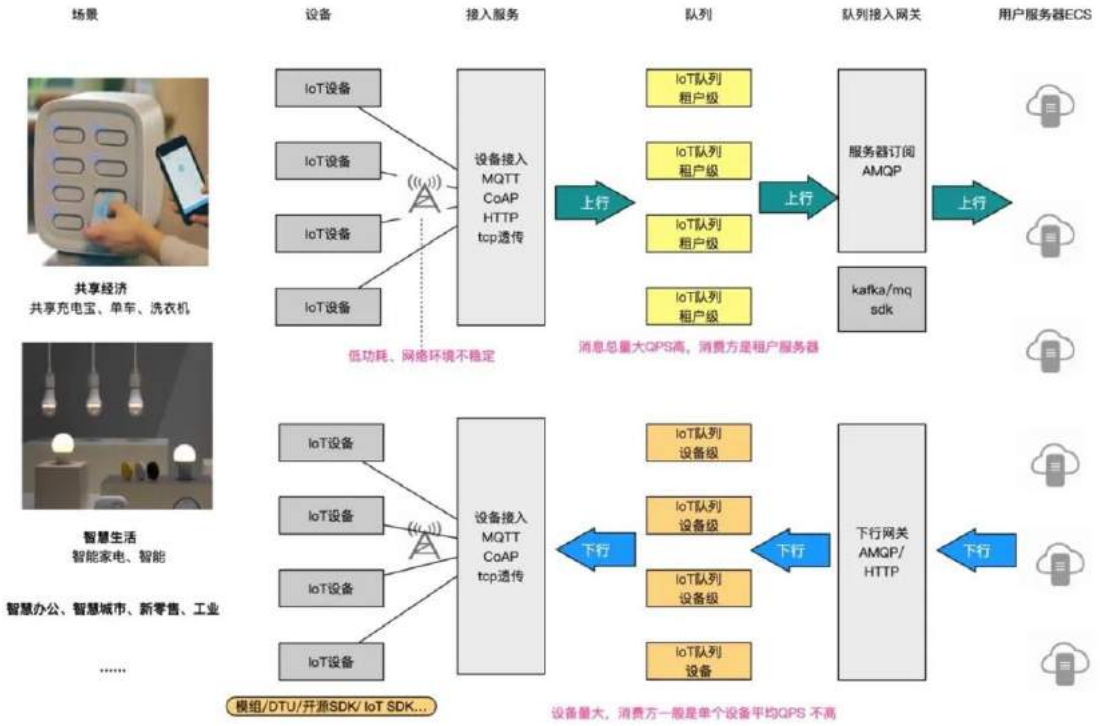
传统的消息队列(Kafka、RocketMQ 等)经过多年打磨，在高性能、海量堆积、消息可靠性等诸多方面都已经做得非常极致，但在 **IoT 物联网场景**中，往往需要面临着海量的消息传递，传统的消息队列表现的“力不从心”。

IoT 场景中，从应用服务器到嵌入式芯片，都需要传递事件消息，比如共享充电宝的开柜子、开灯指令从服务器发到设备、工业网关高频消息流等，在这些信息传递的过程中，队列最大意义在于让整个**消息在不可控的环境中平稳运行**，因为 IoT 设备时不时会由于故障或网络抖动会导致大量消息洪峰。

## 一、IoT 队列和普通队列的差异点

### 1. 上下行隔离拆分

在 IoT 场景中，我们把需要队列分为两个场景，一个是上行队列，一个是下行队列。拆分之后，可以隔离上下行链路，控制一个设备，比如支付成功要下发打开柜子等，上行出任何问题，千万不能影响到下行业务。另外，上下行两条链路的特点差异非常大。设备上行消息，并发量非常高，但很多场景下对于可靠性和时延要求低，而设备下行消息，并发量则比较低，但下行消息（一般是控制设备指令）要求到达成功率很高。




## 2. 支持设备级的海量 topic

传统队列的核心诉求是，不论堆积多少不影响它的性能。kafka 的 topic 一多，原本消息顺序写文件优势就会导致一个 broker 要退化到随机写，失去优势，另外要 zookeeper 来协调这么多 topic 也是有局限，所以这些队列本身有提供一个外挂代理桥接器对外入口是多个设备 topic，再桥接映射到少量的实际 kafka topic，这方案有一定可行性，但做不到隔离效果，治标不治本。

通过，图 1 和图 2 对比较明显，一个队列拥塞尽量减少对其它设备影响。我们需要的是“海量 topic 尽量相互隔离，并且不影响整体性能”，尽量做到设备 A 的消息堆积 topic，不影响设备 B。

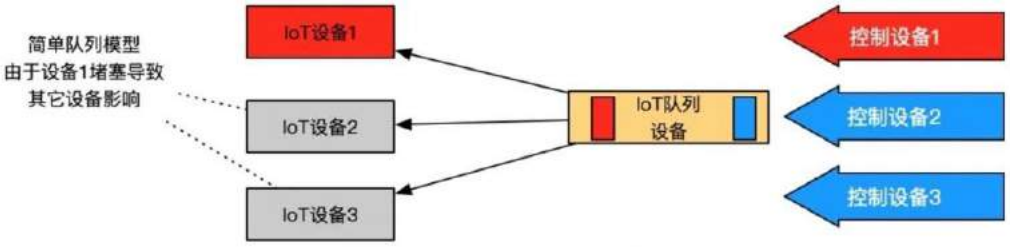



图1: 简单队列模型

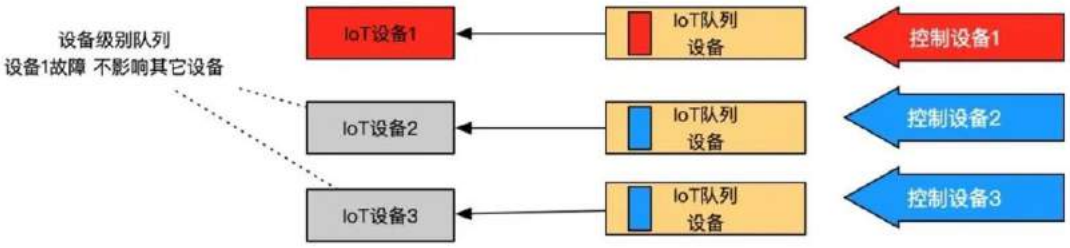


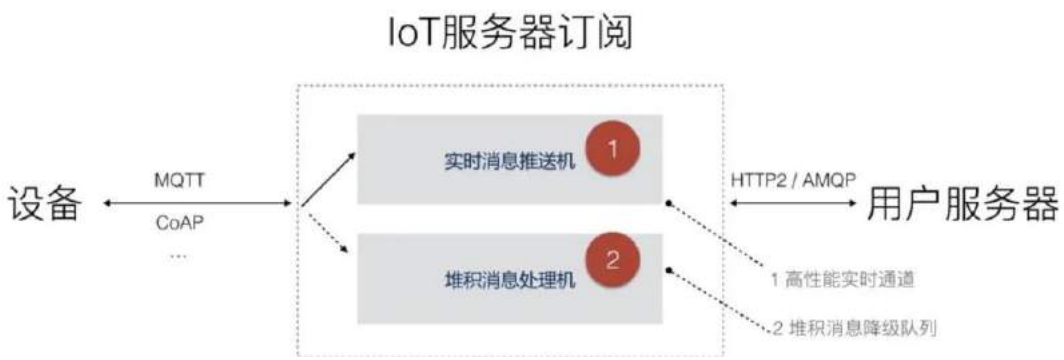

图2: 多泳道队列模型

### 3. 实时生成消息优先发送

先举一个例子，一个快递柜业务的队列堆积，然后“此时此刻”在柜子旁边的用户死命的在旁边用手机点开柜子怎么也打不开（此时后端系统都恢复了），问题就是队列里面还有几十万条的消息，新来的消息需要排队，等着之前的那些消息消费完，甭管这些消息还有没有用。 因此，实时生成消息优先发送，堆积的消息进入降级模式。

## 二、IoT 消息队列诞生

### 1. IoT 队列的设计思路

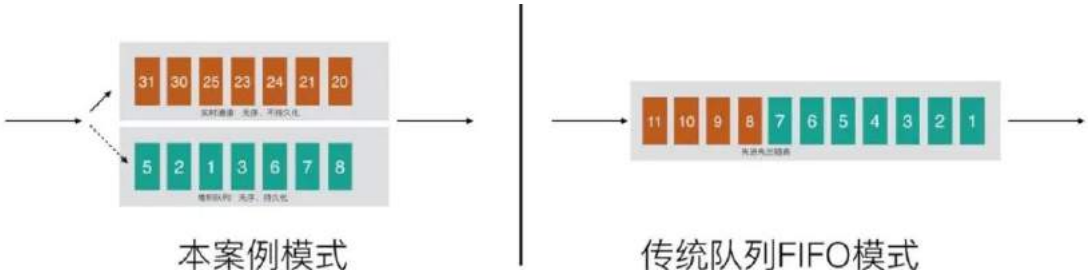


设计目标是为了打造一个支持上下行隔离、实时优先、及海量 topic 的队列网关，设计原则如下：

- 完全 follow 开源生态、和传统队列互补兼容。
- 保序降级，实时优先，堆积退化；仅实时消息相对有序。
- 海量 topic，多租户隔离。
- 连接、计算、存储分离。

### 2. 消息模式

图片只是个片段，从这个模式可以看出来机制差别，大家都没有错，只是出发点不同。

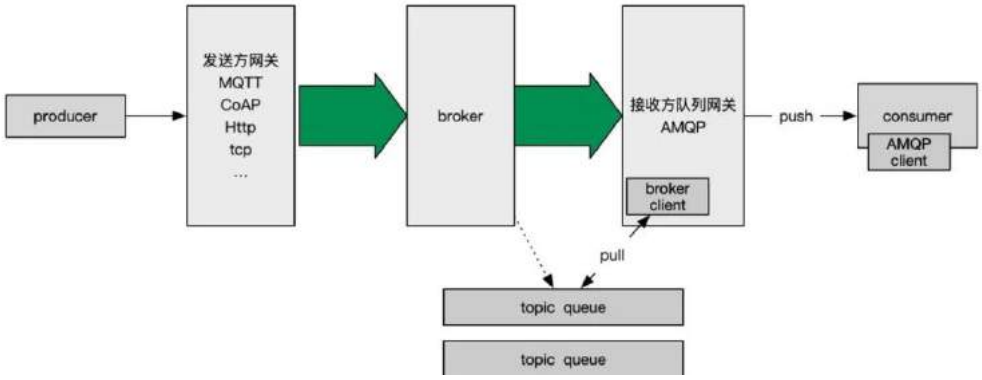



### 3. 连接、计算、存储分离


broker 不做连接，连接网关代理，broker 只做流转分发，无状态+水平扩展；存储交给 nosql DB，高吞吐写。

### 4. 消息策略-推拉结合

这个应该是队列的核心难点之一，和传统队列区分在于，我们考虑为平台化模式，独享资源过于昂贵。但带来问题是消费端不可控，所以使用结合模式，只有在消费者在线时会拉取堆积消息，而拉取是由 AMQP 队列网关来做，给到用户接口始终是推送过去的 onMessage 回调。




- broker 不是直接让 consumer 来连接，而是把队列网关剥离出来， 这样会更灵活，甚至对于部分用户我们的 queue 可以切换到 ons、kafka 等实现。kafka、rocketmq 做法是在连接时会分配给客户端一个 broker 接入地址。
- broker 实时消息优先推送给 consumer，失败才会落到 queue ；这是一个完整事件，如果没有完成则不给 producer commit。
- 异步 ACK。

### 5. 线性扩展-离线消息部分

实时部分消息采用推方式，基本上不会成为瓶颈，消费不过来消息进入堆积模式。由于底层依赖存储已经帮我们解决核心存储的扩展，剩下主要问题点在于如何消除写入热点和消费热点，这样 broker 可以完全做到无状态。

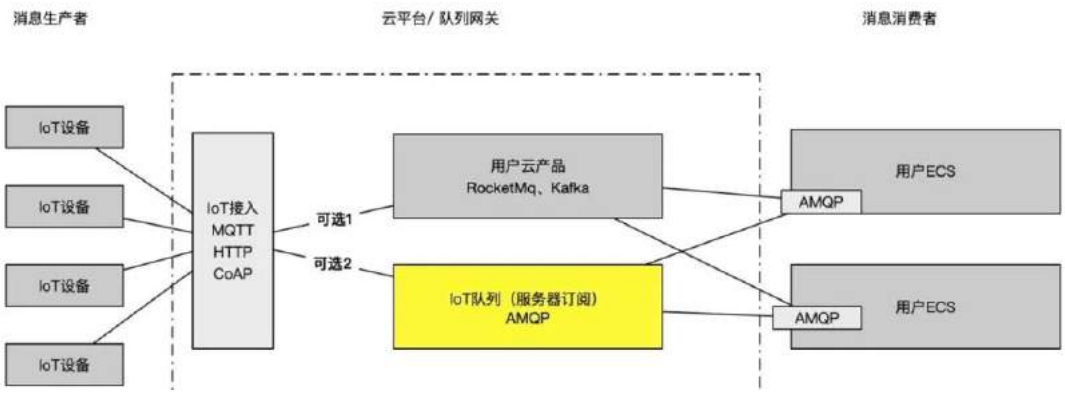
## 三、一个思考——如何解决海量 topic 问题?

首先面对“大量”的问题一般都是考虑分区，单元化，分组等隔离和拆分，这里海量 topic 我们讨论针对一个单实例模式下如何尽可能做到更多 topic，完全任意数量都能 100% 没问题肯定是不现实的。

由于 broker 和存储已经隔离，broker 和 topic 已经没有什么关系，或者说任何 topic 数据生成，broker 做的事情就是写入和分发。

- 海量 topic，每个 topic 有限数量订阅： topic 和订阅者关系使用 redis 缓存或本地缓存，针对 mqtt topic 匹配有个 topic tree 的树算法，hivemq 有实现版本。
- 单个 topic 海量订阅： 这个场景其实是组播和广播，我们不会考虑在队列本身上面去做这个事情，而是在上层封装广播组件来协调任务和批量发送。

## 四、阿里云 AIoT 消息队列




目前阿里云 AIoT 队列，也叫服务端订阅，意思就是用户用服务端订阅他们设备消息。为了降低接入成本，用户可以使用 AMQP1.0 协议接入，符合开源生态。 同时兼容传统队列和新队列，交给用户按场景来选择，用户即可选择使用 kafka、mq，也可以选用 iot 队列，甚至组合模式，比如按消息特征规则来配置流转队列。

阿里云 AIoT 的场景队列实践，在现有 mq 队列、kafka 队列融合之外，加了种自有的实时优先队列实现，同时，加入了队列网关代理，既能让用户选择普通消息队列，也可以选择轻便的 IoT 消息队列。

# 亿级 IoT 设备连接底层逻辑

作者 | IoT 物联网技术

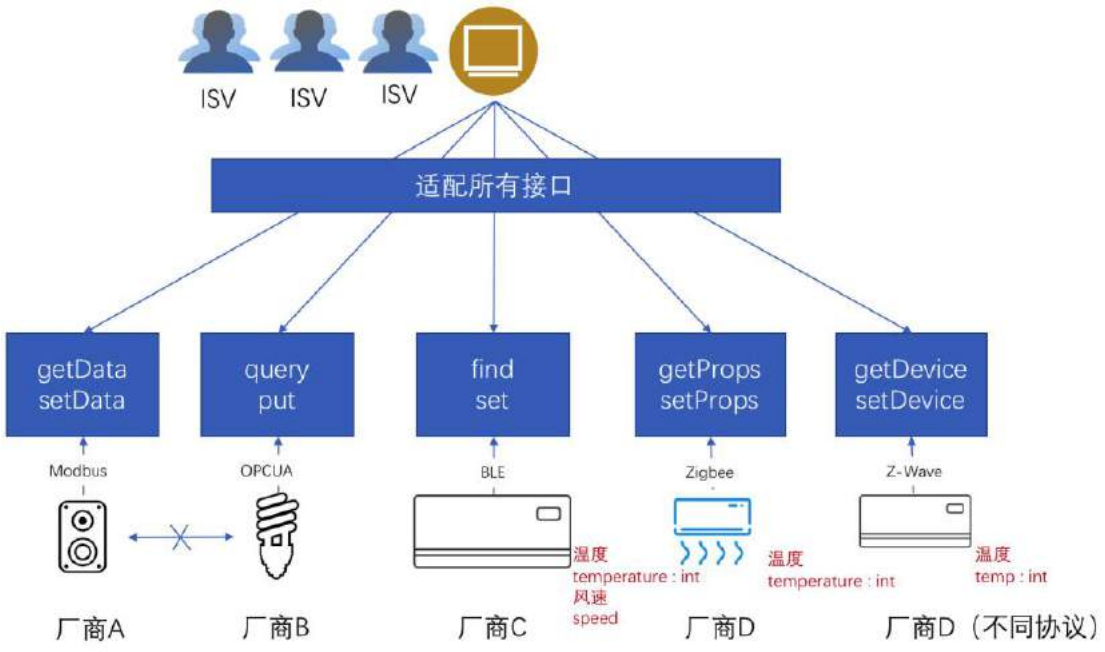
**物模型**技术对于物联网企业来说是一项非常重要的技术，因为要实现万物互联，必须要有物模型体系沉淀，才能够让各种**硬件实现智能化连接**。

今天，阿里云 AIoT 物模型技术专家**熊益群**，为大家带来了一份物模型技术全攻略，解析物模型技术为什么这么重要?

## 一、物模型三个关键问题:

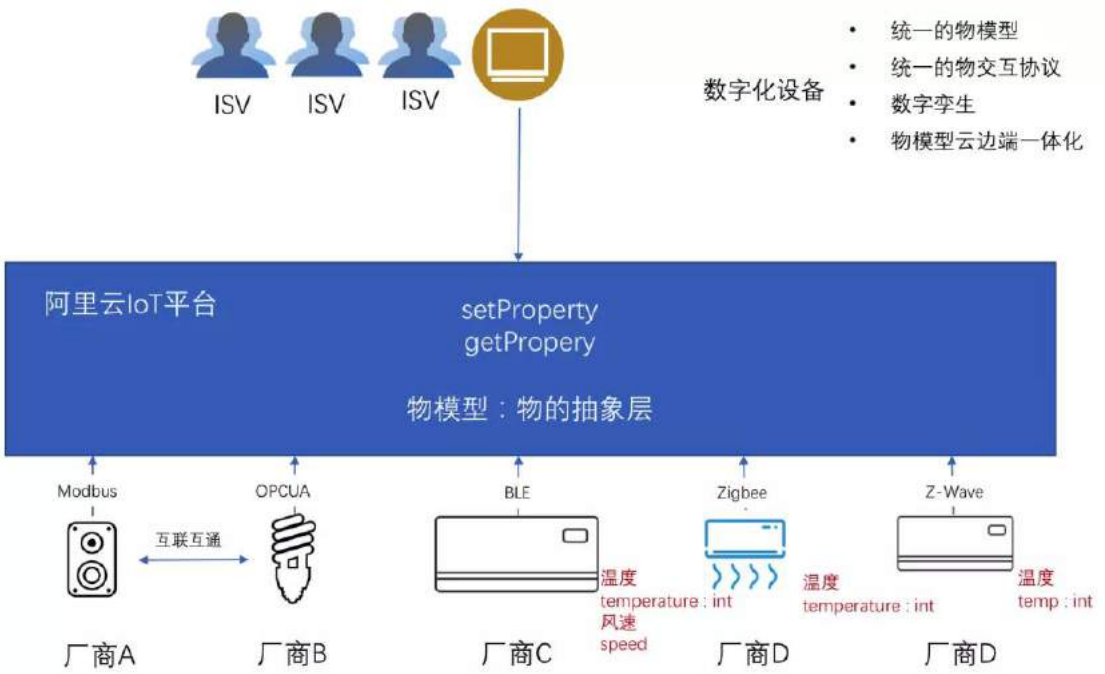
### 1. 为什么需要物模型?

海量的物联网数据、设备、业务，异构的设备和数据描述方式，难以理解，互通困难，首先，产业链内部自成体系，模组、芯片、平台、方案商角色多样，跨角色协作时，数据标准各异，协作困难；其次，采集数据解析困难，难以结构化，数据利用效率低，数据价值难挖掘；最后，随着行业应用和设备量增长，新增应用需要针对不同的设备协议重复开发，难以规模化。



### 2. 物模型能解决那些行业问题?

目前物联网行业普遍存在着设备孤岛、软硬开发强耦合的问题，需要构建模型统一描述语言、面向物理实体的统一建模，物模型作为物的抽象层屏蔽了底层终端差异，标准化了设备的能力表达和交互方式，极大降低了物联网应用开发和快速复制的成本。



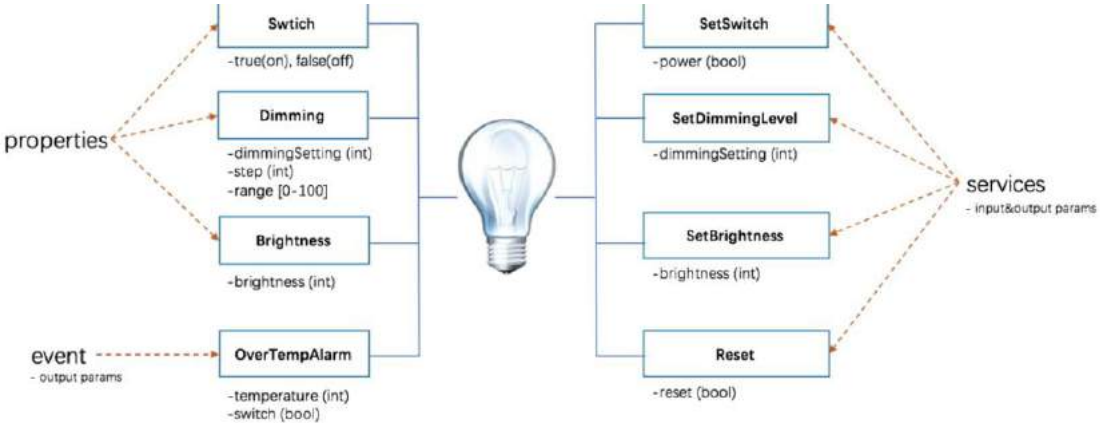

### 3. 物模型带来什么价值?

- **低门槛接入：**提供设备建模和交互协议基础能力。这是最基础的价值，所有设备上云都需要建模和交互协议。物模型和协议设计是否足够专业，这其实是绝大多数中小企业的门槛，他们刚开始意识不到，随意设计，随着规模和业务变化弊端就会体现出来。
- **标准化：**物模型作为物联网的抽象层，类似操作系统屏蔽硬件、JVM 屏蔽 OS 的差异性一样，通过标准化设备的能力表达和交互方式，解决了物联网严重碎片化情况下协议差异、软硬开发耦合、全链路验证流程长、设备孤岛、数据孤岛等问题。
- **生态化：**软、硬件一旦基于物模型标准化开发和交互，围绕物联网的多角色，包括 ISV，SI，IHV 等在设备开发、生产、运维、售卖、集成、运行等环节相互之间能够解耦，提升了设备的流通性，促进生态化。

## 二、物模型面临哪些技术挑战?

**以一个灯泡为例：**

我们先来看一看一盏普通的智能灯会有哪些能力或特性，比如开关、色调、亮度、过温告警、恢复出厂设置等能力，其中包含有传感器采集的状态、有危险告警、也有控制器可执行的指令。那么不同行业场景设备复杂度、差异性都不一样，简单到消费类设备"灯"、复杂到工业类设备"锅炉"都需要可表达，定义一套足够抽象通用面向万物的物模型还是非常有挑战的，因此需要遵循一定的设计原则，比如简单、普适、可扩展、模块化、易用性。




**延展开来说，物模型的技术挑战具体有这几项：**

物模型由于描述所有异构设备完整能力，而且在设备全生命周期都发挥着作用，因此物模型设计过程中存在以下需要解决的难题：

- **普适性：**物模型的定义和设计能够适应所有设备，需要可覆盖工业、生活、农业、交通多个不同行业。因此在设计上需要找到设备最本质的共性、抽象出一套模型。
- **超大点位和超复杂结构：**尤其工业场景，通常需要对包含大量传感器（万级别）的传统自动化系统进行数字化，对物模型提出了非常大的挑战，物模型复杂度变成了和物理实体和环境复杂度成正相关，我们需要从中找到最本质的破解方法，避免物模型复杂度变得不可控。
- **国际化：**设备可以在任何阶段售往全球各地，物模型能够让设备在各地具备多语言的能力。
- **可插拔：**工业文明发展快很重要的一点在于标准化，大量复杂设备都可以标准化组装而成，比如汽车、轮船、家居等，模块可以根据产品特性进行动态插拔，因此物模型同样需要能够适应物理设备模块可插拔特性。

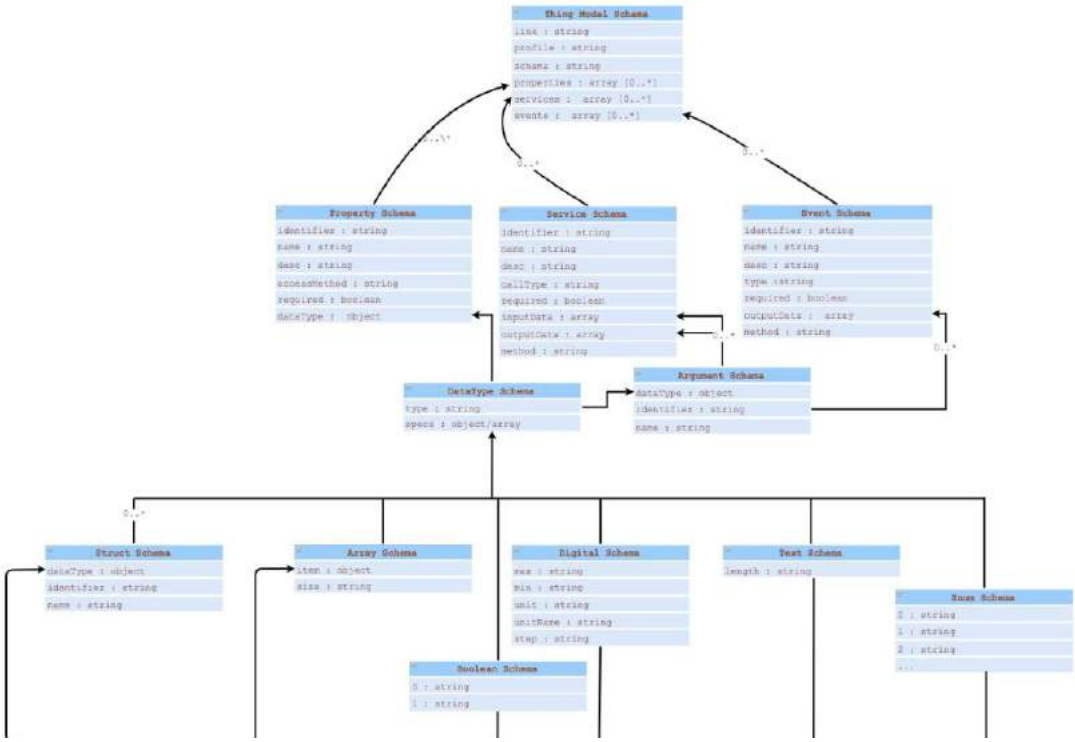
- **安全开发：**物模型在设备开发阶段定义，在设备运行阶段被引用，需要保障开发阶段定义或调试的物模型不影响生产阶段正在运行的设备。
- **快速调试：**传统硬件开发和软件开发需要全链路一起配合调试，周期长成本高。有了物模型，调试阶段需要确保软硬解耦，不相互依赖。
- **高可靠：**线下化是物联网与互联网很大的差异点，大量线下的物理设备，地理位置和应用场景及其广泛，设备出现问题现场运维成本非常高，而且对社会影响大，因此物模型在设备运行阶段的可靠性要求非常高。
- **可回滚：**为了保障高可靠，物模型在开发到运行过程中，一旦出现异常需要确保可快速回滚。
- **可适配：**由于行业里面已经有不少设备模型和交互协议，比如工业场景的 Modbus，opc 等，生活场景的 ble，zigbee 等，当然还有大量三方平台私有协议，为了帮助这些存量设备能够使用物模型，物模型需要具备模型和协议适配能力。
- **统一交互协议：**设备除了需要可表达之外，还需要可访问，物模型不仅需要定义设备能力描述规范，还需要定义设备被访问方式，所有设备都能够使用同一套交互协议进行访问，设计上也存在着不小挑战，比如资源受限设备、弱网环境设备、工业边缘设备对协议要求都会不一样，有些追求低功耗、有些追求少带宽、有些追求大点位高频、也有追求网络多级级联等等。
- **孪生代理：**物模型核心价值在于物理实体数字化，物理实体在云上数字化后会构建数字孪生体，数字孪生体的数据模型、访问方式均基于物模型，数字孪生体代理物理设备与行业应用进行交互，从而达到软件与硬件的解耦。然而孪生代理应该具备哪些特性以确保硬件能力都可以高效可靠地访问是非常有挑战的，比如设备断网或异常情况孪生代理如何与应用交互，是确保指令必达还是快速失败，可能不同场景诉求不一样。当然具备海量数据的孪生体如何基于数据智能化，反向指导物理设备生产运维，达到和物理设备共智的目标这是更大的挑战。

**我们应该如何设计物模型呢？**

早期大多数物联网平台比如 Azure、AWS 都只做连接和基础管理能力，并没有围绕数字化的设备建模和数字孪生能力，不过这两年几乎所有物联网平台都开始重视物模型和数字孪生的建设。

大多数对于设备建模都采用的是面向对象语言的思路，比如 WoT、OPC、OMA、OCF、CWMP、AllJoin 等，面向对象语言的抽象能力在计算机编程发展的几十年已经被证明，我们物模型定义也充分借鉴，却又因物联网而有所不同。

我们以面向对象语言 java 里面的 class 做类比，class 用属性和方法描述对象的状态和行为；物模型也可以用属性和方法来描述物的状态和行为。同时结合设备特性，我们将物模型 schema 进行了一定的扩展，定义为属性、服务(方法)和事件三要素，事件是一类特殊的属性，比如空调的故障告警，这类属性严重性高，实时性强，一般需要监控并及时响应。为了对设备更精确的描述，物模型针对每种数据类型还定义了非常严谨的数据规范，比如在数据类型之外，还需要定义数据范围、精度、步长等规范。




【图为物模型基础 schema（没有包括模块化、多语言、多版本等一系列高阶特性）】

**解决了这些挑战后，物模型的技术架构就呈现出来了。**

阿里云 AIoT 物模型除了通过属性、事件、服务三要素描述了物理实体能力之外，物模型还支持千级大点位、多语言、多版本、多模块、多级级联、协议适配、云边端一体化等能力，达到可以应对生活、城市、工业等不同场景定义诉求。当然为了应对上文提到的一系列技术挑战，我们还通过构建 Alink 协议、数字孪生搭建了一整套面向物理实体的数字化能力。

**还有一点要注意，物模型和数据标准是不一样的。**

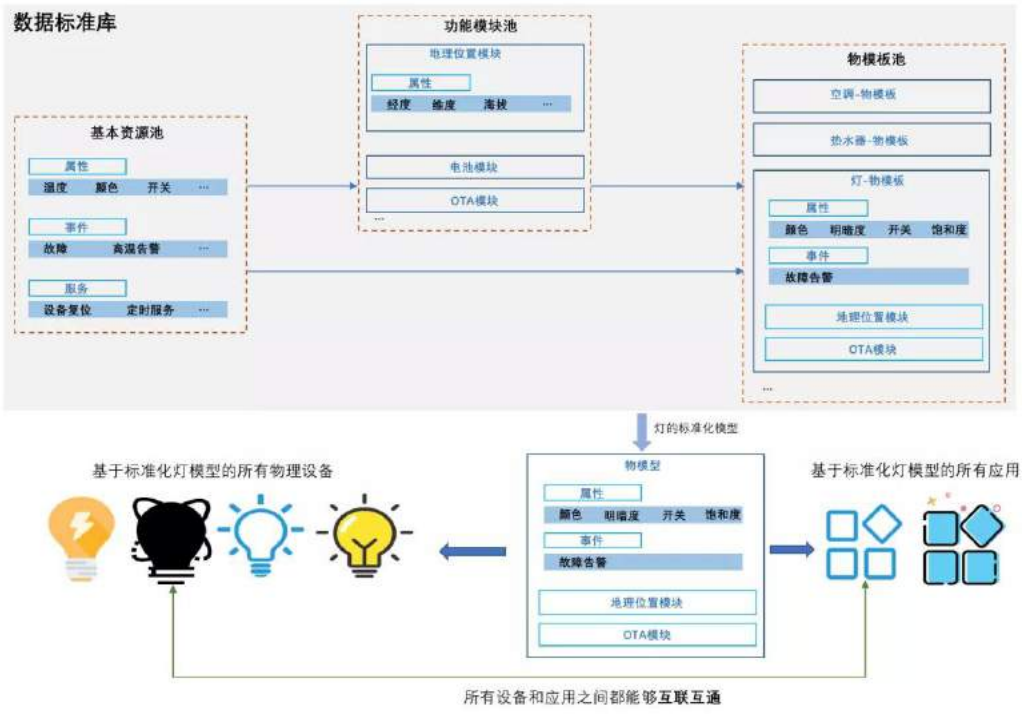
物模型能够以同一套 schema 描述设备的能力，但由于物联网碎片化，大家对于设备能力的定义差异性非常大，同样一款空调，不同厂商定义的能力会不一样。相当于面向对象语言里面接口标准化了，但实现没有标准化。数据标准核心在于降低差异化。

数据标准是一批可用于组装物模型的标准化素材，物模型构建过程可以方便地从数据标准库中选择素材进行积木式搭建。

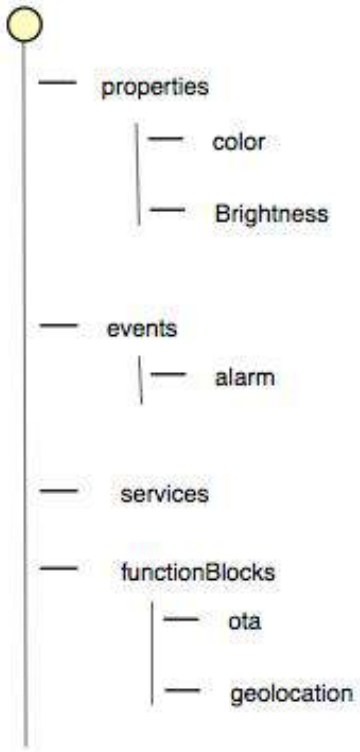
在传统领域碎片化严重的情况下，定义数据标准非常有挑战，通常只有深耕传统行业才能定义出来，因此我们更多的是引入这些行业领先者贡献数据标准，而不是自己制定。阿里云 IoT 数据标准的沉淀主要来自 ICA 标准联盟，ICA 标准库包括基本资源、功能模块、物模板三类素材：

- **资源:**标准库中最原子的能力，有属性、事件、服务三种类型（三要素）；
- **功能模块:**一组资源的集合。集合中的资源可以是标准库中已有资源的组装，也可以是在当前功能模块新增的资源；
- **物模板:**一组功能模块和一组资源的集合。集合中的模块和资源可以是标准库中已有模块和资源的组装，也可以是在当前物模板新增的资源；

下图描述了物模型、数据标准之间的关系：



最终我们看下灯泡物模型示意图：

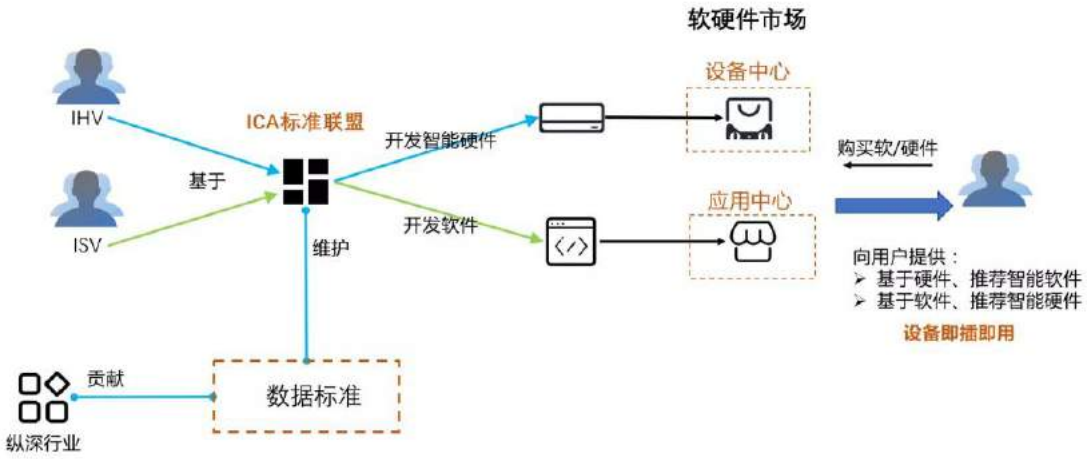



灯物模型TSL

```
{
  "properties":[
    {
      "identifier":"color",
      "name":"颜色",
      "accessMode":"rw",
      "dataType":{
        "type":"text",
        "specs":{
          "length":"32"
        }
      }
    },
    {
      "identifier":"brightness",
      "name":"明暗度",
      "accessMode":"rw",
      "dataType":{
        "type":"int",
        "specs":{
          "min":"0",
          "max":"100"
        }
      }
    }
  ],
  "events":[
    {
      "identifier":"alarm",
      "name":"故障警报",
      "outputData":[
        {
          "identifier":"errorCode",
          "name":"错误码",
          "dataType":{
            "type":"text",
            "specs":{
              "length":"255"
            }
          }
        }
      ]
    }
  ],
  "services":[],
  "functionBlocks":[
    "ota",
    "geolocation"
  ]
}
```

**数据标准的核心价值是什么呢?**

阿里牵头的 ICA 标准联盟，已经沉淀了海量标准化的数据模型，核心价值一是为了建模过程可以快速组装、积木式搭建、提高建模效率；另一方面标准物模板可以促进软硬件标准化，从而实现软件商、集成商对购买的硬件即插即用。



# IoT 平台广播消息 Broadcast 实战

作者 | 苏堤嘉木


## 一、消息广播实现原理

### 1. 业务广播 Topic 定义

```
/broadcast/${YourProductKey}/此处替换为自定义标识
```

### 2. 服务器通过 PubBroadcast API 发送广播

https://help.aliyun.com/document_detail/69909.html

请求参数

| 名称 | 类型 | 是否必需 | 描述 |
| --- | --- | --- | --- |
| Action | String | 是 | 要执行的操作，取值：PubBroadcast。 |
| ProductKey | String | 是 | 要发送广播消息的产品Key。 |
| TopicFullName | String | 是 | 要接收广播消息的Topic全称。格式为：/broadcast/${productKey}/自定义字段 。其中，${productKey}是要接收广播消息的具体产品Key；自定义字段中您可以指定任意字段。<br>一个广播Topic最多可被1,000个设备订阅。<br>如果您的设备超过数量限制，您可以对设备进行分组。例如，如果您有5,000个设备，您可以将设备按每组1,000个，而分成5组。您需要分5次调用广播Topic，自定义字段分别设置为group1/2/3/4/5，然后让每组设备分别订阅各自分组的广播Topic。 |
| MessageContent | String | 是 | 要发送的消息主体。您需要将消息原文转换成二进制数据，并进行Base64编码，从而生成消息主体。 |
| 公共请求参数 | - | 是 | 请参见公共参数。 |

### 3. 各个设备端收到的 payload

```
{"broadcast":"this is broadcast data"}
```

## 二、广播开发实战

### 1. 产品和设备创建

| 设备 | 产品 |
| --- | --- |
| button | a1p36XsaOS7 |
| webApp | a1p36XsaOS7 |
| camera | a1p36XsaOS7 |

### 2. 消息广播 topic 定义

注：broadcast 的 topic 不需要在 IoT 平台预先定义，设备端可以直接订阅

```
/broadcast/a1p36XsaOS7/shareData
```

### 3. 设备端应用程序代码

业务主程序：

```
/**
 * node broadcast-device.js
 */const mqtt = require('aliyun-iot-mqtt');

//设备身份三元组+区域
const options = require("./iot-device-config.json");

//建立连接
const client = mqtt.getAliyunIotMqttClient(options);

//订阅广播
client.subscribe(`/broadcast/${options.productKey}/shareData`)

client.on('message', function(topic, message) {
        console.log("topic " + topic)
        console.log("message " + message)
})
```

iot-device-config.json 设备配置参数：

```
{
    "productKey": "替换 productKey",
    "deviceName": "替换 deviceName",
    "deviceSecret": "替换 deviceSecret",
    "regionId": "cn-shanghai"
}
```

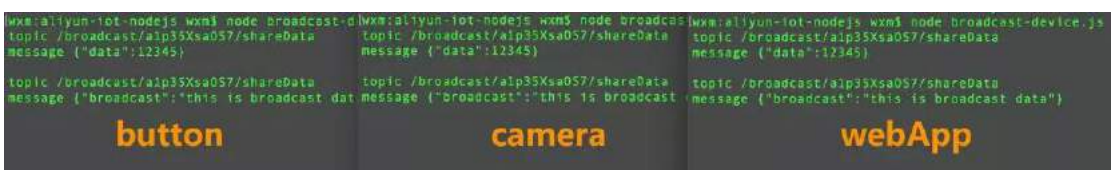
### 4. 业务服务器发送广播

```
/**
 * package.json 添加依赖："@alicloud/pop-core": "1.5.2"
 */const co = require('co');
```

```
    const RPCClient = require('@alicloud/pop-core').RPCClient;

const options = {
        accessKey: "自己的 accessKey",
        accessKeySecret: "自己的 accessKeySecret"
};

//1.创建 client
const client = new RPCClient({
        accessKeyId: options.accessKey,
        secretAccessKey: options.accessKeySecret,
        endpoint: 'https://iot.cn-shanghai.aliyuncs.com',
        apiVersion: '2018-01-20'
});

co(function*() {
        // 2.构造 iot API
        // 这里是 POP API 的 Action
        const action ='PubBroadcast';
        // 这里是 POP API 的入参 params
        const params = {
            ProductKey: "a1p35XsaOS7",
            TopicFullName: "/broadcast/a1p35XsaOS7/shareData",
            MessageContent: new Buffer('{"broadcast":"this is broadcast data"}').toString('ba
se64'),
        };
        //3.发送请求
        const response = yield client.request(action, params);

        console.log(JSON.stringify(response));
});
```

## 5. 运行结果

业务服务器调用 PubBroadcast API 发送广播后，3 个设备收到广播的日志：



# IoT 设备离线时，下行消息方案

作者 | 苏堤嘉木

在物联网场景中，由于**网络不稳定**，导致设备**间歇性离线**状态；电池容量限制，很多 IoT 设备无法做到 24 小时在线，**设备沉睡处于离线状态**；这些现状带来一个新的挑战：**在设备离线时，云端如何发送控制指令给设备？**才能保证设备上线后，按照新的指令执行业务逻辑？

## 一、技术解决方案

面对设备离线场景消息触达诉求，我们有两种通用解决方案：

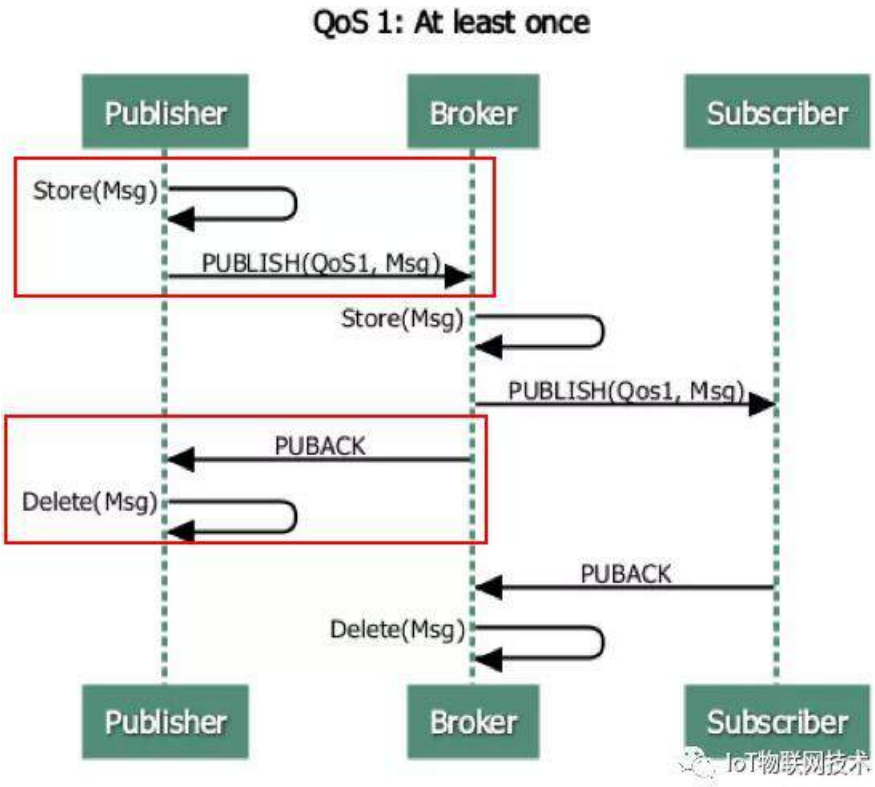
- 基于 MQTT 协议 QoS=1 消息。
- 基于 IoT 物联网平台的设备影子功能。

**方案一、发送 QoS=1 离线消息**

MQTT 协议设计了一套保证消息稳定传输的机制，包括消息应答、存储和重传。在这套机制下，提供了三种不同层次服务质量 QoS（Quality of Service）：

- QoS=0，至多一次；
- QoS=1，至少一次；
- QoS=2，只有一次。

QoS=1 代表，Sender 发送的一条消息，Receiver 至少能收到一次，也就是说 Sender 向 Receiver 发送消息，如果发送失败，会继续重试，直到 Receiver 收到消息为止，但是因为重传的原因，Receiver 有可能会收到重复的消息；




当我们采用 QoS=1 方式发布消息到 IoT 物联网平台，即可保证消息至少到达设备端一次，再结合去重逻辑，重连时保留 Session 信息来实现离线消息触达。

### 1. 设备端配置

设备建立 MQTT 连接时需要配置 CONNECT 参数 CleanSession=0，即保留之前建立的 session 状态，这包括：

- 客户端的订阅信息；
- 未完成确认的 QoS=1 的消息；
- 未发送给客户端的 QoS=1 的消息。

Node.js 实现 CONNECT 参数示例：

```
const options = {
clientId: `${id}|securemode=3,signmethod=hmacsha1,timestamp=${timestamp}|`,
username: `${deviceName}&${productKey}`,
password: "根据文档规则进行 hmacsha1 加密",
```

```
host: `${productKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com`,
protocol: "mqtt",
clean: false, //重连后保持 Session
keepalive: 300
}
```

### 2. 服务端调用

服务端调用 IoT 物联网平台的 Pub API，并指定 Qos =1 。完整 API 文档参考 https://help.aliyun.com/document_detail/69793.html

**请求参数**

| 名称 | 类型 | 是否必需 | 描述 |
| --- | --- | --- | --- |
| Action | String | 是 | 要执行的操作，取值：Pub。 |
| ProductKey | String | 是 | 要发送消息产品Key。 |
| TopicFullName | String | 是 | 要接收消息的Topic，<br>如 /a1Q5XoY****/device1/user/update 。<br>您可以调用QueryProductTopic接口查询产品下的Topic类列表，或在设备详情页的**Topic列表**页签下查看设备的具体Topic。 |
| MessageContent | String | 是 | 要发送的消息主体。您需要将消息原文转换成二进制数据，并进行Base64编码，从而生成消息主体。 |
| Qos | Integer | 否 | 指定消息的发送方式。取值：<br>0：最多发送一次。<br>1：最少发送一次。IoT平台最多保留7天<br>如果不传入此参数，则使用默认值0。 |
| 公共请求参数 | - | 是 | 请参见公共参数。 |



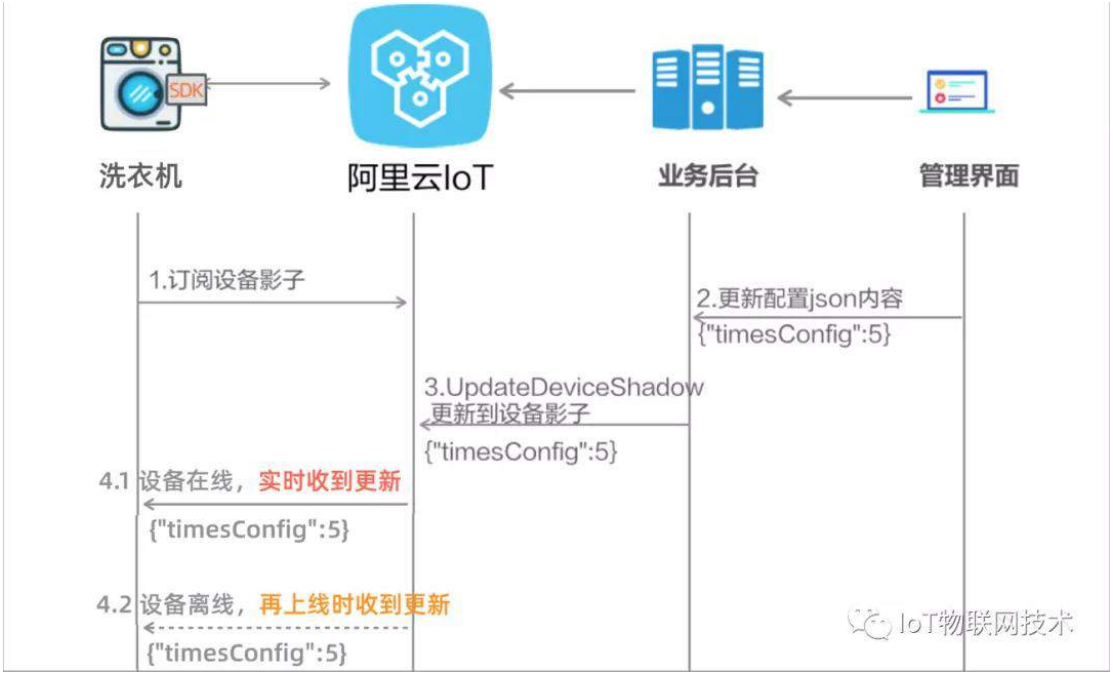
Node.js 调用 Pub API 示例:

```
//1.构建 Pub API 请求报文
const params = {
TopicFullName: "下行指令的完整 Topic",
MessageContent: "消息体的 base64 编码",
ProductKey: "产品 ProductKey",
IotInstanceId: "实例化 Id",
Qos: 1 // 设备离线时，IoT 平台缓存 7 天
};
```

```
//2.发起 Pub API 调用
const response = yield client.request('Pub', params);
```

### 方案二、设备影子实现离线控制

IoT 物联网平台提供设备影子功能，可以实现离线设备的消息触达，完整消息链路如下如:




#### 1. 设备端开发

为了实现设备影子功能，IoT 设备端需要做两件事情:

- 订阅设备影子更新的 Topic：/shadow/get/${productKey}/${deviceName} 以便实时获取云端控制指令消息;
- 设备 CONNECT 成功后，主动发布 Topic 和 Payload 查询设备影子，用于获取云端最新影子数据。

| | |
|---|---|
| **Topic:** | /shadow/update/${productKey}/${deviceName} |
| **Payload:** | {"method": "get"} |

Node.js 示例代码如下:

```
const mqtt = require('aliyun-iot-mqtt');

//设备身份三元组+区域
const deviceConfig = {
"productKey": "产品",
"deviceName": "设备",
"deviceSecret": "设备 deviceSecret",
"regionId": "cn-shanghai"
};

//1.建立连接
const client = mqtt.getAliyunIotMqttClient(deviceConfig);
//2.订阅设备影子 topic
const getShadow = `/shadow/get/${deviceConfig.productKey}/${deviceConfig.deviceName}`;
client.subscribe(getShadow)

client.on('message', function(topic, message) {
//收到消息后，显示设备影子中的远程配置参数
if (topic == getShadow) {
message = JSON.parse(message);
console.log(new Date().Format("yyyy-MM-dd HH:mm:ss.S"))
console.log("\tappConfig.content :", JSON.stringify(message.payload.state.desired.appConfig))
console.log("\tappConfig.timestamp :", JSON.stringify(message.payload.metadata.desired.appConfig.timestamp))
}

})
//3.主动获取设备影子中的远程配置参数
const updateShadow = `/shadow/update/${deviceConfig.productKey}/${deviceConfig.deviceName}`;
client.publish(updateShadow, JSON.stringify({method: "get"}), { qos: 1 })
```

## 2. 服务端调用

服务端调用设备影子接口 UpdateDeviceShadow 把新的配置参数保存到设备影子的 desired 中，接口文档:https://help.aliyun.com/document_detail/69954.html

**请求参数**

| 名称 | 类型 | 是否必需 | 描述 |
| --- | --- | --- | --- |
| Action | String | 是 | 要执行的操作，取值：UpdateDeviceShadow。 |
| ProductKey | String | 是 | 要修改影子信息的设备所隶属的产品Key。 |
| DeviceName | String | 是 | 要修改影子信息的设备名称。 |
| ShadowMessage | String | 是 | 修改后的设备影子信息。示例如下：<br>{<br>"method": "update",<br>"state": {<br>"desired": {<br>"color": "green"<br>}<br>},<br>"version": 2<br>}<br>详情参见 ShadowMessage。 |
| 公共请求参数 | - | 是 | 请参见公共参数。 |



Node.js 示例代码如下:

```
const co = require('co');
const RPCClient = require('@alicloud/pop-core').RPCClient;

const options = {
accessKey: "你的 accessKey",
accessKeySecret: "你的 accessKeySecret",};

//1.初始化 client
const client = new RPCClient({
accessKeyId: options.accessKey,
secretAccessKey: options.accessKeySecret,
endpoint: 'https://iot.cn-shanghai.aliyuncs.com',
apiVersion: '2018-01-20'
});
```

```
//2.desired 中 appConfig 变更
const shadowMessage = {
method: "update",
state: {
desired: {
appConfig:{
maxTemperature: 39.5,
}
 }
   },
   version: Date.now()
  }
  const params = {
  ProductKey: "你的 ProductKey",
   DeviceName: "你的 DeviceName",
  ShadowMessage: JSON.stringify(shadowMessage)
  };

  co(function*() {
  try {
  //3.发起 API 调用，更新影子中配置参数
   const response = yield client.request('UpdateDeviceShadow', params);
   console.log(JSON.stringify(response));
  } catch (err) {
  console.log(err);
   }
  });
```

### 3. 设备影子运行

云端业务系统调用成功后，我们在 IoT 物联网平台的控制台，**设备详情>设备影子**，查看设备影子信息。具体如下：


**在线设备，实时获取更新。**

设备在线时，设备通过订阅设备影子的 Topic 实时获得云端配置参数。


**离线设备，上线后获取更新。**

设备离线时，设备影子缓存云端配置参数，设备上线后，主动从云端拉取最新的配置参数。这时配置参数更新的时间会比当前时间早，设备端可以根据这个时间来判断是否要使用新的配置参数。

# 自定义 Topic 同步调用 RRPC 实战(二)

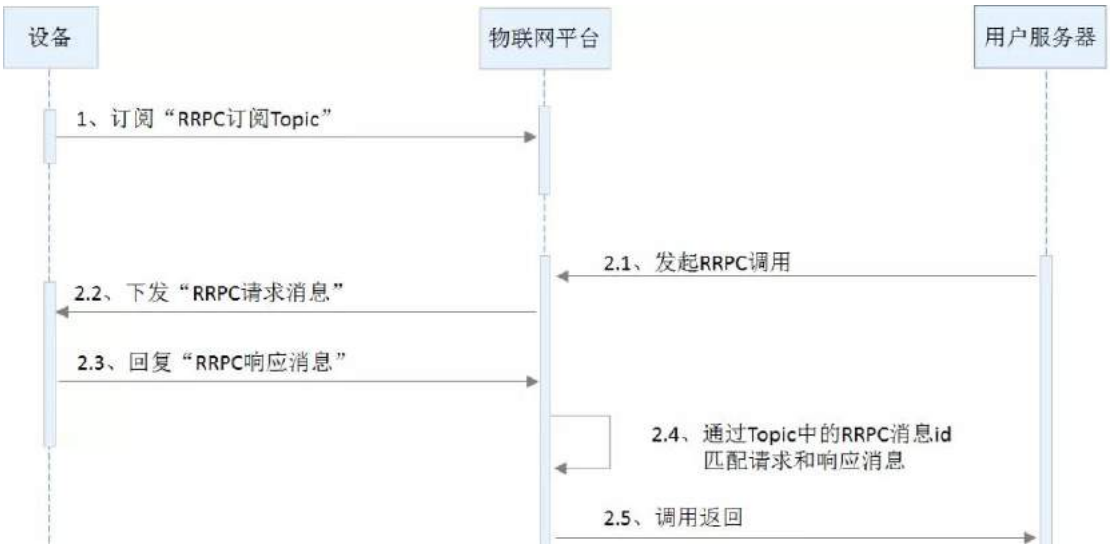
作者 | 孙承旭


RRPC：Revert-RPC。RPC（Remote Procedure Call）采用客户机/服务器模式，用户不需要了解底层技术协议，即可远程请求服务。RRPC 则可以实现由**服务端请求设备端并能够使设备端响应的功能**。

为了适应智能灯**开灯**，智能锁**开锁**，充电宝**弹出**，自动售货机**付款后出货**，按摩椅**启动**等业务场景，应用服务器通过 POP API 发起 RRPC 调用，IoT 设备端只需要在 Time out 内，按照固定的格式回复 Pub 消息，**服务端即可同步获取 IoT 设备端的响应结果**。

## 一、技术原理

服务端同步 RRPC 调用业务流程如下：



自定义Topic的RRPC调用的Topic格式如下：

- RRPC请求消息Topic：**/ext/rrpc/${messageId}/this/is/my/topic**
- RRPC响应消息Topic：**/ext/rrpc/${messageId}/this/is/my/topic**

其中${messageId}是IoT物联网平台生成的唯一的RRPC消息id，
**黑体**部分是IoT物联网平台约定，**红色**部分可以根据业务场景自定义。

## 二、设备端开发

我们以充电桩场景为例，用户完成付款后，服务端推送充电指令，并实时获取设备处理结果。指令示例如下：

```
Topic: /ext/rrpc/1234252323/charging/cmd
Payload: {"power": 200,"port":"3"}
```

设备响应示例：

```
Topic: /ext/rrpc/1234252323/charging/cmd
Payload: {"bizCode": 0,"errMsg":"xxxxx"}
```

创建充电桩产品，并注册设备。

为了配合自定义 Topic 的 RRPC 调用，设备端需要在 MQTT 的 **CONNECT 参数 clientId 中增加 ext=1** 标识：


设备端监听 RRPC 指令 Topic，开启指定把枪充电，并返回响应结果。示例代码如下：

```
const mqtt = require('aliyun-iot-mqtt');
// 1.设备身份三元组
const options = {
 productKey: "Your productKey",
deviceName: "Your deviceName",
deviceSecret: "Your deviceSecret",
regionId:"cn-shanghai"
};
// 2.建立连接
const client = mqtt.getAliyunIotMqttClient(options)
;// 3.订阅 RRPC 主题
client.subscribe(`/ext/rrpc/+`)

client.on('message', function(topic, message) {

console.log("topic <<=" + topic)
console.log("payload <<=" + message)
if(topic.indexOf(`/ext/rrpc/`)>-1){
// 接收并处理业务系统 RRPC 指令
 handleRrpc(topic, message)
 }
})

function handleRrpc(topic, message){
//响应 RRPC 指令 payload 自定义
 const payloadJson = {bizCode:0};// 0 成功，400 充电失败
console.log()
console.log("reply topic =>>" + topic)
console.log("reply payload =>>" + JSON.stringify(payloadJson))
client.publish(topic, JSON.stringify(payloadJson));
}
```

## 三、服务端开发

服务端通过 RRPC API 即可发起同步调用，实时获取设备端响应结果。

RRPC API 文档地址：https://help.aliyun.com/document_detail/69797.html

**请求参数**

| 名称 | 类型 | 是否必需 | 描述 |
| --- | --- | --- | --- |
| Action | String | 是 | 要执行的操作，取值：RRpc。 |
| ProductKey | String | 是 | 要发送消息的产品Key。 |
| DeviceName | String | 是 | 要接收消息的设备名称。 |
| RequestBase64Byte | String | 是 | 要发送的请求消息内容经过Base64编码得到的字符串格式数据。 |
| Timeout | Integer | 是 | 等待设备回复消息的时间，单位是毫秒，取值范围是1,000 ~5,000。 |
| Topic | String | 否 | 使用自定义的RRPC相关Topic。即上文介绍的红色部分 |
| 公共请求参数 | - | 是 | 请参见公共参数。 |

**返回数据**

| 名称 | 类型 | 示例值 | 描述 |
| --- | --- | --- | --- |
| Code | String | iot.system.SystemException | 调用失败时，返回的错误码。错误码详情，请参见错误码。 |
| ErrorMessage | String | 系统异常 | 调用失败时，返回的出错信息。 |
| MessageId | Long | 88945594212434739 2 | 成功发送请求消息后，云端生成的消息ID，用于标识该消息。 |
| PayloadBase64Byte | String | d29ybGQgaGVsbG8= | 设备返回结果Base64编码后的值。 |
| RequestId | String | 41C4265E-F05D-4E2E-AB09-E031F501AF7F | 阿里云为该请求生成的唯一标识符。 |
| RrpcCode | String | SUCCESS | 调用成功时，生成的调用返回码，标识请求状态。取值：<br>• UNKNOWN：系统异常<br>• SUCCESS：成功<br>• TIMEOUT：设备响应超时<br>• OFFLINE：设备离线<br>• HALFCONN：设备离线（设备连接断开，但是断开时间未超过一个心跳周期） |
| Success | Boolean | true | 是否调用成功。true表示调用成功，false表示调用失败。 |

我们以 Node.js 发起同步调用，代码示例：

```
const co = require('co');
const RPCClient = require('@alicloud/pop-core').RPCClient;

const options = {
 accessKey: "your accessKey",
accessKeySecret: "your accessKeySecret"
};

//1.初始化 client
const client = new RPCClient({
accessKeyId: options.accessKey,
secretAccessKey: options.accessKeySecret,
endpoint: 'https://iot.cn-shanghai.aliyuncs.com',
apiVersion: '2018-01-20',
opts: {
  timeout: 9000
}
});
// 指令内容
const payload = {
power: 200,
port: 3
};

//2.构建 RRPC 请求
const params = {
ProductKey: "",
DeviceName: "",
RequestBase64Byte: new Buffer(JSON.stringify(payload)).toString("base64"),
Timeout: 8000,
Topic:"/charging/cmd
"};

co(function*() {
 //3.发起 API 调用
try {
const response = yield client.request('Rrpc', params);
```

```
 console.log(JSON.stringify(response));
  console.log(response.RrpcCode);

 if (response.RrpcCode == "SUCCESS") {
var resultJSON = new Buffer(response.PayloadBase64Byte, 'base64').toString();
console.log("RRPC SUCCESS =====>", JSON.stringify(JSON.parse(resultJSON)));
 }
} catch (err) {
console.log("RRPC ERROR =====>", JSON.stringify(err.data));
 }
});
```

## 四、联机调试

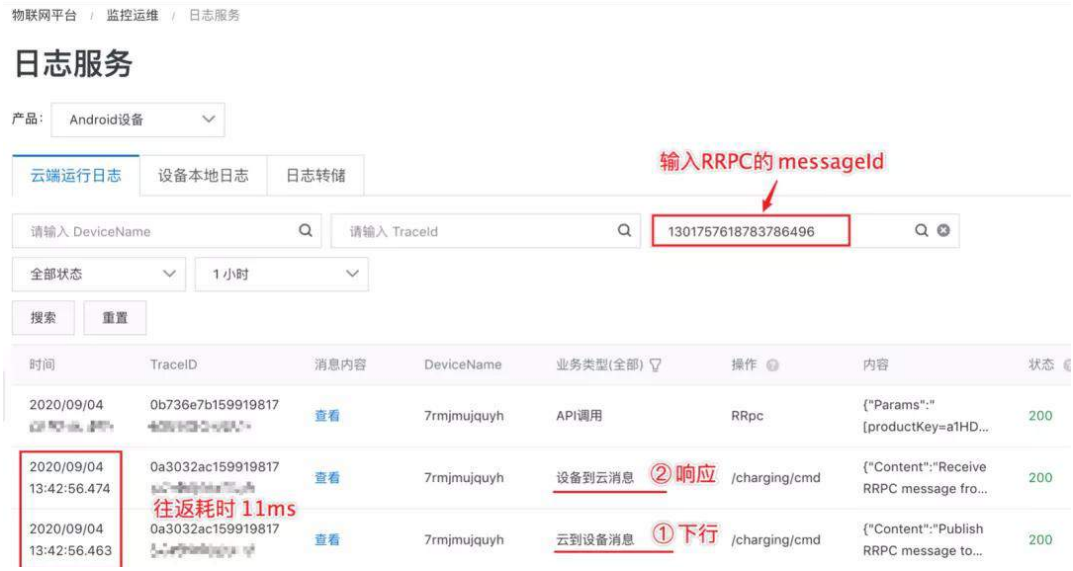
正常响应结果：

```
wxm-mbp:iot wxm$ node StartChargingCmdRRPC2.js
{"RequestId":"847E0EDC-30E6-42A6-B7DA-D37FE570E5FE","PayloadBas
e64Byte":"eyJiaXpDb2RlIjowfQ==","RrpcCode":"SUCCESS","Success":
true,"MessageId":"13017576187837864 96"}
SUCCESS
RRPC SUCCESS ======> {"bizCode":0}
```

设备端日志：

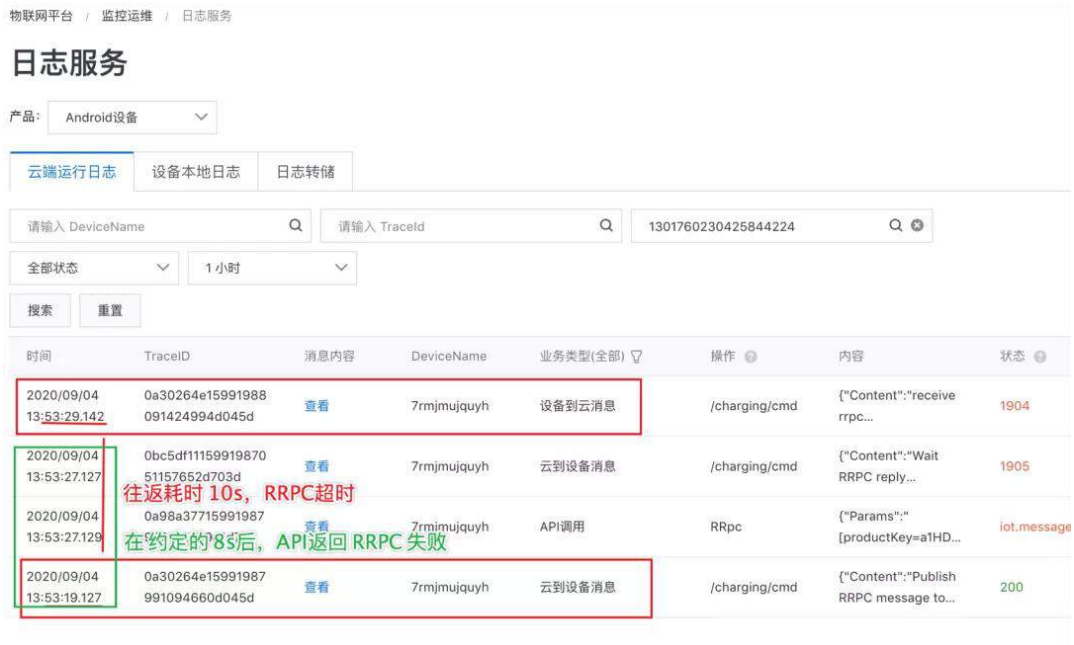
```
[redacted]$ node ChargingStationRRPC2.js

topic <<=/ext/rrpc/13017576187837864 96/charging/cmd
payload <<={"power":200,"port":3}

reply topic =>>/ext/rrpc/13017576187837864 96/charging/cmd
reply payload =>>{"bizCode":0}
```

控制台日志：



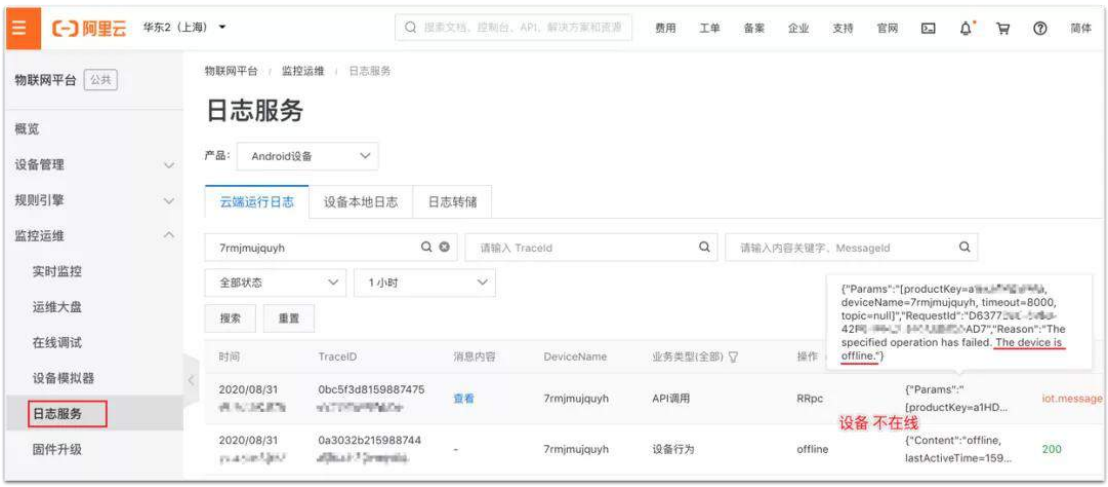

异常响应结果，设备超时：


控制台日志：



异常响应结果，设备离线：


控制台日志：




【IoT 平台 RRPC 实战】

基于系统 Topic 实现 RRPC 同步调用(1)

# 系统 Topic 实现云端同步调用 RRPC(一)

作者 | 孙承旭


MQTT 协议是基于 Pub/Sub 的异步通信模式，无法实现 HTTP 协议的同步响应业务处理结果，导致云端业务系统开发难度高。

为了适应智能灯**开灯**，智能锁**开锁**，充电宝**弹出**，自动售货机**付款后出货**，按摩椅**启动**等业务场景，IoT 物联网平台基于 MQTT 协议制定了一套请求和响应的同步机制，无需改动 MQTT 协议即可实现同步通信。应用服务器通过 POP API 发起 RRPC 调用，IoT 设备端只需要在 Timeout 内，按照固定的格式回复 Pub 消息，**服务端即可同步获取 IoT 设备端的响应结果**。

## 一、技术原理

服务端同步 RRPC 调用业务流程如下：

Topic 格式约定：

**请求：/sys/${productKey}/${deviceName}/rrpc/request/${messageId}**
**响应：/sys/${productKey}/${deviceName}/rrpc/response/${messageId}**

> $表示变量，每个设备不同
> messageId为IoT平台生成的消息ID，
> 设备端回复responseTopic里的messageId要与requestTopic一致

**示例：**

**设备端需要订阅：**

> /sys/${productKey}/${deviceName}/rrpc/request/+

运行中设备收到Topic：

> /sys/PK100101/DN213452/rrpc/request/443859344534

收到消息后，在timeout时间内回复Topic：

> /sys/PK100101/DN213452/rrpc/response/443859344534

## 二、设备端开发

我们以充电桩场景为例，用户完成付款后，服务端推送充电指令，并实时获取设备处理结果。指令如下：

```
{"power":200,"port":3}
```

设备响应：

```
{"bizCode": 0,"errMsg":"xxxxx"};// 0 成功，400 充电失败
```

创建充电桩产品，并注册设备。


设备端监听 RRPC 指令 Topic，开启指定把枪充电，并返回响应结果。示例代码如下：

```
const mqtt = require('aliyun-iot-mqtt');
// 1.设备身份三元组
const options = {
productKey: "Your productKey",
deviceName: "Your deviceName",
deviceSecret: "Your deviceSecret",
regionId:"cn-shanghai"
};
// 2.建立连接
const client = mqtt.getAliyunIotMqttClient(options);
// 3.订阅 RRPC 主题
client.subscribe(`/sys/${options.productKey}/${options.deviceName}/rrpc/request/+`)

client.on('message', function(topic, message) {
if(topic.indexOf(`/sys/${options.productKey}/${options.deviceName}/rrpc/request/`)>-1){

// 4.接收并处理业务，响应 RRPC 指令
handleRrpc(topic, message)
 }
})

function handleRrpc(topic, message){
topic = topic.replace('/request/','/response/');
console.log("topic=" + topic)
```

```
console.log("payload=" + message)
//响应 RRPC 指令 payload 自定义
const payloadJson = {bizCode:0};// 0 成功，400 充电失败
client.publish(topic, JSON.stringify(payloadJson));
}
```

## 三、服务端开发

服务端通过 RRPC API 即可发起同步调用，实时获取设备端响应结果。

RRPC API 文档地址：https://help.aliyun.com/document_detail/69797.html

**请求参数**

| 名称 | 类型 | 是否必选 | 示例值 | 描述 |
| --- | --- | --- | --- | --- |
| Action | String | 是 | RRpc | 系统规定参数。取值：RRpc。 |
| DeviceName | String | 是 | device1 | 要接收消息的设备名称。 |
| ProductKey | String | 是 | aldfeSe**** | 要发送消息的产品Key。 |
| RequestBase64Byte | String | 是 | dGhpcyBpcyBhbiBleGFtcGxl | 要发送的消息内容经过Base64编码得到的字符串格式数据，例如 `dGhpcyBpcyBhbiBleGFtcGxl` 。 |
| Timeout | Integer | 是 | 1000 | 等待设备回复消息的时间，单位是毫秒，取值范围是1,000 ~8,000。 |
| IotInstanceId | String | 否 | iot_instc_pu****_c*-v64******** | 共享实例不传此参数；您购买的实例需传入实例ID。 |
| Topic | String | 否 | /a1uZfYb****/A_Vol****/user/update | 使用自定义的RRPC相关Topic。需要设备端配合使用，请参见设备端开发自定义Topic。不传入此参数，则使用系统默认的RRPC Topic。 |

## 返回数据

| 名称 | 类型 | 示例值 | 描述 |
| --- | --- | --- | --- |
| Code | String | iot.system.SystemException | 调用失败时，返回的错误码。错误码详情，请参见错误码。 |
| ErrorMessage | String | 系统异常 | 调用失败时，返回的出错信息。 |
| MessageId | Long | 889455942124347392 | 成功发送请求消息后，云端生成的消息ID，用于标识该消息。 |
| PayloadBase64Byte | String | d29ybGQgaGVsbG8= | 设备返回结果Base64编码后的值。 |
| RequestId | String | 41C4265E-F05D-4E2E-AB09-E031F501AF7F | 阿里云为该请求生成的唯一标识符。 |
| RrpcCode | String | SUCCESS | 调用成功时，生成的调用返回码，标识请求状态。取值：<br>• **UNKNOWN**：系统异常<br>• **SUCCESS**：成功<br>• **TIMEOUT**：设备响应超时<br>• **OFFLINE**：设备离线<br>• **HALFCONN**：设备离线（设备连接断开，但是断开时间未超过一个心跳周期） |
| Success | Boolean | true | 是否调用成功。**true**表示调用成功，**false**表示调用失败。 |

我们以 Node.js 发起同步调用，代码示例：

```
const co = require('co');
const RPCClient = require('@alicloud/pop-core').RPCClient;

const options = {
accessKey: "Your accessKey",
accessKeySecret: "Your accessKeySecret"
};

// 1.初始化 client
const client = new RPCClient({
accessKeyId: options.accessKey,
secretAccessKey: options.accessKeySecret,
endpoint: 'https://iot.cn-shanghai.aliyuncs.com',
apiVersion: '2018-01-20',
```

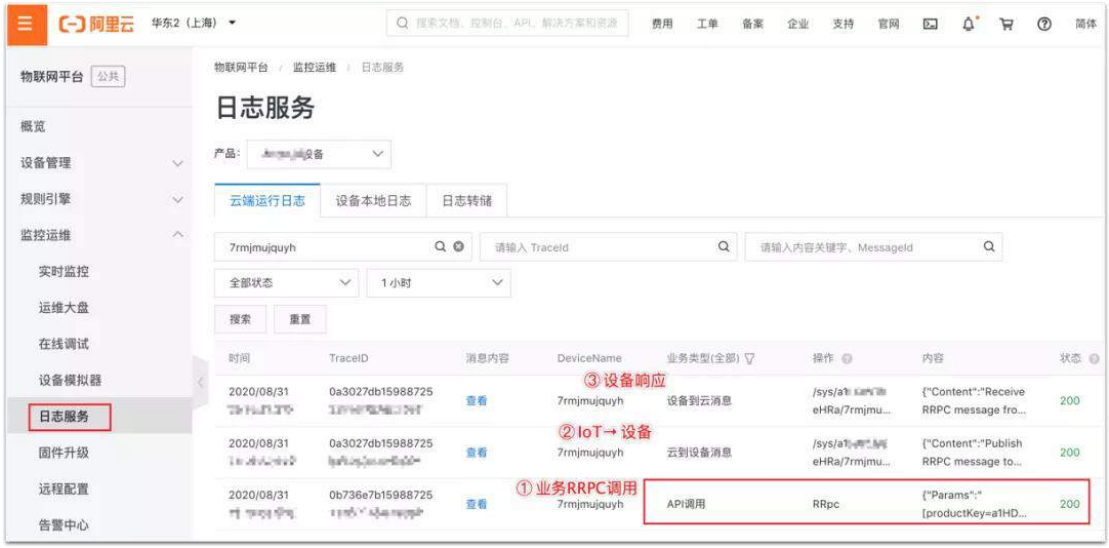
```
opts: {
timeout: 9000
}
});
// 指令内容
const payload = {
power: 200,
 port: 3};

// 2.构建 RRPC 请求
const params = {
ProductKey: "Your ProductKey",
DeviceName: "Your DeviceName",
RequestBase64Byte: new Buffer(JSON.stringify(payload)).toString("base64"),
Timeout: 8000
};

co(function*() {
// 3.发起 API 调用
try {
const response = yield client.request('Rrpc', params);

console.log(JSON.stringify(response));

 console.log(response.RrpcCode);

if (response.RrpcCode == "SUCCESS") {
 var resultJSON = new Buffer(response.PayloadBase64Byte, 'base64').toString();
    console.log("RRPC SUCCESS =====>", JSON.stringify(JSON.parse(resultJSON)));
    }
} catch (err) {
 console.log("RRPC ERROR =====>", JSON.stringify(err.data));
    }
});
```

## 四、联机调试

正常响应结果：

```
$ node StartChargingCmd.js
{"RequestId":"70E                                    9C50","Payload
Base64Byte":"eyJiaXpDb2RlIjowfQ==","RrpcCode":"SUCCESS","Suc
cess":true,"MessageId":"130                          33"}
SUCCESS
RRPC SUCCESS ======> {"bizCode":0}
```

控制台日志：



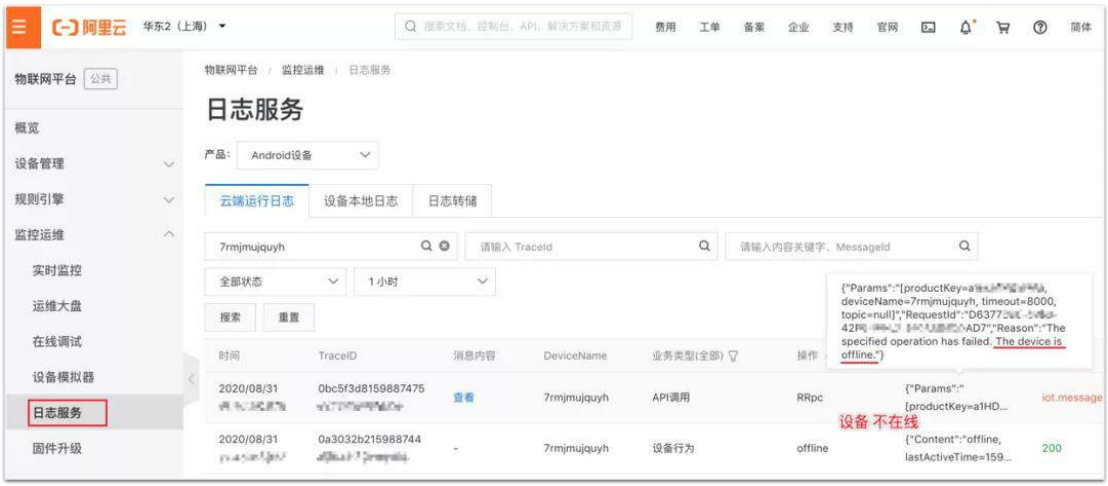

异常响应结果，设备超时：

```
$ node StartChargingCmd.js
RRPC ERROR ======> {"RequestId":"01                                  FA
303B78A1","RrpcCode":"TIMEOUT","ErrorMessage":"The specified
 operation has failed. The device has timed out.","Code":"io
t.messagebroker.TIMEOUT","Success":false,"MessageId":"130
                                                      8"}
```

控制台日志：

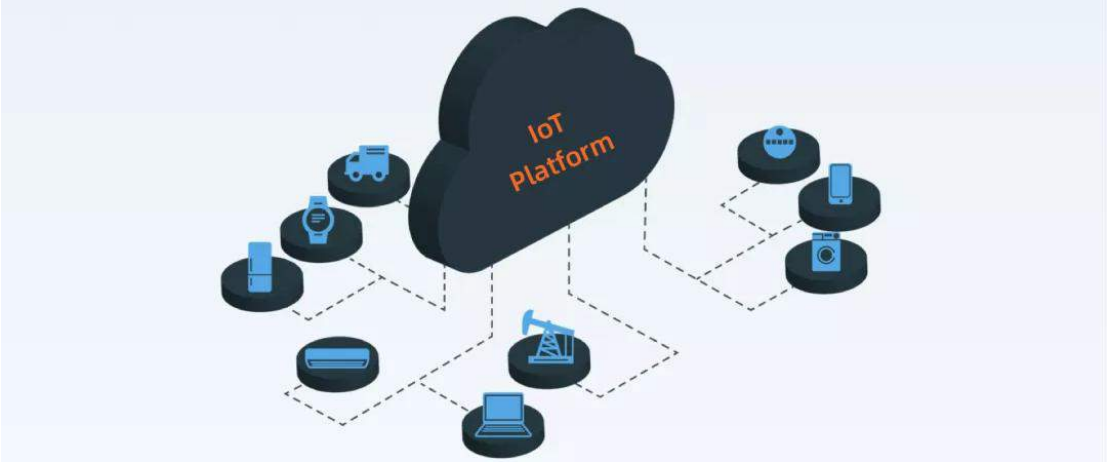

异常响应结果，设备离线：


控制台日志：



# 设备上报二进制数据云端解析

作者 | 苏堤嘉木




在 IoT 场景中，很多传感器采集到的是私有协议二进制数据流，设备端又不具备转换成结构化 JSON 的能力，这时设备可以通过**自定义 Topic 上报二进制数据**，在 IoT 物联网平台支持云端配置解析脚本，**动态转换成结构化的 JSON 数据**。

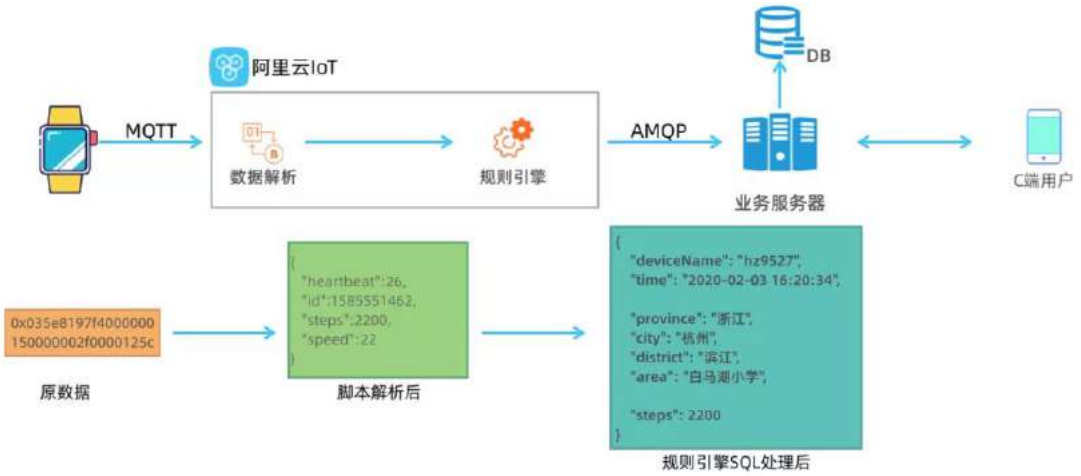
完整端到端开发过程如下：

1. 明确二进制上报 Topic 和数据规则。
2. 云端预先配置针对指定 Topic 的原始数据配置 JS 解析脚本。
3. 在脚本解析模拟数据输入，校验脚本业务逻辑正确后，提交到 IoT 云端。
4. 运行设备，指定 Topic 上报原始数据。
5. 云端日志服务查看数据解析过程。

我们以手表为例，传感器上报 hex 进制数据，到 IoT 物联网平台，在云端解析，最终以结构化 JSON 流转到业务系统。

## 一、技术架构

设备端二进制数据在云端 IoT 平台转换链路，如下图：

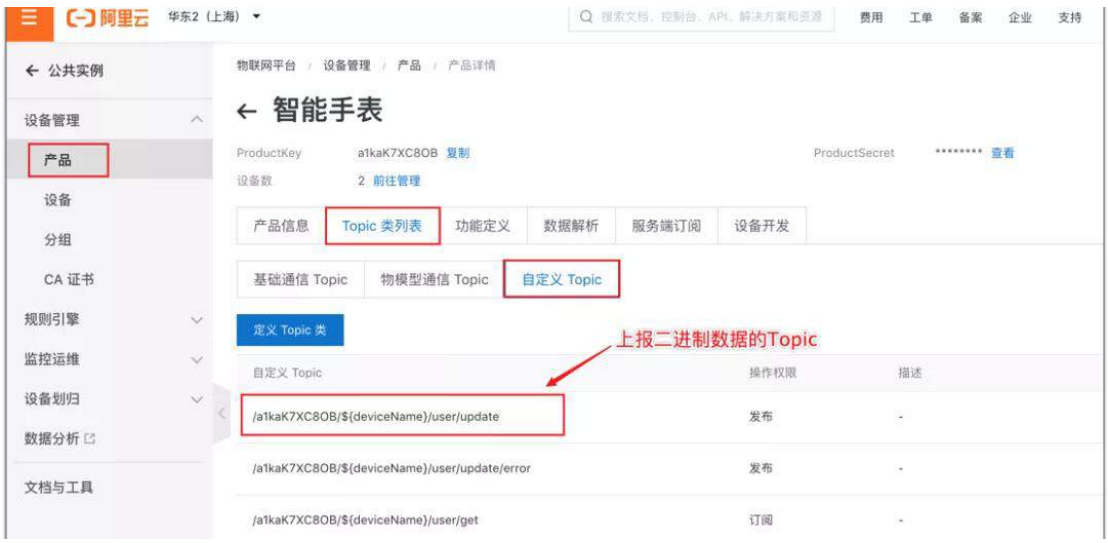



消息转换前后变化：

| | 设备-上报原始消息 | 云上-脚本解析后 |
|---|---|---|
| Topic | /a1YcQ8bPGbE/dn308/user/update?sn_default | /a1YcQ8bPGbE/dn308/user/update |
| Payload | 0x035e8197f40000001500000002f0000125c | {<br>"heartbeat":26,<br>"id":1585551462,<br>"steps":2200,<br>"speed":22<br>} |

## 二、IoT 物联网平台 云端开发

创建产品和消息通信 Topic 选择：

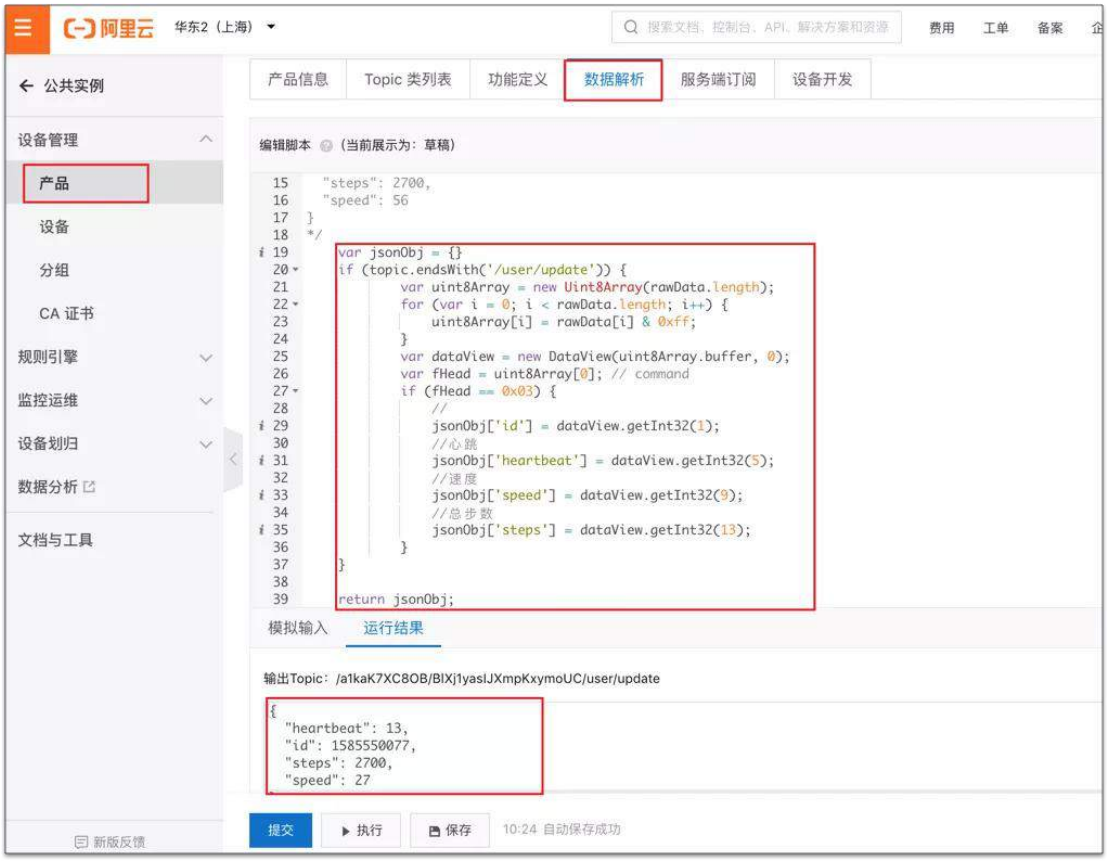


**原始数据：0x035e8192fd0000000d0000001b00000a8c**

**数据业务格式：**

| hex进制字节 | 业务 | 示例 |
| --- | --- | --- |
| 1 | 上报标识位 | 03 |
| 2-5 | 消息业务id | 5e8192fd |
| 6-9 | 心跳次数 | 0000000d |
| 10-13 | 移动速度 | 0000001b |
| 14-17 | 当天累计总步数 | 00000a8c |

在控制台**产品详情>数据解析** 配置并提交脚本：



完整脚本内容如下：

```
function transformPayload(topic, rawData) {
/*
原始 hex 数据 ：0x035e8192fd0000000d0000001b00000a8c
转换后 JSON 数据 ：
 {
"heartbeat": 15,
"id": 1585549855,
"steps": 2700,
"speed": 56
}
*/
var jsonObj = {}
if (topic.endsWith('/user/update')) {
var uint8Array = new Uint8Array(rawData.length);
for (var i = 0; i < rawData.length; i++) {
uint8Array[i] = rawData[i] & 0xff;
}
var dataView = new DataView(uint8Array.buffer, 0);
var fHead = uint8Array[0];
if (fHead == 0x03) {// command
//
 jsonObj['id'] = dataView.getInt32(1);
//心跳
 jsonObj['heartbeat'] = dataView.getInt32(5);
//速度
jsonObj['speed'] = dataView.getInt32(9);
//总步数
jsonObj['steps'] = dataView.getInt32(13);
    }
 }
return jsonObj;
}
```

在产品下注册设备，并获取**设备身份三元组**，如下：

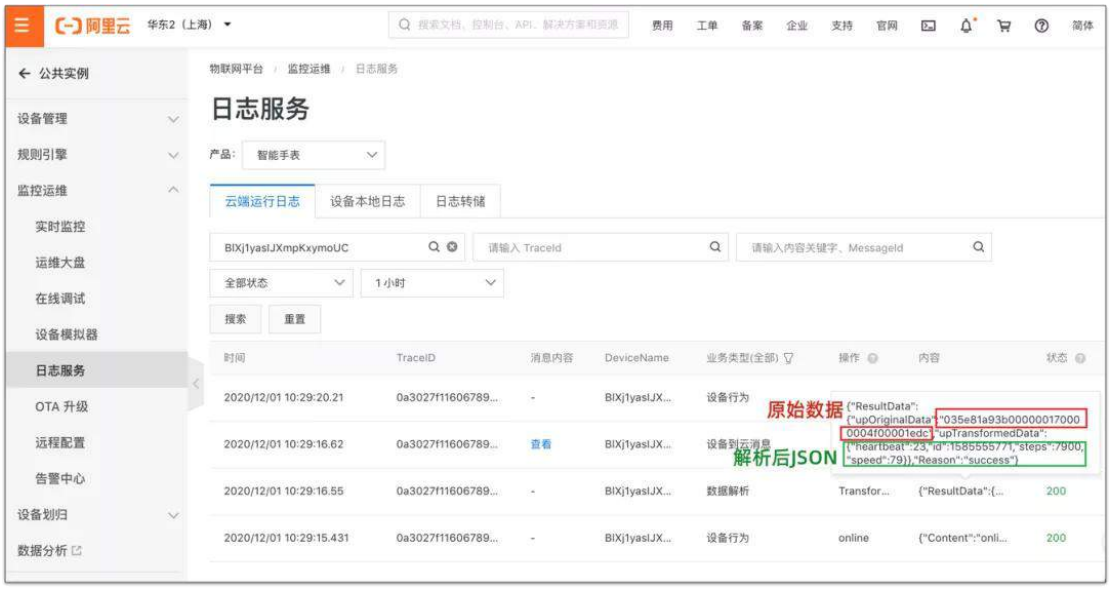

## 三、设备端开发

我们通过 Node.js 程序模拟设备端上报二进制数据：

```
const mqtt = require('aliyun-iot-mqtt');

//设备身份
const options = {
productKey: "a1kaK7XC8OB",
deviceName: "BlXj1yasIJXmpKxymoUC",
deviceSecret: "41798535d799760c8f67f02efd28b01c",
regionId: "cn-shanghai"};

//建立连接
const client = mqtt.getAliyunIotMqttClient(options);

// 消息 Topic 携带?_sn=default 标识
const topic = `/${options.productKey}/${options.deviceName}/user/update?_sn=default`;
// 原始数据
var payloadArray = [ 3, 94, 129, 169, 59, 0, 0, 0, 23, 0, 0, 0, 79, 0, 0, 30, 220 ];
var payload = new Buffer(payloadArray);

// 发布数据到 topic
client.publish(topic, payload);
```

# 四、联机运行

设备运行后，我们可以在 IoT 控制台的日志服务里查看到完整的数据处理过程，包括设备上报的原始数据，以及脚本解析处理后的结构化 JSON 数据，如下图：




钉钉扫一扫加入
AIoT 物联网平台群

阿里云开发者“藏经阁”
海量免费电子书下载