

AIoT物联网 开发实战(下)

6大场景案例详解，速懂规则引擎、监控运维等操作





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

规则引擎-数据流转	5
阿里云 IoT 规则引擎 SQL 参考	6
两张图秒懂 IoT 设备数据云端流转	8
IoT 设备数据转存到 DB 表格存储	10
实时监听设备在线/离线事件	16
IoT 设备运行状态缓存到 Redis	20
IoT 设备之间 M2M 场景联动实战	28
App 和 IoT 设备数据实时同步和控制	38
监控运维	44
IoT 物联网平台 3 种计费方式	45
IoT 平台消息轨迹全景图	49
IoT 平台性能压测工具 JMeter	53
IoT 设备 OTA 固件升级实践	60
用 Wireshark 抓包分析 IoT 设备网络行为	69
IoT 平台日志服务详解	76
IoT 设备日志采集利器服务	80
全链路开发实战	84
IoT 平台端到端开发实践	85
7 分钟视频讲解全链路开发	102
自建 MQTT 迁移阿里云 IoT	103
TCP 协议 IoT 设备迁移上云	122
场景开发实践	131
AI 人脸识别，监控老板行踪！	132
IoT+DB + DataV 搭建实时环境监控大屏	147
IoT+TSDB+Quick BI 环境监控	156
20 元自制 Arduino 环境监测仪	165
IoT 二维码收款播报音箱	177
IoT 智能手持测温枪开发	179

规则引擎-数据流转

阿里云 IoT 规则引擎 SQL 参考

作者 | 苏堤嘉木

一、阿里云 IoT SQL 语法

在使用规则引擎时，处理数据逻辑通过一种类似 SQL 的语法来定义。

SQL 语句结构如下：

- SELECT
必需。可以使用上报消息的 payload,也可以使用阿里云 IoT 平台内置的函数
- FROM
必需。用于匹配需要处理的消息 Topic
- WHERE
可选。规则触发条件，条件表达式

二、.SQL 处理示例

1. SQL 语句的示例如下：

```
SELECT
  deviceName() as deviceName,
  attribute('site') as site,
  pm25,
  temperature FROM
  '/pk/dn/data'
WHERE
  pm25 > 60
```

2. 设备在 IoT 平台注册的信息

```
productKey: pk
```

```
deviceName: dn
标签:
  imei = XIXI2018034532
  site = 西溪湿地-洪园
```

3. 设备 mqtt 消息的示例:

```
topic: '/pk/dn/data'
payload: {
  "pm25": 63,
  "temperature": 31
}
```

4. 处理结果

当设备消息在 '/pk/dn/data' 主题上发布, 则触发规则引擎数据处理 SQL 语句。如果 "pm25" 属性大于 60, SQL 语句将提取 select 字段的值。

处理结果如下所示:

```
{
  "deviceName": "dn",
  "site": "西溪湿地-洪园",
  "pm25": 63,
  "temperature": 31
}
```

三、参考文档

1. 阿里云 IoT SQL 函数列表

https://help.aliyun.com/document_detail/30555.html

2. 物模型数据格式

https://help.aliyun.com/document_detail/73736.html

两张图秒懂 IoT 设备数据云端流转

作者 | 苏堤嘉木

当设备接入阿里云 IoT 物联网平台，基于 Topic 进行通信时，我们可以使用规则引擎，编写 SQL 对 Topic 中的数据进行处理，并配置转发规则将处理后的数据转发到阿里云其他服务。

- 可以转发到 **RDS**、**表格存储**、**HiTSDB** 中进行存储。
- 可以转发到 **DataHub** 中，使用 Streamcompute 进行流计算，使用 Maxcompute 进行大规模离线计算。
- 可以转发到**函数计算**进行事件计算。
- 可以转发到另一个 Topic 实现 **M2M** 通信。
- 可以转发到队列 **MQ** 实现可靠消费数据。
- 可以转发到**消息服务 MNS** 实现消费数据。

使用规则引擎时，我们需要基于 Topic 编写 SQL 处理数据。

- 基础版**自定义 Topic** 和高级版自定义的 Topic，数据格式是自定义的，物联网平台不做处理，直接流转到规则引擎处理。
- 高级版系统默认的**物模型 Topic**，设备 payload 会在云端转换成物模型 payload 后，再流转到规则引擎处理。

基础版产品数据流转过程：



高级版产品数据流转过程：



IoT 设备数据转存到 DB 表格存储

作者 | 苏堤嘉木



一、IoT 云端开发

开通物联网套件：<https://www.aliyun.com/product/iot>

1. 创建产品

创建基础版产品，添加消息通信 Topic:



设备上报数据结构:

```
topic: "/a8*****i3B/d43*****nK/temperatureData"
payload: {
  "temperature": 23,
  "humidity": 63
}
```

2. 添加设备

添加设备获取三元组，添加标签信息:

The screenshot shows the '设备详情' (Device Details) page in the IoT Platform console. The left sidebar contains navigation links: 物联网平台, 产品管理, 设备管理, 边缘管理, 规则引擎, 扩展服务, 我的服务, and 产品文档. The main content area shows the device 'temperature' with status '未激活'. It displays the product '环境监测', ProductKey, DeviceName, and DeviceSecret. Below this is a table of device information.

设备信息	
产品名称	环境监测
节点类型	设备
当前状态	未激活
添加时间	2018/07/11 15:53:19

Below the table, there is a '标签信息' (Tag Information) section. It shows the device tag 'imei: lm329en9k' and a tag '客厅' (Living Room) with an '立即添加' (Add Immediately) button.

二、表格存储

开通表格存储服务：<https://www.aliyun.com/product/ots>

1. 创建用于存储设备数据的表

The screenshot shows the 'temp' data table configuration page. On the left is a sidebar with navigation options: 基本详情, 数据管理, 触发器管理, and 数据监控. The main area is titled '基本信息' and contains the following details:

- 数据表名称: temper
- 数据生命周期: -1
- 最大数据版本: 1
- 数据有效版本偏差: 86400
- 最近一次调整时间: 2018-07-25 11:49:46
- 表格大小: 0 B
- 主键: deviceid (STRING) (分片键) time (STRING)

三、IoT 物联网平台-规则引擎

The screenshot shows the '规则引擎' (Rule Engine) page. On the left is a sidebar with navigation options: 物联网平台, 产品管理, 设备管理, 边缘管理, 规则引擎, 扩展服务, 我的服务, and 产品文档. The main area is titled '规则引擎' and contains a '规则列表' (Rule List) table.

规则名称	数据格式	规则描述	创建时间	状态
数据流转OTS	JSON		2018/07/11 11:37:42	运行中

At the bottom right, there is a pagination bar showing '共有 2 条' (Total 2 items), '< 上一页' (Previous page), '1' (Current page), and '下一页 >' (Next page).

1. 处理数据 SQL

```
SELECT
deviceName() as deviceName,
timestamp('yyyy-MM-dd HH:mm:ss') as time,
attribute('tag') as tag,
attribute('imei') as imei,
humidity,
temperature
FROM
"/a8*****i3B/+/temperatureData"
```

*** 规则查询语句:**

字段:

Topic:

+temperatureData

可以使用规则引擎函数,例如:deviceName()=mydevice

2. 转发数据

存储到表格存储(Table Store)

* 地域:

华东 2

* 实例：

10¹⁰ molecules

* 数据表:

temperatura

* 主键:

deviceId *值: \${deviceName}

* 主键:

time * 值:	\$(time)
-----------	----------

* 角色:

AliyunIoTAccessingOTSRole

3. 完整规则引擎



4. 启动规则引擎



四、设备端开发

模拟设备的 nodejs 脚本：device-2-iot-2-ots.js

```

/**
 * npm依赖: "aliyun-iot-mqtt": "0.0.4"
 */
const mqtt = require('aliyun-iot-mqtt');

//设备三元组
const options = {
  productKey: "替换自己productKey",
  deviceName: "替换自己deviceName",
  deviceSecret: "替换自己deviceSecret",
  regionId: "cn-shanghai"
};

//设备与云 建立连接, 设备上线
const client = mqtt.getAliyunIotMqttClient(options);

//主题topic
const topic = `${options.productKey}/${options.deviceName}/temperatureData`;
//指定topic发布payload数据到IoT云端
setInterval(function() {
  client.publish(topic, getPostData());
}, 10 * 1000);

function getPostData() {
  const data = {
    temperature: Math.floor((Math.random() * 20) + 10),
    humidity: Math.floor((Math.random() * 80) + 20),
  };
  console.log("===postData topic=" + topic, data);

  return JSON.stringify(data);
}

```

五、启动运行

启动虚拟设备脚本: `$ node device-2-iot-2-ots.js`

表格存储数据查看:

temperat

基本详情

数据管理

触发器管理

数据监控

表格数据

插入数据 查询数据 更新数据 删除数据

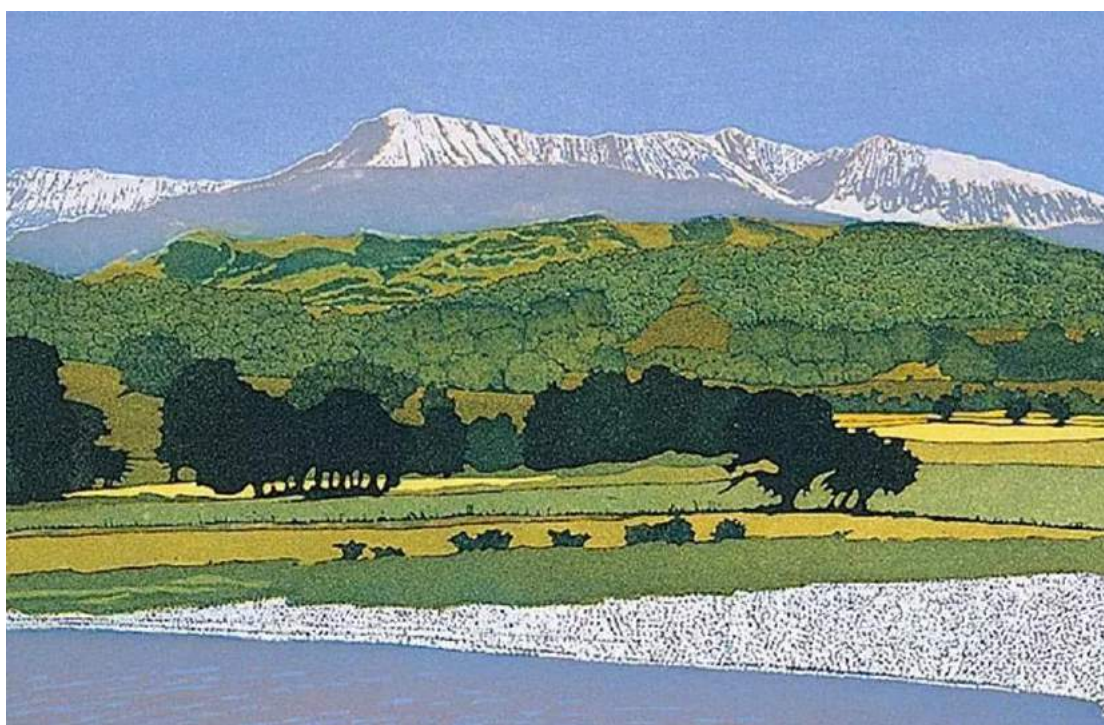
表格数据最多显示50行。

	deviceId(主键)	time(主键)	humidity	imei	tag	temperature
☐	temp	2018-07-25 13:45:16	43	lm329en9k	客厅	25
☐	tempe	2018-07-25 13:45:26	72	lm329en9k	客厅	15
☐						

共有2条, 每页显示: 10条

实时监听设备在线/离线事件

作者 | 苏堤嘉木



一、设备上下线状态消息

当设备连接到 IoT 物联网平台，设备离线，在线状态变更会生成特定 topic 的消息，我们服务端可以通过订阅这个 topic 获得设备状态变更信息。

设备的上下线状态流转的 Topic 格式：

```
/as/mqtt/status/{productKey}/{deviceName}
```

payload 数据格式：

```
{  
  "status": "online|offline",  
  "productKey": "pk13543",
```

```

"deviceName":"deviceName1234",
"time":"2018-08-31 15:32:28.205",
"utcTime":"2018-08-31T07:32:28.205Z",
"lastTime":"2018-08-31 15:32:28.195",
"utcLastTime":"2018-08-31T07:32:28.195Z",
"clientIp":"123.123.123.123"
}

```

参数说明：

参数	类型	说明
status	String	设备状态，online上线，offline离线
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
time	String	此条消息发送的时间点
utcTime	String	此条消息发送的UTC时间点
lastTime	String	状态变更前 最后一次通信时间 说明 可能是PUBLISH，PING等。
utcLastTime	String	状态变更前最后一次通信的UTC时间
clientIp	String	设备公网出口IP

二、通过规则引擎流转设备状态

1. 配置 SQL

```

SELECT
productKey,deviceName,
timestamp() as timestamp ,
status,clientIp,
time as currentTime ,lastTime

FROM
"/as/mqtt/status/a*****t/+"

```

这样我们就可以从消息体获取到设备的 status, currentTime 和 lastTime 了。

数据处理 SQL 配置界面：

规则查询语句：

复制语句

```
SELECT
productKey,deviceName,timestamp() as timestamp,status,time as currentTime ,lastTime,clientIp
FROM      "/as/mqtt/status/a[redacted]st/+"
WHERE
```

● 字段：

productKey,deviceName,timestamp() as timestamp,status,time as currentTim

● Topic：

/as/mqtt/status/a[redacted]st/+/

设备状态

模拟

全部设备(+)

● 条件 (选填)

可以使用规则引擎函数，例如：deviceName()=mydevice

2. 配置数据流转目的地

数据流转配置界面：

选择操作:

存储到表格存储 (Table Store) 该方法将数据插入到 [表格存储\(Table Store\)](#) 中, 详情请参考 [文档](#)

* 地域:

华东 2 

* 实例:

 [创建实例](#)

* 数据表:

 [创建数据表](#)

* 主键:

deviceId键

`${deviceName}`

* 主键:

time键

`${timestamp}`

* 角色:

AliyunIoTAccessingOTSRole [创建RAM角色](#)

三、运行结果

当有设备上线, 业务交互, 下线, 我们就可以在 DB 看到如下记录。

status 是设备状态, lastTime 是最后一次通信时间, currentTime 是这条消息生成时间, 可以理解为 设备上线时间, 下线时间。

详细数据	deviceId(主键)	time(主键)	deviceId	currentTime	lastTime	productKey	status
详细数据	Li...As	1559199508645	42.120.75.128	2019-05-30 14:58:28...	2019-05-30 14:58:28...	ali...get	online
详细数据	Li...As	1559199610486	42.120.75.128	2019-05-30 15:00:10...	2019-05-30 14:58:28...	ali...get	offline

共有2条, 每页显示: 10条

IoT 设备运行状态缓存到 Redis

作者 | 苏堤嘉木

今天，我们给大家带来基于云产品组合的方案，无需编写应用程序代码，实时缓存 IoT 设备上报的当前运行状态，供业务系统查询的技术方案。

一、技术原理

借助物联网平台提供的规则引擎模块，我们可以轻松的把设备上报的最新状态存储到 Key-Value 的数据库--表格存储 OTS/Redis/MongoDB 中，技术方案如下：



二、IoT 物联网开发

首先我们在 IoT 物联网控制台创建产品**水泵**，并注册 2 个设备：

物联网平台 / 设备管理 / 设备

设备

水泵

设备总数 2

● 激活设备 0

设备列表

批次管理

添加设备

批量添加

DeviceName

请输入 DeviceName

☐	DeviceName/备注名称	设备所属产品	节点类型
☐	BIXj1yasIJXmpKxymoUC	水泵	设备
☐	bYUKclQxVo6VyrG8cw73	水泵	设备

删除

禁用

启用

然后，给每个设备添加 city 标签，如下：

物联网平台

设备管理 > 设备详情

产品: 水泵

ProductKey: [复制]

DeviceSecret: [复制]

设备信息

Topic列表

运行状态

事件管理

服务调用

设备影子

文件管理

日志服务

在线调试

产品名称	水泵	ProductKey	区域
节点类型	设备	DeviceName	认证方式
备注名称	编辑	IP地址	固件版本
添加时间		激活时间	最后上线时间
当前状态	离线	实时延迟	测试

设备标签

设备标签: coordinate: 120.081901.30.130368 city: 苏州

水泵监控场景中，每 10 分钟会定时上报运行状态到 IoT 平台，具体通信 Topic 和 payload 如下：

```
// 定时上报运行状态的 Topic
/${productKey}/${deviceName}/user/bizHeart/post
// 对应 payload 结构体
{
  status: 'RUNNING', //运行状态 RUNNING, STOP, SHUTDOWN
  speed: 3000, //当前转速
  waterOutput: 125, //当前出水量
  workingTime: 72 //工作时长 xx 分钟
}
```

三、表格存储数据库

我们在表格存储 OTS 控制台创建一个数据表，以设备 `deviceName` 为主键，扩展信息为设备当前状态属性，无需预先定义。



创建完成后，数据表基本信息，如下：



四、规则引擎配置

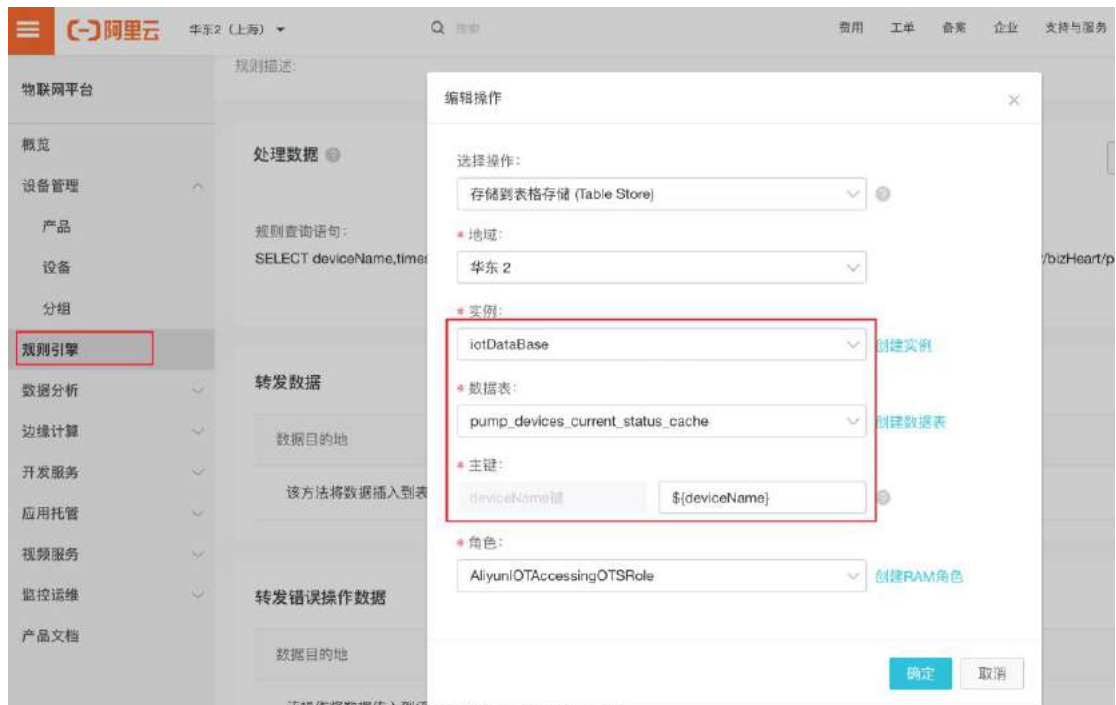
当设备上报状态数据后，需要通过预先配置规则引擎实时流转到表格存储中，规则引擎配置如下：



数据处理的 SQL 参考:

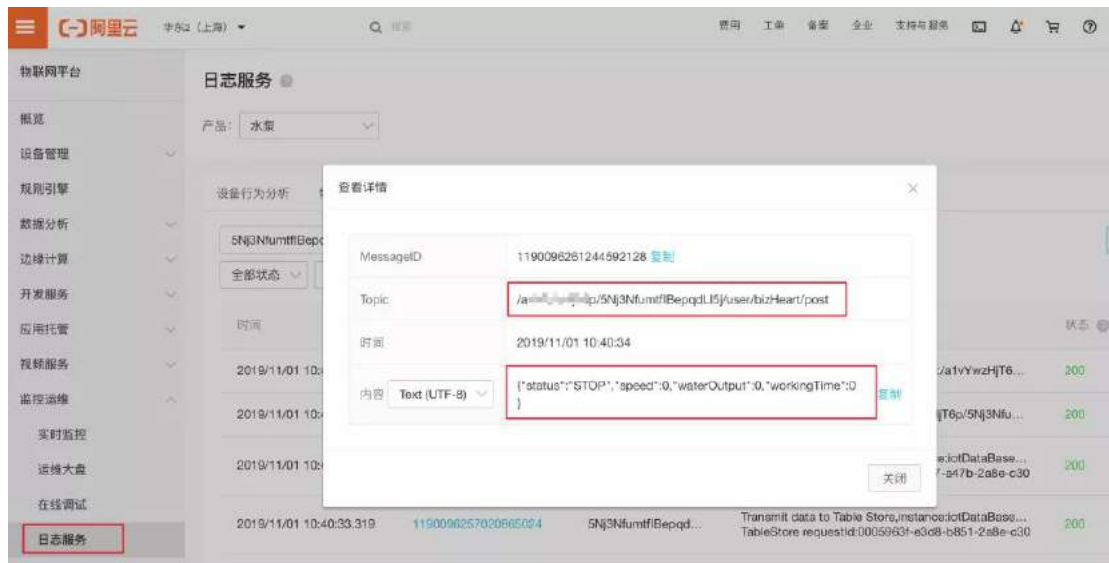
```
SELECT
deviceName() as deviceName,
attribute('coordinate') as coordinate,
attribute('city') as city,
timestamp('yyyy-MM-dd HH:mm:ss') as currentTime,
status,speed,waterOutput,workingTime
FROM
"/a1vYwzHjT6p/+user/bizHeart/post"
```

数据转发配置如下：

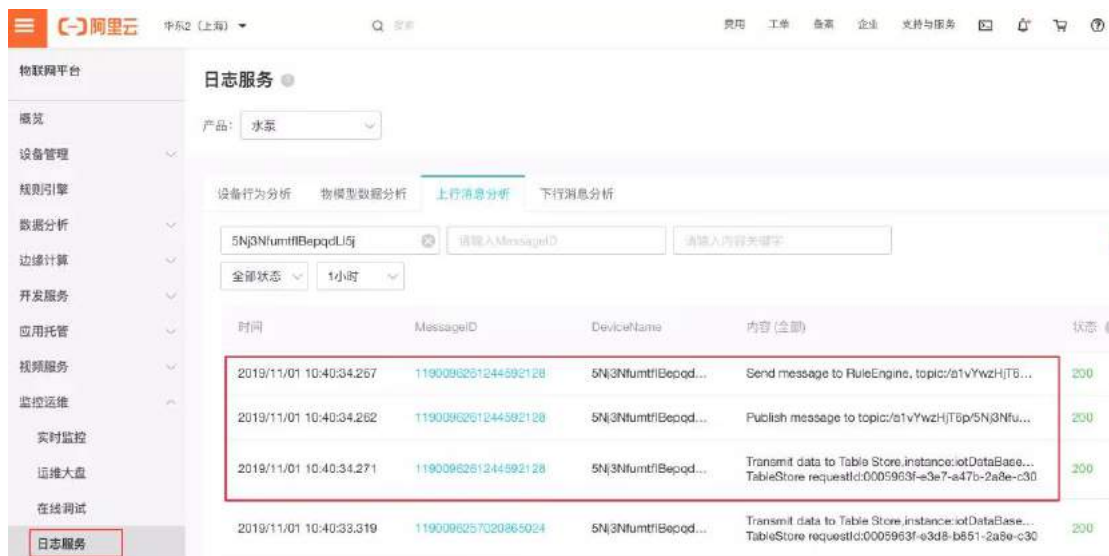


五、联机运行

当设备启动，上报状态数据后，我们在 IoT 控制台的日志服务里可以查看到如下消息报文记录：



以及规则引擎流转日志：



同时在表格存储数据库中，我们也可以查看到设备当前状态数据：

数据源: pump_devices_current_status_cache

通过deviceName可以查询设备当前运行状态(最后一份定时上报数据)

表格数据最多显示50行。

详细数据	deviceName(主键)	city	coordinates	currentTime	speed	status	waterOutput	workingTime
详细数据	2wC56xa1NoVpdgkDri	苏州	120.14916130.233887	2019-11-01 10:37:53	3600	RUNNING	137	68
详细数据	6Nj3HunmfiBopqLJis	苏州	120.06190130.139388	2019-11-01 10:40:34	0	STOP	0	0

共有2条, 每页显示: 10条

六、查询设备状态

Node.js 方式从表格存储 OTS 中读取指定 key 的数据参考代码如下:

```
var TableStore = require('tablestore');
var client = new TableStore.Client({
  accessKeyId: '账号 accessKey',
  secretAccessKey: '账号 secretAccessKey',
  endpoint: '表格存储接入点',
  instancename: '表格存储实例 ID',
  maxRetries: 3
});

var params = {
  tableName: '表名',
  primaryKey: [{ 'deviceName': '设备 Id' }],
  maxVersions: 1
};

client.getRow(params, function(err, data) {
  if (err) {
    console.log('error:', err);
    return;
  }

  if (data.row.primaryKey) {
    data.row.primaryKey.forEach(function(item) {
      console.log('=====> getRow: ', item.name + '=' + item.value);
    })
  }
})
```

```
data.row.attributes.forEach(function(item) {  
  console.log('\t\t' + item.columnName + '=' + item.columnValue);  
  })  
}  
});
```

设备最新状态数据查看：

```
$ node queryOTS.js  
=====> getRow:  deviceName=xxxxxx  
city=xxxxxx  
coordinate=xxxxxx  
currentTime=xxxxxx  
status=xxxxxx  
speed=xxxxxx  
waterOutput=xxxxxx  
workingTime=xxxxxx
```

IoT 设备之间 M2M 场景联动实战

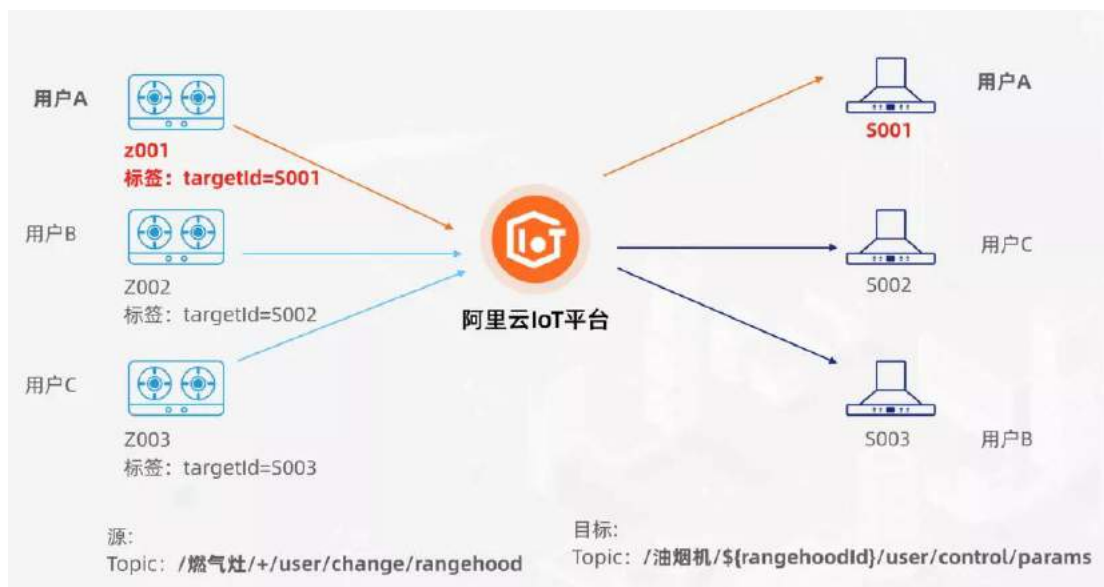
作者 | 苏堤嘉木

M2M（即 Machine-to-Machine）是一种端对端通信技术。IoT 物联网平台支持使用规则引擎的 Topic 转发功能，实现 M2M 通信，您不用担心高并发场景下的稳定通信、低延时等技术难点，也不需要购买大量服务器去承载这些请求，您只需要根据自己的业务配置好转发规则即可。

本次实战，我们讲解在厨房场景中如何实现不同消费者购买的燃气灶和抽油烟机的联动，同样适用于智能家居场景中温湿度传感器和空调联动，空气质量传感器和空气净化器联动等。

一、技术方案

基于 IoT 物联网平台中规则引擎的 M2M 能力和设备标签功能，我们可以组合出设备之间联动的技术方案，如下：



消费者 A 购买我们的燃气灶 z001 和油烟机 s001 后，业务系统会在 IoT 物联网平台给燃气灶 z001 打上设备标签 targetId:s001。

当燃气灶 z001 发送特定 Topic: /a***h/z001/user/change/rangehood 后, 会在云端规则引擎处理后, 转发给油烟机 s001 订阅的 Topic: /a***i/s001/user/control/params, 并携带业务信息, 以便抽油烟机判断是否开机和转速等级。

二、创建产品

1. 抽油烟机

在控制台创建抽油烟机产品。



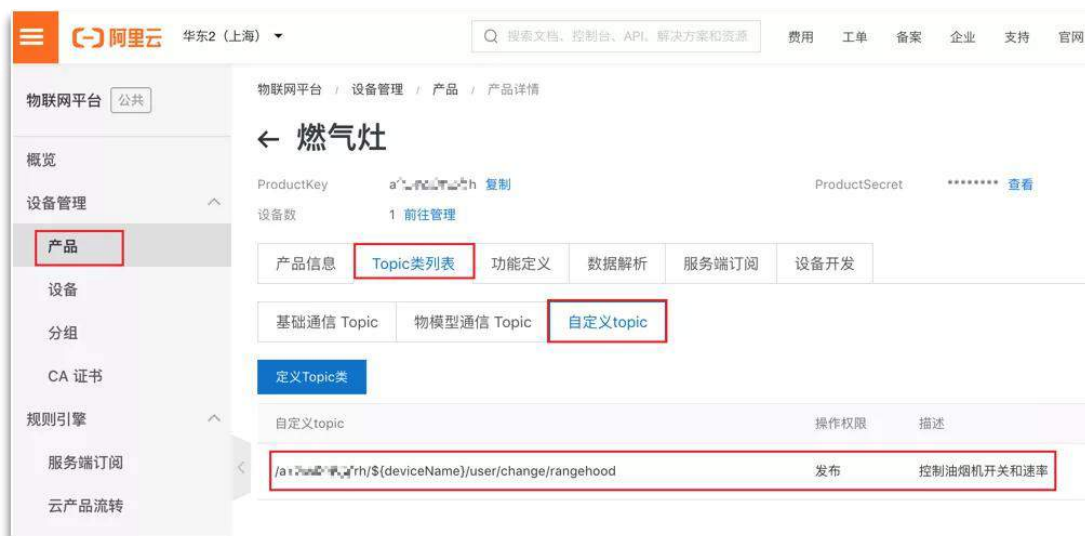
添加自定义 Topic, 用来监听运行指令。



2. 燃气灶

在控制台创建燃气灶产品。

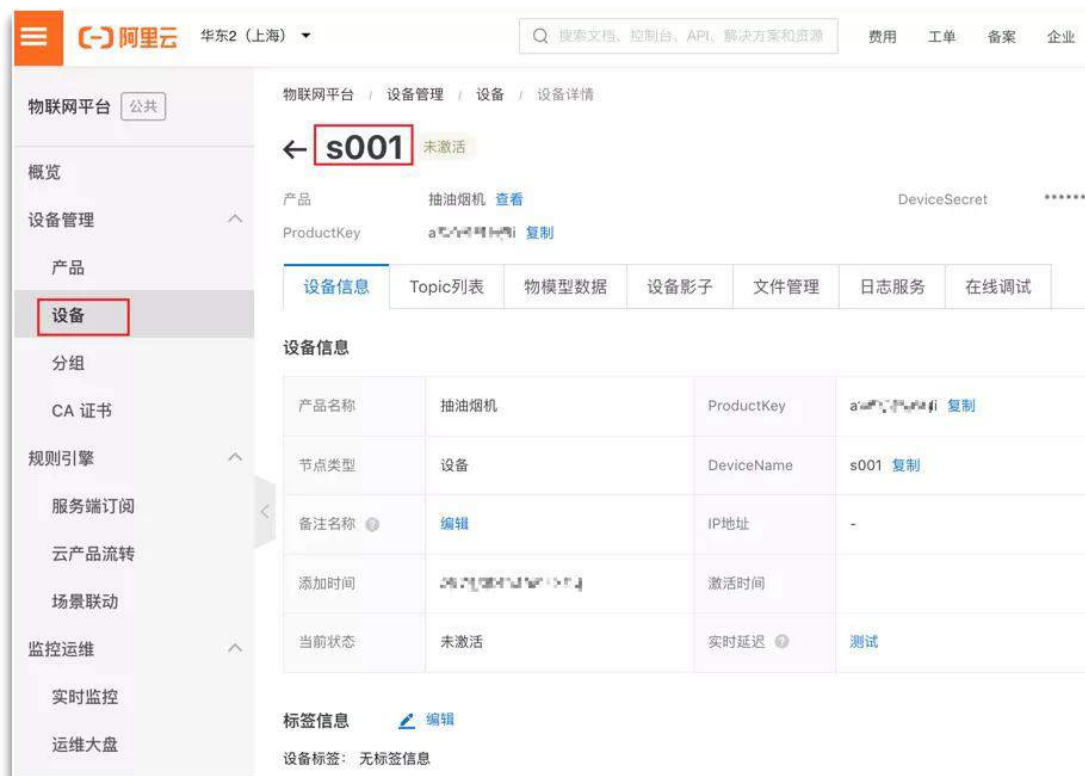




三、注册设备

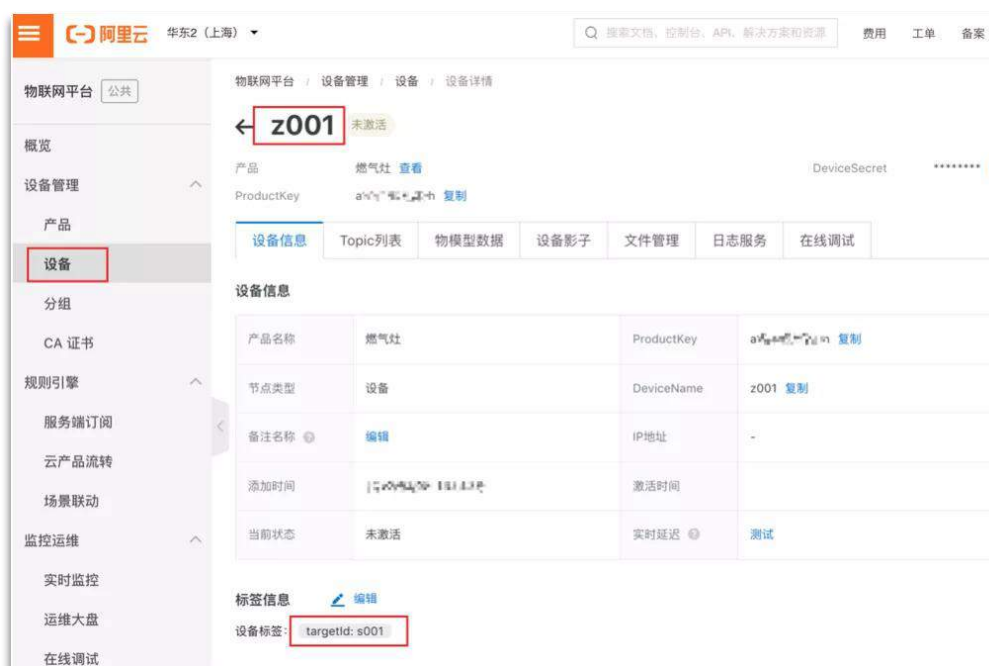
1. 油烟机设备

在控制台注册油烟机设备 s001。



2. 燃气灶设备

在控制台注册燃气灶设备 z001，并添加标签信息: targetId:s001，来绑定抽油烟机设备 s001。



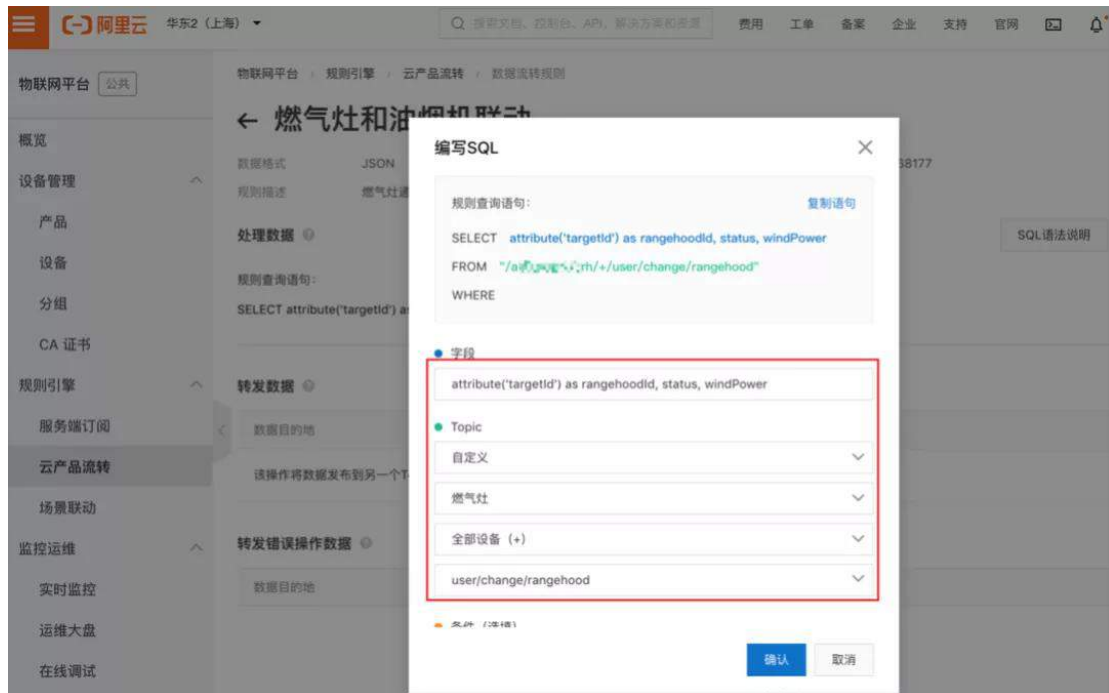
四、配置 M2M 规则

通过规则引擎，配置云产品流转规则，如下：



1. 处理数据

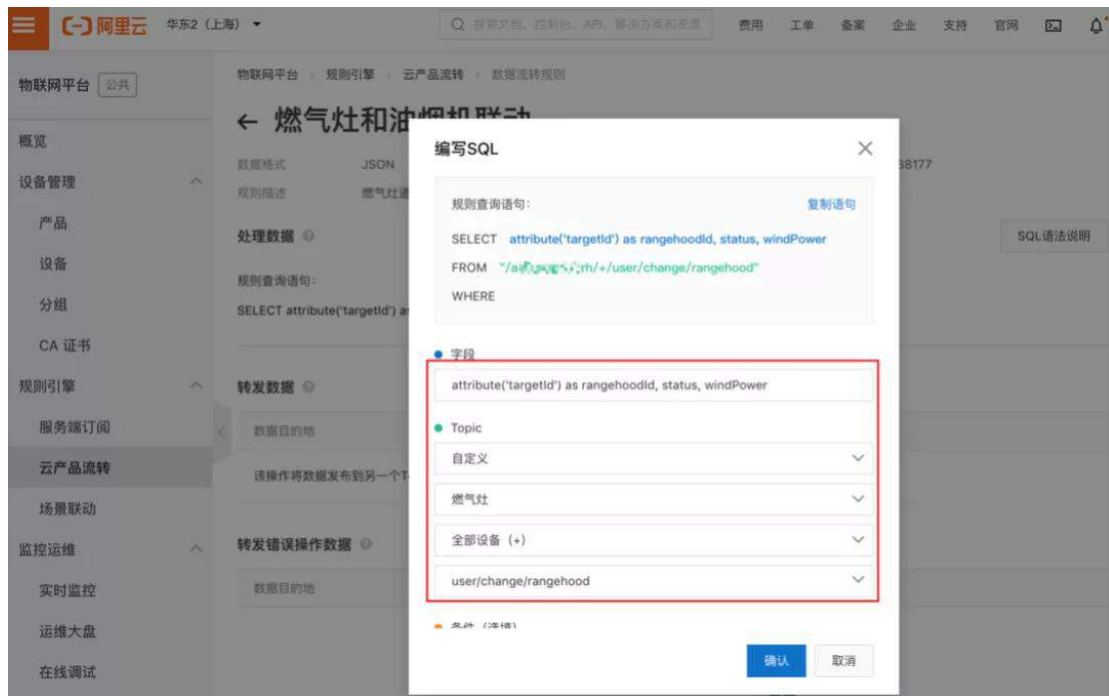
通过 SQL 规则, 我们抽取出 rangehoodId, status, windPower 共 3 个业务参数。



```
SELECT
attribute('targetId') as rangehoodId,
status,
windPower
FROM
"/a***/h/+/user/change/rangehood"
```

2. 转发数据

转发数据过程中, 我们用 `${rangehoodId}` 来动态替换目标设备。



五、设备开发

我们以 Nodejs 脚本来模拟设备行为。

1. 油烟机设备

油烟机设备 s001 主要通过 subscribe 特定 Topic 来监听云端指令。

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

var options = {
  productKey: "替换",
  deviceName: "替换",
  deviceSecret: "替换",
  regionId: "cn-shanghai"
};

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

client.subscribe(`${options.productKey}/${options.deviceName}/user/control/params`)
```

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

2. 燃气灶设备

燃气灶设备 z001 根据自身状态，上报业务数据到云端。

```
const mqtt = require('aliyun-iot-mqtt');  
  
var options = {  
  productKey: "替换",  
  deviceName: "替换",  
  deviceSecret: "替换",  
  regionId: "cn-shanghai"  
};  
  
var pubTopic = `${options.productKey}/${options.deviceName}/user/change/rangehood`;  
//建立连接  
const client = mqtt.getAliyunIotMqttClient(options);  
  
//上报数据  
client.publish(pubTopic, getPostData(), { qos: 0 });  
  
function getPostData() {  
  const payloadJson = {  
    status: "on", //on,off  
    windPower: "high" //low,middle,high  
  }  
  console.log("Pub Topic => " + pubTopic)  
  console.log("Payload => " + JSON.stringify(payloadJson))  
  return JSON.stringify(payloadJson);  
}
```

六、联机运行

我们先启动油烟机 s001 模拟程序，再启动燃气灶 z001 模拟程序。

油烟机

```
1 $ node range-hood.js
2
3 sub topic => /a1zTlSPM9Ni/s001/user/control/params
4 message => {"windPower":"high","rangehoodId":"s001","status":"on"}
```

燃气灶

```
1 $ node gas-stove.js
2
3 Pub Topic => /a***h/z001/user/change/rangehood
4 Payload => {"status":"on","windPower":"high"}
```

1. 燃气灶设备运行日志

我们看到燃气灶 z001 上报运行数据到 IoT 物联网平台，通过预先配置 M2M 规则，流转控制指令给油烟机设备 s001。

物联网平台 / 监控运维 / 日志服务

日志服务

产品: 燃气灶

云端运行日志 | 设备本地日志

请输入DeviceName | 请输入TraceId | 请输入MessageID

请输入内容关键字 | 全部状态 | 1小时

搜索 | 重置

时间	TraceID	MessageID	DeviceName	业务类型(全部)	操作	内容	状态
2020/06/30	0a30327115935032	127787134	z001	② 流转到油烟机设备	规则引擎	republish	200
2020/06/30	0a30327115935032	127787134	z001	① 燃气灶上报数据	设备到云消息	publish	200
2020/06/30	0a30327115935032	-	z001	设备行为	online	online	200

2. 油烟机设备运行日志

我们看到油烟机 s001 收到了 IoT 物联网平台流转过来的设备运行指令。

物联网平台 / 公共

概览

设备管理

规则引擎

监控运维

实时监控

运维大盘

在线调试

设备模拟器

日志服务

固件升级

远程配置

物联网平台 / 监控运维 / 日志服务

日志服务

产品: 抽油烟机

云端运行日志 设备本地日志

请输入DeviceName 请输入TraceId 请输入MessageID

请输入内容关键字 全部状态 1小时

搜索 重置

时间	TraceId	MessageID	DeviceName	业务类型	设备ID	内容	状态
2020/06/30 10:00:00	0a30327115935032	127787134	s001	云到设备消息	/at+MQTT=4/s001/user/cont...	{"Content": "Publish message to..."}	200
2020/06/30 10:00:00	0a3027d815935032	-	s001	设备行为	online	{"Content": "online, clientip=...", "clientip": "..."}	200

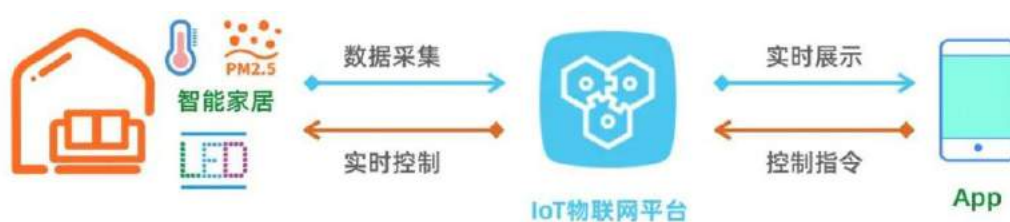
App 和 IoT 设备数据实时同步和控制

作者 | 苏堤嘉木



[点击阅读原文查看视频](#)

在物联网场景中我们经常会遇到手机 App 和智能设备实时同步状态，App 控制设备行为的需求。通过 IoT 物联网平台的规则引擎 Topic 转发(M2M)功能，即可以轻松实现 App 和智能设备之间通信。



一、产品开发

智能家居设备产品和通信 Topic 定义：



智能家居 App 产品和通信 Topic 定义：



二、设备间 M2M 通信

设备间 M2M 通信 Topic 对于关系，和 payload 结构体，如下表：

业务	传输	主题Topic	报文Payload
传感器数据	Device发送	/pk/dn/user/updata	<pre>{ "temperature":20, "humidity":55, "lightStatus":"off", "lightRGB":"black", "pm25":37, "pm10":66 }</pre>
	App订阅	/pk/dn/user/subdata	
控制指令	App发送	/pk/dn/user/cmd	<pre>{ "lightRGB":"blue", "lightStatus":"on" }</pre>
	Device订阅	/pk/dn/user/subcmd	

三、配置规则引擎

智能家居设备采集环境数据→手机 App 实时流转规则：

The screenshot shows the Alibaba Cloud IoT Platform console. The left sidebar contains navigation options: 物联网平台, 概览, 设备管理, 规则引擎, 服务端订阅, 云产品流转 (highlighted with a red box), 场景联动, 监控运维, 边缘计算, 视频服务, IoT Studio, and 数据分析. The main content area is titled '智能家居设备数据-App' (highlighted with a red box). It shows a rule configuration with the following details:

- 数据格式:** JSON
- 规则ID:** (empty)
- 规则描述:** (empty)
- 处理数据:**
 - 规则查询语句: `SELECT * FROM "/aemf-1234567890/homekit/user/updata"` (highlighted with a red box and labeled '设备上报Topic' with a red arrow).
- 转发数据:**
 - 数据目的地: (empty)
 - 该操作将数据发布到另一个Topic中: `/aemf-1234567890/miniApp/user/subdata` (highlighted with a green box and labeled 'App接收topic' with a green arrow).

A red arrow labeled '转发(republish)' points from the source topic to the destination topic.

手机 App 控制指令→设备实时流转规则：



四、设备开发

硬件设备：

名称	图片
Ruff主板	
LED灯	
温湿度传感器	
空气质量传感器	
LCD显示屏	

设备端核心代码参考：

```
var MQTT = require('aliyun-iot-device-mqtt');

// 设备身份信息
var options = {
  productKey: "",
  deviceName: "",
  deviceSecret: "",
  regionId: "cn-shanghai",
};

var sendDataTopic = "/" + options.productKey + "/" + options.deviceName + "/user/updata";
var subCmdTopic = "/" + options.productKey + "/" + options.deviceName + "/user/subcmd";

var client = MQTT.createAliyunIotMqttClient(options);

$.ready(function (error) {
  if (error) {
    console.log(error);
    return;
  }

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

    console.log('message', JSON.stringify(message));
    // 执行 App 的控制指令
    if (topic == subCmdTopic) {
      message = JSON.parse(message.toString())
      doAction(message)
    }

  })

  // 上报传感器采集的环境数据
  setInterval(function(){
    client.publish(sendDataTopic, getHomeData());
  }, 10 * 1000);
```

```
});  
  
$.end(function () {  
    $('#led-r').turnOff();  
});
```

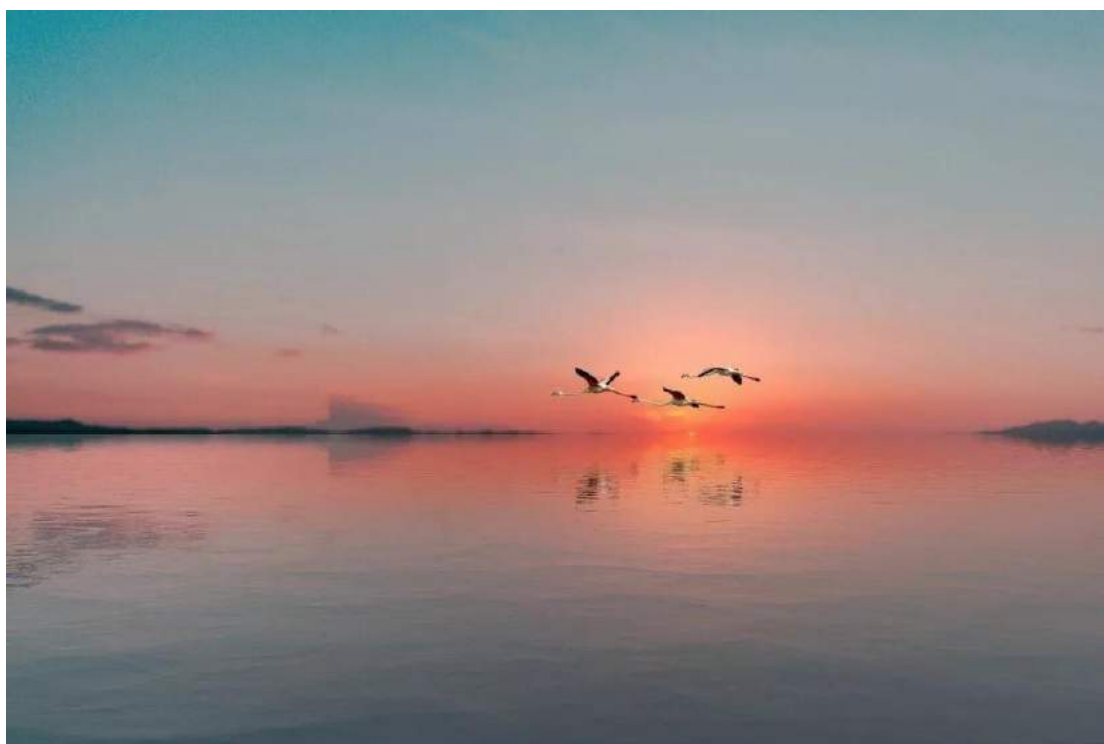
五、微信小程序

小程序以 MQTT 进入 IoT 平台，简单交互界面：



IoT 物联网平台 3 种计费方式

作者 | 苏堤嘉木



阿里云 IoT 物联网服务云产品目前有 3 种计费方式，总结起来如下：

公共版 <u>后付费</u> 特性： ✓ 消息量 1.8元/100万 ✓ 在线时长 1.0元/100万 提供设备接入、管理、运维、数据存储的企业物联网服务 公共道路	标准版实例 提供设备接入、管理、运维、数据存储的企业物联网服务 特性： ✓ 购买实例可享受更好的资源隔离和管控策略， <u>实例间互不影响</u> ✓ 现在购买实例，可享受 <u>时序数据存储</u> 在实例有效期内免费的优惠 专用车道	铂金版实例 <u>资源完全独享</u> 的企业物联网服务公共云实例 特性： ✓ 铂金版实例的计算、网络、存储资源完全独享，稳定性好，SLA达到 <u>99.95%</u> ✓ 购买铂金版实例可享受大客户支持， <u>7*24小时专业运维团队保障</u> 独享道路
--	---	--

一、公共版(后付费)

阿里云 IoT 物联网服务云产品支持默认的后付费模式，按消息量，在线时长，固件升级次数来收费，用多少付多少。

开通入口：<https://iot.console.aliyun.com/>



消息通信计费：

消息数量N（条/月）	单价（元/百万条消息）
$N \leq 1\text{亿}$	1.8
$1\text{亿} < N \leq 10\text{亿}$	1.4
$10\text{亿} < N$	1

连接时长计费：

计费方式：1 元/每百万分钟

固件升级费：

计费方式：按 0.2 元/次/设备。

资源包：

后付费模式下，支持购买资源包，进一步降低我们的成本。

资源包购买入口：https://common-buy.aliyun.com/?&commodityCode=iot_resource_bag#/buy

物联网平台资源包

基本配置

资源包类型

消息通信包

连接时长包

固件升级次数包

温馨提示：资源包购买后优先抵用资源包，所有地域通用。超出后会按量计费。计费详情参考[文档](#)

消息数

250000万消息

500000万消息

1000000万消息

10000 万消息

购买量

购买时长

1个月

3个月

6个月

1年

3年

资源包购买成功后即刻生效。购买数量和时长越多折扣越多。资源包生效后不支持退订

购买数量是指每个月资源包所含的数量。当月未使用完不会累积到下一个月。当月超出后将会按量计费

购买时长是指连续购买对应数量资源包的月数。每个月资源包所含的数量是独立的

二、标准版实例

标准版实例，提供面向企业客户的物联网平台实例，支持设备接入、管理运维、数据存储等功能。根据企业 IoT 业务情况，评估设备接入量，消息上下行 TPS，规则引擎流转 TPS，时序数据库写入 TPS，时序数据库存储规格，来确定实例费用。

标准版实例优势：

- 实例自带时序数据存储功能，无需额外购买其他存储，即可保存设备上报数据。
- 购买实例可享受更丰富的产品功能，更好的数据隔离策略，更高的 SLA 可用性保障。

购买入口：https://common-buy.aliyun.com/?&commodityCode=iot_instc_public_cn#/buy

三、铂金版实例

铂金版实例是在标准版实例的基础上，各种运行资源完全独享的物联网平台服务实例。目前只能通过联系客户经理来开通。

铂金版实例优势：

- 铂金版实例的计算、网络、存储资源完全独享，稳定性好，SLA 达到 99.95%
- 购买铂金版实例可享受大客户服务支持，7*24 小时专业运维团队保障。



IoT 平台消息轨迹全景图

作者 | IoT 物联网技术

阿里云 IoT 企业物联网平台上线了消息轨迹全景图功能，帮助 IoT 开发者追踪消息通信的完整轨迹，快速分析和定位问题，及时恢复业务。



消息轨迹功能：

- 通过 Traceld 展示消息流转的全路径。
- 展示消息流转总耗时和平台发送耗时。
- 绿色=成功，红色=失败，并有错误原因。

一、消息轨迹 实战

设备接入 IoT 企业物联网平台完整开发过程请移步：[IoT 设备到业务服务端全链路开发实战](#)。

1. 设备上行消息轨迹

设备启动后，我们在[设备详情](#)，看到设备状态为**在线**，物模型数据中可以看到最新上报的**温度**和**湿度值**。

物联网平台 / 设备管理 / 设备 / 设备详情

← hxt93489234 在线 ① 设备状态

产品 家庭温控器 查看 DeviceSecret ***** 查看

ProductKey g9... 复制

设备信息 Topic 列表 物模型数据 设备影子 文件管理 日志服务 在线调试 分组

运行状态 事件管理 服务调用

请输入属性名称或标识符

湿度 17% 查看数据 2020/12/15 15:15:57.78

温度 20 °C 查看数据 2020/12/15 15:15:57.78

② 上报采集数据

IoT物联网技术

在监控运维的日志服务里，也可以看到设备上报数据的日志。如下图：

物联网平台 / 监控运维 / 日志服务

日志服务

产品: 家庭温控器

云端运行日志 设备本地日志 日志转储

hxt93489234 请输入 TraceId 请输入内容关键字, MessageId

全部状态 1 小时

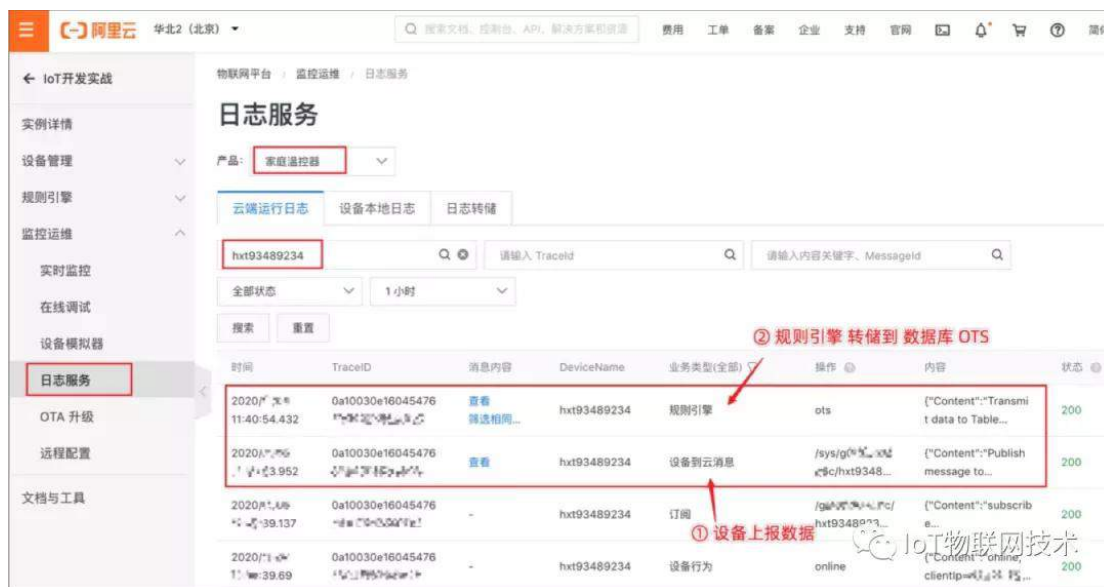
搜索 重置

① 设备上报数据日志

时间	TraceId	消息内容	DeviceName	业务类型(设备到云消息)	操作	内容	状态
2020-12-15 15:15:57.78	0a10030616045463	查看	hxt93489234	设备到云消息	/sys/g... q6c/hxt9348...	("Content": "Publish message to...	200
2020-12-15 15:15:57.78	0a10030616045462	查看	hxt93489234	设备到云消息	/sys/g... q6c/hxt9348...	("Content": "Publish message to...	200
2020-12-15 15:15:57.78	0a10030616045462	查看	hxt93489234	设备到云消息	/sys/g... q6c/hxt9348...	message to...	200

IoT物联网技术

规则引擎流转日志，如下图：



在日志服务的消息轨迹页面，输入 TraceID，查看可视化的全链路流转图。如下图：



消息轨迹全景图，我们可以追踪到完整消息流转日志：

从设备端发出，到达云端接入网关，流转到消息中心，通过规则引擎分发到表格存储 OTS 和服务端订阅 AMQP 队列全过程和对应时间点。

2. 云端下行消息轨迹

当我们通过 Pub API 下发控制指令到设备后，在企业实例的控制台，**日志服务**中，可以追踪到完整的下行链路日志，如下图：



在日志服务的消息轨迹页面，输入 TraceID，查看可视化的全链路流转图。如下图：



消息轨迹全景图，我们可以追踪到 Pub 下行控制指令消息流转日志：

从应用服务器发出，到达 IoT 云平台的消息中心，通过接入网关到推送到 IoT 硬件设备全过程和对应时间点。

IoT 平台性能压测工具 JMeter

作者 | 苏堤嘉木



一、背景

MQTT 是超轻量级消息协议，用于连接移动端与云服务双向通信，广泛应用于物联网（IoT）领域，如设备向云端上报状态、云端向设备推送消息、设备端 A 向设备端 B 发送消息等场景。

本文以充电宝机柜上报状态场景为例介绍如何使用 MQTT 插件和 JMeter 脚本压测 IoT 物联网平台的 MQTT 服务性能。

二、准备工作

开通 IoT 物联网设备接入服务。<https://www.aliyun.com/product/iot-deviceconnect>

安装 JMeter 5.1.1 版本 版本 <https://jmeter.apache.org>

三、压测实战

1. 创建产品和注册设备

我们在 IoT 物联网平台设备管理中，创建一个充电宝产品，并注册 10 个设备，获得身份三元组。

通信Topic	消息Payload
/{{pk}}/{{dn}}/user/update	{"battery":0.86}

2. 准备设备身份信息 CSV 文件

每个设备和 IoT 物联网平台建立连接时，需要提供 Username 、 Password 、 ClientId 这 3 个身份信息，上报状态数据时需要知道自身的 productKey 和 deviceName 来确定通信 Topic。

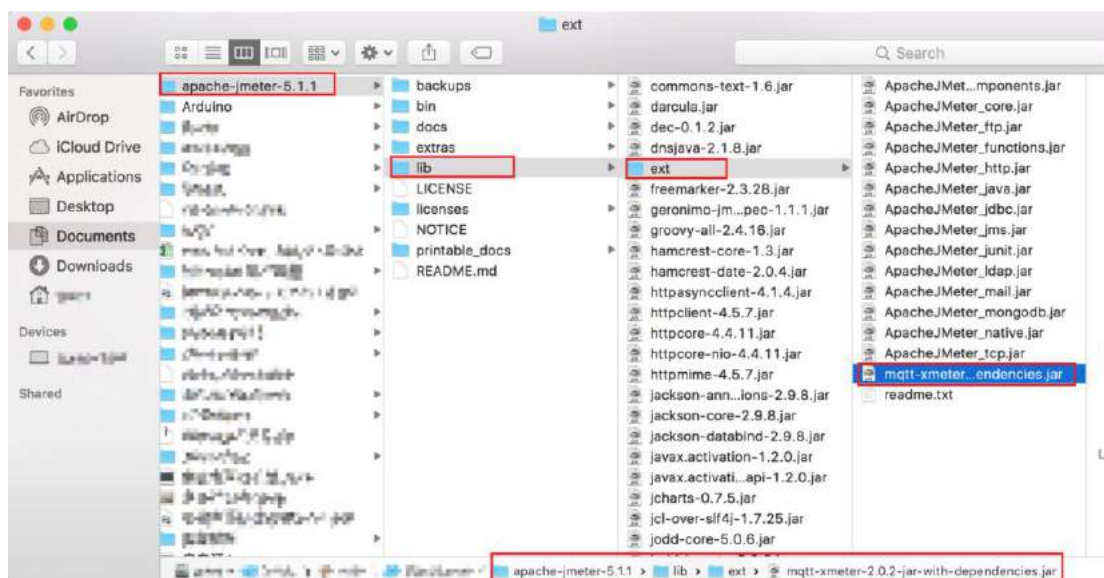
依照 IoT 物联网平台身份认证文档，我们把三元组转换成 Username 、 Password 、 ClientId，存储到 client.csv 文件中。示例如下：

```
1 username#password#clientId#pk#dn
2 tdxZyTqh0I6yA8zBIBQ7&a2YcQKbPGbE#d47fca23df873
3 gdxZyTqh0I6yA8zBIBQ7&a2YcQKbPGbE#d47fca23df873
4 cdxZyTqh0I6yA8zBIBQ7&a2YcQKbPGbE#d47fca23df873
```

3. 下载 Jmeter ， 安装 MQTT 插件

下载 mqtt-jmeter 插件最新版本 JAR 包：mqtt-xmeter-2.0.2-jar-with-dependencies.jar 。下载地址：<https://github.com/emqx/mqtt-jmeter>

拷贝插件 JAR 包到 JMeter 安装目录的 lib/ext/ 子目录下。操作过程如下：

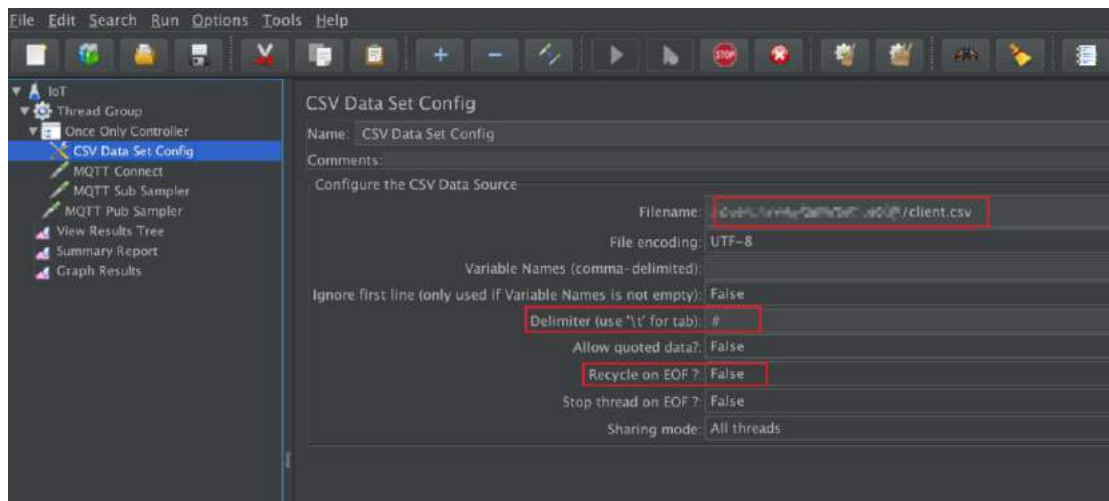


4. 编辑 JMeter 脚本

我们以 JMeter 5.1.1 英文图形界面为例。

配置客户端 CSV 数据文件

- 打开 JMeter，并新建脚本。
- 在 JMeter 左侧文件目录中右键单击 Test Plan，选择 Add > Threads (Users) > Thread Group。
- 在 JMeter 左侧文件目录中右键单击 Test Plan，选择 Add > Listener > View Results Tree，添加 View Results Tree 监听器，方便本地调试测试脚本。
- 在 Test Plan 区域右键单击 Thread Group，选择 Add > Logic Controller > Once Only Controller。JMeter 中一个线程模拟一个 MQTT 客户端设备，使用 Once Only Controller 保证一个线程仅读取一次客户端 CSV 数据文件，绑定一条客户端信息。
- 在 Test Plan 区域右键单击 Once Only Controller，选择 Add > Config Element > CSV Data Set Config。并在 CSV Data Set Config 对话框中配置以下信息。

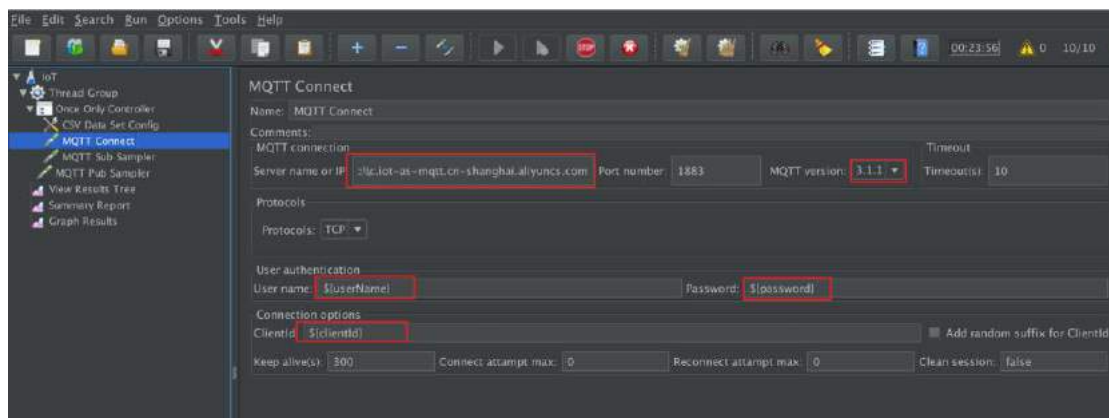


- Filename：客户端信息 CSV 文件路径 client.csv。
- File encoding：CSV 文件编码格式，本示例使用 UTF-8。
- Delimiter：这里我们输入 #。
- Recycle on EOF：是否循环读取文件。选择 False。

建立 MQTT 连接

我们使用 Once Only Controller 控制一个客户端只需执行一次建连操作。

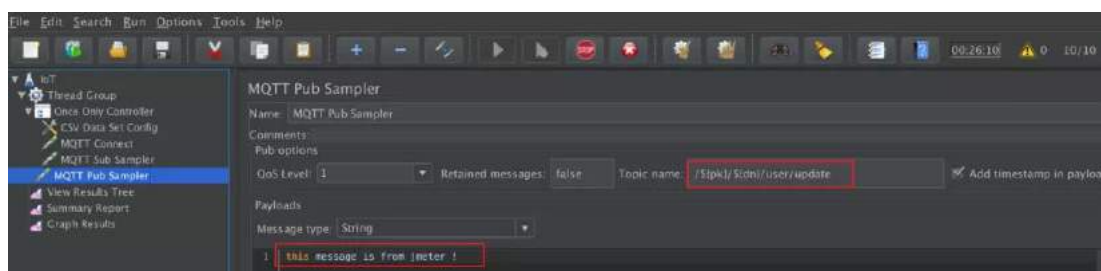
- 在 Test Plan 区域右键单击 Once Only Controller，选择 Add > Sampler > MQTT Connect。
- 在 MQTT Connect 对话框中配置以下信息。



配置	参数	说明
MQTT 连接配置	Server name or IP	MQTT 服务器公网地址。客户端设备通常使用公网访问 MQTT 服务。
	Port number	MQTT 服务器端口。例如 1883，即使用标准 TCP 端口。
	MQTT version	MQTT 版本。例如 3.1.1，目前主流 MQTT 服务器都支持 3.1.1 版本。
	Timeout(s)	超时秒数填写，即客户端建立连接、发送消息等相关操作的超时时间。例如 10。
	Protocols	连接协议。选择 TCP，即使用标准 TCP 连接协议。
MQTT 客户端配置	User name	从 CSV 文件读取 userName 字段。填写 \${userName}。
	Password	从 CSV 文件读取 Password 字段。填写 \${password}。
	ClientId	从 CSV 文件读取 ClientId 字段。填写 \${clientId}。
	Add random suffix for ClientId	是否添加后缀。本例使用预先准备好的固定客户端 ID，不要添加后缀，则取消勾选。
	Keep alive(s)	活动心跳间隔秒数。例如 300，即连接空闲时，每 5 分钟发送一次活动心跳。

配置发布消息

- 在 Test Plan 区域右键单击 Thread Group，选择 Add > Sampler > MQTT Pub Sampler。
- 在 MQTT Pub Sampler 对话框中配置以下信息。

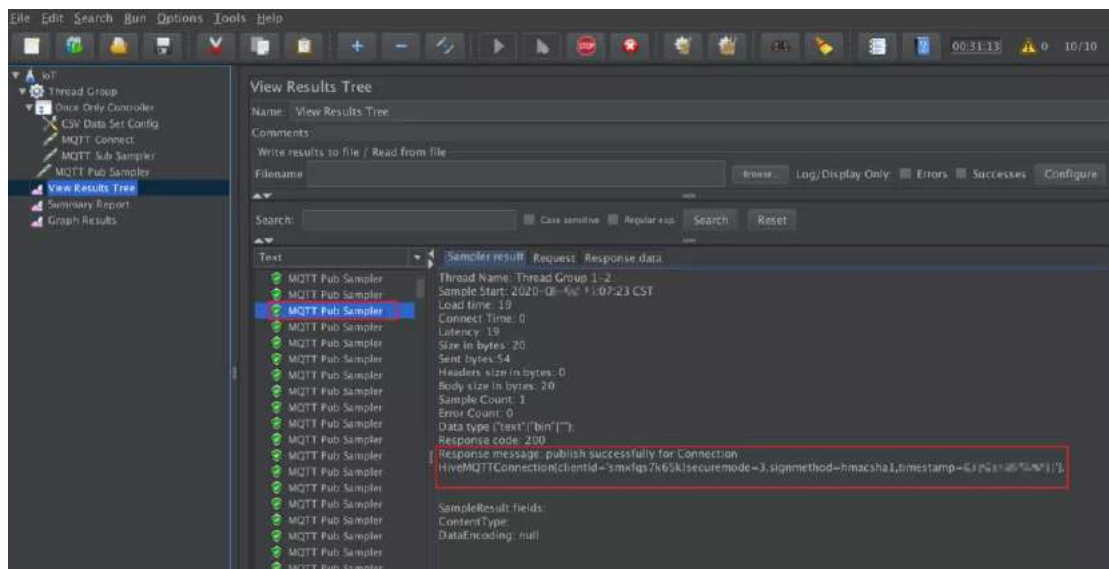


- QoS Level:** 客户端向服务器发布消息的服务质量。本示例中选择 0，即只发送一次，丢失不重发，可按需选择其他级别。

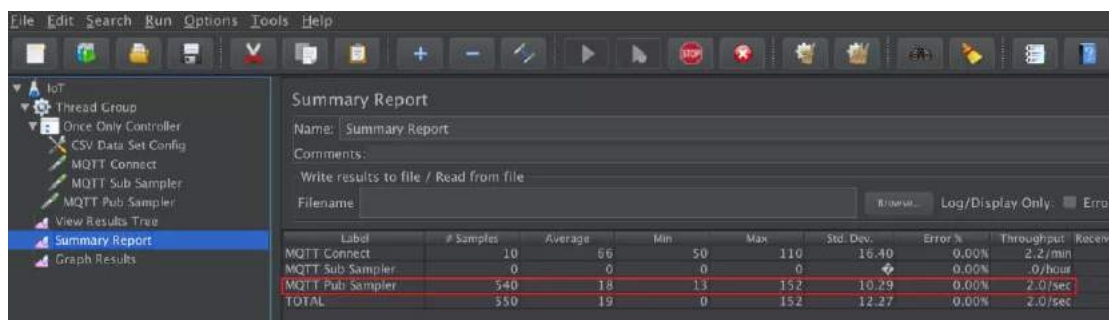
- **Topic name:** 消息 topic 。MQTT topic 支持层次结构，使用 / 分割，类似文件路径，如 pts_test/jmeter 等。
- **Add timestamp in payload:** 是否添加消息头添加发送时间戳。一般勾选此项，方便测试时检查消息延迟。
- **Payloads:** 消息体。本示例中填写 this message is from jmeter \${clientId}!, 即在消息体中添加客户端 ID，方便测试和调试检查。

启动压测脚本

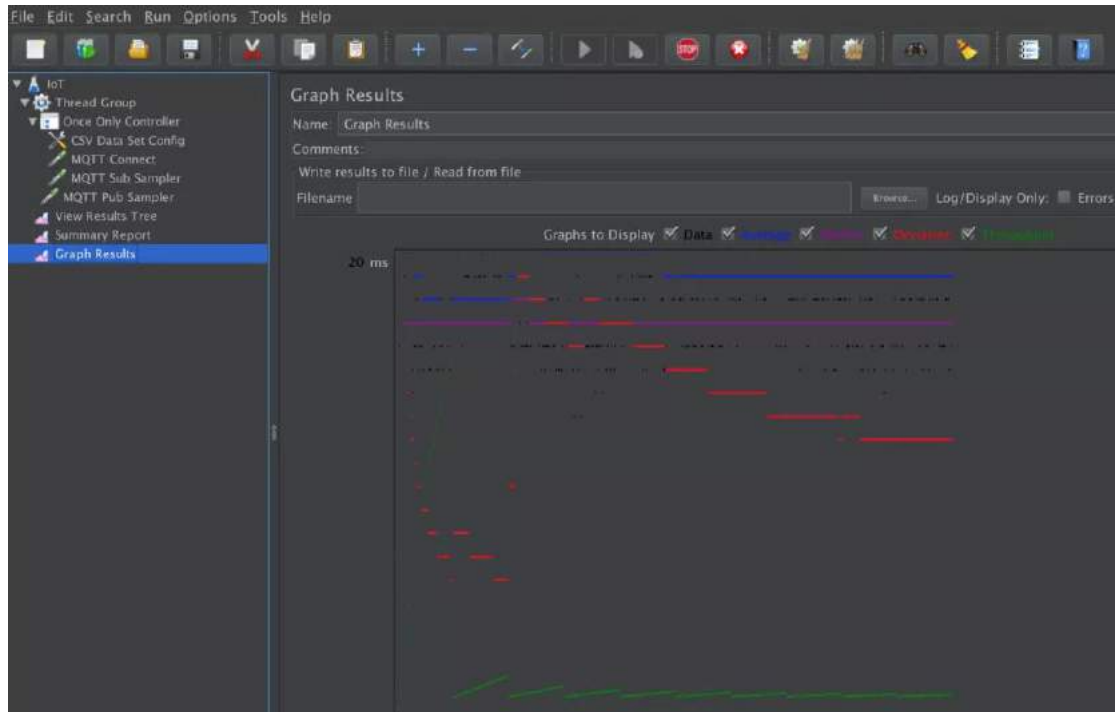
- 在 Test Plan 区域单击 Thread Group，配置 Loop Count 为 10（循环执行 10 次）。
- 在 JMeter 页面左上角单击保存，然后执行脚本。
- 在 View Results Tree 页面查看脚本执行结果：



- 在 Summary Report 页面查看脚本执行结果：



- 在 Graph Results 页面查看脚本执行结果：



IoT 物联网平台控制台日志

物联网平台 / 监控运维 / 日志服务

日志服务

产品: 物联网平台

云端运行日志 设备本地日志

请输入DeviceName 请输入TraceId 请输入MessageID

请输入内容关键字 全部状态 24小时

搜索 重置

时间	TraceID	MessageID	DeviceName	业务类型(设备到云消息)	操作	内容	状态
2020/06/03 10:10:10	0a30265d15011682840	1268077800	dn308	设备到云消息	308/user/update	("Content": "Publish message to...	200
2020/06/03 10:10:10	0a30265d15011682842	1268077800	dn308	设备到云消息	308/user/update	("Content": "Publish message to...	200
2020/06/03 10:10:10	0a30265d15011682841	1268077800	dn308	设备到云消息	308/user/update	("Content": "Publish message to...	200
2020/06/03 10:10:10	0a30265d15011682841	1268077800	dn308	设备到云消息	308/user/update	("Content": "Publish message to...	200
2020/06/03 10:10:10	0a30265d15011682841	1268077800	dn308	设备到云消息	308/user/update	("Content": "Publish message to...	200

IoT 设备 OTA 固件升级实践

作者 | 苏堤嘉木



一、什么是固件升级

固件升级 OTA (Over-the-Air Technology) 即空中下载技术, 是 IoT 物联网平台必备的一项基础功能。通过 OTA 方式, 我们可以对分布在全球各地的 IoT 设备进行设备固件升级, 而不必让运维人员各地奔波。本文以 MQTT 协议下的固件升级为例, 介绍 OTA 固件升级流程、数据流转使用的 Topic 和数据格式。

二、固件升级 OTA 流程

MQTT 协议下固件升级流程如下图所示:



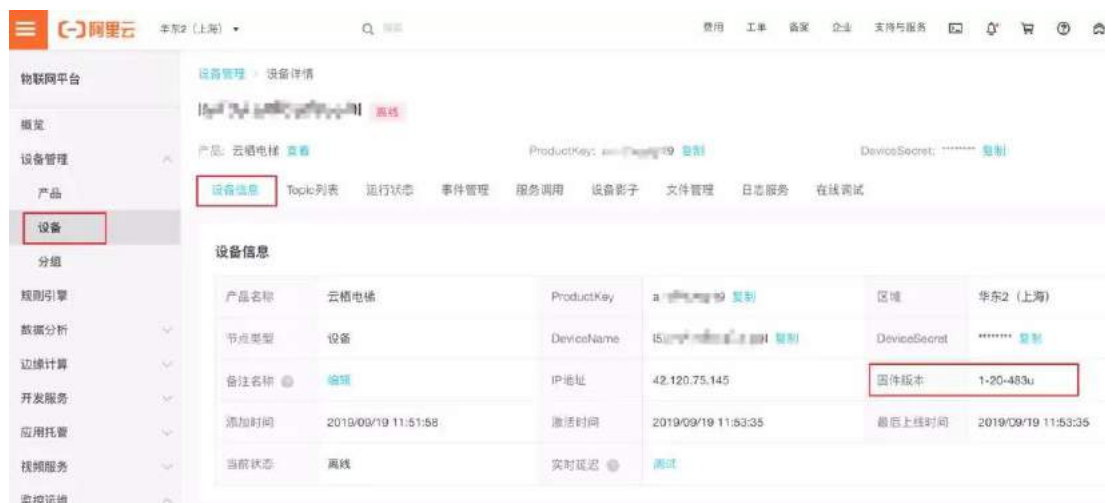
固件升级过程使用的 Topic 如下列表：

1. 设备端通过以下 Topic 上报固件版本给物联网平台。
`/ota/device/inform/${YourProductKey}/${YourDeviceName}`
2. 设备端订阅以下 Topic 接收物联网平台的固件升级通知。
`/ota/device/upgrade/${YourProductKey}/${YourDeviceName}`
3. 设备端通过以下 Topic 上报固件升级进度。
`/ota/device/progress/${YourProductKey}/${YourDeviceName}`

三、固件升级实战

1. 设备版本信息

为了实现固件升级功能，首先设备要正确上报当前固件版本，我们在设备详情可以查看到。



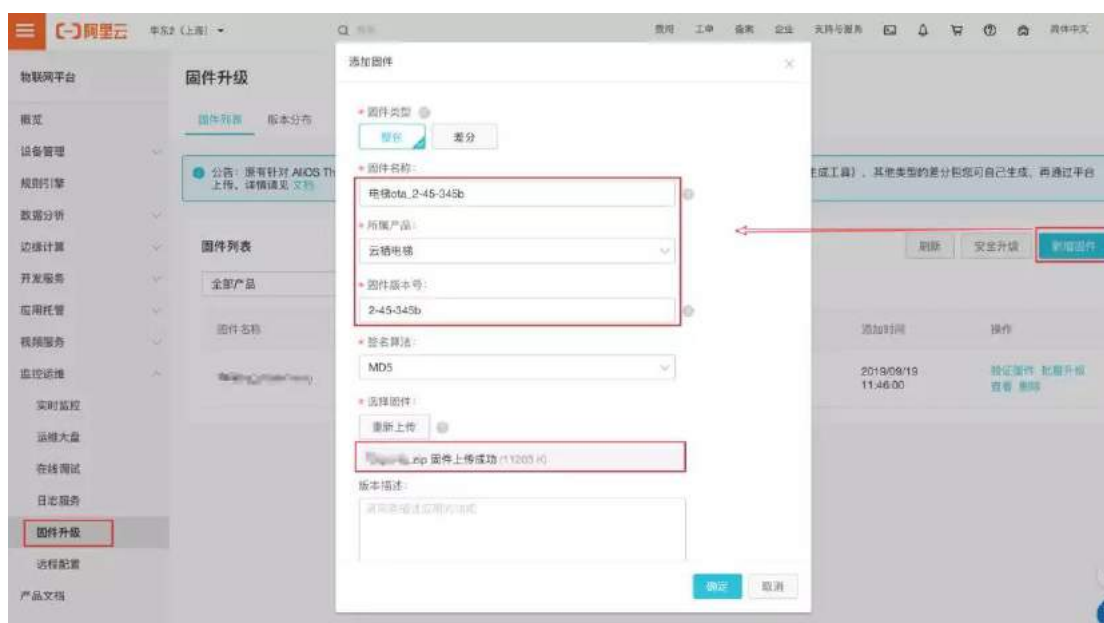
2. 固件版本分布

当每个设备都准确上报固件版本时，我们可以在控制台查看到全量设备的版本发布情况。



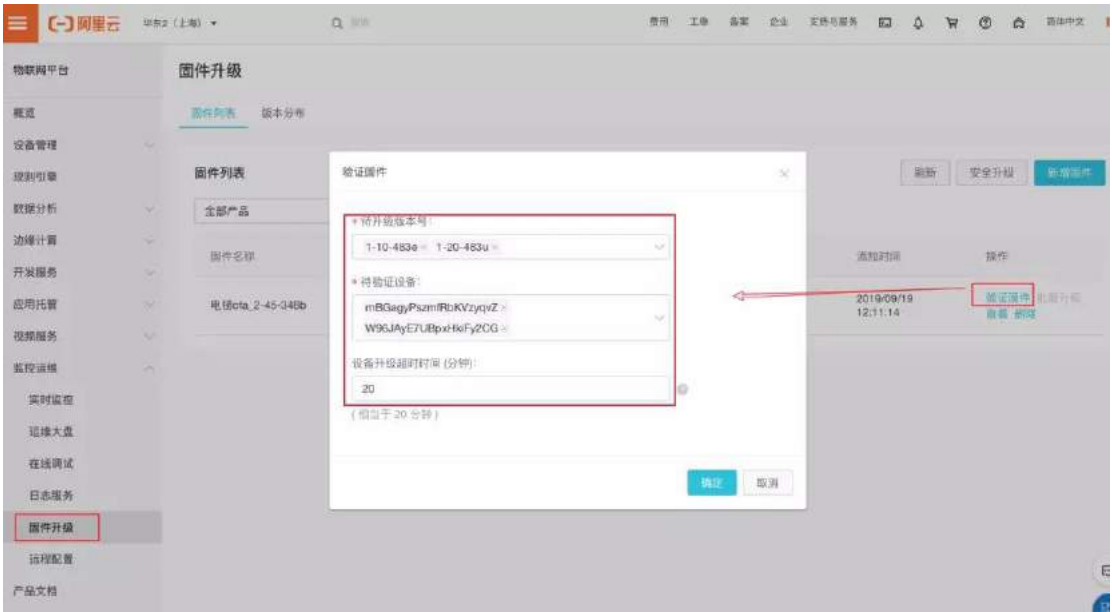
3. 上传新版固件

当我们需要做设备固件升级时，首先要上传新版本固件到 IoT 物联网平台，标记新版本号。

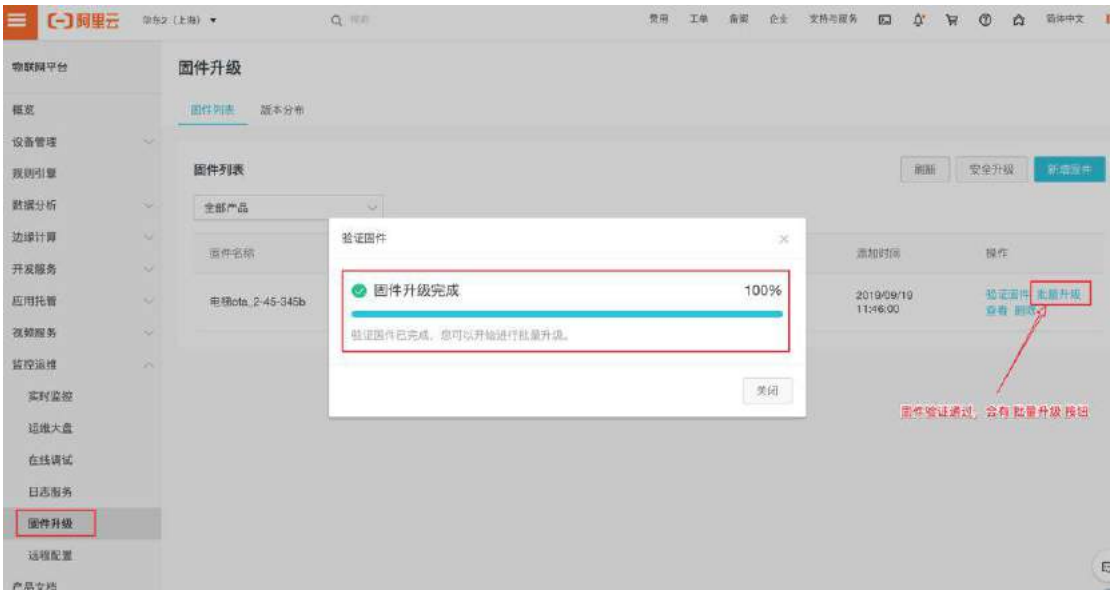


4. 验证固件

新固件上传后，我们需要筛选测试设备，来验证固件是否正常，避免新固件导致设备业务异常。



验证通过后，会看到批量升级功能变为可用状态。

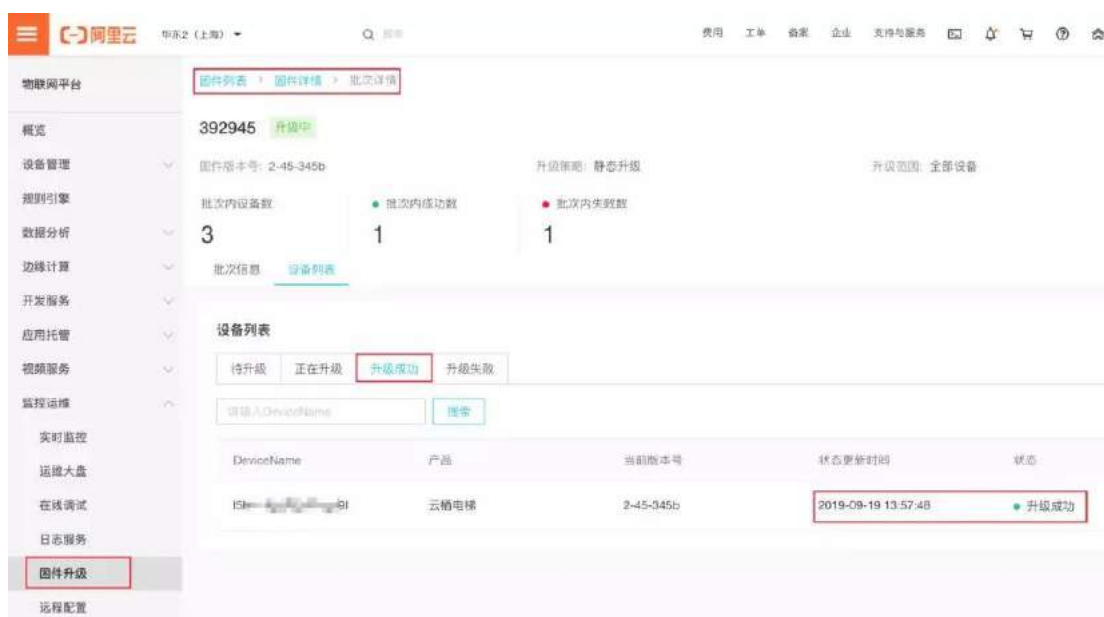


5. 批量升级

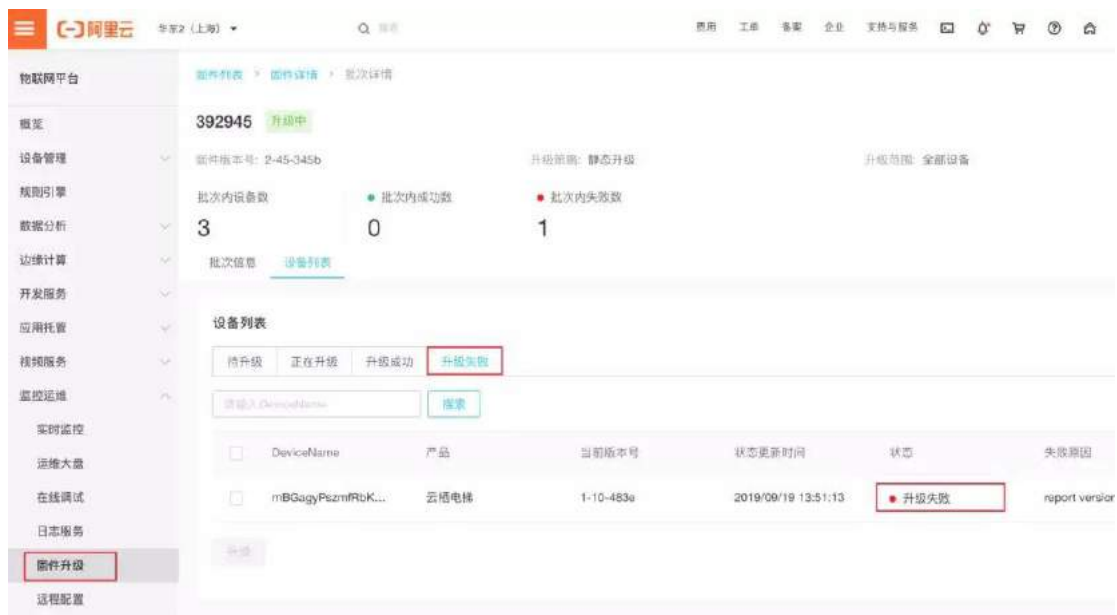
点击批量升级菜单，进入升级配置页面。我们可以从多个维度筛选待升级的设备，配置升级策略。



升级成功 Tab 会展示已经完成固件升级的设备列表。包括当前固件版本，更新时间，状态。



升级失败 Tab 会展示已经升级失败的设备列表。包括当前固件版本，更新时间，失败原因。



四、附录

IoT 物联网平台推送到设备端的升级消息 Payload 示例：

```
{
  "code": "1000",
  "data": {
    "size": 11472299,
    "sign": "83254ac96e141affb8aa42cbfec93723",
    "version": "2-45-345b",
    "url": "https://iotx-ota.oss-cn-shanghai.aliyuncs.com/ota/dbab6f742ae389b40db88fc2500b08d0/ck0q5lyav00003i7hezxe0cbg.zip?Expires=1568951190&OSSAccessKeyId=cS8uRRy54RszYWna&Signature=nk0sogaxtyp7dYvKZnjNQ%2BZ8Q9w%3D",
    "signMethod": "Md5",
    "md5": "83254ac96e141affb8aa42cbfec93723"
  },
  "id": 1568864790381,
  "message": "success"
}
```

设备固件升级模拟代码：

```
const fs = require('fs');
const path = require('path');
const mqtt = require('aliyun-iot-mqtt');
//设备身份三元组+区域

const options = {
  productKey: "替换 pk",
  deviceName: "替换 dn",
  deviceSecret: "替换 ds",
  regionId: "cn-shanghai"
}

//建立连接
const client = mqtt.getAliyunIotMqttClient(options);
//订阅 ota 消息的 Topic
const deviceUpgrade = `/ota/device/upgrade/${options.productKey}/${options.deviceName}`
client.subscribe(deviceUpgrade)

//每次连接后，上报当前固件版本
const deviceInform = `/ota/device/inform/${options.productKey}/${options.deviceName}`
client.publish(deviceInform, getFirmwareVersion("1-45-345a"))

//OTA 过程中，上报进度
const deviceProgress = `/ota/device/progress/${options.productKey}/${options.deviceName}`

// 消息处理
client.on('message', function(topic, message) {

  if (topic == deviceUpgrade) {
    //收到 ota 消息，开始升级过程
    doUpgrade(message)
  }

})

// 本地更新
function doUpgrade(message) {
  message = JSON.parse(message)
```

```
// 1.从 url 下载固件包，更新下载进度...
client.publish(deviceProgress, getOTAUpgradeData(23))
// 2.根据 signMethod 验证文件签名是否和 sign 值一致
// verifyFirmware()
// 3.重启设备，升级固件
// burn & reboot()
```

```
}
```

```
// 更新升级进度
```

```
function getOTAUpgradeData(step) {
  const payloadJson = {
    "id": 1,
    "params": {
      "step": step,
      "desc": "xxxxxxxx "
    }
  }
  console.log(payloadJson)
  return JSON.stringify(payloadJson);
}
```

```
// 设备当前固件版本
```

```
function getFirmwareVersion(version) {
  const payloadJson = {
    "id": 1,
    "params": {
      "version": version
    }
  }
  console.log(payloadJson)
  return JSON.stringify(payloadJson);
}
```

用 Wireshark 抓包分析 IoT 设备网络行为

作者 | 苏堤嘉木



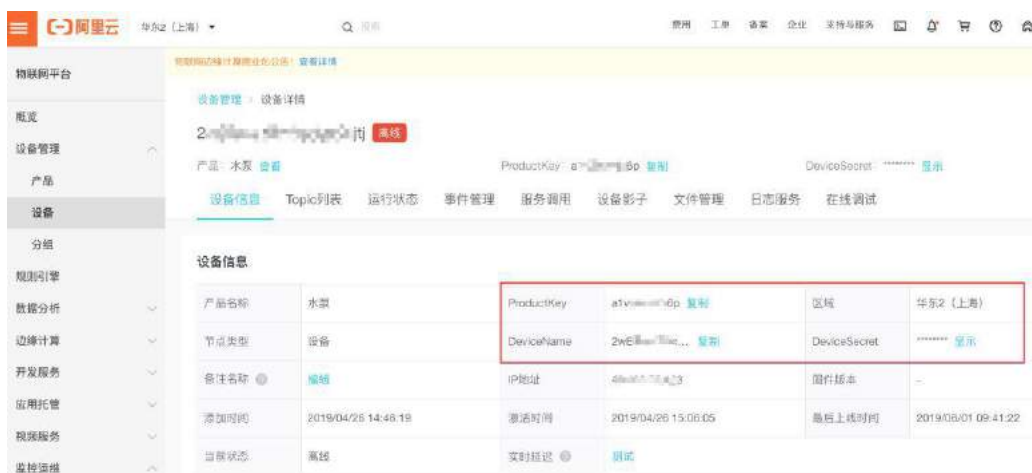
一、背景

当我们进行物联网开发过程中，设备调试有时候很难进行，就需要借助网络抓包工具 Wireshark 来帮我们分析设备行为，定位问题。下面我们通过一个简单案例，给大家讲解使用 Wireshark 分析设备与阿里云 IoT 物联网平台通信的过程。

二、准备工作

1. 创建设备

在阿里云 IoT 物联网平台创建产品，并注册设备，获取三元组。



2. 设备模拟程序

我们在电脑上用 Nodejs 编写 device 模拟程序，建立连接，订阅，发布，断开连接。

```
/**
 * node aliyun-iot-device.js
 */
const mqtt = require('aliyun-iot-mqtt');
//设备身份三元组+区域
const options = {
  "productKey": "设备 PK",
  "deviceName": "设备 DN",
  "deviceSecret": "设备 Secret",
  "regionId": "cn-shanghai"
};

//1.建立连接
const client = mqtt.getAliyunIotMqttClient(options);
//2.订阅主题
setTimeout(function() {
  client.subscribe(`/${options.productKey}/${options.deviceName}/user/get`)
}, 3 * 1000);
//3.发布消息
setTimeout(function() {
  client.publish(`/${options.productKey}/${options.deviceName}/user/update`, getPostData(),
    {qos:1});
}, 5 * 1000);
//4.关闭连接
setTimeout(function() {
  client.end();
}, 8 * 1000);

function getPostData() {
  const payloadJson = {
    temperature: Math.floor((Math.random() * 20) + 10),
    humidity: Math.floor((Math.random() * 20) + 10)
  }
}
```

```
console.log("payloadJson " + JSON.stringify(payloadJson))
return JSON.stringify(payloadJson);
}
```

3. 使用 Wireshark 抓取网络包

IoT 物联网平台使用 MQTT 协议通信，我们只需要配置如下规则即可：

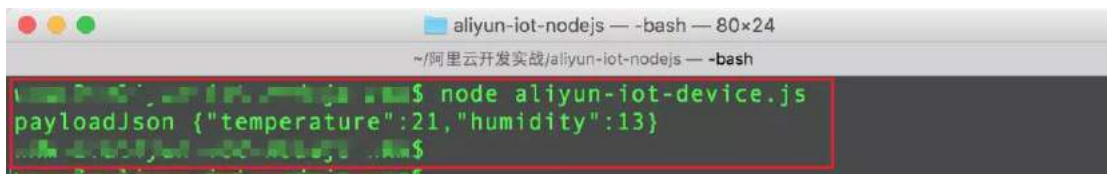
tcp and port 1883

Capture

...using this filter: tcp and port 1883

Wi-Fi: en0

4. 启动模拟程序



三、网络抓包分析

随着模拟脚本执行完毕，完整的 MQTT 网络交互过程都在 wireshark 捕捉到了。

为了方便我把设备 ip 标记成了 device，本次连接的阿里云 IoT 的 IP 保持不变。

1. TCP 的三次握手

A screenshot of the Wireshark network protocol analyzer. The packet list pane shows 11 captured packets. The first three packets (1, 2, 3) represent the TCP three-way handshake. Packet 1 is a SYN from device to 139.196.135.135. Packet 2 is a SYN, ACK from 139.196.135.135 to device. Packet 3 is an ACK from device to 139.196.135.135. The packet details pane for packet 3 shows the 'TCP' section with 'Seq=1', 'Ack=167', 'Win=1536', and 'Len=0'.

No.	Time	Source	Destination	Protocol	Length	Info
1	2019-08-01 09:41:30.468410	device	139.196.135.135	TCP	78	56150 → 1883 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2	2019-08-01 09:41:30.490440	139.196.135.135	device	TCP	78	1883 → 56150 [SYN, ACK] Seq=0 Ack=1 Win=12960 Len=0
3	2019-08-01 09:41:30.490524	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	2019-08-01 09:41:30.491449	device	139.196.135.135	MQTT	220	Connect Command
5	2019-08-01 09:41:30.503962	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=1 Ack=167 Win=1536 Len=0
6	2019-08-01 09:41:30.543583	139.196.135.135	device	MQTT	60	Connect Ack
7	2019-08-01 09:41:30.543699	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=167 Ack=5 Win=65535 Len=0
8	2019-08-01 09:41:33.418227	device	139.196.135.135	MQTT	103	Subscribe Request
9	2019-08-01 09:41:33.430329	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=5 Ack=216 Win=1536 Len=0
10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	Subscribe Ack
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=18 Win=65535 Len=0

下图展示了 device 向 IoT 物联网平台订阅 topic 的过程。这里 device 主动订阅了一个 Topic，见红框部分。

4	2019-08-01 09:41:30.491449	device	139.196.135.135	MQTT	220	Connect Command
5	2019-08-01 09:41:30.503962	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=1 Ack=167
6	2019-08-01 09:41:30.543583	139.196.135.135	device	MQTT	60	Connect Ack
7	2019-08-01 09:41:30.543699	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=167 Ack=5
8	2019-08-01 09:41:33.418227	device	139.196.135.135	MQTT	103	Subscribe Request
9	2019-08-01 09:41:33.430329	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=5 Ack=216
10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	Subscribe Ack
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=14
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	Publish Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	Publish Ack
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=14
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	Disconnect Req
16	2019-08-01 09:41:35.454673	device	139.196.135.135	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302
19	2019-08-01 09:41:35.469565	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=14

MQ Telemetry Transport Protocol, Subscribe Request	
Header Flags: 0x02 (Subscribe Request)	
Msg Len: 47	
Message Identifier: 48396	
Topic Length: 42	
Topic: /alvYwzHjT6p/2wE56xa1NcVpdgkDrjtj/user/get	
Requested QoS: At most once delivery (Fire and Forget) (0)	

下图展示了 IoT 物联网平台响应 device 订阅的行为。

6	2019-08-01 09:41:30.543583	139.196.135.135	device	MQTT	60	Connect Ack
7	2019-08-01 09:41:30.543699	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=1 Ack=167
8	2019-08-01 09:41:33.418227	device	139.196.135.135	MQTT	103	Subscribe Request
9	2019-08-01 09:41:33.430329	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=5 Ack=216
10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	Subscribe Ack
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=14
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	Publish Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	Publish Ack
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=14
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	Disconnect Req
16	2019-08-01 09:41:35.454673	device	139.196.135.135	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302

3. MQTT 的 PUBLISH 行为

下图展示了 device 向 IoT 物联网平台 PUBLISH 一条 QoS=1 的消息。在报文信息里，我们可以看到消息对应的 Topic 和 Payload。

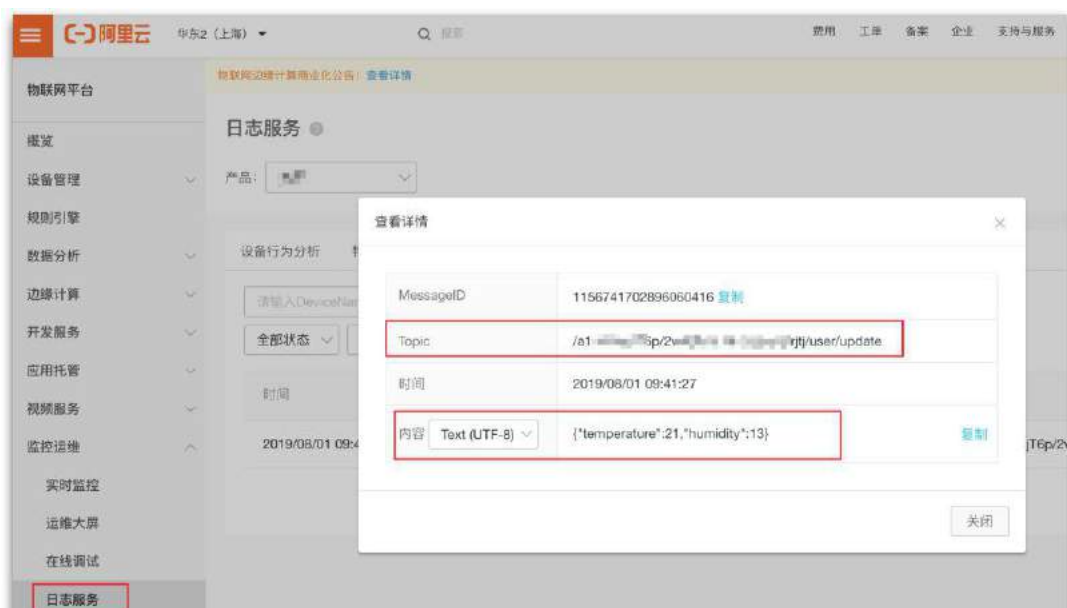
10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	Subscribe Ack
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=14
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	Publish Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	Publish Ack
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=14
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	Disconnect Req
16	2019-08-01 09:41:35.454673	device	139.196.135.135	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302
19	2019-08-01 09:41:35.469565	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=14

MQ Telemetry Transport Protocol, Publish Message	
Header Flags: 0x32 (Publish Message)	
0011 = Message Type: Publish Message (3)	
.... 0... = DUP Flag: Not set	
.... 01.. = QoS Level: At least once delivery (Acknowledged deliver) (1)	
.... ...0 = Retain: Not set	
Msg Len: 81	
Topic Length: 45	
Topic: /alvYwzHjT6p/2wE56xa1NcVpdgkDrjtj/user/update	
Message Identifier: 48397	
Message: {"temperature":21,"humidity":13}	

由于是 QoS=1 消息，IoT 物联网平台会回复一条 PUBACK 给 device。

10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	SUBSCRIBE ACK
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	PUBLISH Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	PUBLISH ACK
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	DISCONNECT Req
16	2019-08-01 09:41:35.454673	device	139.196.135.135	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14 Win=65535 Len=0
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301 Win=1536 Len=0
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302 Win=1536 Len=0

在 IoT 物联网控制台的日志服务也能看到这条消息日志。



4. MQTT 的 DISCONNECT 行为

下图展示了 device 主动发起 DISCONNECT 命令，断开 MQTT 连接通道。

9	2019-08-01 09:41:33.430329	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=5 Ack=216 Win=1536 Len=0
10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	SUBSCRIBE ACK
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=10 Win=65535 Len=0
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	PUBLISH Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	PUBLISH ACK
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=14 Win=65535 Len=0
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	DISCONNECT Req
16	2019-08-01 09:41:35.454673	device	139.196.135.135	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14 Win=65535 Len=0
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301 Win=1536 Len=0
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302 Win=1536 Len=0
19	2019-08-01 09:41:35.469633	device	139.196.135.135	TCP	54	[TCP Out-Of-Order] 56150 → 1883 [FIN, ACK] Seq=301 Ack=14 Win=65535 Len=0

MQ Telemetry Transport Protocol, Disconnect Req	
Header Flags: 0xe0 (Disconnect Req)	
1110 Message Type: Disconnect Req (14)	
.... 0000 = Reserved: 0	
Msg Len: 0	

5. TCP 的四次挥手

10	2019-08-01 09:41:33.443413	139.196.135.135	device	MQTT	60	Subscribe Ack
11	2019-08-01 09:41:33.443543	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=216 Ack=10 Win=65535 Len=0
12	2019-08-01 09:41:35.421398	device	139.196.135.135	MQTT	137	Publish Message
13	2019-08-01 09:41:35.440333	139.196.135.135	device	MQTT	60	Publish Ack
14	2019-08-01 09:41:35.440470	device	139.196.135.135	TCP	54	56150 → 1883 [ACK] Seq=299 Ack=14 Win=65535 Len=0
15	2019-08-01 09:41:35.453121	device	139.196.135.135	MQTT	56	Disconnect Req
16	2019-08-01 09:41:35.454673	139.196.135.135	device	TCP	54	56150 → 1883 [FIN, ACK] Seq=301 Ack=14 Win=65535
17	2019-08-01 09:41:35.469561	139.196.135.135	device	TCP	60	1883 → 56150 [FIN, ACK] Seq=14 Ack=301 Win=1536 Len=0
18	2019-08-01 09:41:35.469565	139.196.135.135	device	TCP	60	1883 → 56150 [ACK] Seq=15 Ack=302 Win=1536 Len=0
19	2019-08-01 09:41:35.469633	device	139.196.135.135	TCP	54	[TCP Out-Of-Order] 56150 → 1883 [FIN, ACK] Seq=30

Frame 16: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0

在 IoT 物联网控制台的日志服务也能看到完整的设备上下线日志，如下。

物联网平台		
日志服务		
产品: [选择]		
设备行为分析 物模型数据分析 上行消息分析 下行消息分析		
请输入DeviceName 请输入内容关键字 1小时		
时间	DeviceName	内容 (全部)
2019/08/01 09:27:56.180	2wE...	offline, lastActiveTime=1564622876172 Device disconnect
2019/08/01 09:27:51.177	2wE...	online clientIp=...

四、结束语

至此，我们掌握了使用 Wireshark 抓包工具分析设备和阿里云 IoT 物联网平台网络通信的基本技能，希望对大家 IoT 物联网开发有所帮助。

五、附录

TCP 层的几个标识

SYN	表示建立连接
FIN	表示关闭连接
ACK	表示响应
PSH	表示有 DATA 数据传输
RST	表示连接重置

IoT 平台日志服务详解

作者 | 苏堤嘉木

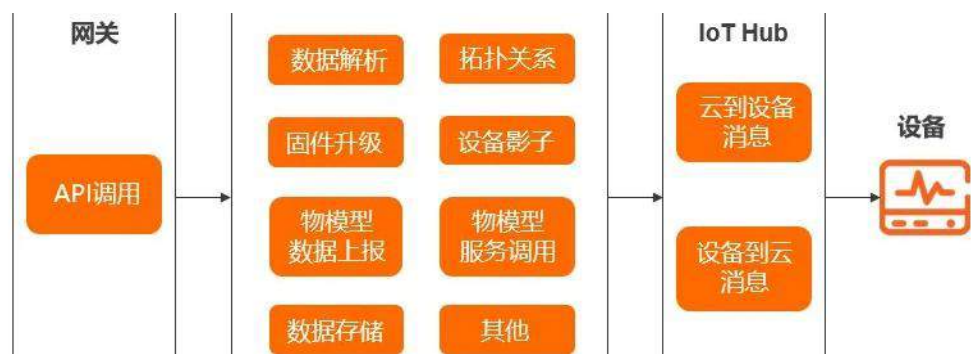
近日，阿里云 IoT 物联网平台控制台日志服务改版升级，我们发现对开发者更友好了，提升了查询日志的效率，还增加了设备运行日志的查询功能。

上行消息的日志业务类型如下：



- 设备上报数据到 IoT Hub，打印设备到云消息日志，包含消息的 Topic。
- 对于数据处理的不同业务模块，打印本模块的日志。
- 如果消息对外通过规则引擎和服务端订阅（AMQP/MNS）发送给客户，规则引擎、服务端订阅将打印本模块的日志。

下行消息的日志业务类型如下



- 用户通过 API 调用产生消息，打印 API 调用日志，包含 API 名称。
- 对于数据处理的不同业务模块，打印本模块的日志。
- 如果有消息发送到设备侧，IoT Hub 打印云到设备消息日志，包含消息的 Topic。

一、设备上线/离线日志

链路：设备→IoT 平台(上线)、设备→IoT 平台(离线)

物联网平台 | 基础运维 | 日志服务

日志服务

产品: 温度探针

云端运行日志 | 设备本地日志

请输入DeviceName | 请输入TraceID | 请输入MessageID

请输入内容关键字 | 全部状态 | 1小时

搜索 | 重置

时间	TraceID	MessageID	DeviceName	业务类型(设备行为)	操作	内容	状态
17:12:43.894	857531068cc	-	hz9527	设备行为	offline	["Content":{"offline, lastActiveTime=15888...}	200
17:12:38.008	703381068cc	-	hz9527	设备行为	online	["Content":{"online, clientIp=47.10.139"}]	200
16:38:32.845	31195281e92	-	hz9527	设备行为	offline	["Content":{"offline, lastActiveTime=15888...}	200
16:35:51.098	46209d1a09	-	hz9527	设备行为	online	["Content":{"online, clientIp=47.10.139"}]	200

二、物模型-属性上报的日志

消息链路：设备→IoT 平台→物模型校验→物模型数据存储

物联网平台 | 基础运维 | 日志服务

日志服务

产品: 温度探针

云端运行日志 | 设备本地日志

请输入DeviceName | 请输入TraceID | 请输入MessageID

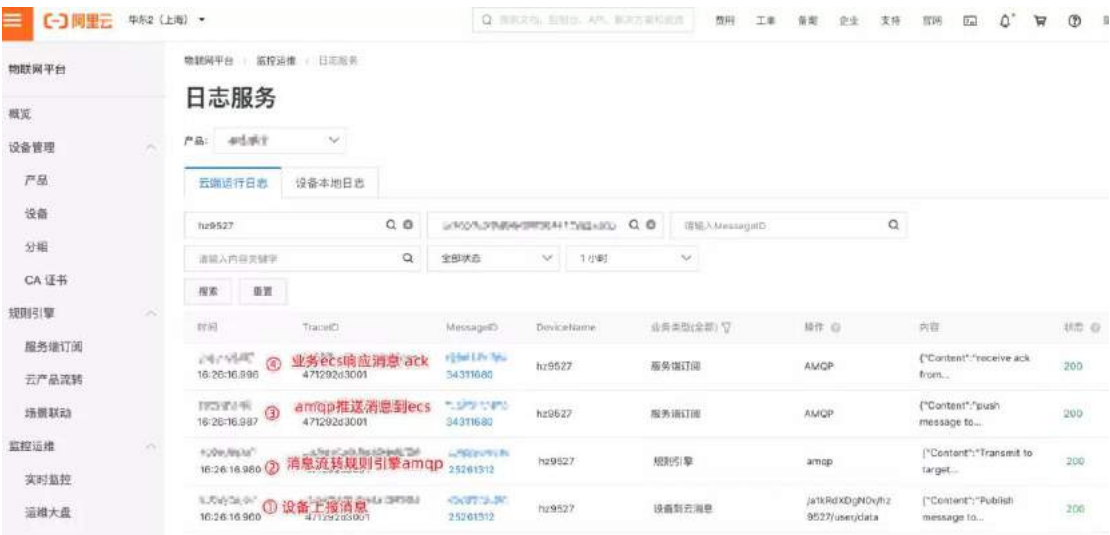
请输入内容关键字 | 全部状态 | 1小时

搜索 | 重置

时间	TraceID	MessageID	DeviceName	业务类型(设备行为)	操作	内容	状态
17:12:43.108	606934068cc	959457280	hz9527	物模型上报	thing event property post	-	200
17:12:43.108	606934068cc	-	hz9527	数据存储	-	-	200
17:12:43.108	606934068cc	-	hz9527	物模型上报	Check	-	200
17:12:43.074	959457280	-	hz9527	设备到云消息	/sys/thing/property/zh20627/thing...	["Content":{"Publish message to...	200
17:12:38.098	703381068cc	-	hz9527	设备行为	online	["Content":{"online, clientIp=47.10.139"}]	200

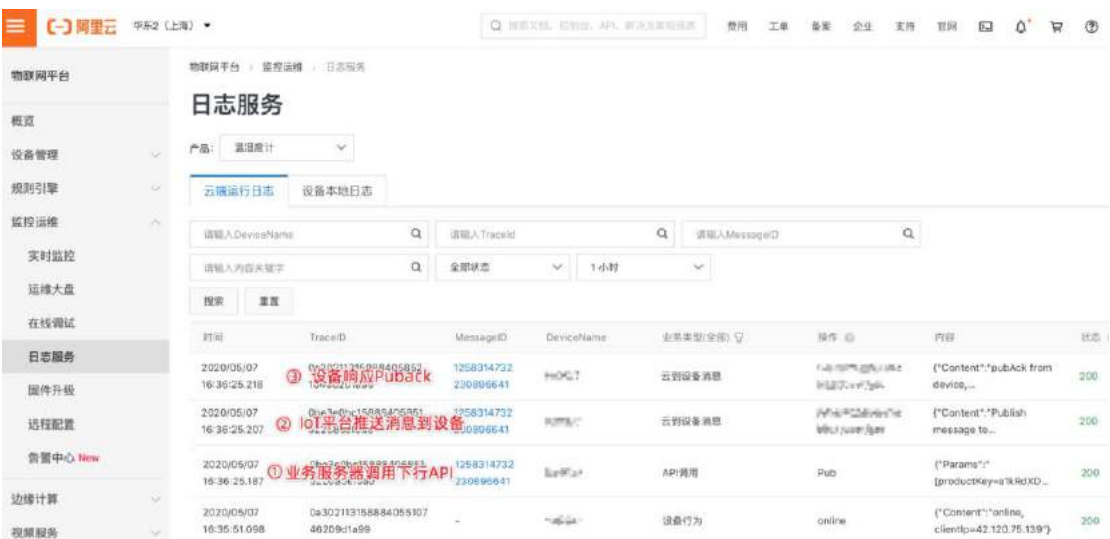
三、自定义消息规则引擎流转

消息链路: 设备→IoT 平台→规则引擎→服务端订阅 AMQP→业务服务器 ECS→服务端订阅 AMQP(ACK 响应)



四、下行控制指令日志

消息链路: 业务服务器 ECS(Pub API)→IoT 平台(Publish)→设备→IoT 平台(PubAck 响应)



五、私有协议脚本解析处理日志

消息链路：设备→IoT 平台→自定义协议脚本解析→规则引擎→服务端订阅 AMQP

The screenshot displays the 'Log Service' (日志服务) interface in the IoT Cloud console. The left sidebar contains navigation options like 'Overview', 'Device Management', 'Rule Engine', 'Monitoring', 'Real-time Monitoring', 'Data Warehouse', 'Online Debug', 'Log Service' (highlighted), 'Firmware Upgrade', 'Remote Configuration', 'Alert Center', 'Edge Computing', 'Video Service', and 'IoT Studio'. The main area shows a table of log entries for device 'h29527'.

时间	TraceID	MessageID	DeviceName	业务类型(全部)	操作	内容	状态
17-46:57:428	08450304a20	107150593	h29527	服务端订阅	AMQP	["Content": "receive ack from..."]	200
17-46:57:347	08450304a20	107150593	h29527	服务端订阅	AMQP	["Content": "push message to..."]	200
17-46:57:337	08450304a20	053995520	h29527	规则引擎	amqp	["Content": "Transmit to target..."]	200
17-46:57:503	08450304a20	-	h29527	数据解析	/api/v1/h29527/user/data	["ResultData": {"upOriginalData": "7b2..."}]	200
17-46:57:317	08450304a20	053995520	h29527	设备到云消息	/api/v1/h29527/user/data	["Content": "Publish message to..."]	200
17-46:51:731	11104764a20	-	h29527	设备行为	online	["Content": "online, clientip=47.159"]	200

IoT 设备日志采集利器服务

作者 | 苏堤嘉木



当我们的物联网设备分布在全球各地运行时，为了排查特定环境下的业务问题，常常有**需要获取设备端运行日志**的诉求。如果我们派一位工作人员出差去设备现场，通过数据线采集设备运行日志，往往成本很高，预算不允许。此时借助 IoT 物联网平台的**设备本地日志采集能力**，我们在电脑前喝着咖啡，动动手指，就能轻松获取远端设备日志，在控制台进行查询和故障分析，免去差旅之苦。

一、设备日志采集方案

设备日志采集基于 MQTT 通信的 Pub 通信，具体约定如下：

Topic:

```
/sys/${productKey}/${deviceName}/thing/log/post
```

Payload:

```
{
  "id": 12345,
  "version": "1.0",
  "params": [{
```

```

    "utcTime": "2020-03-06T15:15:27.464+0800",
    "logLevel": "ERROR",
    "module": "wifi",
    "code": "",
    "traceContext": "ai1289sa8dss",
    "logContent": "network error"
  }],
  "method": "thing.log.post"
}

```

Payload 结构体参数说明：

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围 <code>0 ~ 4294967295</code> ，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为 <code>1.0</code> 。
params	List	请求业务参数。数组元素最多40个。
utcTime	String	日志的采集时间，为设备本地UTC时间，包含时区信息，以毫秒计，格式为“yyyy-MM-dd'T'HH:mm:ss.SSSZ”。可上报其它字符串格式，但不利于问题排查，不推荐使用。
logLevel	String	日志级别。可以使用默认日志级别，也可以自定义日志级别。默认日志级别从高到低为： - FATAL - ERROR - WARN - INFO - DEBUG
module	String	模块名称。 - 当设备端使用Android SDK时，模块名称为ALK-LK。 - 当设备端使用C SDK时，需自定义模块名称。
code	String	结果状态码。 - 当设备端使用Android SDK时，请参见Android SDK错误码 - 当设备端使用C SDK时，请参见C SDK状态码。
traceContext	String	可选参数，上下文跟踪内容，设备端使用Alink协议消息的 id ，APP端使用 TraceId （追踪ID）。
logContent	String	日志内容详情。
method	String	请求方法，取值 <code>thing.log.post</code> 。

二、设备上报日志实战

当设备端运行出现异常后,应用程序会及时保存日志到本地存储,再次建立云端连接后,即可上报异常日志到云端。

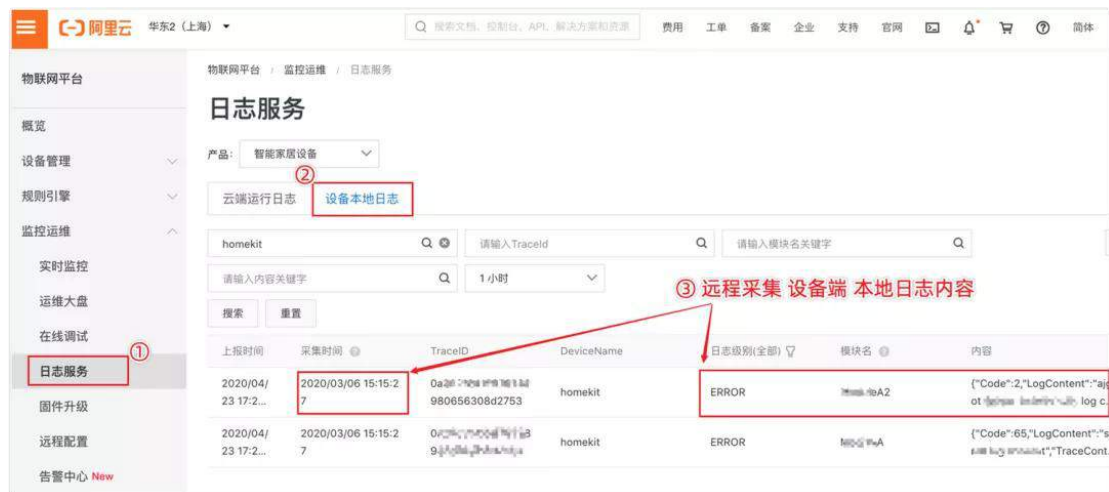
```
const logTopic = `sys/${options.productKey}/${options.deviceName}/thing/log/post`;
// 异常恢复后,上报设备日志
client.publish(logTopic, getLogData());

// 设备本地日志
function getLogData() {
  const payloadJson = {
    id: Date.now(),
    params: [{
      "utcTime": "2020-03-06T15:15:27.464+0800",
      "logLevel": "ERROR",
      "module": "os",
      "code": "65",
      "traceContext": "123456",
      "logContent": "this error log from device xxx"
    },
    {
      "utcTime": "2020-03-06T15:15:27.664+0800",
      "logLevel": "ERROR",
      "module": "wifi",
      "code": "2",
      "traceContext": "123456536",
      "logContent": "network error,xxx"
    }
  ],
  method: "thing.log.post"
}

console.log("payloadJson " + JSON.stringify(payloadJson))
return JSON.stringify(payloadJson);
}
```

三、控制台查看设备日志

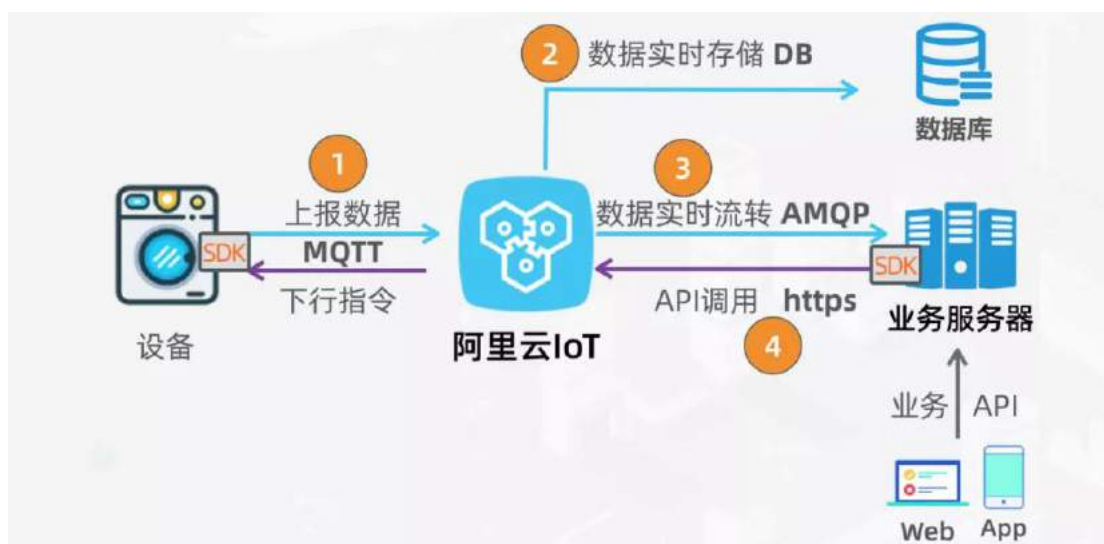
设备上报日志后，我们登录控制台，在日志服务里选择产品，点击设备本地日志的 Tab，输入设备名 `deviceName`，即可查看设备上报的异常信息。同时，也可以根据日志级别，快速筛选日志。



全链路开发实战

IoT 平台端到端开发实践

作者 | 苏堤嘉木

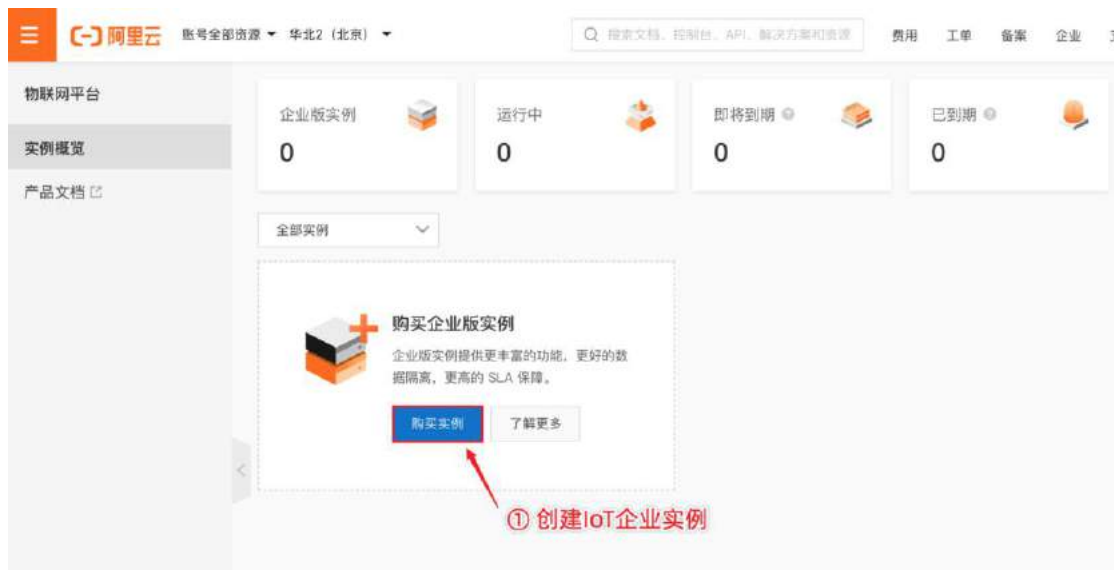


通过本次 IoT 开发实战教程你将学会一下技能：

1. 设备通过 MQTT 协议与您在阿里云上购买的 IoT 企业实例**建立双向连接**，设备上报**采集的数据**，监听云端下达的指令；
2. 通过规则引擎配置把上报的数据实时**存储到指定数据库**，无需编写代码；
3. 通过规则引擎配置把上报的数据实时**流转到业务服务器**，需要使用 AMQP 协议 SDK；
4. 业务服务器调用 IoT 平台的 API，**下达控制指令**到设备端。

一、创建 IoT 企业实例

首先，我们登录物联网平台控制台(<https://iot.console.aliyun.com>)，点击**购买实例**来创建一个企业实例。



然后, 在购买页面, 根据实际业务需求, 选择地域、实例类型、设备数量、消息上下行 TPS、规则引擎 TPS 等参数, 点击立即购买, 付费成功后, 即可看到企业实例创建中。



稍等几分钟后, 企业实例创建完成。进入企业实例, 我们可以看到当前规格参数, 设备接入点信息, AMQP 订阅接入点信息, 云端 API 调用接入点信息。如下图:

The screenshot displays the Alibaba Cloud IoT Platform console. The top navigation bar includes the Alibaba Cloud logo, account information (华北2 (北京)), and a search bar. The left sidebar shows the navigation menu with options like 实例详情, 设备管理, 规则引擎, 监控运维, and 文档与工具.

The main content area is titled "IoT开发实战" (IoT Development Practice). It shows the instance ID (iot-cn-*) and the instance name (IoT开发实战). The instance information table lists various configurations:

实例 ID	iot-cn-*	实例类型	基础型
已添加设备	1/2000	消息上下行 TPS 峰值	200 条/秒
规则引擎 TPS 峰值	200 条/秒	Link WAN 服务	未启用
ID² 设备安全认证服务	未启用	时序存储空间	0GB / 40GB
时序存储时间线数	0万 / 50万	时序存储写入 TPS 规格	10000 条/秒
时序存储用户名	root	时序存储密码	***** 查看
时序存储时效	已关闭	时序存储连接地址	ts-*.cn-beijing.aliyuncs.com:3306
到期时间	2024-12-31 23:59:59		

Below the instance information, there are three panels for "公网终端节点 (Endpoint)":

- 设备接入点 (③):** Shows MQTT device authentication (iot-auth-global.aliyuncs.com), MQTT device connection (iot-cn-*.cn-beijing.aliyuncs.com), and VPC network endpoint (iot-cn-*.cn-beijing.aliyuncs.com).
- AMQP订阅接入点 (④):** Shows AMQP service subscription (iot-cn-*.cn-beijing.aliyuncs.com) and VPC network endpoint (iot-cn-*.cn-beijing.aliyuncs.com).
- 云端API调用接入点 (⑤):** Shows cloud API invocation (iot-cn-beijing.aliyuncs.com) and VPC network endpoint (iot-vpc-cn-beijing.aliyuncs.com).

二、创建产品和注册设备

在企业实例的**设备管理**页面，我们需要先创建一个产品**家庭温控器**，数据通信以JSON 格式，认证方式为**设备秘钥**。



在产品的功能定义页面，我们添加**温度**和**湿度**两个属性，具体细节如下图：



最后，我们在**设备管理**页面，基于**家庭温控器**产品，注册一个物理设备，并获取设备身份认证的三元组。如下图：



三、设备开发和数据上报

获取设备身份三元组后，即可通过 MQTT 协议接入到我们开通的企业实例。设备端应用程序逻辑如下图：

```
const mqtt = require('aliyun-iot-mqtt'); // ① MQTT库
// 1. 设备身份信息
var options = {
  productKey: "g0x489234", // ② 设备身份三元组
  deviceName: "hxt93489234",
  deviceSecret: "de44f933195e2b641c448dc26e3008567",
  host: "iot-cn-hz11561-1011.mqtt.iothub.aliyuncs.com" // ③ 实例化接入点
};

// 2. 建立MQTT连接
const client = mqtt.getAliyunIotMqttClient(options); // ④ 建立MQTT长连接

client.subscribe(`${options.productKey}/${options.deviceName}/user/get`)
client.on('message', function(topic, message) {
  console.log("topic " + topic) // ⑤ 订阅云端指令 Topic
  console.log("message " + message)
})

setInterval(function() {
  // 3. 上报温湿度数据 // ⑥ 定时上报采集的数据
  client.publish(`/sys/${options.productKey}/${options.deviceName}/thing/event/property/post`,
    JSON.stringify({
      id: Date.now(),
      version: "1.0",
      params: {
        temperature: Math.floor((Math.random() * 20) + 10),
        humidity: Math.floor((Math.random() * 20) + 10)
      }
    }), 5 * 1000);

  function getPostData() {
    const payloadJson = {
      id: Date.now(),
      version: "1.0",
      params: {
        temperature: Math.floor((Math.random() * 20) + 10),
        humidity: Math.floor((Math.random() * 20) + 10)
      },
      method: "thing.event.property.post"
    }
    console.log("payloadJson " + JSON.stringify(payloadJson))
    return JSON.stringify(payloadJson);
  }
}
}
```

完整的 Nodejs 示例代码如下：

```
const mqtt = require('aliyun-iot-mqtt');
// 1. 设备身份信息
var options = {
  productKey: "产品 productKey",
  deviceName: "设备 deviceName",
  deviceSecret: "设备 deviceSecret",
  host: "实例化 MQTT 接入点"
};

// 2. 建立 MQTT 连接
const client = mqtt.getAliyunIotMqttClient(options);

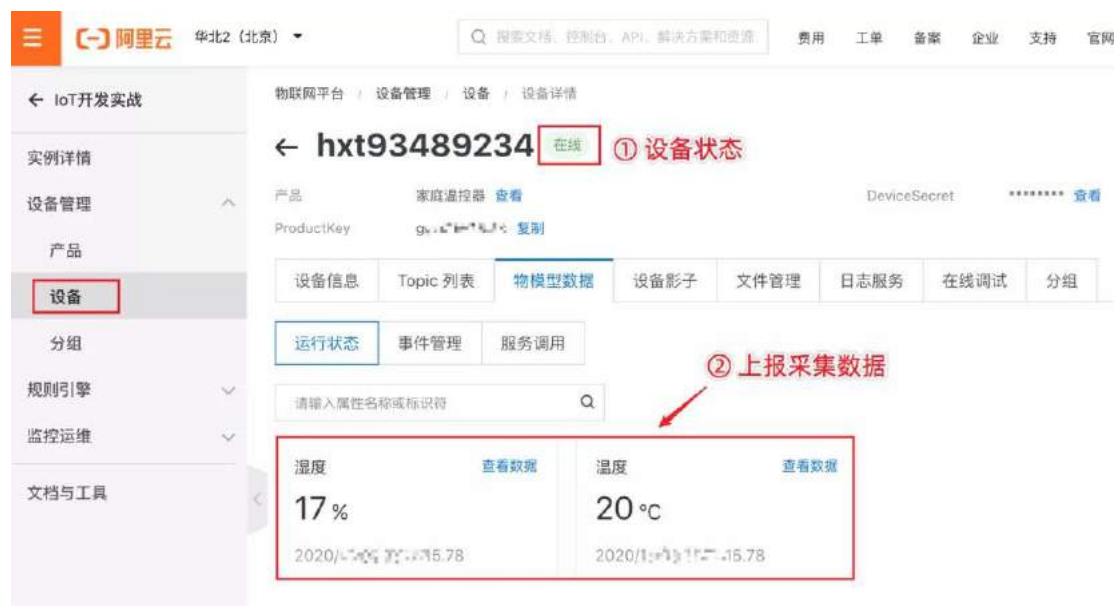
client.subscribe(`${options.productKey}/${options.deviceName}/user/get`)
client.on('message', function(topic, message) {
  console.log("topic " + topic)
  console.log("message " + message)
})

setInterval(function() {
  // 3. 上报温湿度数据
  client.publish(`/sys/${options.productKey}/${options.deviceName}/thing/event/property/post`, getPostData(), { qos: 0 });
}, 5 * 1000);

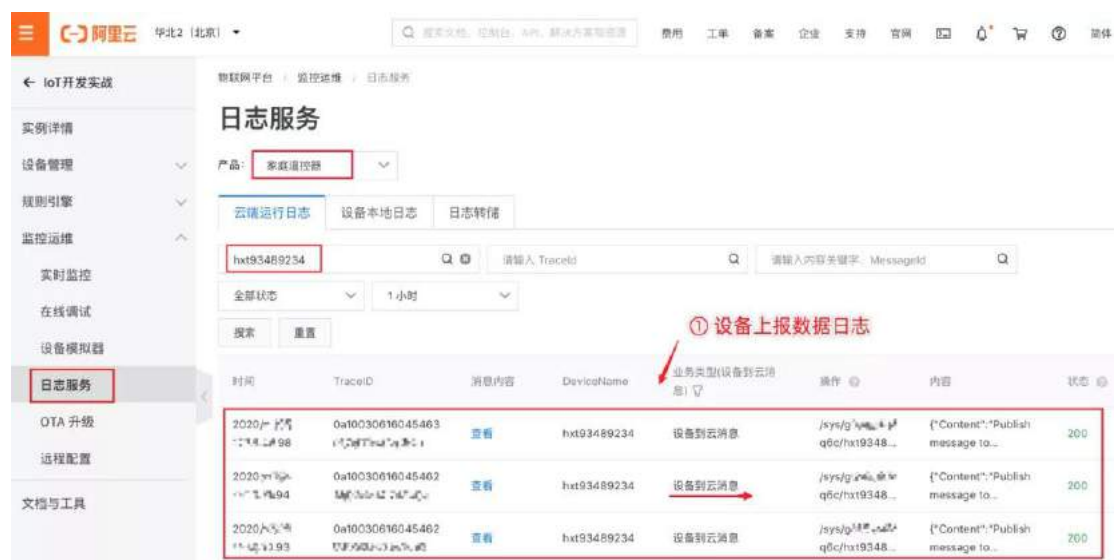
function getPostData() {
  const payloadJson = {
    id: Date.now(),
    version: "1.0",
    params: {
      temperature: Math.floor((Math.random() * 20) + 10),
      humidity: Math.floor((Math.random() * 20) + 10)
    },
    method: "thing.event.property.post"
  }

  console.log("payloadJson " + JSON.stringify(payloadJson))
  return JSON.stringify(payloadJson);
}
```

启动模拟脚本后，我们看到设备状态为**在线**，**物模型数据**中可以看到最新上报的**温度**和**湿度值**。

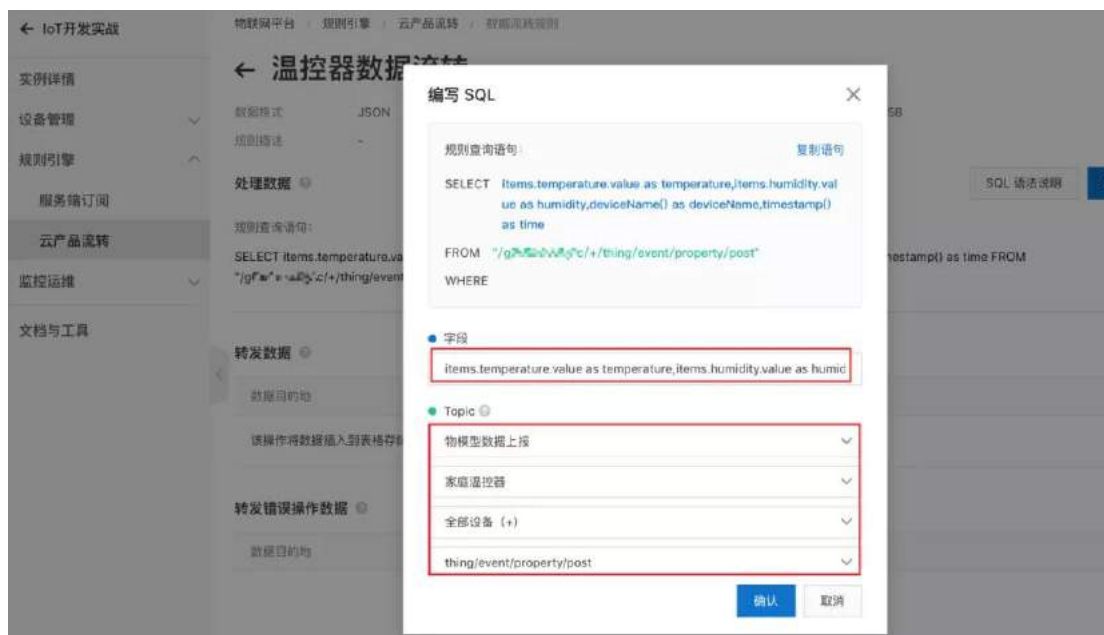


在**监控运维**的**日志服务**里，也可以看到设备上报数据的日志。如下图：

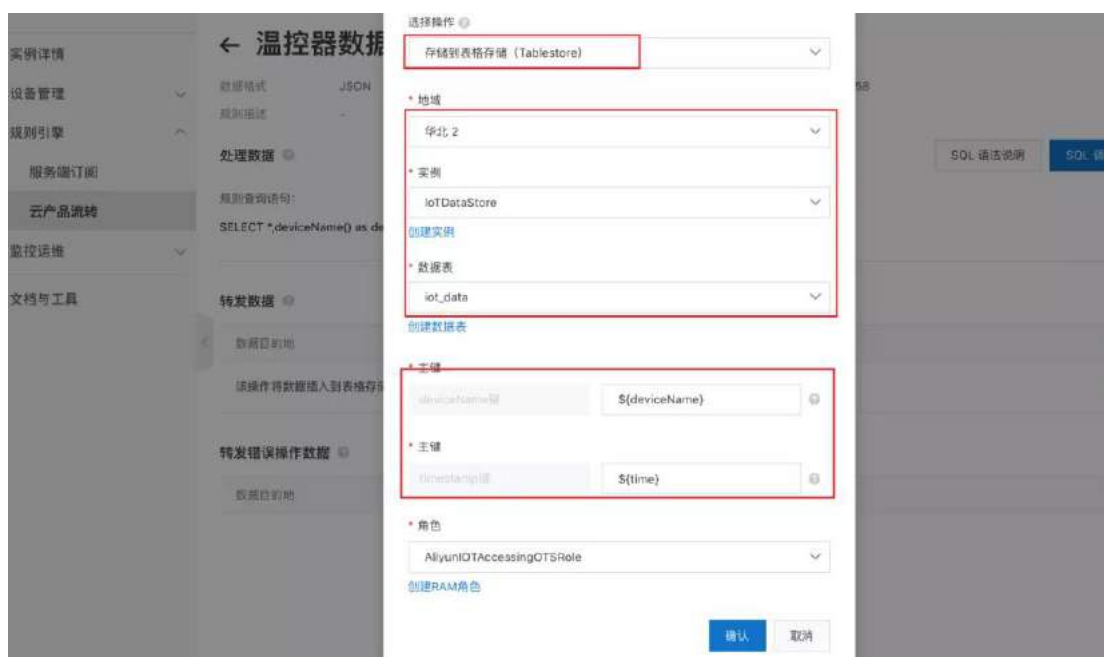


四、数据存储到 DataBase

首先，我们创建一个表格存储实例 `IoTDataStore`，建立一张数据表 `iot_data`，以 `deviceName` 和 `timestamp` 为主键。如下图：



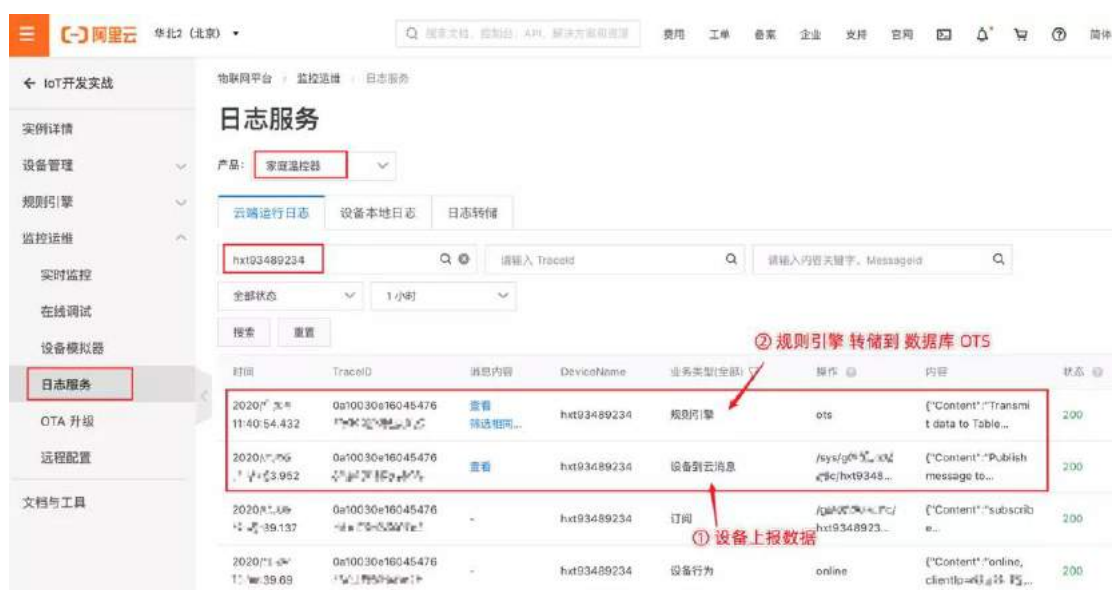
数据流转表格存储编辑界面：



当设备有数据上报后,我们就可以在表格存储的 iot_data 表中实时看到存储的数据了。
如下图:



在企业实例的日志服务中，我们可以查看到完整的流转日志，协助我们排查数据链路异常。如下图：



五、资源受限类设备

IoT 场景中有些数据需要业务系统实时处理，这时我们可以通过服务端订阅 AMQP 方式，实时接收设备上报的数据。

首先，我们要在企业实例的服务端订阅中，创建一个新的消费组，用来接收特定类型的消息。如下图：



然后，我们在**云产品流转**中创建规则引擎，编写数据处理 SQL，配置流转目的地为上一步创建的**服务端订阅消费组**。



最后，我们在业务服务器编写程序，使用阿里云账号的 AccessKey 与 IoT 企业实例建立 AMQP 连接，参考代码如下：

```
public static void main(String[] args) throws Exception {  
    //参数说明  
    String accessKey = "子账号 accessKey";  
    String accessSecret = "子账号 accessSecret";  
    String consumerGroupId = "消费组 Id";  
    String IoTInstanceId = "企业实例 Id";  
}
```

```

long timeStamp = System.currentTimeMillis();
//签名方法: 支持 hmacmd5, hmacsha1 和 hmacsha256
String signMethod = "hmacsha1";

String clientId = "ecs_"+System.currentTimeMillis();
String userName = clientId + "|authMode=aksign"
+ ",signMethod=" + signMethod
+ ",timestamp=" + timeStamp
+ ",authId=" + accessKey
+ ",iotInstanceId=" + iotInstanceId
+ ",consumerGroupId=" + consumerGroupId
+ "|";

//password 组装
String signContent = "authId=" + accessKey + "&timestamp=" + timeStamp;
String password = doSign(signContent,accessSecret, signMethod);
//按照 qpid-jms 的规范, 组装连接 URL。
String connectionUrl = "failover:(amqps://"+iotInstanceId+".amqp.iothub.aliyuncs.com:56
71?amqp.idleTimeout=80000)"
+ "?failover.reconnectDelay=30";
Hashtable<String, String> hashtable = new Hashtable<>();
hashtable.put("connectionfactory.SBCF",connectionUrl);
hashtable.put("queue.QUEUE", "default");
hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");

Context context = new InitialContext(hashtable);
ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
Destination queue = (Destination)context.lookup("QUEUE");
// Create Connection
Connection connection = cf.createConnection(userName, password);
((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
// Create Session
// Session.CLIENT_ACKNOWLEDGE: 收到消息后, 需要手动调用 message.acknowledge()
// Session.AUTO_ACKNOWLEDGE: SDK 自动 ACK (推荐)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
// Create Receiver Link
MessageConsumer consumer = session.createConsumer(queue);
consumer.setMessageListener(messageListener);
}

```

在 AMQP 的回调中处理收到的业务数据，参考代码如下：

```
private static MessageListener messageListener = new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            //如果要对收到的消息做耗时的处理，请异步处理，确保这里不要有耗时逻辑。
            byte[] body = message.getBody(byte[].class);
            String content = new String(body);
            String topic = message.getStringProperty("topic");
            String messageId = message.getStringProperty("messageId");
            String tag = message.getStringProperty("tag");
            logger.info("receive message"
                + ",\n topic = " + topic
                + ",\n messageId = " + messageId
                + ",\n tag = " + tag
                + ",\n content = " + content
                + "\n");
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
};
```

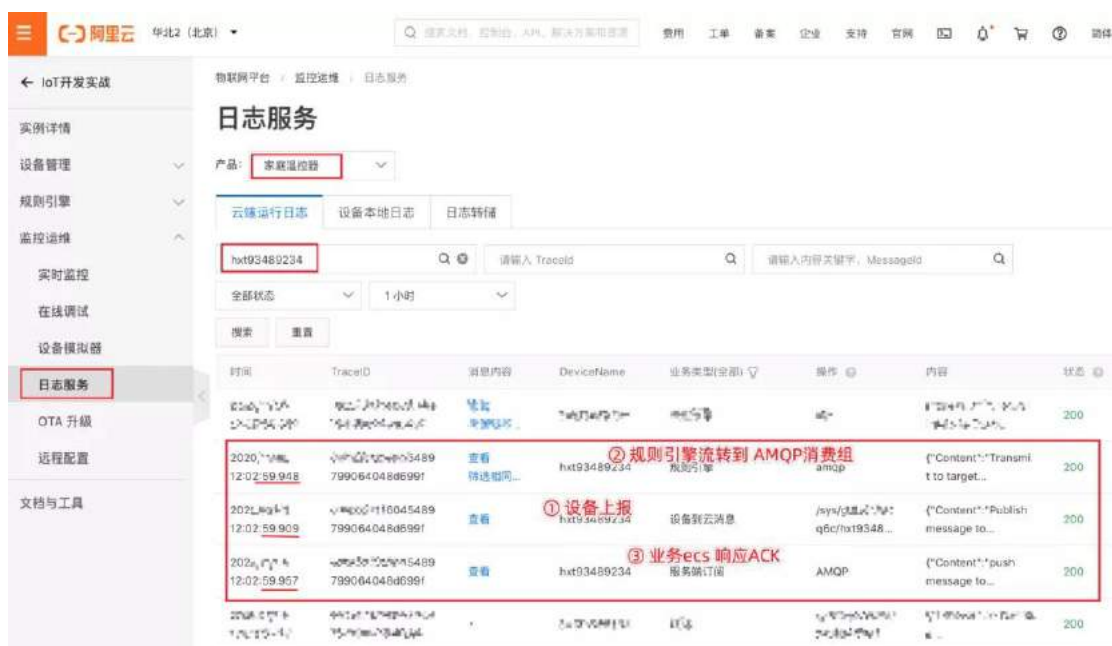
启动业务服务器后，我们看到不断有设备数据流转过来，如下图：



在企业实例的控制台，服务端订阅中，我们也可以看到消费组的运行情况，包括消费速率，消息堆积量，消费客户端列表，如下图：



在企业实例的控制台，日志服务中，我们可以看到完整的消息流转日志，如下图：



六、云端下达控制指令

业务系统通过调用 IoT 物联网平台提供的 HTTPS API 可以给指定设备下发控制指令，调用过程如下：

```

const co = require('co');
const RPCClient = require('@alicloud/pop-core').RPCClient; ① 云端SDK

const options = {
  accessKey: "LTAI4C89wFm8LjwvQ11j0g8Ks",
  accessKeySecret: "Mn%&@qg*7y277adH17f7p0u3e2" ② 账号身份 AccessKey
};

//1.初始化 API Client
const client = new RPCClient({
  accessKeyId: options.accessKey,
  secretAccessKey: options.accessKeySecret,
  endpoint: 'https://iot.cn-beijing.aliyuncs.com', ③ 构造Client
  apiVersion: '2018-01-20',
});

// 指令内容
const payload = {
  washingMode: 2,
  washingTime: 30 ④ 控制指令内容
};

//2.构建Pub API 请求
const params = {
  TopicFullName: "/g00k4EjPP46c/hxt93489234/user/get",
  MessageContent: new Buffer(JSON.stringify(payload)).toString("base64"),
  ProductKey: "g00k4EjPP46c",
  IotInstanceId: "iot-cn-n151mgx1461",
  Qos: 1
};

co(function*() {
  //3.发起Pub API调用 ⑤ 发起云端 API调用
  try {
    const response = yield client.request('Pub', params);
    console.log("Pub SUCCESS =====>", JSON.stringify(response));
  } catch (err) {
    console.log("Pub ERROR =====>", JSON.stringify(err));
  }
});

```

Pub API 调用的参考代码:

```

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

const options = {
  accessKey: "子账号 accessKey",
  accessKeySecret: "子账号 accessKeySecret"
};

//1.初始化 API Client
const client = new RPCClient({
  accessKeyId: options.accessKey,
  secretAccessKey: options.accessKeySecret,

```


在企业实例的控制台，**日志服务**中，我们也可以追踪到完整的**下行链路日志**，如下图：

物联网平台 / 监控运维 / 日志服务

日志服务

产品: **家庭温控器**

云端运行日志 | 设备本地日志 | 日志转储

设备ID: **hxt193489234** | 请输入 TraceId | 请输入内容或键字: MessageId

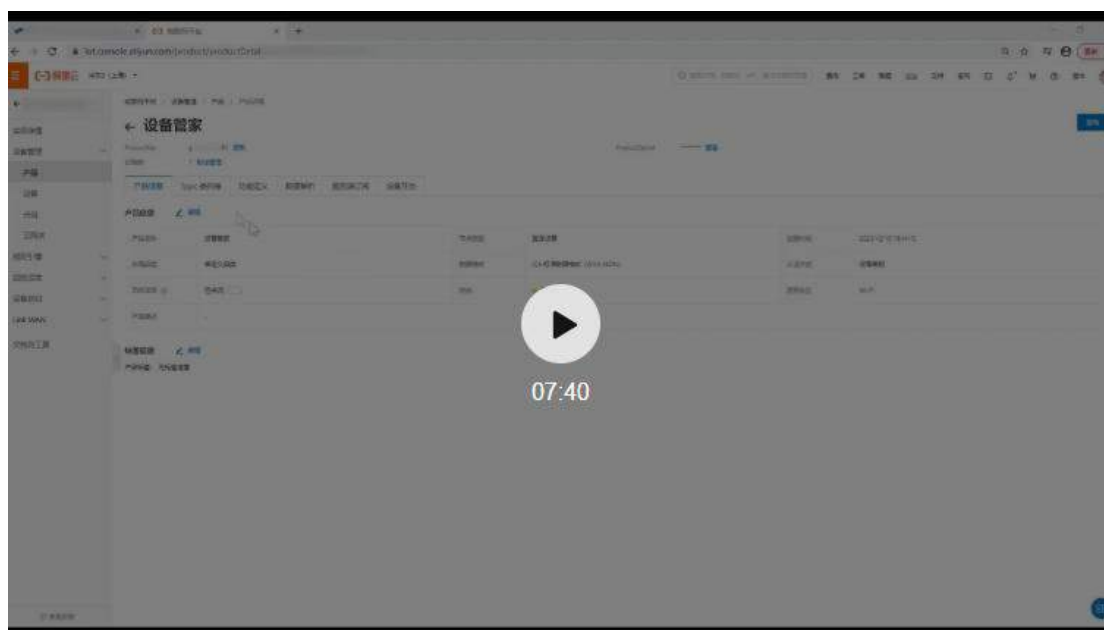
全部状态 | 1小时

搜索 | 重置

时间	TraceID	消息内容	DeviceName	业务类型(全部)	操作	内容	状态
2020/7/24 12:04:36	0a10031016045501	设备响应ACK	hxt193489234	云到设备消息	设备响应ACK	{"Content": "pubAck from device, ..."	200
2020/7/24 12:04:32	0bc3adc016045501	IoT 平台 Publish 消息到设备	hxt193489234	云到设备消息	IoT 平台 Publish 消息到设备	{"Content": "Publish message to ..."	200
2020/7/24 12:04:26	0bc3adc016045501	云端 API 调用	hxt193489234	API 调用	Pub	{"Params": {"productKey": "g..."/>	200
2020/7/24 12:04:26	0bc3adc016045501	设备行为	hxt193489234	设备行为	online	设备行为	200

7 分钟视频讲解全链路开发

作者 | 苏堤嘉木



[点击原文查看视频](#)

自建 MQTT 迁移阿里云 IoT

作者 | 苏堤嘉木



随着企业 IoT 业务增长，终端设备增加，自建 IoT 平台不断遇到上层业务团队的挑战：

- 连接稳定性变弱，设备频繁掉线。
- 消息通信时延高，业务响应慢，消费者投诉。
- 服务器扩容，硬件投入和运维成本越来越高。

此时，设备全面上云，迁移到安全、稳定、低时延、高可用、免运维的阿里云 IoT 物联网平台便是非常明智的最佳选择。

一、企业现有业务架构

本次案例的企业，现有的基于自建 MQTT 集群的业务架构如下图：



企业 10 万终端设备接入自建 MQTT 集群，设备 15 分钟上报一次业务数据到接入 MQTT 集群，实时流转 to 消息队列 Kafka 中，业务系统从 Kafka 消费数据，按业务逻辑处理后落库，满足条件的做实时短信推送运维人员。

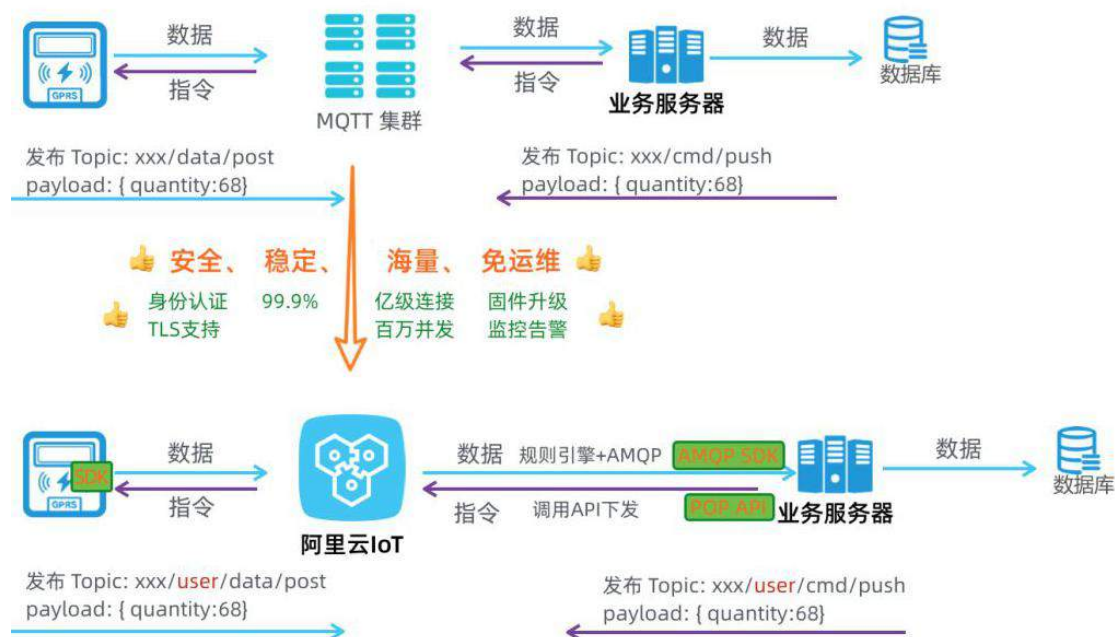
管理人员通过手机 App 下发配置参数到业务系统，业务系统调用 MQTT 集群的业务 API 接口，把配置指令推送到终端设备上。

基于 MQTT 协议的上报数据和下行指令的业务协议定义如下：

业务场景	主题Topic	报文Payload
设备上报数据	cdb/data/post	DE02,10,17,011101010,am024,1d478f
服务端控制指令	cdb/cmd/push	CMD,82923,ad322

二、设备上云迁移方案

为了减少设备端固件程序改造，尽量保持云上业务系统稳定，实现低成本，快速迁移到 IoT 物联网平台，定制了如下技术方案。



IoT 设备迁移上云三步走：

- 设备做**固件升级**，修改连接域名为 IoT 物联网平台 endpoint。
- 调整现有通信 topic 为 IoT 物联网平台产品**自定义 Topic**，保留业务报文格式不变。
- 配置**规则引擎**，把设备**数据流转**到服务端订阅 **AMQP 消费组**，业务服务器实时接收设备数据。

三、设备迁移上云实战

1. 创建产品

首先，我们开通 IoT 物联网平台服务，进入物联网平台控制台，在左侧栏选择**产品**，创建新产品：充电宝机柜。如下图：

- 所属品类：自定义品类
- 节点类型：直连设备
- 连网方式：WiFi （可以根据情况选择）
- 数据格式：标准数据格式 JSON（也可选 透传/自定义）



然后，我们把现有系统业务 Topic 映射到自定义通信的 Topic 类，如下表：

自建平台通信	迁移IoT平台后，自定义Topic和Payload	权限	描述
cdb/data/post	/\${productKey}/\${deviceName}/user/data/up	发布	设备→云端 上报数据
DE02,10,17,01 1101010,am024 ,1d478f	DE02,10,17,01 1101010,am024 ,1d478f		报文格式不变
cdb/cmd/push	/\${productKey}/\${deviceName}/user/cmd/down	订阅	云端→设备 下发指令
CMD,82923,ad 322	CMD,82923,ad 322		报文格式不变

最后，我们进入产品详情页面，选择 Topic 类列表，点击自定义 Topic，添加 Topic，如下图：



至此，我们完成了在 IoT 物联网平台的产品创建和业务通信协议定义。

2. 注册设备

基于已创建的产品，我们注册一个具体设备，获取到设备身份认证信息（三元组），用来和 IoT 物联网平台建立连接时身份认证。如下图：



3. 配置数据流转

接下来我们要做的是在 IoT 物联网平台配置数据流转方案：



创建服务端订阅 AMQP 消费组

我们进入控制台的服务端订阅，创建业务的 AMQP 消费组，用来消费设备生成的数据。如下图：

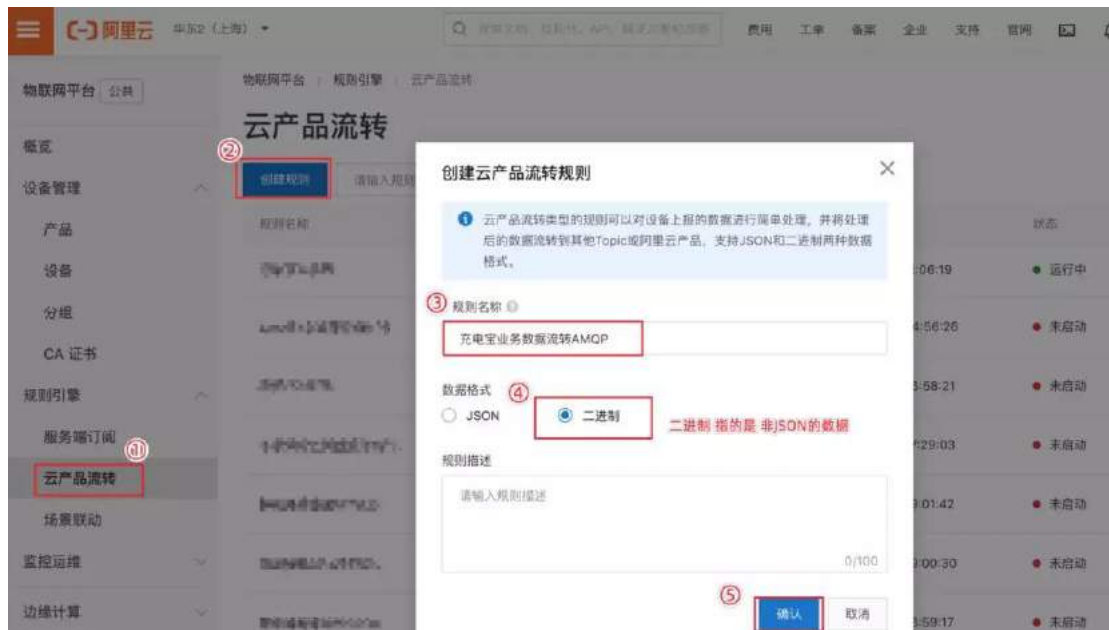


消费组创建完成后，我们进入消费组详情，可以查看消费组状态。由于我们设备端和服务端都没有连接到平台，数据都是为空。如下图：



配置规则引擎

接下来，我们要做的是配置规则引擎来处理数据，并流转到已创建的消费组上。我们选择云产品流转，点击创建规则，选择二进制格式（二进制泛指非 JSON 结构数据）。如下图：



然后，我们编写 SQL，让 IoT 物联网平台处理设备数据，并携带设备信息。

```
SELECT
deviceName() as deviceName,
timestamp() as timestamp ,
payload() as payload
FROM "/a*****E+/user/data/up"
```

如下图：



最后，我们配置数据转发操作到指定的 AMQP 消费组，如下图：



完整的规则引擎配置如下图所示：



不要忘记，我们要在规则引擎列表，启动规则。如下图：



四、设备端应用程序开发

在完成了云上控制台的配置工作后，我们要做的就是设备端业务开发。这里我们在 Mac 上用 nodejs 脚本模拟设备业务行为，设备 MQTT 连接，数据上报，指令接收。

完整代码如下：

```
// 引入依赖 mqtt 库，或自己实现
const mqtt = require('aliyun-iot-mqtt');

// 设备身份
var options = {
  productKey: "设备 pk",
  deviceName: "设备 dn",
  deviceSecret: "设备 ds",
  regionId: "cn-shanghai"
};

// 1.建立连接
const client = mqtt.getAliyunIotMqttClient(options);

// 2.设备接收云端指令数据
client.on('message', function(topic, message) {
  console.log("topic " + topic)
  console.log("message " + message)
```

```
    })  
// 3. 设备订阅指令的 Topic  
client.subscribe(`${options.productKey}/${options.deviceName}/user/cmd/down`)  
  
// 4. 模拟设备 上报数据（原始报文）  
setInterval(function() {  
    client.publish(`${options.productKey}/${options.deviceName}/user/data/up`, getPostData(),  
    {qos:1});  
  
}, 1000);  
  
// 模拟 设备原有报文格式  
function getPostData() {  
    let payload = "DE02,"+Math.floor((Math.random() * 20) + 10)  
    +"," +Math.floor((Math.random() * 20) + 10)  
    +",011101010,am024,1d478f";  
  
    console.log("payload=[ " + payload+" ]")  
    return JSON.stringify(payload);  
}
```

五、业务服务器开发

服务端我们以 Java 为例，演示设备数据接收和控制指令的下发。

服务端 maven 依赖如下：

```
<!-- amqp 1.0 qpid client -->  
<dependency>  
    <groupId>org.apache.qpid</groupId>  
    <artifactId>qpid-jms-client</artifactId>  
    <version>0.47.0</version>  
</dependency>  
<!-- aliyun pop sdk -->  
<dependency>  
    <groupId>com.aliyun</groupId>  
    <artifactId>aliyun-java-sdk-core</artifactId>
```

```
<version>4.4.4</version>
</dependency>
<!-- IoT pop sdk -->
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-iot</artifactId>
  <version>5.0.0</version>
</dependency>
```

1. 数据接收

我们参考 IoT 物联网平台服务端订阅 AMQP 文档：

https://help.aliyun.com/document_detail/143601.html

完整代码如下：

```
import org.apache.commons.codec.binary.Base64;
import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionListener;
import org.apache.qpid.jms.message.JmsInboundMessageDispatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.net.URI;
import java.util.Hashtable;

public class AMQPClient {

    private final static Logger logger = LoggerFactory.getLogger(AMQPClient.class);

    public static void main(String[] args) throws Exception {
        //参数说明，请参见上一篇文档：AMQP 客户端接入说明。
        String accessKey = "阿里云账号 ak";
```

```
String accessSecret = "阿里云账号 as";
String consumerGroupId = "服务端订阅消费组 ID";
long timeStamp = System.currentTimeMillis();

//签名方法: 支持 hmacmd5, hmacsha1 和 hmacsha256
String signMethod = "hmacsha1";
//控制台服务端订阅中消费组状态页客户端 ID 一栏将显示 clientId 参数。
//建议使用机器 UUID、MAC 地址、IP 等唯一标识等作为 clientId。便于您区分识别不同的客户端。
String clientId = "ecs_" + System.currentTimeMillis();

//UserName 组装方法, 请参见上一篇文档: AMQP 客户端接入说明。
String userName = clientId + "|authMode=aksign"
    + ",signMethod=" + signMethod
    + ",timestamp=" + timeStamp
    + ",authId=" + accessKey
    + ",consumerGroupId=" + consumerGroupId
    + "|";

//password 组装方法, 请参见上一篇文档: AMQP 客户端接入说明。
String signContent = "authId=" + accessKey + "&timestamp=" + timeStamp;
String password = doSign(signContent, accessSecret, signMethod);

//按照 qpid-jms 的规范, 组装连接 URL。
String connectionUrl = "failover:(amqps://替换你的阿里云账号 UID.iot-amqp.cn-shanghai.aliyuncs.com:5671?amqp.idleTimeout=80000)"
    + "?failover.reconnectDelay=30";

Hashtable<String, String> hashtable = new Hashtable<>();
hashtable.put("connectionfactory.SBCF", connectionUrl);
hashtable.put("queue.QUEUE", "default");
hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");

Context context = new InitialContext(hashtable);
ConnectionFactory cf = (ConnectionFactory) context.lookup("SBCF");
Destination queue = (Destination) context.lookup("QUEUE");

// Create Connection
Connection connection = cf.createConnection(userName, password);
((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);

// Create Session
// Session.CLIENT_ACKNOWLEDGE: 收到消息后, 需要手动调用 message.acknowledge()
// Session.AUTO_ACKNOWLEDGE: SDK 自动 ACK (推荐)
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
```

```

        connection.start();

// Create Receiver Link
MessageConsumer consumer = session.createConsumer(queue);
consumer.setMessageListener(messageListener);
}

private static MessageListener messageListener = new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            byte[] body = message.getBody(byte[].class);
            String content = new String(body);
            String topic = message.getStringProperty("topic");
            String messageId = message.getStringProperty("messageId");
            logger.info("receive message"
                + ", topic = " + topic
                + ", messageId = " + messageId
                + ", content = " + content);
            System.out.println();
            //如果创建 Session 选择的是 Session.CLIENT_ACKNOWLEDGE，这里需要手动 ACK。
            //message.acknowledge();
            //如果要对收到的消息做耗时的处理，请异步处理，确保这里不要有耗时逻辑。
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
};

private static JmsConnectionListener myJmsConnectionListener = new JmsConnectionList
ener() {
    /**
     * 连接成功建立。
     */
    @Override
    public void onConnectionEstablished(URL remoteURI) {
        logger.info("onConnectionEstablished, remoteUri:{}, remoteURI);
    }

    /**

```

```
* 尝试过最大重试次数之后，最终连接失败。

*/
@Override
public void onConnectionFailure(Throwable error) {
    logger.error("onConnectionFailure, {}", error.getMessage());
}

/**
 * 连接中断。
 */
@Override
public void onConnectionInterrupted(Uri remoteUri) {
    logger.info("onConnectionInterrupted, remoteUri:{}", remoteUri);
}

/**
 * 连接中断后又自动重连上。
 */
@Override
public void onConnectionRestored(Uri remoteUri) {
    logger.info("onConnectionRestored, remoteUri:{}", remoteUri);
}

@Override
public void onInboundMessage(JmsInboundMessageDispatch envelope) {}

@Override
public void onSessionClosed(Session session, Throwable cause) {}

@Override
public void onConsumerClosed(MessageConsumer consumer, Throwable cause) {}

@Override
public void onProducerClosed(MessageProducer producer, Throwable cause) {}
};

/**
 * password 签名计算方法，请参见上一篇文档：AMQP 客户端接入说明。
 */
```

```
private static String doSign(String toSignString, String secret, String signMethod) throws Exception {
    SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(), signMethod);
    Mac mac = Mac.getInstance(signMethod);
    mac.init(signingKey);
    byte[] rawHmac = mac.doFinal(toSignString.getBytes());
    return Base64.encodeBase64String(rawHmac);
}
```

2. 指令下发

参考云端下行指令的 Pub API 文档

https://help.aliyun.com/document_detail/69793.html

完整代码如下：

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.IAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.iot.model.v20170420.PubRequest;
import com.aliyuncs.iot.model.v20170420.PubResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.google.gson.Gson;

public class PubClient {

    public static void main(String[] args) {

        DefaultProfile profile = DefaultProfile.getProfile(
            "cn-shanghai",
            "替换你的 ak",
            "替换你的 as");

        IAcsClient client = new DefaultAcsClient(profile);

        PubRequest request = new PubRequest();
        request.setTopicFullName("/替换设备 pk/替换设备 dn/user/cmd/down");
```

```
request.setMessageContent("Q01ELDgyOTIzLGFkMzlyCiA="); //原始报文 : CMD,82923,ad322

request.setProductKey("替换设备 pk");
request.setQos(1);

try {
    PubResponse response = client.getAcsResponse(request);
    System.out.println(new Gson().toJson(response));

} catch (ClientException e) {
    System.out.println("ErrCode:" + e.getErrCode());
    System.out.println("ErrMsg:" + e.getErrMsg());
    System.out.println("RequestId:" + e.getRequestId());
}
}
```

至此，我们完成了服务端业务开发。

六、业务和 IoT 平台集成运行

我们先启动服务端程序和 IoT 物联网平台建立连接，然后启动设备端模拟脚本。

1. 设备上报数据

在设备端日志，我们看到打印出来的业务报文如下图：



```
node mqtt2IoT.js
payload=[ DE02,28,15,011101010,am024,1d478f ]
payload=[ DE02,24,16,011101010,am024,1d478f ]
payload=[ DE02,27,24,011101010,am024,1d478f ]
payload=[ DE02,18,11,011101010,am024,1d478f ]
payload=[ DE02,29,13,011101010,am024,1d478f ]
payload=[ DE02,15,13,011101010,am024,1d478f ]
payload=[ DE02,29,25,011101010,am024,1d478f ]
payload=[ DE02,14,12,011101010,am024,1d478f ]
payload=[ DE02,25,18,011101010,am024,1d478f ]
payload=[ DE02,10,26,011101010,am024,1d478f ]
payload=[ DE02,10,10,011101010,am024,1d478f ]
```

设备原始报文

在设备打印出来日志同时，我们能看到服务端实时在打印从 IoT 物联网平台获取到的消息数据，如下图：

```

Run: AMQPClient
13:46:43,185 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060706261960192, content = {"payload":"\\DE02,26,15,01101010,am024,1d478f\\", "timestamp":157828960192}
13:46:44,167 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060710519202384, content = {"payload":"\\DE02,24,16,01101010,am024,1d478f\\", "timestamp":1578289604192}
13:46:45,167 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060714675751937, content = {"payload":"\\DE02,27,24,01101010,am024,1d478f\\", "timestamp":1578289605199}
13:46:46,155 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060718632306688, content = {"payload":"\\DE02,16,11,01101010,am024,1d478f\\", "timestamp":1578289606188}
13:46:47,160 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060723047391936, content = {"payload":"\\DE02,29,13,01101010,am024,1d478f\\", "timestamp":1578289607105}
13:46:48,158 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060727258666496, content = {"payload":"\\DE02,15,13,01101010,am024,1d478f\\", "timestamp":1578289608192}
13:46:49,155 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060731444588544, content = {"payload":"\\DE02,25,25,01101010,am024,1d478f\\", "timestamp":1578289609190}
13:46:50,164 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 121406073567610500, content = {"payload":"\\DE02,14,12,01101010,am024,1d478f\\", "timestamp":1578289610190}
13:46:51,186 INFO [JmsSession [ID:bfb2ba729-3ec3-42ef-89c6-41a1dff2df19:1:1] delivery dispatcher] AMQPClient:76 - receive message, topic = /a1bbmzf9JRE/x202001dA29189334, messageId = 1214060739891911169, content = {"payload":"\\DE02,25,18,01101010,am024,1d478f\\", "timestamp":1578289611203}
  
```

2. 服务端下发控制指令

我们在服务端执行 Pub 接口调用，像设备下发控制指令。如下图：

```

PubClient x
/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home/bin/java ...
objc[30933]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/Home/bin/java ...
{"requestId":"33D520DD-07BB-4D0A-AEFD-BD854AC8D282","success":true,"messageId":"1214070152740565504"}
Process finished with exit code 0
  
```

此时，设备端会实时打印出来指令信息，如下图：

```

$ node mqtt2IoT.js
topic /a1bbmzf9JRE/x202001dA29189334/user/cmd/down
message CMD,82923,ad322
  
```

七、全业务链路日志

在阿里云 IoT 的控制台日志服务里，我们可以查看设备行为日志：



在日志服务的上行消息分析里，可以查看消息详情，包括 Topic 和 Payload。



也可以跟踪上行消息的流转过程，如下图：



同样，我们可以在控制台的日志服务，下行消息分析中，查看到指令消息流转完整过程，如下图：



至此，我们在尽量少改动设备端程序和云上业务系统的前提下，完成了存量设备迁移到阿里云 IoT 物联网平台的工作。

TCP 协议 IoT 设备迁移上云

作者 | 苏堤嘉木



一、GPS 定位器

GPS 定位器是内置了 GPS 模块和移动通信模块的终端，用来将 GPS 模块获得的定位数据通过移动通信模块传至云上的一台服务器，从而可以实现在电脑或手机上查询终端位置。

GPS 定位器可用于儿童和老人的行踪掌控，公路巡检，贵重货物跟踪，追踪与勤务派遣，个人财物跟踪，宠物跟踪，野生动物追踪，货运业，汽车防盗，电动车防盗，摩托车防盗，银行运钞车，公务车管理等。

二、存量设备 TCP 方案

定位器行业发展多年，形成了基于 TCP 协议的成熟方案。

定位器设备启动后，基于 TCP/IP 协议和云端的业务服务器建立连接，然后通过校时指令来同步时钟，之后定时发送心跳包来保持 TCP 长连接，GPS 数据变化超过阈值时，自动上报当前坐标的经纬度值，设备电量变化信息也会定时上报到云端，以便优化 GPS 数据采集规则。

云端服务器也可以推送配置信息和控制指令到定位器设备，以改变定位器行为模式。

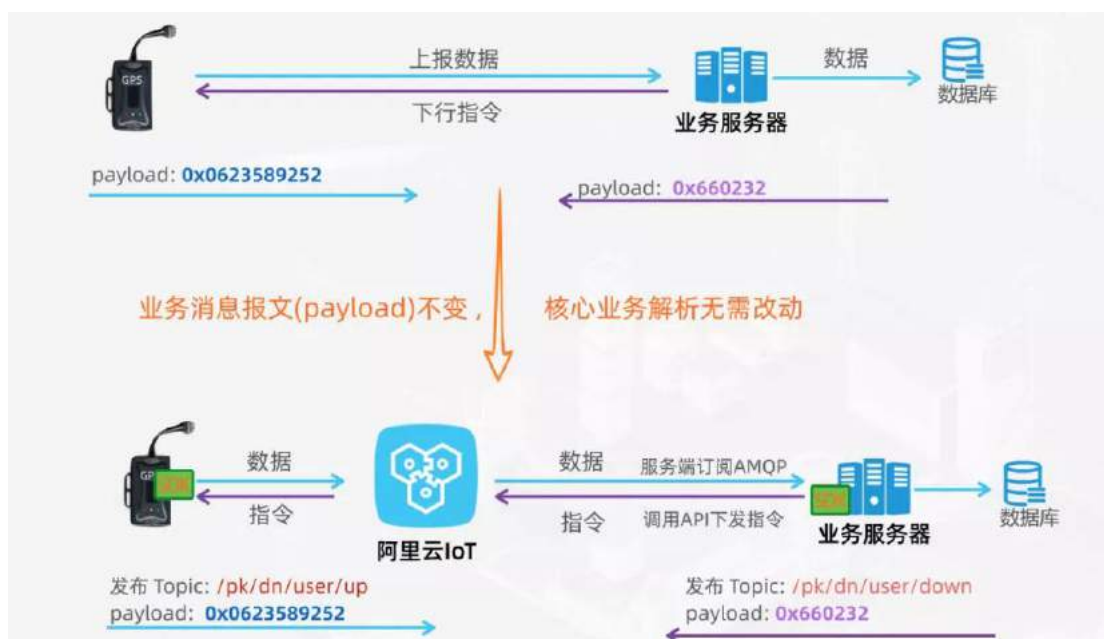
简单分为以下数据交互过程：



业务行为分类	Device → Server	Server → Device
登录	0x01+身份信息	0x61+认证结果
时钟同步	0x03+本地时间戳	0x63+服务端时间戳
心跳包	0x05	0x65
上报数据	0x06+业务数据	0x66+响应
下行指令	0x08+响应	0x68+业务数据

三、TCP 设备上云方案

随着云计算厂商布局物联网场景技术产品，基于 MQTT 协议的全托管的 IoT 云服务逐渐成为 70% 中小企业做物联网业务的首选方案。从 TCP 协议迁移到 MQTT 协议的方案也逐渐成熟，以实现存量设备低成本的快速迁移上云，减少设备端和业务系统的改造，极大的提升整体安全性，稳定性，大大降低业务时延，借助云上动态无限扩容能力承载海量规模增长。



当我们把设备连接迁移到阿里云 IoT 物联网平台之后，重新梳理业务链路，会发现 IoT 物联网服务承载了繁重的和设备交互的工作，云上的业务服务器压力变得小了很多。



身份认证

定位器设备和 IoT 物联网平台基于 MQTT 协议通信，TCP 报文调整为 MQTT 的 CONNECT/CONNACK 报文，此时业务服务器不需要做身份校验工作，IoT 物联网平台会把设备上线/离线消息通过规则引擎实时推送到业务服务器。

时钟同步

企业基于 TCP 搭建的时钟同步服务也可以下线了，IoT 物联网平台提供了完整的 NTP 服务，解决嵌入式设备资源受限，端上没有精确时间戳的问题。

详细文档 https://help.aliyun.com/document_detail/102509.html

请求Topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/request`

```
1 {  
2   "deviceSendTime": "1571724098000"  
3 }
```

响应Topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`

```
1 {  
2   "deviceSendTime": "1571724098000",  
3   "serverRecvTime": "1571724098110",  
4   "serverSendTime": "1571724098115",  
5 }
```

设备上的精确时间为:
$$(\${serverRecvTime} + \${serverSendTime} + \${deviceRecvTime} - \${deviceSendTime}) / 2$$

心跳

MQTT 协议本身约定了 PINGREQ/PINGRESP 的心跳机制，此时也不需要业务服务器介入，IoT 物联网平台会响应设备心跳行为。

双向消息通信

基于 MQTT 协议通信是需要约定 topic 和 payload，为了减少业务系统改动，我们增加两个 Topic 定义，消息报文结构体保持不变。

	设备上报	云端下行
TCP	二进制消息报文，以0x0开头	二进制消息报文，以0x6开头
MQTT	Topic: <code>/pk/{dn}/user/up</code> Payload: 二进制消息报文	Topic: <code>/pk/{dn}/user/down</code> Payload: 二进制消息报文

设备上报业务数据后，通过规则引擎配置，我们把上行的 Topic: `/pk/{dn}/user/up` 中的 payload 数据实时流转到业务系统，数据格式不变；业务系统推送配置信息或指令时，IoT 物联网平台封装到下行的 Topic: `/pk/{dn}/user/up` 中的 payload 里，设备接收到的业务数据格式不变。

四、设备迁移实战

1. 创建产品

我们进入 IoT 物联网平台控制台，创建 GPS 定位器产品，并添加自定义通信 Topic，如下：





选择消费组，并携带 Tag 信息到业务系统。



4. 设备启动

设备烧录三元组，启动后，上报数据，在业务系统会收到如下格式的 GPS 数据。



5. 控制台查看消费组情况

在服务端订阅的消费组详情，我们可以看到消费速率，消息堆积量，以及消费者列表。



消息详情:



数据流转完整过程。

物联网平台 / 公共

物联网平台 / 监控运维 / 日志服务

日志服务

产品: GPS定位器

云端运行日志 设备本地日志

请输入DeviceName 请输入TraceId 请输入MessageID

请输入内容关键字 全部状态 1小时

搜索 重置

时间	TraceID	MessageID	DeviceName	业务类型(全部)	操作	内容	状态
2020-07-20 10:10:10	1270188283	1270188283	Shuang... ThePc	① 设备 → IoT 设备到云消息	设备到云消息	["Content": "Publish message to..."]	200
2020-07-20 10:10:10	1270188283	1270188283	Shuang... ThePc	② IoT → 规则引擎 规则引擎	规则引擎	["Content": "Transmit to target..."]	200
2020-07-20 10:10:10	1270188283	1270188283	Shuang... ThePc	④ 业务服务器响应ACK 服务订阅	AMQP	["Content": "receive ack from..."]	200
2020-07-20 10:10:10	1270188283	1270188283	Shuang... ThePc	③ IoT → 业务服务器 服务订阅	AMQP	["Content": "push message to..."]	200

AI 人脸识别，监控老板行踪！

作者 | 苏堤嘉木

虽然已融入打工人队伍多年，但学生时代，老师站在后窗外的阴影依然挥之不去。在刷头条，看 B 站的时候，总是担心着老板来了！

直到有了 ESP-EYE 低成本的人脸识别开发板，结合云上 IoT 物联网服务，轻松搞定老板行踪监控。无论老板从何处来，钉钉摸鱼群总会收到实时提醒。

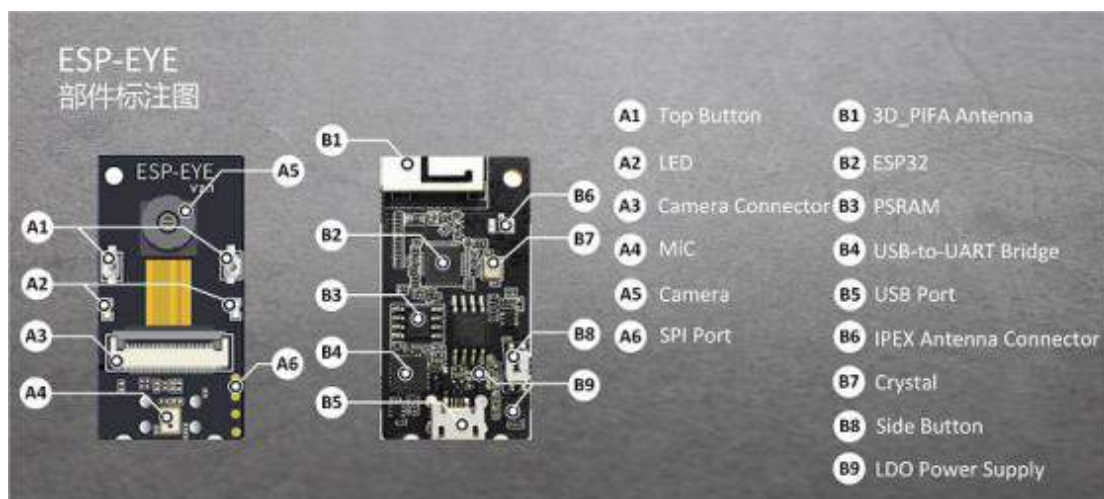
一、技术方案

首先，众筹一块 ESP-EYE 本地人脸识别开发板；其次，录入老板人脸信息；然后，把开发板连接云端 IoT 物联网平台；接着，通过规则引擎把数据流转到函数计算；最后，函数计算中调用钉钉群机器人，完成老板来了告警！



二、ESP-EYE 人脸识别

ESP-EYE 是一款专注于本地图像识别的开发板，板载 ESP32 芯片，集成 200 万像素摄像头，拥有 8 MByte PSRAM 和 4 MByte flash 的丰富存储，支持 Wi-Fi 图像传输与 Micro USB 调试与供电，可广泛应用于 AI 智能物联网领域的应用开发。



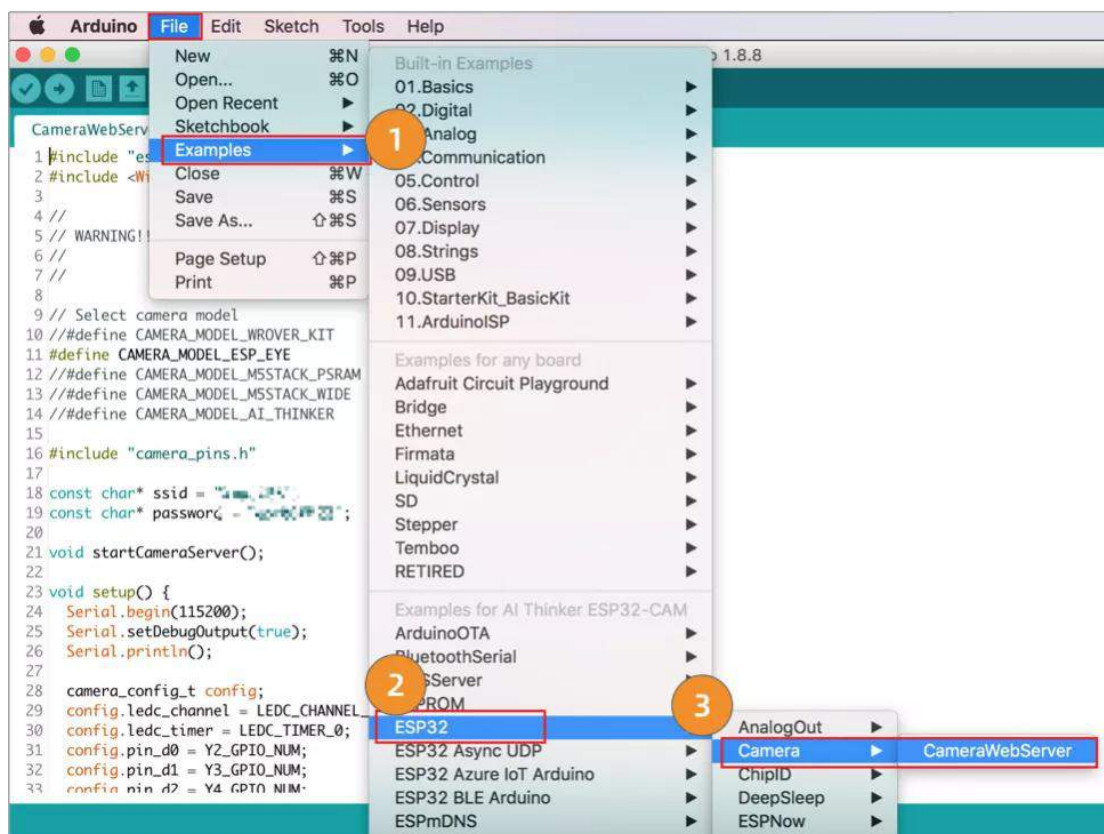
1. 烧录人脸识别程序

我们基于 Arduino 编程来降低 ESP-EYE 人脸识别程序开发难度。

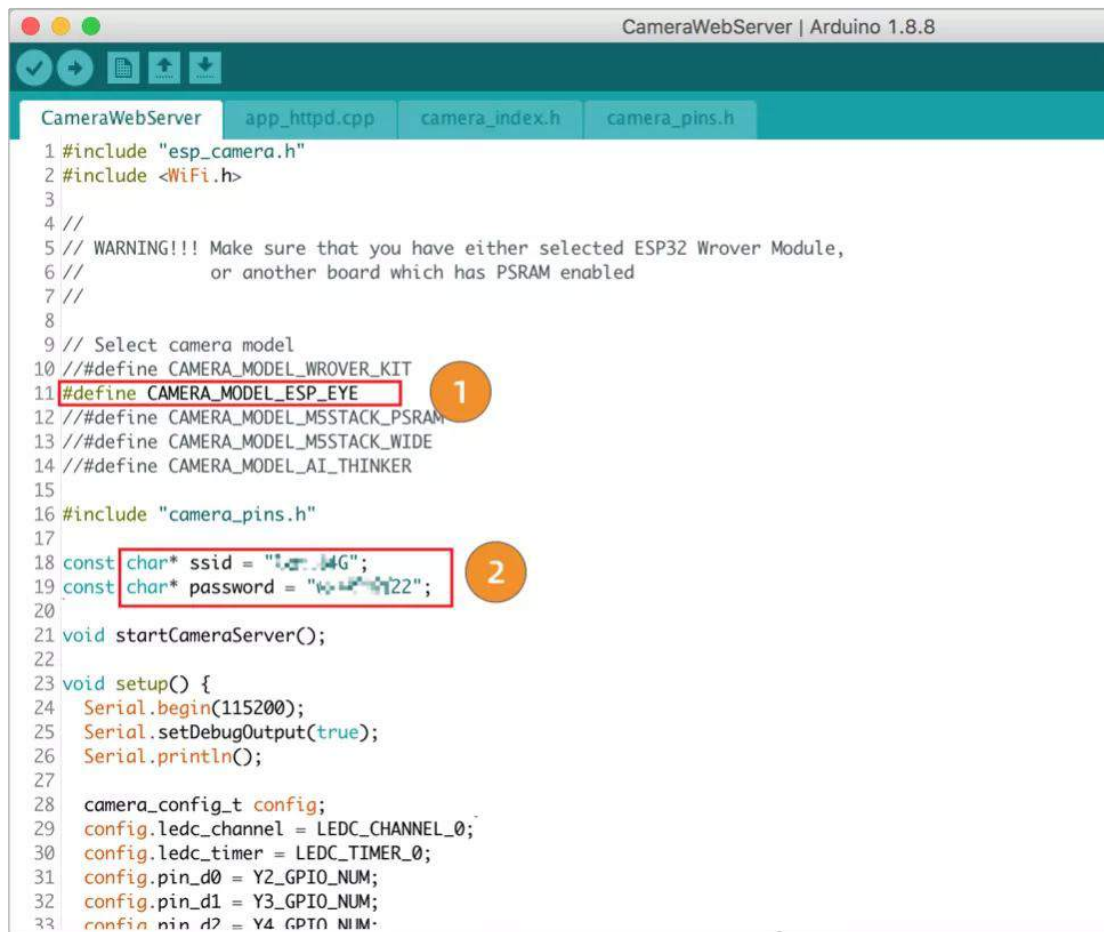
首先，我们在 Preferences 中新增 arduino-esp32 配置 URL：

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

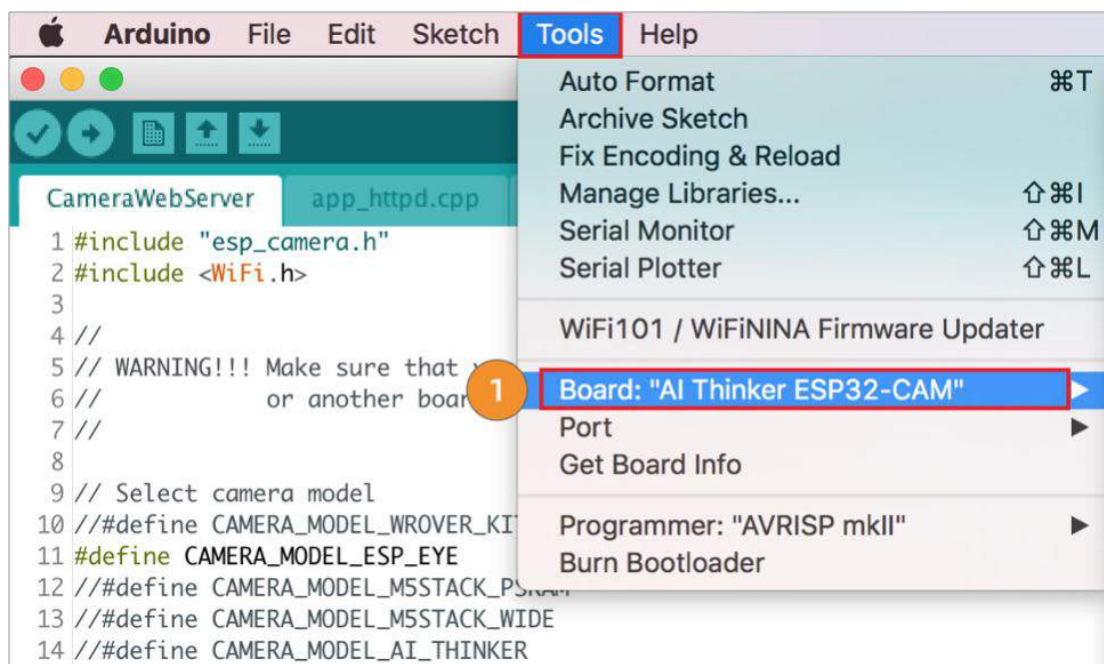
安装完成后，我们基于 CameraWebServer 示例程序做二次开发。



打开 CameraWebServer 文件，选择 CAMERA_MODEL_ESP_EYE，设置办公室的 ssid 和 password，用于开发板联网。



修改完成后，我们在 Tools 中选择 Board 为 AI Thinker ESP32-CAM。



The screenshot shows the Arduino IDE interface with the 'CameraWebServer' sketch loaded. The code files 'app_httpd.cpp', 'camera_index.h', and 'camera_pins.h' are visible in the top editor pane. The Serial Monitor window is open, displaying the output of the program. The output shows the selection of camera module 1, the start of the web server on port 80, and the start of the stream server on port 81. A red box highlights the message 'Camera Ready! Use 'http://192.168.3.23' to connect'. The Serial Monitor settings are set to 115200 baud and Autoscroll is checked.

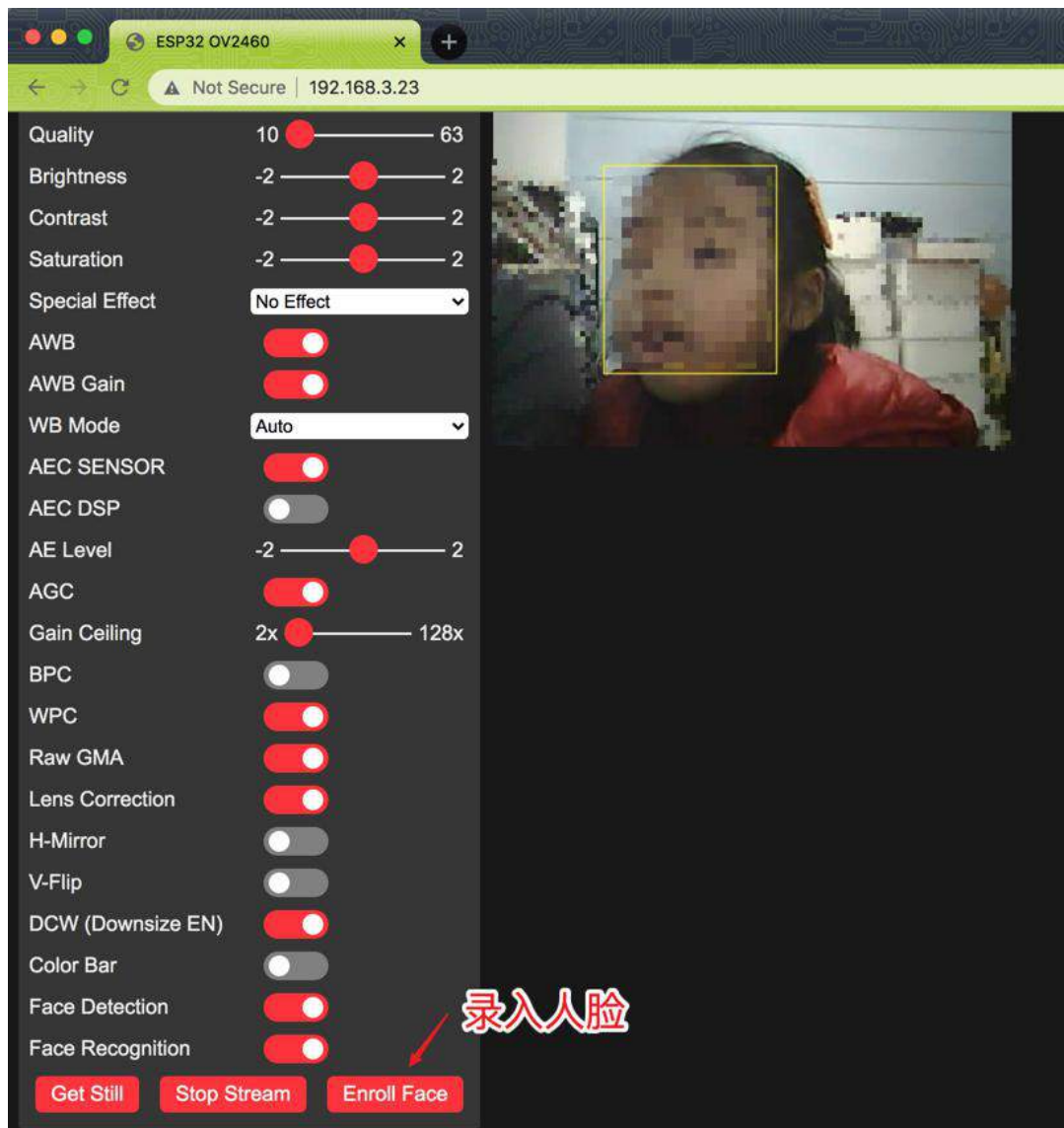
Code files: `app_httpd.cpp`, `camera_index.h`, `camera_pins.h`

Serial Monitor Output:

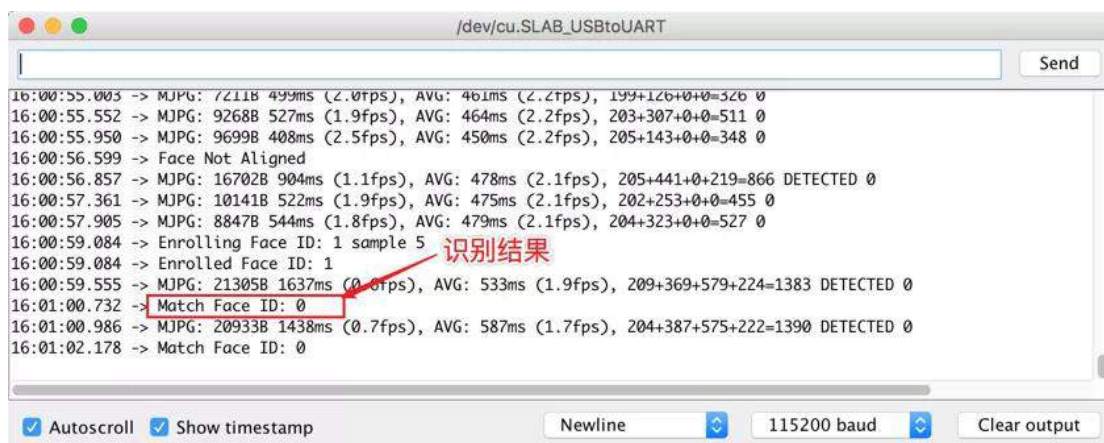
```

1 // #define CAMERA_MODULE 1
2 // #include <WiFi.h>
3
4 //
5 // WARNING!!! Make sure that you have configured the correct pin numbers in each of the source files!
6 //
7 //
8
9 // Select camera module
10 // #define CAMERA_MODULE 1
11 // #define CAMERA_MODEL_ESP8266
12 // #define CAMERA_MODEL_ESP8266_PIN_1
13 // #define CAMERA_MODEL_ESP8266_PIN_2
14 // #define CAMERA_MODEL_ESP8266_PIN_3
15 // #define CAMERA_MODEL_ESP8266_PIN_4
16 // #define CAMERA_MODEL_ESP8266_PIN_5
17 // #define CAMERA_MODEL_ESP8266_PIN_6
18 // #define CAMERA_MODEL_ESP8266_PIN_7
19 // #define CAMERA_MODEL_ESP8266_PIN_8
20 // #define CAMERA_MODEL_ESP8266_PIN_9
21 // #define CAMERA_MODEL_ESP8266_PIN_10
22 // #define CAMERA_MODEL_ESP8266_PIN_11
23 // #define CAMERA_MODEL_ESP8266_PIN_12
24 // #define CAMERA_MODEL_ESP8266_PIN_13
25 // #define CAMERA_MODEL_ESP8266_PIN_14
26 // #define CAMERA_MODEL_ESP8266_PIN_15
27 // #define CAMERA_MODEL_ESP8266_PIN_16
28 // #define CAMERA_MODEL_ESP8266_PIN_17
29 // #define CAMERA_MODEL_ESP8266_PIN_18
30 // #define CAMERA_MODEL_ESP8266_PIN_19
31 // #define CAMERA_MODEL_ESP8266_PIN_20
32 // #define CAMERA_MODEL_ESP8266_PIN_21
33 // #define CAMERA_MODEL_ESP8266_PIN_22
34 // #define CAMERA_MODEL_ESP8266_PIN_23
35 // #define CAMERA_MODEL_ESP8266_PIN_24
36 // #define CAMERA_MODEL_ESP8266_PIN_25
37 // #define CAMERA_MODEL_ESP8266_PIN_26
38 // #define CAMERA_MODEL_ESP8266_PIN_27
39 // #define CAMERA_MODEL_ESP8266_PIN_28
40 // #define CAMERA_MODEL_ESP8266_PIN_29
41 // #define CAMERA_MODEL_ESP8266_PIN_30
42 // #define CAMERA_MODEL_ESP8266_PIN_31
43 // #define CAMERA_MODEL_ESP8266_PIN_32
44 // #define CAMERA_MODEL_ESP8266_PIN_33
45 // #define CAMERA_MODEL_ESP8266_PIN_34
46 // #define CAMERA_MODEL_ESP8266_PIN_35
47 // #define CAMERA_MODEL_ESP8266_PIN_36
48 // #define CAMERA_MODEL_ESP8266_PIN_37
49 // #define CAMERA_MODEL_ESP8266_PIN_38
50 // #define CAMERA_MODEL_ESP8266_PIN_39
51 // #define CAMERA_MODEL_ESP8266_PIN_40
52 // #define CAMERA_MODEL_ESP8266_PIN_41
53 // #define CAMERA_MODEL_ESP8266_PIN_42
54 // #define CAMERA_MODEL_ESP8266_PIN_43
55 // #define CAMERA_MODEL_ESP8266_PIN_44
56 // #define CAMERA_MODEL_ESP8266_PIN_45
57 // #define CAMERA_MODEL_ESP8266_PIN_46
58 // #define CAMERA_MODEL_ESP8266_PIN_47
59 // #define CAMERA_MODEL_ESP8266_PIN_48
60 // #define CAMERA_MODEL_ESP8266_PIN_49
61 // #define CAMERA_MODEL_ESP8266_PIN_50
62 // #define CAMERA_MODEL_ESP8266_PIN_51
63 // #define CAMERA_MODEL_ESP8266_PIN_52
64 // #define CAMERA_MODEL_ESP8266_PIN_53
65 // #define CAMERA_MODEL_ESP8266_PIN_54
66 // #define CAMERA_MODEL_ESP8266_PIN_55
67 // #define CAMERA_MODEL_ESP8266_PIN_56
68 // #define CAMERA_MODEL_ESP8266_PIN_57
69 // #define CAMERA_MODEL_ESP8266_PIN_58
70 // #define CAMERA_MODEL_ESP8266_PIN_59
71 // #define CAMERA_MODEL_ESP8266_PIN_60
72 // #define CAMERA_MODEL_ESP8266_PIN_61
73 // #define CAMERA_MODEL_ESP8266_PIN_62
74 // #define CAMERA_MODEL_ESP8266_PIN_63
75 // #define CAMERA_MODEL_ESP8266_PIN_64
76 // #define CAMERA_MODEL_ESP8266_PIN_65
77 // #define CAMERA_MODEL_ESP8266_PIN_66
78 // #define CAMERA_MODEL_ESP8266_PIN_67
79 // #define CAMERA_MODEL_ESP8266_PIN_68
80 // #define CAMERA_MODEL_ESP8266_PIN_69
81 // #define CAMERA_MODEL_ESP8266_PIN_70
82 // #define CAMERA_MODEL_ESP8266_PIN_71
83 // #define CAMERA_MODEL_ESP8266_PIN_72
84 // #define CAMERA_MODEL_ESP8266_PIN_73
85 // #define CAMERA_MODEL_ESP8266_PIN_74
86 // #define CAMERA_MODEL_ESP8266_PIN_75
87 // #define CAMERA_MODEL_ESP8266_PIN_76
88 // #define CAMERA_MODEL_ESP8266_PIN_77
89 // #define CAMERA_MODEL_ESP8266_PIN_78
90 // #define CAMERA_MODEL_ESP8266_PIN_79
91 // #define CAMERA_MODEL_ESP8266_PIN_80
92 // #define CAMERA_MODEL_ESP8266_PIN_81
93 // #define CAMERA_MODEL_ESP8266_PIN_82
94 // #define CAMERA_MODEL_ESP8266_PIN_83
95 // #define CAMERA_MODEL_ESP8266_PIN_84
96 // #define CAMERA_MODEL_ESP8266_PIN_85
97 // #define CAMERA_MODEL_ESP8266_PIN_86
98 // #define CAMERA_MODEL_ESP8266_PIN_87
99 // #define CAMERA_MODEL_ESP8266_PIN_88
100 // #define CAMERA_MODEL_ESP8266_PIN_89
101 // #define CAMERA_MODEL_ESP8266_PIN_90
102 // #define CAMERA_MODEL_ESP8266_PIN_91
103 // #define CAMERA_MODEL_ESP8266_PIN_92
104 // #define CAMERA_MODEL_ESP8266_PIN_93
105 // #define CAMERA_MODEL_ESP8266_PIN_94
106 // #define CAMERA_MODEL_ESP8266_PIN_95
107 // #define CAMERA_MODEL_ESP8266_PIN_96
108 // #define CAMERA_MODEL_ESP8266_PIN_97
109 // #define CAMERA_MODEL_ESP8266_PIN_98
110 // #define CAMERA_MODEL_ESP8266_PIN_99
111 // #define CAMERA_MODEL_ESP8266_PIN_100
112 // #define CAMERA_MODEL_ESP8266_PIN_101
113 // #define CAMERA_MODEL_ESP8266_PIN_102
114 // #define CAMERA_MODEL_ESP8266_PIN_103
115 // #define CAMERA_MODEL_ESP8266_PIN_104
116 // #define CAMERA_MODEL_ESP8266_PIN_105
117 // #define CAMERA_MODEL_ESP8266_PIN_106
118 // #define CAMERA_MODEL_ESP8266_PIN_107
119 // #define CAMERA_MODEL_ESP8266_PIN_108
120 // #define CAMERA_MODEL_ESP8266_PIN_109
121 // #define CAMERA_MODEL_ESP8266_PIN_110
122 // #define CAMERA_MODEL_ESP8266_PIN_111
123 // #define CAMERA_MODEL_ESP8266_PIN_112
124 // #define CAMERA_MODEL_ESP8266_PIN_113
125 // #define CAMERA_MODEL_ESP8266_PIN_114
126 // #define CAMERA_MODEL_ESP8266_PIN_115
127 // #define CAMERA_MODEL_ESP8266_PIN_116
128 // #define CAMERA_MODEL_ESP8266_PIN_117
129 // #define CAMERA_MODEL_ESP8266_PIN_118
130 // #define CAMERA_MODEL_ESP8266_PIN_119
131 // #define CAMERA_MODEL_ESP8266_PIN_120
132 // #define CAMERA_MODEL_ESP8266_PIN_121
133 // #define CAMERA_MODEL_ESP8266_PIN_122
134 // #define CAMERA_MODEL_ESP8266_PIN_123
135 // #define CAMERA_MODEL_ESP8266_PIN_124
136 // #define CAMERA_MODEL_ESP8266_PIN_125
137 // #define CAMERA_MODEL_ESP8266_PIN_126
138 // #define CAMERA_MODEL_ESP8266_PIN_127
139 // #define CAMERA_MODEL_ESP8266_PIN_128
140 // #define CAMERA_MODEL_ESP8266_PIN_129
141 // #define CAMERA_MODEL_ESP8266_PIN_130
142 // #define CAMERA_MODEL_ESP8266_PIN_131
143 // #define CAMERA_MODEL_ESP8266_PIN_132
144 // #define CAMERA_MODEL_ESP8266_PIN_133
145 // #define CAMERA_MODEL_ESP8266_PIN_134
146 // #define CAMERA_MODEL_ESP8266_PIN_135
147 // #define CAMERA_MODEL_ESP8266_PIN_136
148 // #define CAMERA_MODEL_ESP8266_PIN_137
149 // #define CAMERA_MODEL_ESP8266_PIN_138
150 // #define CAMERA_MODEL_ESP8266_PIN_139
151 // #define CAMERA_MODEL_ESP8266_PIN_140
152 // #define CAMERA_MODEL_ESP8266_PIN_141
153 // #define CAMERA_MODEL_ESP8266_PIN_142
154 // #define CAMERA_MODEL_ESP8266_PIN_143
155 // #define CAMERA_MODEL_ESP8266_PIN_144
156 // #define CAMERA_MODEL_ESP8266_PIN_145
157 // #define CAMERA_MODEL_ESP8266_PIN_146
158 // #define CAMERA_MODEL_ESP8266_PIN_147
159 // #define CAMERA_MODEL_ESP8266_PIN_148
160 // #define CAMERA_MODEL_ESP8266_PIN_149
161 // #define CAMERA_MODEL_ESP8266_PIN_150
162 // #define CAMERA_MODEL_ESP8266_PIN_151
163 // #define CAMERA_MODEL_ESP8266_PIN_152
164 // #define CAMERA_MODEL_ESP8266_PIN_153
165 // #define CAMERA_MODEL_ESP8266_PIN_154
166 // #define CAMERA_MODEL_ESP8266_PIN_155
167 // #define CAMERA_MODEL_ESP8266_PIN_156
168 // #define CAMERA_MODEL_ESP8266_PIN_157
169 // #define CAMERA_MODEL_ESP8266_PIN_158
170 // #define CAMERA_MODEL_ESP8266_PIN_159
171 // #define CAMERA_MODEL_ESP8266_PIN_160
172 // #define CAMERA_MODEL_ESP8266_PIN_161
173 // #define CAMERA_MODEL_ESP8266_PIN_162
174 // #define CAMERA_MODEL_ESP8266_PIN_163
175 // #define CAMERA_MODEL_ESP8266_PIN_164
176 // #define CAMERA_MODEL_ESP8266_PIN_165
177 // #define CAMERA_MODEL_ESP8266_PIN_166
178 // #define CAMERA_MODEL_ESP8266_PIN_167
179 // #define CAMERA_MODEL_ESP8266_PIN_168
180 // #define CAMERA_MODEL_ESP8266_PIN_169
181 // #define CAMERA_MODEL_ESP8266_PIN_170
182 // #define CAMERA_MODEL_ESP8266_PIN_171
183 // #define CAMERA_MODEL_ESP8266_PIN_172
184 // #define CAMERA_MODEL_ESP8266_PIN_173
185 // #define CAMERA_MODEL_ESP8266_PIN_174
186 // #define CAMERA_MODEL_ESP8266_PIN_175
187 // #define CAMERA_MODEL_ESP8266_PIN_176
188 // #define CAMERA_MODEL_ESP8266_PIN_177
189 // #define CAMERA_MODEL_ESP8266_PIN_178
190 // #define CAMERA_MODEL_ESP8266_PIN_179
191 // #define CAMERA_MODEL_ESP8266_PIN_180
192 // #define CAMERA_MODEL_ESP8266_PIN_181
193 // #define CAMERA_MODEL_ESP8266_PIN_182
194 // #define CAMERA_MODEL_ESP8266_PIN_183
195 // #define CAMERA_MODEL_ESP8266_PIN_184
196 // #define CAMERA_MODEL_ESP8266_PIN_185
197 // #define CAMERA_MODEL_ESP8266_PIN_186
198 // #define CAMERA_MODEL_ESP8266_PIN_187
199 // #define CAMERA_MODEL_ESP8266_PIN_188
200 // #define CAMERA_MODEL_ESP8266_PIN_189
201 // #define CAMERA_MODEL_ESP8266_PIN_190
202 // #define CAMERA_MODEL_ESP8266_PIN_191
203 // #define CAMERA_MODEL_ESP8266_PIN_192
204 // #define CAMERA_MODEL_ESP8266_PIN_193
205 // #define
```

在同一局域网内,输入控制台网址,进入 ESP32 人脸识别程序控制台,启动视频推流,开启人脸识别,点击 **Enroll Face** 来完成老板们的人脸库录入。



当有老板经过，我们会在控制台看到如下日志输出: Match Face ID: 0。



三、IoT 物联网平台开发

本地识别成功后，我们需要把老板来了的消息发送到云端，以便完成钉钉群告警。

1. 创建产品和注册设备

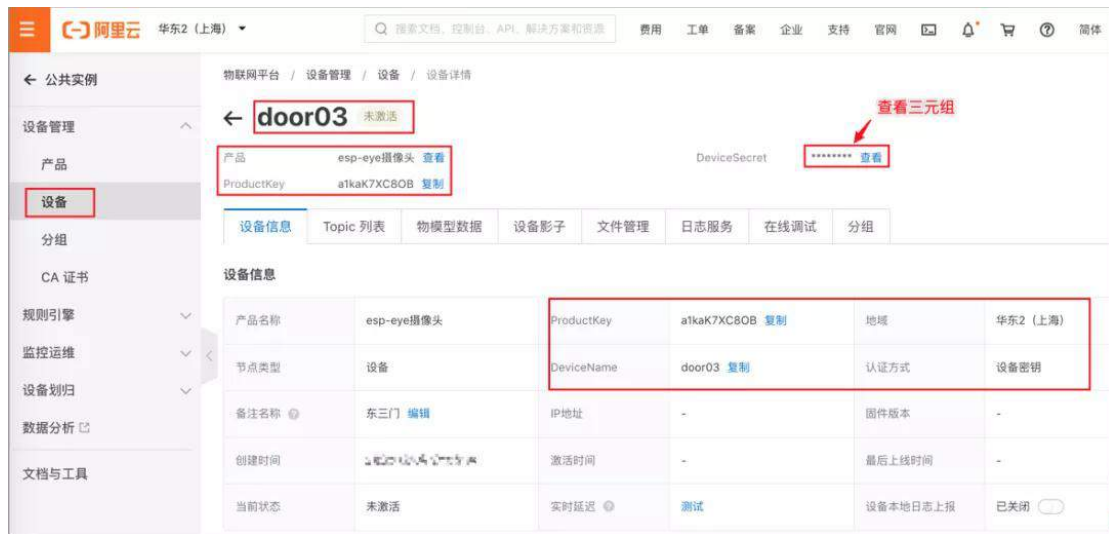
我们在 IoT 物联网平台控制台，创建产品 esp-eye 摄像头，并添加物模型-属性：老板编号，标识符为 bossId，取值范围 0~10。



物模型属性:

类型	名称	标识符	备注
属性	老板编号	bossId	0：唐僧； 1：观音菩萨； 2：如来佛祖。

我们在 esp-eye 摄像头产品下，注册一个设备，并获取到设备身份三元组，用于和云端建立连接时的身份认证。

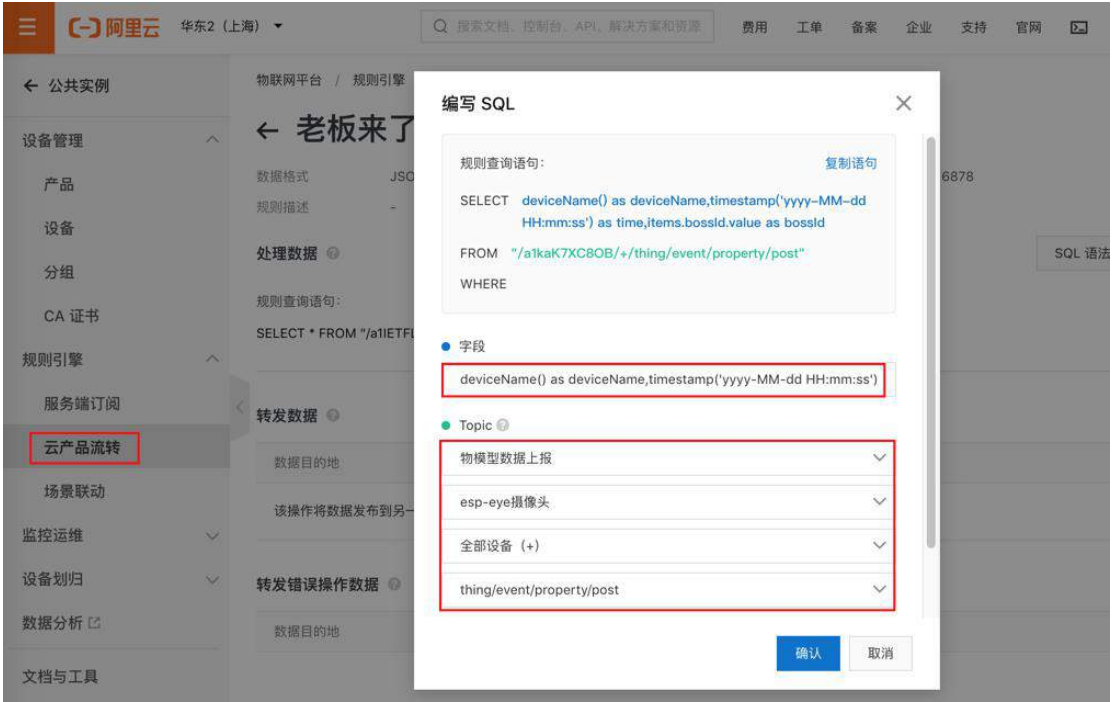


2. 配置规则引擎

在云产品流转中，我们需要配置一条消息流转的规则引擎，把设备上报的物模型数据，流转到函数计算 pushData2DingTalk 中。



数据处理 SQL 编写：



转发数据到函数计算的配置：



3.0 函数计算脚本

用于接收 IoT 平台 传递过来的消息，并实时推送到钉钉群中的函数计算脚本完整内容如下：

```
const https = require('https');
const accessToken = '钉钉机器人 token';
const boss = ["唐僧","观音菩萨","如来佛祖"];

module.exports.handler = function(event, context, callback) {
  //IoT 平台 传递过来的 event 数据
  var eventJson = JSON.parse(event.toString());

  const postData = JSON.stringify({
    "msgtype": "markdown",
    "markdown": {
      "text": "老板["+boss[eventJson.bossId]+"]来了，安心工作！"
    },
    "at": {
      "isAtAll": true
    }
  });

  const options = {
    hostname: 'oapi.dingtalk.com',
    port: 443,
    path: '/robot/send?access_token='+accessToken,
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Content-Length': Buffer.byteLength(postData)
    }
  };

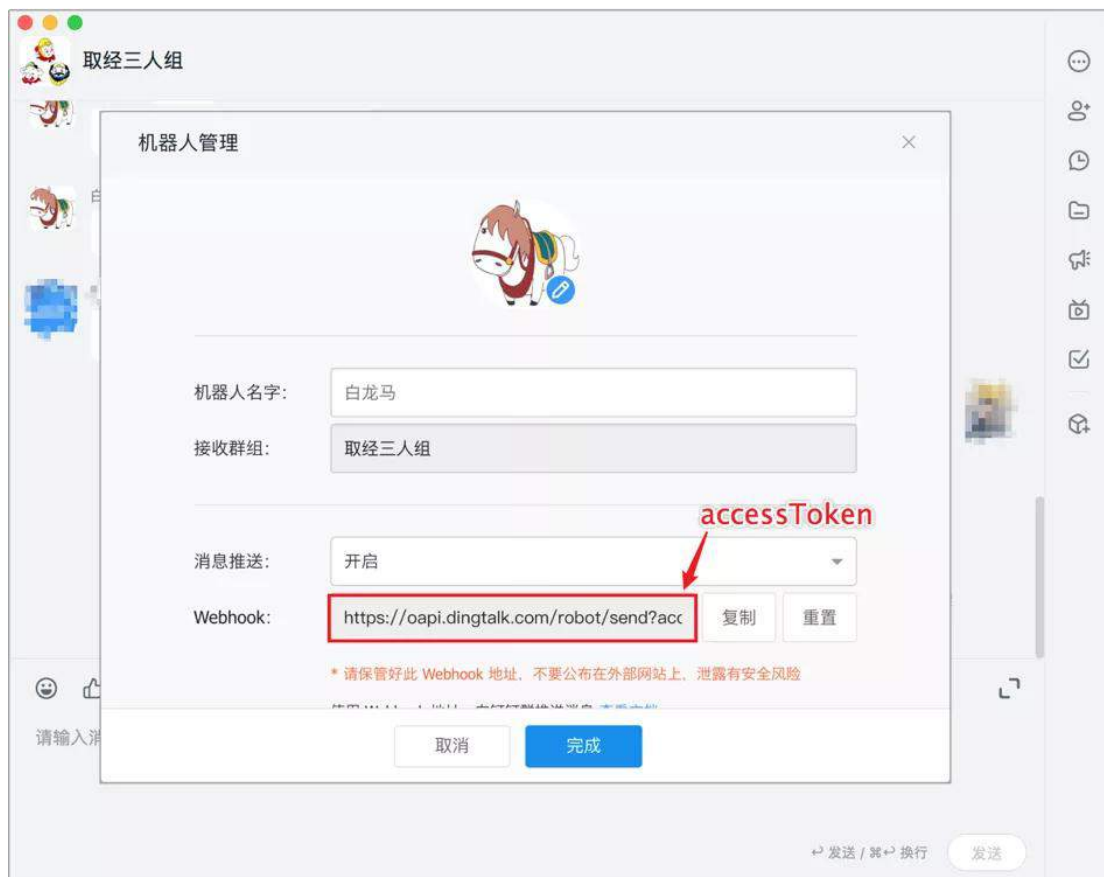
  const req = https.request(options, (res) => {
    res.setEncoding('utf8');
    res.on('data', (chunk) => {});
    res.on('end', () => {
      callback(null, 'success');
    });
  });

  req.on('error', (e) => {
    callback(e);
  });
};
```

```
req.write(postData);  
req.end();  
};
```

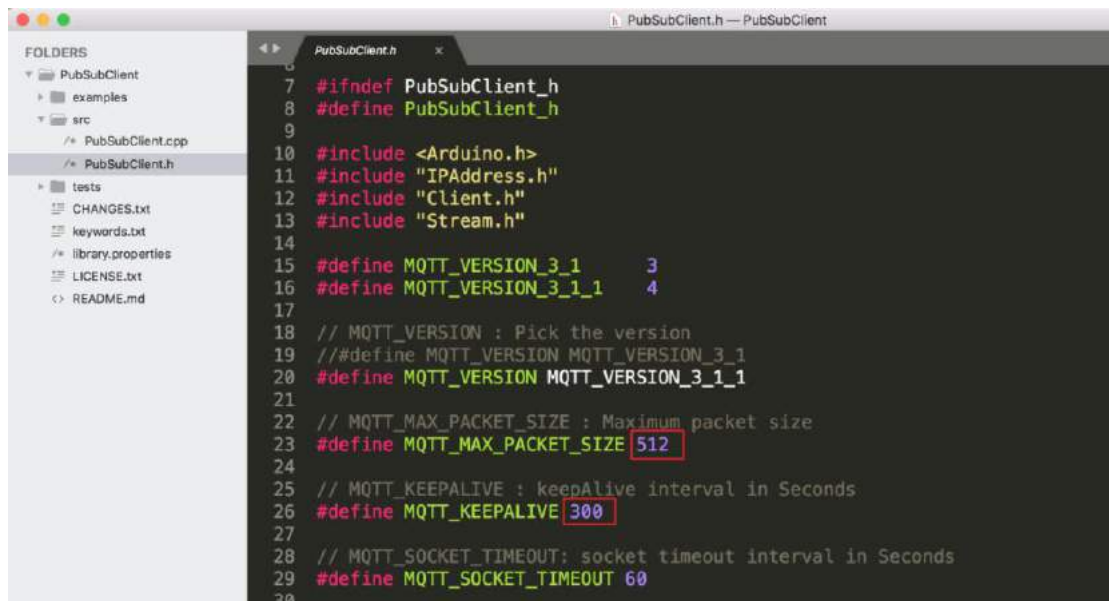
4. 钉钉群机器人

打开摸鱼群, 群管理>智能群助手, 添加自定义机器人, 获取到消息推送的 Webhook 地址, 配置到函数计算脚本中。



四、ESP-EYE 开发板 上云

设备上云基于 MQTT 协议, 因此我们引入 PubSubClient 库, 并修改 MQTT 连接的默认参数, 具体如下:



设备身份三元组信息和连接参数：

```

/* 设备的三元组信息 */
#define PRODUCT_KEY    "替换 PRODUCT_KEY"
#define DEVICE_NAME    "替换 DEVICE_NAME"
#define DEVICE_SECRET  "替换 DEVICE_SECRET"
#define REGION_ID      "cn-shanghai"
/* IoT 物联网平台 Endpoint 域名和端口号 */
#define MQTT_SERVER    PRODUCT_KEY ".iot-as-mqtt." REGION_ID ".aliyuncs.com"
#define MQTT_PORT      1883
#define MQTT_USERNAME  DEVICE_NAME "&" PRODUCT_KEY
//用于身份验证的 MQTT_PASSWD 和 CLIENT_ID
#define CONTENT_STR_FORMAT  "clientIdesp32deviceName" DEVICE_NAME "productKey" PRODUCT_KEY "timestamp%d"
char CLIENT_ID[80] = {"\0"};
char * MQTT_PASSWD;

WiFiClient espClient;
PubSubClient client(espClient);

```

物模型通信的 Topic 和 Payload 模板：

```

/* topic 和 payload */
#define PROP_POST_TOPIC "/sys/" PRODUCT_KEY "/" DEVICE_NAME "/thing/event/property/post"
#define BODY_FORMAT "{\"id\":\"1234\",\"version\":\"1.0\",\"method\":\"thing.event.property.post\",\"params\":{\"bossId\":%d}}"

```

当有老板人脸匹配成功后，及时上报到云端：

```

char jsonBuf[128];
sprintf(jsonBuf, BODY_FORMAT, face_id);

if (client.connected()) {
    boolean d = client.publish(PROP_POST_TOPIC, jsonBuf);
    Serial.print("publish:1=成功, 0=失败, Code=");
    Serial.println(d);
}

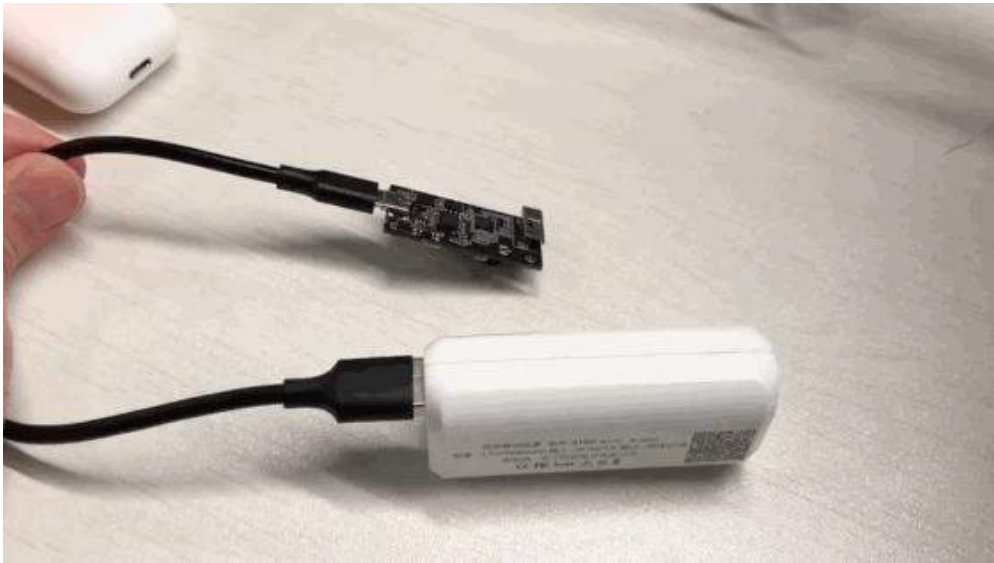
```

最后，我们把完整版程序烧录到 ESP-EYE 开发板上。



五、部署监控

辛苦了一整天，大功告成！让我们找个隐蔽的角落，部署起来，守株待老板们！



[点击查看动图演示](#)

终于，老板来了！

而且是老板三连，是来发年终奖了吧！



IoT+DB + DataV 搭建实时环境监控大屏

作者 | 苏堤嘉木

今天给大家带来基于阿里云 IoT 物联网平台 + Tablestore 表格存储数据库 + DataV 大屏 三大云产品组合搭建实时环境监控大屏的开发实战。

少啰嗦，先看效果。



部署后效果

一、技术架构

我们在室内每层部署 4 个温湿度传感器，实时采集数据，每 10 秒发送到阿里云 IoT 物联网平台，通过规则引擎写入 表格存储 Tablestore 数据库。在 DataV 大屏工作台，创建可视化大屏，实时展示室内温湿度变化曲线。

技术架构如下：



二、IoT 物联网开发

1. 创建产品和注册设备

首先，我们登录 IoT 物联网平台的控制台，创建产品**温湿度环境监测器**，并在功能定义中添加**温度**和**湿度**两个属性，如下图：



然后，我们在产品下注册一个设备，获取**设备身份证书**，用于设备和 IoT 云平台建立 MQTT 长连接时的身份认证。



2. 配置云产品流转

1.4.配置规则引擎，实时流转数据到 TSDB 中：



编写数据处理 SQL：

编写 SQL



规则查询语句:

复制语句

```
SELECT deviceName() as deviceName, timestamp('yyyy-MM-dd HH:mm:ss') as time, attribute('floor') as floor, items.temperature.value as temperature, items.humidity.value as humidity
FROM "/a1kRdXDgN0v/+/thing/event/property/post"
WHERE
```

● 字段

```
deviceName() as deviceName, timestamp('yyyy-MM-dd HH:mm:ss')
```

● Topic ?

物模型数据上报



温湿度环境监测器



全部设备 (+)



thing/event/property/post

IoT物联网技术

完整 SQL 参考:

```
SELECT
deviceName() as deviceName,
timestamp('yyyy-MM-dd HH:mm:ss') as time,
attribute('floor') as floor,
items.temperature.value as temperature,
items.humidity.value as humidity

FROM
"/a1kRdXDgN0v/+/thing/event/property/post"
```

数据转发:

选择操作 ?

存储到表格存储 (Tablestore)

* 地域

华东 1

注意：当前所选地域与帐号地域不一致，跨地域数据流转容易受到网络影响，建议配置同一地域内的流转规则。

* 实例

iotMsg

创建实例

* 数据表

iot_thermometer_data

创建数据表

* 主键

deviceName键 \${deviceName} ?

* 主键

time键 \${time} ?

* 角色

AliyunIOTAccessingOTSRole

创建RAM角色

IoT物联网技术

3. 设备端程序脚本

我们以 Node.js 脚本来模拟设备上报温度和湿度，代码如下：

```
// 依赖 mqtt 库
const mqtt = require('aliyun-iot-mqtt');
// 设备身份
var options = {
```

```
    productKey: "设备 productKey",
    deviceName: "设备 deviceName",
    deviceSecret: "设备 deviceSecret",
    regionId: "cn-shanghai"
  };

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

//模拟 设备 上报数据（原始报文）
setInterval(function() {
  client.publish(
    `/sys/${options.productKey}/${options.deviceName}/thing/event/property/post`
    , getPostData()
  );
}, 10 * 1000);

// 模拟 温湿度
function getPostData() {

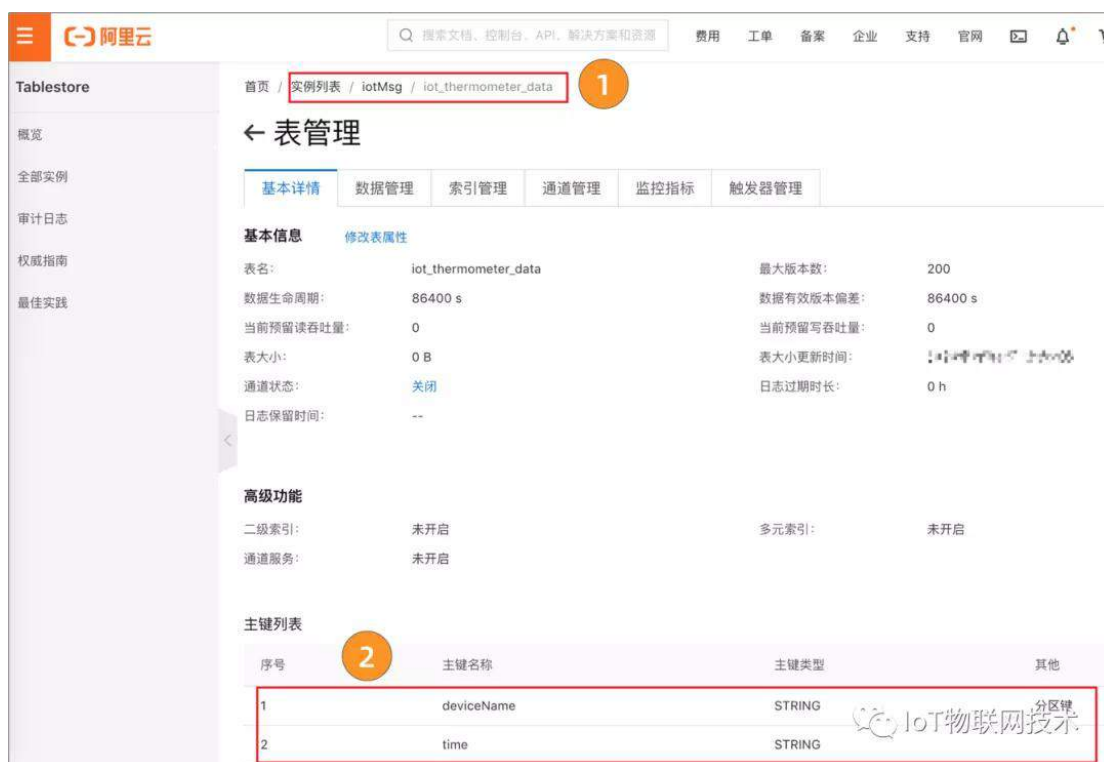
  const payload = {
    id: Date.now(),
    version: "1.0",
    params: {
      temperature: 10+Math.floor(Math.random() * Math.floor(50)),
      humidity: 10+Math.floor(Math.random() * Math.floor(50))
    },
    method: "thing.event.property.post"
  }

  console.log("payload=[ " + payload + " ]")
  return JSON.stringify(payload);
}
```

三、表格存储 Tablestore

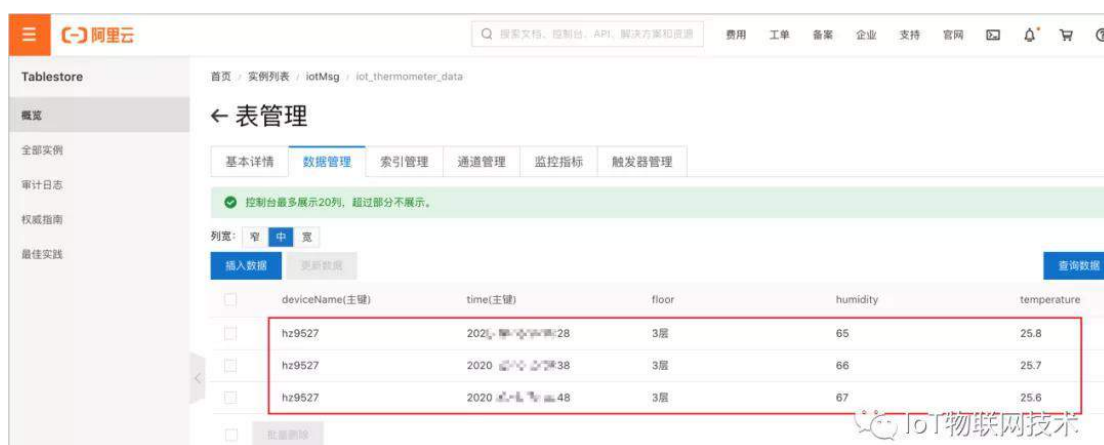
1. 创建表格存储数据表

我们在表格存储控制台，创建数据库 `iotMsg`，并创建 `iot_thermometer_data` 数据表，其中以 `deviceName` 和 `time` 为主键列，如下图：



2. 实时存储的环境数据

当设备启动后，我们会在数据库中看到实时的温湿度数据，如下图：



四、DataV 可视化大屏

设备实时上报数据通过 IoT 物联网平台实时流转到表格存储数据库后，接下来的工作就是把数据可视化呈现出来，这时就用到了 DataV 可视化大屏云产品。

1. 添加数据源

我们进入 DataV 控制台，在我的数据下，添加数据源，输入表格存储实例 `iot_data` 的访问路径和 AccessKey 身份信息，如下图：



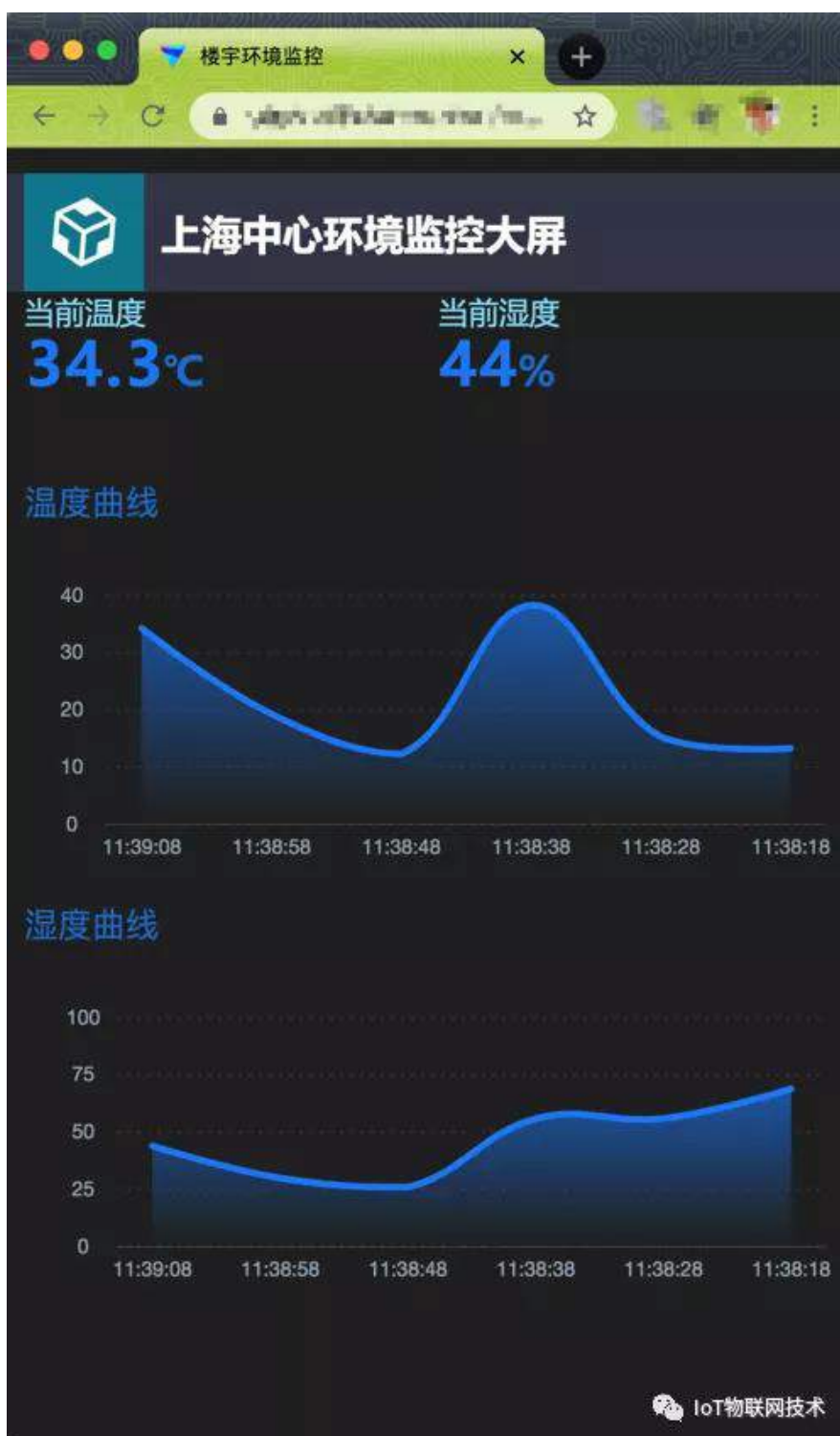
2. 搭建大屏页面

配置完数据源后，在 DataV 控制的我的可视化下面创建一个大屏，根据业务需求添加可视化组件，并为每个组件配置相关的数据源，以及过滤器用来适配数据格式，如下图：



3. 发布实时监控大屏

当我们完成组件的数据源配置后，就可以发布可视化大屏了。发布后，用户可以通过浏览器访问，如下图：



IoT+TSDB+Quick BI 环境监控

作者 | 苏堤嘉木

今天给大家带来基于阿里云 IoT 物联网平台 + TSDB 时序时空数据库 + Quick BI 报表三大云产品组合实现楼宇环境监控端到端开发实战。

少啰嗦，先看效果。



一、技术架构

本次 IoT 物联网开发实战我们在室内部署 4 个温湿度传感器，实时采集数据，每 10 秒发送到阿里云 IoT 物联网平台，通过规则引擎写入 TSDB 时序数据库。在 Quick BI 工作台，创建数据报表以分钟维度展示室内温湿度变化曲线。

技术架构如下：



二、物联网平台开发

1. 免费开通阿里云 IoT 物联网云服务：

<https://www.aliyun.com/product/iot-deviceconnect>



2. 创建产品**室内温湿度计器**，选择自定义品类，直连设备，定义物模型，包含 2 个属性温度，湿度：

物联网平台 / 设备管理 / 产品 / 产品详情

← 室内温湿度计器

ProductKey [a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v2w3x4y5z6](#) 复制 ProductSecret ***** 查看

设备数 5 前往管理

产品信息 Topic 类列表 **功能定义** 数据解析 服务端订阅 设备开发

1 当前展示的是已发布到线上的功能定义，如需修改，请点击 [编辑草稿](#)

物模型 TSL 生成设备端代码

功能类型	功能名称 (全部) ▾	标识符	数据类型	数据定义
属性	温度 自定义	temperature	float (单精度浮点型)	取值范围: -20 ~ 80
属性	湿度 自定义	humidity	float (单精度浮点型)	取值范围: 0 ~ 100

3. 注册设备，获取身份三元组。

物联网平台 / 设备管理 / 设备 / 设备详情

← Mj12u4wma9y4g230A6DR 离线

产品 室内温湿度计器 [查看](#) DeviceSecret ***** [查看](#)

ProductKey [a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v2w3x4y5z6](#) 复制

设备信息 Topic 列表 物模型数据 设备影子 文件管理 日志服务 在线调试 分组

设备信息

产品名称	室内温湿度计器	ProductKey	a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6q7r8s9t0u1v2w3x4y5z6 复制	区域	华东2 (上海)
节点类型	设备	DeviceName	Mj12u4wma9y4g230A6DR 复制	认证方式	设备密钥
备注名称	编辑	IP地址	192.168.1.100	固件版本	0.1.0
添加时间	2019/12/12 14:30:00	激活时间	2020/02/10 10:00:00	最后上线时间	2020/08/10 10:00:00
当前状态	离线	实时延迟	测试	设备本地日志上报	已关闭 ☐

设备扩展信息

SDK 语言	Java	版本号	1.0.0	模组商	-
模组信息	-				

4. 配置规则引擎，实时流转数据到 TSDB 中：

物联网平台 / 规则引擎 / 云产品流转 / 数据流转规则

← 温湿度转TSDB

数据格式JSON规则ID

规则描述-

处理数据

SQL 语法说明

规则查询语句:

SELECT deviceName() deviceName,timestamp() timestamp,items.temperature.value temperature,items.humidity.value humidity FROM "/sys/{productKey}/{deviceName}/thing/event/property/post"

转发数据

数据目的地

存储到时序时空数据库 (TSDB) 中: ts-uid

转发错误操作数据

数据目的地

5. 完成设备端开发，实时上报温湿度数据。

我们以 Node.js 脚本来模拟设备上报，代码如下：

```
// 依赖 mqtt 库
const mqtt = require('aliyun-iot-mqtt');
// 设备身份
var options = {
  productKey: "device productKey",
  deviceName: "device deviceName",
  deviceSecret: "device deviceSecret",
  regionId: "cn-shanghai"};

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

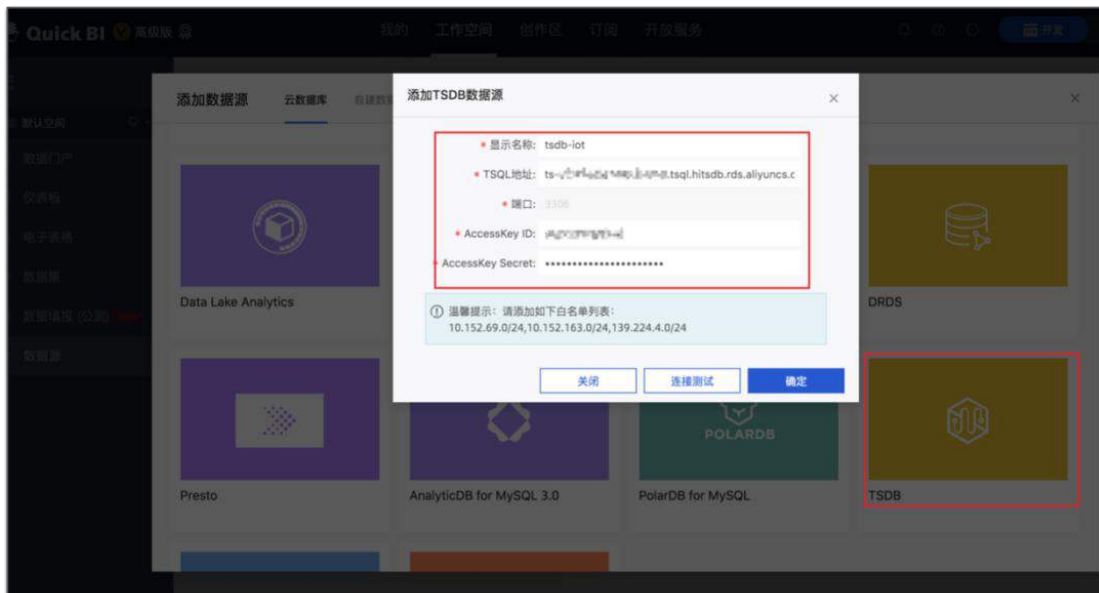
//模拟 设备 上报数据（原始报文）
setInterval(function() {
  client.publish(
    `/sys/${options.productKey}/${options.deviceName}/thing/event/property/post`
    , getPostData()
  )
}, 1000);
```


2. IoT 设备数据写入 TSDB 的记录：



四、Quick BI

1. 开通 Quick BI 服务，添加数据源，输入 TSDB 连接参数。



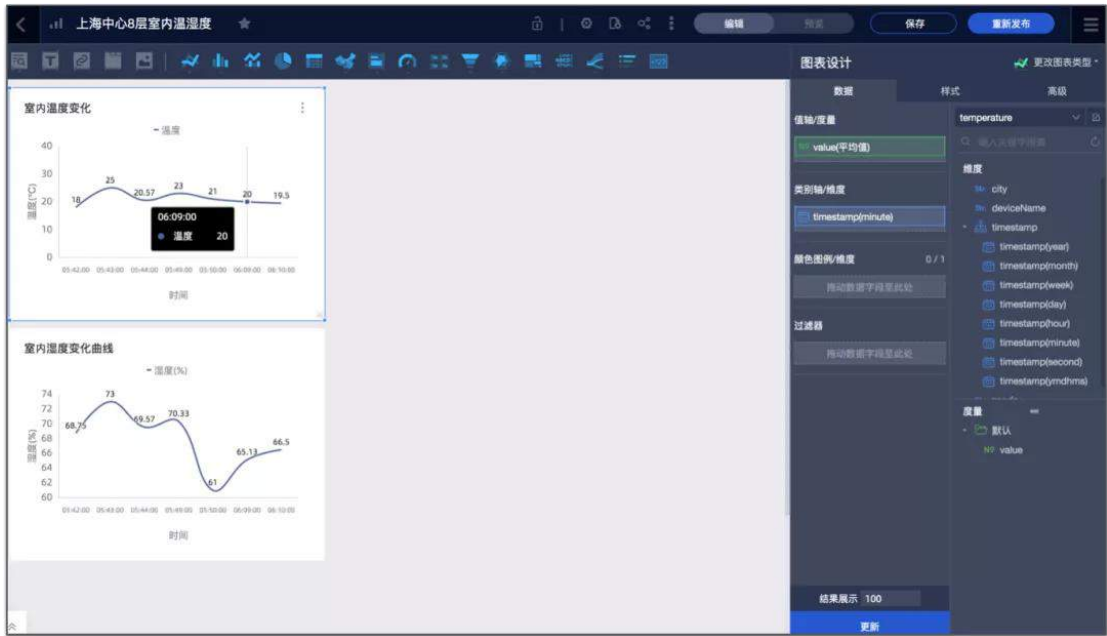
数据源添加成功：



2. 基于数据源的温度和湿度指标，创建数据集。



3. 创建数据仪表盘，并根据业务需求编辑图表。



4. 发布仪表板。

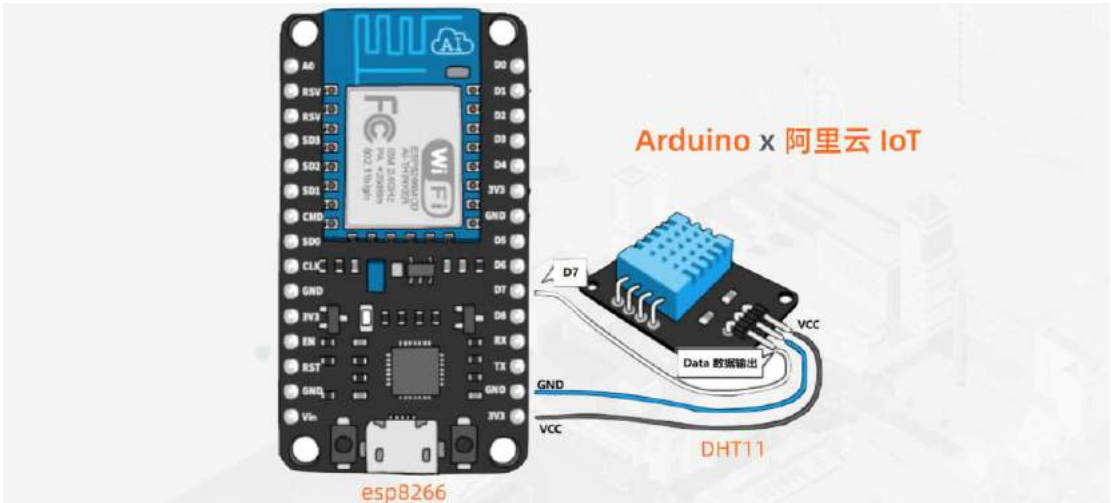


5. 在浏览器中查看楼宇环境监控报表。



20 元自制 Arduino 环境监测仪

作者 | 苏堤嘉木



一、 硬件准备

今天我们在只有 20 元预算前提下，带领大家完成一次 IoT 物联网开发之旅！

感谢万能的淘宝，让我们能采购到 esp8266 开发板，dht11 温湿度传感器：

名称	图片
NodeMCU(ESP8266) 主板	
DHT11 温湿度传感器	
母对母排线	

二、架构方案

我们通过 DHT11 采集温度，湿度数据，通过 MQTT 协议上报到阿里云 IoT 物联网平台，并通过规则引擎转发到表格存储 OTS 数据库中，整体技术方案如下：



三、创建产品和注册设备

我们登录阿里云 IoT 物联网平台控制台，创建产品温湿度计。



在温湿度计产品详情的 Topic 类列表可以看到系统默认创建的 Topic，这就是我们用来上报温湿度数据的 Topic。

物联网平台 / 设备管理 / 产品 / 产品详情

← 温湿度计

ProductKey: a%4%9%8%0%v 复制

ProductSecret: ***** 查看

设备数: 2 前往管理

产品信息

Topic 类列表

功能定义

数据解析

服务端订阅

设备开发

基础通信 Topic

物模型通信 Topic

自定义 Topic

定义 Topic 类

自定义 Topic	操作权限	描述
/t/a%4%9%8%0%v/\${deviceName}/user/update	发布	上报数据的Topic

接下来, 我基于温湿度计产品注册一个设备 hz9527, 并获取到设备身份三元组。

物联网平台 / 设备管理 / 设备 / 设备详情

← hz9527 离线

产品: 温湿度计 查看

DeviceSecret: ***** 查看

← 身份三元组

ProductKey: a%4%9%8%0%v 复制

设备信息

Topic 列表

物模型数据

设备影子

文件管理

日志服务

在线调试

设备信息

产品名称	温湿度计	ProductKey	a%4%9%8%0%v 复制	区域	华东2 (上海)
节点类型	设备	DeviceName	hz9527 复制	认证方式	设备密钥
备注名称	编辑	IP地址	192.168.1.100	固件版本	-
添加时间	2020/05/06 16:23	激活时间	2020/05/06 16:13.999	最后上线时间	2020/05/06 16:34.106
当前状态	离线	实时延迟	测试	设备本地日志上报	已关闭

四、配置数据流转 规则引擎

我们在表格存储控制台, 预先创建数据库实例 `iotMsg` 和 `iot_thermometer_data` 表, 如下图:

首页 / 实例列表 / iotMsg / iot_thermometer_data

← 表管理

基本详情

数据管理

索引管理

通道管理

监控指标

触发器管理

基本信息

修改表属性

表名:

iot_thermometer_data

最大版本数:

200

数据生命周期:

86400 s

数据有效版本偏差:

86400 s

当前预留读写容量:

0

当前预留写吞吐量:

0

表大小:

108 B

表大小更新时间:

2020-09-27 15:28:00

通道状态:

关闭

日志过期时长:

0 h

日志保留时间:

--

高级功能

二级索引:

未开启

多元索引:

未开启

通道服务:

未开启

主键列表

序号	主键名称	主键类型	其他
1	deviceName	STRING	分区键
2	time	STRING	

然后，我们回到 IoT 物联网平台控制台，配置数据流转规则，把设备上报数据存储到已有的表格存储数据库实例 iotMsg 中的 iot_thermometer_data 表中，规则引擎配置如下：

物联网平台

规则引擎

云产品流转

数据流转规则

← 温湿度数据To表格存储OTS

数据格式

JSON

规则 ID

7425

规则描述

-

处理数据

规则查询语句:

SELECT *, deviceName() as deviceName,timestamp() as time, timestamp('yyyy-MM-dd HH:mm:ss') as date_time FROM "/a+&id=0v/+ /user/update"

转发数据

数据目的地

该操作将数据插入到表格存储 (Table Store) 中: iotMsg

转发错误操作数据

数据目的地

完整的数据处理 SQL：

```
SELECT *,
deviceName() as deviceName,
timestamp() as time,
timestamp('yyyy-MM-dd HH:mm:ss') as date_time
FROM "/a1k3547Gr0v/+user/update"
```

数据转发配置如下：

The screenshot shows the '温湿度数据转发配置' (Temperature and Humidity Data Forwarding Configuration) page in the Alibaba Cloud IoT console. The page is divided into several sections:

- 数据格式 (Data Format):** Set to JSON.
- 规则描述 (Rule Description):** Empty.
- 处理数据 (Process Data):** Shows the SQL query: `SELECT *, deviceName() as deviceName, timestamp() as time, timestamp('yyyy-MM-dd HH:mm:ss') as date_time FROM "/a1k3547Gr0v/+user/update"`.
- 转发数据 (Forward Data):** This section is highlighted and contains the following configuration:
 - 选择操作 (Select Action):** 存储到表格存储 (Table Store)
 - 地域 (Region):** 华东 1
 - 实例 (Instance):** iotMsg
 - 数据表 (Data Table):** iot_thermometer_data
 - 主键 (Primary Key):** Two keys are defined: `deviceName` with value `${deviceName}` and `time` with value `${time}`.
 - 角色 (Role):** AliyunIoTAccessingOTSRole
- 转发错误操作数据 (Forward Error Operation Data):** Empty.

设备和云端通信 Topic 和 Payload 如下：

Topic:
/sys/a1k823sJ0v/hz9527/thing/event/property/post

Payload:
{
 "id":1596087445906,
 "params":{
 "temperature":23,

```
"humidity":65
},
"method":"thing.event.property.post"
}
Topic:/a1k823sJ0v/hz9527/user/update
Payload:
{
  "temperature":23,
  "humidity":65
}
```

五、硬件开发

Arduino 开发依赖 C 库，如下：

```
#include <ESP8266WiFi.h>
/* PubSubClient 2.4.0 */
#include <PubSubClient.h>
/* ArduinoJson 5.13.4 */
#include <ArduinoJson.h>
/* DHT sensor library 1.3.0 */
#include "DHT.h"
/* Crypto 0.2.0 */
#include "SHA256.h"
```

设备和云端通过 PubSubClient 建立 MQTT 连接代码：

```
/* 连接 WiFi 之后，连接 MQTT 服务器 */
client.setServer(MQTT_SERVER, MQTT_PORT);
client.setCallback(callback);
client.connect(CLIENT_ID, MQTT_USERNAME, MQTT_PASSWD)
```

设备读取传感器数据，并通过 MQTT 通道发送到云端：

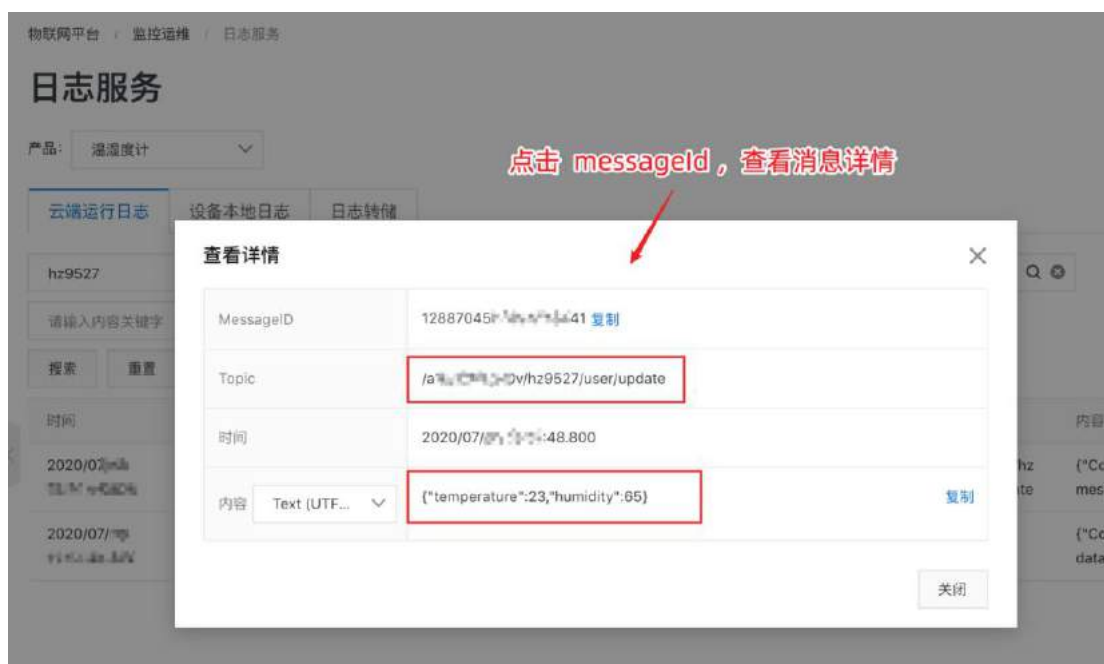
```
void loop() {  
  // 从传感器获取温度，湿度数据  
  float temperature = dht.readTemperature();  
  float humidity = dht.readHumidity();  
  char jsonBuf[128];  
  sprintf(jsonBuf, BODY_FORMAT, temperature, humidity);  
  
  // 通过 MQTT 发送数据上云  
  if (client.connected()) {  
    boolean d = client.publish(PROP_POST_TOPIC, jsonBuf);  
  }  
  
  client.loop();  
  // delay  
  delay(DELAY_TIME);  
}
```

使用 Arduino IDE 烧录程序到 esp8266 开发板：



六、联机运行

烧录完成后，程序启动，我们就可以在 IoT 控制台的日志服务中查看到上报的数据，如下图：



观察日志服务中的记录，可以看到消息上报和数据流转的过程，如下图：



在表格存储控制台，可以看到存储在 `iot_thermometer_data` 里的数据



七、物模型开发

物模型指将物理空间中的实体数字化，并在云端构建该实体的数据模型。物模型描述产品是什么、能做什么、可以对外提供哪些服务。

功能类型	说明
属性（Property）	一般用于描述设备运行时的状态，如环境监测设备所读取的当前环境温度等。属性支持GET和SET请求方式。应用系统可发起对属性的读取和设置请求。
服务（Service）	设备可被外部调用的能力或方法，可设置输入参数和输出参数。相比于属性，服务可通过一条指令实现更复杂的业务逻辑，如执行某项特定的任务。
事件（Event）	设备运行时的事件。事件一般包含需要被外部感知和处理的通知信息，可包含多个输出参数。如，某项任务完成的信息，或者设备发生故障或告警时的温度等，事件可以被订阅和推送。

我们在**物联网平台控制台**，**产品详情**页面的**功能定义**，添加**温度**和**湿度**两个属性，如

物联网平台 / 设备管理 / 产品 / 产品详情

← 温湿度计

ProductKey a11111111111111111111 复制 ProductSecret ***** 查看

设备数 2 前往管理

产品信息

Topic 类列表

功能定义

数据解析

服务端订阅

设备开发

当前展示的是已发布到线上的功能定义，如需修改，请点击 编辑草稿

物模型 TSL

生成设备端代码

功能类型	功能名称 (全部)	标识符	数据类型	数据定义	操作
属性	湿度 自定义	humidity	int32 (整数型)	取值范围: 1 ~ 100	查看
属性	温度 自定义	temperature	float (单精度浮点型)	取值范围: 0 ~ 50	查看

物模型属性上报的通信 Topic，如下图：

物联网平台 / 设备管理 / 产品 / 产品详情

← 温湿度计

ProductKey a11111111111111111111 复制 ProductSecret ***** 查看

设备数 2 前往管理

产品信息

Topic 类列表

功能定义

数据解析

服务端订阅

设备开发

基础通信 Topic

物模型通信 Topic

自定义 Topic

物模型通信 Topic 列表

功能	Topic类	操作权限	描述
属性上报	/sys/a11111111111111111111v/\${deviceName}/thing/event/property/post	发布	设备属性上报
	/sys/a11111111111111111111v/\${deviceName}/thing/event/property/post_reply	订阅	云端响应属性上报
属性设置	/sys/a11111111111111111111v/\${deviceName}/thing/service/property/set	订阅	设备属性设置
事件上报	/sys/a11111111111111111111v/\${deviceName}/thing/event/\${tsl.event.identifer}/post	发布	设备事件上报
	/sys/a11111111111111111111v/\${deviceName}/thing/event/\${tsl.event.identifer}/post...	订阅	云端响应事件上报

物模型通信 Topic 和 Payload 如下：

```

Topic:
/sys/a1k823sJ0v/hz9527/thing/event/property/post

Payload:
{
  "id":1596087445906,
  "params":{
    "temperature":23,
    "humidity":65
  },
  "method":"thing.event.property.post"
}

```

我们修改 Arduino 查询中 Topic 和 Payload 后，重新烧录，设备上报数据后，在控制台可以查看日志，如下图：

物联网平台 / 监控运维 / 日志服务

日志服务

产品: 温湿度计

云端运行日志 | 设备本地日志 | 日志转储

hz9527 0a3027d915960874 请输入 MessageID

请输入内容关键字 全部状态 1 小时

搜索 重置

时间	TraceID	MessageID	DeviceName	业务类型(全部)	操作	内容	状态
2020/07/15 10:16:16	0a3027d915960874	128871027 2789529...	hz9527	物模型上报	/sys/a1k823sJ0v/hz9527/t...	-	200
2020/07/15 10:16:16	0a3027d915960874	-	hz9527	数据存储	store	-	200
2020/07/15 10:16:16	0a3027d915960874	-	hz9527	物模型上报	Check	-	200
2020/07/15 10:16:35	0a3027d915960874	128871027 2986661...	hz9527	云到设备消息	/sys/a1k823sJ0v/hz9527/t...	{"Content":"Publish message to..."}	200
2020/07/15 10:16:35	0a3027d915960874	128871027 2789529...	hz9527	设备到云消息	/sys/a1k823sJ0v/hz9527/t...	{"Content":"Publish message to..."}	200

在设备详情页面，物模型数据 Tab 也可以实时看到设备运行状态，如下图：

物联网平台 / 设备管理 / 设备 / 设备详情

← hz9527 离线

产品 温湿度计 查看 DeviceSecret ***** 查看
ProductKey a1*****V0v 复制

设备信息	Topic 列表	物模型数据	设备影子	文件管理	日志服务	在线调试
------	----------	-------	------	------	------	------

运行状态	事件管理	服务调用
------	------	------

实时刷新 ☐

湿度 查看数据 65 % ⓘ 2020/07/27 17:26.816	温度 查看数据 23 °C ⓘ 2020/07/27 17:26.816
---	--

IoT 二维码收款播报音箱

作者 | 苏堤嘉木



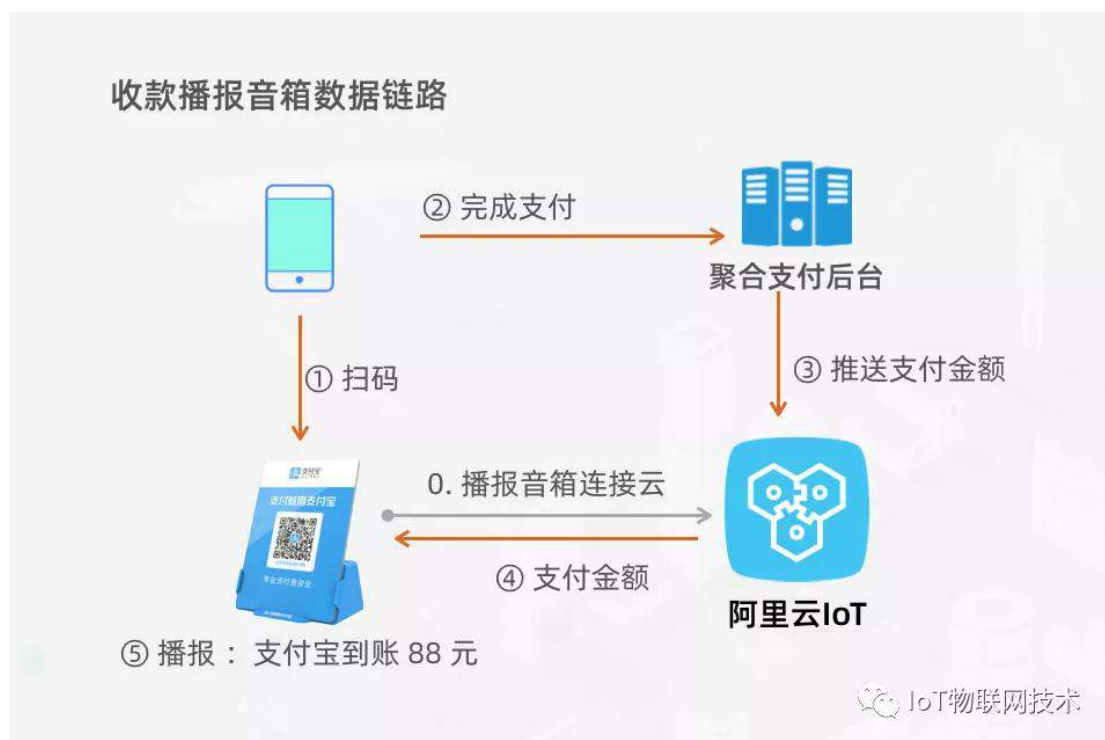
《梦粱录》记载，南宋临安城，坊市制度被打破，宵禁制逐步瓦解。华灯初上，市民百姓不再待在家中睡觉，而是开始丰富的夜生活，**清河坊、羊坝头、官巷口、众安桥**一带“与日间无异”，**酒楼、歌馆和瓦子**(游乐场)分布甚密，十里长街。

《武林旧事》记载一则故事：南宋临安太和楼有三百个包厢，每日可接待客人三千名。入夜，灯烛辉煌,人流如潮,摩肩接踵。除市井百姓之外，常有夜鬼慕名而来享受人间美食：**百味羹、莲子头、鹅鸭包、镜面粉、鱼桐皮面**等。他们用铜钱结账，翌日，那些铜钱都变成了一堆灰烬。于是，店家便在收到铜钱后，放在水里面检验，如果铜钱漂在水上，那么买东西的人**就是鬼**。

如今，支付宝和微信大力推广手机二维码收款，**假币消失殆尽**。扫码付款之初，消费者面对一柜多码，误操作频繁，体验欠佳。商家手机 App 收款消息推送不及时，逃单漏单时有发生；多个付款渠道每日对账繁琐。

收钱吧，新大陆，哆啦宝等聚合支付公司顺势而生：**一柜一码**，基于 IoT 物联网和 TTS 语音合成技术的到账**实时语音播报**，每日对账报表汇总，让收款顿时轻松起来。

聚合支付播报音箱原理：



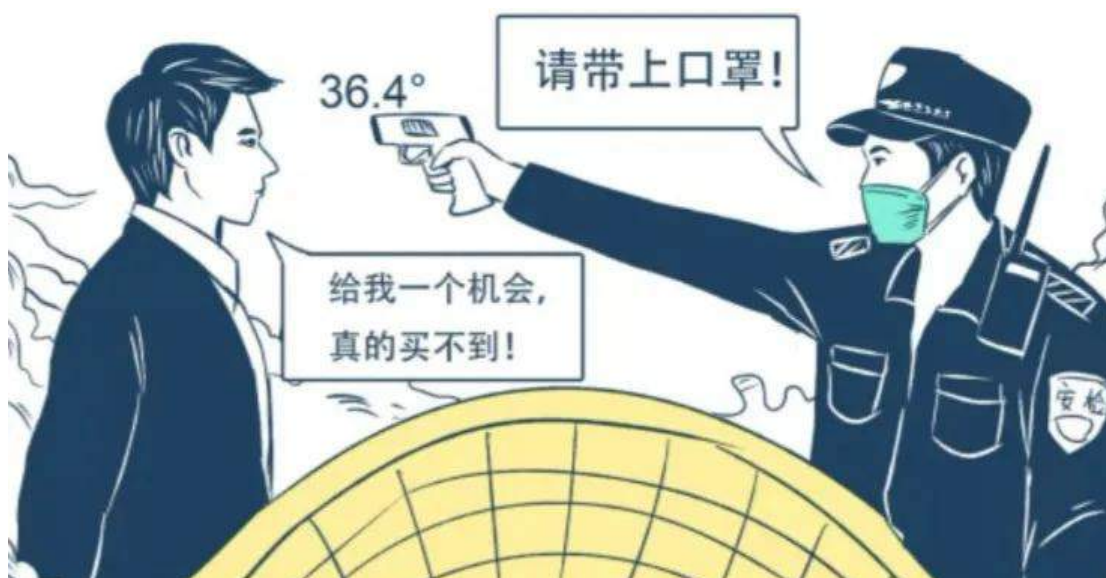
相比支付宝和微信，第三方聚合支付有以下优势：

- 聚合多种支付工具，一键扫码支付。
- 实时语音播报功能，防止逃单漏单。
- 无需收银员操作，提高收银效率。
- 拥有对账功能，查询每日交易数据汇总。

阿里云 IoT 物联网平台，提供基于 MQTT 协议的 PUB/SUB 的安全、可靠、低延时的异步通信，百万级消息并发性能，毫秒级实时触达能力。同时制定了一套同步请求和响应机制(RRPC)，使得聚合支付业务服务端能同步获取设备端语音播报的响应结果。

IoT 智能手持测温枪开发

作者 | 苏堤嘉木

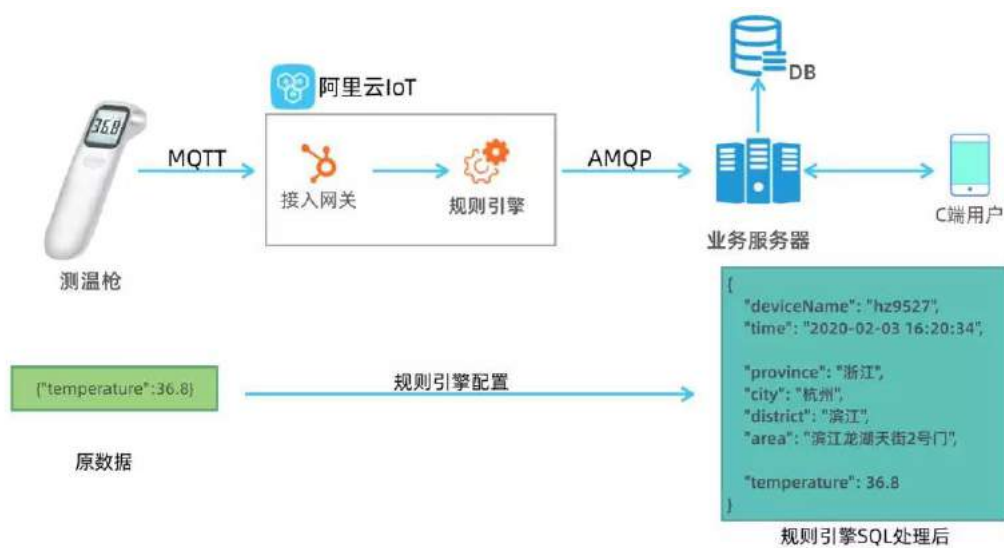


一、IoT 助力智能化体温采集

随着新型冠状病毒疫情发展，社区居家隔离成为有效手段，而体温排查是社区工作的重中之重！借助 IoT 物联网技术可以方便的完成居民体温实时监控和历史数据的完整追溯。

二、技术架构方案

基于稳定性，高并发，低时延的考量我们选择阿里云 IoT 物联网平台搭建整套系统。首先手持测温枪通过蓝牙连接到 DTU 模块，DTU 模块以 MQTT 协议接入物联网平台。数据上云后，通过规则引擎流转服务端订阅的 AMQP 消费组，实时推送到我们业务服务器。管理人员使用手机小程序即可实时看到出入人员的体温数据。



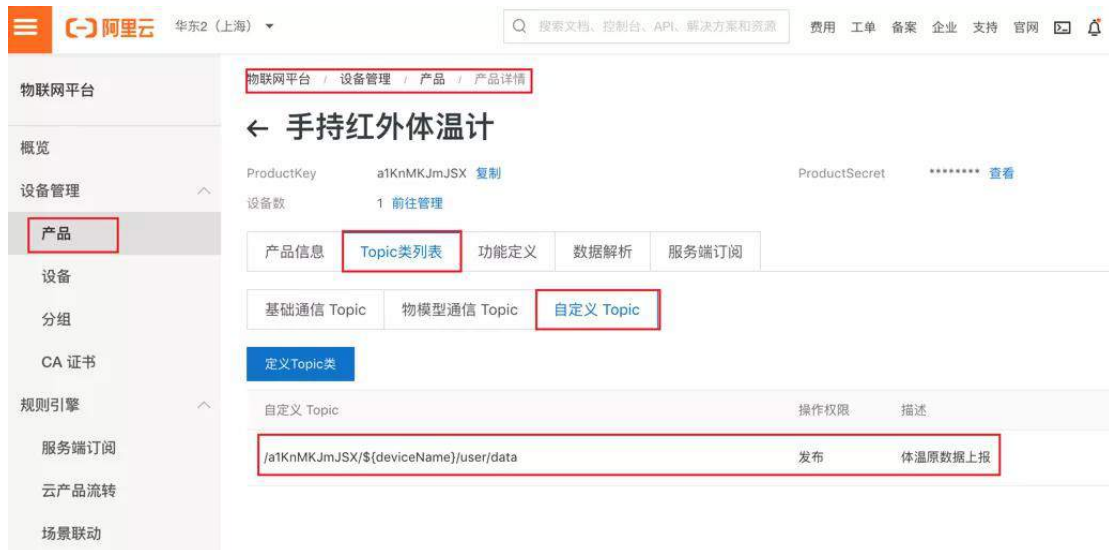
三、云端开发

1. 创建产品

进入物联网平台控制台，创建产品。



在产品详情 Topic 列表，增加用于数据传输的 Topic，如下：



2. 注册设备

产品定义好后，我们基于这个产品创建一个具体设备，获取到设备身份三元组。



3. 创建消费组

接下来，我们要在服务端订阅创建用来接收数据的消费组，参看下图：



4. 配置数据流转的规则

最后，我们通过规则引，把设备上报的数据做业务处理后，流转到我们服务器的消费组，从而实现企业自己的设备采集的业务数据到达企业自己的后台服务器的流转过程。



四、设备开发

在完成了云上控制台的配置工作后，我们要做的就是设备端业务开发。这里我们在 Mac 上用 nodejs 脚本模拟设备业务行为，设备 MQTT 连接，数据上报。

完整代码如下：

```
// 引入依赖 mqtt 库，或自己实现
const mqtt = require('aliyun-iot-mqtt');
// 设备身份
var options = {
  productKey: "设备 pk",
  deviceName: "设备 dn",
  deviceSecret: "设备 ds",
  regionId: "cn-shanghai"
};

// 1.建立连接
const client = mqtt.getAliyunIotMqttClient(options);

// 2.设备接收云端指令数据
client.on('message', function(topic, message) {
  console.log("topic " + topic)
  console.log("message " + message)
})

// 3. 模拟设备 上报数据（原始报文）
setInterval(function() {
  client.publish(`${options.productKey}/${options.deviceName}/user/data`, getPostData(), {qs:1});
}, 1000);

// 模拟 设备原有报文格式
function getPostData() {
  let payload = {
    temperature: Math.floor((Math.random() * 20) + 10)
  };

  console.log("payload=[ " + payload+" ]")
  return JSON.stringify(payload);
}
```

五、服务端开发

服务端我们以 Java 为例，演示如何接收 IoT 平台推送过来的设备上报数据。

参考服务端订阅 AMQP 文档：

https://help.aliyun.com/document_detail/143601.html

核心代码如下：

```
public class AMQPClient {

    private final static Logger logger = LoggerFactory.getLogger(AMQPClient.class);
    //消费组配置参数
    private static String accessKey = "阿里云账号 ak";
    private static String accessSecret = "阿里云账号 as";
    private static String consumerGroupId = "服务端订阅消费组 ID";
    private static String aliUID = "替换你的阿里云账号 UID";

    public static void main(String[] args) throws Exception {

        long timeStamp = System.currentTimeMillis();
        //签名方法
        String signMethod = "hmacsha1";
        //控制台服务端订阅中消费组状态页客户端 ID 一栏将显示 clientId 参数。
        //建议使用机器 UUID、MAC 地址、IP 等唯一标识等作为 clientId。便于您区分识别不同的客户端。
        String clientId = "ecs_" + System.currentTimeMillis();

        //UserName 组装
        String userName = clientId + "|authMode=aksign"
            + ",signMethod=" + signMethod
            + ",timestamp=" + timeStamp
            + ",authId=" + accessKey
            + ",consumerGroupId=" + consumerGroupId
            + "|";
        //password 组装
        String signContent = "authId=" + accessKey + "&timestamp=" + timeStamp;
        String password = doSign(signContent, accessSecret, signMethod);
        //按照 qpid-jms 的规范，组装连接 URL。
        String connectionUrl = "failover:(amqps://"+aliUID+".iot-amqp.cn-shanghai.aliyuncs.com:5671?amqp.idleTimeout=80000)"
            + "?failover.reconnectDelay=30";
```

```

Hashtable<String, String> hashtable = new Hashtable<>();
hashtable.put("connectionfactory.SBCF",connectionUrl);
hashtable.put("queue.QUEUE", "default");
hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");

Context context = new InitialContext(hashtable);
ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
Destination queue = (Destination)context.lookup("QUEUE");
// 创建和 IoT 平台的 AMQP 连接
Connection connection = cf.createConnection(userName, password);
((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
// 创建 Session
// Session.CLIENT_ACKNOWLEDGE: 收到消息后, 需要手动调用 message.acknowledge()
// Session.AUTO_ACKNOWLEDGE: SDK 自动 ACK ( 推荐 )
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
connection.start();
// 创建消费者
MessageConsumer consumer = session.createConsumer(queue);
consumer.setMessageListener(messageListener);
}

private static MessageListener messageListener = new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            byte[] body = message.getBody(byte[].class);
            String content = new String(body);
            String topic = message.getStringProperty("topic");
            String messageId = message.getStringProperty("messageId");
            logger.info("receive message"
                + ", topic = " + topic
                + ", messageId = " + messageId
                + ", content = " + content);
            System.out.println();
            //如果创建 Session 选择的是 Session.CLIENT_ACKNOWLEDGE, 这里需要手动 ACK。
            //message.acknowledge();
            //如果要对收到的消息做耗时的处理, 请异步处理, 确保这里不要有耗时逻辑。
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```
};

private static MessageListener messageListener = new MessageListener() {
//回调
};

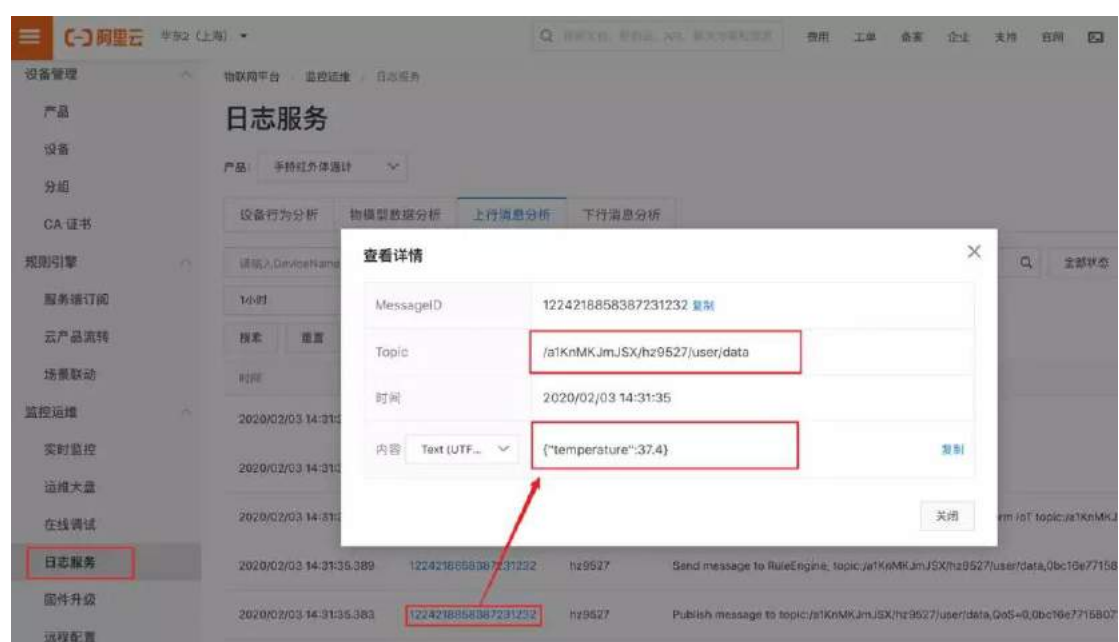
/**
 * password 签名计算方法，请参见上一篇文档：AMQP 客户端接入说明。
 */

private static String doSign(String toSignString, String secret, String signMethod) throws Exception {
    SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(), signMethod);
    Mac mac = Mac.getInstance(signMethod);
    mac.init(signingKey);
    byte[] rawHmac = mac.doFinal(toSignString.getBytes());
    return Base64.encodeBase64String(rawHmac);
}
}
```

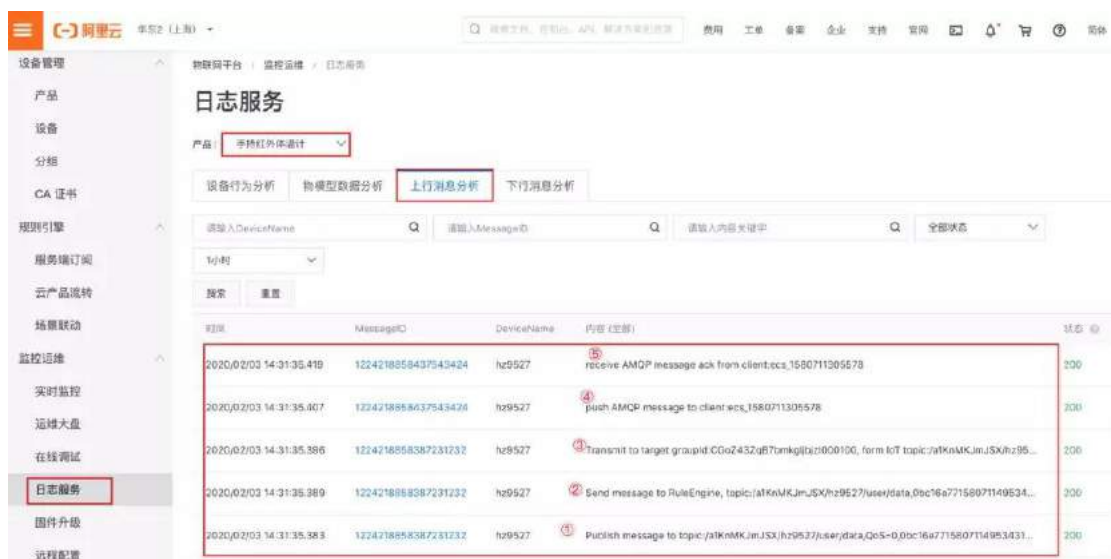
六、设备运行日志

我们启动设备模拟脚本，就可以在服务端控制台实时看到设备上报的业务数据。在控制台也可以查看完整数据记录。

1. 数据上报日志



2. 数据流转日志



日志服务

产品: 手持红外非接触测温计

设备行为分析 物模型数据分析 上行消息分析 下行消息分析

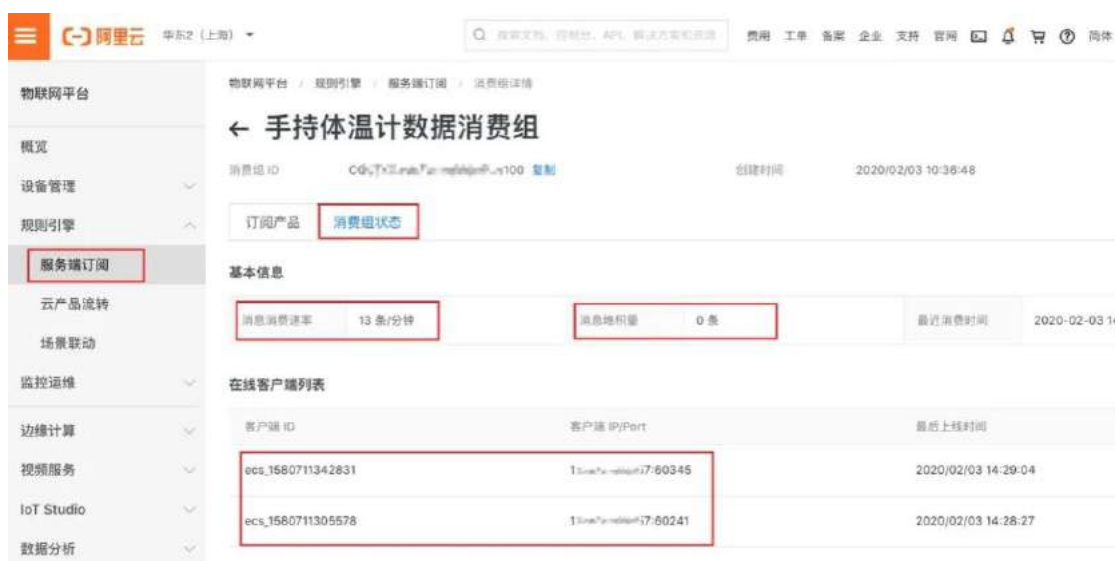
请输入DeviceName 请输入MessageID 请输入内容关键字 全部状态

1小时

搜索 重置

时间	MessageID	DeviceName	内容 (主链)	状态
2020/02/03 14:31:35.419	1224218858437545424	hz9527	receive AMQP message ack from client:ecs_1580711305578	200
2020/02/03 14:31:35.407	1224218858437545426	hz9527	push AMQP message to client:ecs_1580711305578	200
2020/02/03 14:31:35.386	1224218858437545427	hz9527	Transmit to target groupId: CQo2432qB7omkg@tcp:600100, from IoT topic:/atKMKJmJSX/hz9527/...	200
2020/02/03 14:31:35.389	1224218858437545428	hz9527	Send message to RuleEngine, topic:/atKMKJmJSX/hz9527/user/data,0bc16a77158071140534...	200
2020/02/03 14:31:35.383	1224218858437545429	hz9527	Publish message to topic:/atKMKJmJSX/hz9527/user/data,0bc16a7715807114053433...	200

3. 服务端订阅消费组情况



物联网平台 / 规则引擎 / 服务端订阅 / 消费组详情

← 手持体温计数据消费组

消费组 ID: CQo2432qB7omkg@tcp:600100 复制 创建时间: 2020/02/03 10:38:48

订阅产品 消费组状态

基本信息

消息消费速率: 13 条/分钟 消息堆积量: 0 条 最近消费时间: 2020-02-03 14:28:27

在线客户端列表

客户端 ID	客户端 IP/Port	最后上线时间
ecs_1580711342831	158.158.158.158:17-60345	2020/02/03 14:29:04
ecs_1580711305578	158.158.158.158:17-60241	2020/02/03 14:28:27



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