目录

[**课程内容** 2](#_Toc524339471)

[**课程目标** 2](#_Toc524339472)

[**课程准备** 3](#_Toc524339473)

[**课程知识点** 3](#_Toc524339474)

[**规则引擎介绍** 3](#_Toc524339475)

[**数据处理SQL** 4](#_Toc524339476)

[**数据转发操作** 5](#_Toc524339477)

[**基础版示例：温湿度计上报数据到钉钉群机器人** 9](#_Toc524339478)

[**钉钉群机器人配置** 9](#_Toc524339479)

[**函数计算脚本编写** 11](#_Toc524339480)

[**物联网平台侧开发** 15](#_Toc524339481)

[**设备端开发** 20](#_Toc524339482)

[**新建项目** 20](#_Toc524339483)

[**主要代码讲解** 21](#_Toc524339484)

[**修改设备相关参数** 23](#_Toc524339485)

[**编译下载** 24](#_Toc524339486)

[**运行调试** 24](#_Toc524339487)

[**高级版示例：温湿度计上报数据到用户服务器** 26](#_Toc524339488)

[**消息队列配置** 26](#_Toc524339489)

[**物联网平台侧开发** 28](#_Toc524339490)

[**MQ Demo配置** 31](#_Toc524339491)

[**设备端开发** 35](#_Toc524339492)

[**主要代码讲解** 36](#_Toc524339493)

[**修改设备相关参数** 38](#_Toc524339494)

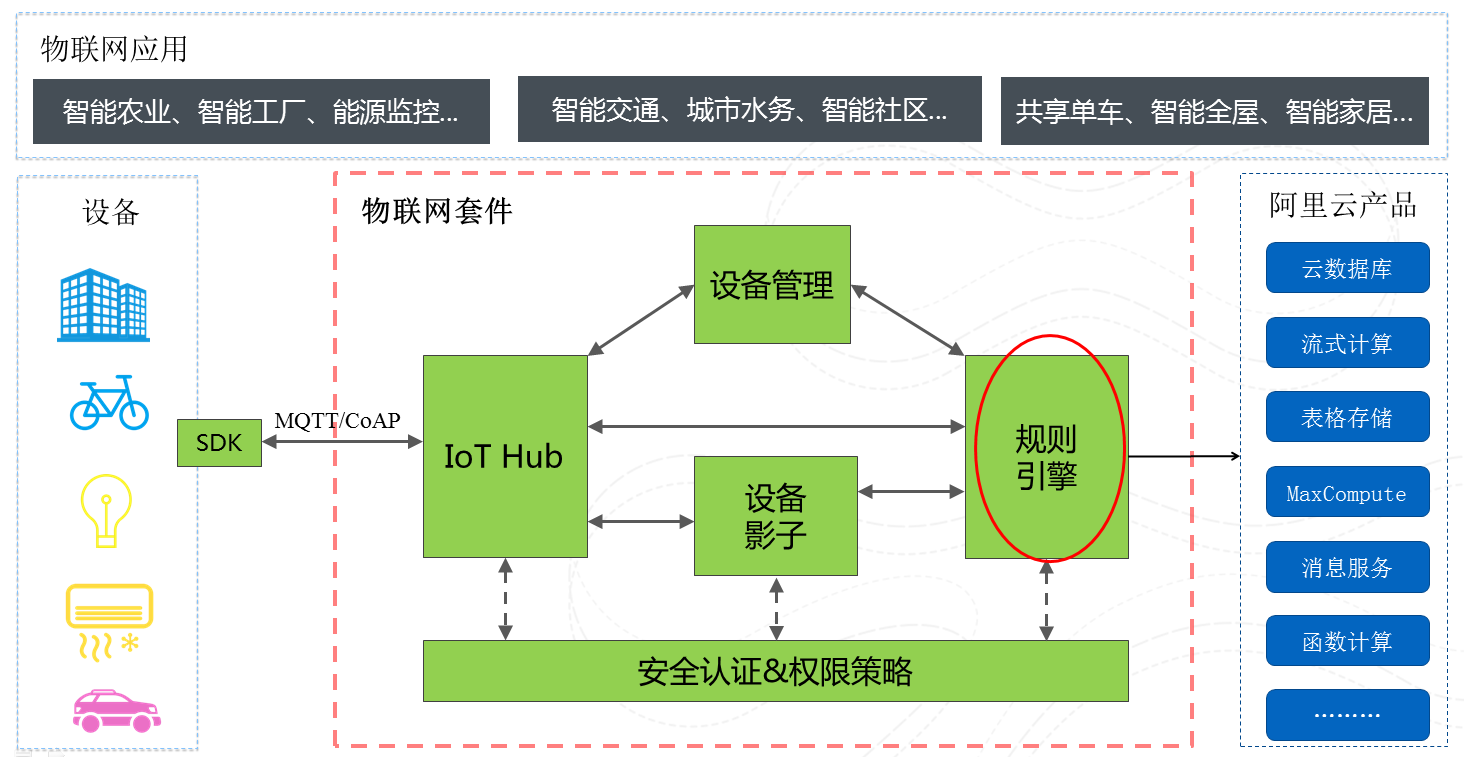
[**编译下载** 38](#_Toc524339495)

[**运行结果** 38](#_Toc524339496)

[**课程小结** 39](#_Toc524339497)

**规则引擎**

本次课程将详细介绍物联网平台的*规则引擎*组件，围绕数据处理与转发方法讲解SQL表达式的编写、转发操作的配置以及物联网平台上的数据流转格式等内容，帮助读者掌握*规则引擎*的整个使用流程，体验从*设备*上报数据到物联网平台再到云端计算或存储*产品*的数据采集+计算+存储的整体流程。



**课程内容**

* 介绍物联网平台的*规则引擎*组件
* 介绍*规则引擎*的数据处理配置
* 介绍*规则引擎*的数据转发配置
* 介绍阿里云函数计算与消息队列*产品*
* 介绍基础版和高级版物联网平台下的数据流转

**课程目标**

* 掌握*规则引擎*的使用方法
* 掌握SQL语句的编写方法和转发操作的配置流程
* 熟悉函数计算与消息队列*产品*
* 通过两个实例了解基础版和高级版下数据通过*规则引擎*的流转过程

**课程准备**

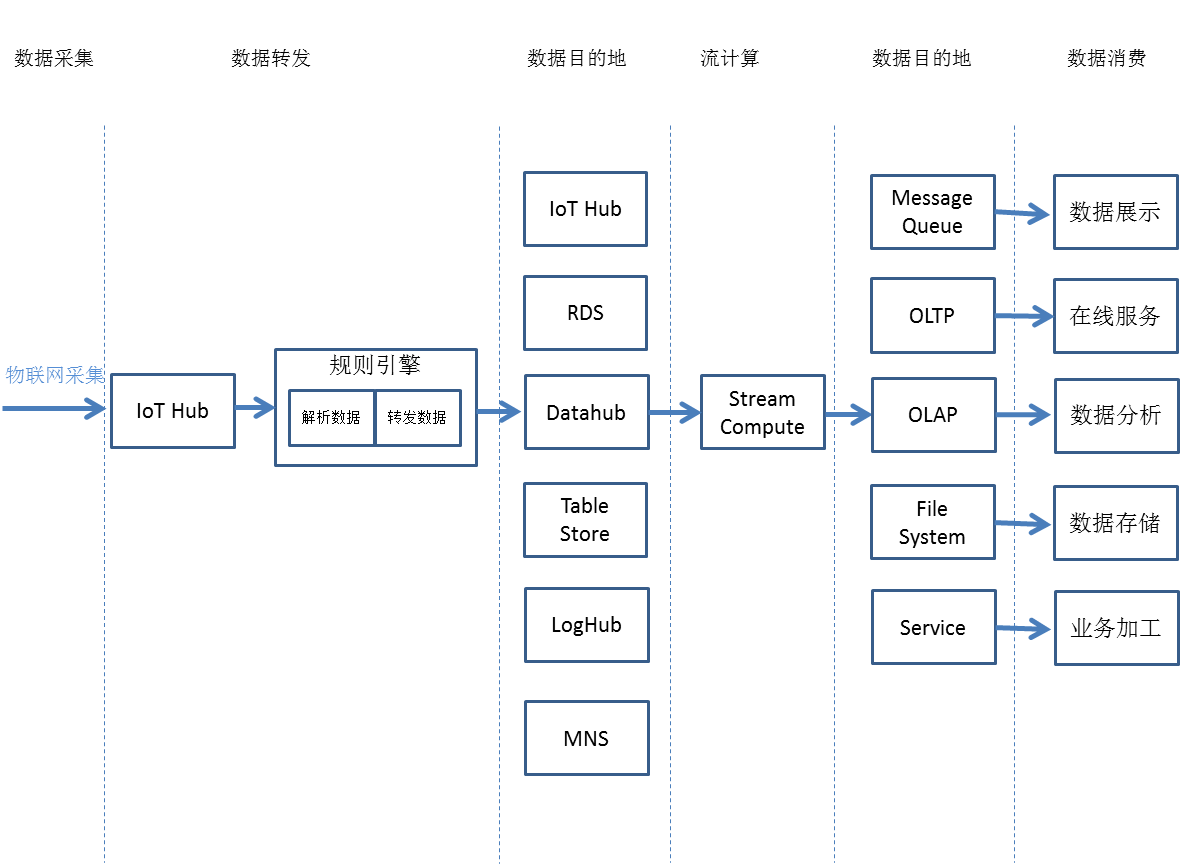
* 运行Windows系统的 PC机一台
* AIoTKIT开发板一块
* Mk3080 wifi模组
* micro USB连接线
* 安装有AliOS Things Studio插件的VSCode
* AliOS Things 1.3.3版本或更高
* ST-Link驱动程序
* 开通阿里云物联网平台*产品*
* 开通阿里云函数计算*产品*
* 开通阿里云消息队列*产品*
* 配置好Java JDK的IntelliJ IDEA

**课程知识点**

**规则引擎介绍**

当*设备*基于*Topic*与平台进行通信时，用户可以使用物联网平台的*规则引擎*组件对*设备*数据进行处理和转发。通过配置规则，用户可以将经过自定义SQL语句处理的数据转发到阿里云的各类*产品*中，十分方便地实现数据采集+计算+存储的全栈*服务*。

* 转发到另一个*Topic*中实现M2M通信
* 转发到RDS、Table Store、HiTSDB中进行存储
* 转发到DataHub中，使用Streamcompute进行流计算或使用Maxcompute进行大规模离线计算
* 转发到消息队列MQ中实现*服务*端高可靠消费数据
* 转发到函数计算中进行*事件*计算



**数据处理SQL**

*规则引擎*中的数据处理部分其实就是一条SQL表达（类似MySQL语法），可以对流转到*规则引擎*中的JSON数据进行解析和处理。

|  |
| --- |
| **SELECT** fields **FROM** “*Topic*” **WHERE** conditions |

* SELECT

select后可以指定消息内容字段，并将解析处理后的结果也构造为json格式。这些字段可以来源于流转到*规则引擎*中的消息的payload，用json表达式形式引用，也可以来源于函数例如*deviceName*()，*规则引擎*的支持函数详细列表可参见https://help.aliyun.com/document\_detail/30555.html。

* FROM

from后填写*Topic*，只有符合*Topic*规则的*设备*消息到达时，规则才会被触发，消息的payload数据才会以json格式被解析并根据SQL语句进行处理；否则该*设备*消息会被忽略。

* WHERE

where后填写的是规则的触发条件，即条件表达式，其中可以使用的字段与select语句一致。当接收到符合*Topic*规则的消息时，where语句的结果会决定规则是否被触发。where语句支持”>=”、”AND”、”%”等条件表达式，详细支持列表可参见https://help.aliyun.com/document\_detail/30554.html。

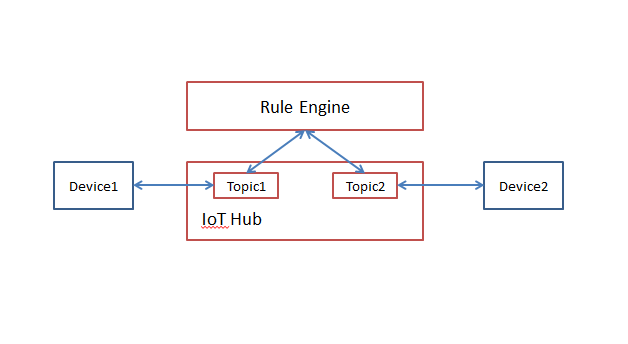
SQL语句执行完毕后，得到的数据结果会被用于下一步的转发操作。在转发数据的表达式中，可以通过使用“${表达式}”引用fields中对应的值。例如SQL语句为：select *deviceName*() as device, temperature t1 from xxx，那么在配置转发操作时可以使用${device}、${t1}分别引用select中的*属性*字段device和t1。

注意：*规则引擎*只能解析和处理Json格式的数据，而无法对二进制格式的数据进行解析和处理。当数据为二进制格式时，编写SQL不支持提取字段，只能将全部二进制数据直接透传，即select后填写“\*”内容；编写条件筛选语句也不支持提取使用消息中的字段。

**数据转发操作**

*规则引擎*中可以配置的转发操作包括八种。用户在配置转发操作前，需要先确认目的云*产品*是否在当前规则所在的地域和可用区内上线，并且支持相应格式数据的转发，各地域和可用区的详细列表可参见：https://help.aliyun.com/document\_detail/85669.html。

* 发布到另一个*Topic*

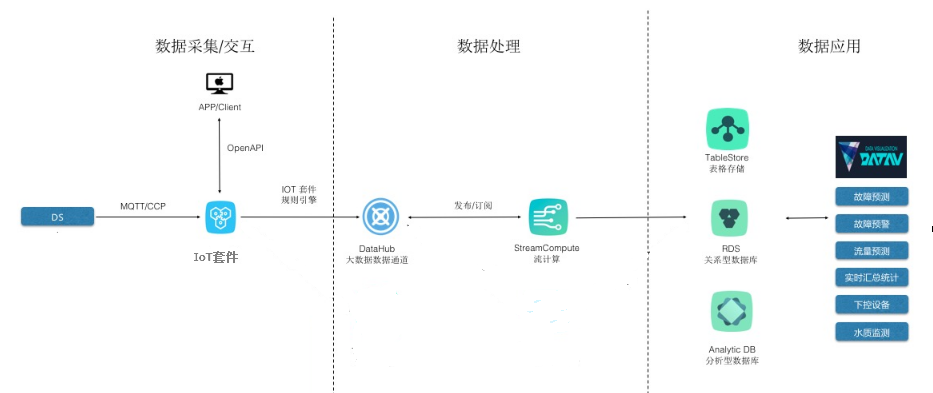


*规则引擎*可以将一个*Topic*中的消息进行处理后转发到另一个*Topic*中，帮助实现M2M*设备*间通信架构。

* 存储到表格存储（Table Store）

*规则引擎*可以将*设备*上报到物联网平台的数据转发到阿里云提供的NoSQL（非关系型）数据存储*服务*，即表格存储*产品*中，实现海量结构化数据的存储和实时访问。关于阿里云表格存储的更多信息参见链接：https://help.aliyun.com/product/27278.html。

* 发送数据到DataHub中



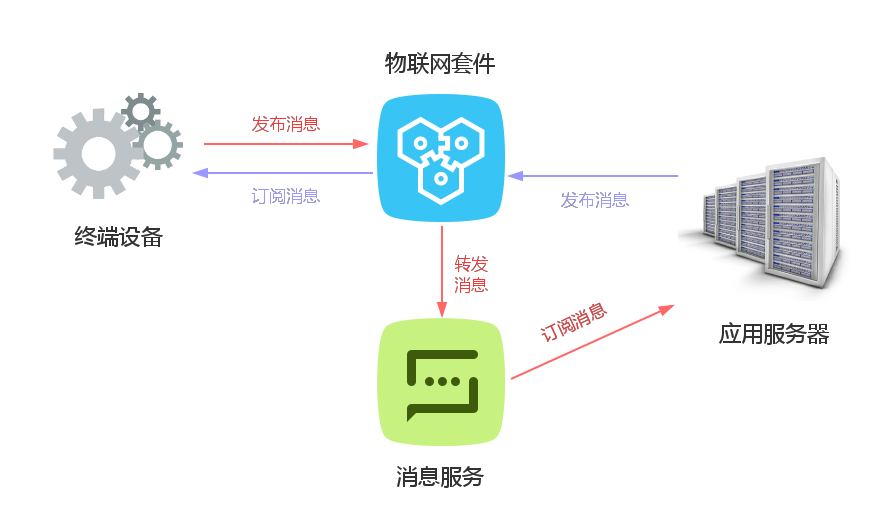
目前在很多物联网场景中，流计算都是刚需。流计算是一种*事件*触发的模型，即一旦有新的*事件*（数据）到达，流计算系统将完成一次计算，并继续转为等待下一次*事件*到来。源源不断的数据流将为下游的流计算提供触发。

*规则引擎*可以将驱动流计算运行的*设备*数据转发到DataHub*产品*中，为下游使用了该DataHub的流计算以及Maxcompute任务提供*事件*触发机制，触发流计算的运行。关于阿里云DataHub的更多信息参见链接：https://help.aliyun.com/product/53345.html。

* 存储到云数据库（RDS）中

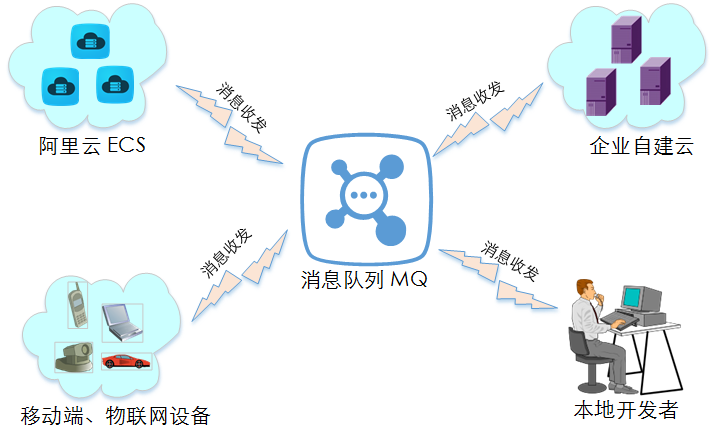
*规则引擎*可以将*设备*上报到平台的数据转发到阿里云提供的SQL（关系型）数据库，即MySQL类型的云数据库（RDS）中。与NoSQL数据库相比，SQL数据库是精确的，适合那些具有精确标准的、定义明确的项目，如存储用户的账号、地址信息；而NoSQL数据库是多变的，适合那些具有不确定需求，看中速度和可扩展性的项目，如存储社交网络的评论。关于阿里云云数据库的更多信息参见链接：https://help.aliyun.com/product/26090.html。

* 发送消息到消息*服务*（Message Service）中



*规则引擎*可以将*设备*数据转发到消息MNS中，结合*服务*端接入MNS来实现*设备*端与*服务*端之间的高性能消息闭环传输。关于阿里云消息*服务*的更多信息参见链接：https://help.aliyun.com/product/27412.html。

* 发送消息到消息队列（Message Queue）中



*规则引擎*可以将*设备*数据转发到消息队列MQ中，从而具备从*设备*到物联网平台到消息队列MQ再到应用*服务*器的全链路、高可靠的消息能力。用户可以将应用部署在阿里云ECS、企业自建云，或者嵌入到移动端和物联网*设备*中与MQ建立连接实现消息收发，同时本地开发者也可以通过公网接入MQ*服务*进行消息收发。关于阿里云消息队列的更多信息可参见链接：https://help.aliyun.com/product/29530.html。

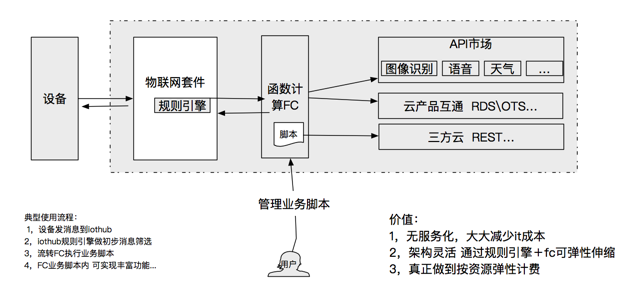
消息队列MQ与前一种操作中涉及到的消息*服务*MNS*产品*都是消息中间件，相比之下，MQ更适合并发高、延时要求低的消息传输场景；而消息*服务*则具备更加强大的扩展辅助功能如短信、邮件。关于消息*服务*MNS与消息队列MQ的更详细对比，可以参见链接：https://help.aliyun.com/document\_detail/27437.html。

* 存储到时间序列数据库（TSDB）中

根据统计，在大数据领域中超过一半的都是时序数据。时序数据是在时间上分布的一系列数值，一般是指标型数据，比如股票价格、气温变化、个人健康数据、工业传感器数据、*服务*器系统监控数据等。时间序列数据的处理与一般的数据库处理有所不同，一般数据基于行，每一个数据点是一行，而时间序列数据则是基于时间线处理数据。

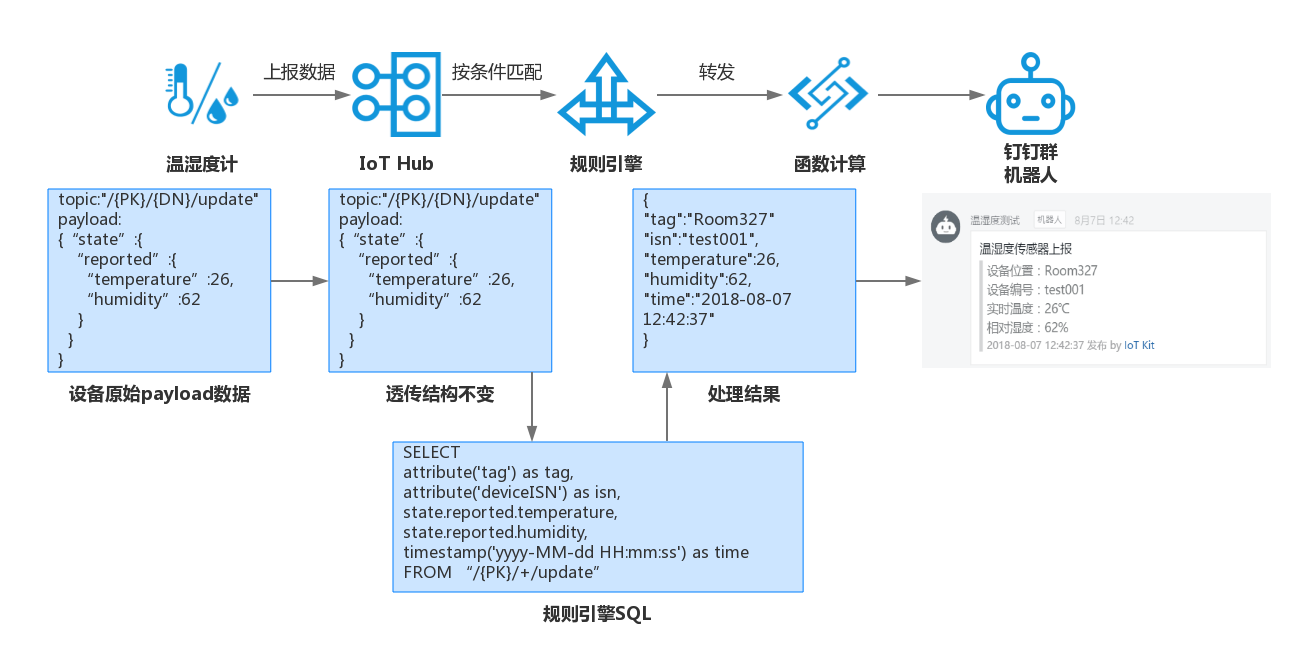
*规则引擎*可以将*设备*数据转发到阿里云提供的在线时间序列数据库（TSDB）*产品*中进行存储。关于阿里云TSDB的更多信息参见链接：https://help.aliyun.com/product/54825.html。

* 发送数据到函数计算（FC）中



*规则引擎*可以将*设备*数据转发到阿里云提供的*事件*驱动全托管计算*服务*，即函数计算（FC）*产品*中。通过函数计算，物联网平台以*事件*驱动的方式连接不同的*服务*，当*事件*源*服务*触发*事件*时，系统自动调用关联的函数处理*事件*。并且，用户无需管理*服务*器等基础设施，只需要编写代码并上传，函数计算会准备好计算资源，以弹性、可靠的方式运行用户代码，并提供日志查询、性能监控和报警等功能。关于阿里云函数计算的更多信息参见链接：<https://help.aliyun.com/product/50980.html>。

**基础版示例：温湿度计上报数据到钉钉群机器人**



本实验中我们将AIoTKIT*设备*通过MQTT协议连接到物联网平台，并通过*规则引擎*模块，将上报的温湿度数据转发到函数计算模块中由Nodejs脚本进行数据处理，最后将消息推送到配置了温湿度机器人的钉钉群组。

**钉钉群机器人配置**

群机器人是钉钉群具备的一项高级扩展功能，通过使用该功能我们可以将第三方*服务*的信息聚合到群聊中，实现自动化的信息同步，例如通过聚合GitHub源码管理*服务*，实现源码更新的同步。下面我们为钉钉群配置温湿度机器人。

登录钉钉电脑版后，点击右上角账户右侧的小下拉按钮，然后点击“机器人管理”。



选择“自定义”Webhook机器人，按照指引输入“机器人名字”和“添加到群组”后，记录下生成的webhook地址，其中包含了访问用的accessToken（webhook链接末尾包含access\_token=xxxxxx），该信息非常重要。





**函数计算脚本编写**

* 创建*服务*与函数

开通阿里云“函数计算”*服务*后，进入“函数计算”控制台。选择“华东2”区域，创建*服务*，新建一个名为IoT\_Service的*服务*。*服务*是资源管理的基本单位，用户可以在*服务*上执行*授权*、创建函数等操作。



*服务*创建成功后，进入该*服务*概览，新建空白Nodejs函数。函数是由用户编写的，由*事件*触发，执行特定功能的一段代码，是调度和运行的基本单位。

创建函数过程中，我们选择“nodejs6”空白函数模板，不创建触发器，函数名称命名为pushData2DingTalk，运行环境为nodejs6，函数入口为index.handler，即函数计算会去加载index.js文件中定义的handler函数,函数内存和超时时间使用默认值，无需配置权限，全部设置完成后，点击“创建”即可。











* 编写函数脚本

本实验希望将温湿度计获取到的温湿度信息以及该*设备*自身的*属性*信息通过机器人发布到钉钉群中，因此在该函数脚本中，我们需要将从上游物联网平台处获取到的*设备*位置信息、*设备*编号信息、实时温度信息、相对湿度信息和上报时间信息依据钉钉消息格式进行组装，然后通过https模块将数据发布到钉钉群机器人的webhook接口。

有关钉钉消息格式规范的文档可参见官网https://open-doc.dingtalk.com/docs/doc.htm?treeId=172&articleId=104972&docType=1。本实验中函数pushData2DingTalk的完整脚本为：

|  |
| --- |
| const https = require('https');  //创建*服务*器https  const accessToken = '此处填写钉钉机器人webhook的accessToken';  //handler函数与创建时的“index.handler”字段相对应；  //event参数是用户调用函数时传入的数据，类型是Buffer;  //context参数中包含一些函数的运行时信息（例如request id等），其类型是object；  //callback参数用于返回调用函数的结果  module.exports.handler = function(event, context, callback) {  var eventJson = JSON.parse(event.toString());  //钉钉消息格式  const postData = JSON.stringify({  "msgtype": "markdown",  "markdown": {  "title": "温湿度传感器",  "text": "#### 温湿度传感器上报\n" +  "> *设备*位置：" + eventJson.tag + "\n\n" +  "> *设备*编号：" + eventJson.isn+ "\n\n" +  "> 实时温度：" + eventJson.temperature + "℃\n\n" +  "> 相对湿度：" + eventJson.humidity + "%\n\n" +  "> ###### " + eventJson.time + " 发布 by [IoT Kit](https://www.aliyun.com/product/iot) \n"  },  "at": {  "isAtAll": false  }  });  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');  });  });  // 异常返回  req.on('error', (e) => {  callback(e);  });  // 写入数据  req.write(postData);  req.end();  }; |

**物联网平台侧开发**

* 创建*产品*，添加*设备*

首先，登录物联网平台控制台https://iot.console.aliyun.com，点击“*产品*管理”栏，在“华东2（上海）”区域下新建一个基础版*产品*“温湿度测试”，节点类型选择为“*设备*”，并在该*产品*下添加test*设备*。



然后为该*设备*添加“*设备标签*”，在“*标签*信息”下点击“立即添加”，向该*设备*添加两个*设备属性*：

|  |  |  |
| --- | --- | --- |
| *属性* | *属性*值 | 描述 |
| tag | … | *设备*所在位置 |
| deviceISN | … | *设备*序列号 |

* 设置*规则引擎*

温湿度计*设备*接入物联网平台后，会通过update *Topic*向平台上报消息，我们需要从中提取出温度值信息temperature和湿度值信息humidity。同时，观察函数计算的脚本可见，钉钉群机器人上报的消息内容除了实时温度和相对湿度，还包括*设备*位置、*设备*编号和时间信息；因此我们还需要从*设备*的自定义*属性*中获取*标签*tag和*设备*序列号deviceISN信息，利用*规则引擎*支持的函数获取时间信息，将这些信息和温湿度信息一并转发到函数计算中去。

新建一条规则，在物联网平台控制台界面上，点击左侧“*规则引擎*”栏，选择区域为“华东2（上海）”后，点击右上角“创建规则”，输入规则名称并选择数据格式后即可成功创建规则。





注意：在选择数据格式时，由于*规则引擎*是基于*Topic*进行数据处理的，因此数据格式需要与被处理的*Topic*中的数据格式保持一致，本实验中选择JSON格式。

规则创建成功后，即可编写SQL语句对某一*Topic*中的数据进行处理。



“规则查询语句”会由系统根据“字段”、“*Topic*”和“条件”自动补充完整，不需用户填写。“*Topic*”为待处理转发的消息的来源，可以选择“自定义”或“sys”。选择“自定义”表明该*Topic*是用户自定义的*产品Topic*，即以/${*productKey*}/${*deviceName*}开头，用户需要在选择*产品*后将该*Topic*补充完整。选择“sys”表明该*Topic*是高级版*产品*内系统定义的*产品Topic*，不以/${*productKey*}/${*deviceName*}开头；此时用户需要选择*产品*，选择*设备*，还需要选择系统定义的某个具体*Topic*。

本实验中完整的SQL语句应为：SELECT attribute('tag') as tag, attribute('deviceISN') as isn, state.reported.temperature, state.reported.humidity, timestamp('yyyy-MM-dd HH:mm:ss') as time FROM “/a1Q9PP17SGy/+/update”。

“/a1Q9PP17SGy/+/update”中的“+”为通配符，代表本级所有类目，即可以代表“/a1Q9PP17SGy/device1/update”,也可以代表“/a1Q9PP17SGy/device2/update”。

此处提取温湿度字段的语句” state.reported.temperature”取决于*设备*上报的消息格式，本实验中*设备*上报的数据格式为：

|  |
| --- |
| {  “state”:{  “reported”:{  “temperature”:26,  “humidity”:62  }  }  } |

上述数据内容通过IoT Hub上报到平台后透传至*规则引擎*模块，结构和内容均不变，因此*规则引擎*中的SQL语句为“state.reported.temperature, state.reported.humidity”。

基于上述SQL语句完成数据处理之后，为了触发后续的函数计算和钉钉群机器人，我们需要将这些数据转发到“函数计算”当中去，选择上一步创建的IoT\_Service*服务*和pushData2Dingtalk函数。如果“角色”下拉栏中没有可选项，点击右侧“创建RAM角色”进行*授权*即可。规则设置完毕后，点击“启动”规则。



**设备端开发**

本次课程使用AIoTKIT开发板通过mk3080 Wi-Fi模组接入网络，与物联网云平台之间采用MQTT协议通信。在本次的*规则引擎*实验中，使用板载的温湿度传感器来采集环境温度。

**新建项目**

在LP应用开发教程2中，我们已经下载了所需要的例程代码。打开VSCode，点击“文件->打开文件夹”，打开已经下载下来的代码，并将当前分支切换到developer分支。点击工程界面左下角，选择此次的例程为humi\_temp\_app例程，开发板选择为AIoTKIT板。

**主要代码讲解**

在本次应用开发教程中，*设备*端要完成的功能较为简单。只需要定时读取外部环境传感器的温湿度信息，并按照指定格式将温湿度信息拼合起来，并上传到云端即可。

应用于本次实验的APP和上手LP应用开发教程2——基于IoT Hub的真实*设备*接入相类似，只是将上报的虚假环境信息转换为真实的传感器读取到的环境信息即可。所以在本次代码讲解中，相关重复部分代码我们将仅仅列出其函数声明和功能部分，函数主体内容将不再涉及，详情可以参考LP应用教程2基于IoT Hub 的实验中。

1、int application\_start(int argc, char \*argv[])

开发者真正的应用入口函数。在本函数中完成的主要功能为

* AT指令初始化，SAL框架初始化；
* 设置输出的LOG 等级 aos\_set\_log\_level(AOS\_LL\_DEBUG)；
* AliOS Things定义了一系列系统*事件*，程序可以通过aos\_register\_event\_filter()注册*事件*监听函数，进行相应的处理，比如 Wifi*事件*；
* 在配网过程中，netmgr负责定义和注册Wifi回调函数netmgr\_init()。
* 通过调用aos\_loop\_run()进入*事件*循环。

|  |
| --- |
| int application\_start(int argc, char \*argv[])  {  netmgr\_ap\_config\_t apconfig;  #if AOS\_ATCMD //AT指令初始化  at.set\_mode(ASYN);  at.init(AT\_RECV\_PREFIX, AT\_RECV\_SUCCESS\_POSTFIX,  AT\_RECV\_FAIL\_POSTFIX, AT\_SEND\_DELIMITER, 1000);  #endif  #ifdef WITH\_SAL //SAL网络框架初始化  sal\_init();  #endif  printf("== Build on: %s %s ===\n", \_\_DATE\_\_, \_\_TIME\_\_);  aos\_set\_log\_level(AOS\_LL\_DEBUG);  aos\_register\_event\_filter(EV\_WIFI, wifi\_service\_event, NULL);  netmgr\_init(); //定义与注册wifi回调函数  netmgr\_start(false);  aos\_cli\_register\_command(&mqttcmd);  aos\_loop\_run();  return 0;  } |

图 15 应用入口函数

2、static void wifi\_service\_event(input\_event\_t \*event, void \*priv\_data)

Wifi*事件*处理函数，当有Wifi*事件*发生时运行该函数。在该函数中完成的主要功能为进行Wifi*事件*的判断，包括*事件*类型的确认等，在确认无误后调用mqtt\_client\_example（）。

3、int mqtt\_client\_example(void)

mqtt\_client\_example()函数是本次mqtt例程中的鉴权连接函数，该函数所实现的主要功能是：

* 获取*设备*进行鉴权注册时的相关参数。
* 通过Wifi连接IoT平台，进行*设备*注册。

4、static void mqtt\_service\_event(input\_event\_t \*event, void \*priv\_data)

该函数为MQTT例程中*事件*触发后调用的函数，其主要功能为进行*事件*合法性检查，以及调用mqtt\_work()主函数。

5、static void mqtt\_work (void \*pclient)

void mqtt\_work(void \*parms)函数是本次mqtt例程中的mqtt上云发送数据函数，该函数所实现的主要功能是：

* 订阅相关*TOPIC*
* 每隔3S读取环境传感器获得的温湿度数据，并将数据按照指定的json格式进行封装
* 循环发送封装好的温湿度数据到指定的*TOPIC*。

|  |
| --- |
| static void mqtt\_work (void \*parms) {  float Temperature=0.0;float humidity=0.0;  int rc = -1;  /\*订阅get *TOPIC*\*/  if (is\_subscribed == 0) {  /\* Subscribe the specific *Topic* \*/  rc = mqtt\_subscribe(*TOPIC*\_GET, mqtt\_sub\_callback, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  }  is\_subscribed = 1;  aos\_schedule\_call(ota\_init, NULL);  }  #ifndef MQTT\_PRESS\_TEST  else {  /\* Generate topic message \*/  get\_humi\_data(&humi\_data, &humi\_timestamp);  get\_temp\_data(&temp\_data, &temp\_timestamp);  Temperature=(float)temp\_data/10.0;humidity=(float)humi\_data;  int msg\_len = sprintf(msg\_pub, DATA\_POST\_FORMAT, Temperature,humidity);  if (msg\_len < 0) {  LOG("Error occur! Exit program");  }  rc = mqtt\_publish(TOPIC\_UPDATE, IOTX\_MQTT\_QOS1, msg\_pub, msg\_len);  if (rc < 0) {  LOG("error occur when publish");  }  LOG("packet-id=%u, publish topic msg=%s", (uint32\_t)rc, msg\_pub);  }  cnt++;  if (cnt < 200) {  aos\_post\_delayed\_action(3000, mqtt\_work, NULL);  } else {  aos\_cancel\_delayed\_action(3000, mqtt\_work, NULL);  mqtt\_unsubscribe(TOPIC\_GET);  aos\_msleep(200);  mqtt\_deinit\_instance();  is\_subscribed = 0;  cnt = 0;  }  #endif  } |

图 19 MQTT例程数据上云主函数

**修改设备相关参数**

在上一小节中，我们打开了humi\_temp\_app@AIoTKIT工程。在本小节中，我们需要修改官方humi\_temp\_app项目工程，添加我们在阿里云物联网控制台上新建*设备*的三元组信息。具体流程如下：

demo程序所在路径是/example/humi\_temp\_app,在此次例程当中，我们需要将*设备*的三元组信息修改为新注册的*设备*三元组信息。具体修改文件为：example/humi\_temp\_app/humi\_temp\_app.c。PRODUCT\_KEY、DEVICE\_NAME、 DEVICE\_SECRET这三个参数是保证*设备*和IOT平台间可靠通信的唯一标识，所以这三个参数必须保证和建立*设备*时的信息相同，*TOPIC*信息也要保证和平台端的*TOPIC*保持一致，因为*设备*在发送与接收消息时都要带有*TOPIC*信息，不一致的话可能会导致数据通信发生错误。具体参数修改如下图所示：

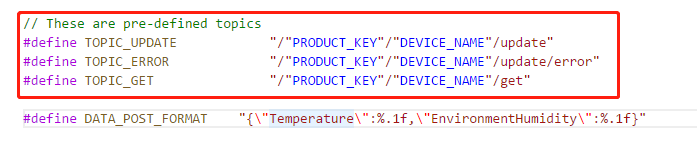


图 42 修改*设备*三元组信息

**编译下载**

工程编译与下载的方式和LP应用开发教程2中相同，详情可参考开发教程2中的*设备*开发编译下载章节，在此处不再赘述。

至此，工程代码烧录进入开发板当中。打开串口助手即可查看程序的运行输出信息，

**运行调试**

等到开发板正常启动之后通过命令行使得Wifi模组能正确连接到对应的AP，即通过串口助手输入netmgr connect WIFINAME WIFIPASSWORD（WIFI名称以及WIFI密码）联网指令如下图所示：

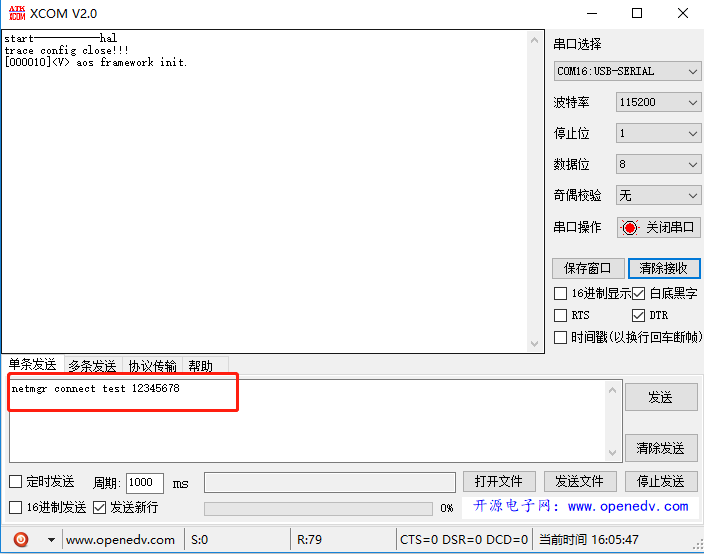


图 50 输入*设备*联网指令

正常联网后，mqttapp会真正开始运行。下图为mqtt运行日志截图：

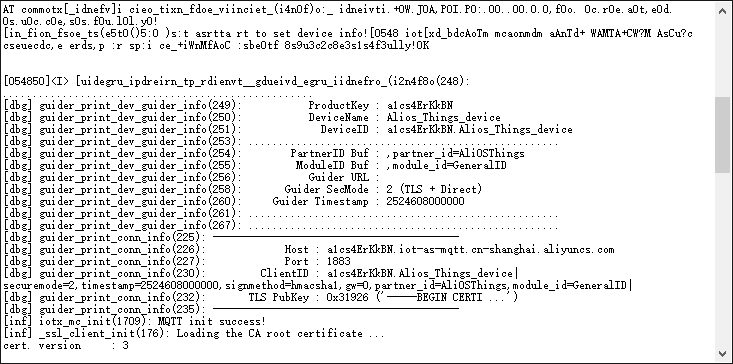
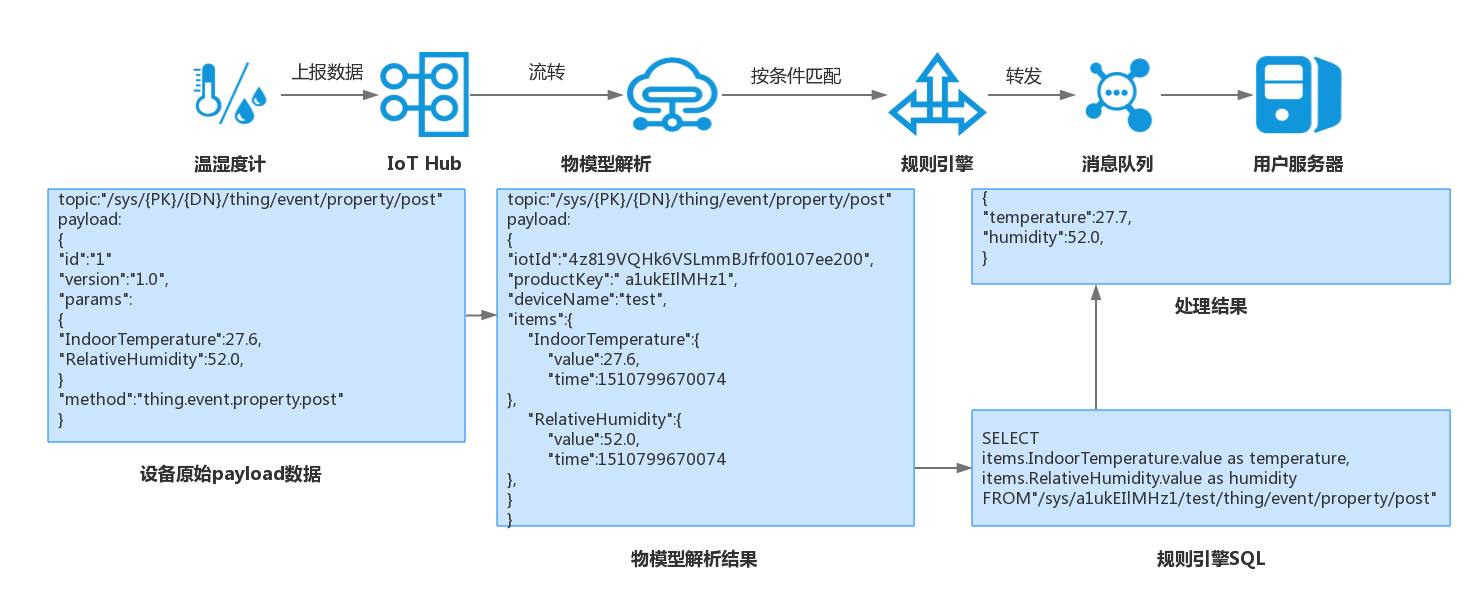


图 51 mqtt例程运行日志

联网成功以后，程序会在每一个循环中上报数据，此时我们便可以在之前配置好的钉钉群组中看到由“温湿度测试”机器人推送的温湿度消息。



**高级版示例：温湿度计上报数据到用户服务器**



本实验中接入物联网平台的仍然是AIoTKIT*设备*，我们通过设置*规则引擎*模块，将上报的温湿度数据借助阿里云的“消息队列”*产品*流转到用户自己的*服务*器上去，实现*设备*→物联网平台→MQ消息队列→用户*服务*器的架构，便于用户的后续应用业务开发。

**消息队列配置**

借助消息队列削峰填谷、缓冲消息的能力，在*设备*→物联网平台→MQ消息队列→用户*服务*器的架构下，用户*服务*器就可以不必担心海量*设备*消息压垮*服务*器的情况发生，用户*服务*器可以根据自身的能力来消费队列中的海量消息。在MQ消息系统中，消息发布者将消息发送到某个指定的消息主题（*Topic*）中，消息订阅者通过订阅该指定的*Topic*来获取和消费消息，因此我们首先需要创建好消息队列*Topic*。

登录消息队列控制台https://ons.console.aliyun.com/，点击“*Topic*管理”栏，在“公网”区域下创建消息队列*Topic*，，建议消息类型选择为普通消息（无序消息中的一种）。若未开通阿里云“消息队列”*产品*，则根据网页提示开通后再进行操作即可。



* 无序消息：不保证先入先出（FIFO）的严格顺序，包含普通消息、定时消息、事务消息。
* 全局顺序消息：所有消息严格按照FIFO的严格顺序进行发布和消费。
* 分区顺序消息：消息根据sharding key（片键）进行分区，提高整体并发度与使用性能。同一个分区的消息严格按照FIFO的严格顺序进行发布和消费。
* Kafka消息：兼容MQ及Kafka协议（kafka-client-0.10 及以上版本）的消息类型。

本实验中，我们的用户*服务*器需要通过消息队列消费*设备*上报到平台的消息，充当的是队列的“订阅者”角色。因此，我们需要申请*Topic*的订阅功能，即创建队列的“消费者”。





**物联网平台侧开发**

* 新建*产品*与*设备*，定义功能

首先，登录物联网平台控制台https://iot.console.aliyun.com，点击“*产品*管理”栏，在“华东2（上海）”区域下新建一个高级版*产品*“温湿度计”，节点类型选择为“*设备*”，*设备*类型为“无”，数据格式为“*Alink* JSON”,并在该*产品*下添加test*设备*。



本实验中，test*设备*需要向平台上报实时温湿度数据，因此我们需要将温湿度定义为该*产品*的*属性*。在创建好的“温湿度计”*产品*详情页面中，点击“功能定义”栏，点击右侧的“新增”按钮，从标准功能库中搜索“室内温度（IndoorTemperature）”和“湿度（RelativeHumidity）”这两个*属性*，系统已经为我们定义好了*属性*的标识符、数据类型与取值范围等，我们将分辨率均设为0.1即可。



* 设置*规则引擎*

温湿度计*设备*接入物联网平台后，会通过上报*属性*的系统*Topic*向平台上报温湿度数据，我们需要从中提取出温度值信息temperature和湿度值信息humidity。

在高级版物联网平台中，*设备*通过IoT Hub将数据上报到平台后，平台会根据该*设备*的物模型将数据内容组织为定义好的数据格式，用户必须根据该数据格式来编写SQL语句才能够正确地解析处理数据。

高级版*产品*的每个系统*Topic*默认的数据格式可参考链接：https://help.aliyun.com/document\_detail/73736.html。

本实验中温湿度计*设备*上报的原始payload数据为：

|  |
| --- |
| payload:  {  “id”:”1”,  “version”:”1.0”,  “params”:  {  “IndoorTemperature”:27.6,  “RelativeHumidity”:52.0,  }  “method”:”thing.event.property.post”  } |

经过物模型解析后的数据格式为：

|  |
| --- |
| payload:  {  “iotId”:4z819VQHk6VSLmmBJfrf00107ee200”,  “*productKey*”:” a1ukEIlMHz1”,  “*deviceName*”:”test”,  “items”:{  “IndoorTemperature”:{  “value”:27.6,  “time”:1510799670074  },  “RelativeHumidity”:{  “value”:52.0,  “time”:1510799670074  },  }  } |

因此在“*规则引擎*”中创建的新规则，其处理数据的SQL语句为：SELECT items.IndoorTemperature.value as temperature, items.RelativeHumidity.value as humidity FROM “/sys/a1ukEIlMHz1/test/thing/event/property/post”。如果不根据平台定义的数据格式采用”items.IndoorTemperature.value as temperature”，而直接根据*设备*上报的原始payload格式将SQL语句的字段写为”IndoorTemperature”，温湿度数据就无法成功被提取出来进行转发。



然后配置数据流转到之前我们在MQ控制台公网区域创建的消息队列*Topic*，注意一定要*授权*，否则平台无法向消息队列中写数据，配置完毕后启动规则即可。



**MQ Demo配置**

此时用IntelliJ IDEA软件打开示例代码，点击“open”后找到电脑中存放示例代码的目录，然后打开。



打开示例代码后，在目录src\main\java\com\aliyun\openservices\tcp\example下找到MqConfig.java文件进行配置。

|  |
| --- |
| Public static final String *TOPIC* = "刚才创建的 *Topic*";  Public static final String CONSUMER\_ID = "刚才创建的 CID";  Public static final String ACCESS\_KEY = "您的阿里云账号的 AK";  Public static final String SECRET\_KEY = "您的阿里云账号的 SK";  Public static final String TAG = "\*";  //表示TAG可为任意  Public static final String ONSADDR = "http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet";  //公网测试 |

* AK即Access Key ID，SK即Access Key Secret，登录阿里云账号，点击accessKeys即可获取AK，SK。



* 队列中的每条消息都有tag，\*表示订阅全部tag。
* ONSADDR 请根据不同Region进行配置：

公网测试:

http://onsaddr-internet.aliyun.com/rocketmq/nsaddr4client-internet

公有云生产:

http://onsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal

杭州金融云:

http://jbponsaddr-internal.aliyun.com:8080/rocketmq/nsaddr4client-internal

深圳金融云:

http://mq4finance-sz.addr.aliyun.com:8080/rocketmq/nsaddr4client-internal

参数配置完成后，运行src\main\java\com\aliyun\openservices\tcp\example\consumer路径下的SimpleMQConsumer.java文件来接收消息。



控制台打印出“Consumer start success.”说明消费者已经上线。



然后我们可以在MQ控制台上查看消费者状态，消费者已经在线，且订阅关系一致。



* 集群订阅：指同一个Consumer ID 所标识的所有 Consumer 平均分摊消费消息；例如某个*Topic*有9条消息，一个Consumer ID 有 3 个 Consumer 实例，那么在集群消费模式下每个实例平均分摊，只消费其中的 3 条消息。
* 广播订阅：指同一个 Consumer ID 所标识的所有 Consumer 都会各自消费某条消息一次；例如某个 *Topic* 有 9 条消息，一个 Consumer ID 有 3 个 Consumer 实例，那么在广播消费模式下每个实例都会各自消费 9 条消息。

**设备端开发**

与“上报数据到钉钉群机器人”实验使用基础版*设备*不同，本实验使用的是高级版的*产品*与*设备*。相比于基础版，高级版物联网平台依据定义好的*产品*物模型，对*设备*上报数据的格式进行了规定，同时*设备*向平台上报数据时使用的*Topic*也不相同，具体区别说明如下：

|  |  |  |
| --- | --- | --- |
| 不同点 | 基础版 | 高级版 |
| 数据格式 | {"state":{  "reported":{  "temperature":%.1f,  "humidity":%.1f }  }} | {"id":"%d",  "version":"1.0",  "params":{  “IndoorTemperature":%.1f, "RelativeHumidity":%.1f} "method":"thing.event.property.post"  } |
| *Topic* | ${*productKey*}/${*deviceName*}/update | /sys/${*productKey*}/${*deviceName*}/thing/event/property/post |

因此，我们重新打开一个新的app项目来进行本次高级版的实验。点击工程界面左下角，选择此次的例程为high\_humi\_tempapp例程，开发板选择为AIoTKIT开发板。

**主要代码讲解**

在本次上传数据到高级版的例程中，与基础版功能相类似，只需要定时读取外部环境传感器的温湿度信息，并按照高级版的json格式封装数据，并上传到特定*TOPIC*即可。

应用于本次实验的APP和基础版的例程类似，只是将上报的数据封装的格式和上报的*TOPIC*有所差别。所以在本次代码讲解中，相关重复部分代码我们将不再涉及，详情可以参考前面部分基础版实验介绍。

1、int application\_start(int argc, char \*argv[])

开发者真正的应用入口函数。在本函数中完成的主要功能为

* AT指令初始化，SAL框架初始化；
* 设置输出的LOG 等级 aos\_set\_log\_level(AOS\_LL\_DEBUG)；
* 打开外部传感器，准备读取数据
* AliOS Things定义了一系列系统*事件*，程序可以通过aos\_register\_event\_filter()注册*事件*监听函数，进行相应的处理，比如 Wifi*事件*；
* 在配网过程中，netmgr负责定义和注册Wifi回调函数netmgr\_init()。
* 通过调用aos\_loop\_run()进入*事件*循环。

|  |
| --- |
| int application\_start(int argc, char \*argv[])  {  netmgr\_ap\_config\_t apconfig;  #if AOS\_ATCMD //AT指令初始化  at.set\_mode(ASYN);  at.init(AT\_RECV\_PREFIX, AT\_RECV\_SUCCESS\_POSTFIX,  AT\_RECV\_FAIL\_POSTFIX, AT\_SEND\_DELIMITER, 1000);  #endif  #ifdef WITH\_SAL //SAL网络框架初始化  sal\_init();  #endif  printf("== Build on: %s %s ===\n", \_\_DATE\_\_, \_\_TIME\_\_);  aos\_set\_log\_level(AOS\_LL\_DEBUG);  sensor\_all\_open(); //打开外部环境传感器  aos\_register\_event\_filter(EV\_WIFI, wifi\_service\_event, NULL);  netmgr\_init(); //定义与注册wifi回调函数  netmgr\_start(false);  aos\_cli\_register\_command(&mqttcmd);  aos\_loop\_run();  return 0;  } |

图 15 应用入口函数

5、static void mqtt\_publish (void \*pclient)

void mqtt\_publish(void \*parms)函数是本次mqtt例程中的mqtt上云发送数据函数，该函数所实现的主要功能是：

* 订阅相关*TOPIC*
* 每隔3S读取环境传感器获得的温湿度数据，并将数据按照指定的json格式进行封装
* 循环发送封装好的温湿度数据到指定的*TOPIC*。

|  |
| --- |
| static void mqtt\_work (void \*parms) {  uint32\_t temp\_data = 0;  uint64\_t temp\_timestamp = 0;  uint32\_t humi\_data = 0;  uint64\_t humi\_timestamp = 0;  int rc = -1; char param[256] = {0};  /\*订阅 *TOPIC*\*/  if (is\_subscribed == 0) {  /\* Subscribe the specific *Topic* \*/  rc = IOT\_MQTT\_Subscribe(pclient, *ALINK*\_*TOPIC*\_PROP\_POSTRSP,  IOTX\_MQTT\_QOS0, handle\_prop\_postrsp, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  } is\_subscribed = 1;  aos\_schedule\_call(ota\_init, NULL);  }  #ifndef MQTT\_PRESS\_TEST  else {  ----------------------------------省略部分代码-----------------------------------------------------  /\*读取环境传感器数据，获取环境温湿度数据\*/  get\_humi\_data(&humi\_data,&humi\_timestamp);  get\_temp\_data(&temp\_data, &temp\_timestamp);  /\*按照高级版数据格式封装数据\*/  sprintf(param, PROP\_POST\_FORMAT, (float)temp\_data\*0.01, (float)humi\_data\*0.01);  int msg\_len = sprintf(msg\_pub, *ALINK*\_BODY\_FORMAT, cnt, *ALINK*\_METHOD\_PROP\_POST, param);  *Topic*\_msg.payload = (void \*)msg\_pub;  *Topic*\_msg.payload\_len = msg\_len;  /\*将封装好的温湿度*属性*数据发送到指定*TOPIC*\*/  rc = IOT\_MQTT\_Publish(pclient, *ALINK*\_*TOPIC*\_PROP\_POST, &*Topic*\_msg);  LOG("packet-id=%u, publish *Topic* msg=%s", (uint32\_t)rc, msg\_pub);  if (cnt < 200) {  aos\_post\_delayed\_action(3000, mqtt\_work, NULL);  } else {  aos\_cancel\_delayed\_action(3000, mqtt\_work, NULL);  mqtt\_unsubscribe(*TOPIC*\_GET);  aos\_msleep(200);  mqtt\_deinit\_instance();  is\_subscribed = 0;  cnt = 0;  }  #endif  } |

图 19 MQTT例程数据上云主函数

**修改设备相关参数**

与基本版相同，在高级版的上报温湿度的实验中，我们仍然需要修改*设备*端的三元组信息和新建的平台端*设备*相同，具体的修改方式参考基础版实验，在这里不再赘述。

**编译下载**

工程编译与下载的方式和LP应用开发教程2中相同，详情可参考开发教程2中的*设备*开发编译下载章节，在此处不再赘述。

至此，工程代码烧录进入开发板当中。打开串口助手即可查看程序的运行输出信息，

**运行结果**

与基础版相类似，我们将AIoTKIT开发板通过wifi连接物联网平台。当*设备*向平台上报温湿度*属性*值时，运行“SimpleMQConsumer”代码的本机就会从队列中消费该消息，我们可以在IDEA控制台上查看到消费的消息内容。



收到的消息内容为{"temperature":27.6,"humidity":52.0}，即*设备*上报的温湿度数据。以上我们可以看到消息成功地从*设备*经过消息队列MQ流转到了用户*服务*器上，之后只需通过在用户*服务*器上编写应用程序来处理*设备*上报的消息即可实现简单的物联网应用。

**课程小结**

本次课程学习了SQL语句的编写、*规则引擎*的设置，了解了物联网平台可实现的各类转发操作，并通过两个数据流转实例帮助读者体验了物联网平台的实际应用价值，在本课程中两个实验的基础上进行简单的修改和业务代码的开发，读者便可以实现自己的小型物联网应用了。