目录

[课程内容 2](#_Toc522794117)

[课程目标 2](#_Toc522794118)

[课程准备 2](#_Toc522794119)

[课程知识点 3](#_Toc522794120)

[IoT Hub功能 3](#_Toc522794121)

[Topic 4](#_Toc522794122)

[通信模式 7](#_Toc522794123)

[物联网平台侧开发 8](#_Toc522794124)

[创建产品 8](#_Toc522794125)

[添加设备 9](#_Toc522794126)

[设备端开发 9](#_Toc522794127)

[新建项目 9](#_Toc522794128)

[代码开发 11](#_Toc522794129)

[编译下载 12](#_Toc522794130)

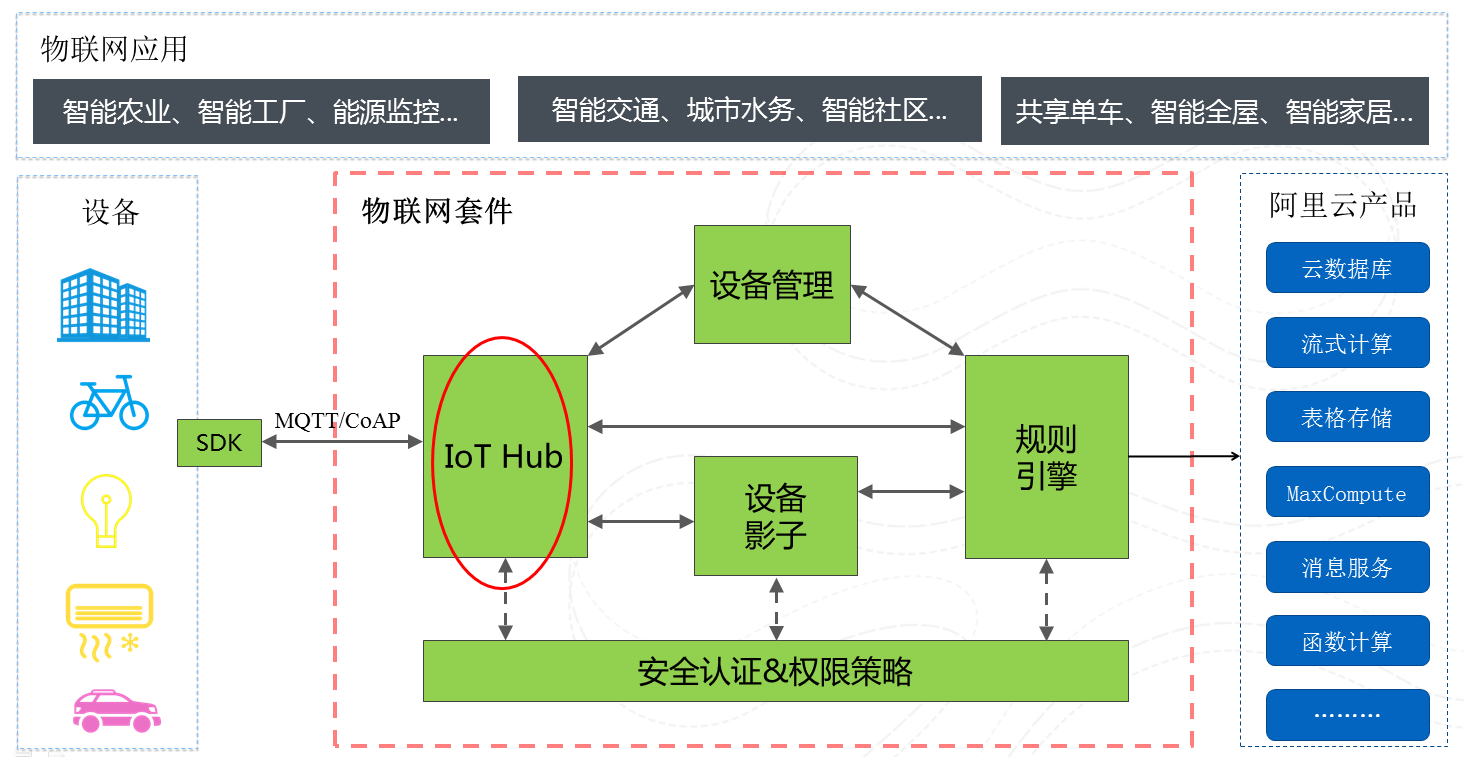
[运行调试 13](#_Toc522794131)

[基础版与高级版设备接入对比 14](#_Toc522794132)

[课程小结 16](#_Toc522794133)

**基于IoT Hub的真实设备接入**

在上一课中，我们介绍了Node.js虚拟设备接入物联网平台的方法，以帮助读者快速入门。但仅仅掌握接入的步骤是远远不够的，本次课程将对物联网平台中负责设备连接与通信的IoT Hub组件进行详细的介绍，并带领读者使用真实的开发板设备完成物联网平台的接入与通信。



**课程内容**

* 介绍IoT Hub组件的功能
* 介绍MQTT协议的通信模式与Topic概念
* 使用AIoTKIT开发板接入物联网平台

**课程目标**

* 理解发布/订阅异步通信机制
* 理解Topic概念并能够熟练使用
* 实现AIoTKIT的平台接入与双向通信

**课程准备**

* 运行Windows系统的 PC机一台
* AIoTKIT开发板一块
* micro USB连接线
* mk3080 wifi模组
* 安装有AliOS Things Studio插件的VSCode
* AliOS Things 1.3.3版本或更高版本
* ST-Link 驱动程序
* 开通阿里云物联网平台*产品*

**课程知识点**

本次课程主要介绍物联网平台的IoT Hub组件，除了组件功能，为了加深读者的理解与掌握，还将对涉及到的MQTT协议中的发布/订阅通信机制和Topic概念进行讲解。

**IoT Hub功能**

IoT Hub组件作为设备和物联网应用程序发布和接收消息的安全通道，帮助设备连接物联网平台，并进行安全可靠的数据通信。具备几大功能：

* 高性能扩展能力

IoT Hub支持线性动态扩展，可以支撑十亿设备同时连接。

* 全链路加密

整个通信链路以RSA，AES加密，保证数据传输的安全。

* 支持设备多种协议接入

目前IoT Hub支持CoAP协议，开源的MQTT协议和HTTPS协议接入。

* 消息实时到达

当设备与IoT Hub建立数据通道后，IoT Hub会与设备保持长连接，减少握手时间，保证消息的实时到达。

* 支持数据透传

IoT Hub支持二进制透传的方式将设备数据传到自己的服务器上，物联网平台不会保存设备业务数据，从而保证用户对数据的安全可控性。

* 支持多种通信模式

IoT Hub支持RRPC以及Pub/Sub两种通信模式，满足用户不同的应用场景。

**MQTT协议**

* 协议简介

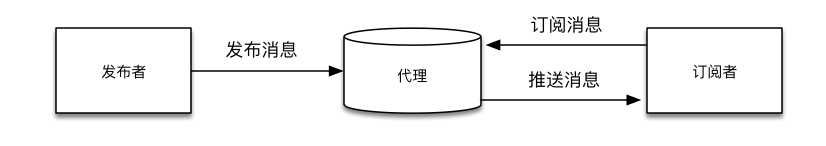
IoT Hub支持CoAP协议， MQTT协议和HTTPS协议接入，本课程中均采用MQTT协议接入方式，即设备基于MQTT协议接入物联网平台并与平台进行双向通信。

MQTT协议（Message Queuing Telemetry Transport Protocol）全称为消息队列遥感传输协议，是IBM开发的一个即时通讯协议，是为大量计算能力有限，且工作在低带宽，不可靠的网络的远程传感器和控制设备通讯而设计的协议。MQTT协议运行在TCP协议栈之上，是一种基于轻量级代理的发布/订阅模式的消息传输协议，能够提供有序、可靠的双向网络连接。

MQTT协议可以通过极少的代码和有限的带宽为远程设备提供实时可靠的消息服务，其低开销和低带宽占用的特点使得它在物联网、小型设备和移动应用等方面有着较为广泛的应用。

* 工作原理

MQTT协议中有三种角色：代理（Broker）、发布者（Publish）和订阅者（Subscribe）。



订阅者：某些设备对特定的信息感兴趣，并希望订阅它，这些设备就叫订阅者。

发布者：负责为其他设备生成数据的就叫发布者。一个设备可以既是订阅者也是发布者。

代理：负责将来自发布者的消息进行存储处理，并将这些消息发送到正确的订阅者中去。

发布/订阅模式使发布者和订阅者互不知道对方的存在，只知道代理服务器，实现了发布者和订阅者的解耦。

MQTT协议传输的消息分为Topic和payload两部分。

Topic：可以理解为消息的类型，发布和订阅操作都是基于Topic进行的。发布者往某个Topic中发布消息，订阅者订阅该Topic后，即可收到消息。

payload：可以理解为消息的内容，是指订阅者具体要使用的消息内容。

* 协议特点

MQTT协议是轻量、简单、开放且易于实现的协议，特别适用于机器与机器（M2M）通信和物联网（IoT）等资源受限的场景中。其特点包括：

1、发布/订阅消息模式提供了一对多消息分发，实现了与应用程序的解耦。

2、具有屏蔽payload内容的消息传输机制。

3、用户可以根据需要从三种传输消息的服务质量（QoS）中选择。

|  |  |  |
| --- | --- | --- |
| QoS等级 | 传输次数 | 说明 |
| 0 | 至多一次 | 可能发生消息丢失 |
| 1 | 至少一次 | 确保消息到达，但可能发生消息重复 |
| 2 | 只有一次 | 确保消息到达一次 |

4、最小化数据传输和协议交换，协议头部仅2字节，以减少网络流量

5、具有通知机制，当连接异常中断时通知传输双方。

**Topic**

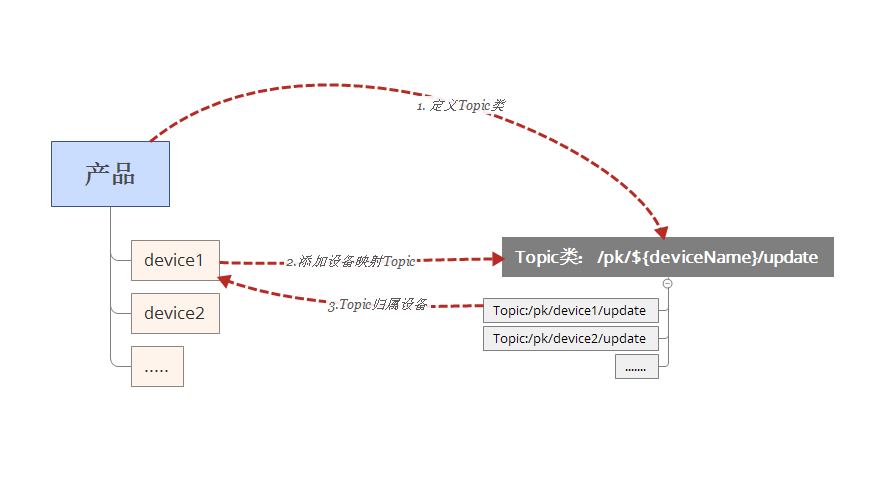
物联网平台与设备之间基于Topic进行消息的路由转发，每类Topic都具有自身的设备操作权限：发布或订阅，设备端通过向物联网平台发布或订阅消息实现通信。

|  |  |
| --- | --- |
| 发布 | 设备可以往Topic中发布消息 |
| 订阅 | 设备可以从Topic中订阅消息 |

下面我们介绍Topic类与Topic，Topic类是针对*产品*的概念，Topic是针对设备的概念。

* Topic类

为了方便海量设备基于海量Topic进行通信，简化授权操作，物联网平台增加Topic类的概念。用户创建*产品*后，只需要在*产品*下定义Topic类或者直接使用平台为*产品*自动创建的默认Topic类，平台会自动将Topic映射到该*产品*下的每个设备上，使用户不需要为每个设备单独授权Topic。



也就是说，Topic类是一类Topic的集合，举个例子，Topic类：/productKey/${deviceName}/update（update为topic名）是具体Topic：/productKey /device1/update， /productKey /device2/update的集合。

对于基础版*产品*的Topic类以及高级版*产品*的部分Topic类，前两个类目已经规定好，第一个${productKey}代表*产品*标识productKey，第二个${deviceName}通配deviceName，我们可以在*产品*的消息通信页面查看*产品*的Topic。





上图中的Topic类都是物联网平台自动生成的默认Topic类，除此以外，用户还可以自定义Topic类，方便用户根据业务需求进行更为灵活的消息通信。用户Topic类一般也采用/${productKey}/${deviceName}开头，只能包含字母，数字和下划线(\_)，每级类目不能为空。





平台上还有一类由系统预定义的，不以${productKey}开头的Topic类，称为系统Topic类。例如高级版中针对物模型所提供的Topic类一般采用/sys/开头，固件升级相关的Topic类采用/ota/开头，设备影子的Topic类采用/shadow/开头。

物联网平台定义的系统Topic详细列表可参见链接：https://help.aliyun.com/document\_detail/73732.html

* Topic

虽然介绍了很多关于*产品*Topic类的内容，但是我们不能直接将Topic类用于通信，真正用于消息通信的必须是具体的Topic。

Topic的格式和Topic类一致，是根据deviceName从Topic类中映射动态创建而来，只有当deviceName存在时，对应的Topic才会创建。

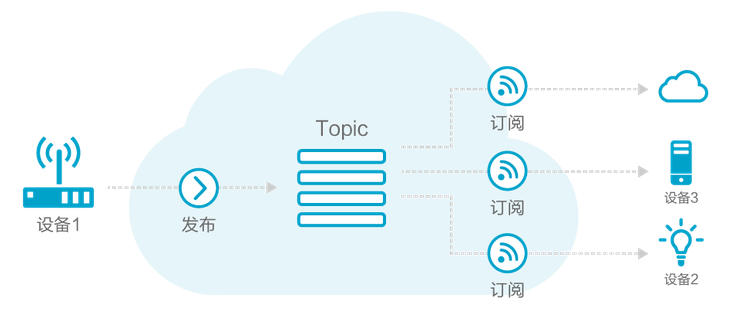
Topic归属于特定的设备，只能被该设备用来通信，不能被其他设备用于通信。举个例子，Topic：/productKey/device1/update归属于设备device1，只能被device1用于发布订阅消息，而不能被设备device2用于发布订阅消息。

**通信模式**

物联网平台支持两种通信模式，PUB/SUB以及RRPC，用户可以根据业务需求灵活选择。

* PUB/SUB

在Topic的基础上，物联网平台基于Pub/Sub机制进行消息的路由转发，让设备端可以发布或订阅消息，实现通信。



设备进行发布或订阅操作的“基本单位”就是Topic，IoT平台维护所有Topic的发布订阅用户列表，当发布者将消息发布到Topic后，IoT平台会检查该Topic的所有订阅用户，然后将消息转发给所有订阅了该Topic的设备，这些设备就会收到该Topic内的消息内容。

* RRPC

物联网平台还基于开源MQTT协议封装了同步的通信模式RRPC。如果采用这种通信模式，服务端下发指令给设备时需要同步得到设备端的响应。

**物联网平台侧开发**

**创建产品**

本次课程以AIoTKIT开发板为例，介绍真实设备的接入开发。在进行开发和接入之前，我们首先需要在物联网平台上创建好*产品*和设备。

登录物联网平台控制台https://iot.console.aliyun.com/，选择区域为“华东2（上海）”，后续的*产品*和设备都会被创建在该区域下。



在“*产品*管理”页面，点击右侧的“创建*产品*”按钮。与“虚拟设备接入”实验不同，本课程中我们选择*产品*版本为“基础版”，学习基础版*产品*和设备的接入。输入*产品*名称，并选择节点类型后，确定即可。注意：*产品*名称必须保证在账号内唯一。此处创建的示例*产品*名称为Example2，节点类型为“设备”。



**添加设备**

*产品*创建成功后，进入“设备管理”栏，选择刚刚创建好的基础版*产品*，点击“添加设备”，输入设备名称即可。用户可以自定义设备名称，保证名称在*产品*内唯一即可。



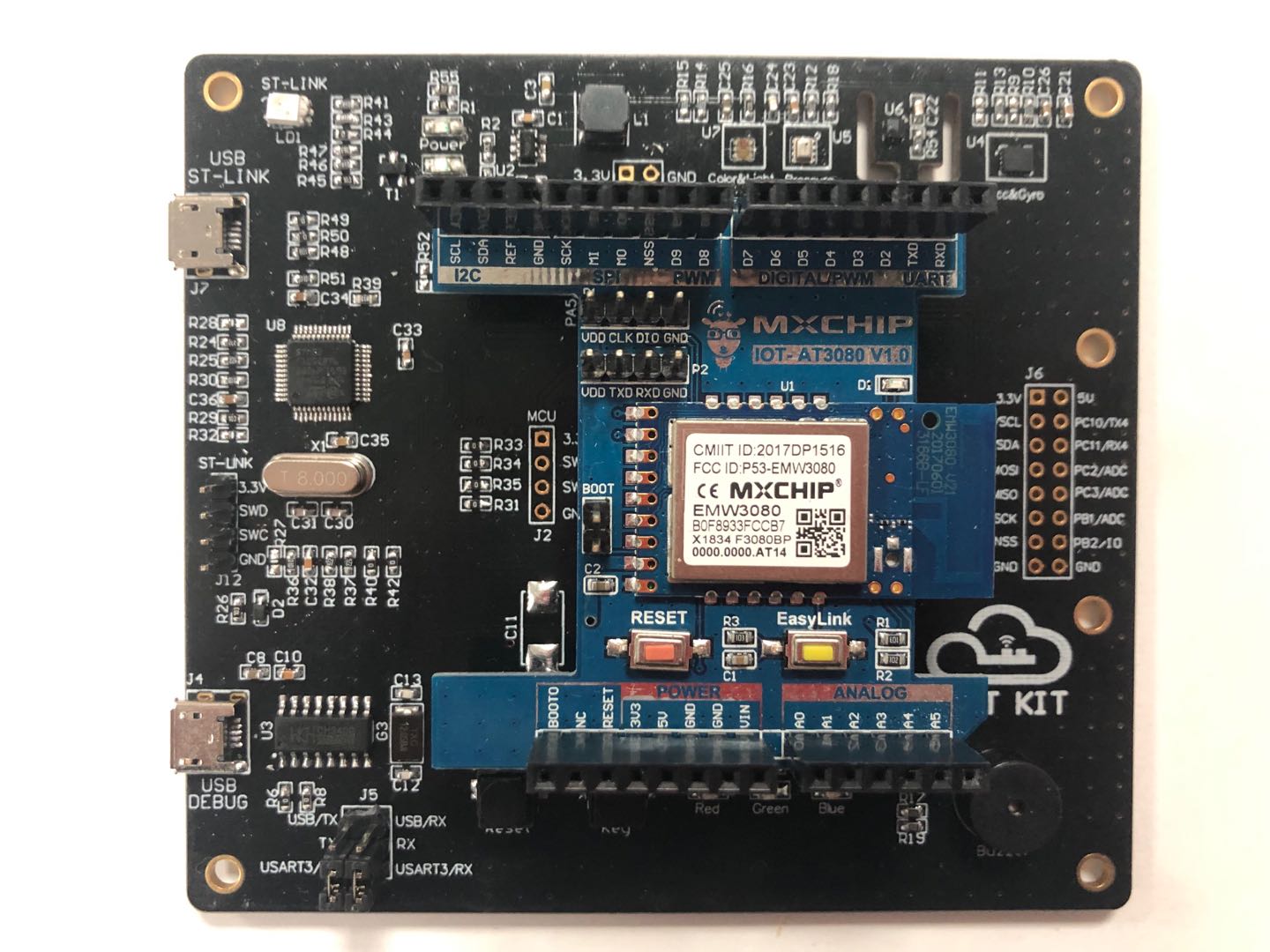
设备创建完成后，平台会自动生成该设备的三元组信息，该信息在后续设备端开发中需要写入设备，用于设备接入物联网平台时的鉴权。

完成*产品*和设备的创建之后，接下来我们进行设备端的开发工作。

**设备端开发**

本次课程使用的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。由于开发板运行AliOS Things物联网操作系统，因此可以十分便捷地实现与云平台的无缝对接。

具体的开发板与wifi模组的连接方式如下图所示：



开发板与模组的连接方式

**新建项目**

适用于本次以及后面的设备端代码位置位于：

https://gitee.com/liubozju/alikit\_developer。

1、Git工具下载

我们已经安装了git bash环境，可以直接使用Git工具来下载代码，命令如下：

|  |
| --- |
| 从GitHub克隆： git clone https://gitee.com/liubozju/alikit\_developer |

git clone命令默认从远程仓库下载master分支。但是对应于stm32f412开发板的代码位于分支developer上，所以下载代码以后我们要切换到developer分支上面去。具体的流程如下：

|  |
| --- |
| git branch –a //查询所有分支  git checkout developer //切换分支  git pull //同步代码 |

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mqttapp例程，开发板选择为AIoTKIT开发板。

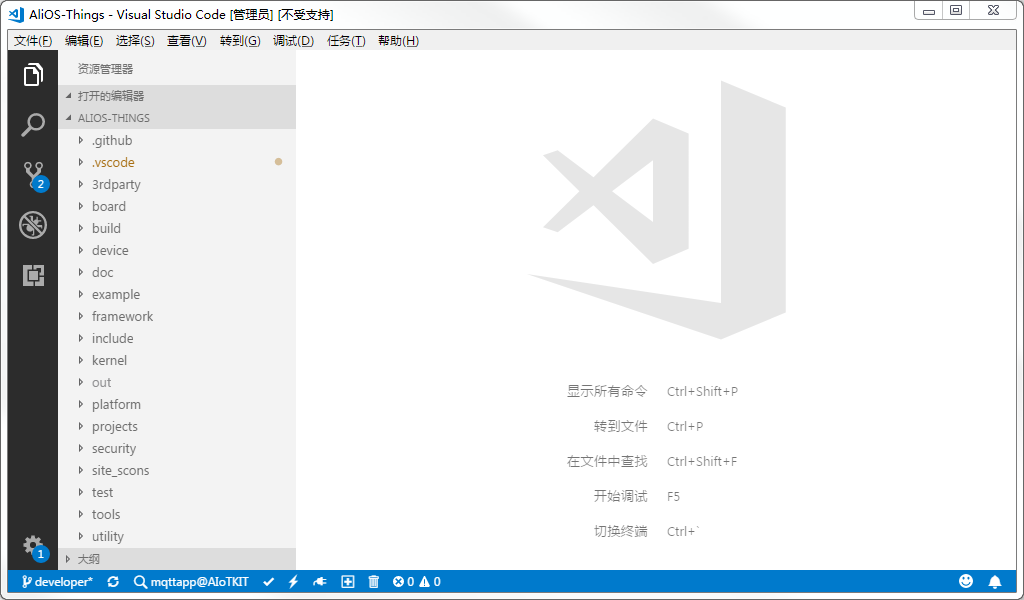


Figure 1 切换分支到developer

点击工程界面左下角，选择项目模板，设置项目为mqtt例程。

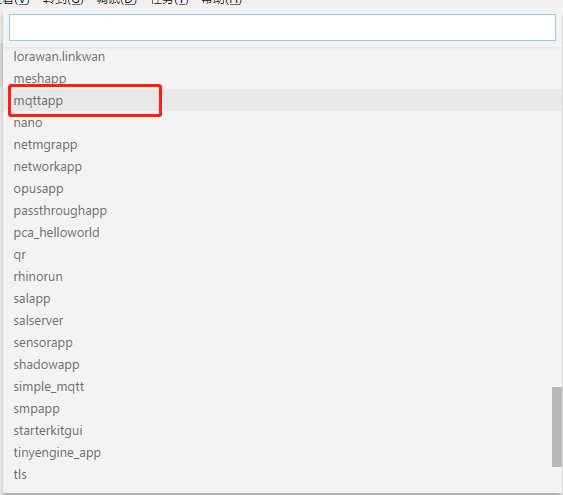


Figure 2 设置项目例程为mqttapp

设置项目模板后选择开发板为AIoTKIT开发板。

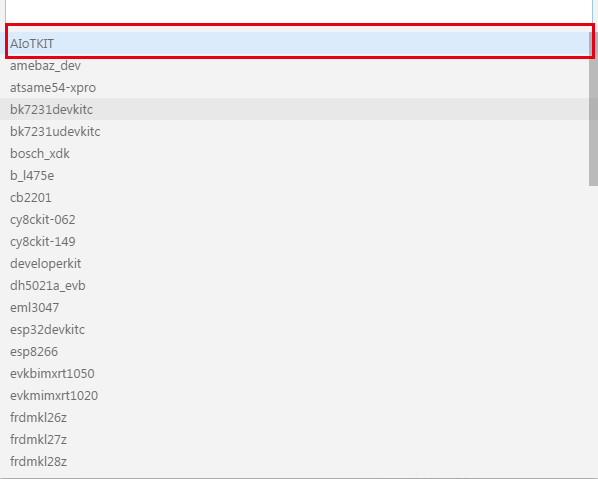


Figure 3 设置开发板为AIoTKIT

**主要代码讲解**

在进入mqttapp例程代码讲解之前，我们首先介绍一下该mqttapp的编译框架。

1) mqttapp.mk编译文件分析

顶层example/mqttapp/mqttapp.mk文件如下图所示：

|  |
| --- |
| NAME := mqttapp  GLOBAL\_DEFINES+= MQTT\_TEST ALIOT\_DEBUG IOTX\_DEBUG USE\_LPTHREAD  CONFIG\_OTA\_CH = mqtt  $(NAME)\_SOURCES := mqtt-example.c  $(NAME)\_COMPONENTS := cli protocol.linkkit.iotkit connectivity.mqtt cjson fota netmgr framework.common  LWIP := 0  ifeq ($(LWIP),1)  $(NAME)\_COMPONENTS += protocols.net  no\_with\_lwip := 0  endif  ifeq ($(no\_tls),1)  GLOBAL\_DEFINES += IOTX\_WITHOUT\_TLS MQTT\_DIRECT  endif  ifeq ($(press\_test),1)  GLOBAL\_DEFINES += MQTT\_PRESS\_TEST  endif |

图 11 mqttapp.mk 编译文件代码

由该mk文件我们可以知道，编译的顶层源程序为mqtt-example.c。在该mk文件中还包含了其他的一些组件，比如cli组件、iotkit组件、mqtt组件、fota组件、netmgr组件和framework组价等。

其中，cli组件为命令行相关的组件，代码位置位于/tools/cli文件夹下。

iotkit组件为Iot连接工具包，主要负责设备端和阿里云物联网平台端的连接、鉴权认证等功能。其代码位置位于/framework/protocol/linkkit/iotkit文件夹下。

mqtt组件是与mqtt协议相关的部分程序代码，主要负责实现mqtt协议的相关方法，mqtt客户端的建立与取消等操作。其代码位置位于/framework/connectivity/mqtt文件夹下。

cjson组件主要应用在数据格式转换方面，因为设备端和平台端是以json格式进行数据通信，所以应用cjson组价可以成功的将要发送的数据转换为json格式，其代码位置位于/utility/cjson文件夹下。

fota组件主要用于固件升级，在本例程中没有用到，详情请参考fota教程。

netmgr组件主要用于配网的功能，其代码位置位于/framework/netmger。

除了以上包含的组件和指定源程序之外，在mqttapp.mk文件中，还进行了全局的宏定义：

GLOBAL\_DEFINES+= MQTT\_TEST ALIOT\_DEBUG IOTX\_DEBUG USE\_LPTHREAD

其中MQTT\_TEST宏定义用于指定在iotkit组件/sdk-encap/imports/iot\_import\_product.h文件中的三元组信息。



图 12 MQTT\_TEST指定三元组信息示例

在mqttapp.mk文件中具有条件编译的选项。

|  |
| --- |
| ifeq ($(press\_test),1)  GLOBAL\_DEFINES += MQTT\_PRESS\_TEST  endif |

图 13 MQTT压力测试条件编译选项

该条件编译表明如果设置press\_test为1（aos make mqttapp@ AIoTKIT press\_test=1），那么就增加一个全局的宏定义MQTT\_PRESS\_TEST。该全局宏定义用于打开压力测试开关。在本次的mqtt例程代码中并没有设置此选项。

|  |
| --- |
| ifeq ($(no\_tls),1)  GLOBAL\_DEFINES += IOTX\_WITHOUT\_TLS MQTT\_DIRECT  endif |

图 14 TSL条件编译选项

mqttapp例程默认使用的是MQTT客户端直连方式，带有TLS。如果不使用TLS的话，可以使用此条件编译指令关闭TLS，不过建议将TLS打开。

2) mqtt 例程代码解析

本节中，MQTT例程代码中mqtt-example.c是通过AT联网指令联网并上传与接收数据。我们将详细介绍mqtt-example.c文件，即使用AIoTKIT开发板和外接mk3080模组进行联网的例程。

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[])  {  #if AOS\_ATCMD /\*AT指令初始化\*／  at.set\_mode(ASYN);  at.init(&at\_uart, AT\_RECV\_DELIMITER, AT\_SEND\_DELIMITER, 1000); /\*AT命令初始化\*／  #endif  #ifdef WITH\_SAL /\*SAL框架初始化\*／  sal\_init();  #endif  aos\_set\_log\_level(AOS\_LL\_DEBUG); /\*设置LOG等级\*／  aos\_register\_event\_filter(EV\_WIFI, wifi\_service\_event, NULL); /\*监听WIFI事件\*／  netmgr\_init();//用于对 netmgr 组件进行初始化。  netmgr\_start(false); /\*是可选的，它的作用是启动配网流程。\*／  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（）。

|  |
| --- |
| static void wifi\_service\_event(input\_event\_t \*event, void \*priv\_data) {  if (event->type != EV\_WIFI) {  return;  }  if (event->code != CODE\_WIFI\_ON\_GOT\_IP) {  return;  }  LOG("wifi\_service\_event!");  mqtt\_client\_example();  } |

图 16 wifi事件处理函数

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。

|  |
| --- |
| int mqtt\_client\_example(void)  {  memset(&mqtt, 0, sizeof(MqttContext));  /\*获取设备连接时的相关参数\*/  strncpy(mqtt.productKey,PRODUCT\_KEY,sizeof(mqtt.productKey)- 1);  strncpy(mqtt.deviceName,DEVICE\_NAME,sizeof(mqtt.deviceName)- 1);  strncpy(mqtt.deviceSecret,DEVICE\_SECRET,sizeof(mqtt.deviceSecret) - 1);  mqtt.max\_msg\_size = MSG\_LEN\_MAX; /\*消息的大小限制\*/  mqtt.max\_msgq\_size = 8; /\*消息队列的大小限制\*/  mqtt.event\_handler = smartled\_event\_handler;  mqtt.delete\_subdev = NULL;  if (mqtt\_init\_instance(mqtt.productKey, mqtt.deviceName, mqtt.deviceSecret, mqtt.max\_msg\_size) < 0) { /\*初始化并建立MQTT连接\*/  LOG("mqtt\_init\_instance failed\n");  return -1;  }  aos\_register\_event\_filter(EV\_SYS, mqtt\_service\_event, NULL);  /\*监听mqtt服务事件\*/  return 0;  } |

图 17 设备鉴权注册函数

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。

|  |
| --- |
| static void mqtt\_service\_event(input\_event\_t \*event, void \*priv\_data) {  if (event->type != EV\_SYS) {  return;  }  if (event->code != CODE\_SYS\_ON\_MQTT\_READ) {  return;  }  LOG("mqtt\_service\_event!"); /\*输出LOG信息\*/  mqtt\_work(NULL);  /\*调用本次例程的主要函数mqtt\_work();\*/  } |

图 18 wifi事件处理函数

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

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
* 向指定TOPIC循环发送自定义数据。该函数中发送的自定义数据为自身模拟的温度数据，该温度值随着循环次数的增加而不断自加，循环两百次后终止发送数据。如果需要可以将要发送到云端的数据内容修改为自身需要的内容数据。

|  |
| --- |
| static void mqtt\_work (void \*parms) {  int rc = -1;  if(is\_subscribed == 0) { /\*订阅TOPIC标记，初始化为0\*/  /\* Subscribe the specific topic \*//\*订阅GET TOPIC\*\*/  rc = mqtt\_subscribe(TOPIC\_GET, mqtt\_sub\_callback, NULL);  if (rc<0) {  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  }  is\_subscribed = 1; /\*置位订阅标记\*/  aos\_schedule\_call(ota\_init, NULL);  }  #ifndef MQTT\_PRESS\_TEST  else{ /\*已经订阅了TOPIC\*/  /\* Generate topic message \*///产生要发送的数据内容  int msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"attr\_name\":\"temperature\", \"attr\_value\":\"%d\"}", cnt);  if (msg\_len < 0) {  LOG("Error occur! Exit program"); }  /\*发送预定数据到指定的TOPIC\*/  rc = mqtt\_publish(TOPIC\_UPDATE, IOTX\_MQTT\_QOS0, msg\_pub, msg\_len);  /\*打印输出发送的message\*/  LOG("packet-id=%u, publish topic msg=%s", (uint32\_t)rc, msg\_pub);  }  cnt++;  if(cnt < 200) {  /\*每隔3S重新发送一次数据\*/  aos\_post\_delayed\_action(3000, mqtt\_work, NULL);  } else { /\*送超过两百次，则取消订阅TOPIC，清除订阅标记，释放存储区，\*/  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mqttapp@AIoTKIT工程。在本小节中，我们需要修改官方mqttapp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/mqttapp,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：Protocl/linkkit/iotkit/sdk-encap/imports/iot\_import\_product.h。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

开发板通过wifi模组将消息上发至物联网平台的函数为example/mqttapp目录下的mqtt-example.c文件中的mqtt\_publish(TOPIC\_UPDATE, IOTX\_MQTT\_QOS1, msg\_pub, msg\_len)函数。该函数将msg\_pub 消息发送至TOPIC\_UPDATE 对应的topic中，本课程中设备发布消息的Topic为update Topic，读者也可根据需要自行修改。



Figure 4 发布消息对应 TOPIC

设备订阅TOPIC\_GET对应的topic实现下行消息的接收。订阅函数将下行消息处理函数mqtt\_sub\_callback注册到回调函数中，在接收到下行消息时将其打印出来。本课程中设备订阅的Topic为get Topic，读者也可以根据需要自行修改。



Figure 5 订阅 get TOPIC对应回调函数

**编译下载**

1. 工程编译

如下图所示，点击Visual Studio Code状态栏中的编译按钮，编译程序。

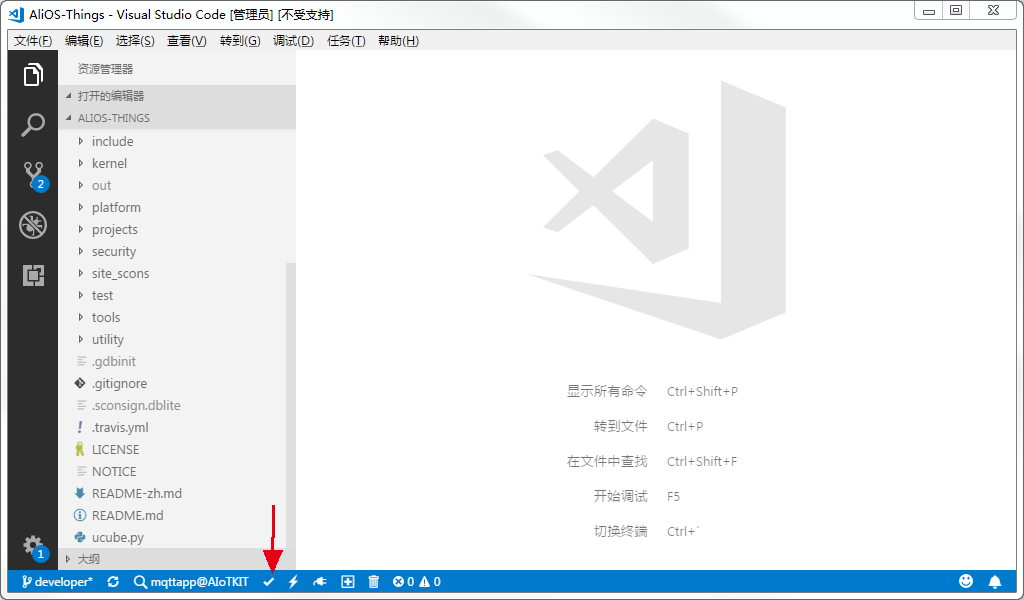


图 9 工程编译

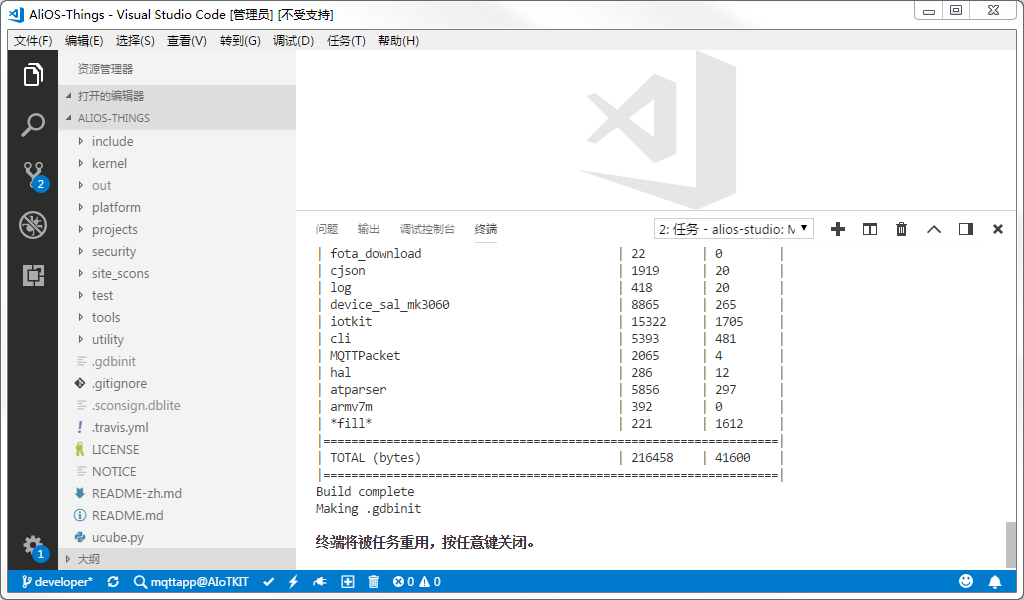
编译时，在 Visual Studio Code的输出栏中可以看到编译的详细 log。编译成功时可以Visual Studio Code的输出栏看到如下信息：

图 10 工程编译成功界面

1. 工程烧录

首先将mk3080模组和AIoTKIT开发板正确连接，并且使用USB串口线将开发板和电脑连接起来。当编译完成后,在确保设备和电脑连接无误且可以通过Visual Studio Code 与设备建立连接的情况下，点击烧录按钮。

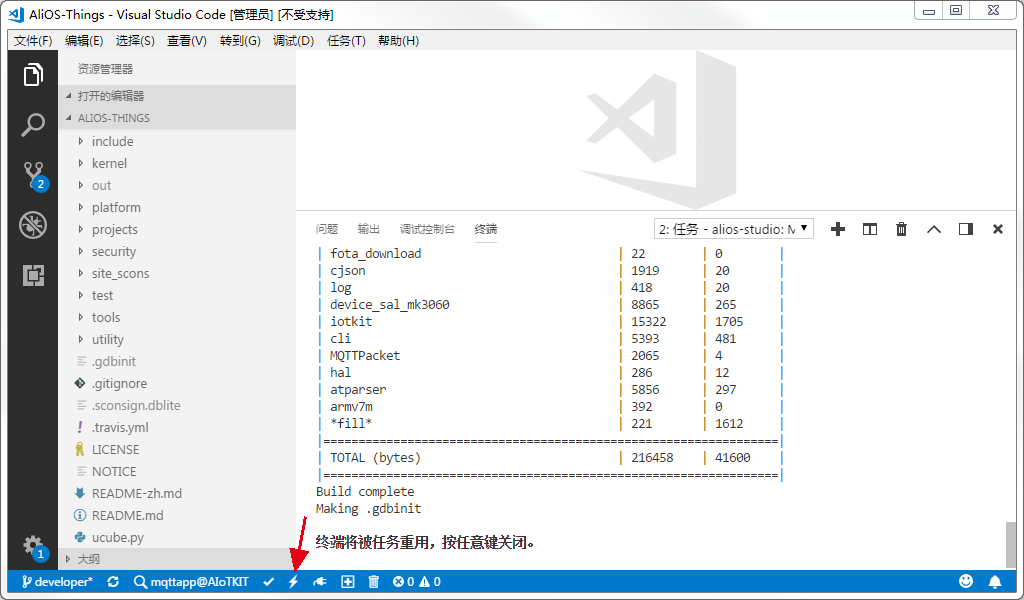


图 11 工程烧录

**运行调试**

经过编译下载操作，相对应的mqttapp已经成功烧写到了开发板当中。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，即在Visual Studio Code的终端下输入netmgr connect WIFINAME WIFIPASSWORD（WIFI名称以及WIFI密码）联网指令如下图所示：

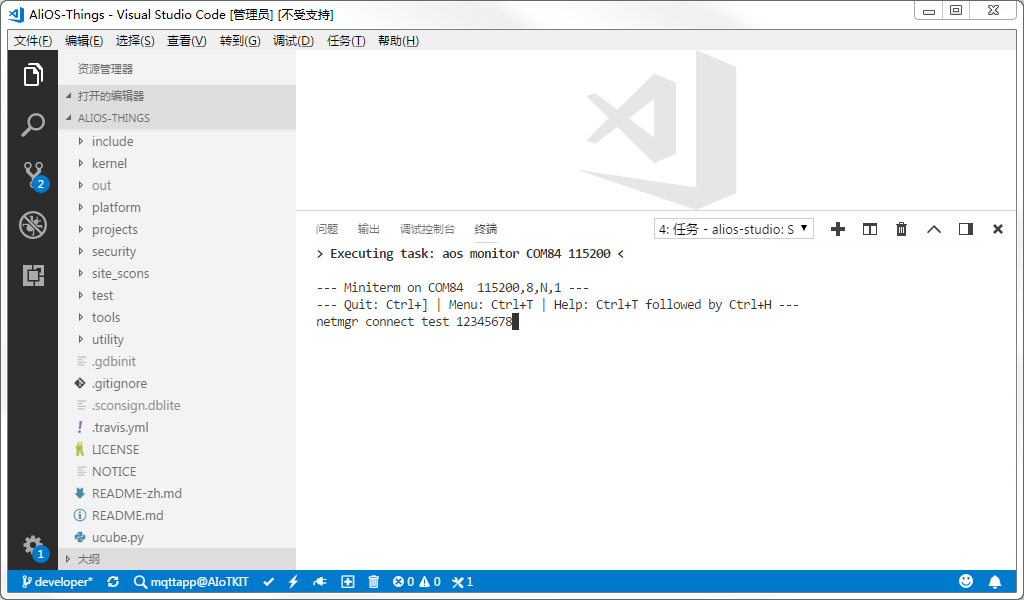


图 12 输入设备联网指令

正常联网后，mqttapp会真正开始运行。下图为mqtt运行日志截图：

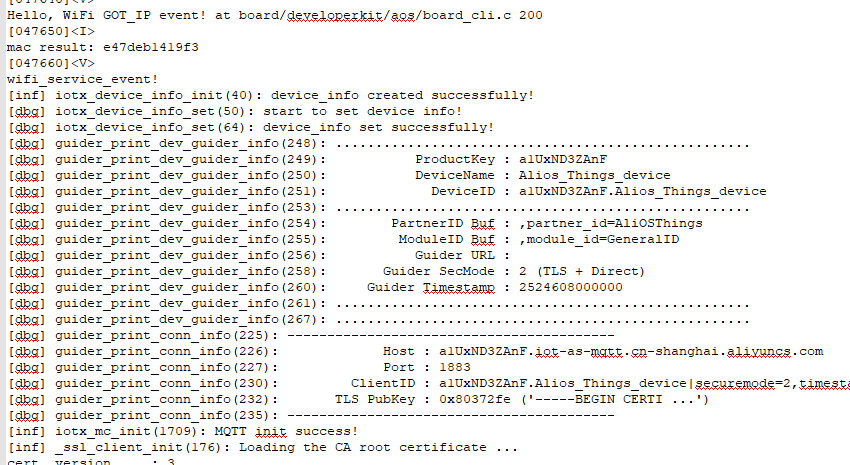


图 13 mqtt例程运行日志

联网成功以后，程序会在每一个循环中上报自定义温度数据，且上报的TOPIC为TOPIC\_UPDATE。由IoT平台控制台可知，该TOPIC具有发布权限，具体产生的日志截图如下图所示：

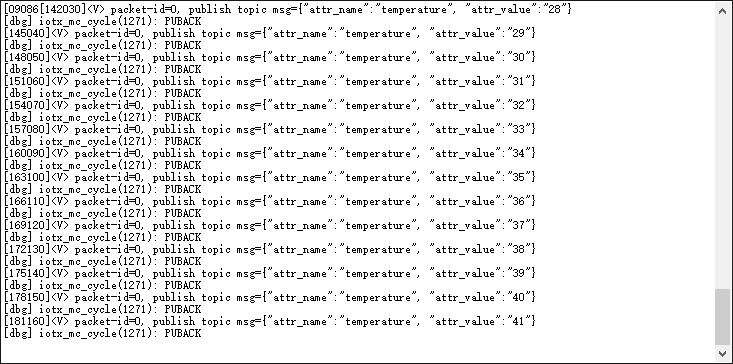


图 14 设备上报数据日志

由设备端上传到阿里云物联网控制台的消息同样可以在物联网控制台上查看消息上报记录，点击*产品*管理->查看->日志服务->上行消息分析，即可在云端查询到设备相关的日志信息：

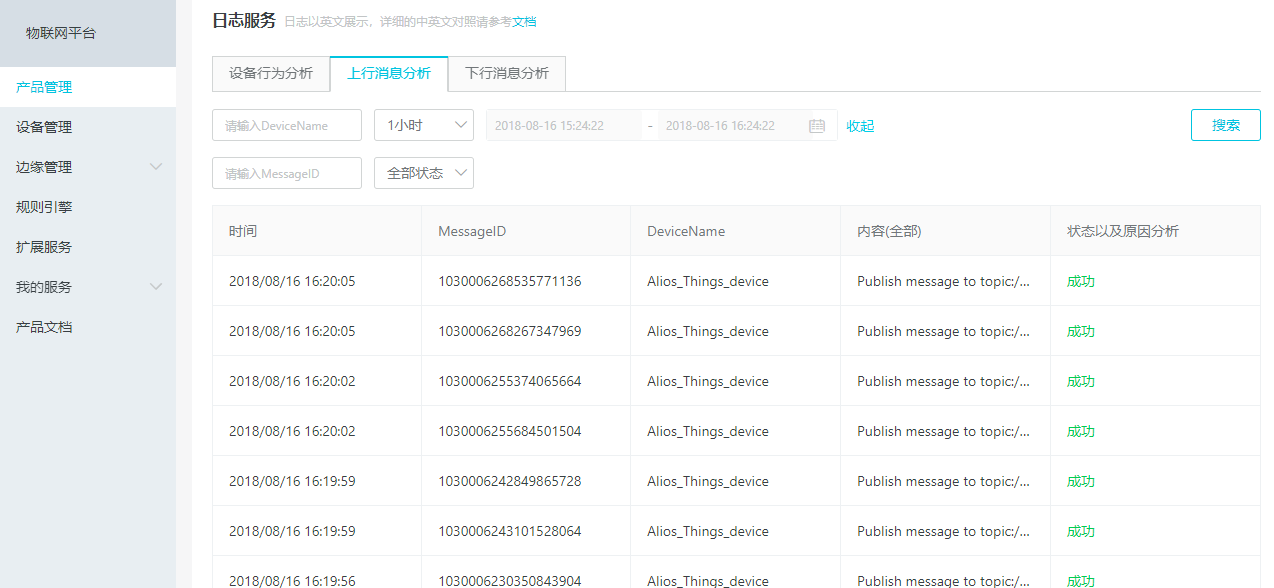


图 15 云端设备上报消息日志

同时，在控制台上，我们可以直接向开发板下发消息。由于设备在代码中订阅了get Topic，因此此处我们选择get Topic，点击右侧的“发布消息”。在消息弹窗中填写消息内容后确定即可。



图16 云端发送消息内容

此时，终端将接收到下行的消息，并将内容打印在日志窗口上。

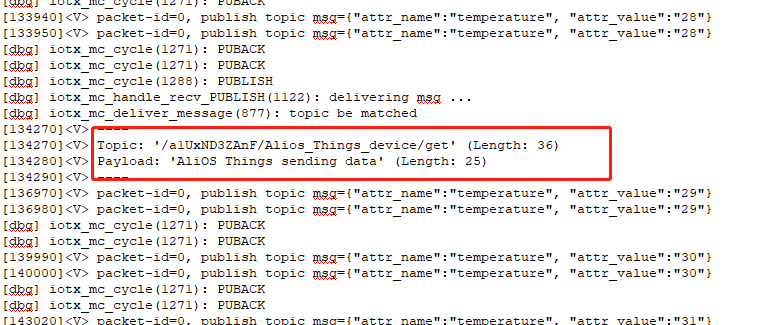


图17 设备端显示云端下发消息

至此，基于AIoTKIT设备的物联网平台接入就已经全部完成了！

**基础版与高级版设备接入对比**

本次课程与上次课程相比，除了接入平台的设备不同，使用的物联网平台的规格也不相同。经过这两课的学习，相信大家已经了解了基础版设备和高级版设备接入物联网平台的流程，那么他们在接入平台时有哪些区别呢？

* 功能定义

由于高级版物联网平台具备基础版没有的“物模型”设备管理能力，用户在创建高级版*产品*时可以为*产品*定义属性、服务和事件等功能，后续设备与平台的通信也都基于这些功能进行。

* Topic

高级版设备和基础版设备与物联网平台进行发布/订阅机制通信时使用的Topic不同，平台为高级版设备默认定义的Topic如下图。高级版根据定义好的物模型中的属性、服务、事件功能，将消息发布到对应的Topic中或者从对应的Topic中订阅获取消息。



平台为基础版设备默认定义的Topic如下图所示，用户根据需要的操作权限为设备选择Topic使用即可。



* 数据格式

基础版设备通过Topic传输的数据格式可以完全由用户自行进行定义，而高级版设备与物联网平台间必须采用平台根据物模型定义好的数据格式进行通信。

下面我们以设备上报温湿度数据为例进行说明。

|  |  |  |
| --- | --- | --- |
|  | 基础版 | 高级版 |
| 功能定义 | × | 定义“温度”与“湿度”属性 |
| Topic | 具有发布权限的Topic即可，如/a1eAzw8oWqy/${deviceName}/update | 属性上报Topic： /sys/a149PHrNEKg/${deviceName}/thing/event/property/post |
| 数据格式 | 任何自定义的JSON格式，如：  {  "Temperature" : 30.5,  "Humidity":60.7  } | {  "id" : "123",  "version":"1.0",  "params" : {  "Temperature" : 30.5,  "Humidity":60.7  },  "method":"thing.event.property.post"  } |

**课程小结**

本次课程学习了物联网平台中负责设备接入与通信的IoT Hub组件的功能以及通信过程中涉及到的Topic概念与发布/订阅通信机制，并基于IoT Hub和Topic实现了真实AIoTKIT开发板到物联网平台的接入与通信。结合本次课程与上次课程，读者可以学习虚拟设备与真实设备各自的接入流程，学习高级版设备与基础版设备各自的接入流程。在此基础上，我们可以进行后续更加复杂的实验。