**目录**

[课程内容 2](#_Toc521852942)

[课程目标 2](#_Toc521852943)

[课程准备 3](#_Toc521852944)

[课程知识点 3](#_Toc521852945)

[TSL物模型 3](#_Toc521852946)

[物联网平台侧开发 4](#_Toc521852947)

[创建产品，添加设备 4](#_Toc521852948)

[明确功能 5](#_Toc521852949)

[新增属性 6](#_Toc521852950)

[新增服务 12](#_Toc521852951)

[新增事件 13](#_Toc521852952)

[设备端开发 16](#_Toc521852953)

[新建项目 16](#_Toc521852954)

[代码开发 18](#_Toc521852955)

[编译下载 19](#_Toc521852956)

[在线调试 20](#_Toc521852957)

[属性调试 21](#_Toc521852958)

[事件调试 22](#_Toc521852959)

[服务调试 23](#_Toc521852960)

[后续课程 23](#_Toc521852961)

**TSL物模型实验**

在上一课中，我们介绍了物联网平台的IoT Hub组件，并借助AIoTKIT开发板接入物联网平台的实验，帮助读者进一步加深理解和掌握。本次课程将介绍物联网平台的另一大组件——设备管理，物联网平台为用户提供了功能丰富的设备管理服务，例如设备生命周期管理、状态管理、标签管理、拓扑关系管理、数据管理、固件升级与远程配置等等。本次课程主要聚焦于高级版物联网平台提供的“物模型”能力和“在线调试”功能，在讲解概念的同时，配合实际开发板进行调试，便于读者学习，并最终实现“温度报警”*产品*的开发。



**课程内容**

* 介绍物模型相关概念
* 介绍在线调试功能
* 基于AIoTKIT开发板，实现*产品*的物模型定义与功能的在线调试

**课程目标**

* 理解属性、服务、事件等概念，并能够完成*产品*的功能定义
* 掌握在线调试功能的使用方法

**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版本或者更高版本
* ST-Link驱动程序
* 开通阿里云物联网平台*产品*

**课程知识点**

**TSL物模型**

高级版物联网平台为用户提供了“物模型”设备管理能力，用户可以借助“物模型”将物理空间中的实体数字化，例如各类传感器，或者各类由传感器组成的设备，并在云端构建该实体的数据模型。用户通过定义物模型来定义*产品*功能，描述设备是什么、能做什么以及能够对外提供哪些服务。

用户在定义物模型时，需要定义属性、服务、事件这三类*产品*功能。

|  |  |
| --- | --- |
| 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；支持GET和SET服务，即应用可以发起对属性的读取和设置请求。 |
| 服务（Service） | 设备可以被外部调用的能力或者方法，可以设置输入参数和输出参数；与下发指令设置属性值相比，服务可以实现更为复杂的业务逻辑，执行某项特定的任务。 |
| 事件（Event） | 设备运行时的事件，一般包含需要被外部感知和处理的通知信息，可以包含多个输出参数，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。 |

以“智能灌溉”*产品*为例，该*产品*的功能定义应该包括：

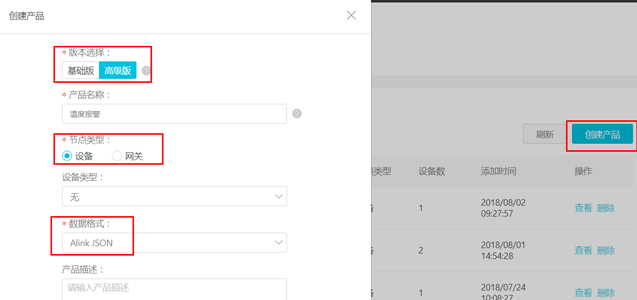
* 属性：电源开关。用户可以查询或者设置当前设备的开关状态。
* 服务：自动喷灌。在业务需要时，用户可以调用该服务实现灌溉，并通过在调用时给定本次喷灌所需的喷灌时间和灌溉量来实现精准灌溉。
* 事件：故障上报。当设备出现电压异常、网络故障时，上报平台。

通过在控制台上操作，我们可以直接新增属性、服务或事件，完成*产品*的功能定义。物联网平台会根据我们的定义自动生成*产品*的“物模型”文件，该文件以JSON格式描述了*产品*的所有功能，称为TSL（Thing Specification Language）。

**物联网平台侧开发**

**创建产品，添加设备**

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*“温度报警”，节点类型选择为“设备”，数据格式选择为“Alink JSON”, “设备类型”选择为“无”。



*产品*创建成功后，在该*产品*下添加设备test。



注意：在创建高级版*产品*时，系统会根据我们选择的“设备类型”自动创建该类型设备的标准功能模板。例如，我们选择创建“设备类型”为“电表”的高级版*产品*，系统就会为我们预定义用电量、电压、电流、总累积量等标准功能。

**明确功能**

在定义物模型前，我们首先需要明确当前“温度报警器”*产品*的功能。我们希望实现的功能包括：当温度传感器感知到高于阈值的温度时，板载LED灯以设定频率闪烁实现报警，同时开发板还需向物联网平台上报告警信息以通知用户；当危险状态解除后，用户还可以手动下发指令来清除LED灯的闪烁及告警信息的上报。因此我们需要为该*产品*定义：

* 属性：温度。只读类型，用户只能获取温度传感器感知的环境温度值，

不能够从云端下发温度值实现温度的设定。

LED灯闪烁频率。读写类型，当高温情况出现时，LED灯会以该

频率进行闪烁报警。用户可以获取当前设备中存储的该值，也可

以下发实现该值的设定。

* 服务：清除报警。当危险状态已经解除，而设备端仍处于高温报警状态

时，用户可以调用该服务清除报警状态。

* 事件：温度报警。当温度传感器感知到的温度高于设定阈值时，设备向

平台上报告警信息。

明确需要定义的属性、服务和事件后，我们前往物联网平台控制台进行定义。进入*产品*详情页面的“功能定义”栏，点击“查看物模型”，可以看到当前设备的JSON格式的物模型文件。



此时还未定义任何功能，物模型中的properties、services、events字段均为空。

|  |
| --- |
| {  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",  //物的TSL描述schema  "profile": {  "productKey": "a1x6pMlAkXS"  },  "services": [],  "properties": [],  "events": []  } |

**新增属性**

在功能列表的右上方点击“新增”，在弹出的窗口中选择功能类型为“属性”，即可新增一个属性类型的功能。此处我们首先新增“温度”属性，功能名称为“温度”，标识符为“Temperature”，数据类型选择为“double”，取值范围0~100，分辨率为0.1，单位为“摄氏度”，且读写类型为“只读”。



其中各字段的含义为：

* 功能名称：输入功能名称时物联网平台会自动从标准功能库中筛选匹配的标准属性，供用户直接选择；用户也可以直接创建自定义属性，功能名称一般为中文，如“温度”，“用电量”，且同一个*产品*下功能名称请勿重复。
* 标识符：物模型Alink JSON格式中的“identifier”，作为设备上报该属性数据的Key，如“Temperature”,“PowerComsuption”，云端会根据该标识符校验是否接收该数据。
* 数据类型：支持32位整型int32，单精度浮点型float，双精度浮点型double，枚举型enum，布尔型bool，字符串text，时间戳date，JSON对象struct和数组array。其中，int32/float/double型需要定义取值范围和单位符号；enum型需要定义枚举项的参数值和参数描述，如1代表加热模式，2代表制冷模式；bool型采用0和1来定义布尔值，如0代表关，1代表开；text型需要定义字符串的数据长度，最长支持1024个字节；date型的格式为string类型的UTC时间戳，支持毫秒；struct用于定义JSON结构体，可新增JSON参数项，如定义灯的颜色是由红、绿、蓝三个参数组成的结构体，不支持结构体嵌套；array需要声明数据内元素的数据类型，支持int32，float，double，text，同一个数据需要确保元素类型相同，数组长度为不定长，最长不超过128个元素。
* 读写类型：支持选择“读写”和“只读”两种类型，“读写”支持get（获取）和set（设置）方法，“只读”仅支持get（获取）的方法。
* 描述：用于对该属性功能进行说明或备注。

“温度”属性定义完毕后，我们可以在控制台上查看最新的物模型文件。我们将看到，当前properties属性字段中新增了标识符为“Temperature”的“温度”属性。

|  |
| --- |
| "properties": [  {  "identifier": "Temperature",  "dataType": {  "specs": {  "unit": "°C",  "min": "0",  "max": "100",  "step": "0.1"  },  "type": "double"  },  "name": "温度",  "accessMode": "r", //r为只读，rw为读写  "required": false  }  ], |

有趣的是，此时我们还并未新增服务和事件功能，但当前物模型文件中的services字段和events字段却已经不再为空。

|  |
| --- |
| "services": [  {  "outputData": [],  "identifier": "set",  "inputData": [],  "method": "thing.service.property.set",  "name": "set",  "required": true,  "callType": "async",  "desc": "属性设置"  },  {  "outputData": [  {  "identifier": "Temperature",  "dataType": {  "specs": {  "unit": "°C",  "min": "0",  "max": "100",  "step": "0.1"  },  "type": "double"  },  "name": "温度"  }  ],  "identifier": "get",  "inputData": [  "Temperature"  ],  "method": "thing.service.property.get",  "name": "get",  "required": true,  "callType": "async",  "desc": "属性获取"  }  ], |

可见，services字段中新增的内容包括两部分，第一个部分为thing.service.property.set方法，即“属性设置”方法；第二个部分为thing.service.property.get方法，即“属性获取”方法。由于当前“温度”属性为“只读”类型，用户无法从平台侧设置该属性，而只能获取该属性的值，因此thing.service.property.set方法中无实质性内容，而thing.service.property.get方法中包含了“温度”属性的相关信息。

|  |
| --- |
| "events": [  {  "outputData": [  {  "identifier": "Temperature",  "dataType": {  "specs": {  "unit": "°C",  "min": "0",  "max": "100",  "step": "0.1"  },  "type": "double"  },  "name": "温度"  }  ],  "identifier": "post",  "method": "thing.event.property.post",  "name": "post",  "type": "info",  "required": true,  "desc": "属性上报"  }  ] |

events字段新增的内容中包含thing.event.property.post方法，即“属性上报”方法。这是由于设备可以向平台上报“温度”属性的当前值，可以看到当前字段中也包含了“温度”属性的相关信息。

通过上面这些解释，相信你已经明白了仅仅新增属性但TSL文件中services和events字段都不再为空的原因。

同理，新增“LED灯闪烁频率”属性。



**新增服务**

在功能列表右上方点击“新增”，在弹出的窗口中选择功能类型为“服务”，即可新增一个服务类型的功能，此处我们新增“清除报警”服务。



其中各字段的含义为：

* 功能名称：该服务的名称，一般为中文，支持从标准功能库中搜索标准服务，如搜索“自动灌溉”服务，也可以自定义功能名称“清除报警”。
* 标识符：物模型Alink JSON格式中的“identifier”，作为该服务被调用时的Key，如“AutoSprinkle”“ClearAlarm”。
* 调用方式：支持选择“异步”和“同步”两种方式，异步调用是指云端执行调用后直接返回，不会关心设备的回复消息；如果服务为同步调用，云端则会等待设备回复，否则会调用超时。
* 输入参数（可选）：该服务的入参，点击新增参数，在弹窗中添加一个服务入参。用户可以直接选择某个属性作为入参，也可以自定义参数，如将“喷灌时间”和“喷灌量”作为“自动灌溉”服务的入参，调用该服务时传入这两个参数，喷灌设备将按照设定的喷灌时间和喷灌量自动进行精准灌溉。注意，一个服务最多支持定义10个入参。
* 输出参数（可选）：该服务的出参，点击新增参数，在弹窗中添加一个服务出参。用户可以直接选择某个属性作为出参，也可以自定义参数，如将“土壤湿度”作为出参，则云端调用“自动灌溉”服务时将返回当前土壤湿度的数据。注意，一个服务最多支持定义10个出参。
* 描述：用于对该属性功能进行说明或备注。

“清除报警”服务定义完毕后，我们可以查看最新的物模型文件。从物模型文件中可以看到，当前services字段增加了刚刚定义的该服务的内容。

|  |
| --- |
| {  "outputData": [],  "identifier": "ClearAlarm",  "inputData": [],  "method": "thing.service.ClearAlarm",  "name": "清除报警",  "required": false,  "callType": "async"  } |

**新增事件**

在功能列表右上方点击“新增”，在弹出的窗口中选择功能类型为“事件”，即可新增一个事件类型的功能，此处我们新增“温度报警”事件。

****

其中各字段含义为：

* 功能名称：该事件的名称，一般为中文，支持从标准功能库中搜索标准事件，如“故障上报”事件。
* 标识符：物模型Alink JSON格式中的“identifier”，作为设备上报该事件时的Key，如“ErrorCode”。
* 事件类型：事件分为信息、告警和故障三种类型，“信息”是设备上报的一般性通知，如完成某项任务等，“告警”和“故障”是设备运行过程中主动上报的突发或异常情况，优先级高，用户可以针对不同的事件类型进行业务逻辑处理和统计分析。
* 输出参数（可选）：该服务的出参，点击新增参数，在弹窗中添加一个事件出参。用户可以直接某个属性作为出参，也可以自定义参数，如将“电压”作为出参，则设备上报该故障事件时将携带当前设备的电压值，用于进一步判断故障原因。注意，一个事件最多支持定义10个出参。
* 描述：用于对该属性功能进行说明或备注。

“温度报警”事件定义完毕后，我们可以查看最新的物模型文件。从物模型文件中可以看到，当前events字段中新增了刚刚定义的事件内容。

|  |
| --- |
| {  "outputData": [  {  "identifier": "AlarmTemperature",  "dataType": {  "specs": {  "unit": "°C",  "min": "0",  "max": "100",  "step": "0.1"  },  "type": "double"  },  "name": "告警温度"  }  ],  "identifier": "Alarm",  "method": "thing.event.Alarm.post",  "name": "温度报警",  "type": "alert",  "required": false  } |

物模型定义完毕后，在功能列表中，我们可以看到当前*产品*的全部功能，还可以对这些功能进行编辑和删除。



**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。开发板有板载的温湿度传感器，可以通过该温湿度传感器获取环境温湿度信息，并将采集到的信息上报到平台。

**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tls\_aiotkit\_sample例程，开发板选择为AIoTKIT开发板。

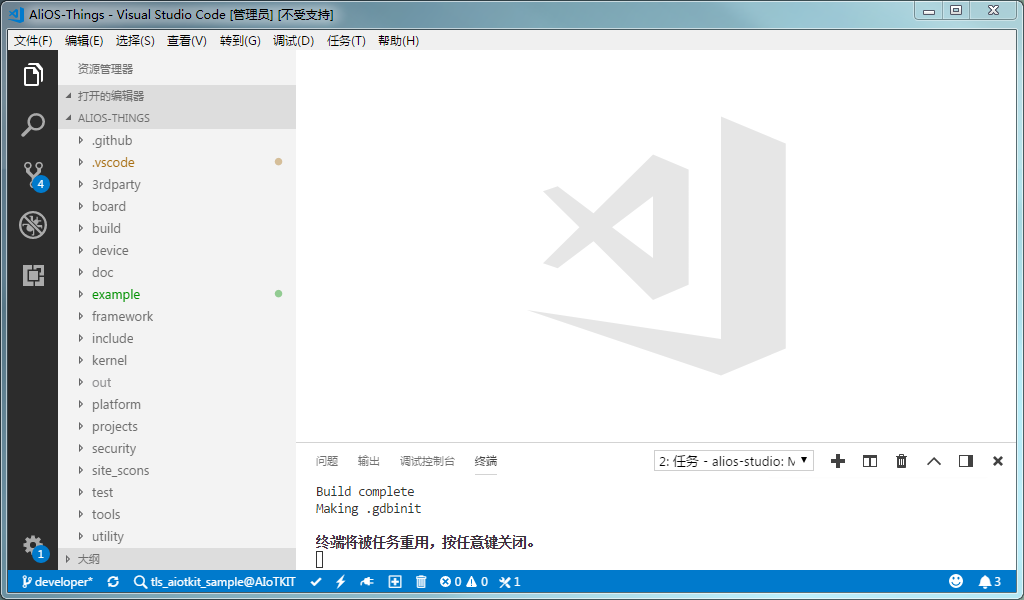


Figure 选择开发板与例程

**主要代码讲解**

本节中，tls\_aiotkit\_sample例程代码中tls\_aiotkit\_sample.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tls\_aiotkit\_sample.c文件，即AIoTKIT开发板和外接mk3080模组进行联网并上传温湿度数据的例程。

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[])  {  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  #if 0  memset(&apconfig, 0, sizeof(apconfig));  strcpy(apconfig.ssid, "LinkDevelop-Workshop");  strcpy(apconfig.pwd, "linkdevelop");  netmgr\_set\_ap\_config(&apconfig);  #endif  netmgr\_start(false);  aos\_cli\_register\_command(&mqttcmd);  //省略部分代码  //每100ms开启定时任务app\_delayed\_action  aos\_post\_delayed\_action(100, app\_delayed\_action, NULL);  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)  { /\* Device AUTH \*/  if (0 != IOT\_SetupConnInfo(PRODUCT\_KEY, DEVICE\_NAME, DEVICE\_SECRET, (void \*\*)&pconn\_info)) {  LOG("AUTH request failed!");  rc = -1;  release\_buff();  return rc; }  /\* Initialize MQTT parameter \*/  memset(&mqtt\_params, 0x0, sizeof(mqtt\_params));  mqtt\_params.port = pconn\_info->port;  mqtt\_params.host = pconn\_info->host\_name;  mqtt\_params.client\_id = pconn\_info->client\_id;  mqtt\_params.username = pconn\_info->username;  mqtt\_params.password = pconn\_info->password;  mqtt\_params.pub\_key = pconn\_info->pub\_key;  mqtt\_params.request\_timeout\_ms = 2000;  mqtt\_params.clean\_session = 0;  mqtt\_params.keepalive\_interval\_ms = 60000;  mqtt\_params.pread\_buf = msg\_readbuf;  mqtt\_params.read\_buf\_size = MSG\_LEN\_MAX;  mqtt\_params.pwrite\_buf = msg\_buf;  mqtt\_params.write\_buf\_size = MSG\_LEN\_MAX;  mqtt\_params.handle\_event.h\_fp = event\_handle\_mqtt;  mqtt\_params.handle\_event.pcontext = NULL;  /\* Construct a MQTT client with specify parameter \*/  gpclient = IOT\_MQTT\_Construct(&mqtt\_params);  if (NULL == gpclient) {  LOG("MQTT construct failed");  rc = -1;  release\_buff();  //aos\_unregister\_event\_filter(EV\_SYS, mqtt\_service\_event, gpclient);  } else {  aos\_register\_event\_filter(EV\_SYS, mqtt\_service\_event, gpclient);  }  return rc;  } |

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\_publish()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!");  mqtt\_publish(priv\_data);  /\*调用本次例程的主要函数mqtt\_publish();\*/  } |

图 18 wifi事件处理函数

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

void mqtt\_publish(void \*pclient)函数是本次tls\_aiotkit\_sample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，并接收云端下发的指令。
* 向指定TOPIC循环发送温度数据。在本次例程中，设计的为每3S采集一次环境温度信息，每60S上报一个温度属性信息。在采集环境温度信息的过程中，如果检测到温度信息超过阈值，则不断向平台发送报警信息，直到平台下发清除报警信息指令或者温度低于阈值为止。
* 在每次设备上线时，向平台发送LED闪烁频率属性信息，之后如果产生温度报警情况，那么开发板上的LED灯则会随着目前设定的LED闪烁频率进行闪烁。并且设备也可以接收平台下发的频率属性设置指令来动态的修改LED闪烁频率。

|  |
| --- |
| static void mqtt\_publish(void \*pclient){  if (is\_subscribed == 0) {  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);  }  /\*订阅属性设置TOPIC。平台可以通过该TOPIC下发设置LED闪烁频率。并定义其TOPIC的回调函数为handle\_prop\_set，回调函数具体内容请参考下文\*/  rc = IOT\_MQTT\_Subscribe(pclient, ALINK\_TOPIC\_PROP\_SET,  IOTX\_MQTT\_QOS0, handle\_prop\_set, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  }  /\*订阅服务service/ClearAlarm TOPIC。平台可以通过该TOPIC下发清除设备报警信息的指令。并定义其TOPIC的回调函数为handle\_prop\_ser ，回调函数具体内容请参考下文\*/  rc = IOT\_MQTT\_Subscribe(pclient, ALINK\_TOPIC\_SER\_SUB,  IOTX\_MQTT\_QOS0, handle\_prop\_ser, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  }  is\_subscribed = 1;  } else {  /\* Initialize topic information \*/  memset(&topic\_msg, 0x0, sizeof(iotx\_mqtt\_topic\_info\_t));  topic\_msg.qos = IOTX\_MQTT\_QOS0;  topic\_msg.retain = 0;  topic\_msg.dup = 0;  memset(param, 0, sizeof(param));  memset(msg\_pub, 0, sizeof(msg\_pub));  get\_temp\_data(&temp\_data, &temp\_timestamp); //每隔3S获取温度数据信息  if(t\_count>=20) //每 20\*3=60S上报一次温度数据  {  t\_count=0;  float temp = (float)temp\_data/100.0; //传感器数据到实际数据格式转换  sprintf(param, "{\"Temperature\":%.1f}",temp);  /\*按照物模型的属性上报格式要求组合温度信息\*/  int msg\_len = sprintf(msg\_pub, ALINK\_BODY\_FORMAT, cnt, ALINK\_METHOD\_PROP\_POST, param);  if (msg\_len < 0) LOG("Error occur! Exit program");  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  /\*定时上报环境温度信息到/event/property/post 属性上报TOPIC\*/  rc = IOT\_MQTT\_Publish(pclient, ALINK\_TOPIC\_PROP\_POST, &topic\_msg);  if (rc < 0) LOG("error occur when publish");  LOG("Alink:\n%s\n",msg\_pub);  cnt++;  }  t\_count++;  /\*根据获取的环境温度信息，判断环境温度是否超过阈值，超过阈值需要上报报警信息\*/  alarm\_judge(temp\_data,pclient);  /\*上报LED的闪烁频率信息，仅仅在设备初次上线或者平台端修改该属性时才会上报该属性信息，平时该属性信息无需上报。\*/  if(led\_fre\_upload\_status==1) //上报LED闪烁频率  {  led\_fre\_upload\_status = 0;  int fre = get\_led\_fre();  printf("get\_led\_fre :%d\n",fre);  sprintf(param, "{\"Frequency\":%d}",fre);  int msg\_len = sprintf(msg\_pub, ALINK\_BODY\_FORMAT, cnt, ALINK\_METHOD\_PROP\_POST, param);  if (msg\_len < 0) LOG("Error occur! Exit program");  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, ALINK\_TOPIC\_PROP\_POST, &topic\_msg);  if (rc < 0) LOG("error occur when publish");  LOG("Alink:\n%s\n",msg\_pub);  cnt++;  }  LOG("system is running %d\n",cnt);  }  if (++cnt < 20000) {  /\*每3s重新运行mqtt\_publish()函数\*/  aos\_post\_delayed\_action(3000, mqtt\_publish, pclient);  } else {  IOT\_MQTT\_Unsubscribe(pclient, ALINK\_TOPIC\_PROP\_POSTRSP);  aos\_msleep(200);  IOT\_MQTT\_Destroy(&pclient);  release\_buff();  is\_subscribed = 0;  cnt = 0;  }  } |

图 19 ldapp例程数据上云主函数

1. static void handle\_prop\_set(void \*pcontext, void \*pclient, iotx\_mqtt\_event\_msg\_pt msg)

该函数为tls\_aiotkit\_sample例程中云端设置属性参数的TOPIC回调函数，云端可以通过/property/set TOPIC实现LED闪烁频率属性的设置。

|  |
| --- |
| /\*  \* MQTT Subscribe handler  \* topic: ALINK\_TOPIC\_PROP\_SET  \*/  static void handle\_prop\_set(void \*pcontext, void \*pclient, iotx\_mqtt\_event\_msg\_pt msg)  {  iotx\_mqtt\_topic\_info\_pt ptopic\_info = (iotx\_mqtt\_topic\_info\_pt)msg->msg;  if(NULL != strstr(ptopic\_info->payload,"\"Frequency\":"))  {  int fre=0;  /\*解析云端下发的指令\*/  char \* result = strstr(ptopic\_info->payload,"\"Frequency\":");  result += strlen("\"Frequency\":");  if(\*(result+1)=='}')  {  fre = \*result-'0';  }  else if(\*(result+2)=='}')  {  fre = (\*result-'0')\*10+(\*(result+1)-'0');  }  printf("Frequency %d\n",fre);  set\_led\_fre(fre); //设置LED闪烁频率  led\_fre\_upload\_status=1; //置位LED闪烁频率上报标记  }  } |

图 18 /property/set TOPIC回调函数

1. static void handle\_prop\_ser(void\*pcontext,void \*pclient, iotx\_mqtt\_event\_msg\_ pt msg)

该函数为tls\_aiotkit\_sample例程中云端服务设置的TOPIC回调函数，云端可以通过/service/ClearAlarm TOPIC实现清除设备报警信息的设置。

|  |
| --- |
| /\*  \* MQTT Subscribe handler  \* topic: ALINK\_TOPIC\_PROP\_SER  \*/  static void handle\_prop\_ser(void \*pcontext, void \*pclient,  iotx\_mqtt\_event\_msg\_pt msg)  {  iotx\_mqtt\_topic\_info\_pt ptopic\_info = (iotx\_mqtt\_topic\_info\_pt)msg->msg;  #if 1  // print topic name and topic message  LOG("----");  LOG("Topic: '%.\*s' (Length: %d)", ptopic\_info->topic\_len,  ptopic\_info->ptopic, ptopic\_info->topic\_len);  LOG("Payload: '%.\*s' (Length: %d)", ptopic\_info->payload\_len,  ptopic\_info->payload, ptopic\_info->payload\_len);  LOG("----");  /\*解析云端下发的指令，并清除报警的标记位\*/  if(NULL != strstr(ptopic\_info->payload,"thing.service.ClearAlarm"))  {  alarm\_clear = 1;  }  #endif  } |

图 18 /service/ClearAlarm TOPIC回调函数

**修改设备相关参数**

在上一小节中，我们打开了tls\_aiotkit\_sample@AIoTKIT工程。在本小节中，我们需要修改官方tls\_aiotkit\_sample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/tls\_aiotkit\_sample,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/tls\_aiotkit\_sample/tls\_aiotkit\_sample.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

在本实验中，设备采用Alink JSON格式，基于Alink Topic与物联网平台进行消息通信，完成设备的属性上报、属性设置、事件上报和服务调用功能。以设备主动上报属性到云端为例，当设备端“温度”属性发生变化时，设备可以通过Topic: /sys/{productKey}/{deviceName}/thing/event/property/post通知云端。

|  |
| --- |
| {  "id" : "123", //标示该次请求id值  "version":"1.0", //协议版本号固定字段  "params" : {  "Temperature" : 31 //对应*产品*属性中的Temperature  },  "method":"thing.event.property.post" //属性上报topic  } |

Figure 设备上报属性格式

**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**

通过复位开发板使其运行烧录进去的程序，接入物联网平台。我们进入*产品*详情页面的“在线调试”栏，选择“调试设备”为我们创建的test设备，可以看到由于此时设备还未上线，控制台上显示“检测到设备未激活”字样，同时控制台上还可以看到 “实时日志”窗口和“调试功能”窗口。



Figure 物联网控制台在线调试界面

当设备与云端连接成功后，我们可以在控制台上查看到设备已经处于在线状态。



Figure 设备在线状态显示

**属性调试**

在实时日志窗口中，我们可以看到设备上线后主动上报给云端的当前“温度”属性值和“LED灯闪烁频率”（对应开发板上的LED2）属性值。这些消息的MessageMethod均为thing.event.property.post，设备上报的LED灯闪烁频率为2，温度为30.7℃。

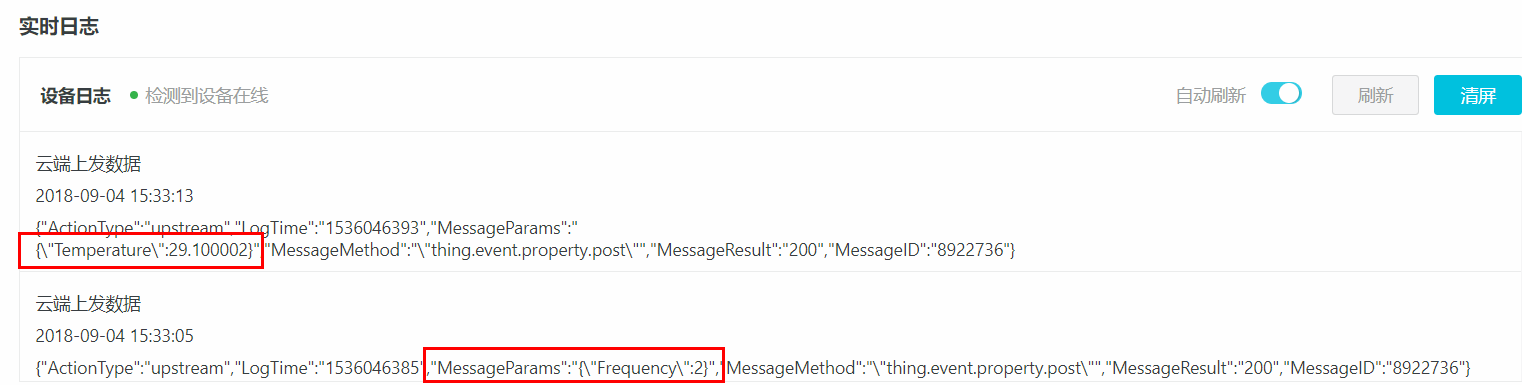


Figure 设备端上报属性信息

我们还可以在串口助手软件中查看设备终端打印的对应日志。



Figure 设备端打印的日志信息

我们还可以主动获取设备端的属性，调试功能选择为“温度”，方法选择为“获取”，点击“发送指令”，即可主动获取“温度”属性值。同理，也可以获取LED灯闪烁频率属性值。



Figure 主动获取温度属性值



Figure 主动获取LED闪烁频率属性值

我们还可以修改设备的属性值，注意，只有属性的“读写类型”为“读写”时，云端才可以修改该属性值；对于“只读”型属性，云端只能执行“获取”操作。当我们下发指令，将“LED灯闪烁频率”属性值设置为1，设备端成功接收“设置”属性指令后，将修改该属性值，并向云端主动上报该属性的最新状态。

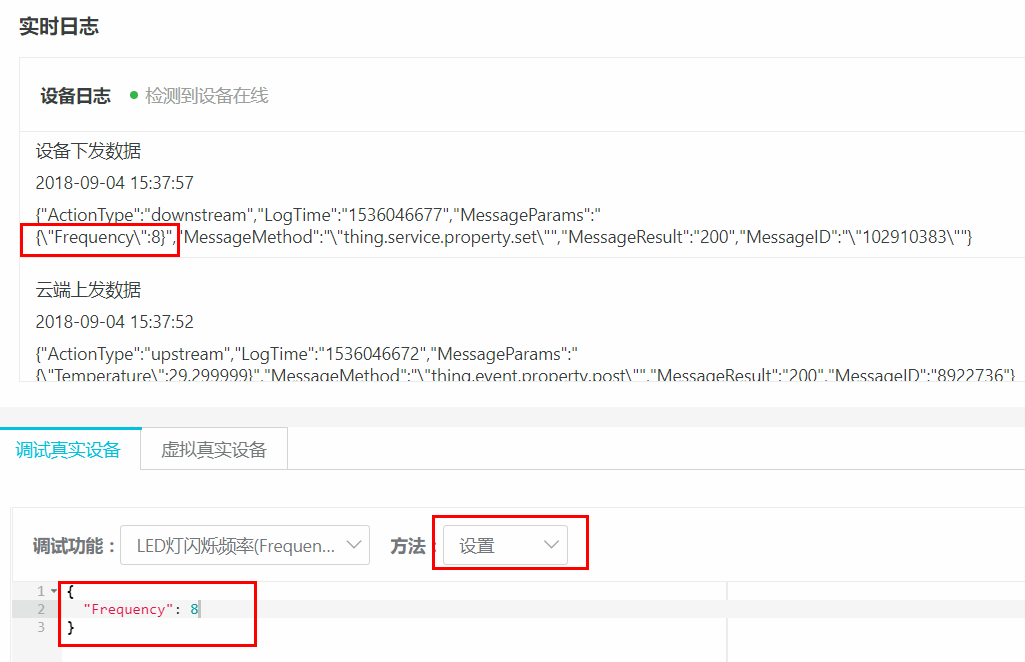


Figure 云端设置LED闪烁频率属性



Figure 云端是指属性时的设备日志信息

**事件调试**

当设备端特定事件被触发时，设备端会向云端上报事件。在本实验中，当温度传感器感知到的温度高于阈值时，开发板在闪烁LED灯的同时会向物联网平台上报“告警信息”，MessageMethod为thing.event.Alarm.post。

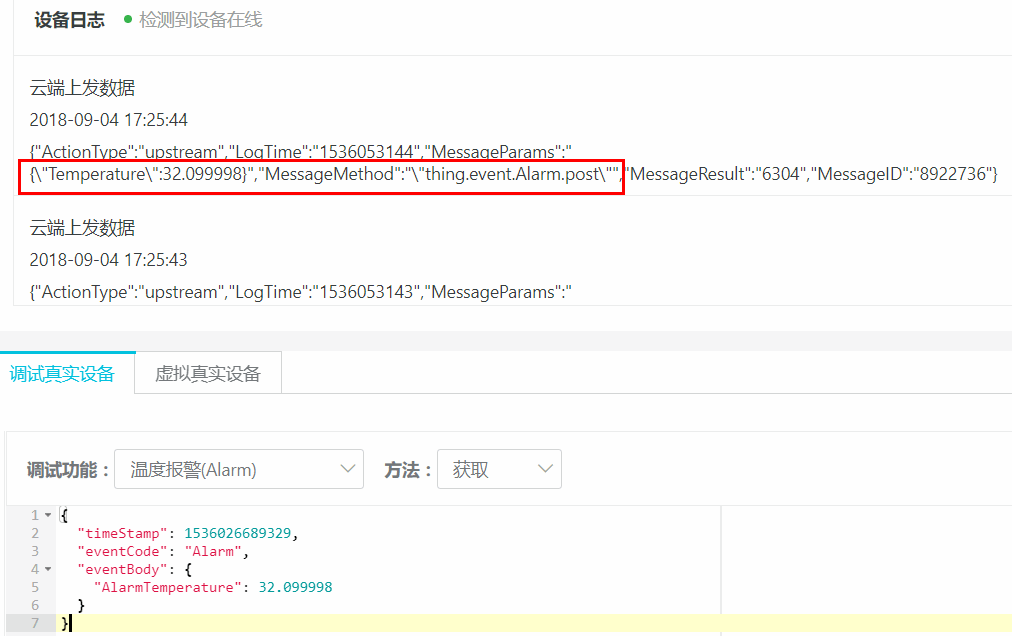


Figure 温度报警

**服务调试**

当高温的危险状态解除后，用户可以手动下发“清除报警”指令取消LED灯的闪烁。



Figure 云端发送清除报警指令时的设备端日志

指令发送后，设备端LED灯将停止闪烁，且设备停止向云端上报告警事件。

至此，我们基于AIoTKIT开发板完成了“温度报警器”*产品*的完整功能定义和在线调试功能！

**课程小结**

本次课程学习了物联网平台设备管理组件中的物模型和在线调试两大功能。在掌握物模型相关的属性、服务、事件概念的基础上，通过对实际设备进行功能定义和在线调试，进一步加深读者对物模型的理解，并让读者熟悉在线调试的相关操作。