目录

[课程内容 2](#_Toc522787631)

[课程目标 2](#_Toc522787632)

[课程准备 3](#_Toc522787633)

[课程知识点 3](#_Toc522787634)

[设备影子介绍 3](#_Toc522787635)

[设备影子数据流 5](#_Toc522787636)

[物联网平台侧开发 15](#_Toc522787637)

[新建产品和*设备* 15](#_Toc522787638)

[设备端开发 16](#_Toc522787639)

[新建项目 16](#_Toc522787640)

[代码开发 18](#_Toc522787641)

[编译下载 19](#_Toc522787642)

[运行调试 19](#_Toc522787643)

[课程小结 22](#_Toc522787644)

**基于设备影子的LED灯闪烁频率设置**

前几次课程中涉及到的云端指令下发均是在*设备*在线情况下进行的，当*设备*掉线时，指令将无法成功到达*设备*端，也无法被*设备*执行。在本次课程中，我们将为读者介绍物联网平台提供的另一个*设备*管理能力——*设备影子*。借助*设备影子*的功能，即使*设备*发生掉线重连的情况，也可以成功接收到云端在它掉线时发送的控制指令，执行相应的业务操作。



**课程内容**

* 介绍*设备影子*概念
* 介绍*设备影子*的JSON格式
* 介绍*设备影子*相关的数据流
* 基于AIoTKIT开发板实现*设备*掉线重连时的指令下发

**课程目标**

* 熟悉*设备影子*的功能和JSON文档格式
* 掌握*设备*、*影子*以及应用程序之间的通信数据流
* 基于*设备影子*功能实现云端对LED灯闪烁频率的控制

**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**

**设备影子介绍**

接入LP的每个*设备*在云端都有唯一的*设备影子*与之对应，*设备影子*以 JSON 文档的形式存储着当前*设备*上报的状态信息以及应用程序期望的状态信息。*设备*可以基于MQTT协议获取和设置*设备影子*，以实现状态的同步。当*设备*获取*影子*时，状态从*影子*同步到*设备*；当*设备*设置*设备影子*时，状态从*设备*同步到*影子*。应用程序也可以基于MQTT协议和阿里云提供的API来通过*设备影子*获取和设置*设备*的状态，无论该*设备*是否连接到网络。

以网络不稳定，*设备*频繁上下线的场景为例，如果应用程序向*设备*下发控制指令时*设备*刚好掉线，那么指令将无法下达到*设备*。我们可以通过设置QoS=1或QoS=2来解决该问题，但是这种方式会给*服务*端带来较大的压力，一般不建议使用。

如果采用*设备影子*功能，这个问题就可以轻松地得到解决了。应用程序在下发指令时不需要关心*设备*是否在线，指令及其时间戳会被保存在*设备影子*中，当*设备*掉线重连时，从*设备影子*中获取指令即可，还可以根据指令的时间戳信息判断指令是否过期，是否需要执行。

作为一个JSON文档，*设备影子*包含desired、reported、version等*属性*。

|  |  |
| --- | --- |
| *属性* | 描述 |
| desired | *设备*的预期状态。应用程序可以向desired部分写入数据来更新事物的状态，且无需*设备*在线。 |
| reported | *设备*的报告状态。*设备*可以向reported部分写入数据，报告其最新状态。应用程序可以读取该部分，获取*设备*的状态。 |
| metadata | 当用户更新desired或reported部分内容时，*设备影子服务*会自动更新该metadata部分。这部分包括每个*属性*的时间戳（以Epoch时间表示），让用户能够确定它们的更新时间。 |
| timestamp | *影子*文档的最新更新时间。 |
| version | 用户主动更新version，那*设备影子*会检查请求中的version是否大于当前版本，如果大于的话，则更新*设备影子*，并将version更新到请求的版本，反之拒绝更新*设备影子*。文档每次更新时，此版本号都会递增，用于确保正在更新的文档为最新版本。 |

*设备影子*的完整JSON格式为：

|  |
| --- |
| {  "state": {  "desired": {  "attribute1": integer2,  ...  "attributeN": boolean2  },  "reported": {  "attribute1": integer1,  ...  "attributeN": boolean1  }  },  "metadata": {  "desired": {  "attribute1": {  "timestamp": timestamp  },  ...  "attributeN": {  "timestamp": timestamp  }  },  "reported": {  "attribute1": {  "timestamp": timestamp  },  ...  "attributeN": {  "timestamp": timestamp  }  }  },  "timestamp": timestamp,  "version": version  } |

其中的desired部分和reported部分都可以为空。只有当*影子*具有预期状态时，才包含desired部分；具有*设备*报告的状态时，才包含reported部分。

**设备影子数据流**

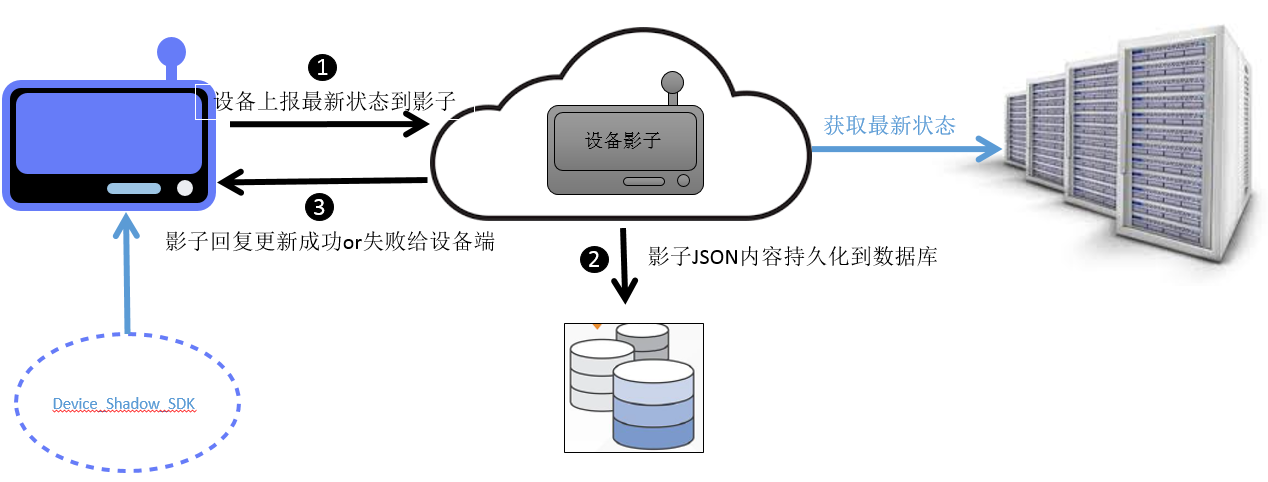
阿里云物联网平台为每个*设备*预先定义好了两个*Topic*，专门用于*设备影子*的数据流转。

|  |  |
| --- | --- |
| /shadow/update/${*productKey*}/${*deviceName*} | *设备*和应用程序发布消息到该*Topic*，物联网平台收到后，将消息中的状态更新到*设备影子*中 |
| /shadow/get/${*productKey*}  /${*deviceName*} | *设备影子*更新状态到该*Topic*，*设备*订阅此*Topic*的消息 |

*Topic*中的${*productKey*}和${*deviceName*}需要分别替换为用户所创建*设备*的真实参数。

下面我们以RGB灯的“颜色”*属性*为例，具体介绍*设备*、*设备影子*以及应用程序之间通信的数据流。

* *设备*主动上报状态



*设备*成功接入物联网平台后，会主动上报状态，即向*Topic*:/shadow/update/${*productKey*}/${*deviceName*}中发送消息。发送的JSON消息格式为：

|  |
| --- |
| {  "method": "update",  "state": {  "reported": {  "color": "red"  }  },  "version": 1  } |

各字段含义为

|  |  |
| --- | --- |
| method | 表示*设备*或者应用程序请求*设备影子*的操作类型。当*设备*或者应用程序更新状态时，该字段为“update”。 |
| state | 表示*设备*发送给*影子*的状态信息。在上报状态时，reported字段为必须，这些信息会被同步到*设备影子*文档中的reported部分。 |
| version | 表示版本信息。*设备影子*会检查请求中的版本是否大于当前*影子*版本。只有在大于时，*设备影子*才会接受*设备*端的请求，更新*设备影子*到相应的版本。 |

当*设备影子*接收到上报的状态信息时，更新*影子*文档。

|  |
| --- |
| {  "state" : {  "reported" : {  "color" : "red"  }  },  "metadata" : {  "reported" : {  "color" : {  "timestamp" : 1469564492  }  }  },  "timestamp" : 1469564492,  "version" : 1  } |

更新完成后，*设备影子*会返回结果给*设备*，即发消息到*Topic*：/shadow/get/${*productKey*}/${*deviceName*}，*设备*订阅该*Topic*即可收到平台返回的更新结果。若更新成功，*设备*收到如下返回结果：

|  |
| --- |
| {  "method":"reply",  "payload": {  "status":"success",  "version": 1  },  "timestamp": 1469564576  } |

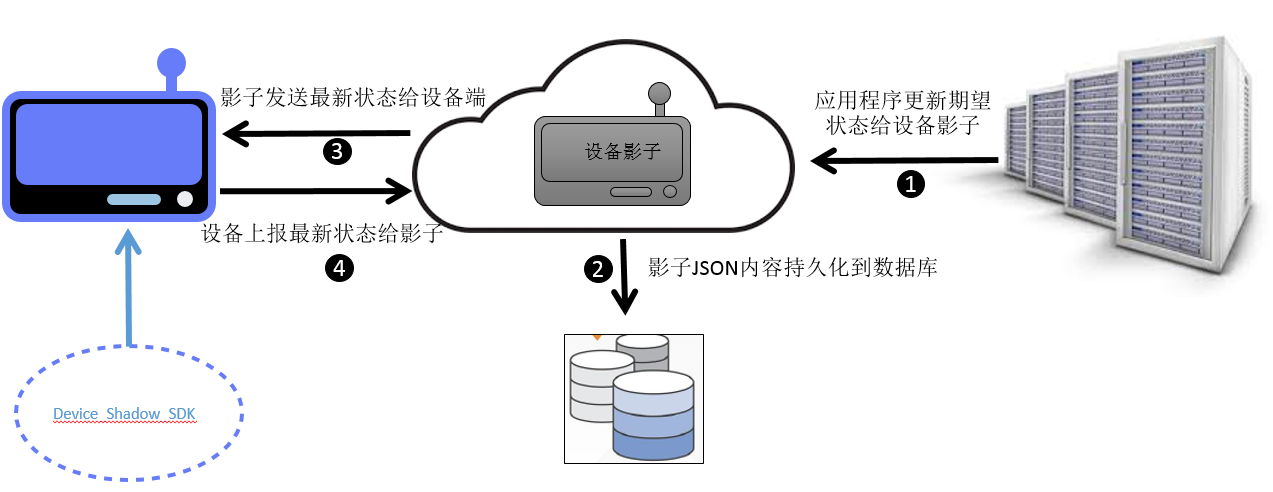
若更新失败，*设备*收到如下返回结果：

|  |
| --- |
| {  "method":"reply",  "payload": {  "status":"error",  "content": {  "errorcode": "${errorcode}",  "errormessage": "${errormessage}"  }  },  "timestamp": 1469564576  } |

errorCode的具体值可以帮助我们定位到不同的失败原因：

|  |  |
| --- | --- |
| errorCode | errorMessage |
| 400 | 不正确的json格式 |
| 401 | *影子*json缺少method信息 |
| 402 | *影子*json缺少state字段 |
| 403 | *影子*json version不是数字 |
| 404 | *影子*json缺少reported字段 |
| 405 | *影子*json reported*属性*字段为空 |
| 406 | *影子*json method是无效的方法 |
| 407 | *影子*内容为空 |
| 408 | *影子*reported*属性*个数超过128个 |
| 409 | *影子*版本冲突 |
| 500 | *服务*端处理异常 |

* 应用程序改变*设备*状态



应用程序可以通过下发指令更改RGB灯的颜色。此时，应用程序应将消息下发到*Topic*：/shadow/update/${*productKey*}/${*deviceName*}中，消息内容为：

|  |
| --- |
| {  "method": "update",  "state": {  "desired": {  "color": "green"  }  },  "version": 2  } |

应用程序发出更新请求后，*设备影子*会更新为：

|  |
| --- |
| {  "state" : {  "reported" : {  "color" : "red"  },  "desired" : {  "color" : "green"  }  },  "metadata" : {  "reported" : {  "color" : {  "timestamp" : 1469564492  }  },  "desired" : {  "color" : {  "timestamp" : 1469564576  }  }  },  "timestamp" : 1469564576,  "version" : 2  } |

*设备影子*更新完成后，会发送消息到*Topic*:/shadow/get/${*productKey*}/${*deviceName*}，*设备*订阅该*Topic*即可收到返回结果。此处示例中返回给*设备*的消息内容为：

|  |
| --- |
| {  "method":"control",  "payload": {  "status":"success",  "state": {  "reported": {  "color": "red"  },  "desired": {  "color": "green"  }  },  "metadata": {  "reported": {  "color": {  "timestamp": 1469564492  }  },  "desired" : {  "color" : {  "timestamp" : 1469564576  }  }  }  },  "version": 2,  "timestamp": 1469564576  } |

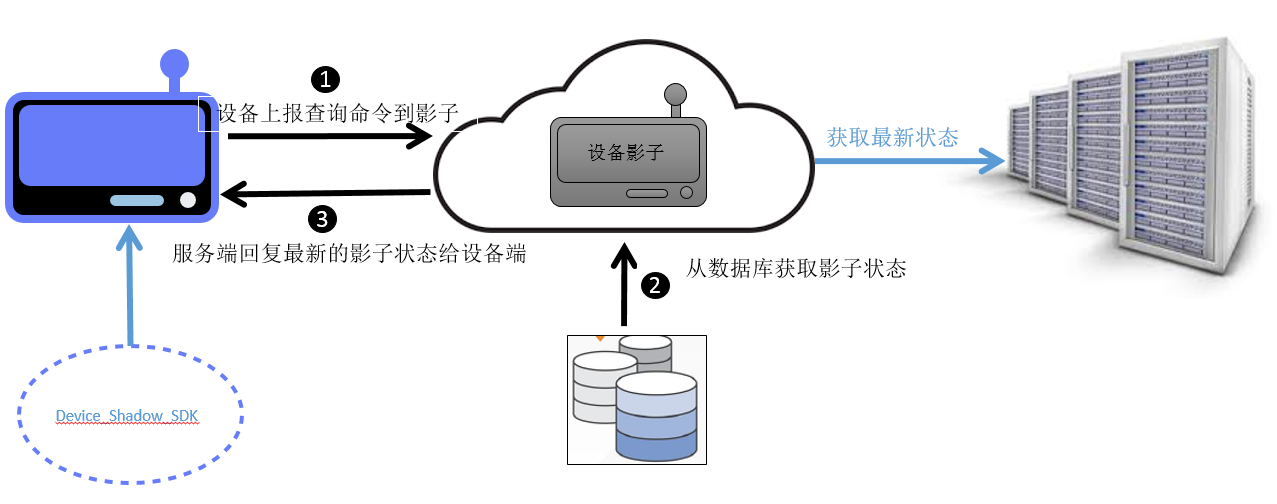
当*设备*在线或者是掉线重连时，都会收到该消息，并且根据消息内容中的“desired”字段更新状态，将RGB更新为绿色，并在更新后通过*Topic*：/shadow/update/${*productKey*}/${*deviceName*}上报最新状态。

|  |
| --- |
| {  "method": "update",  "state": {  "desired":"null"  },  "version": 3  } |

*设备影子*随着上报的状态进行更新：

|  |
| --- |
| {  "state" : {  "reported" : {  "color" : "green"  }  },  "metadata" : {  "reported" : {  "color" : {  "timestamp" : 1469564577  }  },  "desired" : {  "timestamp" : 1469564576  }  },  "version" : 3  } |

* *设备*主动获取*设备影子*



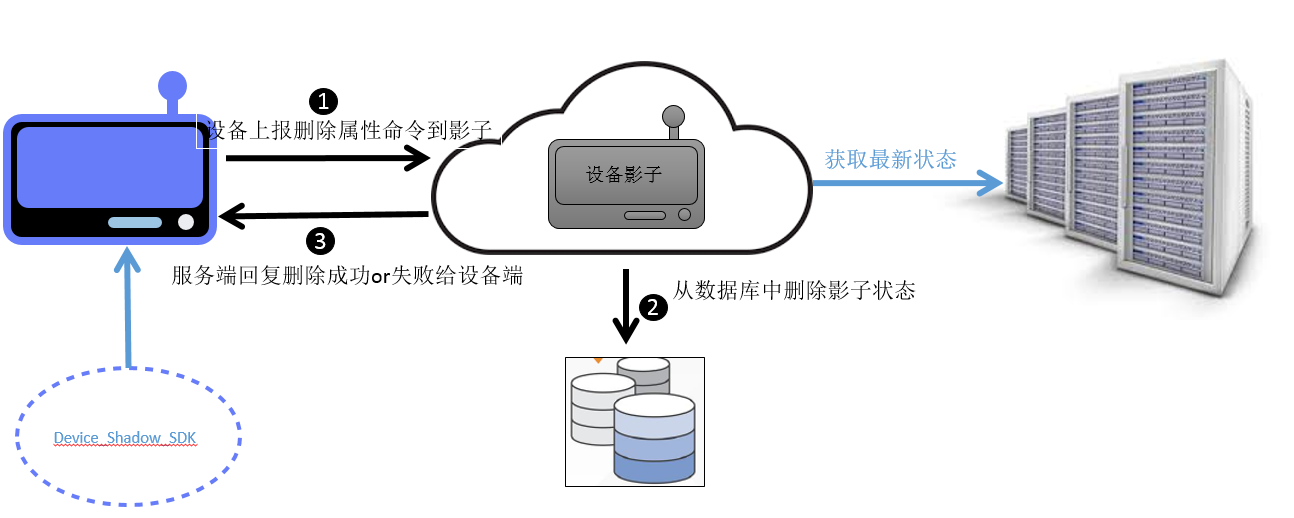
当*设备*想要获取*设备影子*中保存的最新状态时，会向*Topic*：/shadow/update/${*productKey*}/${*deviceName*}发送消息：

|  |
| --- |
| {  "method": "get"  } |

*设备影子*收到该消息后，将*影子*内容发送到*Topic*:/shadow/get/${*productKey*}/${*deviceName*}中，*设备*订阅该*Topic*即可收到*影子*内容。

|  |
| --- |
| {  "method":"reply",  "payload": {  "status":"success",  "state": {  "reported": {  "color": "red"  },  "desired": {  "color": "green"  }  },  "metadata": {  "reported": {  "color": {  "timestamp": 1469564492  }  },  "desired": {  "color": {  "timestamp": 1469564492  }  }  }  },  "version": 2,  "timestamp": 1469564576  } |

* *设备*删除*影子属性*



如果*设备*想删除*设备影子*中的某条*属性*状态，向*Topic*：/shadow/update/${*productKey*}/${*deviceName*}中发送消息，将method字段置为delete，待删除*属性*置为null即可。

|  |
| --- |
| 删除*影子*中的某一*属性*  {  "method": "delete",  "state": {  "reported": {  "color": "null",  "temperature":"null"  }  },  "version": 1  }  删除*影子*中的全部*属性*  {  "method": "delete",  "state": {  "reported":"null"  },  "version": 1  } |

**物联网平台侧开发**

**新建产品和设备**

在本次课程实验中，我们希望利用*设备影子*功能实现*设备*掉线重连情况下，云端指令的成功下发。以*设备*上的LED灯为例，实现云端对LED灯闪烁频率的控制。

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*“Shadow”，节点类型选择为“*设备*”，并在该*产品*下添加test*设备*。



此时，点击*设备*右侧的“查看”按钮，选择“*设备影子*”栏目，可看到当前的*设备影子*内容为空。



**设备端开发**

在本次课程—基于*设备影子*的LED灯闪烁频率实验中，我们使用AIoTKIT开发板通过mk3080 Wi-Fi模组接入网络。

**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shadowapp例程，开发板选择为AIoTKIT开发板。

**主要代码讲解**

在本次基于*设备影子*的LED闪烁频率设置的实验中，我们设定开发板在每次上线之后要使用get方法获取*设备影子*当中的LED闪烁频率最新状态，如果*设备*此时的状态滞后于*设备影子*的话，*设备*需要按照从*设备影子*中获得的最新状态进行更新，并且在更新完成后主动向云端上报，更新*设备影子*。

应用于*设备影子*的实验app和前面的实验内容除通信*TOPIC*和上下行消息格式外，大部分内容均是相同的。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4物模型 的实验中。

本节中，shadowapp例程代码中shadowapp.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shadowapp.c文件，即使用AIoTKIT开发板和外接mk3080模组进行联网并通信的例程。

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  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 。该函数实现LED闪烁的功能  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（）。

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\_publish()主函数。

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

void mqtt\_publish(void \*pclient)函数是本次shadowapp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*，接收并解析云端下发的指令。
* 每次上线获取*设备影子*的最新状态，并判断是否需要更新*设备*LED闪烁频率。如果需要更新的话，更新LED闪烁频率，并上报到云端。
* *设备*初次上线时，需要上报*设备*的LED闪烁频率*属性*信息。之后，除非*设备影子*的状态发生改变，*设备*接收到云端下发的消息。否则*设备*不再主动上报信息。

|  |
| --- |
| static void mqtt\_publish(void \*pclient){  if (is\_subscribed == 0) {  /\*订阅*设备影子TOPIC*。平台可以通过更新*设备影子*的状态来下发消息给*设备*，*设备*接收*设备影子*的消息的*TOPIC*是"/shadow/get/"PRODUCT\_KEY"/"DEVICE\_NAME""。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\_shadow\_get，回调函数具体内容请参考下文\*/  /\* Subscribe the specific *Topic* \*/  rc = IOT\_MQTT\_Subscribe(pclient, *TOPIC*\_SHADOW\_GET,  IOTX\_MQTT\_QOS0, handle\_shadow\_get, NULL);  if (rc < 0) {  // IOT\_MQTT\_Destroy(&pclient);  LOG("IOT\_MQTT\_Subscribe() failed, rc = %d", rc);  } is\_subscribed = 1;  } else { //省略部分代码  if(first\_get\_shadow == 1 ) //*设备*每次上线时，主动获取*设备影子*最新状态  {  first\_get\_shadow = 0;  int msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"method\":\"get\"}");  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, *TOPIC*\_SHADOW, &*Topic*\_msg);  if (rc < 0)  LOG("error occur when publish");  }  /\**设备*初次上线时，向云端发送LED闪烁频率*属性*，并更新*设备影子*\*/  if(first\_send\_shadow == 1 )  {  int ver = get\_shadow\_version();  int fre = get\_led\_fre();  if(fre==2) { pfre = 0; }  else if(fre ==10) { pfre = 1; }  if(ver>0)  {  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"method\": \"update\",\"state\": {\"reported\": {\"frequency\": \"fre%d\"}},\"version\": %d}",pfre,ver);  }  else  {  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"method\": \"update\",\"state\": {\"reported\": {\"frequency\": \"fre0\"}},\"version\": %d}",1);  }  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_SHADOW, &topic\_msg);  }  /\**设备影子*状态发生变化时，或者接收到*设备影子*下发的消息时，主动向云端回复，清除desired，并上报更新reported\*/  if(shadow\_reply == 1 )  {  int ver=get\_shadow\_version();  int fre = get\_led\_fre();  if(fre==2) { pfre = 0; }  else if(fre ==10) { pfre = 1; }  memset(msg\_pub, 0, sizeof(msg\_pub));  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"method\": \"update\",\"state\":{\"desired\": \"null\"},\"version\": %d}",ver);  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_SHADOW, &topic\_msg);    ver++;set\_shadow\_version(ver);  memset(msg\_pub, 0, sizeof(msg\_pub));  msg\_len = snprintf(msg\_pub, sizeof(msg\_pub), "{\"method\": \"update\",\"state\": {\"reported\": {\"frequency\": \"fre%d\"}},\"version\": %d}",pfre,ver);  topic\_msg.payload = (void \*)msg\_pub;  topic\_msg.payload\_len = msg\_len;  rc = IOT\_MQTT\_Publish(pclient, TOPIC\_SHADOW, &topic\_msg);  }  }  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 shadowapp例程数据上云主函数

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

该函数为shadowapp例程中*设备影子*设置*属性*参数的*TOPIC*回调函数，云端可以通过"/shadow/get/"PRODUCT\_KEY"/"DEVICE\_NAME"" *TOPIC*实现LED闪烁频率*属性*的设置。

|  |
| --- |
| /\*  \* MQTT Subscribe handler  \* *Topic*: RAW\_*TOPIC*\_PROP\_DOWN  \*/  static void handle\_shadow\_get(void \*pcontext, void \*pclient, iotx\_mqtt\_event\_msg\_pt msg)  {  iotx\_mqtt\_*Topic*\_info\_pt p*Topic*\_info = (iotx\_mqtt\_*Topic*\_info\_pt)msg->msg;  /\*初次上线时，*设备影子*为空，需要*设备*主动上报消息到云端，更新*设备影子*状态\*/  if(NULL != strstr(p*Topic*\_info->payload,"shadow content is empty"))  {  first\_send\_shadow = 1;  led\_fre = 2;  set\_shadow\_version(1);  }  /\**设备影子*不为空，每次*设备*上线均需要检查*设备影子*，并根据*设备影子*的最新状态更新本地*设备*信息\*/  if(NULL != strstr(p*Topic*\_info->payload,"\"desired\":{\"frequency\":\"fre0"))  {  led\_fre = 2;  shadow\_reply = 1;  update\_shadow\_version(p*Topic*\_info->payload);  }  if(NULL != strstr(p*Topic*\_info->payload,"\"desired\":{\"frequency\":\"fre1"))  {  led\_fre = 10;  shadow\_reply = 1;  update\_shadow\_version(p*Topic*\_info->payload);  }  /\*设备影子不为空时，需要同步版本信息，并更新设备状态\*/  else if(NULL != strstr(ptopic\_info->payload,"\"reported\":{\"frequency\":\"fre0"))  {  led\_fre = 2;  update\_shadow\_version(ptopic\_info->payload);  }  else if(NULL != strstr(ptopic\_info->payload,"\"reported\":{\"frequency\":\"fre1"))  {  led\_fre = 10;  update\_shadow\_version(ptopic\_info->payload);  }  } |

图 18"/shadow/get/"PRODUCT\_KEY"/"DEVICE\_NAME"" *TOPIC*回调函数

**修改设备相关参数**

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

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。成功接入后*设备*会主动上报当前LED灯的闪烁频率状态到*设备影子*，可以在控制台上刷新查看。当前LED灯（对应AIoTKIT开发板上的LED1）的闪烁频率为reported字段内的值fre0，处于慢闪状态。



Figure 2 云端查看*设备影子*具体内容

为了让大家充分体会到*设备影子*缓存机制的作用，此时让*设备*断电离线。然后点击右侧的“更新*影子*”按钮，将LED灯闪烁频率设置为fre1，即快闪状态。



Figure 3 *设备*离线状态下更新*设备影子*

更新完成后，刷新*设备影子*，我们可以观察到*影子*增加了desired字段，且字段内容即为我们刚刚设置的fre1频率值。



Figure 4更新后的*设备影子*具体内容

在传统的业务场景中，此时由于*设备*处于离线状态，平台端下发的设置LED灯闪烁频率的命令会丢失。而本实验中由于使用了*设备影子*功能，命令被保存在了*设备影子*中。我们让*设备*重新上电连接到物联网平台，此时*设备*会主动获取*设备影子*中的内容，并据此进行更新，随后上报最新状态。可以观察到开发板上的LED灯闪烁频率加快，同时*设备影子*内容更新。



Figure 5 *设备*重新上线后的*设备影子*内容

**课程小结**

本次课程学习了物联网平台提供的*设备影子*功能，并对*设备影子*JSON文件的格式以及*设备*、*影子*和应用程序之间通信的数据流进行了详细的介绍。在此基础上，通过控制LED灯闪烁的实验实现了*设备*掉线重连情况下的指令成功下发，帮助读者理解其实际应用价值。