Alibaba Cloud IoT Platform

Best Practices

Issue: 20190718

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. **You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.**
2. **No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.**
3. **The content of this document may be changed due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.**
4. **This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults " and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity , applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use**

or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified , reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates . The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 Danger: Resetting will result in the loss of user configuration data.
	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 Warning: Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus , page names, and other UI elements.	Click OK.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich {stand | slave}`

Contents

1 Connect to IoT Platform using MQTT.fx

This article uses MQTT.fx as an example to describe the method for using a third-party MQTT client to connect to IoT Platform. MQTT.fx is a MQTT client that is written in Java language and based on Eclipse Paho. It supports subscribing to messages and publishing messages through topics.

Prerequisites

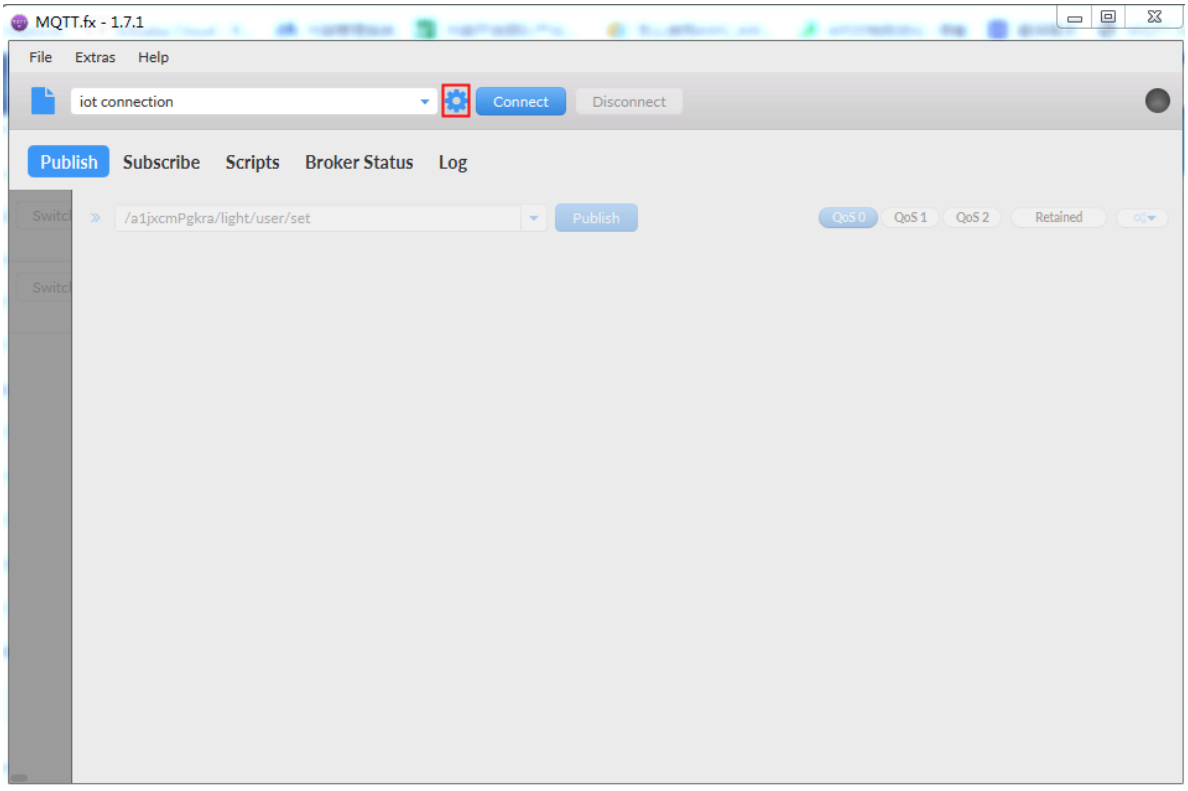
You have created products and devices in the IoT Platform console, and have got the ProductKey, DeviceName, and DeviceSecret of the devices. When you set the connection parameters for MQTT.fx, you will use the values of the ProductKey, DeviceName, and DeviceSecret. See Create a product, Create a device, and Create multiple devices at a timefor help when creating products and devices.

Procedure

1. Download and install the MQTT.fx software.

   Download the MQTT.fx software for Windows from MQTT.fx website.

2. Open MQTT.fx, and click the settings icon.



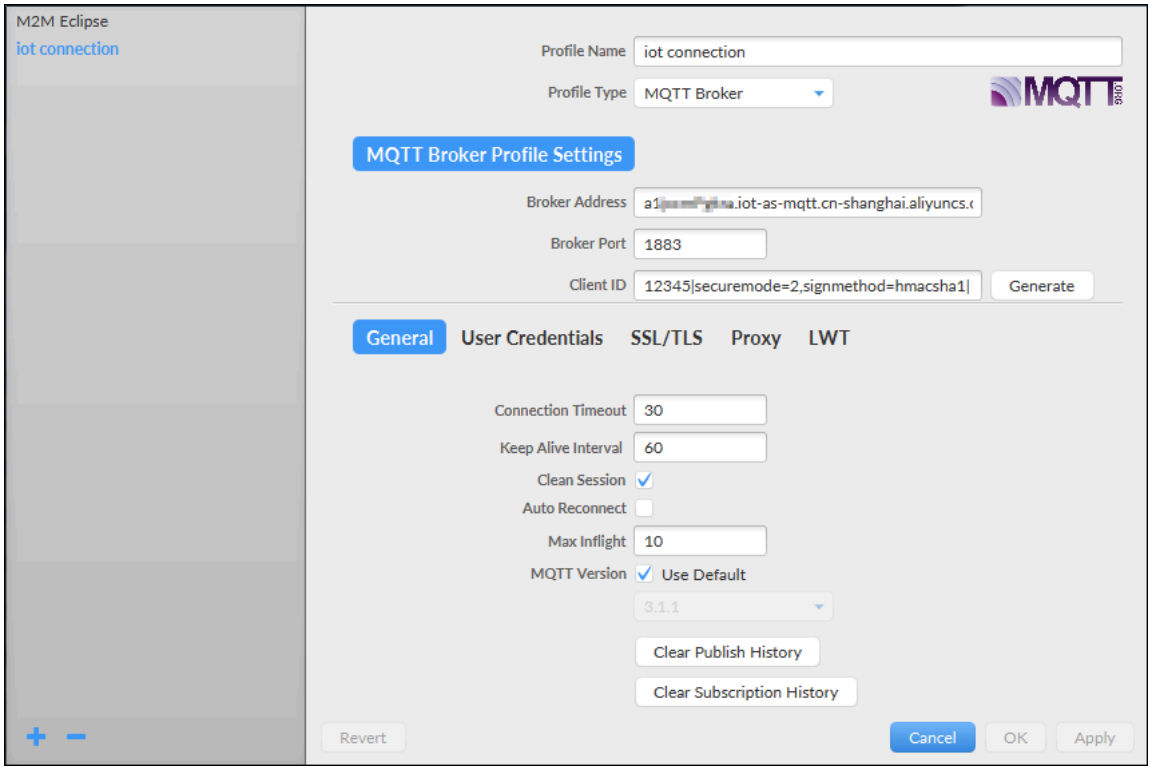
3. Set the connection parameters.

   Currently, two types of connection modes are supported: TCP and TLS. These two modes only differ in settings of Client ID and SSL/TLS.

   The procedure is as follows:

   a. Enter basic information. See the following table for parameter descriptions.

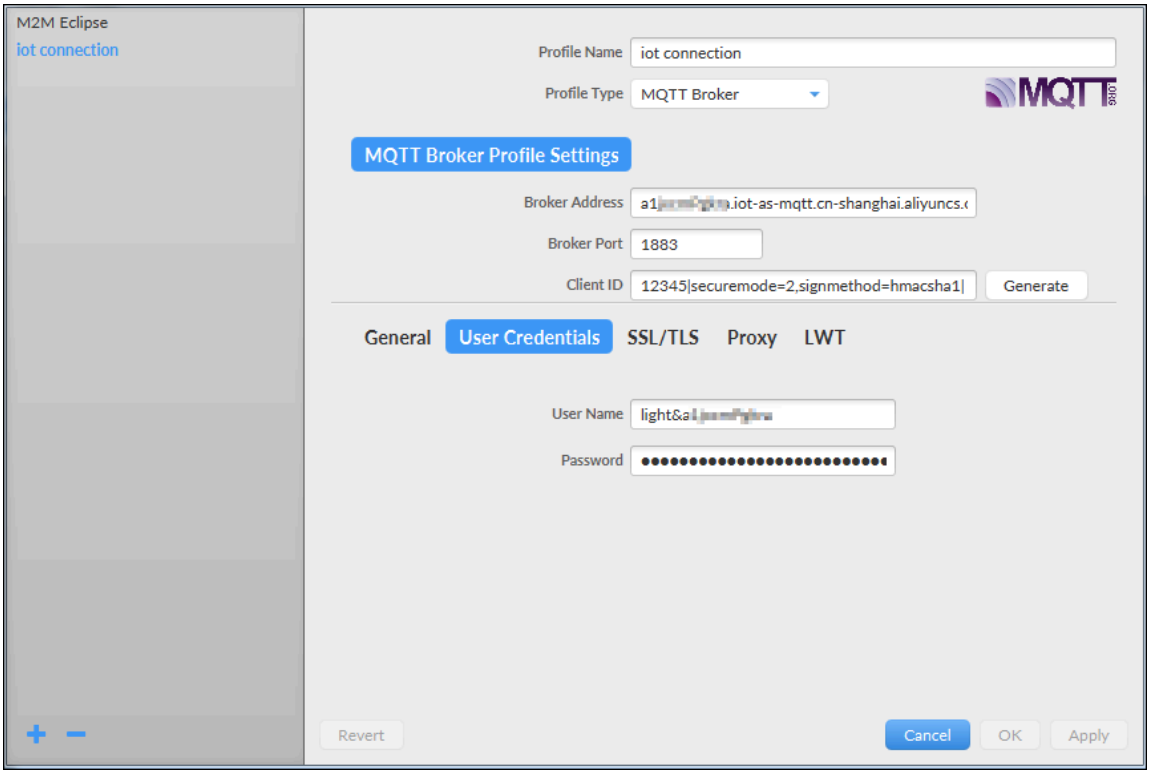
      You can keep the default parameters for General, or set the values according to your needs.



Parameter	Description
Profile Name	Enter a custom profile name.
Profile Type	Select MQTT Broker.
Broker Address	Enter the connection domain in the format of `${YourProductKey}.iot-as-mqtt.${region}.aliyuncs.com`. In this format, variable *`${region}`* indicates the region ID of your IoT Platform service region. For region IDs, see Regions and zones. Example: alPUPCoxxxx.iot-as-mqtt.cn-shanghai.aliyuncs.com.
Broker Port	Set to `1883`.

Parameter	Description
Client ID	Enter a value in the format of `${clientId}\|securemode=3,signmethod=hmacsha1\|`. Example: `12345\|securemode=3,signmethod=hmacsha1\|`. The parameters are described as follows: · *`${clientId}`* is a custom client ID. It can be any value within 64 characters. We recommend that you use the MAC address or SN code of the device as the value of clientId. · *`securemode`* is the security mode of the connection. If you use the TCP mode, set it as securemode=3; if you use the TLS mode, set it as securemode=2. · *`signmethod`* is the signature method that you want to use. IoT Platform supports `hmacmd5` and `hmacsha1`. Note: Do not click Generate after you enter the Client ID information.

b. Click User Credentials, and enter your User Name and Password.

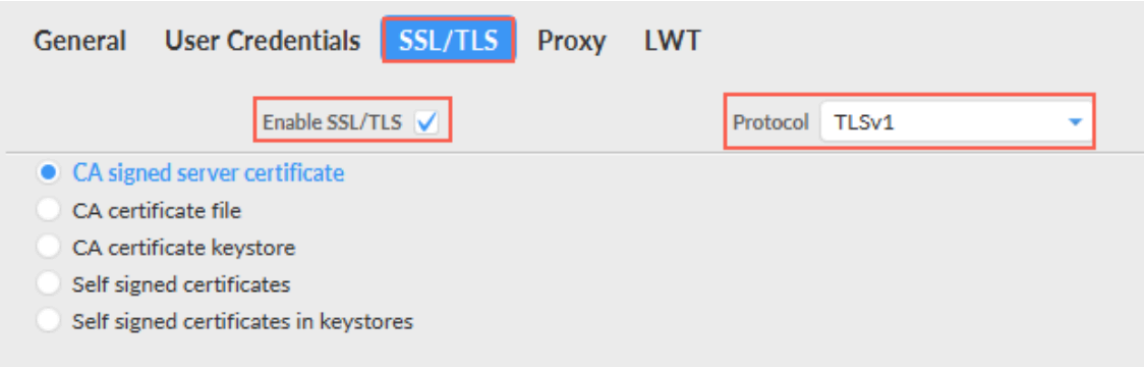


Parameter	Description
User Name	It must be the device name directly followed by the character "&" and the product key. Format: `${YourDeviceName}&${YourPrductKey}`. For example, device&fOAt5H5TOWF.

Parameter	Description
Password	You must enter an encrypted value of the input parameters. · IoT Platform provides a Password Generator for you to generate one easily. Parameters in the password generator: - productKey: The unique identifier of the product to which the device belongs. You can view this information on the device details page in the console. - deviceName: The name of the device. You can view this information on the device details page in the console. - deviceSecret: The device secret. You can view this information on the device details page in the console. - timestamp: (Optional) Timestamp of the current system time. - clientId: The custom client ID, which must be the same as the value of *${clientId}* in Client ID. - method: The signature algorithm, which must be the same as the value of *signmethod* in Client ID. · You can also encrypt a password by yourself. Generate a password manually: A. Sort and join the parameters. Sort and join the parameters `clientId` , `deviceName` , `productKey` , and `timestamp` in a lexicographical order. (If you have not set a timestamp, do not include timestamp in the string.) Joint string example: `clientId12 345deviceN amedevicep roductKeyf OAt5H5TOWF` B. Encrypt. Use the `deviceSecr et` of the device as the secret key to encrypt the joint string by the signature algorithm defined in Client ID. Suppose the `deviceSecr et` of the device is abc123, the encryption format is `hmacsha1 ( abc123 , clientId12 345deviceN amedevicep roductKeyf OAt5H5TOWF )`.

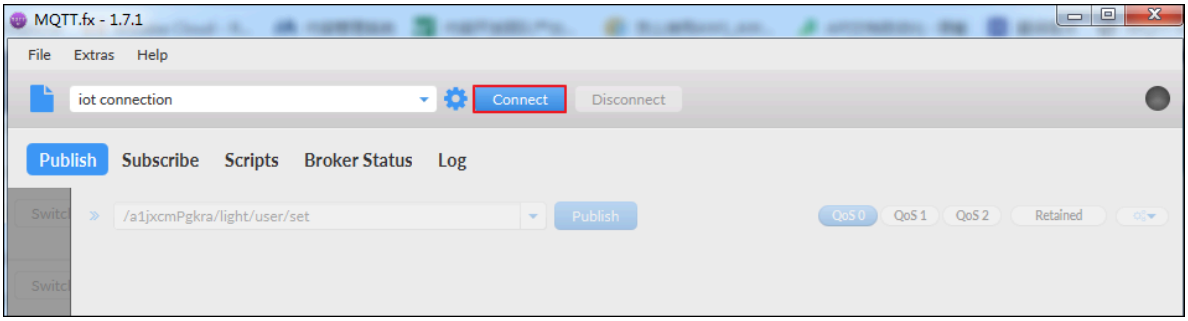
c. If you use TLS connection mode, you are required to set information for SSL/TLS. SSL/TLS settings are not required when the connection mode is TCP.

Check the box for Enable SSL/TLS, and select TLSv1 as the protocol.



d. Enter all the required information, and then click OK.

4. Click Connect to connect to IoT Platform.

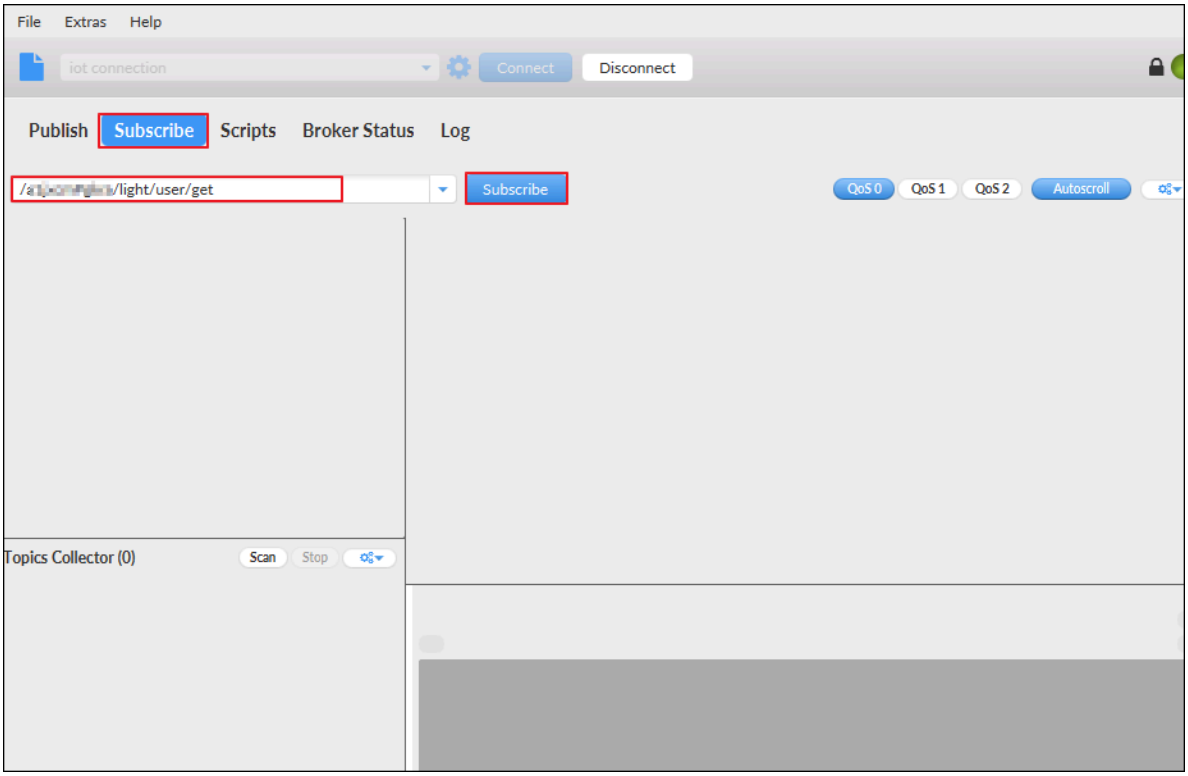


Message communication test

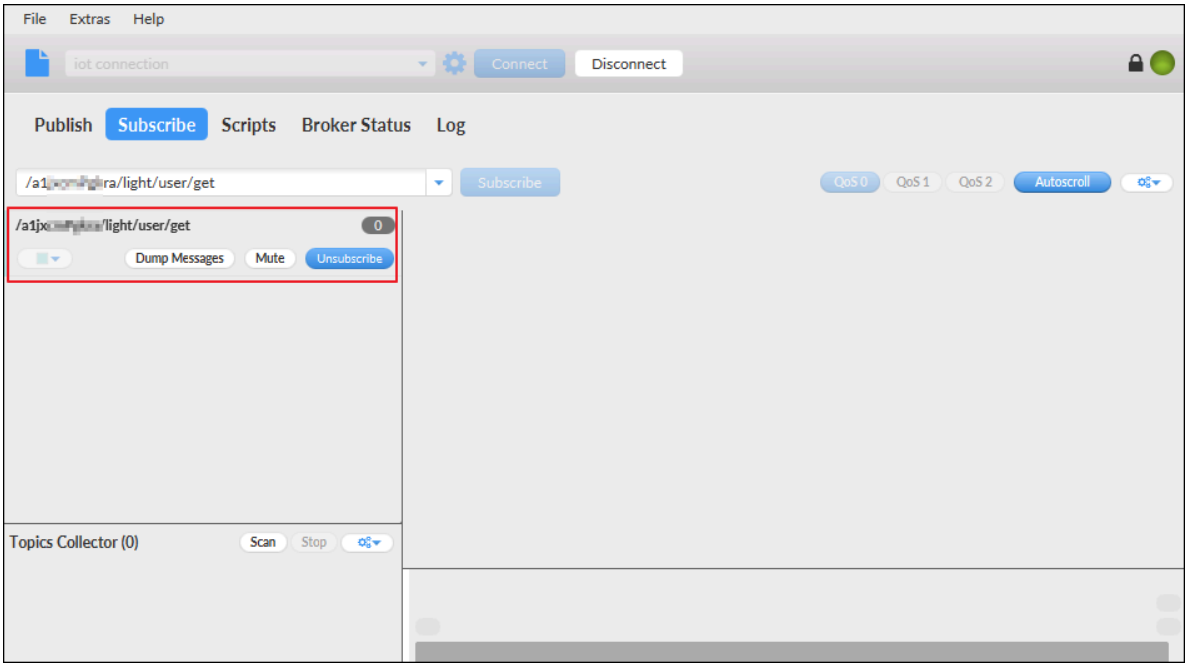
Test whether MQTT.fx and IoT Platform are successfully connected.

1. In MQTT.fx, click Subscribe.

2. Enter a topic of the device, and then click Subscribe.

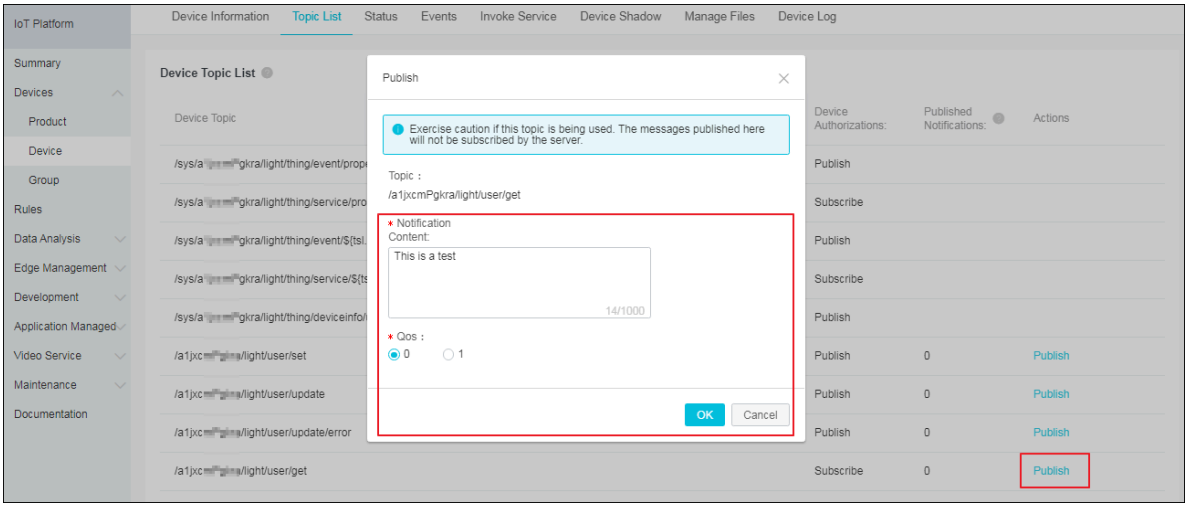


After you have successfully subscribed to a topic, it is displayed in the topic list.

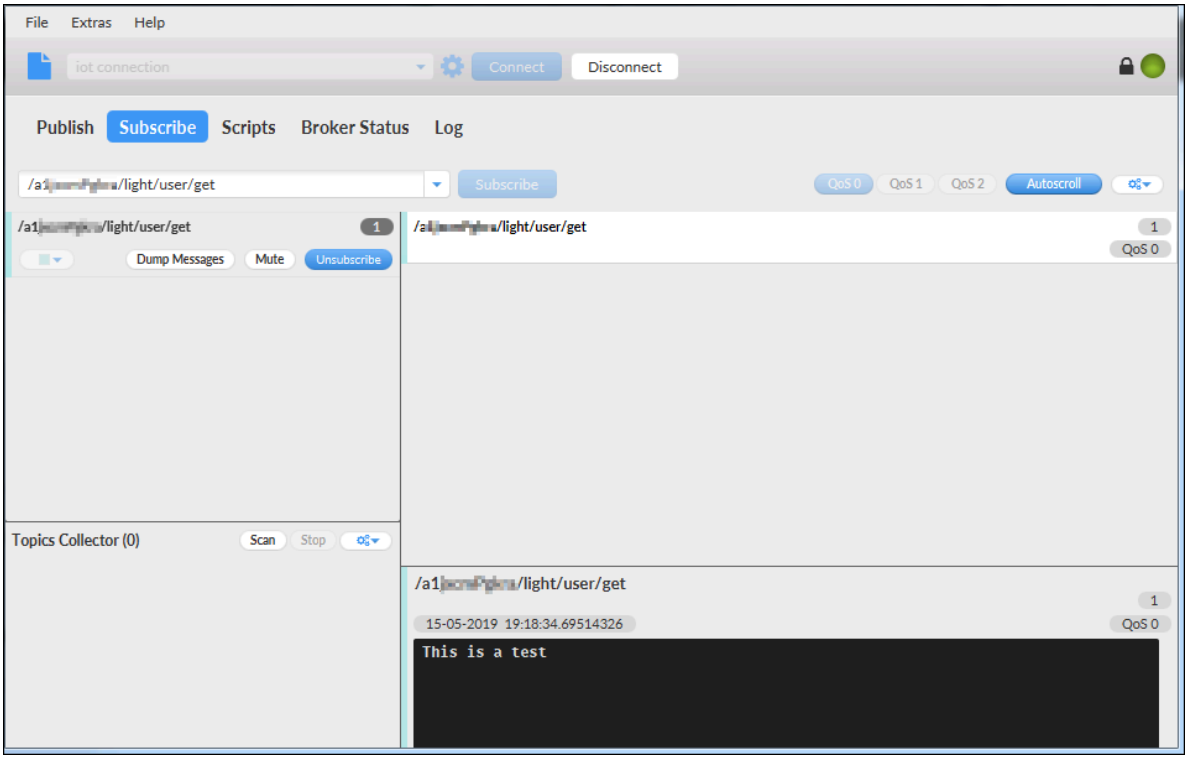


3. In the IoT Platform console, in the Topic List of the Device Details page, click the Publish button of the topic that you have subscribed to.

4. Enter message content, and click OK.

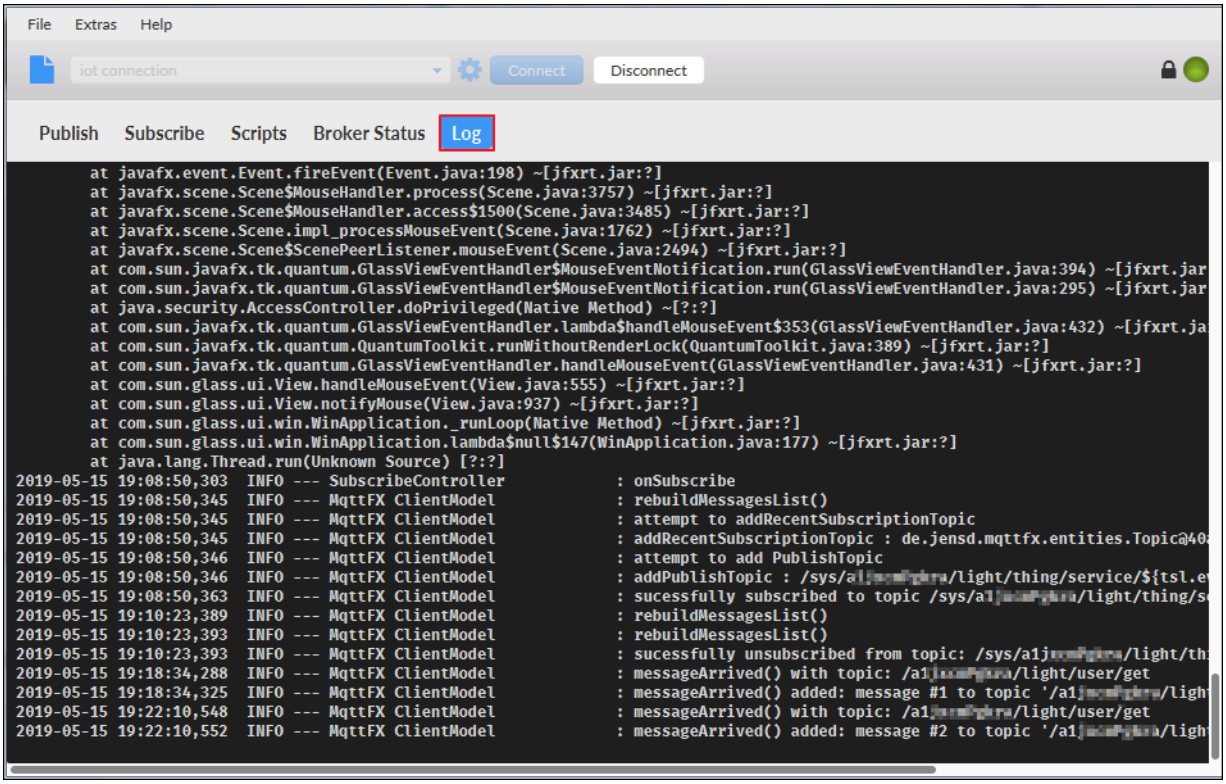


5. Go back to MQTT.fx to check if the message has been received.



View logs

In MQTT.fx, click Log to view the operation logs and error logs.


File
Extras
Help
iot connection
Connect
Disconnect
Publish
Subscribe
Scripts
Broker Status
Log
at javafx.event.Event.fireEvent(Event.java:198) ~[jfxrt.jar:?]
at javafx.scene.Scene$MouseHandler.process(Scene.java:3757) ~[jfxrt.jar:?]
at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485) ~[jfxrt.jar:?]
at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762) ~[jfxrt.jar:?]
at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:394) ~[jfxrt.jar
at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:295) ~[jfxrt.jar
at java.security.AccessController.doPrivileged(Native Method) ~[?:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$353(GlassViewEventHandler.java:432) ~[jfxrt.ja
at com.sun.javafx.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389) ~[jfxrt.jar:?]
at com.sun.javafx.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:431) ~[jfxrt.jar:?]
at com.sun.glass.ui.View.handleMouseEvent(View.java:555) ~[jfxrt.jar:?]
at com.sun.glass.ui.View.notifyMouse(View.java:937) ~[jfxrt.jar:?]
at com.sun.glass.ui.win.WinApplication._runLoop(Native Method) ~[jfxrt.jar:?]
at com.sun.glass.ui.win.WinApplication.lambda$null$147(WinApplication.java:177) ~[jfxrt.jar:?]
at java.lang.Thread.run(Unknown Source) [?:?]
2019-05-15 19:08:50,303 INFO --- SubscribeController : onSubscribe
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : attempt to addRecentSubscriptionTopic
2019-05-15 19:08:50,345 INFO --- MqttFX ClientModel : addRecentSubscriptionTopic : de.jensd.mqttfx.entities.Topic@40
2019-05-15 19:08:50,346 INFO --- MqttFX ClientModel : attempt to add PublishTopic
2019-05-15 19:08:50,346 INFO --- MqttFX ClientModel : addPublishTopic : /sys/a
/light/thing/service/${tsl.e
2019-05-15 19:08:50,363 INFO --- MqttFX ClientModel : sucessfully subscribed to topic /sys/a
/light/thing/s
2019-05-15 19:10:23,389 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:10:23,393 INFO --- MqttFX ClientModel : rebuildMessagesList()
2019-05-15 19:10:23,393 INFO --- MqttFX ClientModel : sucessfully unsubscribed from topic: /sys/a1
/light/th
2019-05-15 19:18:34,288 INFO --- MqttFX ClientModel : messageArrived() with topic: /a1
/light/user/get
2019-05-15 19:18:34,325 INFO --- MqttFX ClientModel : messageArrived() added: message #1 to topic '/a1
/ligh
2019-05-15 19:22:10,548 INFO --- MqttFX ClientModel : messageArrived() with topic: /a1
/light/user/get
2019-05-15 19:22:10,552 INFO --- MqttFX ClientModel : messageArrived() added: message #2 to topic '/a1
/ligh

2 Connect Android Things to Alibaba Cloud IoT Platform

This article uses an indoor air test project as an example to explain how to connect Google Android Things to Alibaba Cloud IoT Platform.

Hardware

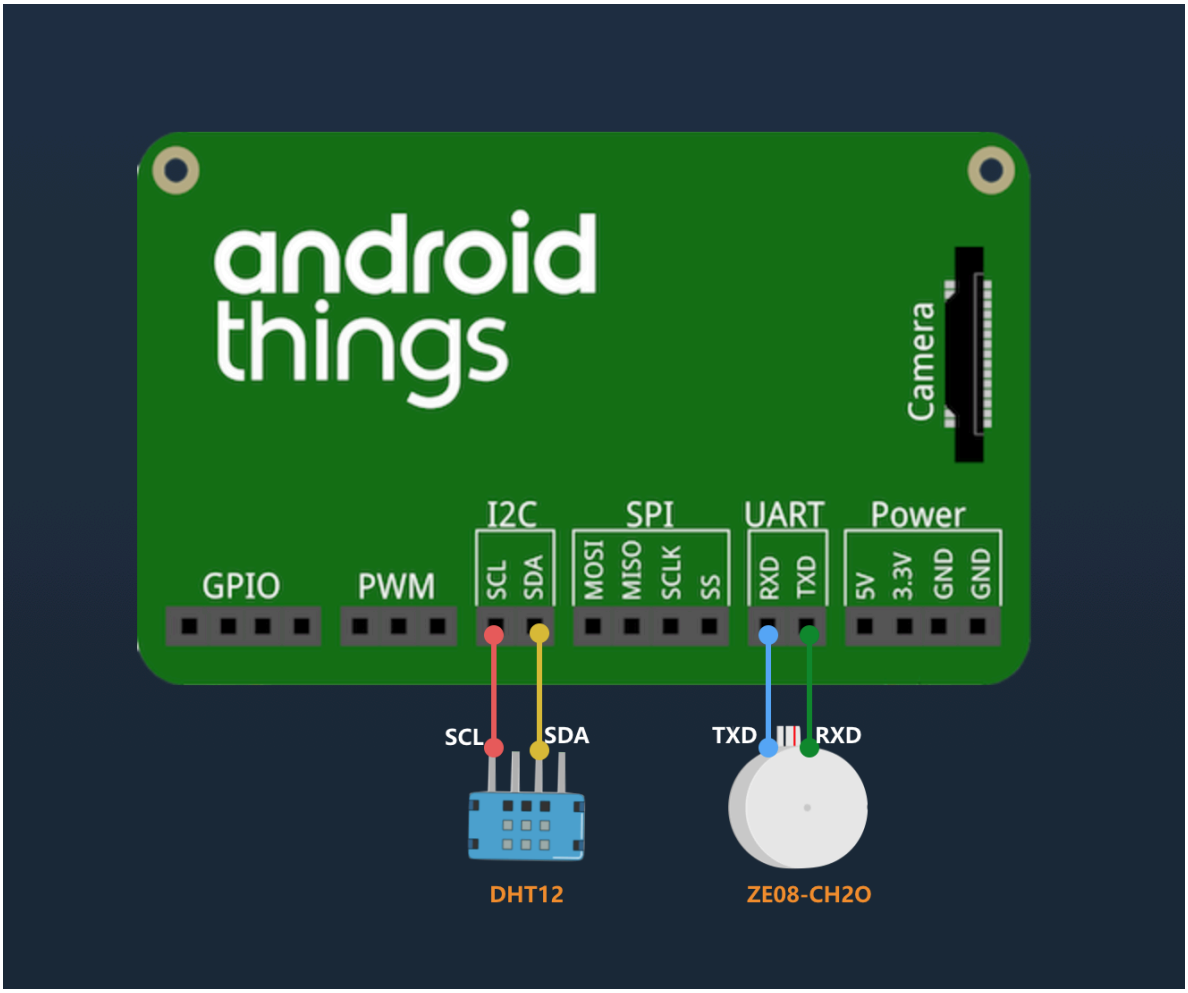
- Hardware list for the project

  The following table lists the hardware required by the indoor air test project.

Hardware	Picture	Remarks
NXP Pico i.MX7D Development board		Android Things 1.0 Note: You can also use Raspberry Pi instead.
DHT12 Temperature and humidity sensor		Supports I2C data communication method.

Hardware	Picture	Remarks
ZE08-CH2O Formaldehyde detection sensor		Supports UART data communication method.

- Diagram of the hardware connection



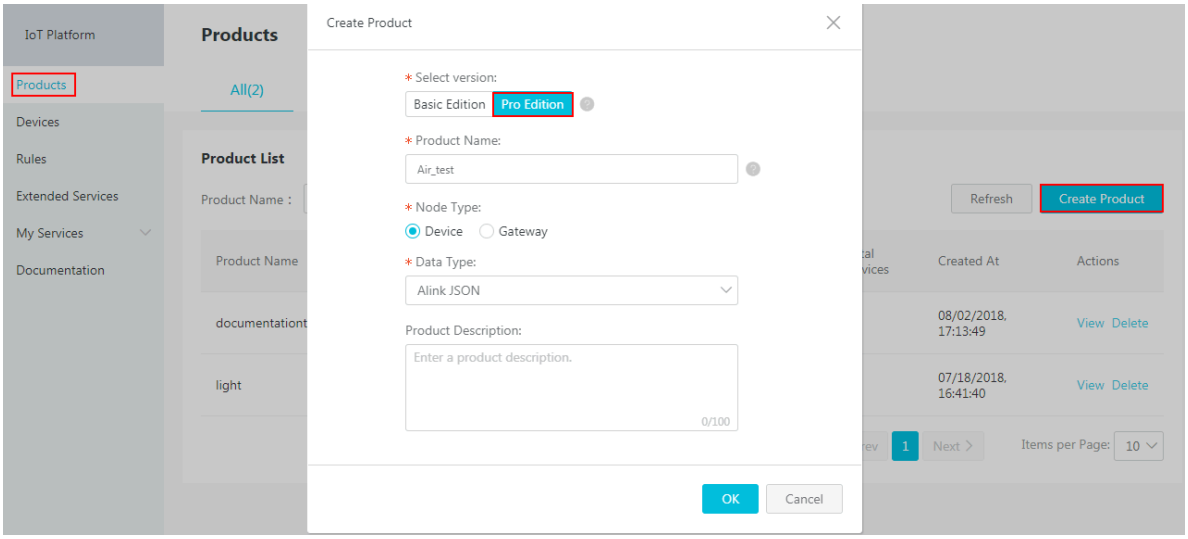
  - Connect the SCL (clock line) and SDA (data line) pins of the temperature and humidity sensor (DHT12) with the I2C SCL and SDA pins of the development board.
  - Connect the TXD (transmit data) pin of the formaldehyde detection sensor (ZE08-CH2O) with the RXD (receive data) pin of the development board, and connect the RXD pin of ZE08-CH2O with the TXD pin of the development board.

Create a product and device in the Alibaba Cloud IoT Platform console

1. Log on to the IoT Platform console.

2. Create a product in IoT Platform.

   On the Products page, click Create Product. For more information, see Create a product.



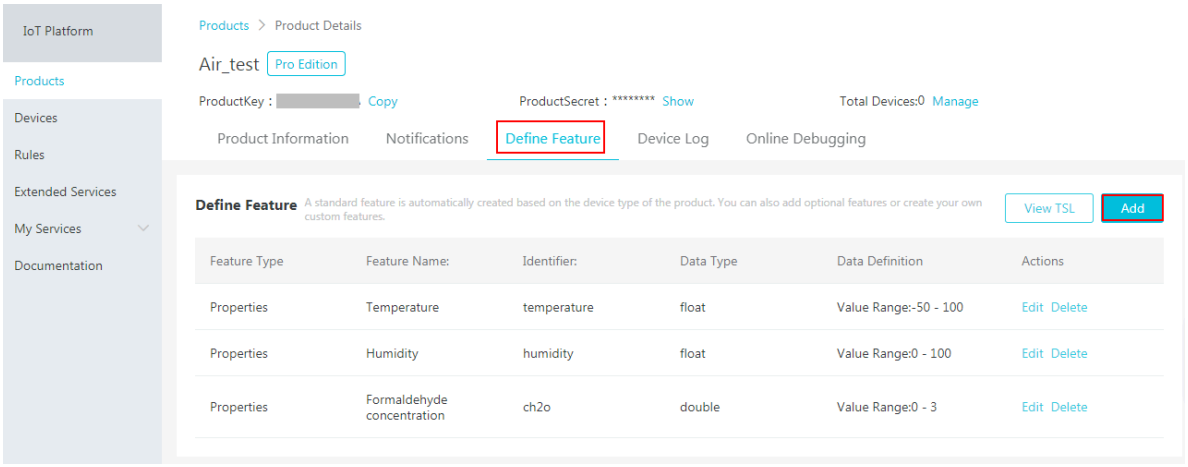
3. Define features for the newly created product.

   On the Product Details page, click Define Feature > Add, and then define properties for the product. For more information, see Overview.

   Table 2-1: Properties required for the indoor air test project

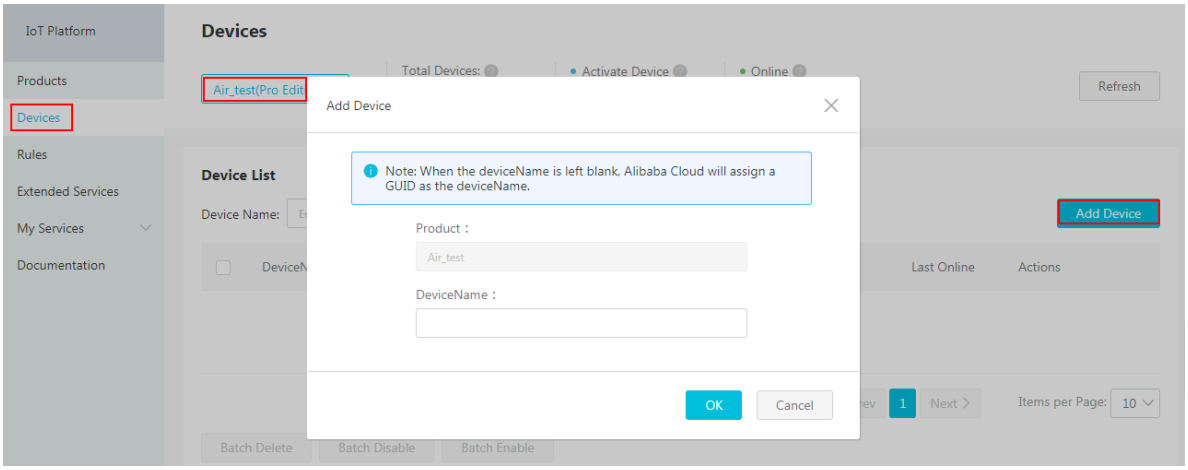
Property name	Identifier	Data type	Value range	Description
Temperature	temperature	float	-50~100	Detected by DHT12.
Humidity	humidity	float	0~100	Detected by DHT12.

Property name	Identifier	Data type	Value range	Description
Formaldehyde concentration	ch2o	double	0~3	Detected by ZE08.



4. Create a device

   On the Devices page, select the name of the newly created product, click Add Device, and then create a device. For more information, see Create a device.



Develop the Android Things client

1. **Use Android Studio to create an Android Things project, and add the permission for Internet.**

```
< uses - permission   android : name =" android . permission .
 INTERNET " />
```

2. **Add eclipse.paho.mqtt to Gradle file.**

```
implementa  tion  ' org . eclipse . paho : org . eclipse . paho .
client . mqttv3 : 1 . 2 . 0 '
```

3. **Set that data of DHT12 is read through I2C.**

```
private   void   readDataFr  omI2C () {

        try  {

            byte []  data  =  new   byte [ 5 ];
            i2cDevice . readRegBuf  fer ( 0x00 ,  data ,  data .
length );

           //  check   data
            if  (( data [ 0 ] +  data [ 1 ] +  data [ 2 ] +  data [
3 ]) %  256  ! =  data [ 4 ]) {
                humidity  =  temperatur  e  =  0 ;
                return ;
           }
           //  humidity   data
            humidity  =  Double . valueOf ( String . valueOf ( data
[ 0 ]) + "." +  String . valueOf ( data [ 1 ]));
            Log . d ( TAG , " humidity : " +  humidity );
           //  temperatur  e   data
            if  ( data [ 3 ] <  128 ) {
                temperatur  e  =  Double . valueOf ( String .
valueOf ( data [ 2 ]) + "." +  String . valueOf ( data [ 3 ]));
           }  else  {
                temperatur  e  =  Double . valueOf ("-" +  String .
valueOf ( data [ 2 ]) + "." +  String . valueOf ( data [ 3 ] -  128
));
           }

            Log . d ( TAG , " temperatur  e : " +  temperatur  e );

       }  catch  ( IOExceptio  n   e ) {
            Log . e ( TAG , " readDataFr  omI2C   error  " +  e .
getMessage (),  e );
       }

   }
```

4. **Set that data of Ze08-CH2O is read through UART.**

```
try  {
               //  data   buffer
                byte []  buffer  =  new   byte [ 9 ];

                while  ( uartDevice . read ( buffer ,  buffer .
length ) >  0 ) {
```

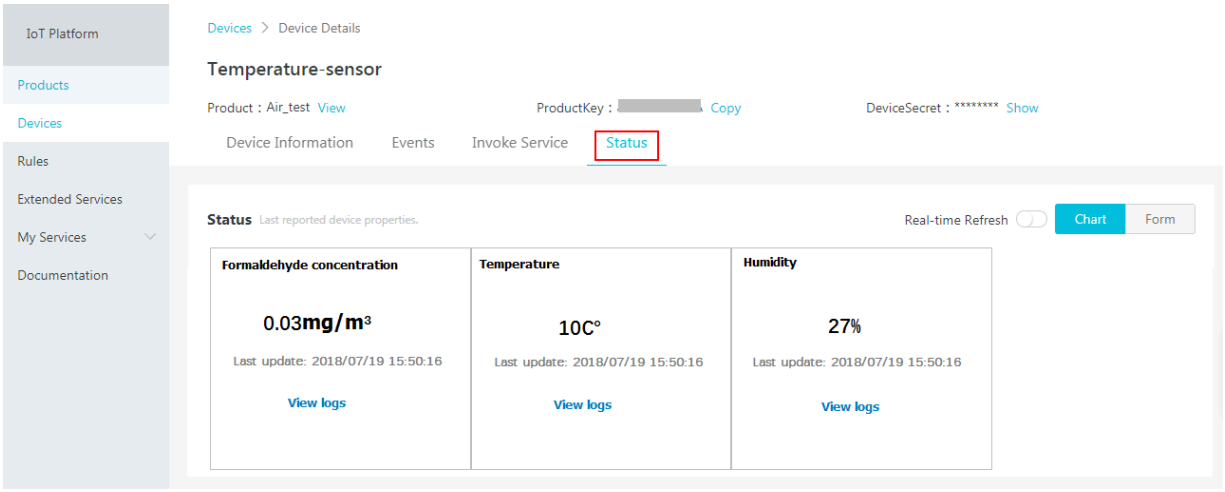
```
                    if  ( checkSum ( buffer )) {
                        ppbCh2o  =  buffer [ 4 ] *  256  +  buffer
[ 5 ];
                        ch2o  =  ppbCh2o  /  66 . 64  *  0 . 08 ;
                    }  else  {
                        ch2o  =  ppbCh2o  =  0 ;
                    }
                     Log . d ( TAG , " ch2o : " +  ch2o );
                }

            }  catch  ( IOExceptio  n   e ) {
                 Log . e ( TAG , " Ze08CH2O  read  data  error  "
+  e . getMessage (),  e );
            }
```

5. Connect Alibaba Cloud IoT Platform and the client, and report data.

```
/*
 Payload   format
{
  " id ":  123243 ,
  " params ": {
    " temperatur  e ":  25 . 6 ,
    " humidity ":  60 . 3 ,
    " ch2o ":  0 . 048
  },
  " method ": " thing . event . property . post "
}
*/
 MqttMessag  e   message  =  new   MqttMessag  e ( payload .
 getBytes (" utf - 8 "));
 message . setQos ( 1 );

 String   pubTopYour  Pc  = "/ sys /${ YourProduc  tKey }/${
 YourDevice  Name }/ thing / event / property / post ";

 mqttClient . publish ( pubTopic ,  message );
```

View real-time data of the device

After the device is enabled, you can view the real-time data of the device from the Status column on the Device Details page in IoT Platform console.



3 Connect NodeMCU (ESP8266) to IoT Platform

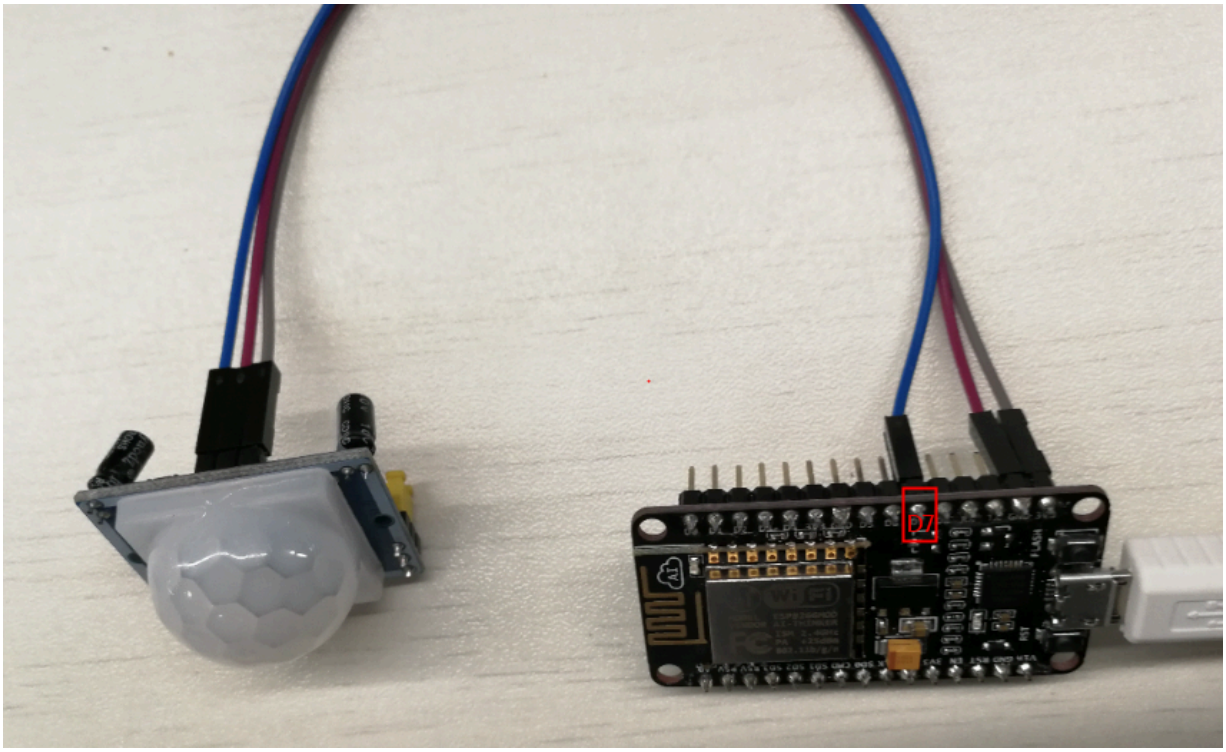
You can use NodeMCU to develop IoT applications and connect your NodeMCU to Alibaba Cloud IoT Platform. In this way, you can control your NodeMCU application server from IoT Platform. This topic describes how to connect a NodeMCU application server to IoT Platform by using C programming code in an example.

Background information

NodeMCU is a development board based on the ESP8266 chip. The ESP8266 chip integrates the Wi-Fi function. Therefore, the NodeMCU application server can be used to control its connected devices in the same LAN. However, if you cannot directly operate the NodeMCU application server, you cannot control the devices. To resolve this issue, connect your NodeMCU application server to Alibaba Cloud IoT Platform so that you can control the NodeMCU application server from IoT Platform.

The following section illustrates how to use NodeMCU to control a light bulb device.

> **Note:**
> Connect the data cable of the sensor to the D7 pin, which corresponds to GPIO 13, on the NodeMCU development board.



Prerequisites

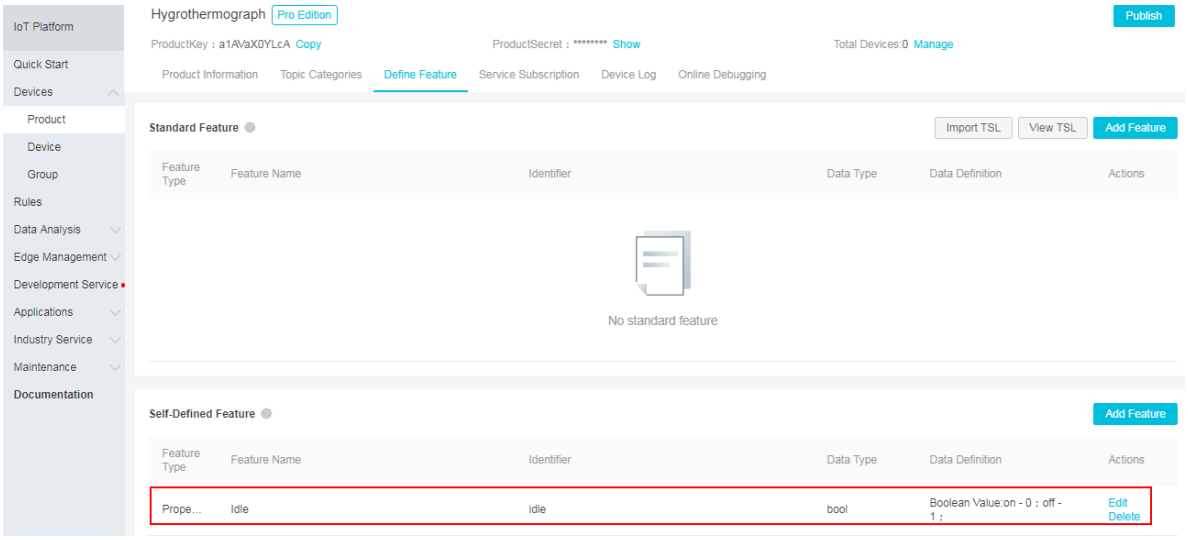
Before you begin, you must see the official NodeMCU IDE Tutorial for development software preparation . This example uses Arduino IDE as development software.

Connect NodeMCU to IoT Platform

After the NodeMCU development environment is ready, you can connect NodeMCU to IoT Platform.

1. Log on to the IoT Platform console.
2. Create a product, as shown in Create a product.
3. Define features for the product, as shown in Define features.

   This example uses the following properties:



4. Create a device, and obtain the device certificate information, including ProductKey, DeviceName, and DeviceSecret. For more information, see Create a device.
5. Write code.

   a. Connect your computer to the NodeMCU development board by using a USB cable.

   b. Open the Arduino IDE and write esp8266.ino code.

   c. Upload code.

   The sample code is as follows:

> **Note:**

You can generate the value of `MQTT_PASSW D` by using the mqttPassword encryption method described in Establish MQTT connections over TCP.

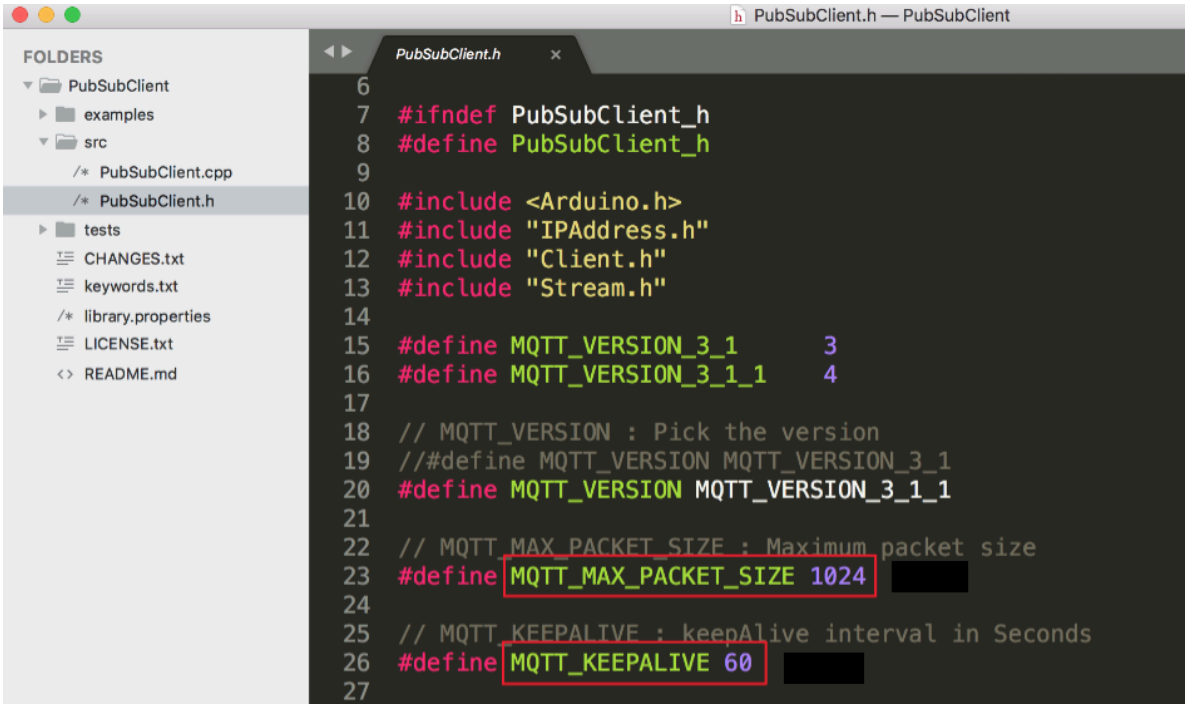
```
# include < ESP8266WiF i . h >
/* The PubSubClie nt 2 . 4 . 0 dependency */
# include < PubSubClie nt . h >
/* The ArduinoJso n 5 . 13 . 4 dependency */
# include < ArduinoJso n . h >

# define SENSOR_PIN 13

/* The SSID and password for Wi - Fi connection */
# define WIFI_SSID " SSID "
# define WIFI_PASSW D " Password "


/* Device certificat e informatio n */
# define PRODUCT_KE Y " The ProductKey of the
 device "
# define DEVICE_NAM E " The DeviceName of the
 device "
# define DEVICE_SEC RET " The DeviceSecr et of the
 device "
# define REGION_ID " The region ID , such as cn
 - shanghai "

/* The domain name and port number that do not
 require any modificati ons . */
# define MQTT_SERVE R PRODUCT_KE Y ". iot - as - mqtt
 ." REGION_ID ". aliyuncs . com "
# define MQTT_PORT 1883
# define MQTT_USRNA ME DEVICE_NAM E "&" PRODUCT_KE Y

# define CLIENT_ID " esp8266 | securemode = 3 , timestamp
 = 1234567890 , signmethod = hmacsha1 |"
// See " Establish MQTT connection s over TCP " ( https
 :// www . alibabaclo ud . com / help / doc - detail / 73742 . html
 ) to generate a password .
// The plain text for encryption is a combinatio n
   of parameter - value pairs in lexicograp hical order
 . For example , clientIdes p8266devic eName ${ deviceName }
 productKey ${ productKey } timestamp1 234567890 .
// The encryption key is the DeviceSecr et .
# define MQTT_PASSW D " f1ef80ee8a dd322c519a
 5a6bec326d c499f854b8 "

# define ALINK_BODY _FORMAT "{\" id \":\" 123 \",\"
 version \":\" 1 . 0 \",\" method \":\" thing . event . property .
 post \",\" params \":% s }"
# define ALINK_TOPI C_PROP_POS T "/ sys /" PRODUCT_KE Y
 "/" DEVICE_NAM E "/ thing / event / property / post "

 unsigned long lastMs = 0 ;
 WiFiClient espClient ;
 PubSubClie nt client ( espClient );


 void callback ( char * topic , byte * payload , unsigned
 int length )
{
     Serial . print (" Message arrived [");
     Serial . print ( topic );
```

```
     Serial . print ("] ");
     payload [ length ] = '\ 0 ';
     Serial . println (( char  *) payload );

}


 void   wifiInit ()
{
     WiFi . mode ( WIFI_STA );
     WiFi . begin ( WIFI_SSID ,  WIFI_PASSW  D );
     while  ( WiFi . status () ! =  WL_CONNECT  ED )
    {
         delay ( 1000 );
         Serial . println (" WiFi   not   Connect ");
    }

     Serial . println (" Connected   to   AP ");
     Serial . println (" IP   address : ");
     Serial . println ( WiFi . localIP ());


 Serial . print (" espClient  [");


     client . setServer ( MQTT_SERVE  R ,  MQTT_PORT );   /*
 Connect   to   the   MQTT   server   after   connecting   to   the
   Wi - Fi  */
     client . setCallbac  k ( callback );
}


 void   mqttCheckC  onnect ()
{
     while  (!  client . connected ())
    {
         Serial . println (" Connecting   to   MQTT   Server  ...");
         if  ( client . connect ( CLIENT_ID ,  MQTT_USRNA  ME ,
 MQTT_PASSW  D ))

        {

             Serial . println (" MQTT   Connected !") ;

        }
         else
        {
             Serial . print (" MQTT   Connect   err :");
             Serial . println ( client . state ());
             delay ( 5000 );
        }
    }
}


 void   mqttInterv  alPost ()
{
     char   param [ 32 ];
     char   jsonBuf [ 128 ];

     sprintf ( param , "{\" idle \":% d }",  digitalRea  d ( 13 ));
     sprintf ( jsonBuf ,  ALINK_BODY  _FORMAT ,  param );
     Serial . println ( jsonBuf );
```

```
    boolean   d  =  client . publish ( ALINK_TOPI  C_PROP_POS  T ,
 jsonBuf );
    Serial . print (" publish : 0   Failure ; 1   Success ");
    Serial . println ( d );
}


 void   setup ()
{

    pinMode ( SENSOR_PIN ,   INPUT );
   /*  initialize   serial   for   debugging  */
    Serial . begin ( 115200 );
    Serial . println (" Demo   Start ");

    wifiInit ();
}


//  the   loop   function   runs   over   and   over   again
 forever
 void   loop ()
{
    if  ( millis () -  lastMs  >=  5000 )
   {
        lastMs  =  millis ();
        mqttCheckC  onnect ();

       /* The   heartbeat   interval  */
        mqttInterv  alPost ();
   }

    client . loop ();
    if  ( digitalRea  d ( SENSOR_PIN ) ==  HIGH ){
    Serial . println (" Motion   detected !") ;
    delay ( 2000 );
     }
    else  {
    Serial . println (" Motion   absent !") ;
    delay ( 2000 );
  }
```

```
}
```

6. Modify the values of `MQTT_MAX_P ACKET_SIZE` and `MQTT_KEEPA LIVE` on the MQTT client as needed.
   a. Locate *PubSubClie nt* in the library manager of Arduino IDE.
   b. Open the *PubSubClie nt . h* file under *src* .
   c. Modify the MQTT_MAX_PACKET_SIZE and MQTT_KEEPALIVE parameters.
      To set the MQTT_MAX_PACKET_SIZE and MQTT_KEEPALIVE parameters, see Limits.



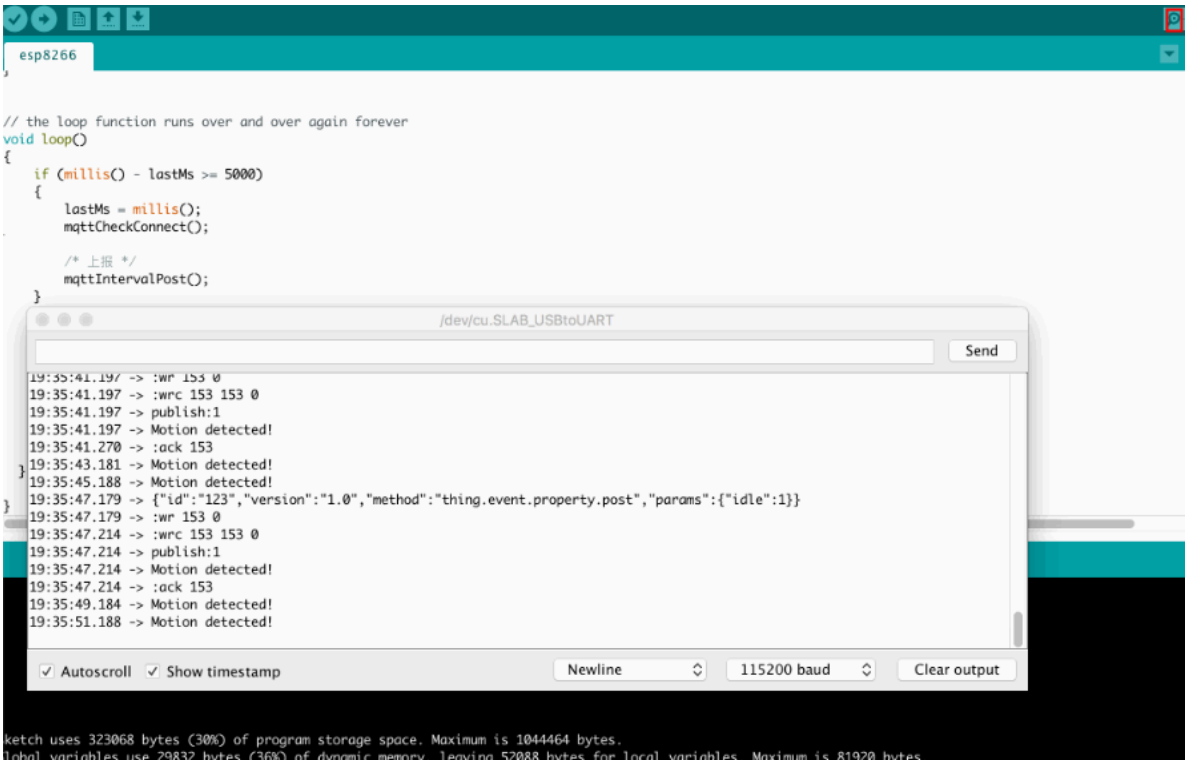
Perform a test

After the development is complete, you can use the device to report properties to test whether the NodeMCU application server and IOT Platform can receive reported properties from the device.

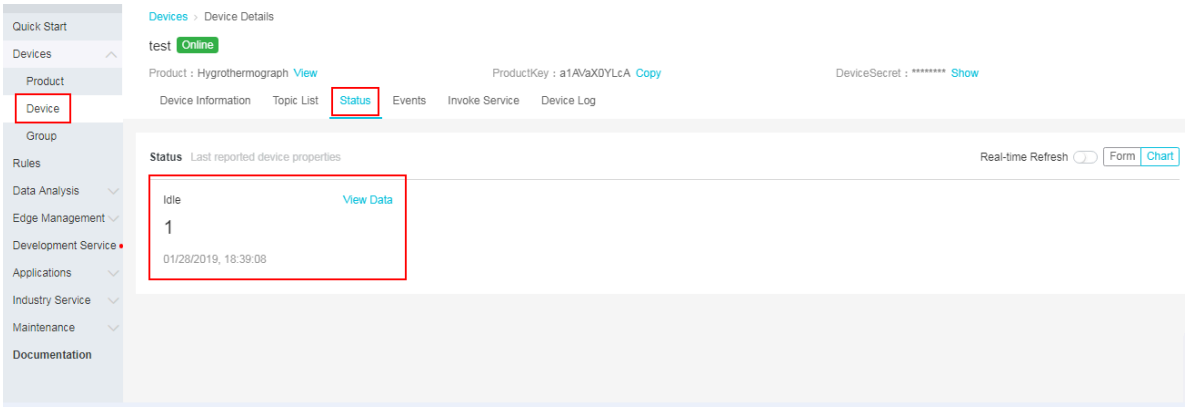
After the device reports properties, you can view the running status of the device on the development software and in the IOT Platform console.

- On the development software, click the serial port listener icon in the upper-right corner of the code editing page.

  You can view the running status of the program, as shown in the following figure:



- In the IoT Platform console, go to the Device Details page of the corresponding device. On the Status tab page , view the reported properties and property history, as shown in the following figure:



4 Upload temperature and humidity data to DingTalk chatbots

Context

- **Scenario:**

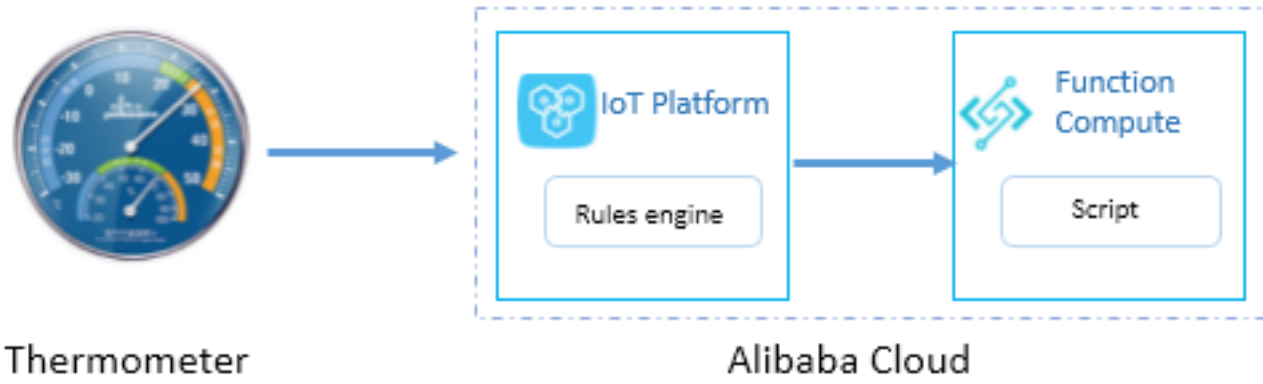
  Upload the data that is collected by the temperature and humidity sensor to DingTalk chatbots.

- **Business logic:**

  Connect the temperature and humidity sensor to IoT Platform through MQTT . Configure the rules engine to send the temperature and humidity data to the pushData2DingTalk function in Function Compute. The function processes the data and posts the results to the Webhook address of the specified DingTalk chatbot. The DingTalk group can then receive the temperature and humidity data sent by this DingTalk chatbot.

  The data flow diagram is shown in Figure 4-1: Data flow diagram.

  Figure 4-1: Data flow diagram



Procedure

1. Configure the DingTalk chatbot.
   a) Log on to the desktop version of DingTalk.
   b) In the upper-right corner of the DingTalk group page, click `...`, and then select ChatBot.
   c) Click Add Robot, select Custom, then click Add.
   d) Enter a name for the robot, click Next, and then click Finish.
2. Create a function.
   a) Activate Alibaba Cloud Function Compute service first.

      Function Compute is an event-driven, fully managed computing service. Currently supported languages include Java, Node.js and Python. For more information, see How to use Function Compute.
   b) Write the function script code. In this example, Node.js is used. The function obtains the device location, device number, temperature, humidity, and the time of recording from IoT Platform, and splices the data according to the specified

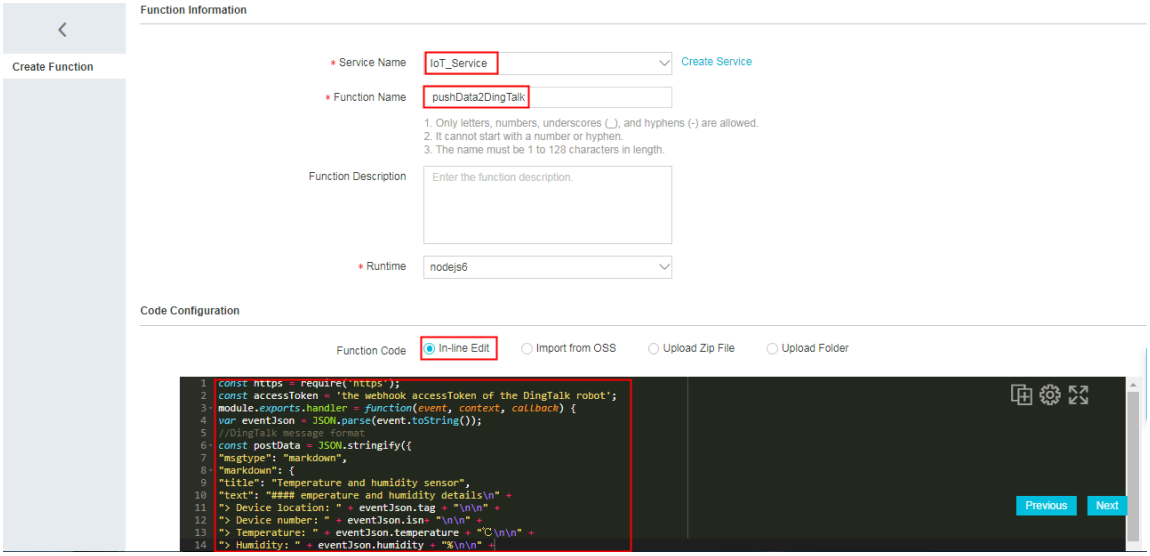
DingTalk message format. Data will be sent to the Webhook address of the specified DingTalk chatbot using HTTPS Post methods.

c) Log on to the Function Compute console and create a service named IoT_Service.

d) Click Create Function, and select the Empty Function template.

e) Select No Trigger and specify the basic configurations as shown in the following picture.

Figure 4-2: Basic configurations



The function pushData2DingTalk is declared as follows:

```
 const   https   =  require (' https ');
 const   accessToke  n  = ' Specify   the   webhook   accessToke
 n   of   the   DingTalk   robot ';
 module . exports . handler  =  function ( event ,  context ,
 callback ) {
 var   eventJson  =  JSON . parse ( event . toString ());
// DingTalk   message   format
 const   postData  =  JSON . stringify ({
" msgtype ": " markdown ",
" markdown ": {
" title ": " Temperatur  e   and   humidity   sensor ",
" text ": "####  Temperatur  e   and   humidity   details \ n " +
">  Device   location : " +  eventJson . tag  + "\ n \ n " +
">  Device   number : " +  eventJson . isn + "\ n \ n " +
">  Temperatur  e : " +  eventJson . temperatur  e  + "℃\ n \ n "
 +
">  Humidity : " +  eventJson . humidity  + "%\ n \ n " +
"> ###### " +  eventJson . time  + "  published   by  [ Alibaba
 Cloud   IoT   Platform ]( https :// www . aliyun . com / product
 / iot ) \ n "
},
" at ": {
" isAtAll ":  false
```

```
}
});
 const   options  = {
 hostname : ' oapi . dingtalk . com ',
 port :  443 ,
 path : '/ robot / send ?  access_tok  en =' +  accessToke  n ,
 method : ' POST ',
 headers : {
' Content - Type ': ' applicatio  n / json ',
' Content - Length ':  Buffer . byteLength ( postData )
}
};
 const   req  =  https . request ( options , ( res ) => {
 res . setEncodin  g (' utf8 ');
 res . on (' data ', ( chunk ) => {});
 res . on (' end ', () => {
 callback ( null , ' success ');
});
});
//  When   an   exception   returns
 req . on (' error ', ( e ) => {
 callback ( e );
});
//  Write   the   data
 req . write ( postData );
 req . end ();
};
```

3. **Configure IoT Platform.**
   a) **In the IoT Platform console, select Products and then create a product for the temperature and humidity sensor.**
   b) **On the Topic Categories tab page of the product details page, create a topic category `/ productKey /${ deviceName }/ user / data` whose Device Operation Authorizations is Publish.**
   c) **On the Define Feature tab page, define two properties: temperature and humidity.**
   d) **Select Devices and then register a device.**
   e) **Click View next to the device name. On the Device Tag tab, click Add to add two device tags.**

Tag Key	Tag Value	Description
tag	Room 007S, 3rd Floor, Building 2, Cloud Town	Device location

Tag Key	Tag Value	Description
deviceISN	T20180102X nbKjmoAnUb	Device number

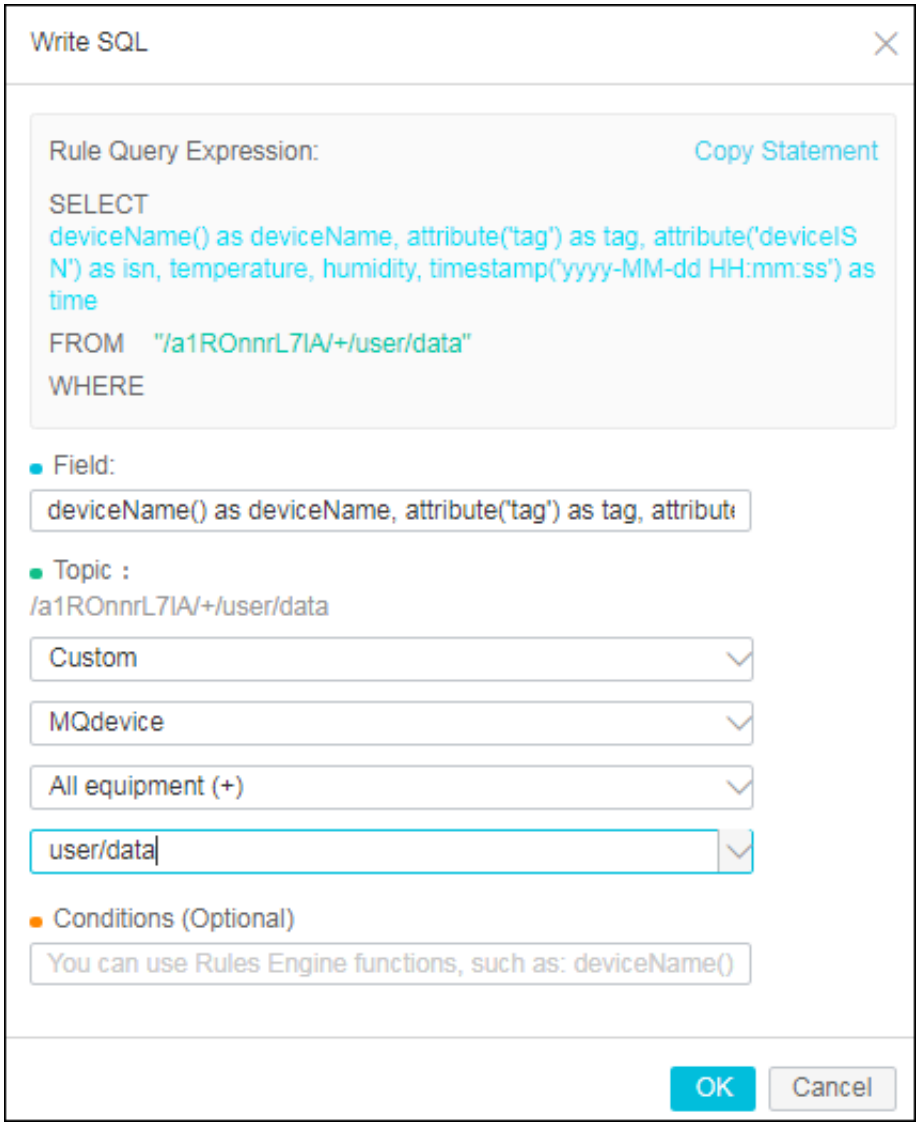
f) Select Rules to create and enable a rule. A complete rule contains three parts: basic information, data processing SQL, and data forwarding. You can specify multiple forwarding actions for a rule.

A. Configure the data processing script.

The rules engine supports SQL statements.

The device name (deviceName) and attributes (tag and deviceISN) are required.

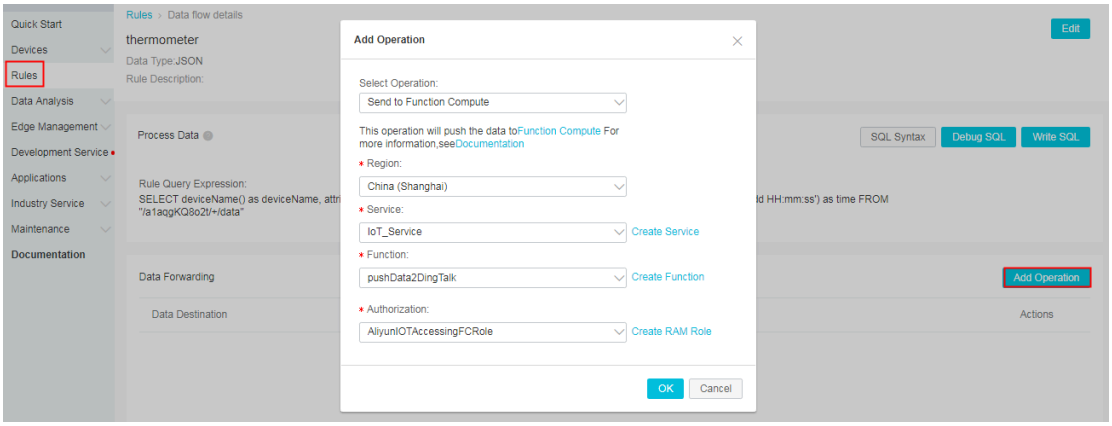
This SQL is to obtain the temperature and humidity values from the payload of the messages that are sent from the temperature and humidity sensor.



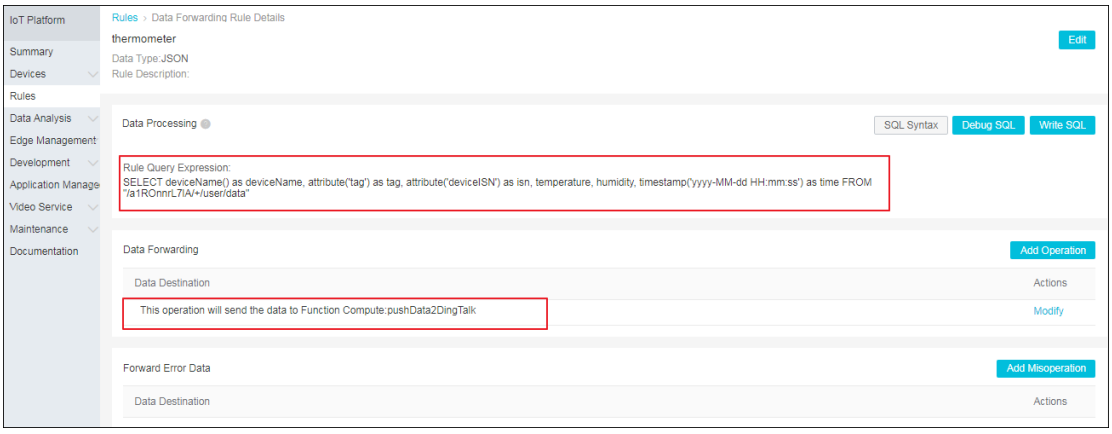
The SQL statements are as follows:

```
SELECT
deviceName ()  as   deviceName ,
attribute (' tag ')  as   tag ,
attribute (' deviceISN ')  as   isn ,
temperatur  e ,
humidity ,
timestamp (' yyyy - MM - dd   HH : mm : ss ')  as   time
FROM
"/ Specify   the   productKey /+/ user / data "
```

B. Configure the data forwarding action.



The complete rule is as follows:



g) On the rules page, click Enable to enable the rule.

4. Temperature and humidity sensor.

To facilitate testing, a Node.js program is used to simulate the temperature and humidity sensor and send the temperature and humidity data. The aliyun-iot-mqtt library is used in this example. The complete code is as follows:

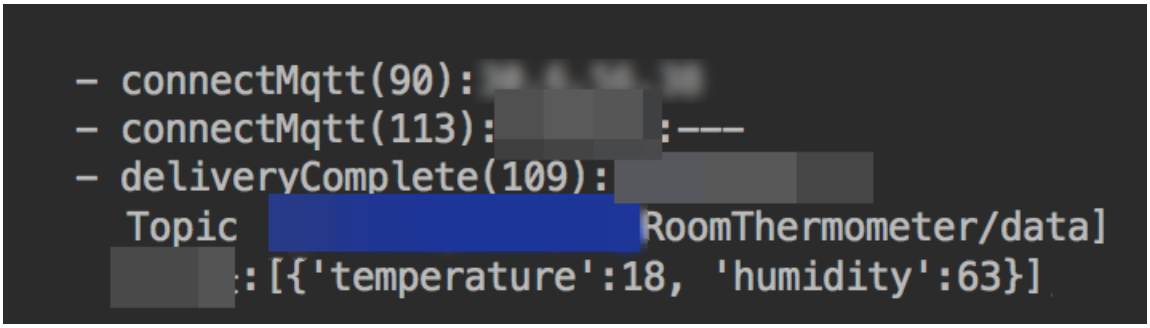
```
const   mqtt   =   require (' aliyun - iot - mqtt ');
const   client   =   mqtt . getAliyunI   otMqttClie   nt ({
```

```
 productKey : " Specify   the   productKey ",
 deviceName : " Specify   the   deviceName ",
 deviceSecr  et : " Specify   the   deviceSecr  et "
});
 const   topic  = ' Specify   the   topic   with   forwarding
 actions ';
 const   data  = {
 temperatur  e :  18 ,
 humidity :  63 ,
};
 client . publish ( topic ,  JSON . stringify ( data ));
```

5. DingTalk robot receives messages.

   a) The program sends the temperature and humidity data.

```
$  npm   install
$  node   demo . js
```



   b) DingTalk robot receives the data, and sends a message to the DintTalk group.

5 Set desired property values to control the light bulb status

You can set a desired property value for a device on the cloud. IoT Platform caches the value and pushes the value to the device when it is online .

Background information

A light bulb does not have storage capability. If you want to control the light bulb from IoT Platform, the light bulb must stay online always. However, the light bulb is rarely online.

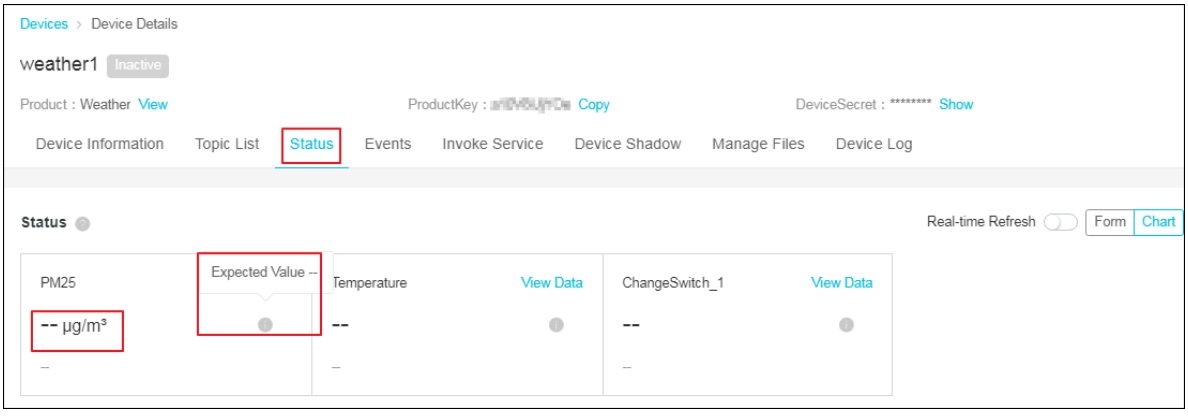
To resolve this issue, IoT Platform allows you to set desired property values.

When the light bulb is connected to IoT Platform, it reads the desired property value stored in IoT Platform. The device then updates the actual property based on the desired value and reports the updated property to IoT Platform. The reported property value will be stored as the latest property value.

Create a product and a device in the IoT Platform console

1. Log on to the IoT Platform console.
2. Create a product named Street Lamp and a light bulb device named Lamp.
3. Go to the Device Details page of the device, and click the Status tab.

   For a newly created device, the latest value and the desired value of a property are both "null", and the desired value version is 0.



4. Record the device certificate information, including ProductKey, DeviceName, and DeviceSecret, and the region information (RegionId).

Develop on the cloud

You call the related APIs to obtain the latest property values of a device and set desired property values to control the device.

For more information, see API reference (cloud). This example uses the Java SDK (cloud).

- Call `QueryDeviceDesiredProperty` to query the desired property values of the device.

```
 DefaultPro  file   profile  =  DefaultPro  file . getProfile (
        "< your - region - id >", // Your   region   ID
        "< your - access - key - id >", // AccessKey   ID   of
 your   Alibaba   Cloud   account
        "< your - access - key - secret >"); // AccessKey   Secret
 of   your   Alibaba   Cloud   account
 IAcsClient   client  =  new   DefaultAcs  Client ( profile );

// Set   parameters   to   call   the   API
 QueryDevic  eDesiredPr  opertyRequ  est   request  =  new
 QueryDevic  eDesiredPr  opertyRequ  est ();
 request . setProduct  Key ( productKey );
 request . setDeviceN  ame ( deviceName );
// The   list   of   property   identifier  s   that   you   want
   to   query   If   you   do   not   specify   the   property
 identifier   list ,  the   desired   values   of   all   properties
   except   the   read - only   properties   will   be   queried .
 request . setIdentif  iers ( Arrays . asList (" LightStatu  s "));

// Initiate   the   request   and   handle   the   response   or
 exception
 QueryDevic  eDesiredPr  opertyResp  onse   response ;
 try  {
     response  =  client . getAcsResp  onse ( request );

     QueryDevic  eDesiredPr  opertyResp  onse . Data   data  =
 response . getData ();
     for  ( QueryDevic  eDesiredPr  opertyResp  onse . Data .
 DesiredPro  pertyInfo   propInfo  :
             data . getList ()) {
         // Return   the   desired   value   and   version   of
 each   property ,  such   as   property = LightStatu  s   desired =
 1   version = 5
         System . out . println (" property =" +  propInfo .
 getIdentif  ier ());
         System . out . println (" desired =" +  propInfo . getValue
 ());
         System . out . println (" version =" +  propInfo .
 getBizVers  ion ());
    }
}  catch  ( ServerExce  ption   e ) {
     e . printStack  Trace ();
}  catch  ( ClientExce  ption   e ) {
     e . printStack  Trace ();
```

```
}
```

- Call `SetDeviceDesiredProperty` to set desired property values for a device.

```
DefaultPro file profile = DefaultPro file . getProfile (
        "< your - region - id >", // Your region ID
        "< your - access - key - id >", // AccessKey ID of
your Alibaba Cloud account
        "< your - access - key - secret >"); // AccessKey Secret
of your Alibaba Cloud account
IAcsClient client = new DefaultAcs Client ( profile );

// Set parameters to call the API
SetDeviceD esiredProp ertyReques t request = new
SetDeviceD esiredProp ertyReques t ();
request . setProduct Key ( productKey );
request . setDeviceN ame ( deviceName );
// The identifier s and values of the properties that
  you want to set .
request . setItems ("{\" LightStatu s \": 1 }");
// You can set the version of a desired property
value if other RAM users also set desired values
for the property for distinguis hment . Typically , you
  can choose not to specify the version .
request . setVersion s ("{}");

// Initiate the request and handle the response or
exception .
SetDeviceD esiredProp ertyRespon se response ;
try {
    response = client . getAcsResp onse ( request );

    SetDeviceD esiredProp ertyRespon se . Data data =
response . getData ();
   // If the operation is successful , the version of
  the desired property value is returned , such as {"
LightStatu s ": 5 }
    System . out . println ( data . getVersion s ());
} catch ( ServerExce ption e ) {
    e . printStack Trace ();
} catch ( ClientExce ption e ) {
    e . printStack Trace ();
}
```

Develop on the device

When the light bulb is connected to IoT Platform, it requests the desired property values from IoT Platform. If a desired property value changes while the light bulb is on, IoT Platform pushes the change in real time to the light bulb.

For more information, see Developer Guide (Devices). This example uses the Java SDK.

The last section of this topic provides a complete set of device demo code for your reference.

1. **Enter the device certificate and region information.**

```
/**
*  ProductKey ,  DeviceName ,  and   DeviceSecr  et
*/
 private   static   String   productKey  = "******";
 private   static   String   deviceName  = "********";
 private   static   String   deviceSecr  et  = "**************";
/**
*  MQTT   connection   informatio  n
*/
 private   static   String   productKey  = "******";
```

2. **Add the following methods. These methods can be used to change the actual property value of the light bulb and automatically report the changes to IoT Platform. The latest property value in IoT Platform will be replaced.**

```
/**
 *  When   the   device   handles   a   property   change ,  the
 methods   will   be   called   in   the   following   scenarios :
 *  Scenario   1   Pull   mode :  After   the   device   is
 connected   to   IoT   Platform ,  the   device   automatica  lly
 requests   the   latest   desired   property   value   from   IoT
 Platform .
 *  Scenario   2   Push   mode :  While   the   device   is   online
 ,  it   receives   the   desired   property   value   that   IoT
 Platform   pushes   to   the   topic   property . set .
 * @ param   identifier     The   property   identifier
 * @ param   value          The   desired   property   value
 * @ param   needReport    Indicates   whether   to   subscribe
 to   the    topic   property . post   to   send   the   property
 value   to   IoT   Platform
 *  In   scenario   2 ,  the   property   report   capability   is
   already   integrated   into   the   processing   function   and
 the   needReport   parameter   will   be   set   to  " false ".
 * @ return
 */
 private   boolean   handleProp  ertySet ( String   identifier ,
 ValueWrapp  er   value ,  boolean   needReport ) {
     ALog . d ( TAG , " The   device   handled   property   changes
 = [" +  identifier  + "],  value  = [" +  value  + "]");
    // Identify   whether   the   property   settings   are
 successful   according   to   the   response .  In   this   example
 ,  a   success   message   is   returned .
     boolean   success  =  true ;
     if  ( needReport ) {
         reportProp  erty ( identifier ,  value );
    }
     return   success ;
}

 private   void   reportProp  erty ( String   identifier ,
 ValueWrapp  er   value ){
     if  ( StringUtil  s . isEmptyStr  ing ( identifier ) ||  value
 ==  null ) {
         return ;
    }

     ALog . d ( TAG , " Reported   property   identity =" +
 identifier );
```

```
    Map < String ,  ValueWrapp  er >  reportData  =  new  HashMap
<>();
    reportData . put ( identifier ,  value );
    LinkKit . getInstanc  e (). getDeviceT  hing (). thingPrope
rtyPost ( reportData ,  new  IPublishRe  sourceList  ener () {

        public  void  onSuccess ( String  s ,  Object  o ) {
           // The  property  is  reported
            ALog . d ( TAG , " Report  success  onSuccess ()
called  with :  s  = [" +  s  + "],  o  = [" +  o  + "]");
       }

        public  void  onError ( String  s ,  AError  aError ) {
           // The  property  fails  to  be  reported
            ALog . d ( TAG , " Report  failure  onError ()
called  with :  s  = [" +  s  + "],  aError  = [" +  JSON .
toJSONStri  ng ( aError ) + "]");
       }
   });
}
```

3. **When the light bulb is online, if a desired property value is set for the light bulb in IoT Platform, the value will be pushed to the light bulb. The light bulb then processes the message and changes the status accordingly.**

   If the device is offline when the desired value is set, use the `connectNot ifyListene r` method to register a function to respond to service calls and property settings. For information about the Alink protocol, see Devices report properties.

   After the device receives an asynchronous message from IoT Platform, the device calls the `mCommonHan dler` method and then the `handleProp ertySet` method to update the property value.

```
/**
 *  Register  a  function  to  respond  to  service  calls
and  property  settings .
 *  When  IoT  Platform  calls  a  service  from  the
device ,  the  device  must  respond  to  the  call  and
return  a  response .
 */
public  void  connectNot  ifyListene  r () {
    List < Service >  serviceLis  t  =  LinkKit . getInstanc  e ().
getDeviceT  hing (). getService  s ();
    for  ( int  i  =  0 ;  serviceLis  t  ! =  null  &&  i  <
serviceLis  t . size ();  i ++) {
        Service  service  =  serviceLis  t . get ( i );
        LinkKit . getInstanc  e (). getDeviceT  hing ().
setService  Handler ( service . getIdentif  ier (),  mCommonHan
dler );
   }
}

 private  ITResReque  stHandler  mCommonHan  dler  =  new
ITResReque  stHandler () {
```

```
    public   void   onProcess ( String   serviceIde  ntifier
, Object   result ,  ITResRespo  nseCallbac  k   itResRespo
nseCallbac  k ) {
        ALog . d ( TAG , " onProcess ()  called   with :  s  = ["
+  serviceIde  ntifier  + "]," +
                "  o  = [" +  result  + "],  itResRespo  nseCallbac
k  = [" +  itResRespo  nseCallbac  k  + "]");
        ALog . d ( TAG , " Received   an   asynchrono  us   service
  call   from   IoT   Platform  " +  serviceIde  ntifier );
        try  {
            if  ( SERVICE_SE  T . equals ( serviceIde  ntifier )) {
                Map < String ,  ValueWrapp  er >  data  = ( Map <
String ,  ValueWrapp  er >)(( InputParam  s ) result ). getData ();
                ALog . d ( TAG , " Received   asynchrono  us
downstream   data  " +  data );
              // Set   the   property   value   of   the   device
  and   then   report   the   property   value   to   IoT
Platform .
                boolean   isSetPrope  rtySuccess  =
                        handleProp  ertySet (" LightStatu  s ",
data . get (" LightStatu  s "),  false );
                if  ( isSetPrope  rtySuccess ) {
                    if  ( result   instanceof   InputParam  s ) {
                       // Return   a   response   to   IoT
Platform
                        itResRespo  nseCallbac  k . onComplete (
serviceIde  ntifier ,  null ,  null );
                    }  else  {
                        itResRespo  nseCallbac  k . onComplete (
serviceIde  ntifier ,  null ,  null );
                    }
                }  else  {
                    AError   error  =  new   AError ();
                    error . setCode ( 100 );
                    error . setMsg (" setPropert  yFailed .");
                    itResRespo  nseCallbac  k . onComplete (
serviceIde  ntifier ,  new   ErrorInfo ( error ),  null );
                }
            }  else   if  ( SERVICE_GE  T . equals ( serviceIde
ntifier )) {
            }  else  {
                // The   device   operation   varies   by   service
.
                ALog . d ( TAG , " The   returned   values
correspond  ing   to   the   service .");
                OutputPara  ms   outputPara  ms  =  new
OutputPara  ms ();
                //  outputPara  ms . put (" op ",  new   ValueWrapp
er . IntValueWr  apper ( 20 ));
                itResRespo  nseCallbac  k . onComplete (
serviceIde  ntifier ,  null ,  outputPara  ms );
            }
        }  catch  ( Exception   e ) {
            e . printStack  Trace ();
            ALog . d ( TAG , " The   data   returned   from   IoT
Platform   is   incorrectl  y   formatted ");
        }
    }

    public   void   onSuccess ( Object   o ,  OutputPara  ms
outputPara  ms ) {
        ALog . d ( TAG , " onSuccess ()  called   with :  o  = ["
+  o  + "],  outputPara  ms  = [" +  outputPara  ms  + "]");
```

```
        ALog . d ( TAG , " The   service   has   been   registered
");
    }

    public   void   onFail ( Object   o ,  ErrorInfo   errorInfo )
{
        ALog . d ( TAG , " onFail ()  called   with :  o  = [" +  o
 + "],  errorInfo  = [" +  errorInfo  + "]");
        ALog . d ( TAG , " Service   registrati  on   failed .");
    }
};
```

4. **When the light bulb is offline, if a desired property value is set for the light bulb in IoT Platform, this value will be stored in IoT Platform.**

   After the light bulb comes online again, it will request for the desired property value from IoT Platform and call the `handleProp ertySet` method to update the property value.

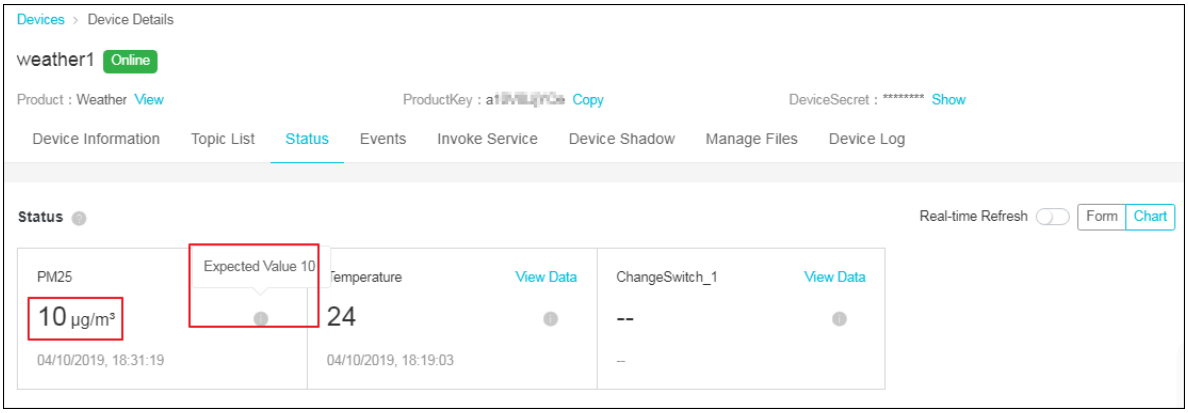
```
LinkKit . getInstanc  e (). init ( params ,  new   ILinkKitCo
nnectListe  ner () {
    public   void   onError ( AError   aError ) {
        ALog . e ( TAG , " Init   Error   error =" +  aError );
    }
    public   void   onInitDone ( InitResult   initResult ) {
        ALog . i ( TAG , " onInitDone   result =" +  initResult );

        connectNot  ifyListene  r ();

       // Obtain   the   latest   desired   property   value
        getDesired  Property ( deviceInfo ,  Arrays . asList ("
LightStatu  s "),  new   IConnectSe  ndListener () {
            public   void   onResponse ( ARequest   aRequest ,
AResponse   aResponse ) {
                if ( aRequest   instanceof   MqttPublis  hRequest
&&  aResponse . data  ! =  null ) {
                    JSONObject   jsonObject  =  JSONObject .
parseObjec  t ( aResponse . data . toString ());
                    ALog . i ( TAG , " onResponse   result =" +
jsonObject );
                    JSONObject   dataObj  =  jsonObject .
getJSONObj  ect (" data ");
                    if  ( dataObj  ! =  null ) {
                        if  ( dataObj . getJSONObj  ect ("
LightStatu  s ") ==  null ) {
                            // No   desired   value   is   set
                        }  else  {
                            Integer   value  =  dataObj .
getJSONObj  ect (" LightStatu  s "). getInteger (" value ");
                            handleProp  ertySet (" LightStatu  s
",  new   ValueWrapp  er . IntValueWr  apper ( value ),  true );
                        }
                    }
                }
            }
            public   void   onFailure ( ARequest   aRequest ,
AError   aError ) {
                ALog . d ( TAG , " onFailure ()  called   with :
aRequest  = [" +  aRequest  + "],  aError  = [" +  aError  + "]");
            }
```

```
        });
    }
});

 private   void   getDesired  Property ( BaseInfo   info ,  List <
 String >  properties ,  IConnectSe  ndListener   listener ) {
     ALog . d ( TAG , " getDesired  Property ()  called   with :
 info  = [" +  info  + "],  listener  = [" +  listener  + "]");
     if ( info  ! =  null  && !  StringUtil  s . isEmptyStr  ing (
 info . productKey ) && !  StringUtil  s . isEmptyStr  ing ( info .
 deviceName )) {
         MqttPublis  hRequest   request  =  new   MqttPublis
 hRequest ();
         request . topic  =  DESIRED_PR  OPERTY_GET . replace ("{
 productKey }",  info . productKey ). replace ("{ deviceName }",
 info . deviceName );
         request . replyTopic  =  DESIRED_PR  OPERTY_GET  _REPLY
 . replace ("{ productKey }",  info . productKey ). replace ("{
 deviceName }",  info . deviceName );
         request . isRPC  =  true ;
         RequestMod  el < List < String >>  model  =  new
 RequestMod  el <>();
         model . id  =  String . valueOf ( IDGenerate  rUtils .
 getId ());
         model . method  =  METHOD_GET  _DESIRED_P  ROPERTY ;
         model . params  =  properties ;
         model . version  = " 1 . 0 ";
         request . payloadObj  =  model . toString ();
         ALog . d ( TAG , " getDesired  Property :  payloadObj =" +
 request . payloadObj );
         ConnectSDK . getInstanc  e (). send ( request ,  listener
 );
    }  else  {
         ALog . w ( TAG , " getDesired  Property   failed ,
 baseInfo   Empty .");
         if ( listener  ! =  null ) {
             AError   error  =  new   AError ();
             error . setMsg (" BaseInfoEm  pty .");
             listener . onFailure ( null ,  error );
         }
    }
}
```

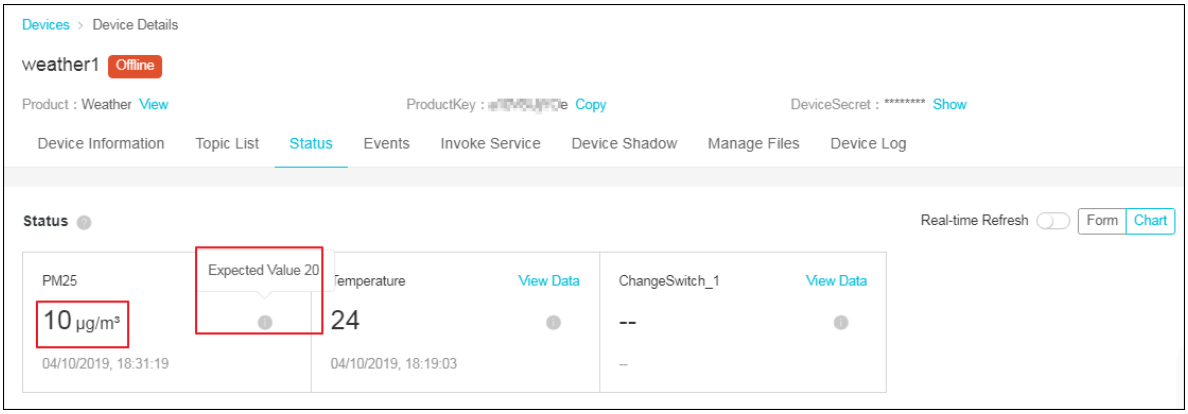
Verify the configuration

Verify that you can change the light bulb status by setting the desired value in IoT Platform regardless of whether the light bulb is online or not.

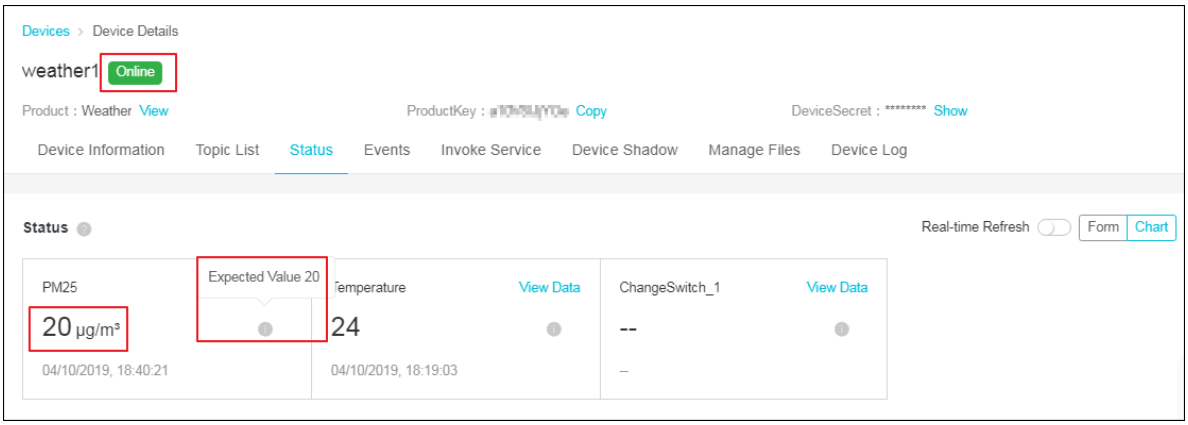
- When the light bulb is online, you change the light bulb status in IoT Platform and the light bulb responds to the change in real time.



- When the light bulb is offline, you change the light bulb status in IoT Platform. As a result, the desired value and the latest property value displayed in IoT Platform console are different.



- After the light bulb comes online again, the light bulb requests the desired property value. The latest property value will be immediately synchronized from the desired value.



Conslusion: To change the light bulb status, you can simply modify the desired property value. You can use the desired value to control the status of the light bulb that has no storage capability.

SDK demo code

```
package   com . aliyun . alink . devicesdk . demo ;

import   com . alibaba . fastjson . JSON ;
import   com . alibaba . fastjson . JSONObject ;
import   com . aliyun . alink . apiclient . utils . StringUtil  s ;
import   com . aliyun . alink . dm . api . BaseInfo ;
import   com . aliyun . alink . dm . api . DeviceInfo ;
import   com . aliyun . alink . dm . api . InitResult ;
import   com . aliyun . alink . dm . model . RequestMod  el ;
import   com . aliyun . alink . dm . utils . IDGenerate  rUtils ;
import   com . aliyun . alink . linkkit . api . ILinkKitCo
nnectListe  ner ;
import   com . aliyun . alink . linkkit . api . IoTMqttCli  entConfig
;
import   com . aliyun . alink . linkkit . api . LinkKit ;
import   com . aliyun . alink . linkkit . api . LinkKitIni  tParams ;
import   com . aliyun . alink . linksdk . cmp . api . ConnectSDK ;
import   com . aliyun . alink . linksdk . cmp . connect . channel .
MqttPublis  hRequest ;
import   com . aliyun . alink . linksdk . cmp . core . base .
ARequest ;
import   com . aliyun . alink . linksdk . cmp . core . base .
AResponse ;
import   com . aliyun . alink . linksdk . cmp . core . listener .
IConnectSe  ndListener ;
import   com . aliyun . alink . linksdk . tmp . api . InputParam  s ;
import   com . aliyun . alink . linksdk . tmp . api . OutputPara  ms
;
import   com . aliyun . alink . linksdk . tmp . device . payload .
ValueWrapp  er ;
import   com . aliyun . alink . linksdk . tmp . devicemode  l .
Service ;
import   com . aliyun . alink . linksdk . tmp . listener .
IPublishRe  sourceList  ener ;
import   com . aliyun . alink . linksdk . tmp . listener .
ITResReque  stHandler ;
import   com . aliyun . alink . linksdk . tmp . listener .
ITResRespo  nseCallbac  k ;
import   com . aliyun . alink . linksdk . tmp . utils . ErrorInfo ;
import   com . aliyun . alink . linksdk . tools . AError ;
import   com . aliyun . alink . linksdk . tools . ALog ;

import   java . util . Arrays ;
import   java . util . HashMap ;
import   java . util . List ;
import   java . util . Map ;

public   class   LampDemo  {
    private   static   final   String   TAG  = " LampDemo ";

    private   final   static   String   SERVICE_SE  T  = " set ";
    private   final   static   String   SERVICE_GE  T  = " get ";

    public   static   String   DESIRED_PR  OPERTY_GET  = "/ sys /{
productKey }/{ deviceName }/ thing / property / desired / get ";
```

```
    public   static   String   DESIRED_PR  OPERTY_GET  _REPLY  =
"/ sys /{ productKey }/{ deviceName }/ thing / property / desired /
get_reply ";
    public   static   String   METHOD_GET  _DESIRED_P  ROPERTY  = "
thing . property . desired . get ";

    public   static   void   main ( String []  args ) {
       /**
        *  ProductKey ,  DeviceName ,  and   DeviceSecr  et
        */
        String   productKey  = "****";
        String   deviceName  = " Lamp ";
        String   deviceSecr  et  = "****";
       /**
        *  MQTT   connection   informatio  n
        */
        String   regionId  = " cn - shanghai ";

        LampDemo   manager  =  new   LampDemo ();

        DeviceInfo   deviceInfo  =  new   DeviceInfo ();
        deviceInfo . productKey  =  productKey ;
        deviceInfo . deviceName  =  deviceName ;
        deviceInfo . deviceSecr  et  =  deviceSecr  et ;

        manager . init ( deviceInfo ,  regionId );
   }

    public   void   init ( final   DeviceInfo   deviceInfo ,  String
  region ) {
        LinkKitIni  tParams   params  =  new   LinkKitIni  tParams
();
       /**
        *  Set   the   MQTT   initializa  tion   parameters
        */
        IoTMqttCli  entConfig   config  =  new   IoTMqttCli
entConfig ();
        config . productKey  =  deviceInfo . productKey ;
        config . deviceName  =  deviceInfo . deviceName ;
        config . deviceSecr  et  =  deviceInfo . deviceSecr  et ;
        config . channelHos  t  =  deviceInfo . productKey  + ". iot
- as - mqtt ." +  region  + ". aliyuncs . com : 1883 ";
       /**
        *  Indicates   whether   to   receive   offline   messages
        *  The   cleanSessi  on   field   correspond  ing   to   the
  MQTT   connection
        */
        config . receiveOff  lineMsg  =  false ;
        params . mqttClient  Config  =  config ;

       /**
        *  Pass   in   the   device   certificat  e   informatio  n
for   initializa  tion
        */
        params . deviceInfo  =  deviceInfo ;

        LinkKit . getInstanc  e (). init ( params ,  new
ILinkKitCo  nnectListe  ner () {
            public   void   onError ( AError   aError ) {
                ALog . e ( TAG , " Init   Error   error =" +  aError
);
           }
            public   void   onInitDone ( InitResult   initResult ) {
```

```
                ALog . i ( TAG , " onInitDone   result =" +
initResult );

                connectNot  ifyListene  r ();

               // Obtain   the   latest   desired   property   value
  from   IoT   Platform
                getDesired  Property ( deviceInfo ,  Arrays . asList
(" LightStatu  s "),  new   IConnectSe  ndListener () {
                    public   void   onResponse ( ARequest   aRequest
,  AResponse   aResponse ) {
                        if ( aRequest   instanceof   MqttPublis
hRequest  &&  aResponse . data  ! =  null ) {
                            JSONObject   jsonObject  =  JSONObject .
parseObjec  t ( aResponse . data . toString ());
                            ALog . i ( TAG , " onResponse   result ="
+  jsonObject );
                            JSONObject   dataObj  =  jsonObject .
getJSONObj  ect (" data ");
                            if  ( dataObj  ! =  null ) {
                                if  ( dataObj . getJSONObj  ect ("
LightStatu  s ") ==  null ) {
                                    // No   desired   value   is
set
                                } else  {
                                    Integer   value  =  dataObj .
getJSONObj  ect (" LightStatu  s "). getInteger (" value ");
                                    handleProp  ertySet ("
LightStatu  s ",  new   ValueWrapp  er . IntValueWr  apper ( value
),  true );
                                }
                            }
                        }
                    }
                    public   void   onFailure ( ARequest   aRequest
,  AError   aError ) {
                        ALog . d ( TAG , " onFailure ()  called
with :  aRequest  = [" +  aRequest  + "],  aError  = [" +  aError  +
"]");
                    }
                });
            }
        });
    }

    /**
     *  When   the   device   handles   a   property   change ,  the
methods   will   be   called   in   the   following   scenarios :
     *  Scenario   1 .  Pull   mode :  After   the   device   is
connected   to   IoT   Platform   again ,  the   device   automatica
lly   requests   the   latest   desired   property   value   from
IoT   Platform .
     *  Scenario   2 .  Push   mode :  While   the   device   is
online ,  it   receives   the   desired   property   value   that
IoT   Platform   pushes   to   the   property . set   topic .
     * @ param   identifier    The   property   identifier
     * @ param   value         The   desired   property   value
     * @ param   needReport    Indicates   whether   to   subscribe
  to   the   property . post   topic   to   send   the   property
value   to   IoT   Platform
     *  In   scenario   2 ,  the   property   report   capability   is
  already   integrated   into   the   processing   function   and
the   needReport   parameter   will   be   set   to  " false ".
     * @ return
```

```
    */
    private   boolean   handleProp  ertySet ( String   identifier ,
ValueWrapp  er   value ,  boolean   needReport ) {
        ALog . d ( TAG , " The   device   handled   property
changes = [" +  identifier  + "],  value  = [" +  value  + "]");
       // Identify   whether   the   property   settings   are
successful   according   to   the   response .  In   this   example
,  a   success   message   is   returned .
        boolean   success  =  true ;
        if  ( needReport ) {
            reportProp  erty ( identifier ,  value );
       }
        return   success ;
   }

    private   void   reportProp  erty ( String   identifier ,
ValueWrapp  er   value ){
        if  ( StringUtil  s . isEmptyStr  ing ( identifier ) ||
value  ==  null ) {
            return ;
       }

        ALog . d ( TAG , " Reported   property   identity =" +
identifier );

        Map < String ,  ValueWrapp  er >  reportData  =  new
HashMap <>();
        reportData . put ( identifier ,  value );
        LinkKit . getInstanc  e (). getDeviceT  hing (). thingPrope
rtyPost ( reportData ,  new   IPublishRe  sourceList  ener () {

            public   void   onSuccess ( String   s ,  Object   o ) {
               // The   property   is   reported
                ALog . d ( TAG , " Report   success   onSuccess ()
called   with :  s  = [" +  s  + "],  o  = [" +  o  + "]");
            }

            public   void   onError ( String   s ,  AError   aError )
{
               // The   property   fails   to   be   reported
                ALog . d ( TAG , " Report   failure   onError
() called   with :  s  = [" +  s  + "],  aError  = [" +  JSON .
toJSONStri  ng ( aError ) + "]");
            }
       });
   }

   /**
    *  Register   a   function   to   respond   to   service   calls
  and   property   settings .
    *  When   IoT   Platform   calls   a   service   from   the
device ,  the   device   must   respond   to   the   call   and
return   a   response .
    */
    public   void   connectNot  ifyListene  r () {
        List < Service >  serviceLis  t  =  LinkKit . getInstanc  e
(). getDeviceT  hing (). getService  s ();
        for  ( int   i  =  0 ;  serviceLis  t  ! =  null  &&  i  <
serviceLis  t . size ();  i ++) {
            Service   service  =  serviceLis  t . get ( i );
            LinkKit . getInstanc  e (). getDeviceT  hing ().
setService  Handler ( service . getIdentif  ier (),  mCommonHan  dler
);
       }
```

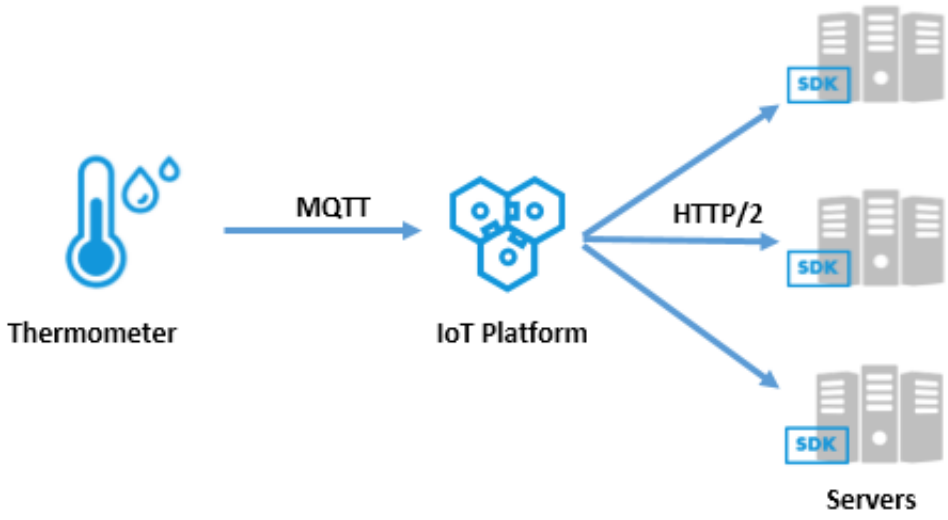
```
    }

    private ITResReque stHandler mCommonHan dler = new
ITResReque stHandler () {
        public void onProcess ( String serviceIde ntifier
, Object result , ITResRespo nseCallbac k itResRespo
nseCallbac k ) {
            ALog . d ( TAG , " onProcess () called with : s = ["
+ serviceIde ntifier + "]," +
                    " o = [" + result + "], itResRespo
nseCallbac k = [" + itResRespo nseCallbac k + "]");
            ALog . d ( TAG , " Received an asynchrono us
service call from IoT Platform " + serviceIde ntifier );
            try {
                if ( SERVICE_SE T . equals ( serviceIde ntifier ))
{
                    Map < String , ValueWrapp er > data = ( Map <
String , ValueWrapp er >)(( InputParam s ) result ). getData ();
                    ALog . d ( TAG , " Received asynchrono us
downstream data " + data );
                    // Set the property of the device and
  then report the property value to IoT Platform .
                    boolean isSetPrope rtySuccess =
                            handleProp ertySet (" LightStatu s ",
data . get (" LightStatu s "), false );
                    if ( isSetPrope rtySuccess ) {
                        if ( result instanceof InputParam s ) {
                           // Return a response to IoT
Platform
                            itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , null );
                        } else {
                            itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , null );
                        }
                    } else {
                        AError error = new AError ();
                        error . setCode ( 100 );
                        error . setMsg (" setPropert yFailed .");
                        itResRespo nseCallbac k . onComplete (
serviceIde ntifier , new ErrorInfo ( error ), null );
                    }
                } else if ( SERVICE_GE T . equals ( serviceIde
ntifier )) {
                } else {
                    // The device operation varies by
service .
                    ALog . d ( TAG , " The returned values
correspond ing to the service .");
                    OutputPara ms outputPara ms = new
OutputPara ms ();
                    // outputPara ms . put (" op ", new
ValueWrapp er . IntValueWr apper ( 20 ));
                    itResRespo nseCallbac k . onComplete (
serviceIde ntifier , null , outputPara ms );
                }
            } catch ( Exception e ) {
                e . printStack Trace ();
                ALog . d ( TAG , " The data returned from IoT
  Platform is incorrectl y formatted ");
            }
        }
```

```
        public   void   onSuccess ( Object   o ,  OutputPara  ms
 outputPara  ms ) {
            ALog . d ( TAG , " onSuccess ()  called   with :  o  = ["
 +  o  + "],  outputPara  ms  = [" +  outputPara  ms  + "]");
            ALog . d ( TAG , " The   service   was   registered ");
        }

        public   void   onFail ( Object   o ,  ErrorInfo   errorInfo
 ) {
            ALog . d ( TAG , " onFail ()  called   with :  o  = [" +
 o  + "],  errorInfo  = [" +  errorInfo  + "]");
            ALog . d ( TAG , " Service   registrati  on   failed .");
        }
    };

    private   void   getDesired  Property ( BaseInfo   info ,  List <
 String >  properties ,  IConnectSe  ndListener   listener ) {
        ALog . d ( TAG , " getDesired  Property ()  called   with :
 info  = [" +  info  + "],  listener  = [" +  listener  + "]");
        if ( info  ! =  null  && !  StringUtil  s . isEmptyStr  ing
 ( info . productKey ) && !  StringUtil  s . isEmptyStr  ing ( info .
 deviceName )) {
            MqttPublis  hRequest   request  =  new   MqttPublis
 hRequest ();
            request . topic  =  DESIRED_PR  OPERTY_GET . replace ("{
 productKey }",  info . productKey ). replace ("{ deviceName }",  info
 . deviceName );
            request . replyTopic  =  DESIRED_PR  OPERTY_GET  _REPLY
 . replace ("{ productKey }",  info . productKey ). replace ("{
 deviceName }",  info . deviceName );
            request . isRPC  =  true ;
            RequestMod  el < List < String >>  model  =  new
 RequestMod  el <>();
            model . id  =  String . valueOf ( IDGenerate  rUtils .
 getId ());
            model . method  =  METHOD_GET  _DESIRED_P  ROPERTY ;
            model . params  =  properties ;
            model . version  = " 1 . 0 ";
            request . payloadObj  =  model . toString ();
            ALog . d ( TAG , " getDesired  Property :  payloadObj ="
 +  request . payloadObj );
            ConnectSDK . getInstanc  e (). send ( request ,  listener
 );
        }  else  {
            ALog . w ( TAG , " getDesired  Property   failed ,
 baseInfo   Empty .");
            if ( listener  ! =  null ) {
                AError   error  =  new   AError ();
                error . setMsg (" BaseInfoEm  pty .");
                listener . onFailure ( null ,  error );
            }
        }
    }
}
```

6 Configure service subscription

In some IoT scenarios, device status information and device collected data are expected to be directly received by business servers rather than be forwarded by cloud services. In such a case, you can use the service subscription function. You can push device messages to the business servers directly through the HTTP/2 channel, so that users can consume data based on their own business scenarios.

This topic describes how to use service subscription for data forwarding by using a hygrothermograph in an example to construct a client.



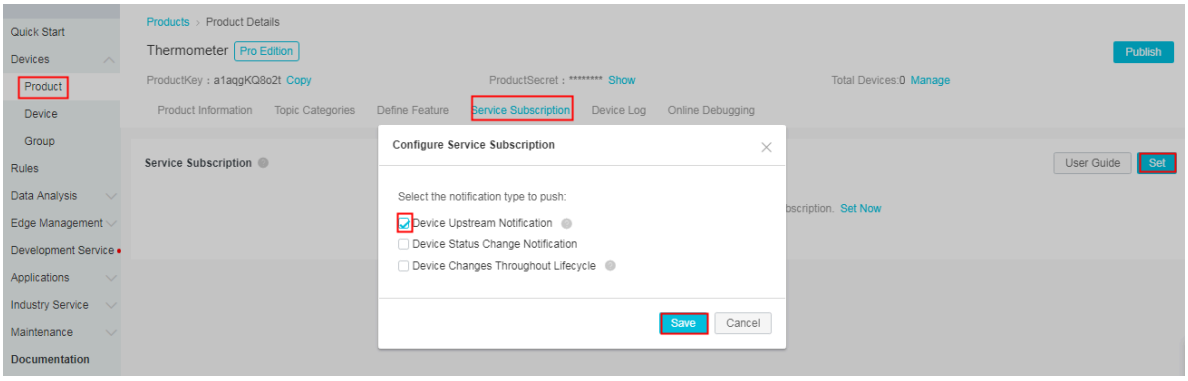
Prerequisites

Before you begin, create a product and a device in the IoT Platform console and connect the device to IoT Platform. For more information, see Quick Start.

Configure service subscription

1. Log on to the IoT Platform console, and choose Devices > Product from the left-side navigation pane.
2. In the product list, locate the product for which you want to configure service subscription and click View. The Product Details page appears.
3. Choose Service Subscription > Set.

4. In the Configure Service Subscription dialog box, select Device Upstream Notification, and click Save.



Use the service subscription SDK

Note:

IoT Platform supports Java 8 SDK.

1. Add the Java SDK dependencies as follows:

```
<! -- Aliyun core -->
< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - core </ artifactId >
    < version > 3 . 7 . 1 </ version >
</ dependency >

<! -- iot message client -->
< dependency >
    < groupId > com . aliyun . openservic es </ groupId >
    < artifactId > iot - client - message </ artifactId >
    < version > 1 . 1 . 2 </ version >
</ dependency >
```

2. Connect the SDK to IoT Platform using your Alibaba Cloud account information for identity authentication.

```
import java . net . UnknownHos tException ;
import java . util . concurrent . ExecutionE xception ;

import com . aliyun . openservic es . iot . api . Profile ;
import com . aliyun . openservic es . iot . api . message .
MessageCli entFactory ;
import com . aliyun . openservic es . iot . api . message . api
. MessageCli ent ;
import com . aliyun . openservic es . iot . api . message .
callback . MessageCal lback ;
import com . aliyun . openservic es . iot . api . message .
entity . Message ;

public class H2Client {

    public static void main ( String [] args ) throws
UnknownHos tException ,
```

```
                                    ExecutionE  xception ,  Interrupte
dException  {
       //  Identity   informatio  n
        String   accessKey  = " Your   AccessKey   ID ";
        String   accessSecr  et  = " Your   AccessKey   Secret >";
        String   regionId  = " cn - shanghai ";
        String   uid  = " Your   Alibaba   Cloud   UID ";
        String   endPoint  = " https ://" +  uid  + ". iot - as -
http2 ." +  regionId  + ". aliyuncs . com ";
       // Configure   connection   parameters
        Profile   profile  =  Profile . getAccessK  eyProfile (
endPoint ,  regionId ,  accessKey ,  accessSecr  et );

       // Construct   a   client
        MessageCli  ent   client  =  MessageCli  entFactory .
messageCli  ent ( profile );
       // Receive   data
        client . connect ( messageTok  en  -> {
            Message   m  =  messageTok  en . getMessage ();
            System . out . println ("\ ntopic ="+ m . getTopic ());
            System . out . println (" payload =" +  new   String (
m . getPayload ()));
            System . out . println (" generateTi  me =" +  m .
getGenerat  eTime ());
          //  CommitSucc  ess   indicates   that   the   current
message   has   been   consumed   and   will   be   deleted   by
IoT   Platform .  If   the   message   is   not   marked   with
CommitSucc  ess ,  the   message   will   be   retained   until
it   expires .
            return   MessageCal  lback . Action . CommitSucc  ess ;
       });
   }
}
```

3. **The device reports the message to IoT Platform and the client receives the device message.**

Data forwarding example

- **Sample data of devices without TSL models**

  The device comes online, publishes a temperature and humidity message, and goes offline.

```
{
    " temperatur  e ":  27 ,
    " humidity ":  26
}
```

  The business server receives the following messages:

```
#  Device   connection   message
topic =/ as / mqtt / status / a1wLOMQOK4  K / temperatur  e -
Monitor
payload ={
   " lastTime ": " 2018 - 09 - 04   14 : 59 : 59 . 782 ",
   " utcLastTim  e ": " 2018 - 09 - 04T06 : 59 : 59 . 782Z ",
   " clientIp ": " 42 . 120 . 75 . 152 ",
   " utcTime ": " 2018 - 09 - 04T06 : 59 : 59 . 794Z ",
   " time ": " 2018 - 09 - 04   14 : 59 : 59 . 794 ",
```

```
    " productKey ": " a1wLOMQOK4  K ",
    " deviceName ": " temperatur  e - Monitor ",
    " status ": " online "
}
 generateTi  me = 1536044399  797

#  Data   message   reported   by   the   device
 topic =/ a1wLOMQOK4  K / temperatur  e - Monitor / temperatur
 eData
 payload ={
    " temperatur  e ":  27 ,
    " humidity ":  26
}
 generateTi  me = 1536044404  762

#  Device   disconnect  ion   message
 topic =/ as / mqtt / status / a1wLOMQOK4  K / temperatur  e -
 Monitor
 payload ={
    " lastTime ": " 2018 - 09 - 04   15 : 00 : 09 . 761 ",
    " utcLastTim  e ": " 2018 - 09 - 04T07 : 00 : 09 . 761Z ",
    " utcTime ": " 2018 - 09 - 04T07 : 00 : 13 . 337Z ",
    " time ": " 2018 - 09 - 04   15 : 00 : 13 . 337 ",
    " productKey ": " a1wLOMQOK4  K ",
    " deviceName ": " temperatur  e - Monitor ",
    " status ": " offline "
}
 generateTi  me = 1536044413  339
```

Note:

The service subscription function is based on the UID of your Alibaba Cloud account. All messages of the products that are created by this account will be received by the HTTP/2 client. You can differentiate service consumption among business servers by the ProductKey and DeviceName information that is contained in the topic.

- **Sample data of devices using TSL models**

  The device comes online, publishes a temperature and humidity message, and goes offline.

```
{
    " id ":  1536045963  829 ,
    " params ":{
        " temperatur  e ":  20 ,
        " humidity ":  22
    },
    " method ": " thing . event . property . post "
}
```

The business server receives the following messages:

```
#  Device   connection   message
 topic =/ as / mqtt / status / a1gMuCoK4m  2 / nuwr5r9hf6  l
 payload ={
    " lastTime ": " 2018 - 09 - 04   15 : 25 : 53 . 771 ",
```

```
    " utcLastTim  e ": " 2018 - 09 - 04T07 : 25 : 53 . 771Z ",
    " clientIp ": " 42 . 120 . 75 . 152 ",
    " utcTime ": " 2018 - 09 - 04T07 : 25 : 53 . 787Z ",
    " time ": " 2018 - 09 - 04   15 : 25 : 53 . 787 ",
    " productKey ": " a1gMuCoK4m  2 ",
    " deviceName ": " nuwr5r9hf6  l ",
    " status ": " online "
}
 generateTi  me = 1536045953  787

#  Device - reported   data   message   that   is   parsed   based
 on   the   TSL   model
 topic =/ a1gMuCoK4m  2 / nuwr5r9hf6  l / thing / event / property /
 post
 payload ={
    " deviceType ": " None ",
    " iotId ": " hutrRN9iuB  vO5QOclH4f  0010885100 ",
    " productKey ": " a1gMuCoK4m  2 ",
    " gmtCreate ":  1536045958  765 ,
    " deviceName ": " nuwr5r9hf6  l ",
    " items ":{
        " temperatur  e ":{
            " value ":  20 ,
            " time ":  1536045958  770
        },
        " humidity ":{
            " value ":  22 ,
            " time ":  1536045958  770
        }
    }
}
 generateTi  me = 1536045958  770

#  Device   disconnect  ion   message
 topic =/ as / mqtt / status / a1gMuCoK4m  2 / nuwr5r9hf6  l
 payload ={
    " lastTime ": " 2018 - 09 - 04   15 : 26 : 03 . 749 ",
    " utcLastTim  e ": " 2018 - 09 - 04T07 : 26 : 03 . 749Z ",
    " utcTime ": " 2018 - 09 - 04T07 : 26 : 06 . 586Z ",
    " time ": " 2018 - 09 - 04   15 : 26 : 06 . 586 ",
    " productKey ": " a1gMuCoK4m  2 ",
    " deviceName ": " nuwr5r9hf6  l ",
    " status ": " offline "
}
 generateTi  me = 1536045966  588
```

Note:

The messages of devices using TSL are not in the original format. The data will be parsed based on TSL models.

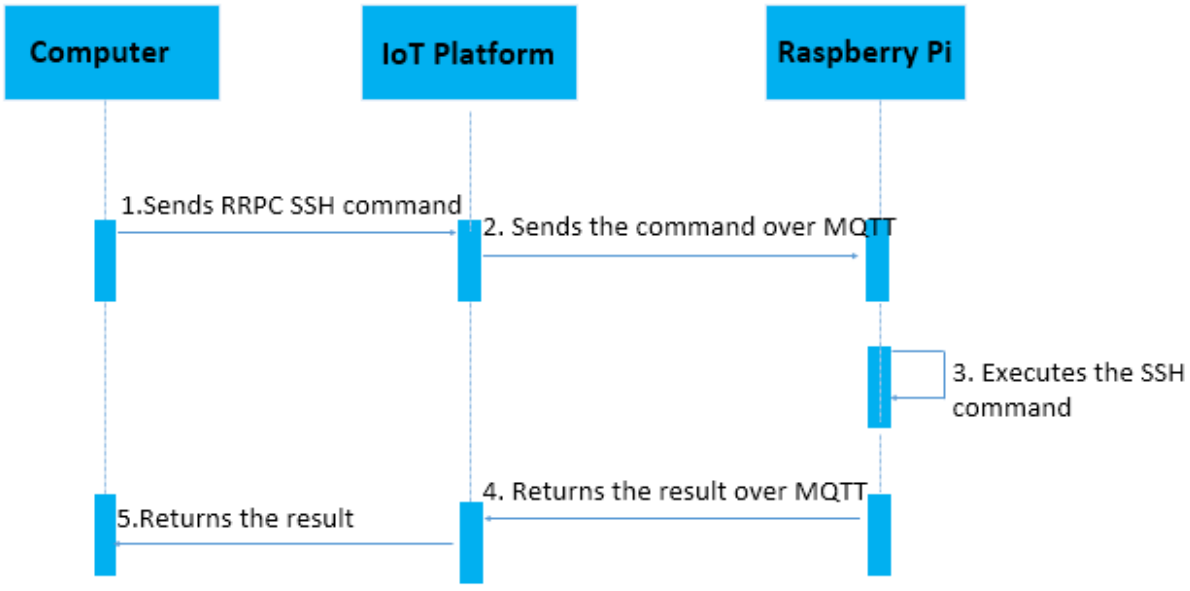
7 Control Raspberry Pi servers from IoT Platform

Alibaba Cloud IoT Platform allows you to remotely control Raspberry Pi servers without public IP addresses. This topic describes how to control Raspberry Pi servers from IoT Platform, and provides sample development code.

Background information

Assume that you use Raspberry Pi to build a server at your company or home to run some simple tasks, such as starting a script or downloading files. However, if the Raspberry Pi server does not have a public IP address, you cannot control the server when you are not in the company or at home. If you use other NAT traversal tools to contol the server, disconnection may occur frequently. To resolve these issues, you can combine the IoT Platform RRPC (remote synchronous process call) function with JSch to control the Raspberry Pi server from IoT Platform.

Process



Use the following process to control a Raspberry Pi server from IoT Platform:

- Call the IoT Platform RRPC method from your computer to send an SSH command.
- After receiving the SSH command, IoT Platform sends the SSH command to the Raspberry Pi server by using MQTT.
- The server runs the SSH command.

- The server encapsulates the command output into an RRPC response and reports the response to IoT Platform by using MQTT.
- IoT Platform returns the RRPC response to the computer.

> **Note:**
> The RRPC call timeout period is 5 seconds. If IoT Platform does not receive any reply from the Raspberry Pi server within 5 seconds, a timeout error will be returned. If the command you have sent takes a relatively long time to process, you can ignore the timeout error.

Download SDKs and demos

To remotely control Raspberry Pi servers, you must first develop the server SDK and device SDK.

- Install the IoT Platform server SDK on the computer. To develop the server, you can use the server Java SDK demo.
- Install the IoT Platform device SDK on the Raspberry Pi server. To develop the device, use the device Java SDK Demo.

The following sections provide examples about how to develop the server SDK and device SDK.

> **Note:**
> In the examples, only simple Linux commands are supported in the code, such as uname, touch, and mv. Complex commands such as file editing are not supported. To implement complex commands, you must write code as needed.

Develop the device SDK

After you have downloaded and installed the device SDK and SDK demo, add project dependencies and the following Java files.

The project can be exported as a JAR package and run on the Raspberry Pi server.

1. Add dependencies to the *pom.xml* file.

```
<! -- Device SDK -->
< dependency >
    < groupId > com . aliyun . alink . linksdk </ groupId >
    < artifactId > iot - linkkit - java </ artifactId >
    < version > 1 . 1 . 0 </ version >
    < scope > compile </ scope >
</ dependency >
< dependency >
```

```
    < groupId > com . google . code . gson </ groupId >
    < artifactId > gson </ artifactId >
    < version > 2 . 8 . 1 </ version >
    < scope > compile </ scope >
</ dependency >
< dependency >
    < groupId > com . alibaba </ groupId >
    < artifactId > fastjson </ artifactId >
    < version > 1 . 2 . 40 </ version >
    < scope > compile </ scope >
</ dependency >

<! --  SSH   client -->
<! --  https :// mvnreposit  ory . com / artifact / com . jcraft /
 jsch  -->
< dependency >
    < groupId > com . jcraft </ groupId >
    < artifactId > jsch </ artifactId >
    < version > 0 . 1 . 55 </ version >
</ dependency >
```

2. **Add the** *SSHShell . java* **file that is used to run SSH commands.**

```
public   class   SSHShell  {

    private   String   host ;

    private   String   username ;

    private   String   password ;

    private   int   port ;

    private   Vector < String >  stdout ;

    public   SSHShell ( final   String   ipAddress ,  final
String   username ,  final   String   password ,  final   int
port ) {
        this . host  =  ipAddress ;
        this . username  =  username ;
        this . password  =  password ;
        this . port  =  port ;
        this . stdout  =  new   Vector < String >();
   }

    public   int   execute ( final   String   command ) {

        System . out . println (" ssh   command : " +  command );

        int   returnCode  =  0 ;
        JSch   jsch  =  new   JSch ();
        SSHUserInf  o   userInfo  =  new   SSHUserInf  o ();

        try  {
            Session   session  =  jsch . getSession ( username ,
host ,  port );
            session . setPasswor  d ( password );
            session . setUserInf  o ( userInfo );
            session . connect ();

            Channel   channel  =  session . openChanne  l (" exec
");
           (( ChannelExe  c )  channel ). setCommand ( command );
```

```
            channel . setInputSt  ream ( null );
            BufferedRe  ader   input  =  new   BufferedRe  ader (
new   InputStrea  mReader ( channel . getInputSt  ream ()));

            channel . connect ();

            String   line  =  null ;
            while  (( line  =  input . readLine ()) ! =  null ) {
                stdout . add ( line );
            }
            input . close ();

            if  ( channel . isClosed ()) {
                returnCode  =  channel . getExitSta  tus ();
            }

            channel . disconnect ();
            session . disconnect ();
        }  catch  ( JSchExcept  ion   e ) {
            e . printStack  Trace ();
        }  catch  ( Exception   e ) {
            e . printStack  Trace ();
        }

        return   returnCode ;
    }

    public   Vector < String >  getStdout () {
        return   stdout ;
    }

}
```

3. **Add the** *SSHUserInf o . java* **file that is used to verify the SSH account and password.**

```
public   class   SSHUserInf  o   implements   UserInfo  {

    @ Override
    public   String   getPassphr  ase () {
        return   null ;
    }

    @ Override
    public   String   getPasswor  d () {
        return   null ;
    }

    @ Override
    public   boolean   promptPass  phrase ( final   String   arg0 )
{
        return   false ;
    }

    @ Override
    public   boolean   promptPass  word ( final   String   arg0 ) {
        return   false ;
    }

    @ Override
    public   boolean   promptYesN  o ( final   String   arg0 ) {
```

```
        if  ( arg0 . contains (" The   authentici  ty   of   host
")) {
            return   true ;
        }
        return   false ;
    }

    @ Override
     public   void   showMessag  e ( final   String   arg0 ) {
    }

}
```

4. Add the *Device . java* file that is used to establish an MQTT connection.

```
public   class   Device  {

    /**
     *  Establish   a   connection
     *
     * @ param   productKey   The   product   key
     * @ param   deviceName   The   device   name
     * @ param   deviceSecr  et   The   device   secret
     * @ throws   Interrupte  dException
     */
     public   static   void   connect ( String   productKey
,  String   deviceName ,  String   deviceSecr  et )  throws
Interrupte  dException  {

        // Initializa  tion   parameters
         LinkKitIni  tParams   params  =  new   LinkKitIni  tParams
();

        // Set   MQTT   initializa  tion   parameters
         IoTMqttCli  entConfig   config  =  new   IoTMqttCli
entConfig ();
         config . productKey  =  productKey ;
         config . deviceName  =  deviceName ;
         config . deviceSecr  et  =  deviceSecr  et ;
         params . mqttClient  Config  =  config ;

        // Set   device   certificat  e   informatio  n   for
initializa  tion   and   import   the   certificat  e   informatio
n
         DeviceInfo   deviceInfo  =  new   DeviceInfo ();
         deviceInfo . productKey  =  productKey ;
         deviceInfo . deviceName  =  deviceName ;
         deviceInfo . deviceSecr  et  =  deviceSecr  et ;
         params . deviceInfo  =  deviceInfo ;

        // Initialize   the   SDK
         LinkKit . getInstanc  e (). init ( params ,  new
ILinkKitCo  nnectListe  ner () {
             public   void   onError ( AError   aError ) {
                 System . out . println (" init   failed  !!  code
=" +  aError . getCode () + ", msg =" +  aError . getMsg () + ",
subCode ="
                         +  aError . getSubCode () + ", subMsg =" +
aError . getSubMsg ());
            }

             public   void   onInitDone ( InitResult   initResult )
{
```

```
                System . out . println (" init   success  !!") ;
            }
        });

        // Perform   the   subsequent   operations   only   after
  the   initializa  tion   is   complete .  You   can   increase
the   sleep   time   as   needed .
         Thread . sleep ( 2000 );
    }

    /**
     *  Publish   a   message
     *
     * @ param   topic   The   topic   to   which   the   message
is   published
     * @ param   payload   The   message   payload
     */
     public   static   void   publish ( String   topic ,  String
payload ) {
         MqttPublis  hRequest   request  =  new   MqttPublis
hRequest ();
         request . topic  =  topic ;
         request . payloadObj  =  payload ;
         request . qos  =  0 ;
         LinkKit . getInstanc  e (). getMqttCli  ent (). publish (
request ,  new   IConnectSe  ndListener () {
            @ Override
             public   void   onResponse ( ARequest   aRequest ,
AResponse   aResponse ) {
            }

            @ Override
             public   void   onFailure ( ARequest   aRequest ,
AError   aError ) {
            }
        });
    }

    /**
     *  Subscribe   to   a   topic
     *
     * @ param   topic   The   topic   of   messages   to
subscribe   to
     */
     public   static   void   subscribe ( String   topic ) {
         MqttSubscr  ibeRequest   request  =  new   MqttSubscr
ibeRequest ();
         request . topic  =  topic ;
         request . isSubscrib  e  =  true ;
         LinkKit . getInstanc  e (). getMqttCli  ent (). subscribe (
request ,  new   IConnectSu  bscribeLis  tener () {
            @ Override
             public   void   onSuccess () {
            }

            @ Override
             public   void   onFailure ( AError   aError ) {
            }
        });
    }

    /**
     *  Unsubscrib  e   from   a   topic
     *
```

```
     * @ param   topic   The   topic   of   messages   to
unsubscrib  e   from
     */
     public   static   void   unsubscrib  e ( String   topic ) {
         MqttSubscr  ibeRequest   request  =  new   MqttSubscr
ibeRequest ();
         request . topic  =  topic ;
         request . isSubscrib  e  =  false ;
         LinkKit . getInstanc  e (). getMqttCli  ent (). unsubscrib
e ( request ,  new   IConnectUn  scribeList  ener () {
            @ Override
             public   void   onSuccess () {
            }

            @ Override
             public   void   onFailure ( AError   aError ) {
            }
        });
    }

    /**
     *  Terminate   the   connection
     */
     public   static   void   disconnect () {
        // Perform   the   deinitiali  zation
         LinkKit . getInstanc  e (). deinit ();
    }

}
```

5. **Add the** *SSHDevice . java* **file. The** *SSHDevice . java* **file includes the main method. You can call the main method to receive RRPC commands, call** `SSHShell` **to run SSH commands, and return RRPC responses. In the** *SSHDevice . java* **file, you must enter the device certificate information, including ProductKey, DeviceName, and DeviceSecret, and the SSH account and password.**

```
public   class   SSHDevice  {

   //=================== Start   to   Enter   Required   Parameters
==========================
   // ProductKey
    private   static   String   productKey  = "";
   //
    private   static   String   deviceName  = "";
   // DeviceSecr  et
    private   static   String   deviceSecr  et  = "";
   // The   message   communicat  ion   topic .  You   can
directly   use   the   topic   without   creating   or   defining
the   topic .
    private   static   String   rrpcTopic  = "/ sys /" +
productKey  + "/" +  deviceName  + "/ rrpc / request /+";
   // The   domain   name   or   IP   address   that   you   want
to   access   by   using   SSH
    private   static   String   host  = " 127 . 0 . 0 . 1 ";
   // The   SSH   username
    private   static   String   username  = "";
   // The   SSH   password
    private   static   String   password  = "";
   // The   SSH   port
    private   static   int   port  =  22 ;
```

```
   //=================== End ==========================

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

       // Listen   to   downstream   data
        registerNo  tifyListen  er ();

       // Establish   the   connection
        Device . connect ( productKey ,  deviceName ,  deviceSecr
et );

       // Subscribe   to   a   topic
        Device . subscribe ( rrpcTopic );
   }

    public   static   void   registerNo  tifyListen  er () {
        LinkKit . getInstanc  e (). registerOn  NotifyList  ener (
new   IConnectNo  tifyListen  er () {
           @ Override
            public   boolean   shouldHand  le ( String   connectId
,  String   topic ) {
               // Only   process   messages   of   the   specified
  topic
                if  ( topic . contains ("/ rrpc / request /")) {
                    return   true ;
               }  else  {
                    return   false ;
               }
           }

           @ Override
            public   void   onNotify ( String   connectId ,  String
  topic ,  AMessage   aMessage ) {
               // Receive   RRPC   requests   and   resturn   RRPC
  responses
                try  {
                   // Run   the   SSH   command
                    String   payload  =  new   String (( byte [])
aMessage . getData (), " UTF - 8 ");
                    SSHShell   sshExecuto  r  =  new   SSHShell (
host ,  username ,  password ,  port );
                    sshExecuto  r . execute ( payload );

                   // Obtain   the   command   output
                    StringBuff  er   sb  =  new   StringBuff  er
();
                    Vector < String >  stdout  =  sshExecuto  r .
getStdout ();
                    for  ( String   str  :  stdout ) {
                        sb . append ( str );
                        sb . append ("\ n ");
                   }

                   // Return   command   output   to   the
server
                    String   response  =  topic . replace ("/
request /", "/ response /");
                    Device . publish ( response ,  sb . toString
());
               }  catch  ( Unsupporte  dEncodingE  xception   e ) {
                    e . printStack  Trace ();
               }
           }
```

```
            @ Override
             public   void   onConnectS  tateChange ( String
 connectId ,  ConnectSta  te   connectSta  te ) {
             }
        });
    }

}
```

Develop the server SDK

After you have downloaded and installed the server SDK and SDK demo, add project dependencies and the following Java files.

1. **Add dependencies to the** *pom . xml* **file.**

```
<! --  Server   SDK  -->
< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - iot </ artifactId >
    < version > 6 . 5 . 0 </ version >
</ dependency >
< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - core </ artifactId >
    < version > 3 . 5 . 1 </ version >
</ dependency >

<! --  commons - codec  -->
< dependency >
    < groupId > commons - codec </ groupId >
    < artifactId > commons - codec </ artifactId >
    < version > 1 . 8 </ version >
</ dependency >
```

2. **Add the** *OpenApiCli ent . java* **file that is used to call the IoT Platform API.**

```
public   class   OpenApiCli  ent  {

    private   static   DefaultAcs  Client   client  =  null ;

    public   static   DefaultAcs  Client   getClient ( String
 accessKeyI  D ,  String   accessKeyS  ecret ) {

        if  ( client  ! =  null ) {
            return   client ;
        }

        try  {
            IClientPro  file   profile  =  DefaultPro  file .
 getProfile (" cn - shanghai ",  accessKeyI  D ,  accessKeyS  ecret
 );
            DefaultPro  file . addEndpoin  t (" cn - shanghai ", "
 cn - shanghai ", " Iot ", " iot . cn - shanghai . aliyuncs . com
 ");
            client  =  new   DefaultAcs  Client ( profile );
        }  catch  ( Exception   e ) {
            System . out . println (" create   Open   API   Client
  failed  !!  exception :" +  e . getMessage ());
```

```
        }

         return   client ;
    }

}
```

3. **Add the** *SSHCommand Sender . java* **file. The** *SSHCommand Sender . java* **file contains the main method that is used to send SSH commands and receive responses to SSH commands. In the** *SSHCommand Sender . java* **file, you must enter your Alibaba Cloud account information including AccessKey ID and AccessKey Secret, and device certificate information including ProductKey and DeviceName, and SSH command.**

```
public   class   SSHCommand  Sender  {

   //=================== Start   to   Enter   Required   Parameters
============================

   // AccessKey   ID   of   your   Alibaba   Cloud   account

    private   static   String   accessKeyI  D  = "";

   // AccessKey   Secret   of   your   Alibaba   Cloud   account

    private   static   String   accessKeyS  ecret  = "";

   // ProductKey

    private   static   String   productKey  = "";

   // DeviceName

    private   static   String   deviceName  = "";

   //=================== End ============================


    public   static   void   main ( String []  args )  throws
ServerExce  ption ,  ClientExce  ption ,  Unsupporte  dEncodingE
xception  {


       // The   Linux   command

        String   payload  = " uname  - a ";


       // Construct   an   RRPC   request

        RRpcReques  t   request  =  new   RRpcReques  t ();

        request . setProduct  Key ( productKey );

        request . setDeviceN  ame ( deviceName );

        request . setRequest  Base64Byte ( Base64 . encodeBase
64String ( payload . getBytes ()));
```

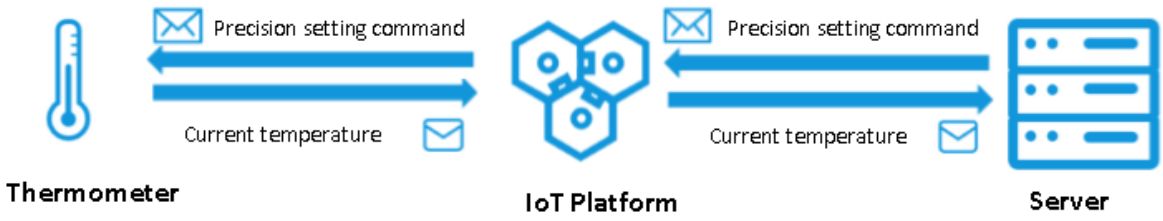
```
        request . setTimeout ( 5000 );


       // Obtain   informatio  n   about   the   client   connected
  to   the   Raspberry   Pi   server

        DefaultAcs  Client   client  =  OpenApiCli  ent . getClient
( accessKeyI  D ,  accessKeyS  ecret );


       // Initiate   an   RRPC   request

        RRpcRespon  se   response  = ( RRpcRespon  se )  client .
getAcsResp  onse ( request );


       // Process   an   RRPC   response

       //" response . getSuccess ()"  only   indicates   that   the
  RRPC   request   has   been   sent .  It   does   not   indicate
  that   the   device   has   received   the   RRPC   request   and
  has   returned   a   response .

       // Identify   whether   the   message   has   been
received   and   a   response   has   been   returned   according
to   the   RrpcCode   value .  For   more   informatio  n ,  see
the   document   of   RRpc   API   https :// www . alibabaclo  ud .
com / help / doc - detail / 69797 . htm .

        if  ( response  ! =  null  && " SUCCESS ". equals (
response . getRrpcCod  e ())) {

           // Output   the   response

            System . out . println ( new   String ( Base64 .
decodeBase  64 ( response . getPayload  Base64Byte ()), " UTF - 8
"));

       }  else  {

           // An   error   occurred   while   outputting   the
response   and   an   RRPC   code   is   displayed .

            System . out . println ( response . getRrpcCod  e ());

       }

   }


}
```

8 Use custom topics for communication

You can define custom topic categories in IoT Platform. Then, a device can send messages to a custom topic, and your server can receive the messages through the HTTP/2 SDK. Your server can also call the API operation Pub to send commands to the device.

Scenario

In this example, an electronic thermometer periodically exchanges data with a server . The thermometer sends the current temperature to the server, and the server sends the precision setting command to the thermometer.



Prepare the development environment

In this example, both the devices and the server use Java SDKs, so you need to prepare the Java development environment first. You can download Java tools at Java official website and install the Java development environment.

Add the following Maven dependencies to import the device SDK (Link Kit Java SDK) and IoT SDK:

```
< dependenci  es >
 < dependency >
     < groupId > com . aliyun . alink . linksdk </ groupId >
     < artifactId > iot - linkkit - java </ artifactId >
     < version > 1 . 2 . 0 . 1 </ version >
     < scope > compile </ scope >
 </ dependency >
 <  dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - core </ artifactId >
      < version > 3 . 7 . 1 </ version >
  </ dependency >
  < dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - iot </ artifactId >
      < version > 6 . 9 . 0 </ version >
  </ dependency >
  < dependency >
```

```
    < groupId > com . aliyun . openservic  es </ groupId >
    < artifactId > iot - client - message </ artifactId >
    < version > 1 . 1 . 2 </ version >
</ dependency >
</ dependenci  es >
```

Create a product and a device

First, you need to create a product, define custom topic categories, define the TSL model, configure service subscription, and create a device in the IoT Platform console .

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose Devices > Product.
3. Click Create Product to create a thermometer product.

   For more information, see Create a product.
4. After the product is created, find the product and click View.
5. On the Topic Categories tab of the Product Details page, add custom topic categories.

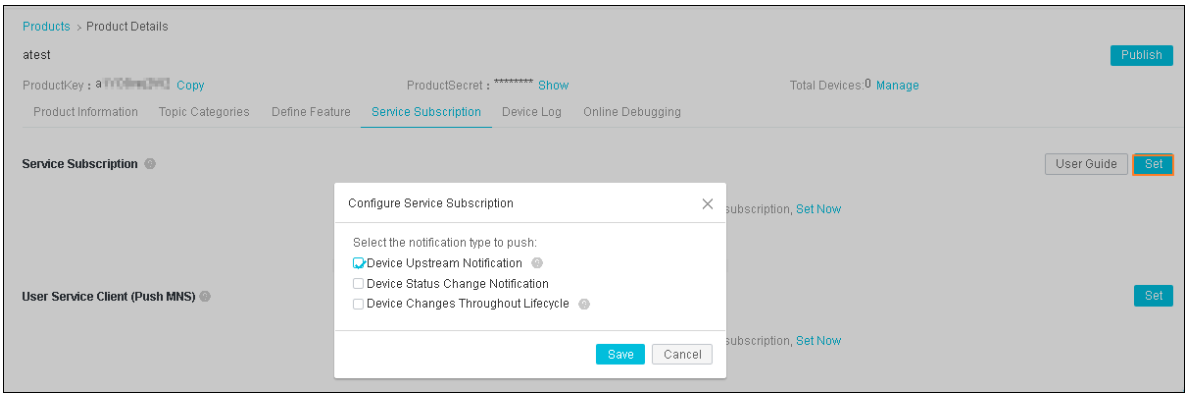
   For more information, see Create a topic category.

   In this example, add the following two topic categories:

   - /${productKey}/${deviceName}/user/devmsg: used by devices to publish messages. Set Device Operation Authorizations to Publish for this topic category .
   - /${productKey}/${deviceName}/user/cloudmsg: used by devices to receive subscribed messages. Set Device Operation Authorizations to Subscribe for this topic category.

6. On the Service Subscription tab, set the type of messages to be pushed to the HTTP/2 SDK to Device Upstream Notification.

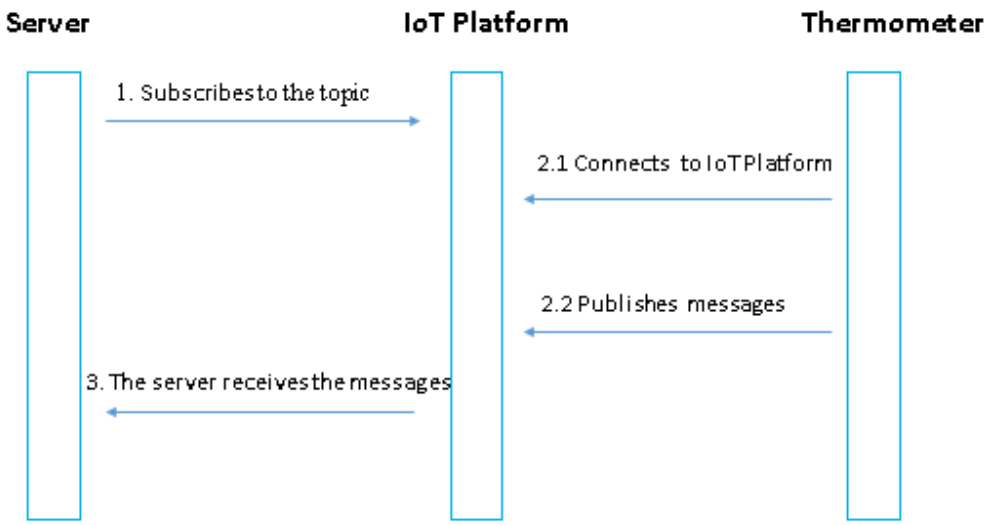
   For more information, see Development guide for Java HTTP/2 SDK.



7. In the left-side navigation pane, choose Devices and add a thermometer device under the thermometer product that has been created. For more information, see Create a device.

The server receives messages from the device

The following figure shows how the device sends a message to the server.



- Configure the HTTP/2 SDK, which will be installed in the server, and set a callback for the specified topic.

  After service subscription is configured, the server listens to all the messages sent by all the devices under the product.

  - Connect the HTTP/2 SDK to IoT Platform.

```
// Configure identity verificati on informatio n .
String accessKey = " The AccessKey ID of your
Alibaba Cloud account ";
String accessSecr et = " The AccessKey Secret of your
  Alibaba Cloud account ";
String uid = " Your Alibaba Cloud account ID ";
String regionId = " The ID of the region where the
  device is located ";
String productKey = " The key of the product ";
String deviceName = " The name of the device ";
String endPoint = " https ://" + uid + ". iot - as - http2
." + regionId + ". aliyuncs . com ";
// Configure the connection .
Profile profile = Profile . getAccessK eyProfile ( endPoint
, regionId , accessKey , accessSecr et );

// Construct a client .
MessageCli ent client = MessageCli entFactory . messageCli
ent ( profile );

// Receive the message .
client . connect ( new MessageCal lback () {
   @ Override
    public Action consume ( MessageTok en messageTok en )
{
        Message m = messageTok en . getMessage ();
        System . out . println ("\ ntopic =" + m . getTopic ());
        System . out . println (" payload =" + new String ( m
. getPayload ()));
        System . out . println (" generateTi me =" + m .
getGenerat eTime ());
       // CommitSucc ess indicates that the current
  message has been consumed . In this case , IoT
Platform will delete the message . If the message
  is not marked with CommitSucc ess , IoT Platform
will retain the message until it expires .
        return MessageCal lback . Action . CommitSucc ess ;
    }
});
```

  - Configure the message receiving interface.

    In this example, the server subscribes to the topic /${productKey}/${deviceName}/user/devmsg. Therefore, you need to set a callback for this topic.

```
// Define the callback .
MessageCal lback messageCal lback = new MessageCal lback
() {
// Obtain and display the message . You can set
the required action for the server here .
           @ Override
```

```
            public   Action   consume ( MessageTok  en
messageTok  en ) {
                Message   m  =  messageTok  en . getMessage ();
                log . info (" receive  : " +  new   String (
messageTok  en . getMessage (). getPayload ()));
                System . out . println ( messageTok  en .
getMessage ());
                return   MessageCal  lback . Action . CommitSucc
ess ;
            }
        };
        //  Register   the   callback .
         client . setMessage  Listener ("/" +  productKey  + "/"
+  deviceName  + "/ user / devmsg ",  messageCal  lback );
```

For more information, see Development guide for Java HTTP/2 SDK.

- Configure the device SDK to send a message.
  - Configure device authentication information.

```
final   String   productKey  = " XXXXXX ";
final   String   deviceName  = " XXXXXX ";
final   String   deviceSecr  et  = " XXXXXXXXX ";
final   String   region  = " XXXXXX ";
```

  - Set connection initialization parameters, including MQTT connection information, device information, and initial device status.

```
 LinkKitIni  tParams   params  =  new   LinkKitIni  tParams ();
//  Configure   MQTT   connection   informatio  n .  Link   Kit
 uses   MQTT   as   the   underlying   protocol .
 IoTMqttCli  entConfig   config  =  new   IoTMqttCli  entConfig
 ();
 config . productKey  =  productKey ;
 config . deviceName  =  deviceName ;
 config . deviceSecr  et  =  deviceSecr  et ;
 config . channelHos  t  =  productKey  + ". iot - as - mqtt ." +
 region  + ". aliyuncs . com : 1883 ";
//  Configure   device   informatio  n .
 DeviceInfo   deviceInfo  =  new   DeviceInfo ();
 deviceInfo . productKey  =  productKey ;
 deviceInfo . deviceName  =  deviceName ;
 deviceInfo . deviceSecr  et  =  deviceSecr  et ;
//  Register   the   initial   device   status .
 Map < String ,  ValueWrapp  er >  propertyVa  lues  =  new
 HashMap < String ,  ValueWrapp  er >();

 params . mqttClient  Config  =  config ;
 params . deviceInfo  =  deviceInfo ;
 params . propertyVa  lues  =  propertyVa  lues ;
```

  - Initialize the connection.

```
//  Initialize   the   connection   and   set   the   callback
   that   is   called   when   the   initializa  tion   is
 successful .
 LinkKit . getInstanc  e (). init ( params ,  new   ILinkKitCo
 nnectListe  ner () {
     @ Override
      public   void   onError ( AError   aError ) {
```

```
            System . out . println (" Init   error :" +  aError );
        }

        //  Set   the   callback   that   is   called   when   the
    initializa  tion   is   successful .
        @ Override
         public   void   onInitDone ( InitResult   initResult ) {
            System . out . println (" Init   done :" +  initResult
    );
        }
    });
```

- **Send a message from the device.**

  After connecting to IoT Platform, the device sends a message to the specified topic. Replace the content of the `onInitDone` callback like the following example:

```
@ Override
  public   void   onInitDone ( InitResult   initResult ) {
     //  Set   the   topic   to   which   the   message   is
    published   and   the   message   content .
      MqttPublis  hRequest   request  =  new   MqttPublis
    hRequest ();
      request . topic  = "/" +  productKey  + "/" +  deviceName  +
    "/ user / devmsg ";
      request . qos  =  0 ;
      request . payloadObj  = "{\" temperatur  e \": 35 . 0 , \"
    time \":\" sometime \"}";
     //  Publish   the   message   and   set   the   callback
    that   is   called   when   the   message   is   published .
      LinkKit . getInstanc  e (). publish ( request ,  new
    IConnectSe  ndListener () {
         @ Override
          public   void   onResponse ( ARequest   aRequest ,
    AResponse   aResponse ) {
             System . out . println (" onResponse :" +  aResponse
    . getData ());
         }

         @ Override
          public   void   onFailure ( ARequest   aRequest ,
    AError   aError ) {
             System . out . println (" onFailure :" +  aError .
    getCode () +  aError . getMsg ());
         }
     });
    }
```

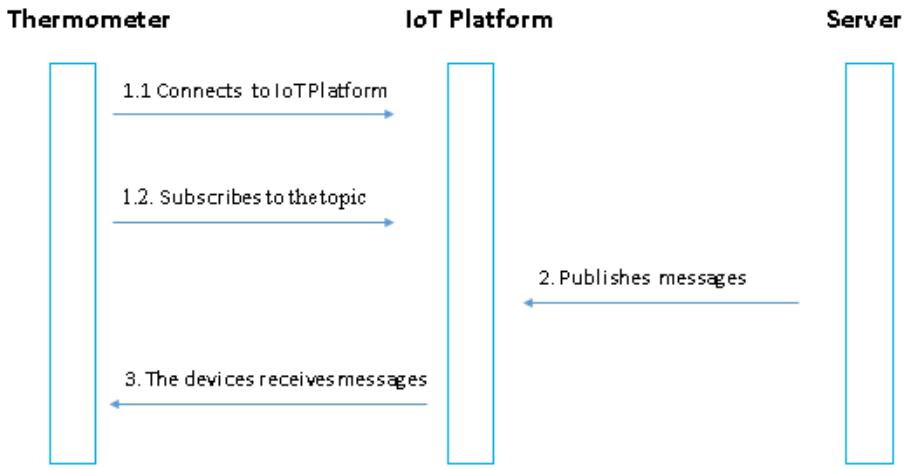
The server receives the following message:

```
 Message
{ payload ={" temperatur  e ": 35 . 0 , " time ":" sometime "},
 topic ='/ a1uzcH0 ****/ device1 / user / devmsg ',
 messageId =' 1131755639  450642944 ',
 qos = 0 ,
```

```
  generateTi  me = 1558666546  105 }
```

The server sends messages to the device

The following figure shows how the server sends a message to the device.



- Configure the device SDK to subscribe to a topic.

  For more information about how to configure device authentication information , set connection initialization parameters, and initialize the connection, see the device SDK configuration in the previous section.

  The device needs to subscribe to a specific topic to receive messages sent by the server.

  The following example demonstrates how to configure the device SDK to subscribe to a topic:

```
// Set the callback that is called when the
 initializa tion is successful .
@ Override
 public void onInitDone ( InitResult initResult ) {
    // Set the topic to which the device subscribes .
     MqttSubscr ibeRequest request = new MqttSubscr
 ibeRequest ();
     request . topic = "/" + productKey + "/" + deviceName + "/
 user / cloudmsg ";
     request . isSubscrib e = true ;
    // Send a subscripti on request and set the
 callbacks that are called when the subscripti on
 succeeds and fails , respective ly .
     LinkKit . getInstanc e (). subscribe ( request , new
 IConnectSu bscribeLis tener () {
        @ Override
         public void onSuccess () {
             System . out . println ("");
        }
```

```
        @ Override
         public   void   onFailure ( AError   aError ) {

        }
    });

    //  Set   the   listener   for   listening   to   subscribed
messages .
     IConnectNo  tifyListen  er   notifyList  ener  =  new
IConnectNo  tifyListen  er () {
        //  Define   the   callback   that   is   called   when   a
  subscribed   message   is   received .
        @ Override
         public   void   onNotify ( String   connectId ,  String
topic ,  AMessage   aMessage ) {
            System . out . println (
               " received   message   from  " +  topic  + ":" +
new   String (( byte []) aMessage . getData ()));
        }

        @ Override
         public   boolean   shouldHand  le ( String   s ,  String
s1 ) {
            return   false ;
        }

        @ Override
         public   void   onConnectS  tateChange ( String   s ,
ConnectSta  te   connectSta  te ) {

        }
    };
     LinkKit . getInstanc  e (). registerOn  NotifyList  ener (
notifyList  ener );
}
```

- **Configure the IoT SDK to call the Pub operation to publish a message.**
  - **Configure identity verification information.**

```
  String   regionId  = " The   ID   of   the   region   where
the   device   is   located ";
  String   accessKey  = " The   AccessKey   ID   of   your
Alibaba   Cloud   account ";
  String   accessSecr  et  = " The   AccessKey   Secret   of
your   Alibaba   Cloud   account ";
  final   String   productKey  = " The   key   of   the   product
";
```

  - **Set connection parameters.**

```
//  Construct   a   client .
 DefaultPro  file   profile  =  DefaultPro  file . getProfile (
 regionId ,  accessKey ,  accessSecr  et );
 IAcsClient   client  =  new   DefaultAcs  Client ( profile );
```

  - **Set the parameters for publishing a message.**

```
 PubRequest   request  =  new   PubRequest ();
 request . setQos ( 0 );
//  Set   the   topic   to   which   the   message   is
 published .
```

```
request . setTopicFu  llName ("/" +  productKey  + "/" +
deviceName  + "/ user / cloudmsg ");
request . setProduct  Key ( productKey );
//  Set   the   message   content .  The   message   content
must   be   encoded   in   Base64 .  Otherwise ,  the   message
content   will   be   garbled   characters .
request . setMessage  Content ( Base64 . encode ("{\" accuracy
\": 0 . 001 ,\" time \": now }"));
```

- **Publish the message.**

```
try  {
     PubRespons  e   response  =  client . getAcsResp  onse (
request );
     System . out . println (" pub   success ?:" +  response .
getSuccess ());
}  catch  ( Exception   e ) {
     System . out . println ( e );
}
```

The device receives the following message:

```
msg  = [{" accuracy ": 0 . 001 ," time ": now }]
```

Appendix: demo

Click here to download and view the complete demo for this example.

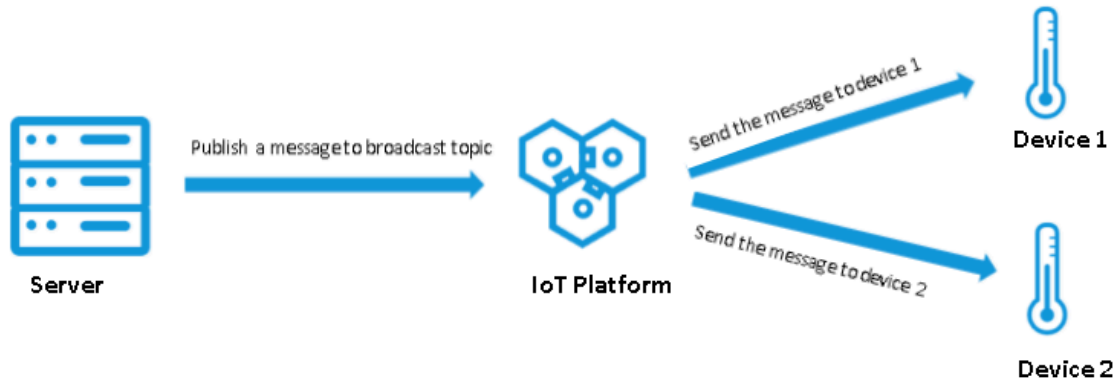
9 Broadcast messages

IoT Platform supports broadcast communication. That is, IoT Platform allows a server to broadcast a message to multiple devices under a product. The devices only need to subscribe the broadcast topic to receive the broadcast message sent by the server. This topic describes how to configure broadcast communication.

Scenario

You have multiple thermometers connected to IoT Platform, and need to send the same precision setting command to all these thermometers from your server.

All the devices subscribe to the same broadcast topic, and the server calls the PubBroadcast operation to publish the message to this topic.



Prepare the development environment

In this example, both the devices and the server use Java SDKs, so you need to prepare the Java development environment first. You can download Java tools at Java official website and install the Java development environment.

Add the following Maven dependencies to import the device SDK (Link Kit Java SDK) and IoT SDK:

```
< dependenci  es >
 < dependency >
     < groupId > com . aliyun . alink . linksdk </ groupId >
     < artifactId > iot - linkkit - java </ artifactId >
     < version > 1 . 2 . 0 . 1 </ version >
     < scope > compile </ scope >
 </ dependency >
 <  dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - core </ artifactId >
```

```
      < version > 3 . 7 . 1 </ version >
  </ dependency >
  < dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - iot </ artifactId >
      < version > 6 . 9 . 0 </ version >
  </ dependency >
  < dependency >
    < groupId > com . aliyun . openservic  es </ groupId >
    < artifactId > iot - client - message </ artifactId >
    < version > 1 . 1 . 2 </ version >
</ dependency >
</ dependenci  es >
```

Create a product and devices

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose Devices > Product.
3. Click Create Product to create a thermometer product.

   For more information, see Create a product.
4. In the left-side navigation pane, choose Devices and add two thermometer devices under the thermometer product that has been created.

   For more information, see Create multiple devices at a time.

Configure the device SDK

Connect the device SDK to IoT Platform and subscribe to the broadcast topic.

- Connect the device SDK to IoT Platform.
  - Configure device authentication information.

    ```
    final   String   productKey  = " XXXXXX ";
    final   String   deviceName  = " XXXXXX ";
    final   String   deviceSecr  et  = " XXXXXXXXX ";
    final   String   region  = " XXXXXX ";
    ```

    `productKey` , `deviceName` , and `deviceSecr et` : device authentication information. You can obtain the values of these parameters on the device details page in the IoT Platform console.

    `region` : the ID of the region where the device is located. For more information about region IDs, see Regions and zones.
  - Set connection initialization parameters, including MQTT connection information, device information, and initial device status.

    ```
    LinkKitIni  tParams  params  =  new  LinkKitIni  tParams ();
    //  Configure  MQTT  connection  informatio  n .  Link  Kit
    uses  MQTT  as  the  underlying  protocol .
    ```

```
IoTMqttCli entConfig config = new IoTMqttCli entConfig
();
config . productKey = productKey ;
config . deviceName = deviceName ;
config . deviceSecr et = deviceSecr et ;
config . channelHos t = productKey + ". iot - as - mqtt ." +
region + ". aliyuncs . com : 1883 ";
// Configure device informatio n .
DeviceInfo deviceInfo = new DeviceInfo ();
deviceInfo . productKey = productKey ;
deviceInfo . deviceName = deviceName ;
deviceInfo . deviceSecr et = deviceSecr et ;
// Register the initial device status .
Map < String , ValueWrapp er > propertyVa lues = new
HashMap < String , ValueWrapp er >();

params . mqttClient Config = config ;
params . deviceInfo = deviceInfo ;
params . propertyVa lues = propertyVa lues ;
```

- Initialize the connection.

```
// Initialize the connection and set the callback
   that is called when the initializa tion is
successful .
LinkKit . getInstanc e (). init ( params , new ILinkKitCo
nnectListe ner () {
    @ Override
     public void onError ( AError aError ) {
         System . out . println (" Init error :" + aError );
    }

    // Set the callback that is called when the
initializa tion is successful .
    @ Override
     public void onInitDone ( InitResult initResult ) {
         System . out . println (" Init done :" + initResult
);
    }
});
```

- Subscribe to the broadcast topic. Set the `onInitDone` callback to send a subscription request.

  You do not need to create the broadcast topic in advance. Instead, directly enter the broadcast topic in the format of /broadcast/${productKey}/Custom field.

  In this example, the broadcast topic is `"/ broadcast /" + productKey + "/ userDefine "`.

  Note:

  The same broadcast topic must be configured in the device SDK and IoT SDK.

```
public void onInitDone ( InitResult initResult ) {
    // Set the broadcast topic to which the device
  subscribes . You do not need to create this topic
  in the IoT Platform console .
```

```
     MqttSubscr  ibeRequest   request  =  new   MqttSubscr
ibeRequest ();
     request . topic  = "/ broadcast /" +  productKey  + "/
userDefine ";
     request . isSubscrib  e  =  true ;
    //  Send   a   subscripti  on   request   and   set   the
callbacks   that   are   called   when   the   subscripti  on
succeeds   and   fails ,  respective  ly .
     LinkKit . getInstanc  e (). subscribe ( request ,  new
IConnectSu  bscribeLis  tener () {
        @ Override
         public   void   onSuccess () {
             System . out . println ("");
        }

        @ Override
         public   void   onFailure ( AError   aError ) {

        }
    });

    //  Set   the   listener   for   listening   to   subscribed
messages .
     IConnectNo  tifyListen  er   notifyList  ener  =  new
IConnectNo  tifyListen  er () {
        //  Define   the   callback   that   is   called   when   a
  subscribed   message   is   received .
        @ Override
         public   void   onNotify ( String   connectId ,  String
topic ,  AMessage   aMessage ) {
             System . out . println (
                " received   message   from  " +  topic  + ":" +
new   String (( byte []) aMessage . getData ()));
        }

        @ Override
         public   boolean   shouldHand  le ( String   s ,  String
s1 ) {
             return   false ;
        }

        @ Override
         public   void   onConnectS  tateChange ( String   s ,
ConnectSta  te   connectSta  te ) {

        }
    };
    //  Register   the   listener   for   listening   to
subscribed   messages .
     LinkKit . getInstanc  e (). registerOn  NotifyList  ener (
notifyList  ener );
}
```

Configure the IoT SDK

Configure the IoT SDK to send a broadcast message.

- **Configure identity verification information.**

```
 String   regionId  = " The   ID   of   the   region   where   the
  device   is   located ";
```

```
String accessKey = " The AccessKey ID of your Alibaba
 Cloud account ";
String accessSecr et = " The AccessKey Secret of your
 Alibaba Cloud account ";
final String productKey = " The key of the product ";
```

- Configure the IoT SDK to call the PubBroadcast operation to publish a broadcast message.

```
// Construct a client .
 DefaultPro file profile = DefaultPro file . getProfile (
regionId , accessKey , accessSecr et );
 IAcsClient client = new DefaultAcs Client ( profile );

 PubBroadca stRequest request = new PubBroadca stRequest
();
// Set the topic to which the broadcast message
 is published . This topic must be same as the
broadcast topic to which the device subscribes .
 request . setTopicFu llName ("/ broadcast /" + productKey + "/
userDefine ");
 request . setProduct Key ( productKey );
// Set the message content . The message content
must be encoded in Base64 . Otherwise , the message
content will be garbled characters .
 request . setMessage Content ( Base64 . encode ("{\" accuracy \":
0 . 001 ,\" time \": now }"));
```

- Publish the broadcast message.

```
try {
    PubBroadca stResponse response = client . getAcsResp onse
( request );
    System . out . println (" broadcast pub success ?:" +
response . getSuccess ());
} catch ( Exception e ) {
    System . out . println ( e );
}
```

Verification

Verify that the message broadcast by the IoT SDK to devices is `"{\" accuracy \": 0 . 001 ,\" time \": now }"`.

In logs of the two thermometers, verify that the message `{" accuracy ": 0 . 001 ," time ": now }` is received.

Appendix: demo

Click here to download and view the complete demo for this example.