ALIBABA CLOUD

Alibaba Cloud

物联网平台

最佳实践

文档版本：20220624

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.设备接入

1.1. 使用Paho接入物联网平台

1.1.1. Paho-MQTT C接入示例

本文介绍如何使用Paho提供的嵌入式C语言MQTT开源工程，将设备接入阿里云物联网平台，并进行消息收发。

前提条件

已在物联网平台控制台，对应实例下，创建产品和设备，并获取MQTT接入域名和设备证书信息（ProductKey、DeviceName和DeviceSerect）。具体操作，请参见：

- 查看实例终端节点。
- 创建产品。
- 创建设备。

准备开发环境

本示例使用Ubuntu 16.04-LTS作为开发环境。执行以下命令构建开发环境。

```
sudo apt-get update
sudo apt-get install build-essential git sed cmake
```

下载C语言Paho MQTT库

执行以下命令，克隆C语言版本的Paho MQTT库。

```
git clone https://github.com/eclipse/paho.mqtt.embedded-c.git
```

> **说明** 编写本Demo示例时，使用master分支，commit id为 `29ab2aa29c5e47794284376d7f8386cfd54c3eed` 。

Paho嵌入式C工程提供了以下三个子项目：

- *MQTTPacket*：提供MQTT数据包的序列化与反序列化，以及部分辅助函数。
- *MQTTClient*：封装*MQTTPacket*生成的高级别C++客户端程序。
- *MQTTClient-C*：封装*MQTTPacket*生成的高级别C客户端程序。

*MQTTClient-C*中包含：

```
├── CMakeLists.txt
├── samples
│   ├── CMakeLists.txt
│   ├── FreeRTOS
│   └── linux
├── src
│   ├── CMakeLists.txt
│   ├── FreeRTOS
│   ├── MQTTClient.c
│   ├── MQTTClient.h
│   ├── cc3200
│   └── linux
└── test
    ├── CMakeLists.txt
    └── test1.c
```

- *samples*目录提供*FreeRTOS*和*Linux*两个例程，分别支持FreeRTOS和Linux系统。
- *src*目录提供*MQTTClient*的代码实现能力，以及用于移植到FreeRTOS、cc3200和Linux的网络驱动。

了解Paho MQTT的更多API细节，可以查看*MQTTClient.h*。

接入物联网平台

1. 单击打开aiot_mqtt_sign.c，复制阿里云提供的计算MQTT连接参数所需的源码，然后粘贴保存为本地的*aiot_mqtt_sign.c*文件。

   *aiot_mqtt_sign.c*文件定义了函数aiotMqttSign()，函数说明如下：

   - 原型：

     ```
     int aiotMqttSign(const char *productKey, const char *deviceName, const char *deviceSecret,
                      char clientId[150], char username[65], char password[65]);
     ```

   - 功能：

     用于计算设备接入物联网平台的MQTT连接参数username、password和clientid。

   - 输入参数：

参数	类型	说明
productKey	const char *	设备所属产品的ProductKey，该设备在物联网平台上的身份证书信息之一。
deviceName	const char *	设备名称，该设备在物联网平台上的身份证书信息之一。
deviceSecret	const char *	设备密钥，该设备在物联网平台上的身份证书信息之一。

   - 输出参数：

参数	类型	说明
username	char *	MQTT连接所需的用户名。
password	char *	MQTT连接所需的密码。

参数	类型	说明
clientId	char *	MQTT客户端ID。

- 返回码说明：

返回码	说明
0	成功
-1	失败

2. 添加实现设备接入物联网平台的程序文件。

   您需编写程序调用*aiot_mqtt_sign.c中的aiotMqttSign()函数*计算MQTT连接参数，实现接入物联网平台和通信。

   开发说明和示例代码如下：

   - 调用aiotMqttSign()接口，生成连接MQTT服务端的三个建连参数clientId、username和password。

```
#define EXAMPLE_PRODUCT_KEY            "a11xsrW****"
#define EXAMPLE_DEVICE_NAME            "paho_****"
#define EXAMPLE_DEVICE_SECRET       "Y877Bgo8X5owd3lcB5wWDjryNPoB****"
extern int aiotMqttSign(const char *productKey, const char *deviceName, const char *d
eviceSecret,
                        char clientId[150], char username[65], char password[65]);
/* invoke aiotMqttSign to generate mqtt connect parameters */
char clientId[150] = {0};
char username[65] = {0};
char password[65] = {0};
if ((rc = aiotMqttSign(EXAMPLE_PRODUCT_KEY, EXAMPLE_DEVICE_NAME, EXAMPLE_DEVICE_SECRE
T, clientId, username, password) < 0)) {
    printf("aiotMqttSign -%0x4x\n", -rc);
    return -1;
}
printf("clientid: %s\n", clientId);
printf("username: %s\n", username);
printf("password: %s\n", password);
```

   - 接入物联网平台。

     需配置以下内容：

     - 调用NetworkInit和NetworkConnect建立TCP连接。
     - 调用MQTTClientInit初始化MQTT客户端。
     - 配置MQTT建连参数结构体MQTTPacket_connectData。

     示例代码：

```
/* network init and establish network to aliyun IoT platform */
NetworkInit(&n);
rc = NetworkConnect(&n, host, port);
printf("NetworkConnect %d\n", rc);
/* init mqtt client */
MQTTClientInit(&c, &n, 1000, buf, sizeof(buf), readbuf, sizeof(readbuf));
/* set the default message handler */
c.defaultMessageHandler = messageArrived;
/* set mqtt connect parameter */
MQTTPacket_connectData data = MQTTPacket_connectData_initializer;
data.willFlag = 0;
data.MQTTVersion = 3;
data.clientID.cstring = clientId;
data.username.cstring = username;
data.password.cstring = password;
data.keepAliveInterval = 60;
data.cleansession = 1;
printf("Connecting to %s %d\n", host, port);
rc = MQTTConnect(&c, &data);
printf("MQTTConnect %d, Connect aliyun IoT Cloud Success!\n", rc);
```

- 发布消息。

  调用MQTTPublish()接口，向指定的自定义Topic发布自定义格式消息。

  通信Topic介绍，请参见什么是Topic。

```
char *pubTopic = "/"EXAMPLE_PRODUCT_KEY"/"EXAMPLE_DEVICE_NAME"/user/update";
int cnt = 0;
unsigned int msgid = 0;
while (!toStop)
{
    MQTTYield(&c, 1000);
    if (++cnt % 5 == 0) {
        MQTTMessage msg = {
            QOS1,
            0,
            0,
            0,
            "Hello world",
            strlen("Hello world"),
        };
        msg.id = ++msgid;
        rc = MQTTPublish(&c, pubTopic, &msg);
        printf("MQTTPublish %d, msgid %d\n", rc, msgid);
    }
}
```

- 订阅Topic，获取云端下发的消息。

```
void messageArrived(MessageData* md)
{
    MQTTMessage* message = md->message;
    printf("%.*s\t", md->topicName->lenstring.len, md->topicName->lenstring.data);
    printf("%.*s\n", (int)message->payloadlen, (char*)message->payload);
}
char *subTopic = "/"EXAMPLE_PRODUCT_KEY"/"EXAMPLE_DEVICE_NAME"/user/get";
printf("Subscribing to %s\n", subTopic);
rc = MQTTSubscribe(&c, subTopic, 1, messageArrived);
printf("MQTTSubscribe %d\n", rc);
```

关于设备、服务器和物联网平台的通信方式介绍，请参见通信方式概述。

3. 将步骤1下载的*aiot_mqtt_sign.c*文件和步骤2编辑的文件放到*../paho.mqtt.embedded-c/MQTTClient-C/samples/linux*中，然后编译工程。

示例代码

使用Demo代码程序接入物联网平台。

1. 下载Demo包，并解压缩。

   解压缩后，代码包里有以下两个文件：

文件	说明
aiot_mqtt_sign.c	该文件中的代码用于生成MQTT建连参数。*aiot_c_demo.c*运行时，会调用该文件中定义的aiotMqttSign()函数，计算出连接参数username、password和clientId。
aiot_c_demo.c	该文件包含设备与物联网平台连接和通信的逻辑代码。

2. 在*aiot_c_demo.c*中，将设备信息修改为您的设备信息。
   - 替换以下代码中EXAMPLE_PRODUCT_KEY、EXAMPLE_DEVICE_NAME和EXAMPLE_DEVICE_SECRET后的值为您的设备证书信息。

     ```
     #define EXAMPLE_PRODUCT_KEY            "产品ProductKey"
     #define EXAMPLE_DEVICE_NAME            "设备名称DeviceName"
     #define EXAMPLE_DEVICE_SECRET          "设备密钥DeviceSecret"
     ```

   - 修改代码 `char *host = EXAMPLE_PRODUCT_KEY".iot-as-mqtt.cn-shanghai.aliyuncs.com"` 中的值为对应接入域名。

     公共实例和企业版实例接入域名的格式说明，请参见查看实例终端信息。

3. 将*aiot_mqtt_sign.c*和已修改的*aiot_c_demo.c*文件放到Paho工程的目录*../paho.mqtt.embedded-c/MQTTClient-C/samples/linux*中。
4. 编译工程，并运行程序。

   有两种方法可以编译出可执行的程序：

   - 使用**CMake**。

a. 在*/paho.mqtt.embedded-c/MQTTClient-C/samples/linux*目录下的*CMakeLists.txt*文件中，增加*aiot_c_demo.c*和*aiot_mqtt_sign.c*。

修改后的*CMakeLists.txt*文件内容如下。

```
add_executable(
  stdoutsubc
  stdoutsub.c
)
add_executable(
  aiot_c_demo
  aiot_c_demo.c
  aiot_mqtt_sign.c
)
target_link_libraries(stdoutsubc paho-embed-mqtt3cc paho-embed-mqtt3c)
target_include_directories(stdoutsubc PRIVATE "../../src" "../../src/linux")
target_compile_definitions(stdoutsubc PRIVATE MQTTCLIENT_PLATFORM_HEADER=MQTTLinu
x.h)
target_link_libraries(aiot_c_demo paho-embed-mqtt3cc paho-embed-mqtt3c)
target_include_directories(aiot_c_demo PRIVATE "../../src" "../../src/linux")
target_compile_definitions(aiot_c_demo PRIVATE MQTTCLIENT_PLATFORM_HEADER=MQTTLin
ux.h)
```

b. 回到*/paho.mqtt.embedded-c*目录，执行以下命令，完成编译。

```
mkdir build.paho
cd build.paho
cmake ..
make
```

c. 编译完成后，在*/paho.mqtt.embedded-c/build.paho*目录下执行以下命令，运行程序。

```
./MQTTClient-C/samples/linux/aiot_c_demo
```

- 使用build.sh。

a. 打开*/paho.mqtt.embedded-c/MQTTClient-C/samples/linux*目录下的*build.sh*文件。

b. 将*build.sh*中的 `stdoutsub.c` 替换为 `aiot_mqtt_sign.c aiot_c_demo.c` ， `-o stdoutsub` 替换为 `-o aiot_c_demo` ，然后保存*build.sh*。

c. 修改完成后，在*/paho.mqtt.embedded-c/MQTTClient-C/samples/linux*目录下，执行命令 `./build.sh` ，完成编译。

完成编译后，生成*aiot_c_demo*可执行文件。

d. 执行命令 `./aiot_c_demo` ，运行程序。

运行成功，接入物联网平台的本地日志如下：

```
clientid: paho_mqtt&a11xsrW****|timestamp=2524608000000,_v=sdk-c-1.0.0,securemode=3,sig
nmethod=hmacsha256,lan=C|
username: paho_mqtt&a11xsrW****
password: 36E955DC3D9D012EF62C80657A29328B1CFAE6186C611A17DC7939FAB637****
NetworkConnect 0
Connecting to a11xsrW****.iot-as-mqtt.cn-shanghai.aliyuncs.com 443
MQTTConnect 0, Connect aliyun IoT Cloud Success!
Subscribing to /a11xsrW****/paho_mqtt/user/get
MQTTSubscribe 0
MQTTPublish 0, msgid 1
MQTTPublish 0, msgid 2
MQTTPublish 0, msgid 3
MQTTPublish 0, msgid 4
MQTTPublish 0, msgid 5
...
```

登录物联网平台控制台，可查看设备状态和日志。

- 选择**设备管理 > 设备**，可看到该设备的状态显示为**在线**。
- 选择**监控运维 > 日志服务**，可查看**云端运行日志**和**设备本地日志**日志。详细内容，请参见云端运行日志、设备本地日志。

错误码

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。服务端错误码说明，请参见错误排查。

1.1.2. Paho-MQTT Java接入示例

本文介绍如何使用Java语言的Paho MQTT库，接入阿里云物联网平台，并进行物模型消息通信。

前提条件

已在物联网平台中，创建了产品和设备，并在产品的**功能定义**页签下，定义一个LightSwitch属性。

请参见创建产品、单个创建设备和单个添加物模型。

准备开发环境

本示例使用的开发环境如下：

- 操作系统：Windows 10
- JDK版本：JDK8
- 集成开发环境：Intellij IDEA社区版

下载Java语言的Paho MQTT库

根据要使用的MQTT协议版本，在Maven工程中添加如下依赖：

- MQTT 3.1和3.1.1版本

```
<dependencies>
  <dependency>
      <groupId>org.eclipse.paho</groupId>
      <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
      <version>1.2.0</version>
    </dependency>
</dependencies>
```

- MQTT 5.0版本

```
<dependency>
   <groupId>org.eclipse.paho</groupId>
   <artifactId>org.eclipse.paho.mqttv5.client</artifactId>
   <version>1.2.5</version>
</dependency>
```

接入物联网平台

1. 单击打开MqttSign.java，下载阿里云提供的获取MQTT连接参数所需的源码。

   *MqttSign.java*文件定义了MqttSign类，类说明如下：

   - 原型：

     ```
     class MqttSign
     ```

   - 功能：

     用于计算设备接入物联网平台的MQTT连接参数username、password和clientid。

   - 成员：

类型定义	方法描述
public void	`calculate(String productKey, String deviceName, String deviceSecret)` 根据设备的productKey、deviceName和deviceSecret计算出MQTT连接参数username、password和clientid。
public String	`getUsername()` 用于获取MQTT建连参数username。
public String	`getPassword()` 用于获取MQTT建连参数password。
public String	`getClientid()` 用于获取MQTT建连参数clientid。

2. 打开IntelliJ IDEA，创建项目。
3. 将*MqttSign.java*导入项目中。
4. 在项目中，添加实现设备接入物联网平台的程序文件。

您需编写程序调用*MqttSign.java*中的MqttSign类计算MQTT连接参数，实现设备接入物联网平台和通信。

开发说明和示例代码如下：

- 调用MqttSign计算MQTT连接参数。

```
String productKey = "a1X2bEn****";
String deviceName = "example1";
String deviceSecret = "ga7XA6KdlEeiPXQPpRbAjOZXwG8y****";
// 计算MQTT连接参数。
MqttSign sign = new MqttSign();
sign.calculate(productKey, deviceName, deviceSecret);
System.out.println("username: " + sign.getUsername());
System.out.println("password: " + sign.getPassword());
System.out.println("clientid: " + sign.getClientid());
```

- 调用Paho MQTT客户端连接物联网平台。

```
//接入物联网平台的域名。
String port = "443";
String broker = "ssl://" + productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com" + ":"
+ port;
// Paho MQTT客户端。
MqttClient sampleClient = new MqttClient(broker, sign.getClientid(), persistence);
// Paho MQTT连接参数。
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setCleanSession(true);
connOpts.setKeepAliveInterval(180);
connOpts.setUserName(sign.getUsername());
connOpts.setPassword(sign.getPassword().toCharArray());
sampleClient.connect(connOpts);
System.out.println("Broker: " + broker + " Connected");
```

> **注意** 修改代码 `String broker = "ssl://" + productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com" + ":" + port;` 中的接入域名，broker值的格式为 `"ssl://" + "${对应实例下MQTT接入域名}" + ":" + port` 。
>
> 公共实例和企业版实例接入域名的格式说明，请参见查看实例终端信息。

- 发布消息。

  以下示例代码上报物模型属性LightSwitch。

```
// Paho MQTT发布消息。
String topic = "/sys/" + productKey + "/" + deviceName + "/thing/event/property/post"
;
String content = "{\"id\":\"1\",\"version\":\"1.0\",\"params\":{\"LightSwitch\":1}}";
MqttMessage message = new MqttMessage(content.getBytes());
message.setQos(0);
sampleClient.publish(topic, message);
```

  如果您使用MQTT 5.0协议通信，可添加以下示例代码，上报消息时携带自定义属性。

```
//MQTT 5.0新特性—用户自定义属性
MqttProperties properties = new MqttProperties();
List<UserProperty> userPropertys = new ArrayList<>();
userPropertys.add(new UserProperty("key1","value1"));
properties.setUserProperties(userPropertys);
//MQTT 5.0新特性—请求/响应模式
properties.setCorrelationData("requestId12345".getBytes());
properties.setResponseTopic("/" + productKey + "/" + deviceName + "/user/get");
message.setProperties(properties);
//支持MQTT 5.0的Paho SDK默认会使用Topic别名
sampleClient.publish(topic, message);
```

物模型通信数据格式，请参见设备属性、事件、服务。

如果您要使用自定义Topic通信，请参见什么是Topic。

- 订阅Topic，获取云端下发消息。

  以下示例中，订阅的是上报属性值后，物联网平台返回应答消息的Topic。

```
class MqttPostPropertyMessageListener implements IMqttMessageListener {
    @Override
    public void messageArrived(String var1, MqttMessage var2) throws Exception {
        System.out.println("reply topic  : " + var1);
        System.out.println("reply payload: " + var2.toString());
    }
}
...
// Paho MQTT消息订阅。
String topicReply = "/sys/" + productKey + "/" + deviceName + "/thing/event/property/
post_reply";
sampleClient.subscribe(topicReply, new MqttPostPropertyMessageListener());
```

关于设备、服务器和物联网平台的通信方式介绍，请参见通信方式概述。

5. 单击Build Project按钮，编译项目。

示例代码

使用Demo代码程序接入物联网平台。

1. 下载Demo代码包，并解压缩。
2. 打开IntelliJ IDEA，导入Demo包中的示例工程*aiot-java-demo*。
3. 在*src/main/java/com.aliyun.iot*下*App*或*Mqtt5App*文件中，修改设备信息为您的设备信息。

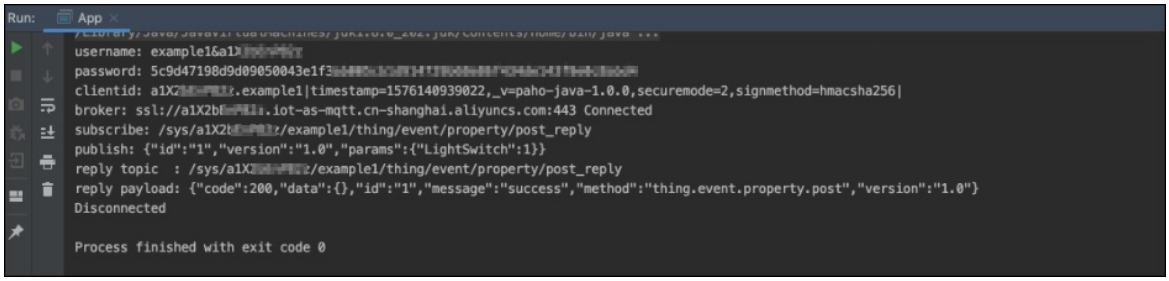
   > **说明** MQTT 3.1和3.1.1协议通信，使用*App*文件，MQTT 5.0协议通信，使用*Mqtt5App*文件。

   - 替换一下代码中productKey、deviceName和deviceSecret的值为您的设备证书信息。

```
String productKey = "${ProductKey}";
String deviceName = "${DeviceName}";
String deviceSecret = "${DeviceSecret}";
```

- 修改代码 `String broker = "ssl://" + productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com" + ":" + port;` 中的接入域名。详细说明，请参见上文“接入物联网平台”中的步骤4。

4. 运行*App*或*Mqtt5App*程序。
   运行成功日志如下图所示。



登录物联网平台控制台，可查看设备状态和日志。

- 选择**设备管理 > 设备**，可看到该设备的状态显示为**在线**。
- 选择**监控运维 > 日志服务**，可查看**云端运行日志**和**设备本地日志**日志。详情请参见云端运行日志、设备本地日志。

  如果使用*Mqtt5App*文件，可在日志详情中查看到上报的自定义属性。

错误码

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。服务端错误码说明，请参见错误排查。

1.1.3. Paho-MQTT Go接入示例

本文介绍如何调用Go语言的Paho MQTT类库，将设备接入阿里云物联网平台，并进行消息收发。

前提条件

已在物联网平台控制台，对应实例下，创建产品和设备，并获取MQTT接入域名和设备证书信息（ProductKey、DeviceName和DeviceSerect）。具体操作，请参见：

- 查看实例终端节点。
- 创建产品。
- 创建设备。

准备开发环境

安装Go语言包。

- 苹果电脑安装命令：

```
brew install go
```

- Ubuntu电脑安装命令：

```
sudo apt-get install golang-go
```

- Windows电脑请从Golang官网下载安装包安装。

> 说明 在中国内地境内，Golang官网需在VPN环境下访问。

下载Go语言Paho MQTT库

请访问Eclipse Paho Downloads了解Paho项目和支持的开发语言详情。

使用以下命令下载Go语言版本的Paho MQTT库和相关依赖的库：

```
go get github.com/eclipse/paho.mqtt.golang
go get github.com/gorilla/websocket
go get golang.org/x/net/proxy
```

接入物联网平台

1. 单击打开MqttSign.go，复制阿里云提供的计算MQTT连接参数所需的源码，然后粘贴保存为本地的*MqttSign.go*文件。

   *MqttSign.go*文件定义了用于计算设备接入物联网平台的MQTT连接参数的函数，您开发的设备端接入物联网平台程序需调用该函数，函数说明如下：

   - 原型：

     ```
     type AuthInfo struct {
         password, username, mqttClientId string;
     }
     func calculate_sign(clientId, productKey, deviceName, deviceSecret, timeStamp string)
     AuthInfo;
     ```

   - 功能：

     用于计算设备接入物联网平台的MQTT连接参数username、password和mqttClientId。

   - 输入参数：

参数	类型	说明
productKey	String	设备所属产品的ProductKey，该设备在物联网平台上的身份证书信息之一。
deviceName	String	设备名称，该设备在物联网平台上的身份证书信息之一。
deviceSecret	String	设备密钥，该设备在物联网平台上的身份证书信息之一。

参数	类型	说明
clientId	String	设备端ID。可以设置为64字符以内的字符串。建议设置为实际设备的SN码或MAC地址。
timeStamp	String	当前时间的毫秒值时间戳。

- 输出参数：

  该函数的返回值为 `AuthInfo` 结构体，包含以下参数：

参数	类型	说明
username	String	MQTT连接所需的用户名。
password	String	MQTT连接所需的密码。
mqttClientId	String	MQTT客户端ID。

2. 添加实现设备接入物联网平台的程序文件。

   您需编写程序调用*MqttSign.go*计算MQTT连接参数，实现接入物联网平台和通信。

   开发说明和代码示例如下：

   - 设置设备信息。

```
// set the device info, include product key, device name, and device secret
    var productKey string = "a1Zd7n5***"
    var deviceName string = "testdevice"
    var deviceSecret string = "UrwclDV33NaFSmk0JaBxNTqgSrJW****"
    // set timestamp, clientid, subscribe topic and publish topic
    var timeStamp string = "1528018257135"
    var clientId string = "192.168.****"
    var subTopic string = "/" + productKey + "/" + deviceName + "/user/get";
    var pubTopic string = "/" + productKey + "/" + deviceName + "/user/update";
```

   - 设置MQTT连接信息。

     调用*MqttSign.go*中定义的calculate_sign函数，根据传入的参数clientId、productKey、deviceName、deviceSecret和timeStamp计算出username、password和mqttClientId，并将这些信息都包含在opts中。

```
// set the login broker url
var raw_broker bytes.Buffer
raw_broker.WriteString("tls://")
raw_broker.WriteString(productKey)
raw_broker.WriteString(".iot-as-mqtt.cn-shanghai.aliyuncs.com:1883")
opts := MQTT.NewClientOptions().AddBroker(raw_broker.String());
// calculate the login auth info, and set it into the connection options
auth := calculate_sign(clientId, productKey, deviceName, deviceSecret, timeStamp)
opts.SetClientID(auth.mqttClientId)
opts.SetUsername(auth.username)
opts.SetPassword(auth.password)
opts.SetKeepAlive(60 * 2 * time.Second)
opts.SetDefaultPublishHandler(f)
```

注意

- 对于旧版公共实例，代码 `raw_broker.WriteString(".iot-as-mqtt.cn-shanghai.aliyuncs.com:1883")` 中的地域代码（cn-shanghai），需设置为您的物联网平台设备所在地域代码。地域代码表达方法，请参见地域列表。
- 对于新版公共实例和企业版实例，代码 `opts := MQTT.NewClientOptions().AddBroker(raw_broker.String());` 中的 `raw_broker.String()` 需设置为 `${MQTT接入地址}:1883` 。例如，`opts := MQTT.NewClientOptions().AddBroker("iot-***.mqtt.iothub.aliyuncs.com:1883");`

  获取MQTT接入地址的具体操作，请参见查看新版公共实例和企业版实例终端节点信息。

实例的详细说明，请参见实例概述。

- 调用MQTT的Connect()函数接入物联网平台。

```
// create and start a client using the above ClientOptions
c := MQTT.NewClient(opts)
if token := c.Connect(); token.Wait() && token.Error() != nil {
    panic(token.Error())
}
fmt.Print("Connect aliyun IoT Cloud Success\n");
```

- 调用Publish接口发布消息。需指定发布消息的目标Topic和消息payload。

```
// publish 5 messages to pubTopic("/a1Zd7n5****/deng/user/update")
for i := 0; i < 5; i++ {
    fmt.Println("publish msg:", i)
    text := fmt.Sprintf("ABC #%d", i)
    token := c.Publish(pubTopic, 0, false, text)
    fmt.Println("publish msg: ", text)
    token.Wait()
    time.Sleep(2 * time.Second)
}
```

通信Topic介绍，请参见什么是Topic。

- 调用Subscribe接口订阅Topic，接收云端下发的消息。

```
    // subscribe to subTopic("/a1Zd7n5***/deng/user/get") and request messages to be
delivered
    if token := c.Subscribe(subTopic, 0, nil); token.Wait() && token.Error() != nil {
        fmt.Println(token.Error())
        os.Exit(1)
    }
    fmt.Print("Subscribe topic " + subTopic + " success\n");
```

关于设备、服务器和物联网平台的通信方式介绍，请参见通信方式概述。

3. 编译项目。

示例代码

使用Demo代码程序接入物联网平台。

1. 下载示例代码包，并提取文件。

   *aiot-go-demo*中包含以下文件：

文件	说明
MqttSign.go	该文件包含以MQTT方式接入物联网平台的连接参数计算代码。*iot.go*运行时，会调用该文件中定义的calculate_sign函数，计算出连接参数username、password和mqttClientId。
iot.go	该文件包含设备与物联网平台连接和通信的逻辑代码。
x509	物联网平台的根证书，是设备接入物联网平台的必须证书。

2. 在*iot.go*中，修改设备信息为您的设备信息。

   可使用Linux vi等工具修改*iot.go*文件：

   - 将productKey、deviceName和deviceSecret替换为您的设备证书信息。
   - （可选）替换timeStamp和clientId。clientId的值可以替换为您的实际设备的SN码和MAC地址。

     这两个参数值不替换也能接入物联网平台，但实际使用时，建议您替换为实际信息。
   - 修改设备接入物联网平台的MQTT连接信息。详细说明，请参见上文接入物联网平台的步骤2。

3. 在命令行里使用以下命令运行*iot.go*：

```
go run iot.go MqttSign.go
```

运行成功，接入物联网平台的本地日志如下：

```
clientId192.168.****deviceNametestdeviceproductKeya1Zd7n5****timestamp1528018257135
1b865320fc183cc747041c9faffc9055fc45****
Connect aliyun IoT Cloud Sucess
Subscribe topic /a1Zd7n5****/testdevice/user/get success
publish msg: 0
publish msg:  ABC #0
publish msg: 1
publish msg:  ABC #1
publish msg: 2
publish msg:  ABC #2
publish msg: 3
publish msg:  ABC #3
publish msg: 4
publish msg:  ABC #4
publish msg: 5
publish msg:  ABC #5
```

登录物联网平台控制台，可查看设备状态和日志。

- 选择**设备管理 > 设备**，可看到该设备的状态显示为**在线**。
- 选择**监控运维 > 日志服务**，可查看**云端运行日志**和**设备本地日志**日志。详细内容，请参见云端运行日志、设备本地日志。

错误码

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。服务端错误码说明，请参见错误排查。

1.1.4. Paho-MQTT C#接入示例

本文介绍如何使用C#语言的Paho MQTT类库接入阿里云物联网平台，并进行物模型数据通信。

前提条件

已在物联网平台中，创建了产品和设备，并在产品的**功能定义**页签下，定义一个LightSwitch属性。

请参见创建产品、单个创建设备和单个添加物模型。

背景信息

Paho提供的MQTT C#开源代码中，已包含Visual Studio解决方案工程。工程中的每个项目针对不同的.NET平台，可生成对应的类库。

本示例中，在工程中新建一个控制台应用项目， 调用Paho的MQTT类库连接阿里云物联网平台。

准备开发环境

本示例使用的操作系统和开发工具：

- 操作系统：Windows10
- 集成开发环境：Visual Studio 2019

安装开发环境：

1. 下载Visual Studio 2019社区版，并解压缩。
2. 打开*Visual Studio Installer*，选择**.NET桌面开发**，单击**安装**。

下载Paho客户端

下载Paho MQTT for C#源代码，其中包含Visual Studio解决方案工程文件*M2MMqtt.sln*。您可使用该工程文件开发自己的设备端，具体操作，请参见下文的接入物联网平台。

您也可访问Eclipse Paho，查看Paho源代码的更多使用说明。

编写本示例Demo时，使用master分支，commit id为 `b2e64bc4485721a0bd5ae805d9f4917e8d040e81` 。

接入物联网平台

1. 单击打开MqttSign.cs，下载阿里云提供的计算MQTT连接参数所需的源码。

   *MqttSign.cs*文件中，定义了 `MqttSign` 类，类说明如下：

   - 原型：

     ```
     class MqttSign
     ```

   - 功能：

     用于计算设备接入物联网平台的MQTT连接参数username、password和clientid。

   - 成员：

类型定义	方法描述
public bool	`calculate(String productKey, String deviceName, String deviceSecret)` 根据设备的productKey、deviceName和deviceSecret计算出MQTT连接参数username、password和clientid。
public String	`getUsername()` 用于获取MQTT建连参数username。
public String	`getPassword()` 用于获取MQTT建连参数password。
public String	`getClientid()` 用于获取MQTT建连参数clientid。

2. 打开Visual Studio，导入Paho源代码中的Visual Studio解决方案文件*M2Mqtt.sln*，并创建一个应用项目。
3. 将步骤1中下载的*MqttSign.cs*文件导入到应用项目中。
4. 在应用项目中，添加实现设备接入物联网平台的程序文件。

   您需编写程序调用*MqttSign.cs*中的MqttSign类计算MQTT连接参数，实现接入物联网平台和通信。

   开发说明和代码示例如下：

   - 计算MQTT连接参数。

     调用*MqttSign.cs*中的MqttSign计算MQTT连接参数。

```
String productKey = "a1X2bEn****";
String deviceName = "example1";
String deviceSecret = "ga7XA6KdlEeiPXQPpRbAjOZXwG8y****";
// 计算MQTT连接参数。
MqttSign sign = new MqttSign();
sign.calculate(productKey, deviceName, deviceSecret);
Console.WriteLine("username: " + sign.getUsername());
Console.WriteLine("password: " + sign.getPassword());
Console.WriteLine("clientid: " + sign.getClientid());
```

- 调用Paho MQTT客户端连接物联网平台。

```
// 使用Paho连接阿里云物联网平台。
int port = 443;
String broker = productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com";
MqttClient mqttClient = new MqttClient(broker, port, true, MqttSslProtocols.TLSv1_2,
null, null);
mqttClient.Connect(sign.getClientid(), sign.getUsername(), sign.getPassword());
Console.WriteLine("Broker: " + broker + " Connected");
```

> **说明** 修改代码 `String broker = productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com";` 中值为对应实例下设备的接入域名。
>
> 公共实例和企业版实例接入域名的格式说明，请参见查看实例终端信息。

- 设备上报数据到物联网平台。

  以下示例代码上报物模型属性LightSwitch。

```
// Paho MQTT消息发布。
String topic = "/sys/" + productKey + "/" + deviceName + "/thing/event/property/post"
;
String message = "{\"id\":\"1\",\"version\":\"1.0\",\"params\":{\"LightSwitch\":0}}";
mqttClient.Publish(topic, Encoding.UTF8.GetBytes(message));
```

  物模型通信数据格式，请参见设备属性、事件、服务。

  如果您要使用自定义Topic通信，请参见什么是Topic。

- 订阅Topic，接收物联网平台下发数据。

  以下示例中，订阅的是上报属性值后，物联网平台返回应答消息的Topic。

```
// Paho MQTT消息订阅。
String topicReply = "/sys/" + productKey + "/" + deviceName + "/thing/event/property/
post_reply";
mqttClient.MqttMsgPublishReceived += MqttPostProperty_MqttMsgPublishReceived;
mqttClient.Subscribe(new string[] { topicReply }, new byte[] { MqttMsgBase.QOS_LEVEL_
AT_MOST_ONCE });
...
private static void MqttPostProperty_MqttMsgPublishReceived(object sender, uPLibrary.
Networking.M2Mqtt.Messages.MqttMsgPublishEventArgs e)
{
    Console.WriteLine("reply topic  :" + e.Topic);
    Console.WriteLine("reply payload:" + e.Message.ToString());
}
```

关于设备、服务器和物联网平台的通信方式介绍，请参见通信方式概述。

5. 编译项目。

示例Demo

使用Demo代码程序接入物联网平台。

1. 下载Demo代码包，然后解压到文件夹*aiot-csharp-demo*。

   文件夹*aiot-csharp-demo\paho.mqtt.m2mqtt-master\aiot-csharp-demo*中，包含了设备接入物联网平台，并上报物模型属性的完整程序。

文件	说明
MqttSign.cs	阿里云提供的MQTT建连参数生成源代码。*Program.cs*运行时，会调用该文件中定义的MqttSign()函数，计算出连接参数username、password和clientId。
Program.cs	该文件包含设备与物联网平台连接，并上报属性数据的逻辑代码。

2. 打开Visual Studio 2019社区版，选择**打开项目或解决方案**，打开*aiot-csharp-demo\paho.mqtt.m2mqtt-master\M2Mqtt.sln*文件。

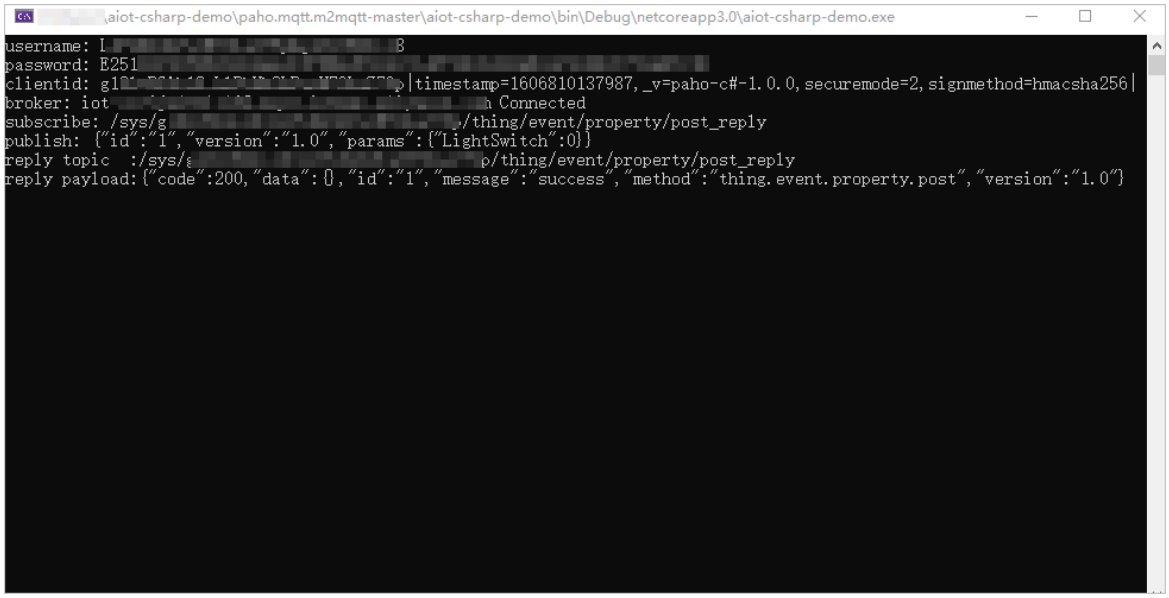
   Visual Studio中即可导入*aiot-csharp-demo*项目文件。

3. 在*Program.cs*中，修改设备信息为您的设备信息。
   - 替换一下代码中productKey、deviceName和deviceSecret的值为您的设备证书信息。

     ```
     String productKey = "${ProductKey}";
     String deviceName = "${DeviceName}";
     String deviceSecret = "${DeviceSecret}";
     ```

   - 修改代码 `String broker = productKey + ".iot-as-mqtt.cn-shanghai.aliyuncs.com";` 中的接入域名。详细说明，请参见上文“接入物联网平台”中的步骤4。

4. 将*aiot-csharp-demo*设为启动项目，然后运行，将设备接入物联网平台。



接入成功后，本地日志中包含连接成功、数据上报成功和订阅消息成功的内容。

```
...
broker: a1X2bEn****.iot-as-mqtt.cn-shanghai.aliyuncs.com Connected
...
publish: {"id":"1","version":"1.0","params":{"LightSwitch":0}}
...
subscribe: /sys/a1X2bEn****/example1/thing/event/property/post_reply
...
```

登录物联网平台控制台，可查看设备状态和日志。

- 选择**设备管理 > 设备**，可看到该设备的状态显示为**在线**。
- 选择**监控运维 > 日志服务**，可查看**云端运行日志**和**设备本地日志**日志。详细内容，请参见云端运行日志、设备本地日志。

错误码

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。服务端错误码说明，请参见错误排查。

1.1.5. Paho-MQTT Android接入示例

本文介绍如何使用Paho Android Service接入阿里云物联网平台，并进行数据收发。

前提条件

已在物联网平台控制台，对应实例下，创建产品和设备，并获取MQTT接入域名和设备证书信息（ProductKey、DeviceName和DeviceSerect）。具体操作，请参见：

- 查看实例终端节点。
- 创建产品。
- 创建设备。

背景信息

Paho Android Service是一个基于Java语言的Paho MQTT库开发的MQTT客户端服务包。

准备开发环境

本示例使用的Android Studio版本为3.5.1，gradle版本为3.5.1。

请访问Android Studio官网下载Android Studio。Android开发相关教程，请查看Android Studio官方文档。

安装Paho Android Client

1. 创建一个新的Android工程。
2. 在gradle文件中，添加Paho Android Client依赖。本示例使用1.1.1版本的PahoAndroidClient，需添加以下依赖：
   - 在工程*build.gradle*中，添加Paho仓库地址。本示例使用release仓库。

     ```
     repositories {
         maven {
             url "https://repo.eclipse.org/content/repositories/paho-releases/"
         }
     }
     ```

   - 在应用*build.gradle*中，添加Paho Android Service。本示例中，使用1.1.1 release版本的Paho服务，底层基于paho.client.mqttv3-1.1.0版本。

     ```
     dependencies {
         implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
         implementation 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'
     }
     ```

3. 为了使App能够绑定到Paho Android Service，需要在*AndroidManifest.xml*中添加以下信息：
   - 声明以下服务：

     ```
     <!-- Mqtt Service -->
     <service android:name="org.eclipse.paho.android.service.MqttService">
     </service>
     ```

   - 添加Paho MQTT Service所需的权限。

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

接入物联网平台

1. 单击打开AiotMqttOption.java，下载阿里云提供的计算MQTT连接参数所需的源码。

   *AiotMqttOption.java*文件中定义了AiotMqttOption()类，类说明如下：

   - 原型：

     ```
     class AiotMqttOption
     ```

   - 功能：

     用于计算设备接入物联网平台的MQTT连接参数username、password和clientid。

- 成员：

类型定义	方法描述
public AiotMqttOption	`getMqttOption(String productKey, String deviceName, String deviceSecret)` 根据设备的productKey、deviceName和deviceSecret计算出MQTT连接参数username、password和clientid。
public String	`getUsername()` 用于获取MQTT建连参数username。
public String	`getPassword()` 用于获取MQTT建连参数password。
public String	`getClientid()` 用于获取MQTT建连参数clientid。

2. 将*AiotMqttOption.java*导入Android项目。
3. 在Android项目中，添加实现设备接入物联网平台的程序文件。

   您需编写程序调用*AiotMqttOption.java*中的AiotMqttOption()类计算MQTT连接参数，实现接入物联网平台和通信。

   开发说明和示例代码如下：

   - 计算MQTT连接参数clientId、username和password，并将username和password设置到MqttConnectOptions对象中。

```
final private String PRODUCTKEY = "a11xsrW****";
final private String DEVICENAME = "paho_android";
final private String DEVICESECRET = "tLMT9QWD36U2SArglGqcHCDK9rK9****";
/* 获取MQTT连接信息clientId、username、password。 */
AiotMqttOption aiotMqttOption = new AiotMqttOption().getMqttOption(PRODUCTKEY, DEVICE
NAME, DEVICESECRET);
if (aiotMqttOption == null) {
    Log.e(TAG, "device info error");
} else {
    clientId = aiotMqttOption.getClientId();
    userName = aiotMqttOption.getUsername();
    passWord = aiotMqttOption.getPassword();
}
/* 创建MqttConnectOptions对象，并配置username和password。 */
MqttConnectOptions mqttConnectOptions = new MqttConnectOptions();
mqttConnectOptions.setUserName(userName);
mqttConnectOptions.setPassword(passWord.toCharArray());
```

   - 接入物联网平台。

     创建一个MqttAndroidClient对象，设置回调接口，然后使用mqttConnectOptions调用connect方法，即可建立连接。

```
/* 创建MqttAndroidClient对象，并设置回调接口。 */
mqttAndroidClient = new MqttAndroidClient(getApplicationContext(), host, clientId);
mqttAndroidClient.setCallback(new MqttCallback() {
    @Override
    public void connectionLost(Throwable cause) {
        Log.i(TAG, "connection lost");
    }
    @Override
    public void messageArrived(String topic, MqttMessage message) throws Exception {
        Log.i(TAG, "topic: " + topic + ", msg: " + new String(message.getPayload()));
    }
    @Override
    public void deliveryComplete(IMqttDeliveryToken token) {
        Log.i(TAG, "msg delivered");
    }
});
/* 建立MQTT连接。 */
try {
    mqttAndroidClient.connect(mqttConnectOptions, null, new IMqttActionListener() {
        @Override
        public void onSuccess(IMqttToken asyncActionToken) {
            Log.i(TAG, "connect succeed");
            subscribeTopic(SUB_TOPIC);
        }
        @Override
        public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
            Log.i(TAG, "connect failed");
        }
    });
} catch (MqttException e) {
    e.printStackTrace();
}
```

- 发布消息。封装publish方法，用于向Topic `/${prodcutKey}/${deviceName}/user/update` 发布指定payload的消息。

```
public void publishMessage(String payload) {
    try {
        if (mqttAndroidClient.isConnected() == false) {
            mqttAndroidClient.connect();
        }
        MqttMessage message = new MqttMessage();
        message.setPayload(payload.getBytes());
        message.setQos(0);
        mqttAndroidClient.publish(PUB_TOPIC, message,null, new IMqttActionListener()
{
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Log.i(TAG, "publish succeed!");
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Log.i(TAG, "publish failed!");
            }
        });
    } catch (MqttException e) {
        Log.e(TAG, e.toString());
        e.printStackTrace();
    }
}
```

通信Topic介绍，请参见什么是Topic。

- 封装subscribe方法，用于实现订阅指定Topic，获取云端下发的消息。

```
public void subscribeTopic(String topic) {
    try {
        mqttAndroidClient.subscribe(topic, 0, null, new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken) {
                Log.i(TAG, "subscribed succeed");
            }
            @Override
            public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
                Log.i(TAG, "subscribed failed");
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

关于设备、服务器和物联网平台的通信方式介绍，请参见通信方式概述。

4. 编译项目。

示例Demo

使用Demo代码程序接入物联网平台。

1. 下载代码Demo包，并解压缩。

2. 将*aiot-android-demo*导入Android Studio。
3. 在*app/src/main/java/com.linkkit.aiot_android_demo*下的*MainActivity*文件中，替换设备信息为您的设备信息。
   - 替换PRODUCTKEY、DEVICENAME和DEVICESECRET的值为您的设备证书信息。
   - 修改代码 `final String host = "tcp://" + PRODUCTKEY + ".iot-as-mqtt.cn-shanghai.aliyuncs.com:443";` 中的值为对应的接入域名。
     - 对于企业版实例： `final String host = "tcp://" + "${企业版实例下MQTT接入域名}"` 。

       您可登录物联网平台控制台，在**实例概览**页，找到并单击对应实例，进入**实例详情**页面，单击右上角的**查看开发配置**获取。具体操作，请参见查看实例终端节点。
     - 对于公共实例：

       替换地域代码（cn-shanghai）为您的物联网平台设备所在地域代码。地域代码的表达方法，请参见地域和可用区。
4. 构建应用，并运行。
   运行成功后，可在Logcat中查看本地日志。

```
2019-12-04 19:44:01.824 5952-5987/com.linkkit.aiot_android_demo W/OpenGLRenderer: Faile
d to choose config with EGL_SWAP_BEHAVIOR_PRESERVED, retrying without...
2019-12-04 19:44:01.829 5952-5987/com.linkkit.aiot_android_demo D/EGL_emulation: eglCre
ateContext: 0xec073240: maj 3 min 0 rcv 3
2019-12-04 19:44:01.830 5952-5987/com.linkkit.aiot_android_demo D/EGL_emulation: eglMak
eCurrent: 0xec073240: ver 3 0 (tinfo 0xec09b470)
2019-12-04 19:44:01.852 5952-5987/com.linkkit.aiot_android_demo W/Gralloc3: mapper 3.x
is not supported
2019-12-04 19:44:01.854 5952-5987/com.linkkit.aiot_android_demo D/HostConnection: creat
eUnique: call
...
...
2019-12-04 19:44:01.860 5952-5987/com.linkkit.aiot_android_demo D/eglCodecCommon: alloc
ate: Ask for block of size 0x1000
2019-12-04 19:44:01.861 5952-5987/com.linkkit.aiot_android_demo D/eglCodecCommon: alloc
ate: ioctl allocate returned offset 0x3ff706000 size 0x2000
2019-12-04 19:44:01.897 5952-5987/com.linkkit.aiot_android_demo D/EGL_emulation: eglMak
eCurrent: 0xec073240: ver 3 0 (tinfo 0xec09b470)
2019-12-04 19:44:02.245 5952-6023/com.linkkit.aiot_android_demo D/AlarmPingSender: Regi
ster alarmreceiver to MqttServiceMqttService.pingSender.a11xsrW****.paho_android|timest
amp=1575459841629,_v=sdk-android-1.0.0,securemode=2,signmethod=hmacsha256|
2019-12-04 19:44:02.256 5952-6023/com.linkkit.aiot_android_demo D/AlarmPingSender: Sche
dule next alarm at 1575459902256
2019-12-04 19:44:02.256 5952-6023/com.linkkit.aiot_android_demo D/AlarmPingSender: Alar
m scheule using setExactAndAllowWhileIdle, next: 60000
2019-12-04 19:44:02.272 5952-5952/com.linkkit.aiot_android_demo I/AiotMqtt: connect suc
ceed
2019-12-04 19:44:02.301 5952-5952/com.linkkit.aiot_android_demo I/AiotMqtt: subscribed
succeed
```

登录物联网平台控制台，可查看设备状态和日志。

- 选择**设备管理 > 设备**，可看到该设备的状态显示为**在线**。
- 选择**监控运维 > 日志服务**，可查看**云端运行日志**和**设备本地日志**日志。详细内容，请参见云端运

行日志、设备本地日志。

错误码

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。服务端错误码说明，请参见错误排查。

1.1.6. 错误排查

如果设备通过MQTT协议接入物联网平台失败，请根据错误码排查问题。

阿里云物联网平台使用的是标准的MQTT协议。了解MQTT协议，请参见MQTT 3.1或3.1.1标准协议文档和MQTT 5.0标准协议文档。

服务端返回码说明如下。

- MQTT 3.1和3.1.1

返回码	返回信息	原因
0	0x00 Connection Accepted	连接成功。
1	0x01 Connection Refused, unacceptable protocol version	服务器不支持设备端请求的MQTT协议版本。
2	0x02 Connection Refused, identifier rejected	clientId参数格式错误，不符合物联网平台规定的格式。例如参数值超出长度限制、扩展参数格式错误等。
3	0x03 Connection Refused, Server unavailable	网络连接已建立成功，但MQTT服务不可用。
4	0x04 Connection Refused, bad user name or password	username或password格式错误。
5	0x05 Connection Refused, not authorized	设备未经授权。

- MQTT 5.0

返回码	返回信息	原因
0	0x00 Success	连接成功。
128	0x80 Unspecified error	未指定错误。
129	0x81 Malformed Packet	畸形报文。
130	0x82 Protocol Error	协议错误。
132	0x84 Unsupported Protocol Version	不支持的协议版本。
136	0x88 Server unavailable	服务器不可用。
137	0x89 Server busy	服务器繁忙。

返回码	返回信息	原因
138	0x8A Banned	禁止访问。
140	0x8C Bad authentication method	错误验证方法。
141	0x8D Keep Alive timeout	保活超时。
144	0x90 Topic Name invalid	Topic名无效。
147	0x93 Receive Maximum exceeded	超出接收最大值。
148	0x94 Topic Alias invalid	Topic别名无效。
149	0x95 Packet too large	报文长度超出限制。
150	0x96 Message rate too high	消息传输速率太高。
151	0x97 Quota exceeded	超出限额。
152	0x98 Administrative action	管理行为。
153	0x99 Payload format invalid	Payload格式无效。
154	0x9A Retain not supported	不支持消息保留。
155	0x9B QoS not supported	不支持的QoS。
156	0x9C Use another server	使用另一台服务器。
157	0x9D Server moved	服务器被移除。
158	0x9E Shared Subscription not supported	不支持的共享订阅。
159	0x9F Connection rate exceeded	超出连接速率。

1.2. CoAP客户端对称加密接入示例

本文提供设备通过CoAP协议接入物联网平台的示例代码。

背景信息

CoAP协议适用于资源受限的低功耗设备，尤其是NB-IoT的设备。配置设备通过CoAP协议与物联网平台连接并通信的说明，请参见CoAP连接通信。

> **说明** 目前仅华东2（上海）地域支持设备通过CoAP通道接入物联网平台。

配置设备通过CoAP协议接入物联网平台时，需要填写相应的参数，计算设备端签名，步骤较为复杂。为了便于您理解相关配置，本文提供基于Californium框架的接入示例代码。本示例中，为保证数据安全，使用对称加密。

开发环境

本文使用Java开发环境：

- 操作系统：Windows10
- JDK版本：JDK8
- 集成开发环境：IntelliJ IDEA社区版

pom.xml配置

在*pom.xml*文件中，添加以下依赖，引入Californium开源框架、Apache commons工具包和阿里云fastjson包。

```
<dependency>
  <groupId>org.eclipse.californium</groupId>
  <artifactId>californium-core</artifactId>
  <version>2.0.0-M17</version>
</dependency>
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.5</version>
</dependency>
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.13</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.61</version>
</dependency>
```

示例代码

```
/*
 * Copyright © 2019 Alibaba. All rights reserved.
 */
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.DecoderException;
import org.apache.commons.codec.binary.Hex;
import org.apache.commons.lang3.RandomUtils;
import org.eclipse.californium.core.CoapClient;
import org.eclipse.californium.core.CoapResponse;
import org.eclipse.californium.core.Utils;
```

```
import org.eclipse.californium.core.coap.CoAP;
import org.eclipse.californium.core.coap.CoAP.Code;
import org.eclipse.californium.core.coap.CoAP.Type;
import org.eclipse.californium.core.coap.MediaTypeRegistry;
import org.eclipse.californium.core.coap.Option;
import org.eclipse.californium.core.coap.OptionNumberRegistry;
import org.eclipse.californium.core.coap.OptionSet;
import org.eclipse.californium.core.coap.Request;
import org.eclipse.californium.elements.exception.ConnectorException;
import com.alibaba.fastjson.JSONObject;
/**
 * CoAP客户端连接阿里云物联网平台，基于eclipse californium开发。
 * 自主接入开发流程及参数填写，请参见《CoAP连接通信》的使用对称加密自主接入章节。
 */
public class IotCoapClientWithAes {
    // ====================需要用户填写的参数，开始。==========================
    // 地域ID，以华东2（上海）为例。
    private static String regionId = "cn-shanghai";
    // 产品productKey。
    private static String productKey = "您的设备productKey";
    // 设备名成deviceName。
    private static String deviceName = "您的设备deviceName";
    // 设备密钥deviceSecret。
    private static String deviceSecret = "您的设备deviceSecret";
    // ====================需要用户填写的参数，结束。==========================
    // 定义加密方式，MAC算法可选以下算法：HmacMD5、HmacSHA1，需与signmethod一致。
    private static final String HMAC_ALGORITHM = "hmacsha1";
    // CoAP接入地址，对称加密端口号是5682。
    private static String serverURI = "coap://" + productKey + ".coap." + regionId + ".link
.aliyuncs.com:5682";
    // 发送消息用的Topic。需要在控制台自定义Topic，设备操作权限需选择为“发布”。
    private static String updateTopic = "/" + productKey + "/" + deviceName + "/user/update
";
    // token option
    private static final int COAP2_OPTION_TOKEN = 2088;
    // seq option
    private static final int COAP2_OPTION_SEQ = 2089;
    // 加密算法sha256。
    private static final String SHA_256 = "SHA-256";
    private static final int DIGITAL_16 = 16;
    private static final int DIGITAL_48 = 48;
    // CoAP客户端。
    private CoapClient coapClient = new CoapClient();
    // token有效期7天，失效后需要重新获取。
    private String token = null;
    private String random = null;
    @SuppressWarnings("unused")
    private long seqOffset = 0;
    /**
     * 初始化CoAP客户端。
     *
     * @param productKey，产品key。
     * @param deviceName，设备名称。
     * @param deviceSecret ，设备密钥。
```

```
     */
    public void connect(String productKey, String deviceName, String deviceSecret) {
        try {
            // 认证uri，/auth。
            String uri = serverURI + "/auth";
            // 只支持POST方法。
            Request request = new Request(Code.POST, Type.CON);
            // 设置option。
            OptionSet optionSet = new OptionSet();
            optionSet.addOption(new Option(OptionNumberRegistry.CONTENT_FORMAT, MediaTypeRe
gistry.APPLICATION_JSON));
            optionSet.addOption(new Option(OptionNumberRegistry.ACCEPT, MediaTypeRegistry.A
PPLICATION_JSON));
            request.setOptions(optionSet);
            // 设置认证uri。
            request.setURI(uri);
            // 设置认证请求payload。
            request.setPayload(authBody(productKey, deviceName, deviceSecret));
            // 发送认证请求。
            CoapResponse response = coapClient.advanced(request);
            System.out.println(Utils.prettyPrint(response));
            System.out.println();
            // 解析请求响应。
            JSONObject json = JSONObject.parseObject(response.getResponseText());
            token = json.getString("token");
            random = json.getString("random");
            seqOffset = json.getLongValue("seqOffset");
        } catch (ConnectorException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * 发送消息。
     *
     * @param topic，发送消息的Topic。
     * @param payload，消息内容。
     */
    public void publish(String topic, byte[] payload) {
        try {
            // 消息发布uri，/topic/${topic}。
            String uri = serverURI + "/topic" + topic;
            // AES加密seq，seq=RandomUtils.nextInt()。
            String shaKey = encod(deviceSecret + "," + random);
            byte[] keys = Hex.decodeHex(shaKey.substring(DIGITAL_16, DIGITAL_48));
            byte[] seqBytes = encrypt(String.valueOf(RandomUtils.nextInt()).getBytes(Standa
rdCharsets.UTF_8), keys);
            // 只支持POST方法。
            Request request = new Request(CoAP.Code.POST, CoAP.Type.CON);
            // 设置option。
            OptionSet optionSet = new OptionSet();
            optionSet.addOption(new Option(OptionNumberRegistry.CONTENT_FORMAT, MediaTypeRe
gistry.APPLICATION_JSON));
```

```
            optionSet.addOption(new Option(OptionNumberRegistry.ACCEPT, MediaTypeRegistry.A
PPLICATION_JSON));
            optionSet.addOption(new Option(COAP2_OPTION_TOKEN, token));
            optionSet.addOption(new Option(COAP2_OPTION_SEQ, seqBytes));
            request.setOptions(optionSet);
            // 设置消息发布uri。
            request.setURI(uri);
            // 设置消息payload。
            request.setPayload(encrypt(payload, keys));
            // 发送消息。
            CoapResponse response = coapClient.advanced(request);
            System.out.println(Utils.prettyPrint(response));
            // 解析消息发送结果。
            String result = null;
            if (response.getPayload() != null) {
                result = new String(decrypt(response.getPayload(), keys));
            }
            System.out.println("payload: " + result);
            System.out.println();
        } catch (ConnectorException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (DecoderException e) {
            e.printStackTrace();
        }
    }
    /**
     * 生成认证请求内容。
     *
     * @param productKey，产品key。
     * @param deviceName，设备名字。
     * @param deviceSecret，设备密钥。
     * @return 认证请求。
     */
    private String authBody(String productKey, String deviceName, String deviceSecret) {
        // 构建认证请求。
        JSONObject body = new JSONObject();
        body.put("productKey", productKey);
        body.put("deviceName", deviceName);
        body.put("clientId", productKey + "." + deviceName);
        body.put("timestamp", String.valueOf(System.currentTimeMillis()));
        body.put("signmethod", HMAC_ALGORITHM);
        body.put("seq", DIGITAL_16);
        body.put("sign", sign(body, deviceSecret));
        System.out.println("----- auth body -----");
        System.out.println(body.toJSONString());
        return body.toJSONString();
    }
    /**
     * 设备端签名。
     *
     * @param params，签名参数。
     * @param deviceSecret，设备密钥。
```

```
     * @return 签名十六进制字符串。
     */
    private String sign(JSONObject params, String deviceSecret) {
        // 请求参数按字典顺序排序。
        Set<String> keys = getSortedKeys(params);
        // sign、signmethod、version、resources除外。
        keys.remove("sign");
        keys.remove("signmethod");
        keys.remove("version");
        keys.remove("resources");
        // 组装签名明文。
        StringBuffer content = new StringBuffer();
        for (String key : keys) {
            content.append(key);
            content.append(params.getString(key));
        }
        // 计算签名。
        String sign = encrypt(content.toString(), deviceSecret);
        System.out.println("sign content=" + content);
        System.out.println("sign result=" + sign);
        return sign;
    }
    /**
     * 获取JSON对象排序后的key集合。
     *
     * @param json，需要排序的JSON对象。
     * @return 排序后的key集合。
     */
    private Set<String> getSortedKeys(JSONObject json) {
        SortedMap<String, String> map = new TreeMap<String, String>();
        for (String key : json.keySet()) {
            String vlaue = json.getString(key);
            map.put(key, vlaue);
        }
        return map.keySet();
    }
    /**
     * 使用HMAC_ALGORITHM加密。
     *
     * @param content，明文。
     * @param secret，密钥。
     * @return 密文。
     */
    private String encrypt(String content, String secret) {
        try {
            byte[] text = content.getBytes(StandardCharsets.UTF_8);
            byte[] key = secret.getBytes(StandardCharsets.UTF_8);
            SecretKeySpec secretKey = new SecretKeySpec(key, HMAC_ALGORITHM);
            Mac mac = Mac.getInstance(secretKey.getAlgorithm());
            mac.init(secretKey);
            return Hex.encodeHexString(mac.doFinal(text));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
```

```
        }
    }
    /**
     * SHA-256
     *
     * @param str，待加密的报文。
     */
    private String encod(String str) {
        MessageDigest messageDigest;
        String encdeStr = "";
        try {
            messageDigest = MessageDigest.getInstance(SHA_256);
            byte[] hash = messageDigest.digest(str.getBytes(StandardCharsets.UTF_8));
            encdeStr = Hex.encodeHexString(hash);
        } catch (NoSuchAlgorithmException e) {
            System.out.println(String.format("Exception@encod: str=%s;", str));
            e.printStackTrace();
            return null;
        }
        return encdeStr;
    }
    // AES加解密算法。
    private static final String IV = "543yhjy97ae7fyfg";
    private static final String TRANSFORM = "AES/CBC/PKCS5Padding";
    private static final String ALGORITHM = "AES";
    /**
     * key length = 16 bits
     */
    private byte[] encrypt(byte[] content, byte[] key) {
        return encrypt(content, key, IV);
    }
    /**
     * key length = 16 bits
     */
    private byte[] decrypt(byte[] content, byte[] key) {
        return decrypt(content, key, IV);
    }
    /**
     * aes 128 cbc key length = 16 bits
     */
    private byte[] encrypt(byte[] content, byte[] key, String ivContent) {
        try {
            SecretKeySpec keySpec = new SecretKeySpec(key, ALGORITHM);
            Cipher cipher = Cipher.getInstance(TRANSFORM);
            IvParameterSpec iv = new IvParameterSpec(ivContent.getBytes(StandardCharsets.UT
F_8));
            cipher.init(Cipher.ENCRYPT_MODE, keySpec, iv);
            return cipher.doFinal(content);
        } catch (Exception ex) {
            System.out.println(
                    String.format("AES encrypt error, %s, %s, %s", content, Hex.encodeHex(k
ey), ex.getMessage()));
            return null;
        }
    }
```

```
    /**
     * aes 128 cbc key length = 16 bits
     */
    private byte[] decrypt(byte[] content, byte[] key, String ivContent) {
        try {
            SecretKeySpec keySpec = new SecretKeySpec(key, ALGORITHM);
            Cipher cipher = Cipher.getInstance(TRANSFORM);
            IvParameterSpec iv = new IvParameterSpec(ivContent.getBytes(StandardCharsets.UT
F_8));
            cipher.init(Cipher.DECRYPT_MODE, keySpec, iv);
            return cipher.doFinal(content);
        } catch (Exception ex) {
            System.out.println(String.format("AES decrypt error, %s, %s, %s", Hex.encodeHex
(content),
                    Hex.encodeHex(key), ex.getMessage()));
            return null;
        }
    }
    public static void main(String[] args) throws InterruptedException {
        IotCoapClientWithAes client = new IotCoapClientWithAes();
        client.connect(productKey, deviceName, deviceSecret);
        client.publish(updateTopic, "hello coap".getBytes(StandardCharsets.UTF_8));
        client.publish(updateTopic, new byte[] { 0x01, 0x02, 0x03, 0x05 });
    }
}
```

1.3. HTTP客户端接入示例

本文提供设备通过HTTP协议接入物联网平台的示例代码。

物联网平台支持设备通过HTTP协议接入。相关配置说明，请参见HTTPS连接通信。

本文提供基于Java HTTP的接入示例代码，介绍如何配置设备通过HTTP协议接入物联网平台的请求参数，计算设备端签名等。

> **说明** 目前仅华东2（上海）地域支持HTTP接入。

pom.xml配置

在*pom.xml*文件中，添加以下依赖，引入阿里fastjson包。

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.68</version>
</dependency>
```

示例代码

以下为设备通过HTTP协议接入物联网平台和消息通信的主体代码示例。

```
/*
 * Copyright © 2010 Alibaba. All rights reserved.
```

```
 * Copyright © 2019 Alibaba. All rights reserved.
 */
package com.aliyun.iot.demo;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import com.alibaba.fastjson.JSONObject;
/**
 * 设备使用HTTP协议接入阿里云物联网平台。
 * 协议规范说明，请参见《HTTP协议规范》。
 * 数据格式，请参见《HTTP连接通信》。
 */
public class IotHttpClient {
    // 地域ID，以华东2（上海）为例。
    private static String regionId = "cn-shanghai";
    // 定义加密方式，MAC算法可选以下算法：HmacMD5、HmacSHA1，需和signmethod一致。
    private static final String HMAC_ALGORITHM = "hmacsha1";
    // token有效期7天，失效后需要重新获取。
    private String token = null;
    /**
     * 初始化HTTP客户端。
     *
     * @param productKey，产品key。
     * @param deviceName，设备名称。
     * @param deviceSecret，设备密钥。
     */
    public void conenct(String productKey, String deviceName, String deviceSecret) {
        try {
            // 注册地址。
            URL url = new URL("https://iot-as-http." + regionId + ".aliyuncs.com/auth");
            HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-type", "application/json");
            conn.setDoOutput(true);
            conn.setDoInput(true);
            // 获取URLConnection对象对应的输出流。
            PrintWriter out = new PrintWriter(conn.getOutputStream());
            // 发送请求参数。
            out.print(authBody(productKey, deviceName, deviceSecret));
            // flush输出流的缓冲。
            out.flush();
            // 获取URLConnection对象对应的输入流。
            BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStrea
m()));
            // 读取URL的响应。
            String result = "";
```

```
            String line = "";
            while ((line = in.readLine()) != null) {
                result += line;
            }
            System.out.println("----- auth result -----");
            System.out.println(result);
            // 关闭输入输出流。
            in.close();
            out.close();
            conn.disconnect();
            // 获取token。
            JSONObject json = JSONObject.parseObject(result);
            if (json.getIntValue("code") == 0) {
                token = json.getJSONObject("info").getString("token");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * 发送消息。
     *
     * @param topic，发送消息的Topic。
     * @param payload，消息内容。
     */
    public void publish(String topic, byte[] payload) {
        try {
            // 注册地址。
            URL url = new URL("https://iot-as-http." + regionId + ".aliyuncs.com/topic" + t
opic);
            HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
            conn.setRequestMethod("POST");
            conn.setRequestProperty("Content-type", "application/octet-stream");
            conn.setRequestProperty("password", token);
            conn.setDoOutput(true);
            conn.setDoInput(true);
            // 获取URLConnection对象对应的输出流。
            BufferedOutputStream out = new BufferedOutputStream(conn.getOutputStream());
            out.write(payload);
            out.flush();
            // 获取URLConnection对象对应的输入流。
            BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStrea
m()));
            // 读取URL的响应。
            String result = "";
            String line = "";
            while ((line = in.readLine()) != null) {
                result += line;
            }
            System.out.println("----- publish result -----");
            System.out.println(result);
            // 关闭输入输出流。
            in.close();
            out.close();
```

```
            conn.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    /**
     * 生成认证请求内容。
     *
     * @param params，认证参数。
     * @return 认证请求消息体。
     */
    private String authBody(String productKey, String deviceName, String deviceSecret) {
        // 构建认证请求。
        JSONObject body = new JSONObject();
        body.put("productKey", productKey);
        body.put("deviceName", deviceName);
        body.put("clientId", productKey + "." + deviceName);
        body.put("timestamp", String.valueOf(System.currentTimeMillis()));
        body.put("signmethod", HMAC_ALGORITHM);
        body.put("version", "default");
        body.put("sign", sign(body, deviceSecret));
        System.out.println("----- auth body -----");
        System.out.println(body.toJSONString());
        return body.toJSONString();
    }
    /**
     * 设备端签名。
     *
     * @param params，签名参数。
     * @param deviceSecret，设备密钥。
     * @return 签名十六进制字符串。
     */
    private String sign(JSONObject params, String deviceSecret) {
        // 请求参数按字典顺序排序。
        Set<String> keys = getSortedKeys(params);
        // sign、signmethod和version除外。
        keys.remove("sign");
        keys.remove("signmethod");
        keys.remove("version");
        // 组装签名明文。
        StringBuffer content = new StringBuffer();
        for (String key : keys) {
            content.append(key);
            content.append(params.getString(key));
        }
        // 计算签名。
        String sign = encrypt(content.toString(), deviceSecret);
        System.out.println("sign content=" + content);
        System.out.println("sign result=" + sign);
        return sign;
    }
    /**
     * 获取JSON对象排序后的key集合。
     *
```

```
     * @param json，需要排序的JSON对象。
     * @return 排序后的key集合。
     */
    private Set<String> getSortedKeys(JSONObject json) {
        SortedMap<String, String> map = new TreeMap<String, String>();
        for (String key : json.keySet()) {
            String vlaue = json.getString(key);
            map.put(key, vlaue);
        }
        return map.keySet();
    }
    /**
     * 使用HMAC_ALGORITHM加密。
     *
     * @param content，明文。
     * @param secret，密钥。
     * @return 密文。
     */
    private String encrypt(String content, String secret) {
        try {
            byte[] text = content.getBytes(StandardCharsets.UTF_8);
            byte[] key = secret.getBytes(StandardCharsets.UTF_8);
            SecretKeySpec secretKey = new SecretKeySpec(key, HMAC_ALGORITHM);
            Mac mac = Mac.getInstance(secretKey.getAlgorithm());
            mac.init(secretKey);
            return byte2hex(mac.doFinal(text));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    /**
     * 二进制转十六进制字符串。
     *
     * @param b，二进制数组。
     * @return 十六进制字符串。
     */
    private String byte2hex(byte[] b) {
        StringBuffer sb = new StringBuffer();
        for (int n = 0; b != null && n < b.length; n++) {
            String stmp = Integer.toHexString(b[n] & 0XFF);
            if (stmp.length() == 1) {
                sb.append('0');
            }
            sb.append(stmp);
        }
        return sb.toString().toUpperCase();
    }
    public static void main(String[] args) {
        String productKey = "您的productKey";
        String deviceName = "您的deviceName";
        String deviceSecret = "您的deviceSecret";
        IotHttpClient client = new IotHttpClient();
        client.conenct(productKey, deviceName, deviceSecret);
```

```
        // 发送消息的Topic。可在控制台自定义，设备有发布权限。
        String updateTopic = "/" + productKey + "/" + deviceName + "/user/update";
        client.publish(updateTopic, "hello http".getBytes(StandardCharsets.UTF_8));
        client.publish(updateTopic, new byte[] { 0x01, 0x02, 0x03 });
    }
}
```

1.4. MQTT-WebSocket认证接入示例

本文提供Node.js语言的示例代码，介绍如何使设备通过MQTT-WebSocket通道接入物联网平台。

背景信息

使用WebSocket方式接入设备的详细说明，请参见MQTT-WebSocket连接通信。

本文以旧版公共实例下设备为例，使用物联网平台提供的设备端Link SDK，模拟设备接入和上下行通信过程。

> **说明** 设备端Link SDK已配置TLS加密，您无需自行配置。

操作步骤

1. 在Windows系统或Linux系统下载并安装Node.js。本文以Windows 10（64位）系统为例，下载安装包node-v14.15.1-x64.msi。
2. 安装成功后，打开CMD窗口，通过以下命令查看node版本。

   ```
   node --version
   ```

   显示如下版本号，表示安装成功。

   ```
   v14.15.1
   ```

3. 在本地计算机创建一个JavaScript文件（例如*iot_device.js*），用来存放Node.js示例代码。

   Node.js示例代码如下：

```
const iot = require('alibabacloud-iot-device-sdk');
// 设备证书信息。
const productKey = 'a1W***';
const deviceName = 'device2';
const deviceSecret = 'ff01e59d1a***';
// 新版公共实例和企业版实例，必须填写实例ID，旧版公实例无需填写。
const instanceId = '';
// 当前产品和设备所属地域的ID。
const region = 'cn-shanghai';
const brokerUrl = instanceId
  ? `wss://${instanceId}.mqtt.iothub.aliyuncs.com:443`
  : `wss://${productKey}.iot-as-mqtt.${region}.aliyuncs.com:443`;
const device = iot.device({
  productKey: `${productKey}`,
  deviceName: `${deviceName}`,
  deviceSecret: `${deviceSecret}`,
  brokerUrl,
  tls: true,
});
// 监听connect事件：建立MQTT连接，订阅自定义Topic，通过自定义Topic向物联网平台发送消息。
device.on('connect', () => {
  device.subscribe(`/${productKey}/${deviceName}/user/get`);
  console.log('connect successfully!');
  device.publish(`/${productKey}/${deviceName}/user/update`, 'hello world!');
});
// 监听message事件。
device.on('message', (topic, payload) => {
  console.log(topic, payload.toString());
});
// 监听error事件。
device.on('error', (error) => {
  console.error(error);
});
// 如果您希望主动断开与物联网平台的连接，可以删除下一行注释符号，调用end函数断开与物联网平台的连接
。
//device.end();
```

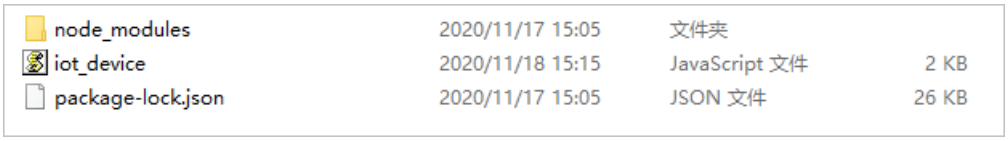
您需参照下表，替换对应参数的值为实际场景中设备的信息。

参数	示例	说明
productKey	a1W***	您添加设备后，保存的设备证书信息，请参见获取设备证书。 您也可在控制台中设备**device2**的**设备详情**页面查看。
deviceName	device2	
deviceSecret	ff01e59d1a***	

参数	示例	说明
instanceId	''	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 ○ 若有ID值，必须传入该ID值。 ○ 若无**实例概览**页面或ID值，传入空值，即 `iotInstanceId = ''` 。 实例的详细说明，请参见实例概述。
region	cn-shanghai	您物联网平台设备所在地域的代码。地域代码表达方法，请参见地域和可用区。

4. 打开CMD窗口，使用cd命令找到*iot_device.js*文件所在路径，在该路径下使用npm命令下载阿里云IoT的Link SDK库。下载后的库文件如下图所示。

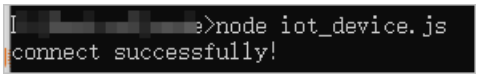
```
npm install alibabacloud-iot-device-sdk --save
```



5. 在CMD窗口输入如下命令，运行*iot_device.js*代码，启动设备。

```
node iot_device.js
```

返回如下信息，表示设备接入成功，并成功发布消息。



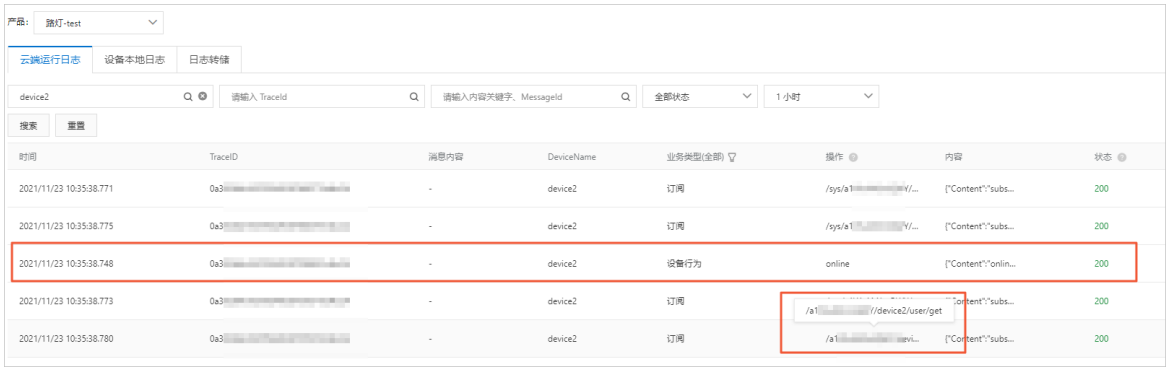
查看运行日志和测试下行通信

1. 登录物联网平台控制台。
2. 在控制台左上方，选择物联网平台所在地域。
3. 在左侧导航栏，选择**设备管理 > 设备**。

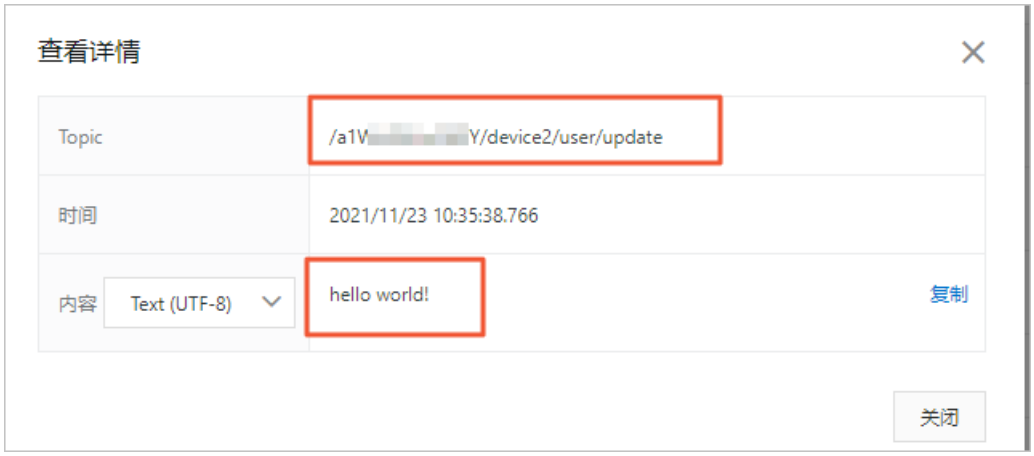
   在**设备列表**页签，可查看设备**device2**状态为**在线**。

4. 单击设备**device2**操作栏的**查看**，在**设备详情**页面，单击**日志服务**，然后单击**前往查看**。

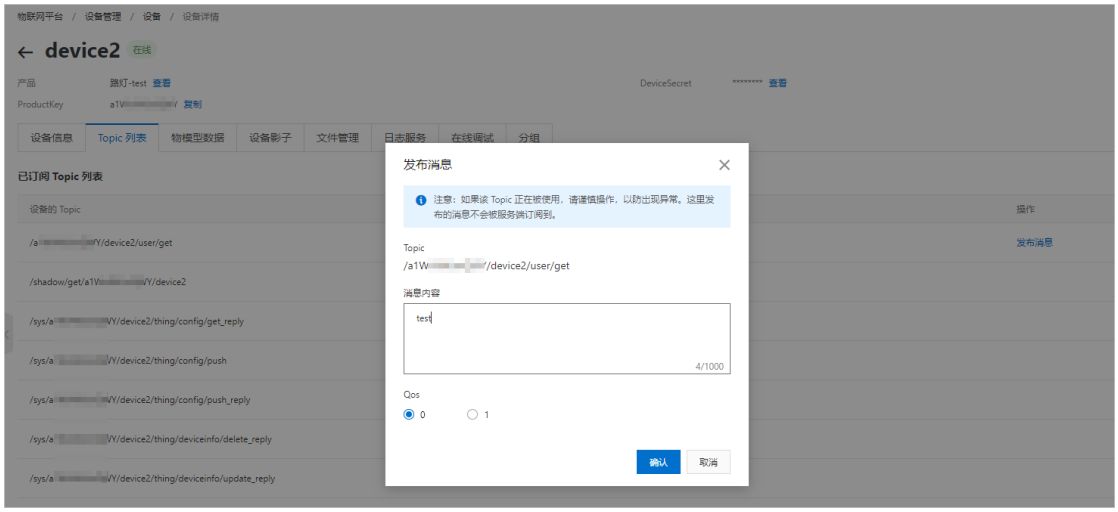
   在**云端运行日志**页签，查看日志消息。



5. 在日志列表，找到**设备到云消息**，单击**查看**，查看设备上报到物联网平台的信息。



6. 测试下行通信：从物联网平台向设备发送消息。
   i. 返回**设备管理 > 设备**页面，在**设备列表**页签，单击设备**device2**操作栏的**查看**。
   ii. 在**设备详情**页面，单击**Topic列表**页签，找到已订阅的Topic：`/a1W***/device2/user/get`，单击**发布消息**。
   iii. 输入消息内容，单击**确认**。



iv. 返回设备运行窗口，查看设备能接收消息，表示通信正常。

您也可返回**云端运行日志**页签，查看详细的通信日志。

1.5. 一型一密动态注册（MQTT通道）

本文以一型一密预注册的Java代码为例，介绍基于MQTT通信协议的设备，如何动态注册并获取接入物联网平台认证需要的DeviceSecret。

前提条件

已完成一型一密文档中的以下步骤：

1. 创建产品。
2. 开启动态注册。
3. 添加设备。
4. 产线烧录。

背景信息

物联网平台支持多种设备安全认证方式，具体认证方式，请参见设备安全认证。

物联网平台支持基于MQTT通道的一型一密预注册和免预注册认证。相关流程和参数说明，请参见基于MQTT通道的设备动态注册。

操作步骤

1. 打开IntelliJ IDEA，创建一个Maven工程。例如**MQTT动态注册**。
2. 在工程中的pom.xml文件中，添加Maven依赖，然后单击Load Maven Changes图标，完成依赖包下载。

```
<dependency>
  <groupId>org.eclipse.paho</groupId>
  <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
  <version>1.2.1</version>
</dependency>
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.61</version>
</dependency>
```

3. 在工程**MQTT动态注册**的路径*/src/main/java*下，创建Java类。例如DynamicRegisterByMqtt，输入以

下代码。

> **说明**
>
> - 设备未激活时，可进行多次动态注册，设备的DeviceSecret以最后一次为准。请确保固化到设备的DeviceSecret为最新。
> - 设备已激活时，您需调用ResetThing API重置云端设备状态为未激活，才能再次动态注册该设备。

```
/*
 * Copyright © 2019 Alibaba. All rights reserved.
 */
package com.aliyun.iot.demo;
import java.nio.charset.StandardCharsets;
import java.util.Random;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import com.alibaba.fastjson.JSONObject;
/**
 * 设备动态注册。
 */
public class DynamicRegisterByMqtt {
    // 地域ID，填写您的产品所在地域ID。
    private static String regionId = "cn-shanghai";
    // 定义加密方式。可选MAC算法：HmacMD5、HmacSHA1、HmacSHA256，需和signmethod取值一致。
    private static final String HMAC_ALGORITHM = "hmacsha1";
    // 接收物联网平台下发设备证书的Topic。无需创建，无需订阅，直接使用。
    private static final String REGISTER_TOPIC = "/ext/register";
    /**
     * 动态注册。
     *
     * @param productKey 产品的ProductKey
     * @param productSecret 产品密钥
     * @param deviceName 设备名称
     * @throws Exception
     */
    public void register(String productKey, String productSecret, String deviceName) th
rows Exception {
        // 接入域名，只能使用TLS。
        String broker = "ssl://" + productKey + ".iot-as-mqtt." + regionId + ".aliyuncs
.com:1883";
        // 表示客户端ID，建议使用设备的MAC地址或SN码，64字符内。
        String clientId = productKey + "." + deviceName;
```

```
        // 获取随机值。
        Random r = new Random();
        int random = r.nextInt(1000000);
        // securemode只能为2表示只能使用TLS；signmethod指定签名算法。
        String clientOpts = "|securemode=2,authType=register,signmethod=" + HMAC_ALGORI
THM + ",random=" + random + "|";
        // MQTT接入客户端ID。
        String mqttClientId = clientId + clientOpts;
        // MQTT接入用户名。
        String mqttUsername = deviceName + "&" + productKey;
        // MQTT接入密码，即签名。
        JSONObject params = new JSONObject();
        params.put("productKey", productKey);
        params.put("deviceName", deviceName);
        params.put("random", random);
        String mqttPassword = sign(params, productSecret);
        // 通过MQTT connect报文进行动态注册。
        connect(broker, mqttClientId, mqttUsername, mqttPassword);
    }
    /**
     * 通过MQTT connect报文发送动态注册信息。
     *
     * @param serverURL 动态注册域名地址
     * @param clientId 客户端ID
     * @param username MQTT用户名
     * @param password MQTT密码
     */
    @SuppressWarnings("resource")
    private void connect(String serverURL, String clientId, String username, String pas
sword) {
        try {
            MemoryPersistence persistence = new MemoryPersistence();
            MqttClient sampleClient = new MqttClient(serverURL, clientId, persistence);
            MqttConnectOptions connOpts = new MqttConnectOptions();
            connOpts.setMqttVersion(4);// MQTT 3.1.1
            connOpts.setUserName(username);// 用户名
            connOpts.setPassword(password.toCharArray());// 密码
            connOpts.setAutomaticReconnect(false); // MQTT动态注册协议规定必须关闭自动重连
。
            System.out.println("----- register params -----");
            System.out.print("server=" + serverURL + ",clientId=" + clientId);
            System.out.println(",username=" + username + ",password=" + password);
            sampleClient.setCallback(new MqttCallback() {
                @Override
                public void messageArrived(String topic, MqttMessage message) throws Ex
ception {
                    // 仅处理动态注册返回消息。
                    if (REGISTER_TOPIC.equals(topic)) {
                        String payload = new String(message.getPayload(), StandardChars
ets.UTF_8);
                        System.out.println("----- register result -----");
                        System.out.println(payload);
                        sampleClient.disconnect();
                    }
```

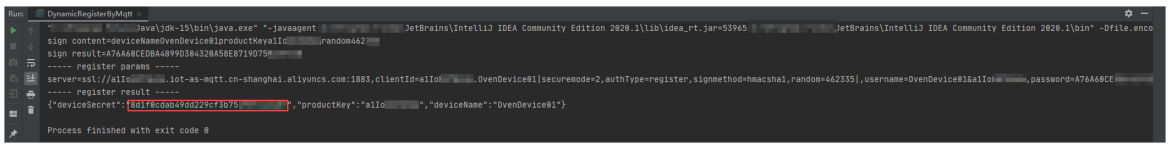
```
                }
                @Override
                public void deliveryComplete(IMqttDeliveryToken token) {
                }
                @Override
                public void connectionLost(Throwable cause) {
                }
            });
            sampleClient.connect(connOpts);
        } catch (MqttException e) {
            System.out.print("register failed: clientId=" + clientId);
            System.out.println(",username=" + username + ",password=" + password);
            System.out.println("reason " + e.getReasonCode());
            System.out.println("msg " + e.getMessage());
            System.out.println("loc " + e.getLocalizedMessage());
            System.out.println("cause " + e.getCause());
            System.out.println("excep " + e);
            e.printStackTrace();
        }
    }
    /**
     * 动态注册签名。
     *
     * @param params 签名参数
     * @param productSecret 产品密钥
     * @return 签名十六进制字符串
     */
    private String sign(JSONObject params, String productSecret) {
        // 请求参数按字典顺序排序。
        Set<String> keys = getSortedKeys(params);
        // sign、signMethod除外。
        keys.remove("sign");
        keys.remove("signMethod");
        // 组装签名明文。
        StringBuffer content = new StringBuffer();
        for (String key : keys) {
            content.append(key);
            content.append(params.getString(key));
        }
        // 计算签名。
        String sign = encrypt(content.toString(), productSecret);
        System.out.println("sign content=" + content);
        System.out.println("sign result=" + sign);
        return sign;
    }
    /**
     * 获取JSON对象排序后的key集合。
     *
     * @param json 需要排序的JSON对象
     * @return 排序后的key集合
     */
    private Set<String> getSortedKeys(JSONObject json) {
        SortedMap<String, String> map = new TreeMap<String, String>();
        for (String key : json.keySet()) {
```

```
            String vlaue = json.getString(key);
            map.put(key, vlaue);
        }
        return map.keySet();
    }
    /**
     * 使用HMAC_ALGORITHM加密。
     *
     * @param content 明文
     * @param secret 密钥
     * @return 密文
     */
    private String encrypt(String content, String secret) {
        try {
            byte[] text = content.getBytes(StandardCharsets.UTF_8);
            byte[] key = secret.getBytes(StandardCharsets.UTF_8);
            SecretKeySpec secretKey = new SecretKeySpec(key, HMAC_ALGORITHM);
            Mac mac = Mac.getInstance(secretKey.getAlgorithm());
            mac.init(secretKey);
            return byte2hex(mac.doFinal(text));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    /**
     * 二进制转十六进制字符串。
     *
     * @param b 二进制数组
     * @return 十六进制字符串
     */
    private String byte2hex(byte[] b) {
        StringBuffer sb = new StringBuffer();
        for (int n = 0; b != null && n < b.length; n++) {
            String stmp = Integer.toHexString(b[n] & 0XFF);
            if (stmp.length() == 1) {
                sb.append('0');
            }
            sb.append(stmp);
        }
        return sb.toString().toUpperCase();
    }
    public static void main(String[] args) throws Exception {
        String productKey = "a1IoK******";
        String productSecret = "6vEu5Qlj5S******";
        String deviceName = "OvenDevice01";
        // 进行动态注册。
        DynamicRegisterByMqtt client = new DynamicRegisterByMqtt();
        client.register(productKey, productSecret, deviceName);
        // 动态注册成功，需要在本地固化deviceSecret。
    }
}
```

参数	示例	说明
regionId	cn-shanghai	您的物联网平台服务所在地域ID。地域代码表达方法，请参见地域和可用区。
productKey	a1IoK******	已烧录至设备的产品ProductKey，可登录物联网平台，在**产品详情**页查看。
productSecret	6vEu5Qlj5S******	已烧录至设备的产品ProductSecret，可登录物联网平台，在**产品详情**页查看。
deviceName	OvenDevice01	您设备的名称。 因设备激活时会校验DeviceName，建议您采用可以直接从设备中读取到的ID，如设备的MAC地址、IMEI或SN码等，作为DeviceName使用。

4. 运行DynamicRegisterByMqtt.java文件，使设备携带DeviceName和所属产品的ProductKey、ProductSecret向云端发起认证请求。

   执行结果如图所示，物联网平台校验通过后，设备接收到云端下发的DeviceSecret（`8d1f0cdab49dd229cf3b75**********`）。



后续步骤

设备获得连接云端所需的设备证书（ProductKey、DeviceName和DeviceSecret）后，您再使用MQTT客户端，将设备接入物联网平台，进行数据通信。

具体操作，请参见Paho-MQTT Java接入示例。

1.6. 一型一密动态注册（HTTPS通道）

本文以一型一密预注册的Java代码为例，介绍基于HTTPS通信协议的设备，如何动态注册并获取接入物联网平台认证需要的DeviceSecret。

前提条件

已完成一型一密文档中的以下步骤：

1. 创建产品。
2. 开启动态注册。
3. 添加设备。
4. 产线烧录。

背景信息

物联网平台支持多种设备安全认证方式，具体认证方式，请参见设备安全认证。

物联网平台支持基于HTTPS通道实现一型一密的预注册认证。相关Topic和参数说明，请参见直连设备的HTTPS动态注册。

操作步骤

1. 打开IntelliJ IDEA，创建一个Maven工程。例如**HTTPS动态注册**。
2. 在工程中的pom.xml文件中，添加Maven依赖，然后单击Load Maven Changes图标，完成依赖包下载。

```
<dependency>
  <groupId>com.alibaba</groupId>
  <artifactId>fastjson</artifactId>
  <version>1.2.61</version>
</dependency>
```

3. 在工程**HTTPS动态注册**的路径*/src/main/java*下，创建Java类。例如DynamicRegisterByHttps，输入以下代码。

> **说明**
>
> - 设备未激活时，可进行多次动态注册，设备的DeviceSecret以最后一次为准。请确保固化到设备的DeviceSecret为最新。
> - 设备已激活时，您需调用ResetThing API重置云端设备状态为未激活，才能再次动态注册该设备。

```
/*
 * Copyright © 2019 Alibaba. All rights reserved.
 */
package com.aliyun.iot.demo;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.util.Random;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.HttpsURLConnection;
import com.alibaba.fastjson.JSONObject;
/**
 * 设备动态注册。
 */
public class DynamicRegisterByHttps {
    // 地域ID，填写您的产品所在地域ID。
    private static String regionId = "cn-shanghai";
    // 定义加密方式。可选MAC算法：HmacMD5、HmacSHA1、HmacSHA256，需和signmethod一致。
    private static final String HMAC_ALGORITHM = "hmacsha1";
    /**
     * 动态注册。
     *
     * @param productKey 产品key
     * @param productSecret 产品密钥
     * @param deviceName 设备名称
```

```
     * @throws Exception
     */
    public void register(String productKey, String productSecret, String deviceName) th
rows Exception {
        // 请求地址。
        URL url = new URL("https://iot-auth." + regionId + ".aliyuncs.com/auth/register
/device");
        HttpsURLConnection conn = (HttpsURLConnection) url.openConnection();
        conn.setRequestMethod("POST");
        conn.setRequestProperty("Content-type", "application/x-www-form-urlencoded");
        conn.setDoOutput(true);
        conn.setDoInput(true);
        // 获取URLConnection对象对应的输出流。
        PrintWriter out = new PrintWriter(conn.getOutputStream());
        // 发送请求参数。
        out.print(registerdBody(productKey, productSecret, deviceName));
        // flush输出流的缓冲。
        out.flush();
        // 获取URLConnection对象对应的输入流。
        BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStrea
m()));
        // 读取URL的响应。
        String result = "";
        String line = "";
        while ((line = in.readLine()) != null) {
            result += line;
        }
        System.out.println("----- register result -----");
        System.out.println(result);
        // 关闭输入输出流。
        in.close();
        out.close();
        conn.disconnect();
    }
    /**
     * 生成动态注册请求内容。
     *
     * @param productKey 产品ProductKey
     * @param productSecret 产品密钥
     * @param deviceName 设备名称
     * @return 动态注册payload
     */
    private String registerdBody(String productKey, String productSecret, String device
Name) {
        // 获取随机值。
        Random r = new Random();
        int random = r.nextInt(1000000);
        // 动态注册参数。
        JSONObject params = new JSONObject();
        params.put("productKey", productKey);
        params.put("deviceName", deviceName);
        params.put("random", random);
        params.put("signMethod", HMAC_ALGORITHM);
        params.put("sign", sign(params, productSecret));
```

```
        // 拼接payload。
        StringBuffer payload = new StringBuffer();
        for (String key : params.keySet()) {
            payload.append(key);
            payload.append("=");
            payload.append(params.getString(key));
            payload.append("&");
        }
        payload.deleteCharAt(payload.length() - 1);
        System.out.println("----- register payload -----");
        System.out.println(payload);
        return payload.toString();
    }
    /**
     * 动态注册签名。
     *
     * @param params 签名参数
     * @param productSecret 产品密钥
     * @return 签名十六进制字符串
     */
    private String sign(JSONObject params, String productSecret) {
        // 请求参数按字典顺序排序。
        Set<String> keys = getSortedKeys(params);
        // sign、signMethod除外
        keys.remove("sign");
        keys.remove("signMethod");
        // 组装签名明文。
        StringBuffer content = new StringBuffer();
        for (String key : keys) {
            content.append(key);
            content.append(params.getString(key));
        }
        // 计算签名。
        String sign = encrypt(content.toString(), productSecret);
        System.out.println("sign content=" + content);
        System.out.println("sign result=" + sign);
        return sign;
    }
    /**
     * 获取JSON对象排序后的key集合。
     *
     * @param json 需要排序的JSON对象
     * @return 排序后的key集合
     */
    private Set<String> getSortedKeys(JSONObject json) {
        SortedMap<String, String> map = new TreeMap<String, String>();
        for (String key : json.keySet()) {
            String vlaue = json.getString(key);
            map.put(key, vlaue);
        }
        return map.keySet();
    }
    /**
     * 使用HMAC_ALGORITHM加密。
     *
```

```
     *
     * @param content 明文
     * @param secret 密钥
     * @return 密文
     */
    private String encrypt(String content, String secret) {
        try {
            byte[] text = content.getBytes(StandardCharsets.UTF_8);
            byte[] key = secret.getBytes(StandardCharsets.UTF_8);
            SecretKeySpec secretKey = new SecretKeySpec(key, HMAC_ALGORITHM);
            Mac mac = Mac.getInstance(secretKey.getAlgorithm());
            mac.init(secretKey);
            return byte2hex(mac.doFinal(text));
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
    /**
     * 二进制转十六进制字符串。
     *
     * @param b 二进制数组
     * @return 十六进制字符串
     */
    private String byte2hex(byte[] b) {
        StringBuffer sb = new StringBuffer();
        for (int n = 0; b != null && n < b.length; n++) {
            String stmp = Integer.toHexString(b[n] & 0XFF);
            if (stmp.length() == 1) {
                sb.append('0');
            }
            sb.append(stmp);
        }
        return sb.toString().toUpperCase();
    }
    public static void main(String[] args) throws Exception {
        String productKey = "a1IoK******";
        String productSecret = "6vEu5Qlj5S******";
        String deviceName = "OvenDevice01";
        // 进行动态注册。
        DynamicRegisterByHttps client = new DynamicRegisterByHttps();
        client.register(productKey, productSecret, deviceName);
        // 动态注册成功后，需要固化deviceSecret。
    }
}
```

参数	示例	说明
regionId	cn-shanghai	您的物联网平台服务所在地域ID。地域代码表达方法，请参见地域和可用区。
productKey	a1IoK******	已烧录至设备的产品ProductKey，可登录物联网平台，在**产品详情**页查看。

参数	示例	说明
productSecret	6vEu5Qlj5S******	已烧录至设备的产品ProductSecret，可登录物联网平台，在**产品详情**页查看。
deviceName	OvenDevice01	您设备的名称。 因设备激活时会校验DeviceName，建议您采用可以直接从设备中读取到的ID，如设备的MAC地址、IMEI或SN码等，作为DeviceName使用。

4. 运行DynamicRegisterByHttps.java文件，使设备携带DeviceName和所属产品的ProductKey、ProductSecret向云端发起认证请求。执行结果如图所示，物联网平台校验通过后，设备接收到云端下发的DeviceSecret（ `6b14088fa377e8f852d82f7f********` ）。

后续步骤

设备获得连接云端所需的设备证书（ProductKey、DeviceName和DeviceSecret）后，您再使用MQTT客户端，将设备接入物联网平台，进行数据通信。

具体操作，请参见Paho-MQTT Java接入示例。

1.7. 使用MQTT.fx接入物联网平台

MQTT.fx是一款基于Eclipse Paho，使用Java语言编写的MQTT客户端，支持Windows、Mac和Linux操作系统，可用于验证设备是否可与物联网平台正常连接，并通过Topic订阅和发布消息。本文以Windows系统下MQTT.fx为例，介绍模拟设备以MQTT协议接入物联网平台。

前提条件

已在物联网平台控制台创建产品和设备，然后在**设备详情**页面，获取设备证书和**MQTT连接参数**的信息。具体操作，请参见：

- 创建产品。
- 创建设备。
- 获取MQTT签名参数值。

本文获取的设备证书和MQTT连接参数值如下表，参数详细说明，请参见MQTT-TCP连接通信。

参数	值
ProductKey	`a1***`
DeviceName	`device1`
DeviceSecret	`f35***d9e`

参数	值
clientId	a1***.device1|securemode=2,signmethod=hmacsha256,timestamp=2524608000000|
username	device1&a1***
passwd	86761***21d
mqttHostUrl	a1***.iot-as-mqtt.cn-shanghai.aliyuncs.com
port	1883

> **注意** MQTT.fx模拟的在线设备，仅支持非透传消息通信。如需实现透传信息通信，您可以使用真实设备或SDK进行测试。

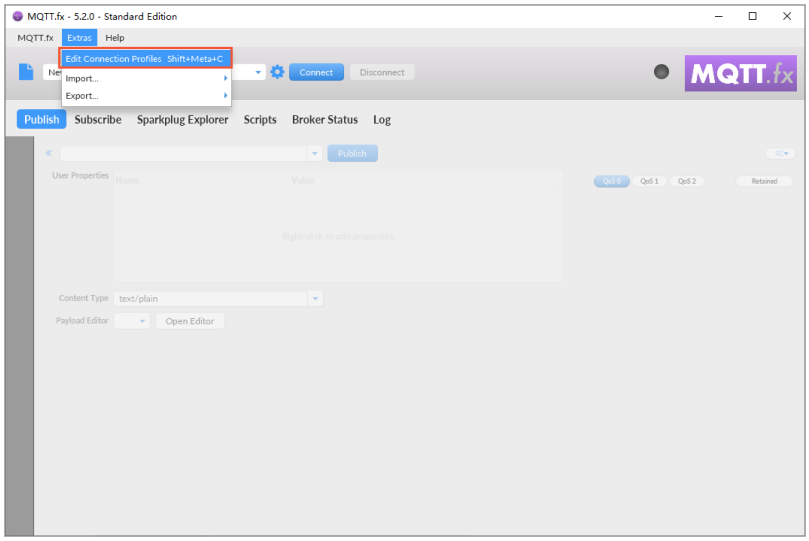
配置MQTT.fx接入

1. 下载并安装MQTT.fx软件。MQTT.fx软件安装和使用说明，请参见MQTT.fx。

   本示例使用mqttfx-5.2.0-windows-x64版本软件。

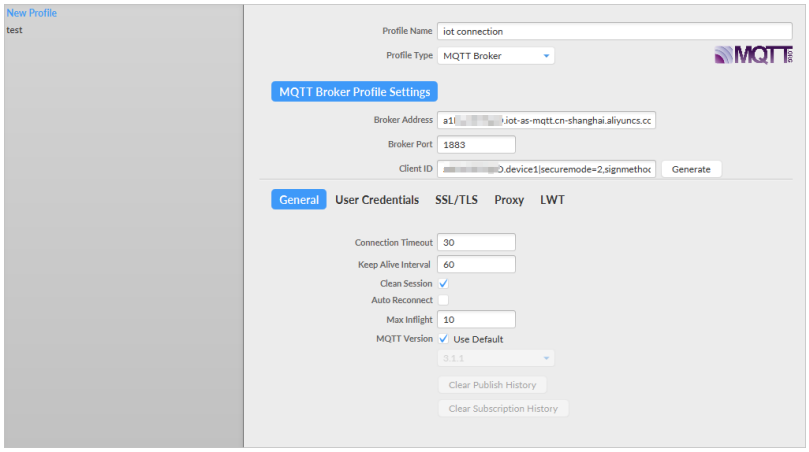
   > **说明** 使用MQTT.fx工具所需的License，请自行申请。

2. 打开MQTT.fx软件，单击菜单栏中的**Extras**，选择**Edit Connection Profiles**。



3. 在**Edit Connection Profiles**页面，完成以下参数的设置。

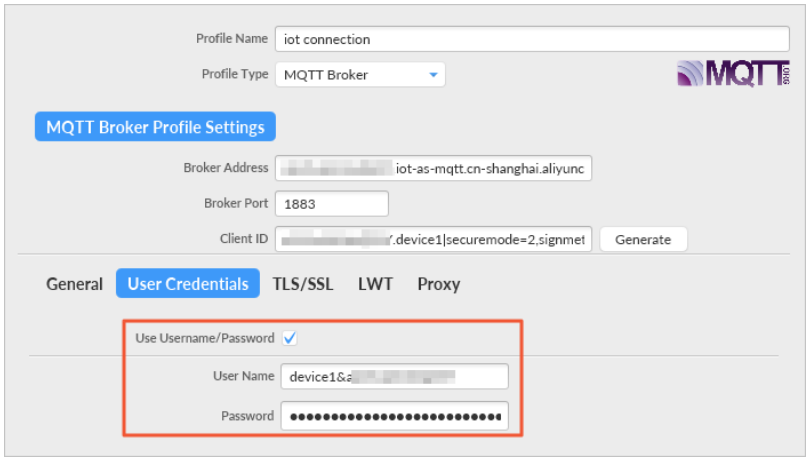
   i. 设置基本信息。



参数	说明
Profile Name	输入您的自定义名称*iot connection*。
Profile Type	MQTT服务器连接，选择**MQTT Broker**。
MQTT Broker Profile Settings	
Broker Address	MQTT接入域名，对应*前提条件*中已获取的mqttHostUrl值：*a1***.iot-as-mqtt.cn-shanghai.aliyuncs.com*。 ■ *a1***为本示例产品的ProductKey。 ■ *cn-shanghai*为本示例所在地域。
Broker Port	设置为*1883*。

参数	说明
Client ID	MQTT的协议字段。 固定格式：`${clientId}\|securemode=${mode},signmethod=${signMethod}\|timestamp=${timestamp}|`。 输入*前提条件*中已获取的clientId值：*a1***.device1|securemode=2,signmethod=hmacsha256,timestamp=2524608000000|*。 参数说明： ■ *${clientId}*：设备、App或Web等场景下的Client ID信息。 **注意** 该值可自定义，长度在64个字符以内。若为设备的ID信息，建议使用您设备的MAC地址或SN码，方便您识别区分不同的设备。 ■ *${mode}*：安全模式。可选值有*2*（TLS直连模式，需要设置SSL/TLS信息）和*3*（TCP直连模式，无需设置SSL/TLS信息）。 ■ *${signMethod}*：算法类型，支持*hmacsha256*、*hmacmd5*和*hmacsha1*。 ■ *${timestamp}*：表示当前时间毫秒值，可以不传递timestamp。 物联网平台提供的连接参数中*${clientId}*默认为`${productKey} + '.' + ${deviceName}`组成的字符串，*${mode}*默认为*2*，*${signMethod}*默认为*hmacsha256*，您可根据需要修改。 **注意** ■ MQTT.fx的Client ID和设备的*${clientId}*，切勿混淆。 ■ 不要遗漏参数之间及最后的竖线（|）。 ■ 设置参数时，请确保参数值中或参数值的前后均没有空格。 ■ 输入Client ID信息后，请勿单击Generate。
General	本示例使用默认值。您也可以根据实际场景需求设置。

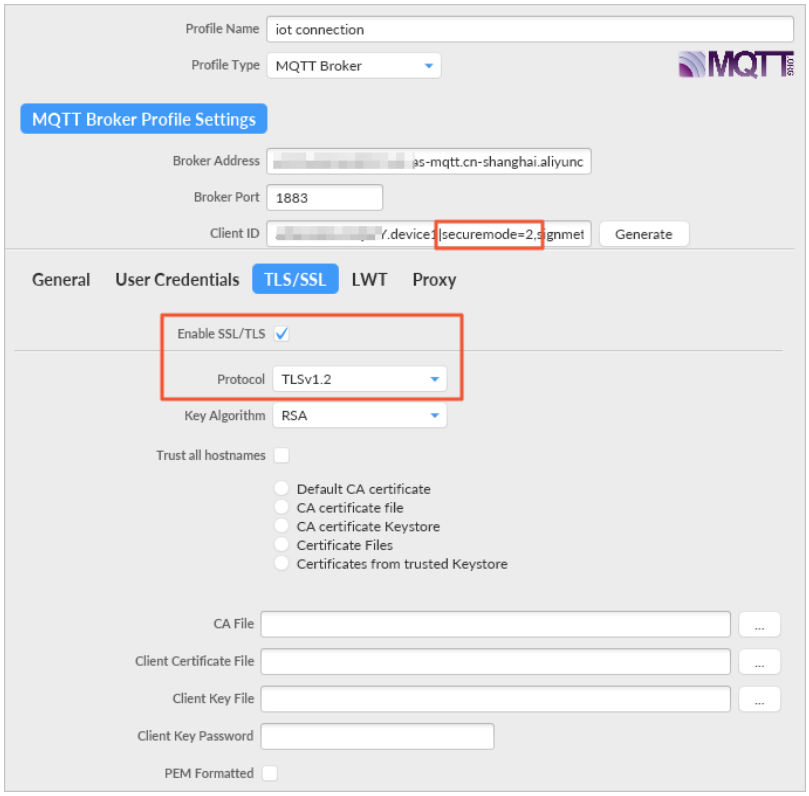
ii. 单击User Credentials，选中Use Username/Password复选框，设置User Name和Password。



参数	参数说明
User Name	由设备名称DeviceName、and（&）和产品ProductKey组成，固定格式为 *${DeviceName}*&*${ProductKey}* 。 输入*前提条件*中已获取的username值：*device1&a1****。 ■ device1 为设备的DeviceName。 ■ a1*** 为设备的ProductKey。
Password	通过选择的加密方法，以设备的DeviceSecret为密钥，将参数和参数值拼接后，加密生成Password。 输入*前提条件*中已获取的passwd值：*86761***21d*。 **注意** ■ 如果您使用的MQTT.fx版本，在粘贴Password后不显示具体的字符串，只要光标已从输入框的前部移至了后部，则表示粘贴成功，请勿重复粘贴。 ■ 请注意参数和参数值中字母的大小写。 ■ 若您设置Client ID时，修改了已获取值中的*${ClientId}*、*${SignMethod}*的设置，必须保证加密参数的值与Client ID中对应参数值一致，需重新计算出Password。相关参数设置与计算方法，请参见使用Node.js语言脚本计算。

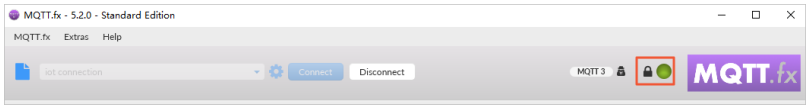
iii. TLS直连模式（即securemode=2）下，单击SSL/TLS，选中Enable SSL/TLS，设置Protocol为TLSv1.2。

> 注意 TCP直连模式（即 `securemode=3` ）下，无需设置SSL/TLS信息，直接进入下一步。



4. 设置完成后，单击右下角的OK。
5. 单击Connect。

   右侧亮绿灯，表示连接成功。



您可在物联网平台控制台，选择**设备管理 > 设备**，选择产品，查看该设备状态，预期设备为**在线**状态。

下文通过测试自定义Topic的上下行通信，验证MQTT.fx与物联网平台连接是否成功。若测试与本示例结果不符，表示通信连接失败，您需根据日志信息，进行修正。

测试下行通信

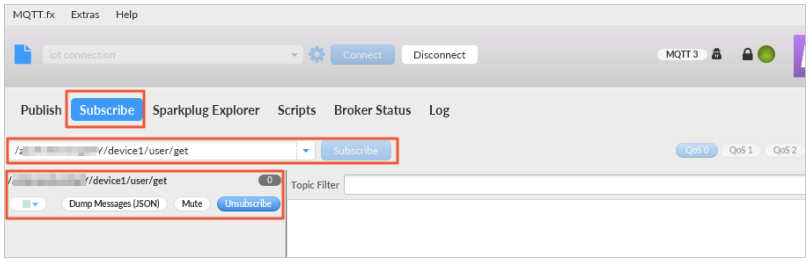
1. 在物联网平台控制台的产品详情页面，单击**Topic类列表 > 自定义Topic**，找到一个具有**订阅**权限的自定义Topic。

   本示例使用Topic：`/a1***/${deviceName}/user/get`，您需替换`${deviceName}`为设备名称`device1`。

   更多信息，请参见自定义Topic。

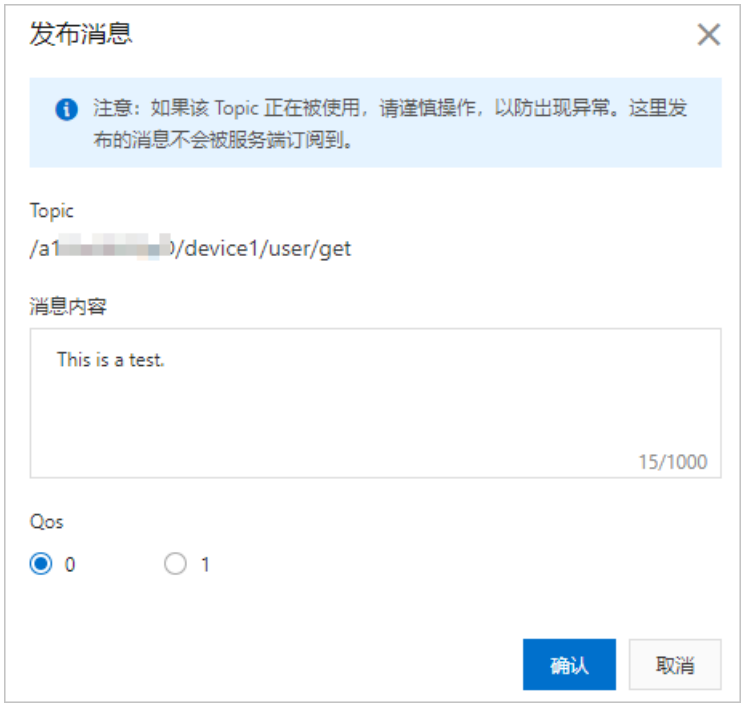
2. 在MQTT.fx上单击**Subscribe**，在**Subscribe**文本框中，输入上一步的Topic，再单击**Subscribe**。

   订阅成功后，该Topic将显示在列表中。

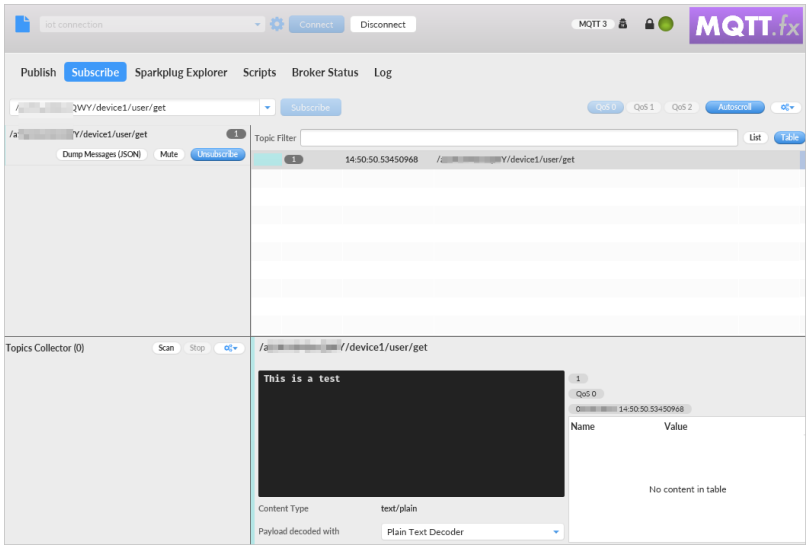


3. 返回物联网平台，进入该设备的**设备详情**页面，在**Topic列表**页签下，单击已订阅Topic对应的**发布消息**。

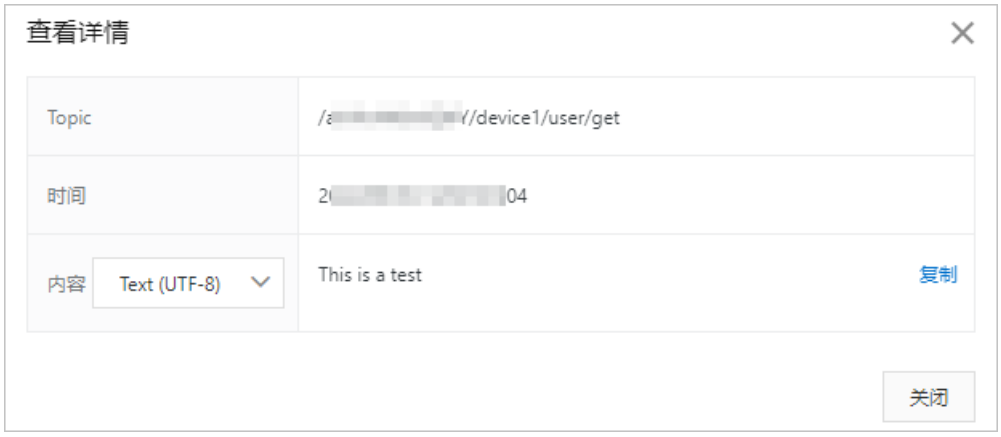
4. 输入消息内容，单击**确认**。



5. 回到MQTT.fx上，查看接收到的消息。



6. 回到物联网平台，在**设备详情**页面，单击**日志服务**页签的**前往查看**，在**日志服务**页面，查看**云到设备消息**。



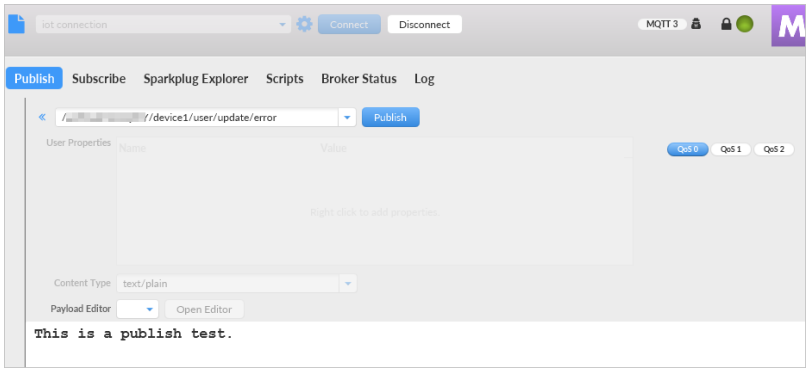
测试上行通信

1. 在物联网平台控制台的产品详情页面，单击**Topic类列表 > 自定义Topic**，找到一个具有**发布**权限的自定义Topic。

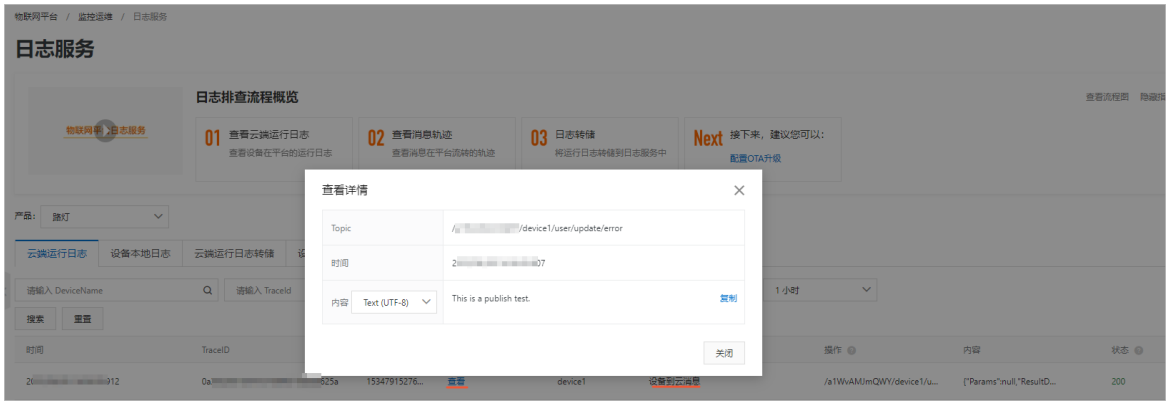
   本示例使用Topic：`/a1***/${deviceName}/user/update/error`，您需替换`${deviceName}`为设备名称`device1`。

   更多信息，请参见自定义Topic。

2. 在MQTT.fx上，单击**Publish**，在**Publish**文本框中，输入上一步的Topic。在文本编辑页面，输入要发送的消息内容，然后单击**Publish**。

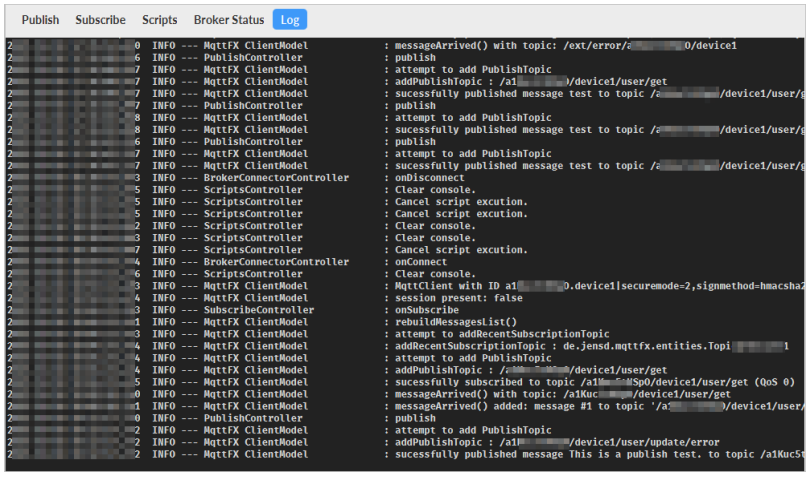


3. 回到物联网平台，在**设备详情**页面，单击**日志服务**页签的**前往查看**，在**日志服务**页面，查看**设备到云消息**。



查看日志

在MQTT.fx上，单击**Log**查看操作日志和错误提示日志。



1.8. Android Things接入物联网平台

本文档以室内空气检测项目为例，介绍如何将谷歌Android Things物联网硬件接入阿里云物联网平台。

硬件设备

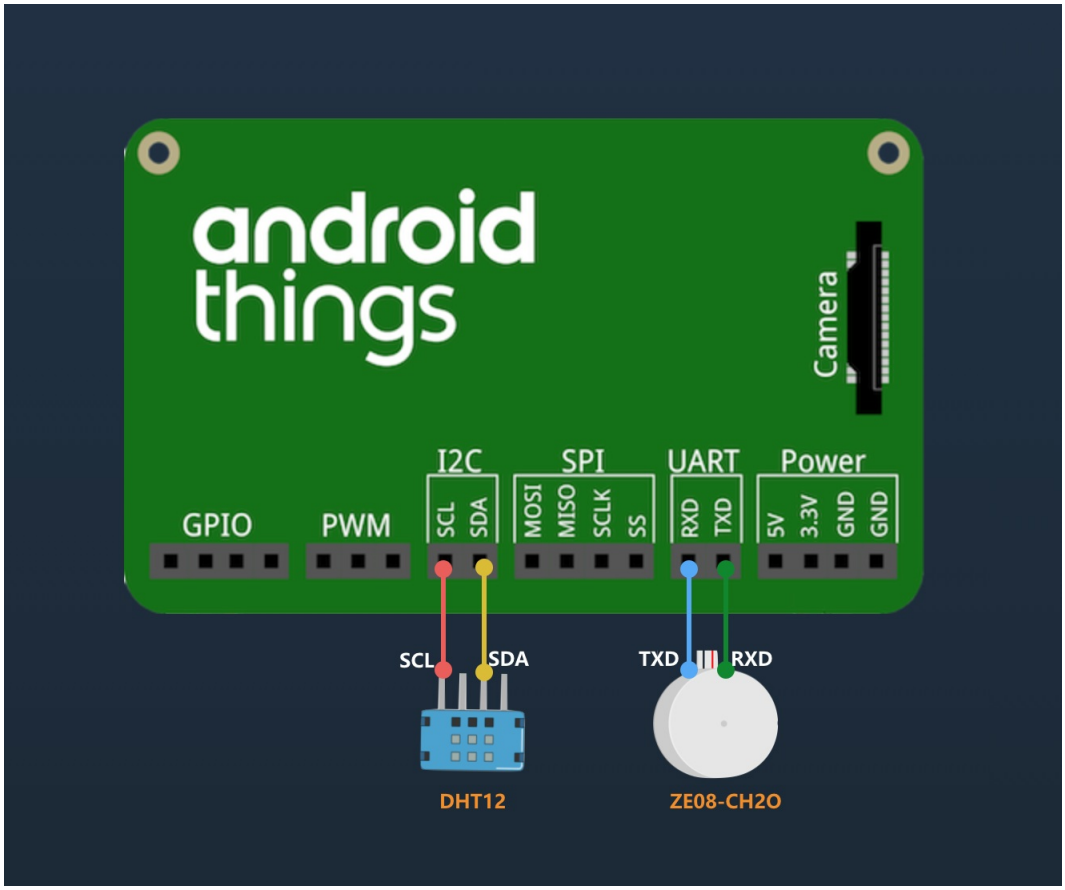
- 项目设备列表

  下表中为室内空气检测所需的硬件设备：

设备	设备图片	备注
NXP Pico i.MX7D 开发板		基于Android things系统1.0。 如需更多帮助，请参见NXP Pico i.MX7D I/O官网接口文档：开发者指南。 **说明** 该硬件可以用树莓派替代。请参见远程控制树莓派服务器。

设备	设备图片	备注
DHT12 温湿度传感器		采用I2C数据通信方式。
ZE08-CH2O 甲醛检测传感器		采用UART数据通信方式。

- 设备接线示意图



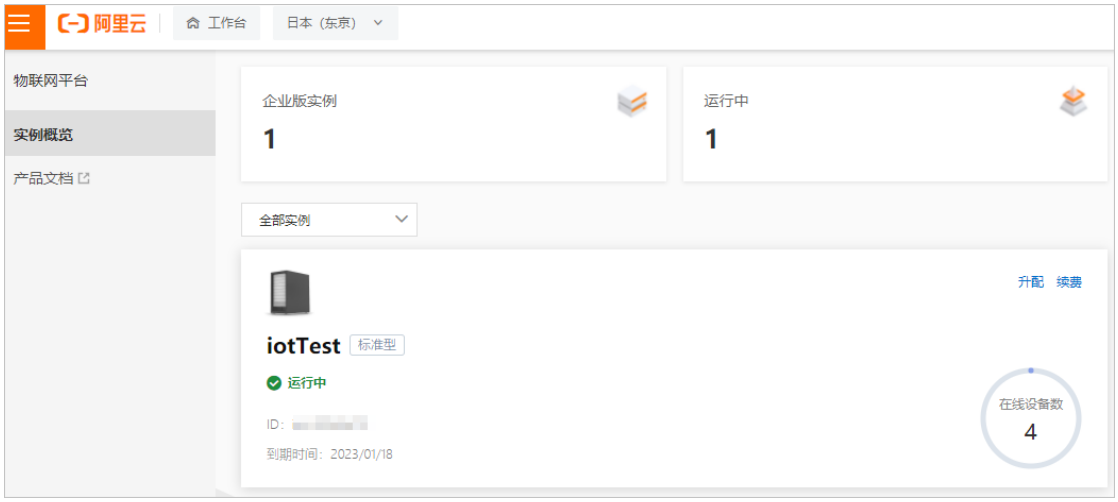
- 将温湿度传感器（DHT12）的时钟信号线引脚SCL和数据线引脚SDA分别与开发板的I2C总线的SCL和SDA引脚相接。
- 将甲醛检测传感器（ZE08-CH2O）的发送数据引脚TXD与开发板的接收数据引脚RXD相接；将ZE08-CH2O的接收数据引脚RXD与开发板的发送数据引脚TXD相接。

创建阿里云物联网平台产品和设备

1. 登录阿里云物联网平台控制台。
2. 创建产品。

i. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



ii. 在左侧导航栏，单击**设备管理**，选择**产品**

iii. 在**产品**页，单击**创建产品**进入产品创建流程。详情操作说明，请参见创建产品。

3. 定义物模型。

i. 在产品创建成功页面，单击**前往定义物模型**。

ii. 在**产品详情**页的**功能定义**页签下，单击**编辑草稿 > 添加自定义功能**。

iii. 添加以下属性。

属性名称	标识符	数据类型	取值范围	描述
温度	temperature	float	-50~100	DHT12温湿度传感器采集的温度数据。
湿度	humidity	float	0~100	DHT12温湿度传感器采集的湿度数据。
甲醛浓度	ch2o	double	0~3	ZE08甲醛检测传感器采集的甲醛浓度。

iv. 单击**发布上线**发布物模型。

4. 创建设备。

在**设备**页面，单击**添加设备**，在产品下创建设备。详情操作说明，请参见单个创建设备。

开发Android things设备

1. 使用Android Studio创建Android things工程，添加网络权限。

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

2. 在gradle中添加 `eclipse.paho.mqtt` 。

```
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.0'
```

3. 配置通过I2C读取温湿度传感器DHT12的数据。

```
private void readDataFromI2C() {
        try {
            byte[] data = new byte[5];
            i2cDevice.readRegBuffer(0x00, data, data.length);
            // check data
            if ((data[0] + data[1] + data[2] + data[3]) % 256 != data[4]) {
                humidity = temperature = 0;
                return;
            }
            // humidity data
            humidity = Double.valueOf(String.valueOf(data[0]) + "." + String.valueOf(da
ta[1]));
            Log.d(TAG, "humidity: " + humidity);
            // temperature data
            if (data[3] < 128) {
                temperature = Double.valueOf(String.valueOf(data[2]) + "." + String.val
ueOf(data[3]));
            } else {
                temperature = Double.valueOf("-" + String.valueOf(data[2]) + "." + Stri
ng.valueOf(data[3] - 128));
            }
            Log.d(TAG, "temperature: " + temperature);
        } catch (IOException e) {
            Log.e(TAG, "readDataFromI2C error " + e.getMessage(), e);
        }
    }
```

4. 配置通过UART获取甲醛检测传感器Ze08-CH2O的数据。

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

5. 创建阿里云物联网平台与设备端的连接，上报数据。

```
/*
payload格式
{
  "id": 123243,
  "params": {
    "temperature": 25.6,
    "humidity": 60.3,
    "ch2o": 0.048
  },
  "method": "thing.event.property.post"
}
*/
MqttMessage message = new MqttMessage(payload.getBytes("utf-8"));
message.setQos(1);
String pubTopYourPc = "/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post";
mqttClient.publish(pubTopic, message);
```

请访问GitHub阿里云IoT ，下载完整的示例工程代码。

查看实时数据

设备启动后，登录物联网平台控制台，找到**设备详情**页，在**运行状态**页签下，查看设备当前的实时属性数据。

1.9. RTOS设备通过TCP模组上云

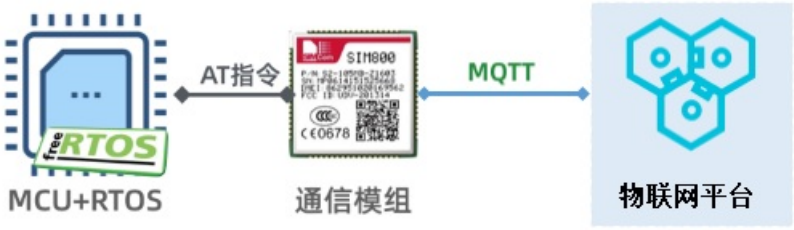
1.9.1. 概述

本实践案例介绍使用物联网平台提供的C语言设备端SDK，将搭载实时操作系统（RTOS）的微控制单元（MCU）的设备接入阿里云物联网平台。

原有的工业自动化设备、数据采集设备、实时控制设备、家电等使用的是搭载实时操作系统（RTOS）的微控制单元（MCU）。在对此类设备进行物联网改造时，可以使用阿里云物联网平台提供的C语言设备端SDK，将此类设备接入物联网平台。

接入方案

将MCU与通信模组相连，MCU与通信模组间通过AT指令进行连接和通信。在通信模组上，使用C语言设备端SDK实现与物联网平台的连接和通信。



准备软硬件

本示例中，使用了如下MCU、通信模组开发板和软件开发环境：

- MCU为ST公司生产的STM32F103，其开发板为NUCLEO-F103RB。
- 通信模组为SIMCom公司（芯讯通无线科技有限公司）生产的SIM800C，其开发板为SIM800C mini V2.0。
- 开发环境为IAR Embedded Workbench for ARM。

实现过程

创建产品和设备

搭建设备端开发环境

开发设备端

1.9.2. 创建产品和设备

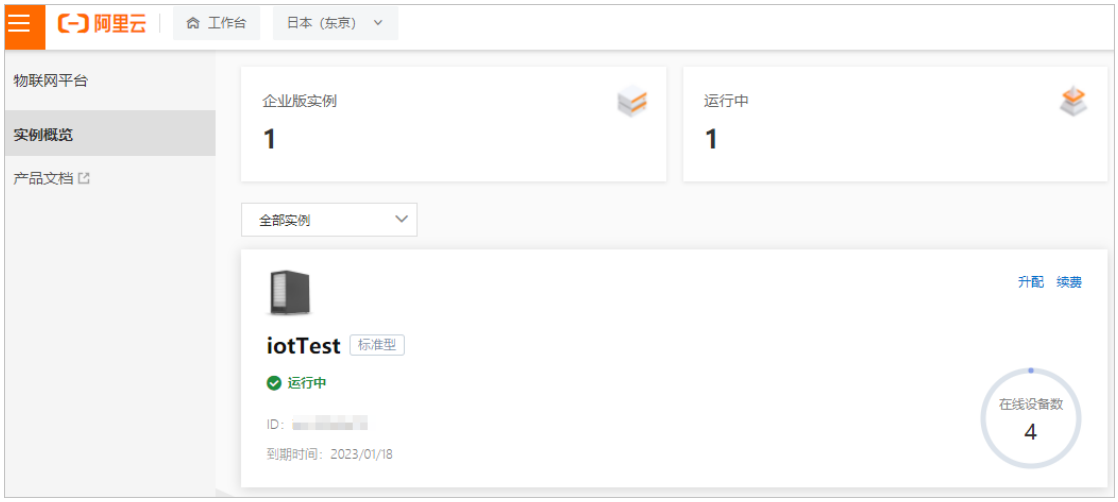
首先，在物联网平台控制台创建产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需配置到设备端SDK中。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

操作步骤

1. 登录物联网平台控制台。
2. 创建产品。

i. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



ii. 在左侧导航栏，选择**设备管理 > 产品**。

iii. 在**产品**页，单击**创建产品**，填入产品信息，然后单击**确认**。

3. 在新创建的产品下添加设备。

i. 在左侧导航栏，选择**设备管理 > 设备**。

ii. 在**设备**页，单击**添加设备**，选择新创建的产品，输入设备DeviceName，然后单击**确认**。

执行结果

设备创建成功后，在弹出的**添加完成**对话框，单击**前往查看**或**一键复制设备证书**，获取设备证书。您也可以在**设备**页，单击设备对应的**查看**，进入**设备详情**页查看设备证书信息。

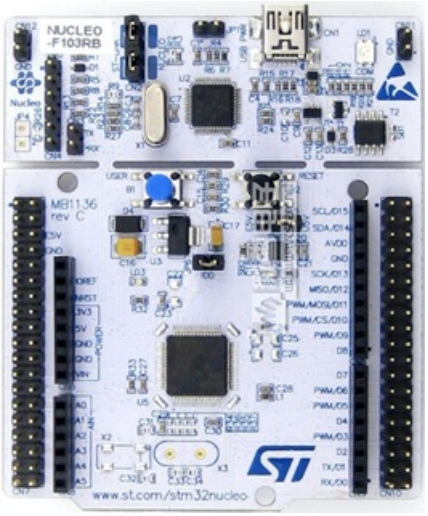
1.9.3. 搭建设备端开发环境

将MCU与通信模组开发板相连，搭建软件开发环境，创建工程项目，导入SDK，完成SDK配置。

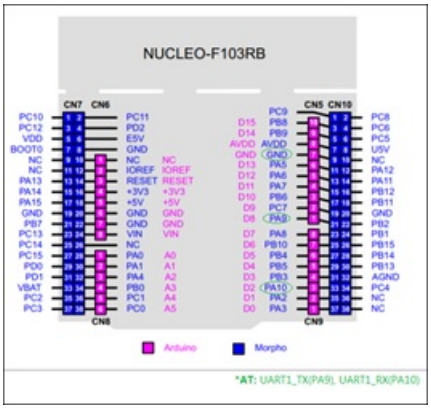
背景信息

本示例中使用了两个开发板示意图如下。

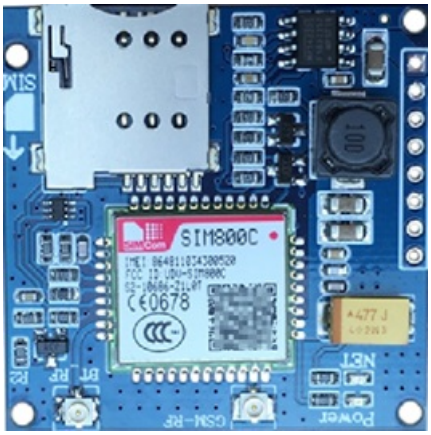
- 开发板NUCLEO-F103RB



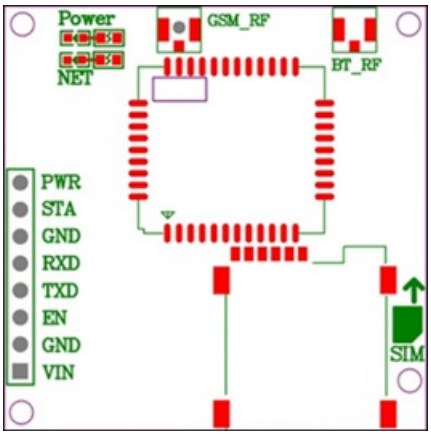
引脚示意图如下。



- SIM800C mini v2.0



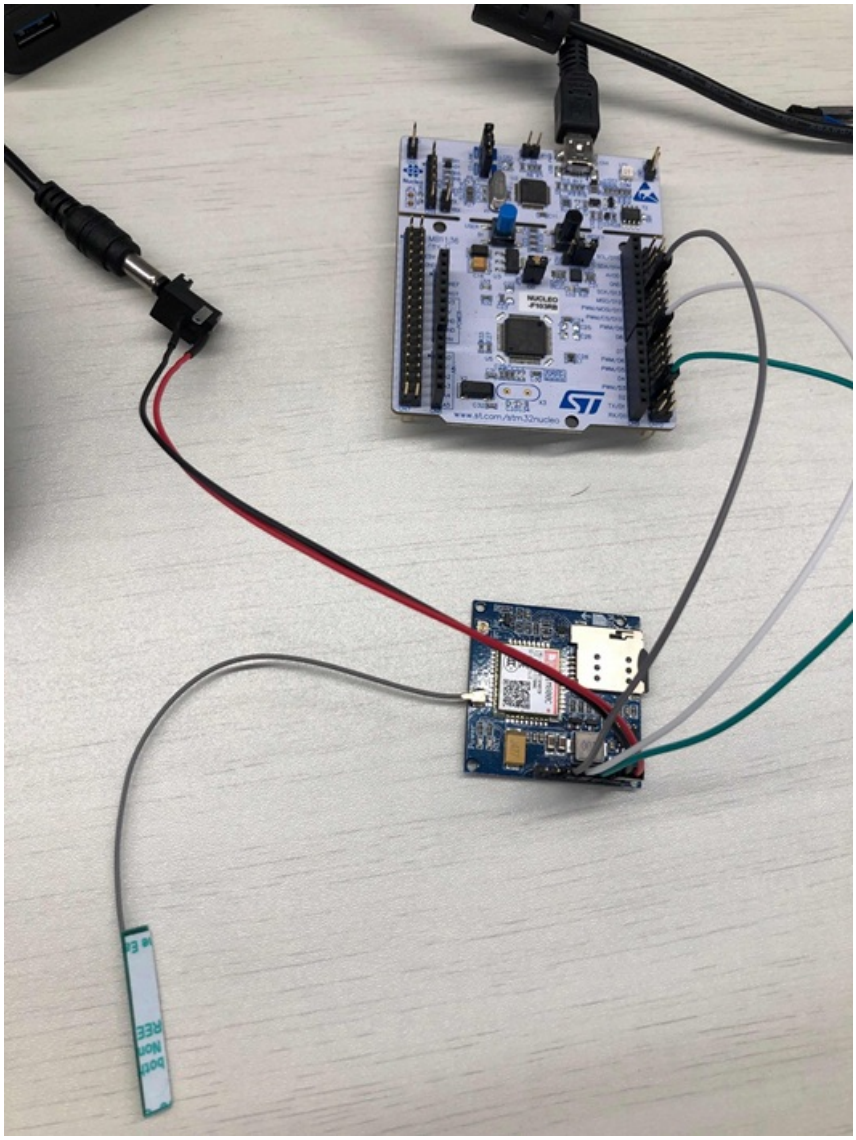
引脚示意图和说明如下。



引脚	说明
PWR	开关机引脚。默认为自动开机。
STA	状态监测引脚。
GND	电源接地引脚。
RXD	接收串口引脚。
TXD	发送串口引脚。
EN	电源使能引脚。
VIN	5~18V电源输入。

连接硬件

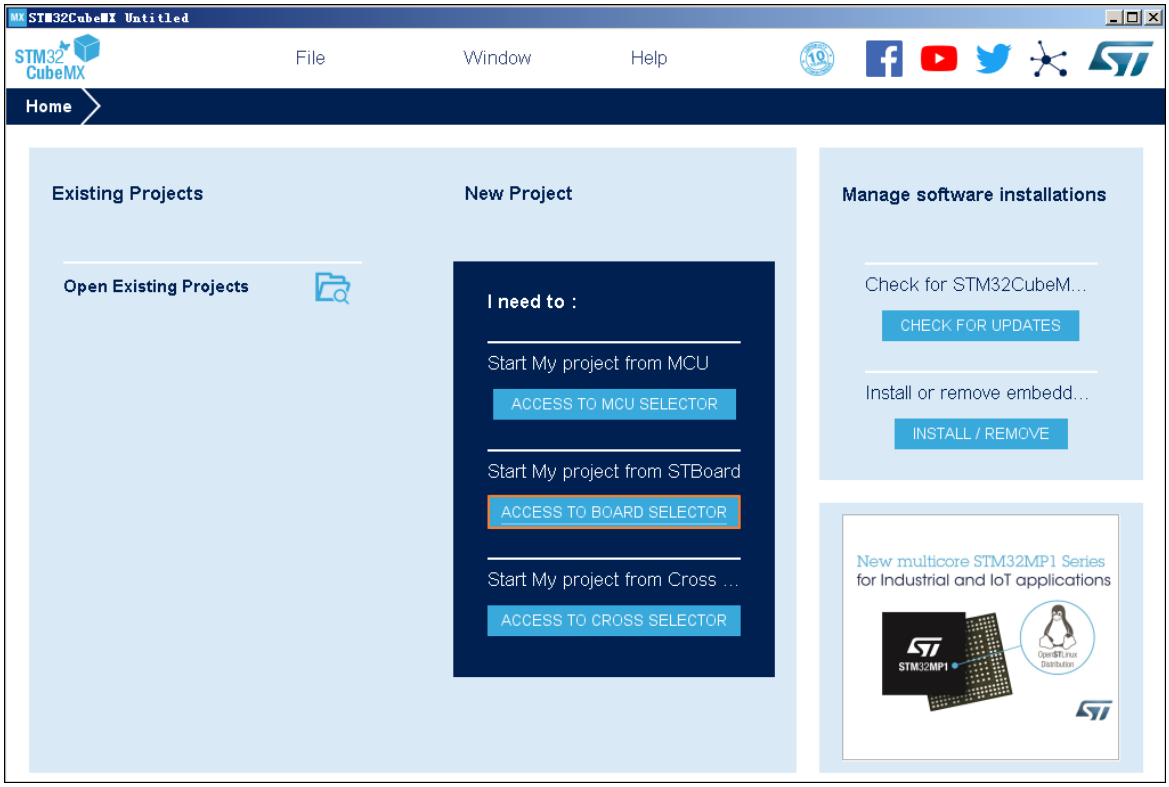
将两个开发板的接收和发送串口连接，作为AT指令通道，如下图所示。



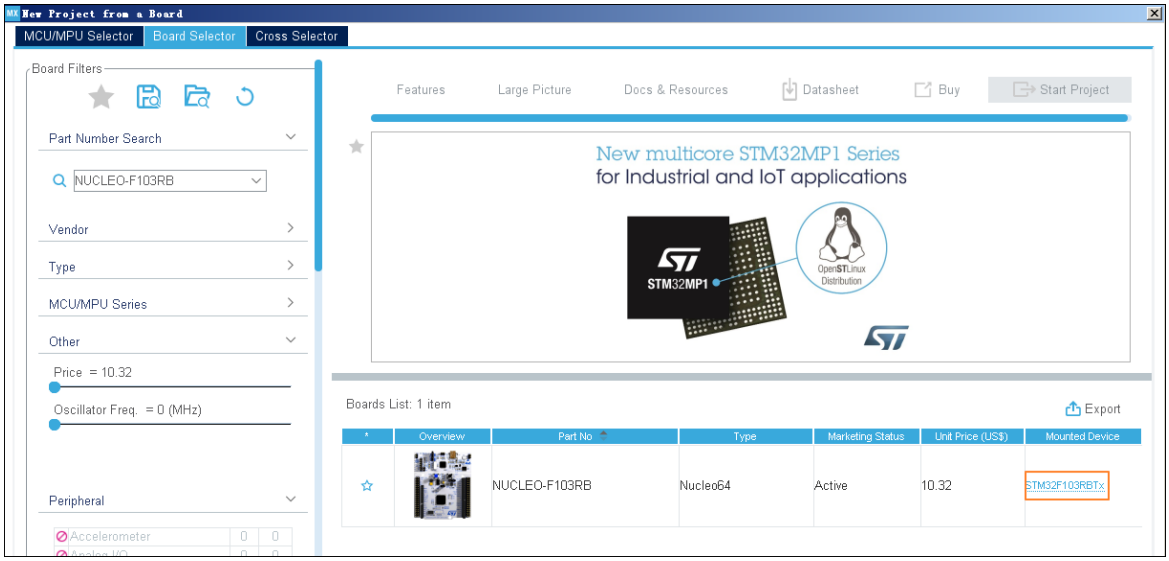
搭建开发环境

本示例开发工具为STM32CubeMX。使用详情请参见STM32Cube Ecosystem。

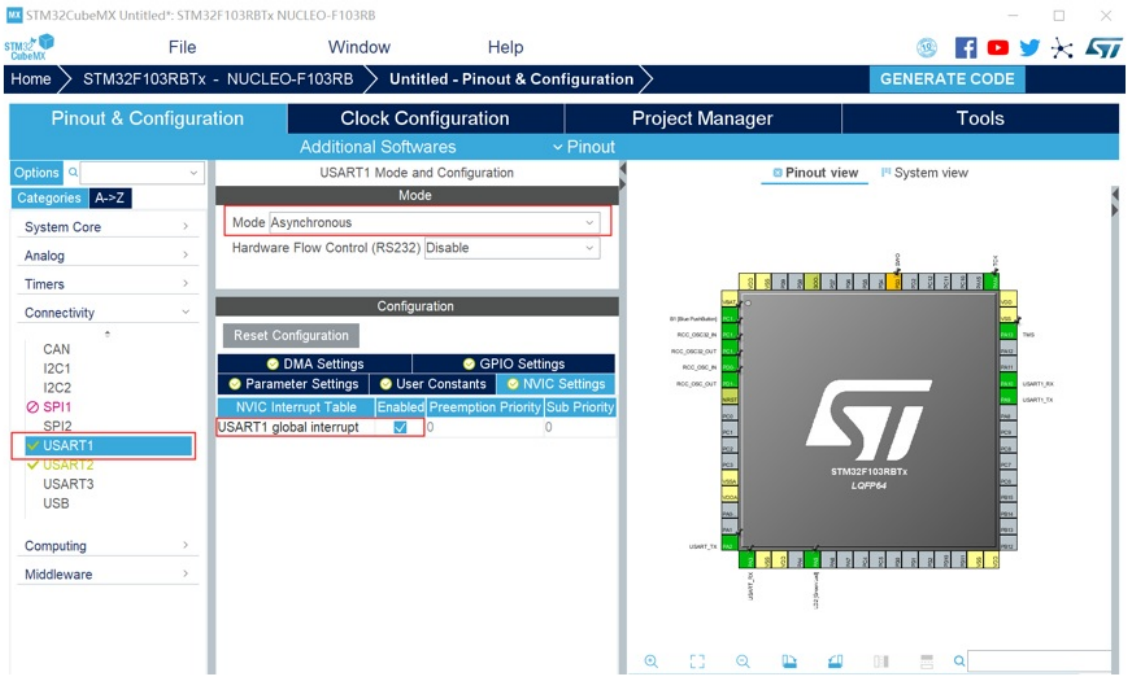
1. 打开STM32CubeMX，并选择新建项目。



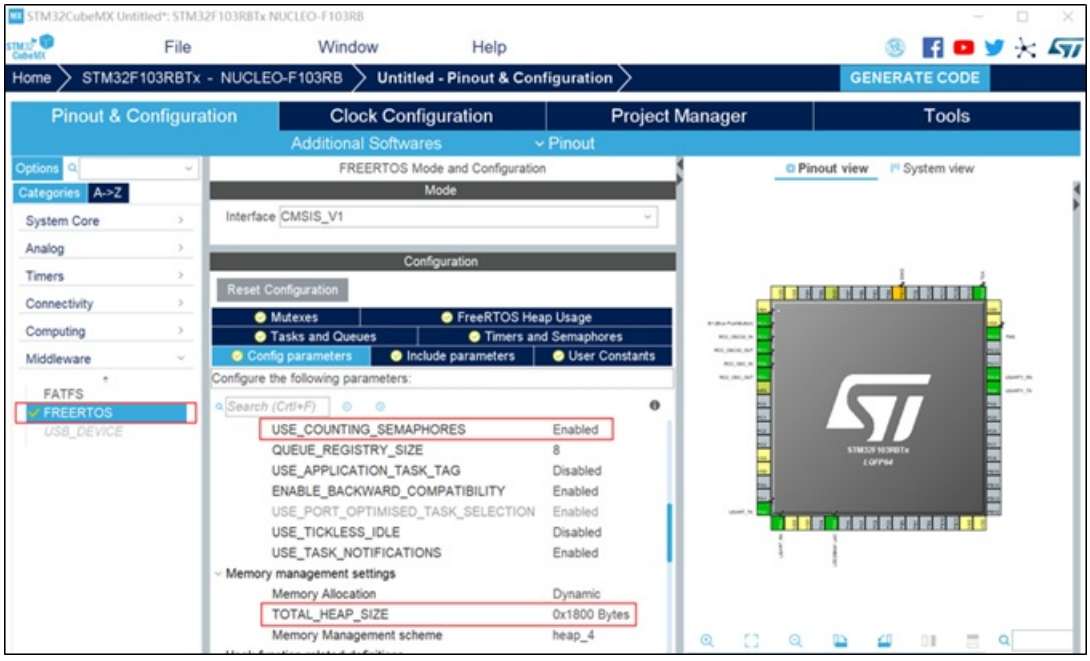
2. 在Board Selector中，搜索NUCLEO-F103RB，并单击STM32F103RBTx。



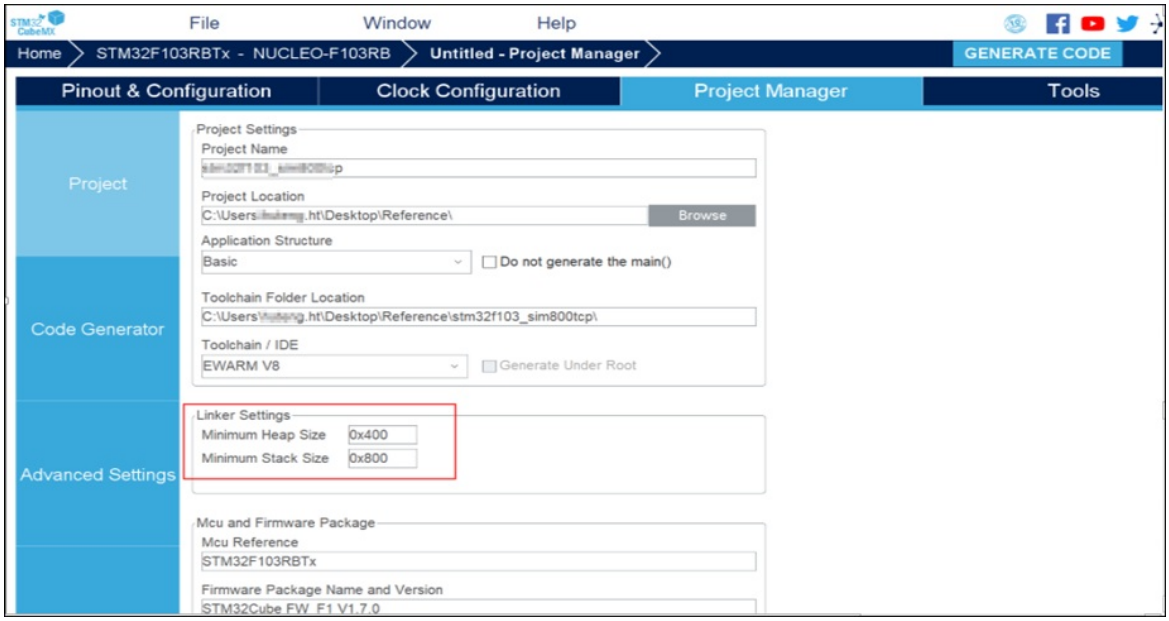
3. 单击右上角Start Project。
4. 在左侧Connectivity菜单中，勾选串口USART1作为MCU与模组通信的端口，并进行以下配置。
   - 设置Mode为Asynchronous。
   - 在Configuration栏，完成以下设置。
     - 在GPIO Settings下，确认Pin为PA9和PA10。
     - 在NVIC Settings下，将USART1 global interrupt设置为Enabled。



5. 在**Middleware**下，选择**FREERTOS**，并配置为使用计数信号量和堆大小，用于给每个线程分配栈。



6. 在**Project Manager**页签下，完成**Project**设置。



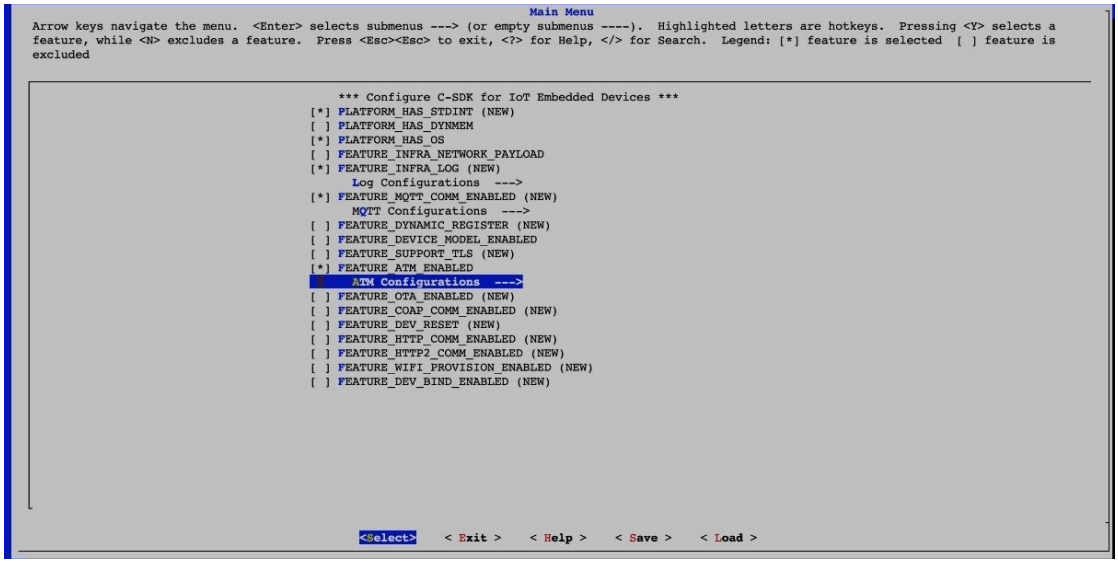
7. 单击右上角GENERATE CODE，生成代码工程。

1.9.4. 开发设备端

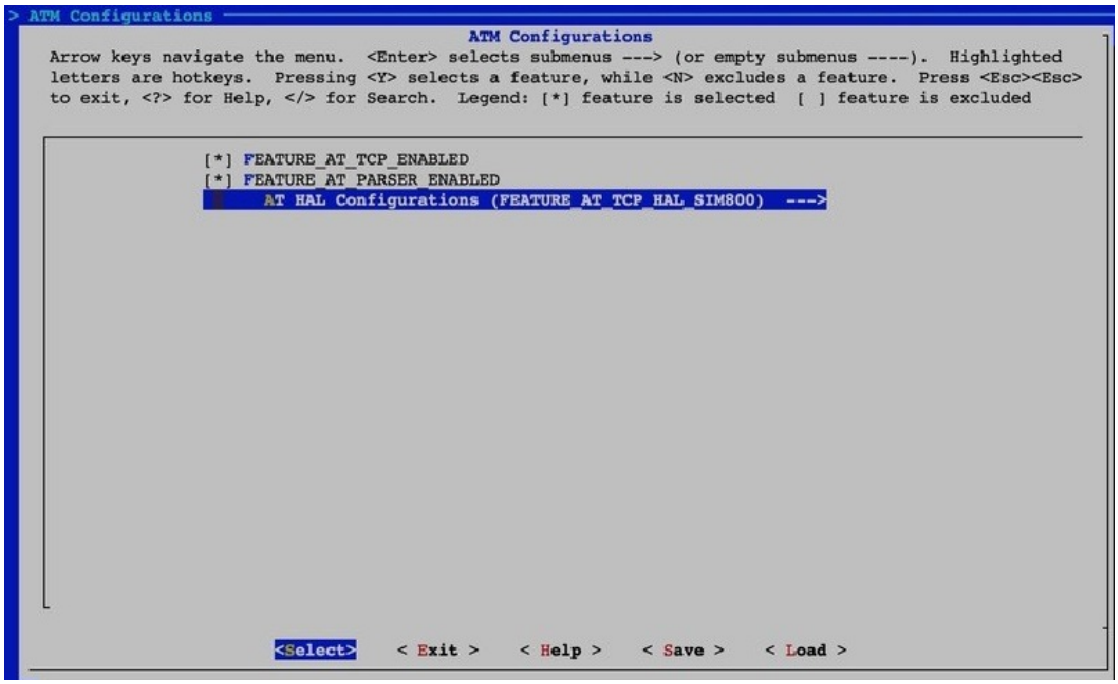
使用的C语言Link SDK将通信模组接入物联网平台。

设备端SDK配置

1. 下载C语言Link SDK 3.0.1版。
2. 从下载包中提取SDK代码。本文以Linux系统操作为例。
   i. 运行make menuconfig。
   ii. 选中ATM Configurations，单击Select。



iii. 选中AT HAL Configurations，单击Select。



iv. 配置如下项目。

```
FEATURE_PLATFORM_HAS_STDINT=y
FEATURE_PLATFORM_HAS_OS=y
FEATURE_INFRA_STRING=y
FEATURE_INFRA_NET=y
FEATURE_INFRA_LIST=y
FEATURE_INFRA_LOG=y
FEATURE_INFRA_LOG_ALL_MUTED=y
FEATURE_INFRA_LOG_MUTE_FLW=y
FEATURE_INFRA_LOG_MUTE_DBG=y
FEATURE_INFRA_LOG_MUTE_INF=y
FEATURE_INFRA_LOG_MUTE_WRN=y
FEATURE_INFRA_LOG_MUTE_ERR=y
FEATURE_INFRA_LOG_MUTE_CRT=y
FEATURE_INFRA_TIMER=y
FEATURE_INFRA_SHA256=y
FEATURE_INFRA_REPORT=y
FEATURE_INFRA_COMPAT=y
FEATURE_DEV_SIGN=y
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_MQTT_DEFAULT_IMPL=y
FEATURE_MQTT_DIRECT=y
FEATURE_DEVICE_MODEL_CLASSIC=y
FEATURE_ATM_ENABLED=y
FEATURE_AT_TCP_ENABLED=y
FEATURE_AT_PARSER_ENABLED=y
FEATURE_AT_TCP_HAL_SIM800=y
```

v. 配置完成后，运行./extract.sh提取代码。

```
[redacted]:~/c-sdk$ ./extract.sh
. Download request sent, waiting respond ...
. Respond generating, wait longer
. Retried 1/20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  107k  100  107k    0     0   408k      0 --:--:-- --:--:-- --:--:--  408k

Please pick up extracted source files in [/disk1/huteng/c-sdk/output]
```

提取的代码位于*output/eng*目录。

```
[redacted]:~/c-sdk/output/eng$ ls
atm  dev_sign  infra  mqtt  sdk_include.h  wrappers
```

其中，各子目录分别包含的代码如下表。

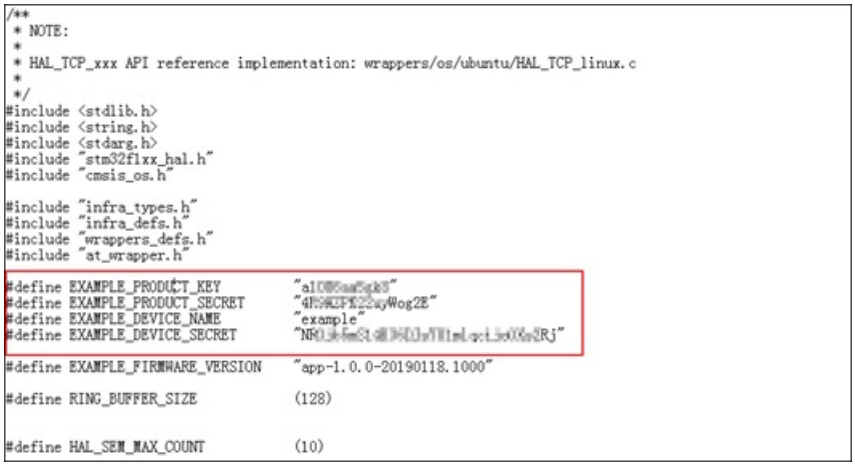
目录	代码内容
atm	AT指令收发模块
dev_sign	设备身份认证模块
infra	内部实现模块
mqtt	MQTT协议模块
wrappers	HAL对接模块

vi. 在*wrappers*目录下，新建文件*wrappers.c*，该文件中的代码需实现以下HAL函数。

```
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size, uint32
_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size, uint32_
t timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void *HAL_SemaphoreCreate(void)
void HAL_SemaphoreDestroy(void *sem)
void HAL_SemaphorePost(void *sem)
int HAL_SemaphoreWait(void *sem, uint32_t timeout_ms)
int HAL_ThreadCreate(void **thread_handle,
                     void *(*work_routine)(void *),
                     void *arg,
                     hal_os_thread_param_t *hal_os_thread_param,
                     int *stack_used)
void HAL_SleepMs(uint32_t ms)
void HAL_Printf(const char *fmt, ...)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

下载*wrappers.c*文件的代码Demo。

在代码Demo中，替换设备证书信息为您的设备证书信息。



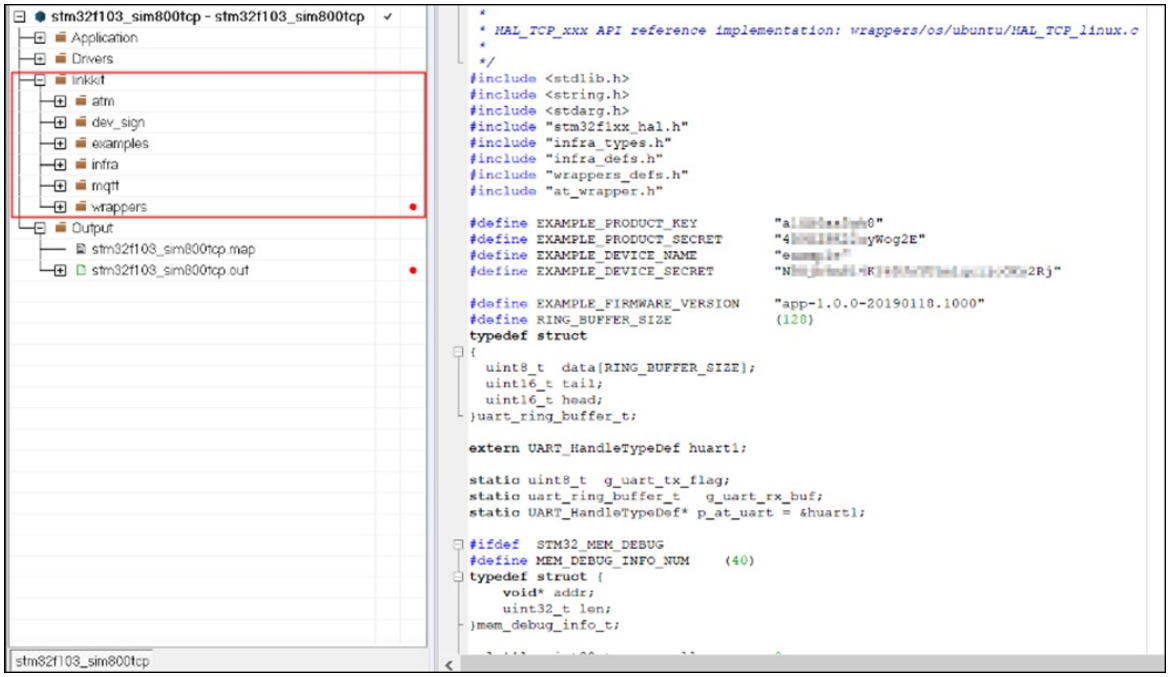
```
/**
 * NOTE:
 *
 * HAL_TCP_xxx API reference implementation: wrappers/os/ubuntu/HAL_TCP_linux.c
 *
 */
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "stm32f1xx_hal.h"
#include "cmsis_os.h"

#include "infra_types.h"
#include "infra_defs.h"
#include "wrappers_defs.h"
#include "at_wrapper.h"

#define EXAMPLE_PRODUCT_KEY         "a1[illegible]"
#define EXAMPLE_PRODUCT_SECRET      "4[illegible]Wog2E"
#define EXAMPLE_DEVICE_NAME         "example"
#define EXAMPLE_DEVICE_SECRET       "NP[illegible]2Rj"

#define EXAMPLE_FIRMWARE_VERSION    "app-1.0.0-20190118.1000"

#define RING_BUFFER_SIZE            (128)


#define HAL_SEM_MAX_COUNT           (10)
```

说明 如果您不是使用NUCLEO-F103RB通信模组开发板，需在配置时，设置 `FEATURE_AT_TCP_HAL_SIM800=n` 。且*wrappers.c*文件中代码需实现以下HAL函数。

```
int HAL_AT_CONN_Close(int fd, int32_t remote_port)
int HAL_AT_CONN_Deinit(void)
int HAL_AT_CONN_DomainToIp(char *domain, char ip[16])
int HAL_AT_CONN_Init(void)
int HAL_AT_CONN_Send(int fd, uint8_t *data, uint32_t len, char remote_ip[16], i
nt32_t remote_port, int32_t timeout)
int HAL_AT_CONN_Start(at_conn_t *conn)
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size, ui
nt32_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size, uin
t32_t timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void *HAL_SemaphoreCreate(void)
void HAL_SemaphoreDestroy(void *sem)
void HAL_SemaphorePost(void *sem)
int HAL_SemaphoreWait(void *sem, uint32_t timeout_ms)
int HAL_ThreadCreate(void **thread_handle,
                     void *(*work_routine)(void *),
                     void *arg,
                     hal_os_thread_param_t *hal_os_thread_param,
                     int *stack_used)
void HAL_SleepMs(uint32_t ms)
void HAL_Printf(const char *fmt, ...)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

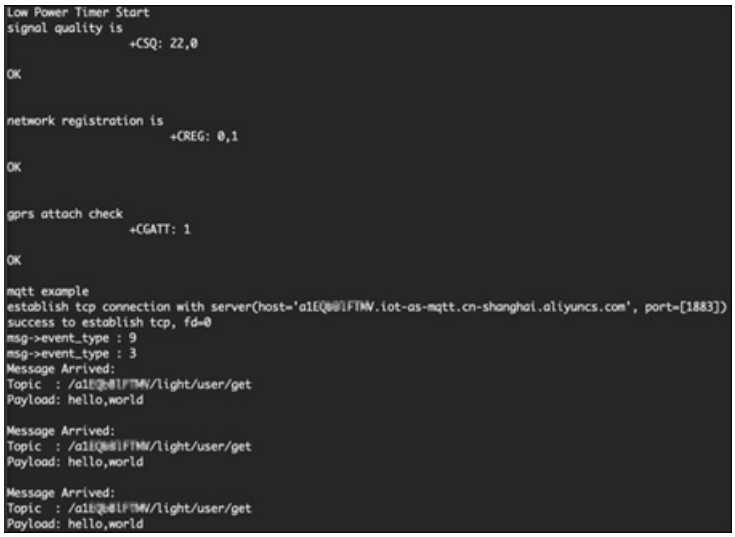
3. 将SDK整合到IAR工程中。

   如下图所示。



4. 运行SDK，进行测试。

   运行成功后，设备端本地日志如下图所示。



   登录物联网平台控制台，在**监控运维 > 日志服务**中，也可查看设备上报数据到云端的日志。

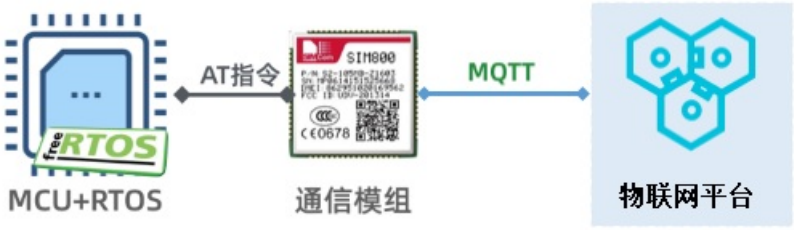
1.10. 无操作系统设备通过TCP模组上云

1.10.1. 概述

本实践案例介绍使用物联网平台提供的C语言设备端SDK，将无操作系统的微控制单元（MCU）的设备，通过MQTT协议接入阿里云物联网平台。

接入方案

将MCU与通信模组相连，MCU与通信模组间通过AT指令进行连接和通信。在通信模组上，使用C语言设备端SDK实现与物联网平台的连接和通信。



准备软硬件

本示例中，使用了如下MCU、通信模组开发板和软件开发环境：

- MCU为ST公司生产的STM32F103，其开发板为NUCLEO-F103RB。
- 通信模组为SIMCom公司（芯讯通无线科技有限公司）生产的SIM800C，其开发板为SIM800C mini V2.0。
- 开发环境为IAR Embedded Workbench for ARM。

开发过程

创建产品和设备

搭建设备端开发环境

开发设备端

1.10.2. 创建产品和设备

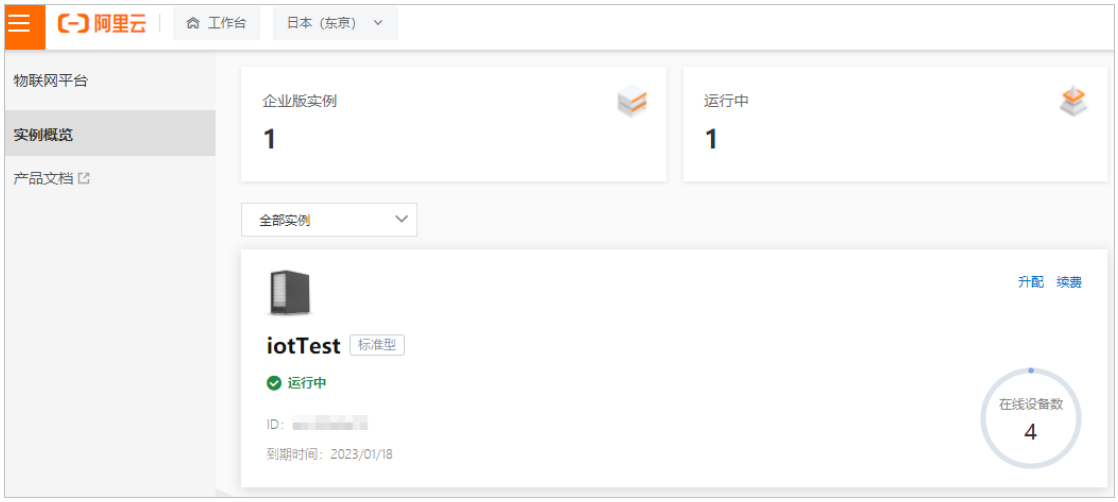
首先，在物联网平台控制台创建产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。设备证书信息需配置到设备端SDK中。当设备请求连接物联网平台时，物联网平台会根据设备证书信息进行设备身份验证。

操作步骤

1. 登录物联网平台控制台。
2. 创建产品。

i. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



ii. 在左侧导航栏，选择**设备管理 > 产品**。

iii. 在**产品**页，单击**创建产品**，填入产品信息，然后单击**确认**。

3. 在新创建的产品下添加设备。

i. 在左侧导航栏，选择**设备管理 > 设备**。

ii. 在**设备**页，单击**添加设备**，选择新创建的产品，输入设备DeviceName，然后单击**确认**。

执行结果

设备创建成功后，在弹出的**添加完成**对话框，单击**前往查看**或**一键复制设备证书**，获取设备证书。您也可以在**设备**页，单击设备对应的**查看**，进入**设备详情**页查看设备证书信息。

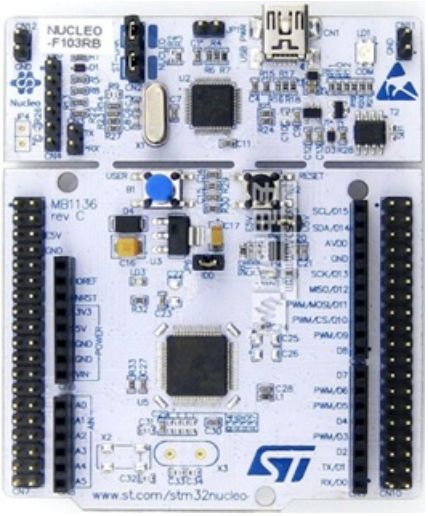
1.10.3. 搭建设备端开发环境

将MCU与通信模组开发板相连，搭建软件开发环境，创建工程项目，导入SDK，完成SDK配置。

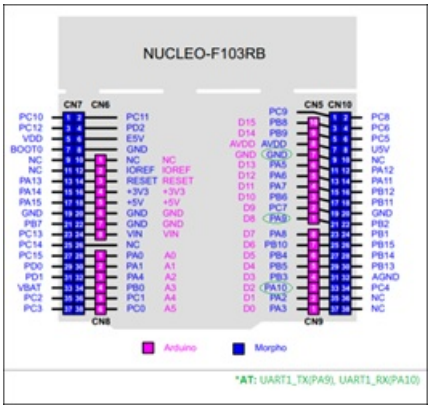
背景信息

本示例中使用了两个开发板示意图如下。

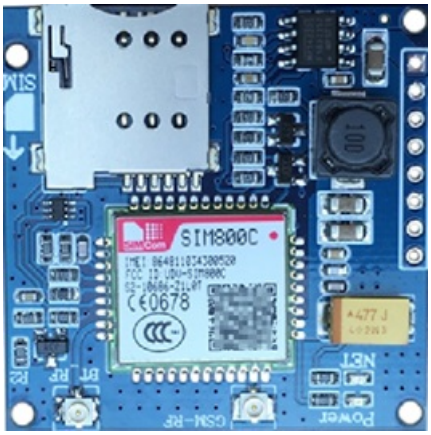
- 开发板NUCLEO-F103RB



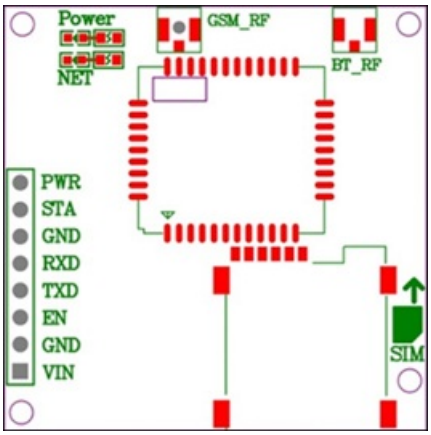
引脚示意图如下。



- SIM800C mini v2.0



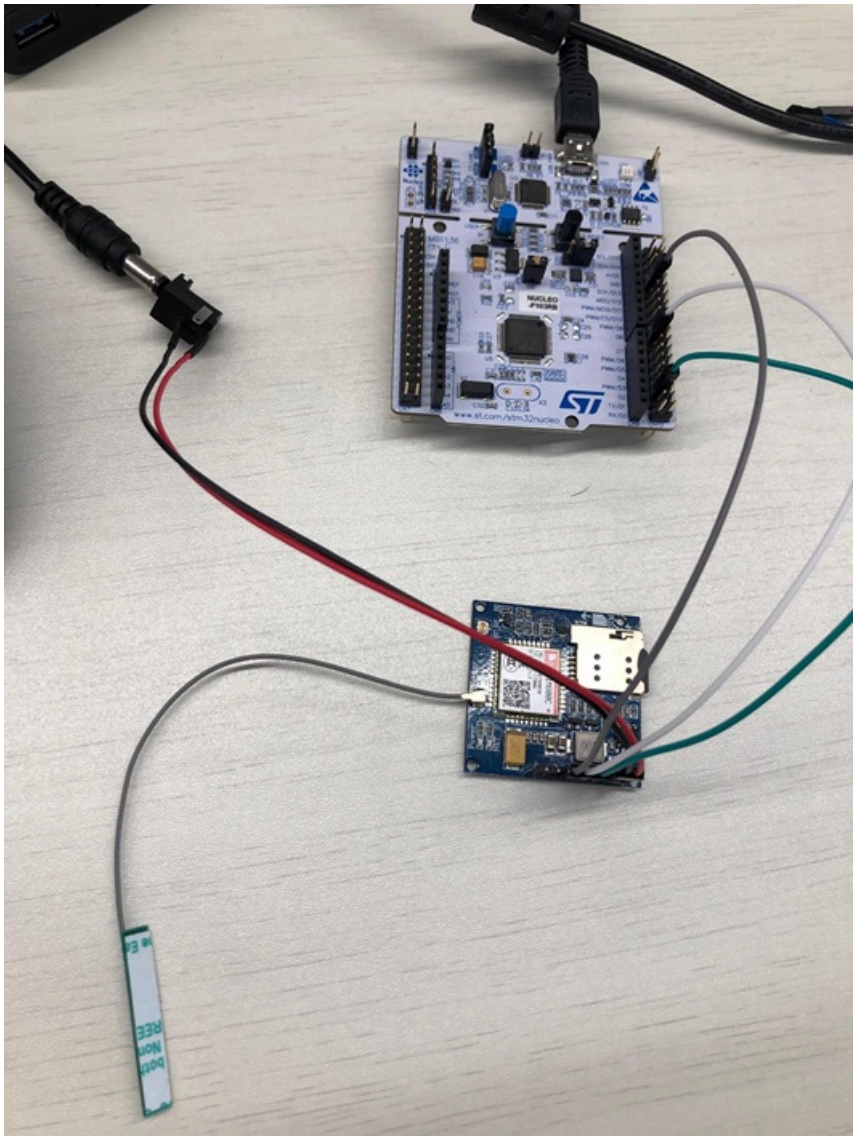
引脚示意图和说明如下。



引脚	说明
PWR	开关机引脚。默认为自动开机。
STA	状态监测引脚。
GND	电源接地引脚。
RXD	接收串口引脚。
TXD	发送串口引脚。
EN	电源使能引脚。
VIN	5~18V电源输入。

连接硬件

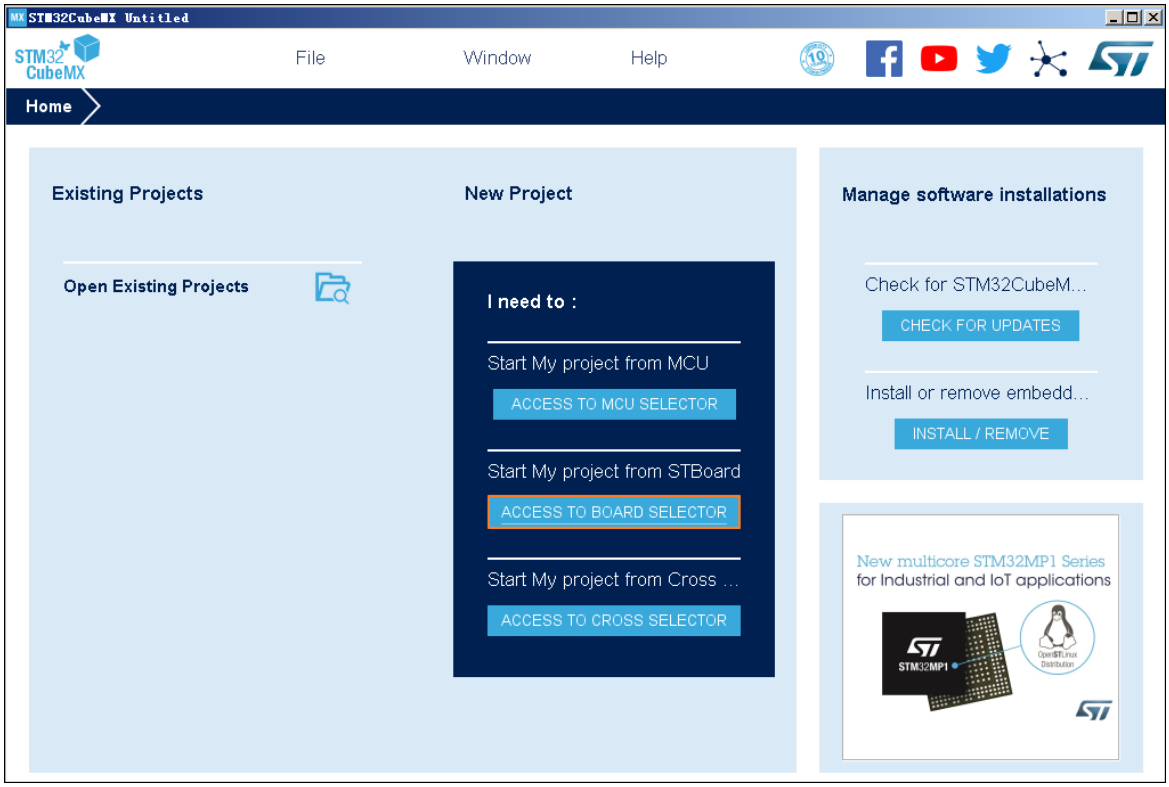
将两个开发板的接收和发送串口连接，作为AT指令通道，如下图所示。



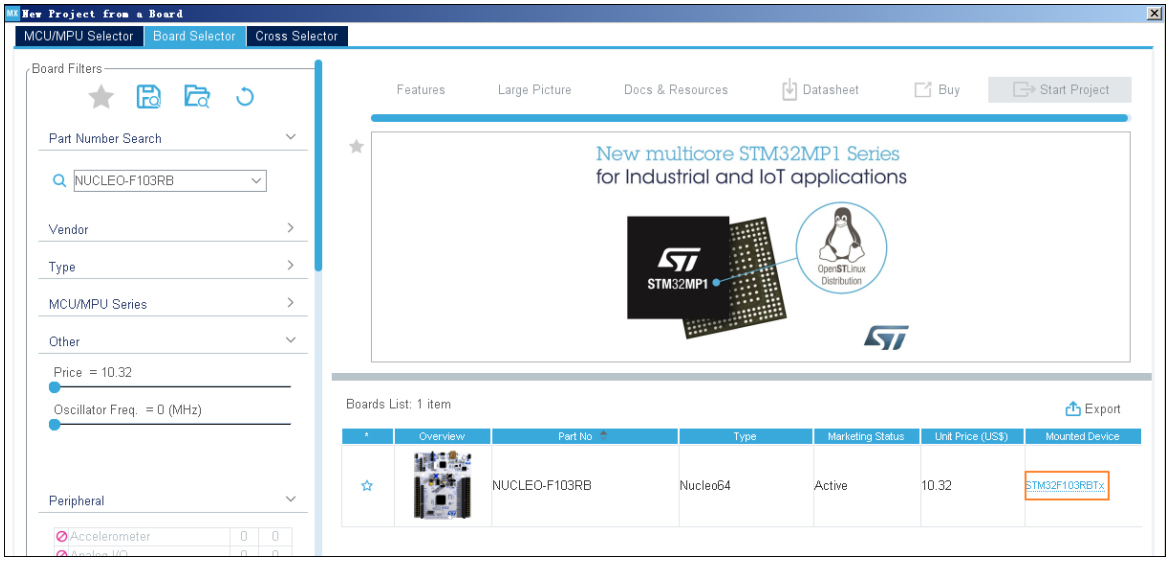
搭建开发环境

本示例开发工具为STM32CubeMX。使用详情请参见STM32Cube Ecosystem。

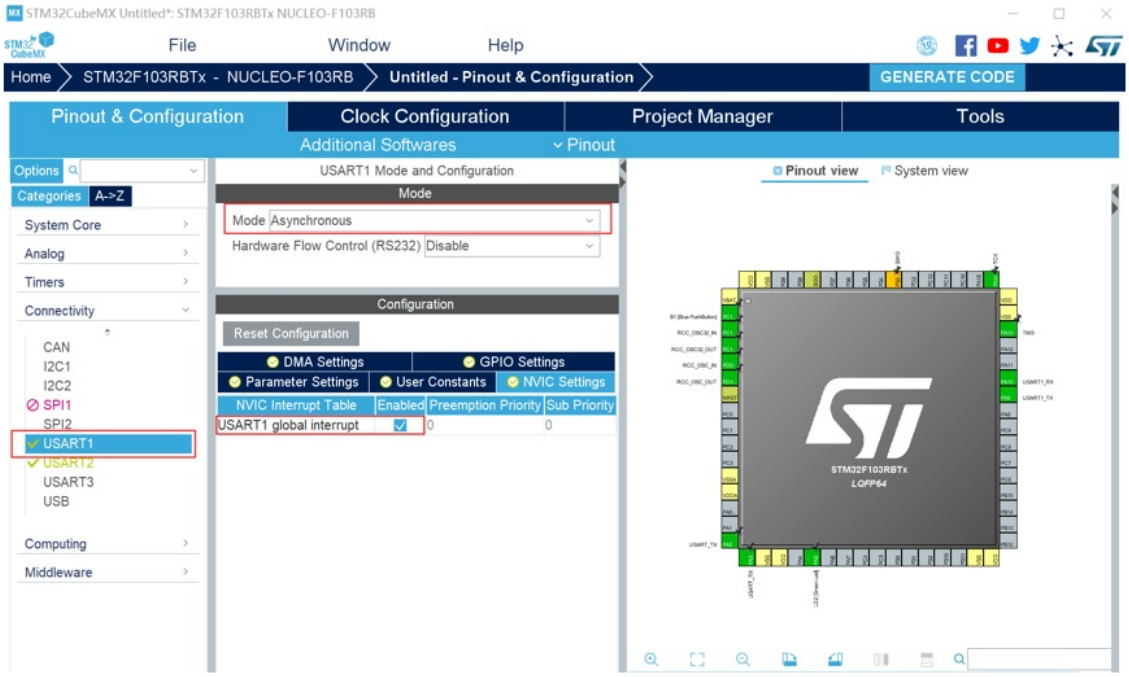
1. 打开STM32CubeMX，并选择新建项目。



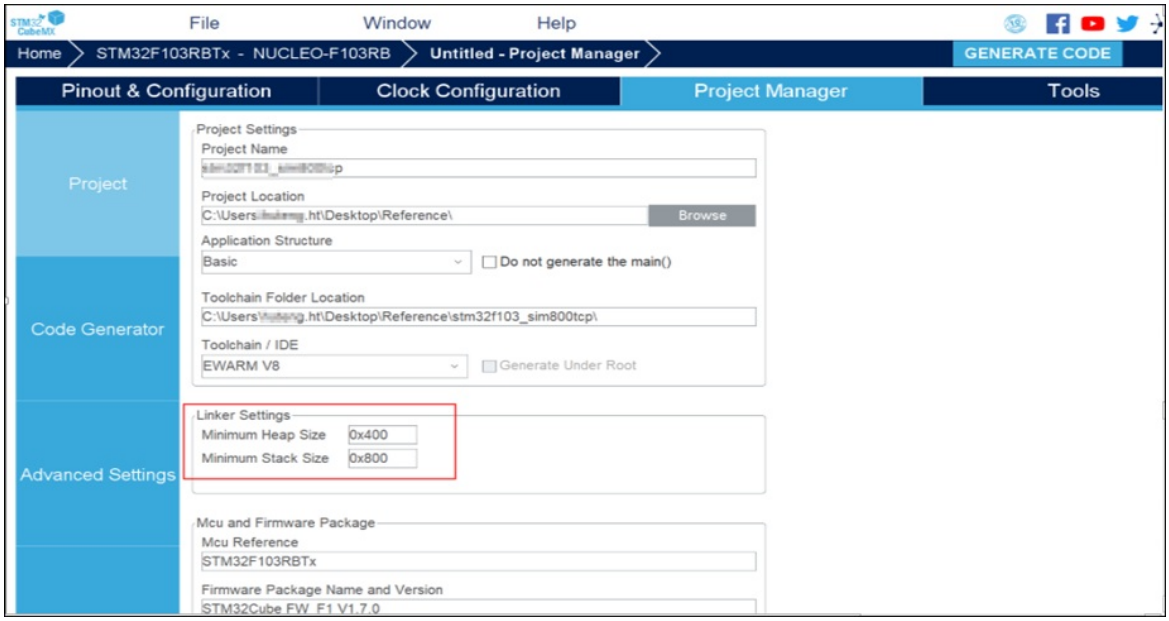
2. 在Board Selector中，搜索NUCLEO-F103RB，并单击STM32F103RBTx。



3. 单击右上角Start Project。
4. 在左侧Connectivity菜单中，勾选串口USART1作为MCU与模组通信的端口，并进行以下配置。
   - 设置Mode为Asynchronous。
   - 在Configuration栏，完成以下设置。
     - 在GPIO Settings下，确认Pin为PA9和PA10。
     - 在NVIC Settings下，将USART1 global interrupt设置为Enabled。



5. 在Project Manager页签下，完成Project设置。
   - Toolchain/IDE选择为EWARM V8。
   - Heap/Stack size按需进行配置。



6. 单击右上角GENERATE CODE，生成代码工程。

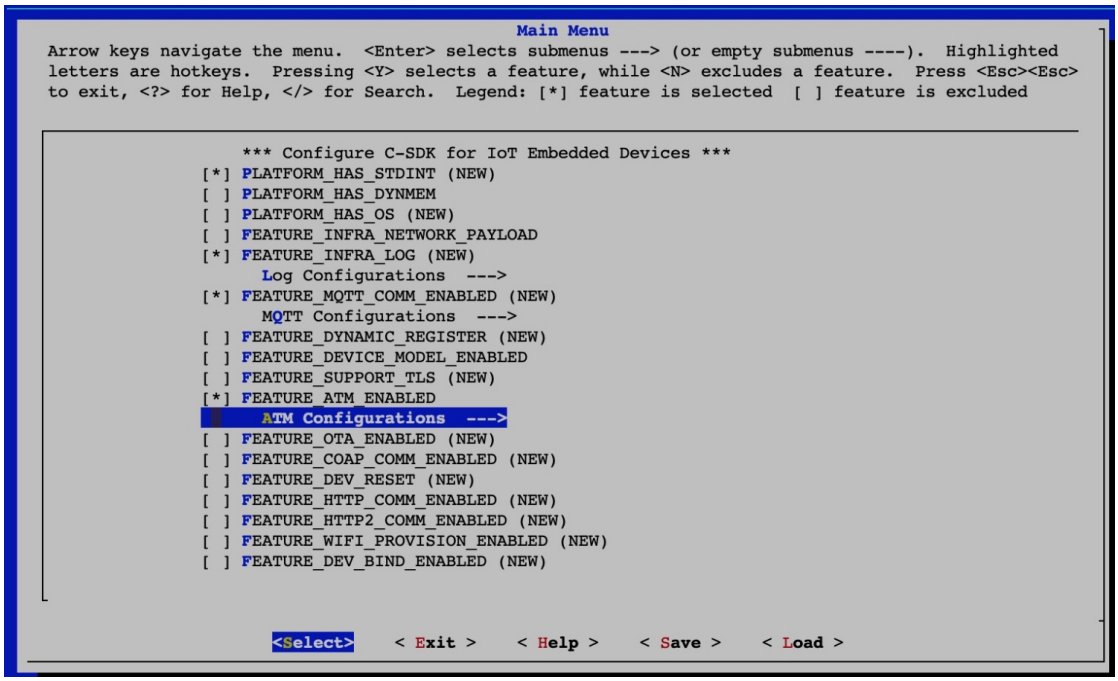
1.10.4. 开发设备端

使用的C语言Link SDK将通信模组接入物联网平台。

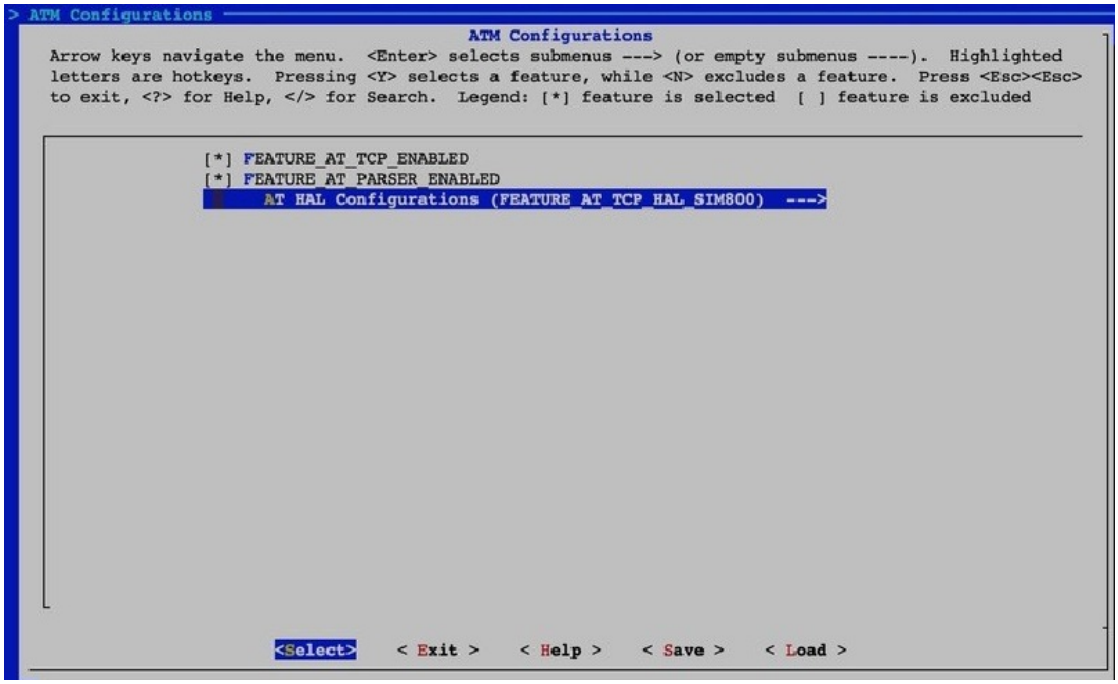
操作步骤

1. 下载C语言Link SDK 3.0.1版。

2. 从下载包中提取SDK代码。本文以Linux系统操作为例。
   i. 运行make menuconfig。
   ii. 选中ATM Configurations，单击Select。



   iii. 选中AT HAL Configurations，单击Select。



iv. 配置如下项目。

```
FEATURE_PLATFORM_HAS_STDINT=y
FEATURE_INFRA_STRING=y
FEATURE_INFRA_NET=y
FEATURE_INFRA_LIST=y
FEATURE_INFRA_LOG=y
FEATURE_INFRA_LOG_ALL_MUTED=y
FEATURE_INFRA_LOG_MUTE_FLW=y
FEATURE_INFRA_LOG_MUTE_DBG=y
FEATURE_INFRA_LOG_MUTE_INF=y
FEATURE_INFRA_LOG_MUTE_WRN=y
FEATURE_INFRA_LOG_MUTE_ERR=y
FEATURE_INFRA_LOG_MUTE_CRT=y
FEATURE_INFRA_TIMER=y
FEATURE_INFRA_SHA256=y
FEATURE_INFRA_REPORT=y
FEATURE_INFRA_COMPAT=y
FEATURE_DEV_SIGN=y
FEATURE_MQTT_COMM_ENABLED=y
FEATURE_MQTT_DEFAULT_IMPL=y
FEATURE_MQTT_DIRECT=y
FEATURE_DEVICE_MODEL_CLASSIC=y
FEATURE_ATM_ENABLED=y
FEATURE_AT_TCP_ENABLED=y
FEATURE_AT_PARSER_ENABLED=y
FEATURE_AT_TCP_HAL_SIM800=y
```

v. 配置完成后，运行./extract.sh提取代码。

```
huteng@r10c05067:~/c-sdk$ ./extract.sh
. Download request sent, waiting respond ...
. Respond generating, wait longer
. Retried 1/20

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  107k  100  107k    0     0   408k      0 --:--:-- --:--:-- --:--:--  408k

Please pick up extracted source files in [/disk1/huteng/c-sdk/output]
```

提取的代码位于*output/eng*目录。

```
[huteng@r10c05067:~/c-sdk/output/eng$ ls
atm  dev_sign  infra  mqtt  sdk_include.h  wrappers
```

其中，各子目录分别包含的代码如下表。

目录	代码内容
atm	AT指令收发模块
dev_sign	设备身份认证模块
infra	内部实现模块
mqtt	MQTT协议模块
wrappers	HAL对接模块

3. 在*wrappers*目录下，新建文件*wrappers.c*，该文件中的代码需实现以下HAL函数。

```
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size,
uint32_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size,
uint32_t timeout)
int HAL_GetFirmwareVersion(char *version)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN])
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void HAL_Printf(const char *fmt, ...)
void HAL_SleepMs(uint32_t ms)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

下载*wrappers.c*文件的代码Demo。

在代码Demo中，替换设备证书信息为您的设备证书信息。

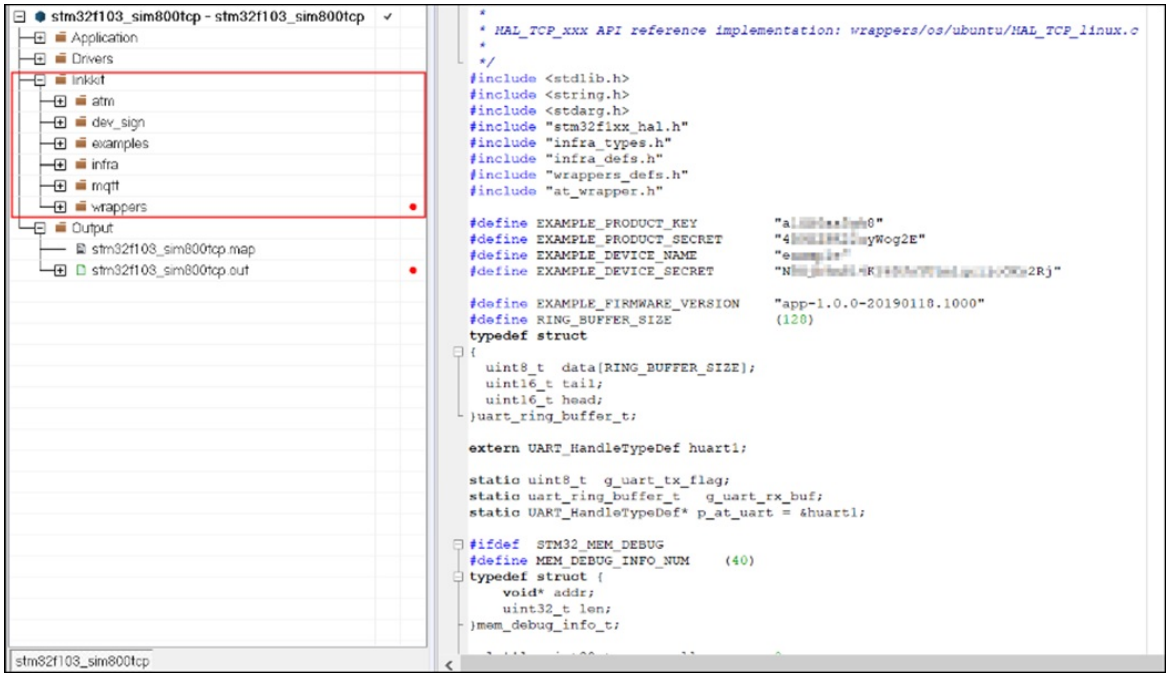
```
* NOTE:
*
* HAL_TCP_xxx API reference implementation: wrappers/os/ubuntu/HAL_TCP_linux.c
*
*/
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include "stm32f1xx_hal.h"
#include "infra_types.h"
#include "infra_defs.h"
#include "wrappers_defs.h"
#include "at_wrapper.h"

#define EXAMPLE_PRODUCT_KEY         "a1[illegible]"
#define EXAMPLE_PRODUCT_SECRET      "4R[illegible]Wog2E"
#define EXAMPLE_DEVICE_NAME         "example"
#define EXAMPLE_DEVICE_SECRET       "NRO[illegible]j"

#define EXAMPLE_FIRMWARE_VERSION    "app-1.0.0-20190118.1000"
#define RING_BUFFER_SIZE            (128)
typedef struct
{
  uint8_t  data[RING_BUFFER_SIZE];
  uint16_t tail;
  uint16_t head;
}uart_ring_buffer_t;
```

说明 如果通信模组为其他模组，则配置 `FEATURE_AT_TCP_HAL_SIM800=n` ，且需实现的HAL函数列表如下所示。

```
int HAL_AT_CONN_Close(int fd, int32_t remote_port)
int HAL_AT_CONN_Deinit(void)
int HAL_AT_CONN_DomainToIp(char *domain, char ip[16])
int HAL_AT_CONN_Init(void)
int HAL_AT_CONN_Send(int fd, uint8_t *data, uint32_t len, char remote_ip[16], int32
_t remote_port, int32_t timeout)
int HAL_AT_CONN_Start(at_conn_t *conn)
int32_t HAL_AT_Uart_Deinit(uart_dev_t *uart)
int32_t HAL_AT_Uart_Init(uart_dev_t *uart)
int32_t HAL_AT_Uart_Recv(uart_dev_t *uart, void *data, uint32_t expect_size, uint32
_t *recv_size, uint32_t timeout)
int32_t HAL_AT_Uart_Send(uart_dev_t *uart, const void *data, uint32_t size, uint32_
t timeout)
int HAL_GetDeviceName(char device_name[IOTX_DEVICE_NAME_LEN + 1])
int HAL_GetDeviceSecret(char device_secret[IOTX_DEVICE_SECRET_LEN + 1])
int HAL_GetFirmwareVersion(char *version)
int HAL_GetProductKey(char product_key[IOTX_PRODUCT_KEY_LEN + 1])
void *HAL_Malloc(uint32_t size)
void HAL_Free(void *ptr)
void *HAL_MutexCreate(void)
void HAL_MutexDestroy(void *mutex)
void HAL_MutexLock(void *mutex)
void HAL_MutexUnlock(void *mutex)
void HAL_Printf(const char *fmt, ...)
void HAL_SleepMs(uint32_t ms)
int HAL_Snprintf(char *str, const int len, const char *fmt, ...)
uint64_t HAL_UptimeMs(void)
```

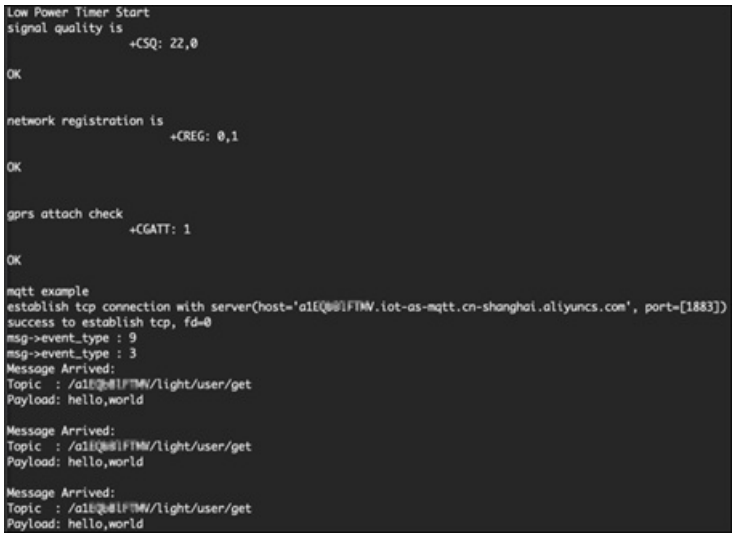
4. 将SDK整合到IAR工程中。

   如下图所示。



5. 运行SDK，进行测试。

   运行成功后，设备端本地日志如下图所示。



   登录物联网平台控制台，在**监控运维 > 日志服务**中，也可查看设备上报数据到云端的日志。

1.11. 设备通过DTU接入物联网平台

1.11.1. 概述

物联网平台支持使用串口通信的设备，在不改变原有的串口传输协议的情况下，通过DTU接入物联网平台。

案例场景

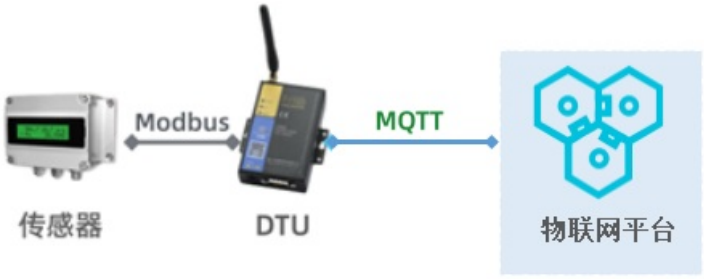
在工业、农业、医疗、城市、楼宇、园区等多种场景中，存在着大量的通过串口与外界通信的设备。对此类设备进行物联网改造时，往往无法修改设备本身的串口传输协议，只能在云端进行数据解析工作。为了快速使此类设备接入和使用阿里云物联网平台，阿里云联合硬件合作伙伴，共同定义了可以通过简单配置即可接入物联网平台的透传数据DTU设备。

本文将以电机转速控制设备为例，介绍如何通过符合阿里云物联网平台接入协议规范的DTU设备，快速实现串口输出设备接入阿里云物联网平台。

实现原理

将物联网平台颁发的设备证书配置到DTU中，由DTU代理设备与物联网平台进行数据通信。设备通过串口与DTU相连，DTU通过2G、3G、4G、5G或Ethernet网络与阿里云物联网平台相连，由DTU实现阿里云物联网平台的接入协议。

数据流转如下图所示。



实现流程

1. 配置产品和设备。
2. 配置DTU设备。
3. 测试数据通信。

1.11.2. 配置产品和设备

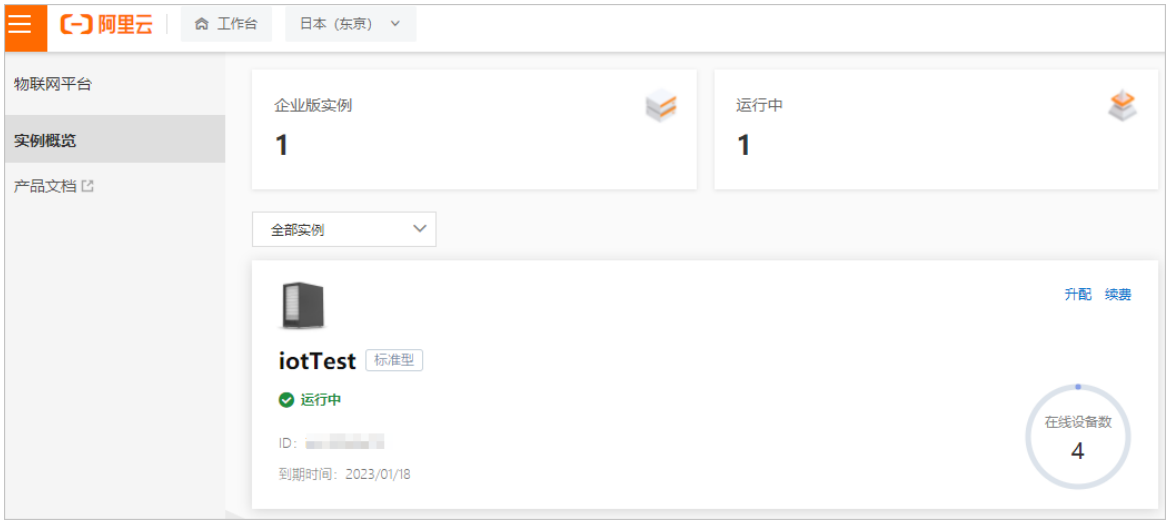
设备通过DTU接入物联网平台前，您需要在物联网平台上依次完成以下操作：创建数据格式为透传/自定义的产品、创建设备、获取设备证书信息、定义物模型、编辑并提交数据解析脚本。

创建产品和设备

在物联网平台创建产品和设备，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，选择**设备管理 > 产品**，再单击**创建产品**，创建一个产品：**电机变频**。

   产品信息

参数	说明
产品名称	自定义产品名称。例如**电机变频**。
所属品类	选择**自定义品类**。
节点类型	选择**直连设备**。
连网方式	选择**蜂窝（2G/3G/4G/5G）**。
数据格式	选择**透传/自定义**。
认证方式	选择**设备密钥**。

4. 在左侧导航栏，选择**设备**，再单击**添加设备**，在创建的**电机变频**产品下，添加设备。
   设备创建成功后，您也可以在**设备**页，单击设备对应的**查看**，进入**设备详情**页查看设备证书信息。该设备证书将被配置到DTU设备端，请妥善保存。

定义物模型

本文以电机变频设备为例，需创建电机转速、电流和设置转速三个属性。物模型相关概念说明，请参见什么是物模型。

1. 在物联网平台控制台左侧导航栏，选择**设备管理 > 产品**。
2. 在**产品**页面，单击**电机变频**产品对应的**查看**。
3. 在**产品详情**页面的**功能定义**页签下，选择**编辑草稿 > 添加自定义功能**。
4. 根据下表逐个添加属性，然后单击**发布上线**，将物模型发布为正式版。

功能类型	功能名称	标识符	数据类型	取值范围	步长	单位	读写类型
属性	转速	speed	int32	0 ~ 3000	1	r/min	只读
属性	电流	current	int32	0 ~ 300	1	A	只读

功能类型	功能名称	标识符	数据类型	取值范围	步长	单位	读写类型
属性	设置转速	setspeed	int32	0 ~ 3000	1	r/min	读写

编写数据解析脚本

阿里云物联网平台支持的标准数据格式为Alink JSON格式，而设备的原始数据通过DTU设备透传到物联网平台，物联网平台不能直接处理此类数据。

物联网平台提供数据解析功能，可将上行的自定义格式数据解析为Alink JSON格式；将下行数据解析为设备的自定义数据格式。您需在物联网平台控制台上，提交数据解析脚本供物联网平台调用。数据解析脚本需根据设备上报数据和物联网平台下发数据进行编写。

1. 在**电机变频**产品的**产品详情**页面，选择**数据解析**页签。
2. 在**数据解析**页签下的**编辑脚本**输入框中，选择脚本语言**JavaScript（ECMAScript 5）**，然后输入数据解析脚本。

   数据解析脚本编写指导，请参见提交数据解析脚本。

   本示例设备发送至物联网平台的数据为16进制格式，因此脚本需将16进制格式数据转换为Alink JSON格式，并将物联网平台下发的Alink JSON格式数据转换为16进制格式。

   本示例的数据解析脚本如下。

```
var ALINK_ID = "12345";
var ALINK_VERSION = "1.1";
var ALINK_PROP_POST_METHOD = 'thing.event.property.post';
// var ALINK_EVENT_TEMPERR_METHOD = 'thing.event.TempError.post';
// var ALINK_EVENT_HUMIERR_METHOD = 'thing.event.HumiError.post';
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';
// var ALINK_SERVICE_THSET_METHOD = 'thing.service.SetTempHumiThreshold';
/* * * * * *
* 上报数据 ->
* 0102 // 共2个字节 * 解析结果 ->
* {"method":"thing.event.TempError.post","id":"12345","params":{"Temperature": 2},"vers
ion":"1.1"}
* 上报数据 ->
* 0202 // 共2个字节 * 解析结果 ->
* {"method":"thing.event.HumiError.post","id":"12345","params":{"Humidity":2}, "version
":"1.1"}
*/
/*此函数将设备上报数据转换为Alink JSON物模型数据。*/
function rawDataToProtocol(bytes) {
    /*将设备上报的原始数据转换为数组。其中bytes对象中存储着设备上报原始数据。*/
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var params = {};                                // 定义属性存放对象。
    var jsonMap = {};                               // 定义模拟Alink数据报对象。
    /*填写Alink数据报协议头部分。*/
    jsonMap['version'] = ALINK_VERSION;             // Alink 协议版本号。
    jsonMap['id'] = ALINK_ID;                       // 消息ID。
    jsonMap['method'] = ALINK_PROP_POST_METHOD;     // 设备上行数据方法：设备属性上报。
    /*填写Alink数据报属性部分。*/
    params['speed'] = uint8Array[0];                // 将收到的第一个字节转换为转速值。
```

```
    params['speed'] = uint8Array[0];                    // 将收到的第一个字节转换为转速值。
    params['current'] = uint8Array[1];                  // 将收到的第二个字节转换为电流。
    jsonMap['params'] = params;                         // 将参数打包到数据帧中。
    return jsonMap;                                     // 返回结果会发送给物联网平台。
}
//以下是部分辅助函数。
function buffer_uint8(value)
{
    var uint8Array = new Uint8Array(1);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value)
{
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value)
{
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value)
{
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
/*此函数实现由物联网平台下发数据转换为设备能识别的16进制数。*/
function protocolToRawData(json)
{
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD)     // 接收来自物联网平台的"设置设备属性"的命令。
    {
        var send_params = json['params'];
        var prop_cur = send_params['setspeed'];    // 将设置的具体值抽取出来。
        //按照自定义协议格式拼接rawdata。
        payloadArray = payloadArray.concat(buffer_uint8(0x55)); // 第一字节数据头，标识数据功能用户自定义。
        payloadArray = payloadArray.concat(buffer_uint8(prop_cur)); // 第二字节，具体的设置值。
    }
    return payloadArray;     // 返回时，将数据发送至设备端。
}
function transformPayload(topic, rawData) {
    var jsonObj = {};
```

```
    return jsonObj;
}
```

3. 测试脚本。
   - 测试解析设备上报数据。
     a. 选择模拟类型为**设备上报数据**。
     b. 在**模拟输入**下的输入框中，输入一个模拟数据。

        本示例脚本的逻辑为：数据的第一个字节为转速值，第二个字节为电流值。例如输入6410，第一个字节64表示转速为100 r/min，第二个字节10表示电流为16 A。

     c. 单击**执行**。

        运行结果栏显示解析结果如下图。

   - 测试物联网平台下行数据解析。
     a. 选择模拟类型为**设备接收数据**。
     b. 在**模拟输入**下的输入框中，输入一个模拟数据。下行数据示例如下：

```
{
    "method": "thing.service.property.set",
    "id": "12345",
    "version": "1.0",
    "params": {
        "setspeed": 123
    }
}
```

c. 单击**执行**。

运行结果栏显示解析结果如下图。

4. 确认脚本能正确解析数据后，单击**提交**，将脚本提交到物联网平台系统。

> **说明** 物联网平台不能调用草稿状态的脚本，只有已提交的脚本才会被调用来解析数据。

后续步骤

配置DTU设备

1.11.3. 配置DTU设备

本文介绍使用DTU配置工具，配置DTU串口和设备证书信息（ProductKey、DeviceName和DeviceSecret），实现DTU代理设备接入物联网平台。

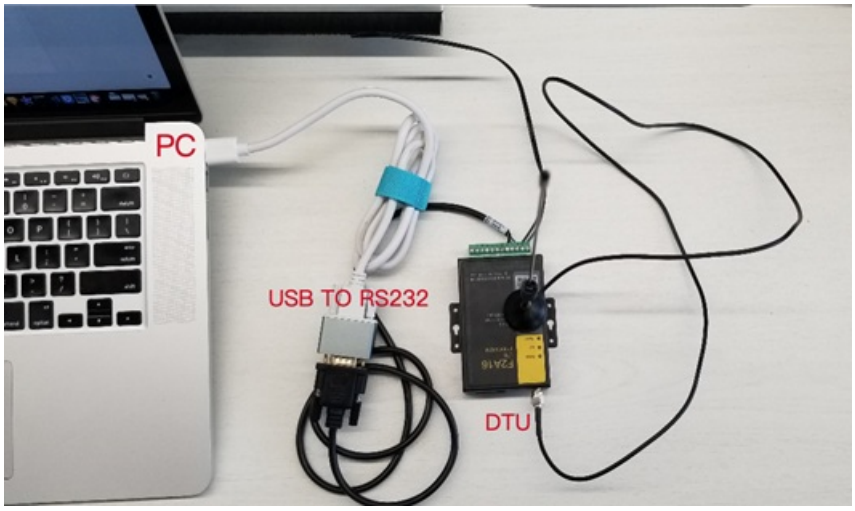
背景信息

本示例使用F2x16 DTU设备，并使用电脑模拟DTU设备端进行开发配置。通过USB转串口将电脑与DTU设备连接。

> **说明** 请确保DTU可以正确连接Internet。

操作步骤

1. 连接DTU设备与电脑USB接口。



2. 在电脑上，打开DTU配置工具。本示例使用F2x16配置工具。
3. 配置正确的串口号，设置波特率，并打开串口。

4. 登录配置（如图①），使DTU进入配置状后，单击**读取配置**（如图②），获取现有DTU的配置。

5. 在右侧**配置界面**下，单击**串口**，再进行本地串口配置。

> 说明 确保工作协议为port。

配置信息如下图。

6. 单击**IoT接入配置**，填入从物联网平台获取的设备证书信息和地域。

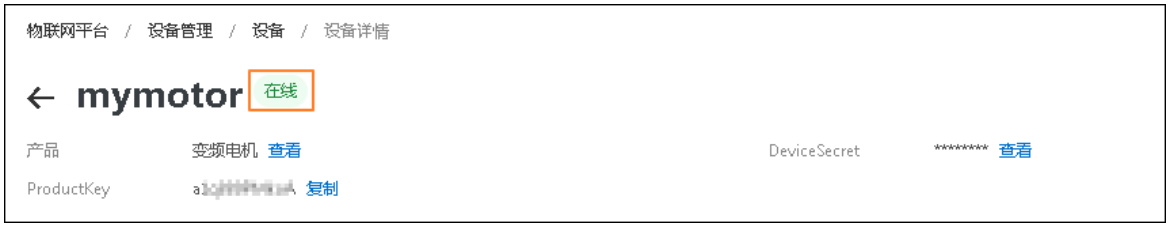
7. 单击下方**下发配置**，使配置生效。

   如果配置下发失败，请单击退出登录的按钮 退出登陆 ，然后重新配置。

8. 完成配置后，请单击退出登录的按钮 退出登陆 ，才能使DTU进入正常工作模式。

9. 将DTU断电，再重新上电。DTU上online灯亮，即表示已连接上物联网平台。

   您还可以在物联网平台控制台上，查看设备的状态。



后续步骤

测试数据通信

1.11.4. 测试数据通信

本文介绍如何测试DTU代替设备上报数据到物联网平台和接收物联网平台下发的数据。

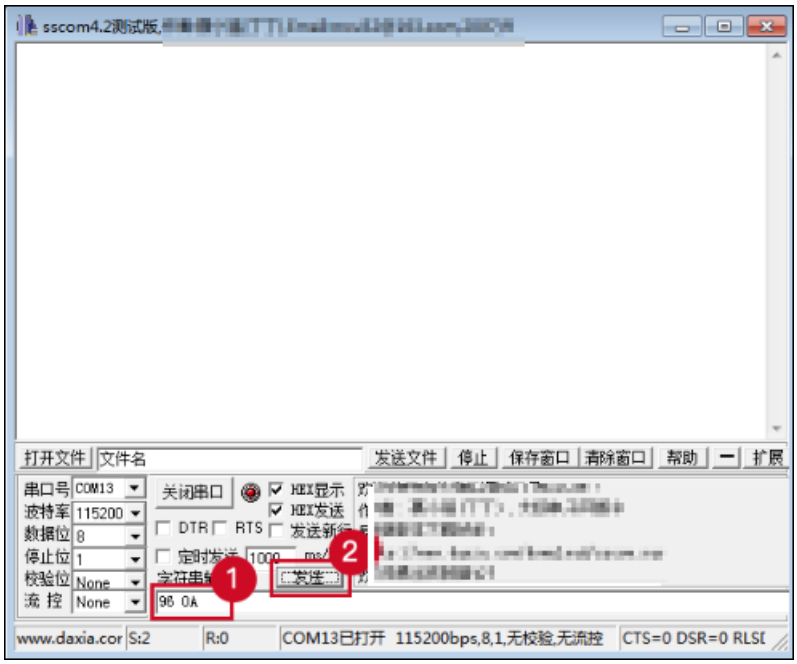
测试上报数据

1. 打开串口调试工具。本示例中使用sscom 4.2测试版。

   > **说明** 在本地电脑上使用串口调试工具模拟收发数据前，请务必确保DTU配置工具已经关闭。

2. 设置串口调试工具的相关参数，并打开串口，然后单击**发送**。

   根据物联网平台上的物模型定义，模拟发送转速和电流两个数据到云端。假定转速为150，电流为10安培，则在串口工具中，按先后顺序填写96 0A两个16进制数。



3. 数据发送后，开启设备状态实时刷新。

   i. 登录物联网平台控制台，在左侧导航栏，选择**设备管理 > 设备**。

   ii. 单击设备对应的**查看**。

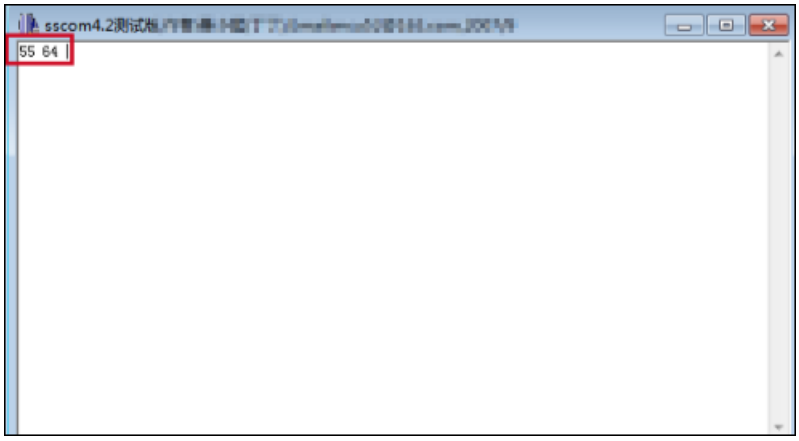
   iii. 在**设备详情**页面**物模型数据**页签的**运行状态**页签下，打开**实时刷新**开关。
   稍后就可以看到上传的属性数据。

测试接收云端下发数据

使用物联网平台**在线调试**功能，下发设置转速指令，测试DTU接收云端下发数据。

1. 登录物联网平台控制台，在左侧导航栏，选择**监控运维 > 在线调试**。
2. 选择要调试的设备。
3. 在**属性调试**页签，选择功能为已定义的转速设置属性，输入一个测试值，单击**设置**。
4. 指令发送成功后，在DTU串口调试工具的接收框中，查看接收到的数据。

   接收到的数据中，55为数据头，数据值为64（即十进制的100）。



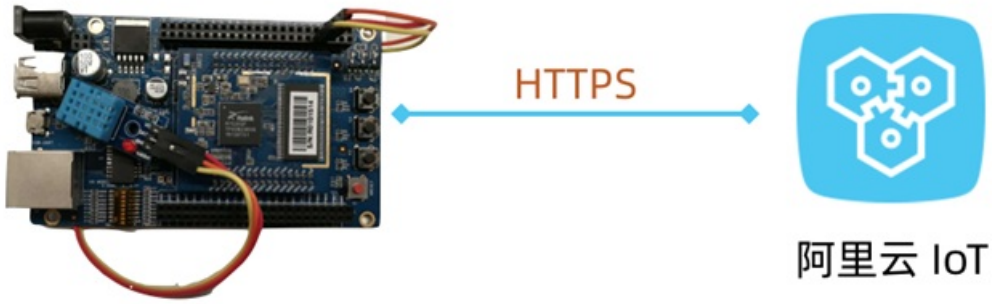
云端和设备端均能接收到正确数据，说明配置成功。

1.12. 温湿度采集设备以HTTPS方式上云

物联网平台华东2（上海）地域支持设备使用HTTPS协议接入。设备与物联网平台通过HTTPS协议进行连接通信仅适用于单纯的设备上报数据场景。请求方式仅支持POST，且设备上报的数据不超过128 KB。

背景信息

本实践案例以温湿度采集器为例，介绍设备通过HTTPS协议连接物联网平台并上报数据的配置和开发方法。

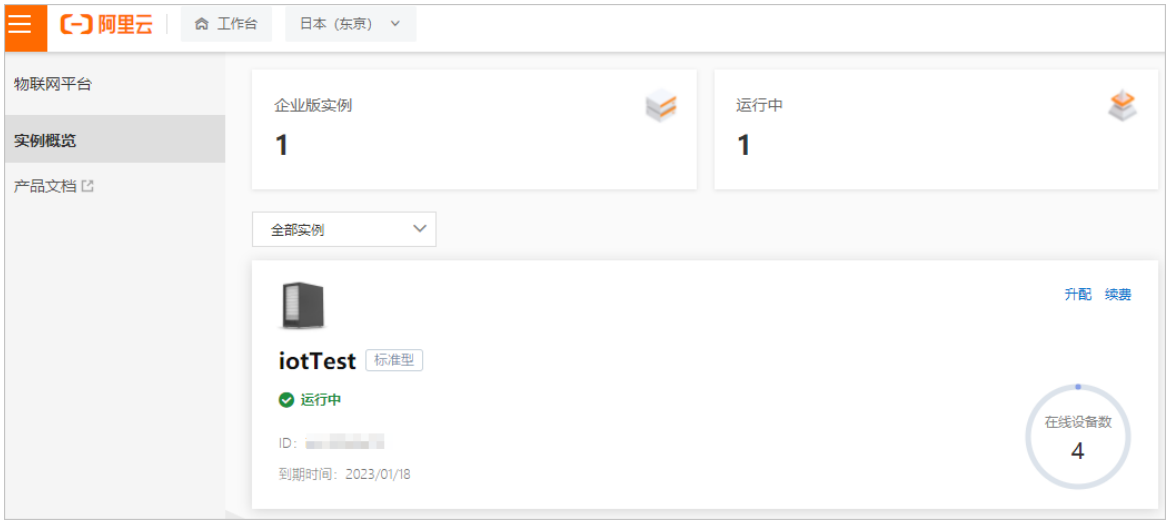


创建产品和设备

在物联网平台控制台创建产品和设备，获取设备证书信息（ProdcutKey、DeviceName和DeviceSecret），并定义物模型。

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，选择**设备管理 > 产品**，再单击**创建产品**，创建一个产品。

参数	说明
产品名称	自定义产品名称。
所属品类	选择**自定义品类**。
节点类型	选择**直连设备**。
连网方式	选择**Wi-Fi**。
数据格式	选择**ICA标准数据格式（Alink JSON）**。
认证方式	选择**设备密钥**。

4. 产品创建成功后，单击**前往定义物模型**。
5. 在**产品详情**页的**功能定义**页签下，选择**编辑草稿 > 添加自定义功能**，添加以下属性。

   本示例中，温湿度采集器会上报温度和湿度，因此需为该产品定义对应的两个属性。

功能类型	功能名称	标识符	数据类型	取值范围	步长	读写类型
属性	温度	temperature	int32	-10~50	1	只读
属性	湿度	humidity	int32	1~100	1	只读

6. 物模型编辑完成后，单击**发布上线**，将物模型发布为正式版。
7. 在左侧导航栏，选择**设备**，单击**添加设备**，在刚创建的产品下添加设备。
   设备创建成功后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

开发设备端

开发设备端实现设备通过HTTPS协议连接物联网平台，并上报温湿度属性数据。

1. 配置设备身份认证。

设备请求与物联网平台建立连接时，物联网平台会进行设备身份认证。认证通过后，下发设备token。设备token将在设备上报数据时使用。

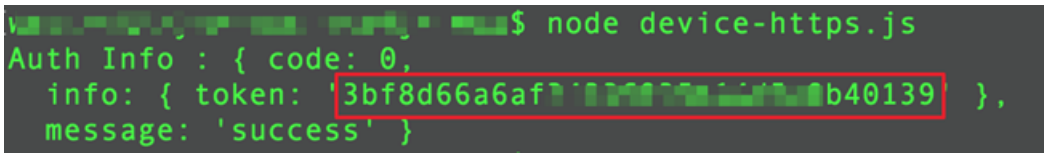
设备身份认证请求参数如下表。

参数	说明
method	请求方法。必须指定为*POST*。
uri	指定为*https://iot-as-http.cn-shanghai.aliyuncs.com/auth*。
productKey	设备所属产品的Key。可从物联网平台的控制台**设备详情**页获取。
deviceName	设备名称。从物联网平台的控制台**设备详情**页获取。
clientId	客户端ID。长度为64字符内，可使用设备的MAC地址或SN码。本示例中，使用函数random()生成随机数。
timestamp	时间戳。本示例中使用函数now()获取当前时间戳。
signmethod	算法类型，支持hmacmd5和hmacsha1。
sign	签名，即计算出的password。password计算方法示例如下。 `password = signHmacSha1(params, deviceConfig.deviceSecret)`

设备身份认证示例代码如下。

```
var rp = require('request-promise');
const crypto = require('crypto');
const deviceConfig = {
    productKey: "<yourProductKey>",
    deviceName: "<yourDeviceName>",
    deviceSecret: "<yourDeviceSecret>"
}
//获取身份token。
rp(getAuthOptions(deviceConfig))
    .then(function(parsedBody) {
        console.log('Auth Info :',parsedBody)
    })
    .catch(function(err) {
        console.log('Auth err :'+JSON.stringify(err))
    });
//生成Auth认证的参数。
function getAuthOptions(deviceConfig) {
    const params = {
        productKey: deviceConfig.productKey,
        deviceName: deviceConfig.deviceName,
        timestamp: Date.now(),
        clientId: Math.random().toString(36).substr(2),
    }
    //生成clientId、username和password。
    var password = signHmacSha1(params, deviceConfig.deviceSecret);
    var options = {
        method: 'POST',
        uri: 'https://iot-as-http.cn-shanghai.aliyuncs.com/auth',
        body: {
            "version": "default",
            "clientId": params.clientId,
            "signmethod": "hmacsha1",
            "sign": password,
            "productKey": deviceConfig.productKey,
            "deviceName": deviceConfig.deviceName,
            "timestamp": params.timestamp
        },
        json: true
    };
    return options;
}
//HmacSha1 sign
function signHmacSha1(params, deviceSecret) {
    let keys = Object.keys(params).sort();
    // 按字典序排序。
    keys = keys.sort();
    const list = [];
    keys.map((key) => {
        list.push(`${key}${params[key]}`);
    });
    const contentStr = list.join('');
    return crypto.createHmac('sha1', deviceSecret).update(contentStr).digest('hex');
}
```

配置完成后，可运行以上程序代码，进行设备认证测试。认证成功，则获得token。



> **注意** 设备认证返回的token会在一定周期后失效（目前token有效期是7天），请务必考虑token失效逻辑的处理。

2. 配置设备上报数据。

   认证通过，设备获得token后，便可使用token作为上报数据的password。

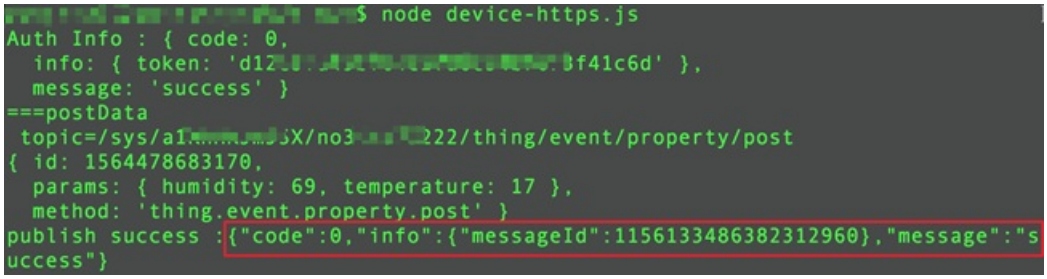
   设备上报数据的请求参数如下表。

参数	说明
method	请求方法。必须指定为*POST*。
uri	endpoint地址和Topic组成uri：`https://iot-as-http.cn-shanghai.aliyuncs.com/topic + topic`。 后一个topic需指定为设备上报属性的Topic： `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/property/post`
body	设备上报的消息内容。
password	指定为设备认证返回的token。
Content-Type	设备上报的数据的编码格式。目前仅支持：application/octet-stream。

   设备上报数据示例代码如下。

```
const topic = `/sys/${deviceConfig.productKey}/${deviceConfig.deviceName}/thing/event/property/post`;
//上报数据。
pubData(topic, token, getPostData())
function pubData(topic, token, data) {
    const options = {
        method: 'POST',
        uri: 'https://iot-as-http.cn-shanghai.aliyuncs.com/topic' + topic,
        body: data,
        headers: {
            password: token,
            'Content-Type': 'application/octet-stream'
        }
    }
    rp(options)
        .then(function(parsedBody) {
            console.log('publish success :' + parsedBody)
        })
        .catch(function(err) {
            console.log('publish err ' + JSON.stringify(err))
        });
}
//模拟物模型数据。
function getPostData() {
    var payloadJson = {
        id: Date.now(),
        params: {
            humidity: Math.floor((Math.random() * 20) + 60),
            temperature: Math.floor((Math.random() * 20) + 10)
        },
        method: "thing.event.property.post"
    }
    console.log("===postData\n topic=" + topic)
    console.log(payloadJson)
    return JSON.stringify(payloadJson);
}
```

配置完成后，可运行以上代码程序，进行设备上报数据测试。运行程序后，可在本地日志中查看运行结果。



在物联网平台控制台上，在该设备的**设备详情**页**运行状态**页签下，可查看设备上报的温湿度属性数据。说明设备端已通过HTTPS协议成功接入物联网平台，并上报了数据。

使用HTTPS连接通信的更多说明，请参见HTTPS连接通信。

1.13. Linux设备接入物联网平台

阿里云提供的设备端C语言SDK可以直接运行于Linux系统，并通过MQTT协议接入物联网平台。本文以在Ubuntu x86_64系统上编译设备端C语言SDK为例，介绍设备上云的配置和开发过程。

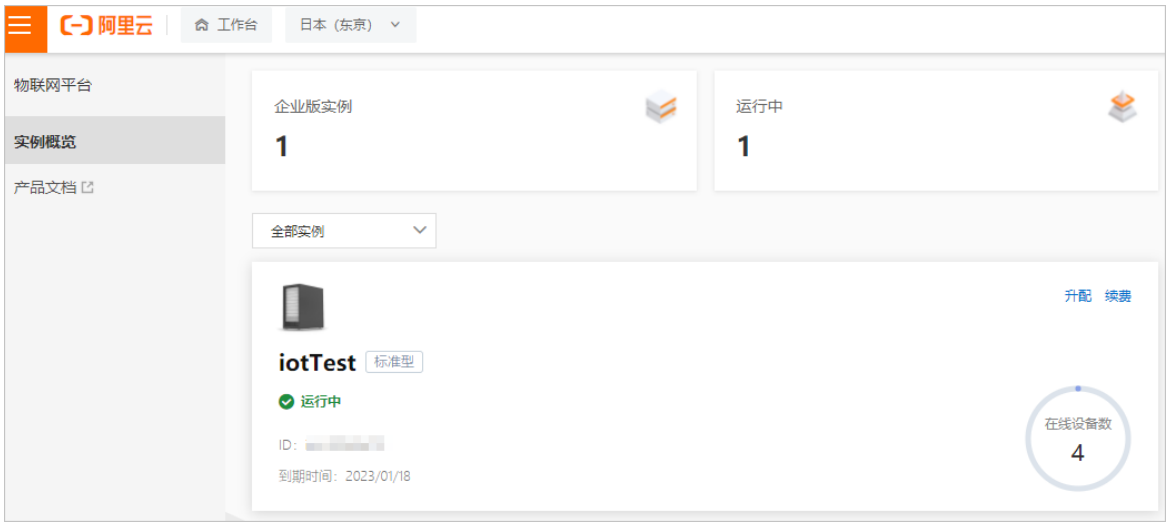
背景信息

有关设备端C语言SDK详细信息，请参见概述。

创建产品和设备

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，选择**设备管理 > 产品**，再单击**创建产品**，创建一个产品。

参数	说明
产品名称	自定义产品名称。
节点类型	选择**直连设备**。
连网方式	选择**Wi-Fi**。
数据格式	选择**ICA标准数据格式（Alink JSON）**。
认证方式	选择**设备密钥**。

4. 在左侧导航栏，选择**设备**，再单击**添加设备**，在刚创建的产品下添加设备。
   设备创建成功后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

定义产品物模型

物联网平台提供的设备端C SDK Demo包中，包含一个完整的物模型JSON文件。本示例中，导入该物模型文件，生成产品的物模型。

1. 编辑物模型文件。
   i. 下载C SDK Demo。
   ii. 解压Demo包后，打开*src/dev_model/examples*目录下的*model_for_examples.json*文件。
   iii. 将物模型JSON文件中的productKey的值替换为您在物联网平台上创建的产品ProductKey，然后保存文件。

```
{
    "schema":"https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
    "profile":{
        "productKey":"a1[redacted]"
    },
    "services":[
        {
            "outputData":[

            ],
            "identifier":"set",
            "inputData":[
                {
                    "identifier":"PowerSwitch",
                    "dataType":{
                        "specs":{
                            "0":"关闭",
                            "1":"开启"
                        },
                        "type":"bool"
                    },
                    "name":"电源开关"
                }
            ],
            "method":"thing.service.property.set",
            "name":"set",
            "required":true,
            "callType":"async",
            "desc":"属性设置"
        },
```

2. 在物联网平台控制台的**产品**页，找到之前创建的产品，单击对应的**查看**。
3. 在**产品详情**页**功能定义**页签下，单击**编辑草稿 > 快速导入**。
4. 在弹出的对话框中，选择**导入物模型**，上传上一步编辑好的物模型JSON文件，单击**确定**。

导入成功后，该文件定义的所有功能将显示在自定义功能列表中。

5. 单击**发布上线**，将物模型发布为正式版。

配置SDK

将C SDK Demo文件导入您的开发环境中，并修改配置文件中的信息为您的设备信息。

1. 在SDK Demo中*wrappers/os/ubuntu*目录下*HAL_OS_linux.c*文件中，修改设备证书信息为您的设备证书信息。

```
#include "infra_config.h"
#include "infra_compat.h"
#include "infra_defs.h"
#include "wrappers_defs.h"

#define PLATFORM_WAIT_INFINITE (~0)

#ifdef DYNAMIC_REGISTER
    char _product_key[IOTX_PRODUCT_KEY_LEN + 1]       = "
    char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "
    char _device_name[IOTX_DEVICE_NAME_LEN + 1]       = "
    char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]   = "";
#else
    #ifdef DEVICE_MODEL_ENABLED
        char _product_key[IOTX_PRODUCT_KEY_LEN + 1]       = "a1Oy
        char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "I5Nv
        char _device_name[IOTX_DEVICE_NAME_LEN + 1]       = "Linu
        char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]   = "6a03";
    #else
        char _product_key[IOTX_PRODUCT_KEY_LEN + 1]       =
        char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] =
        char _device_name[IOTX_DEVICE_NAME_LEN + 1]       =
        char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]   =
    #endif
#endif

char _firmware_version[IOTX_FIRMWARE_VER_LEN] = "app-1.0.0-20180101.1000";
```

2. 编译SDK。在SDK根目录中，执行**make reconfig**，并选择*3*，然后**make**。

```
root@                                   :~/c-sdk-v3.0.1# make reconfig
SELECT A CONFIGURATION:

1) config.alios.esp8266
2) config.alios.mk3080
3) config.ubuntu.x86
#? 3
```

3. 测试运行SDK。

   在SDK根目录中，执行./output/release/bin/linkkit-example-solo。执行结果如下图。

```
root@                                   :~/c-sdk-v3.0.1# ./output/release/bin/linkkit-example-solo
establish tcp connection with server(host='a1                .iot-as-mqtt.cn-shanghai.aliyuncs.com', port=[1883])
success to establish tcp, fd=3

> {
>     "id": "1",
>     "version": "1.0",
>     "params": [
>         {
>             "attrKey": "SYS_LP_SDK_VERSION",
>             "attrValue": "3.0.1",
>             "domain": "SYSTEM"
>         },
>         {
>             "attrKey": "SYS_SDK_LANGUAGE",
>             "attrValue": "C",
>             "domain": "SYSTEM"
>         }
>     ],
>     "method": "thing.deviceinfo.update"
> }
```

   SDK运行成功后，可在物联网平台控制台，设备对应的**设备详情**页，查看设备状态和设备上报的物模型数据。

2.消息通信

2.1. 使用自定义Topic进行通信

您可以在物联网平台上自定义Topic类，设备将消息发送到自定义Topic中，服务端通过AMQP SDK获取设备上报消息；服务端通过调用物联网平台接口Pub向设备发布指令。自定义Topic通信不使用物模型，消息的数据结构由您自定义。

背景信息

本示例中，电子温度计定期与服务器进行数据的交互，传递温度和指令等信息。温度计向服务器上行发送当前的温度；服务器向温度计下行发送精度设置指令。

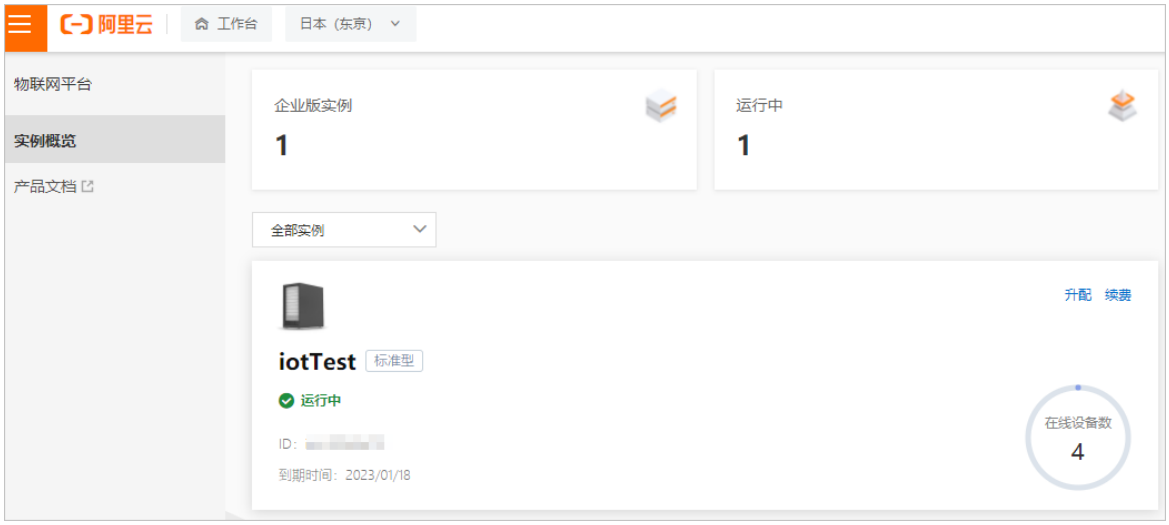
准备开发环境

本示例中，设备端和云端均使用Java语言的SDK，需先准备Java开发环境。您可从Java 官方网站下载，并安装Java最新开发环境。

创建产品和设备

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，单击**设备管理 > 产品**。

4. 单击**创建产品**，创建温度计产品，获取**productKey**，例如 `a1uzcH0****` 。

   详细操作指导，请参见创建产品。
5. 产品创建成功后，单击该产品对应的**查看**。
6. 在**产品详情**页的**Topic类列表**页签下，单击**自定义Topic**，增加自定义Topic类。

   详细操作指导，请参见自定义Topic。

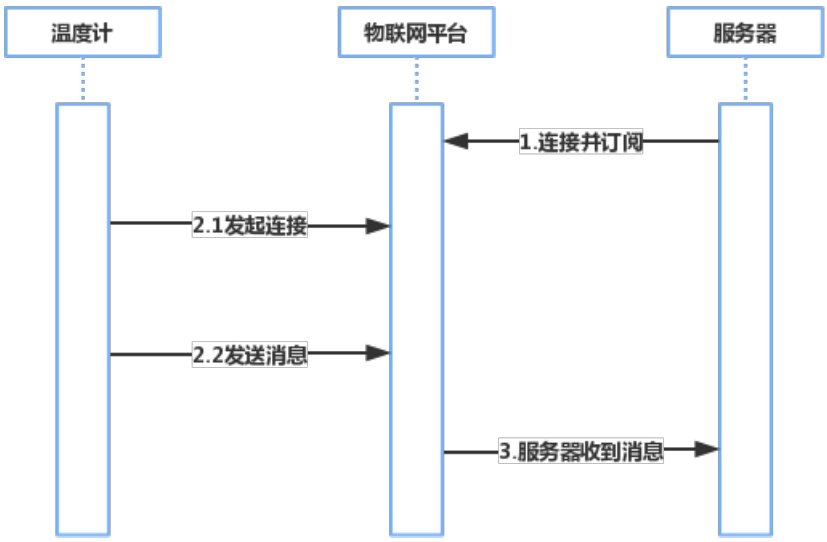
   本示例中，定义了以下两个Topic类：
   - 设备发布消息Topic：/a1uzcH0****/${deviceName}/user/devmsg，权限为发布。
   - 设备订阅消息Topic：/a1uzcH0****/${deviceName}/user/cloudmsg，权限为订阅。
7. 在**服务端订阅**页签下，单击**创建订阅**，设置AMQP服务端订阅，订阅**设备上报消息**到**默认消费组**。

   设备上报消息包含自定义Topic消息和物模型消息。详细操作和说明，请参见配置AMQP服务端订阅。
8. 在左侧导航栏，单击**设备**，然后在刚创建的温度计产品下，添加设备**device1**，获取设备证书**ProductKey**、**DeviceName**和**DeviceSecret**。

   详细操作指导，请参见单个创建设备。

设备发送消息给服务器

流程图：



在整个流程中：

- 服务器通过AMQP客户端接收消息，需配置AMQP客户端接入物联网平台，监听设备消息。具体操作，请参见Java SDK接入示例。

  > 注意 AMQP订阅消息的消费组，必须与设备在相同的物联网平台实例下。

- 配置设备端SDK接入物联网平台，并发送消息。

- 配置设备认证信息。

```
final String productKey = "a1uzcH0****";
final String deviceName = "device1";
final String deviceSecret = "uwMTmVAMnxB***";
final String region = "cn-shanghai";
```

- 设置初始化连接参数，包括MQTT连接配置、设备信息和初始物模型属性。下文代码示例是将设备接入物联网平台公共实例。

```
LinkKitInitParams params = new LinkKitInitParams();
//LinkKit底层是MQTT协议，设置MQTT的配置。
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = productKey;
config.deviceName = deviceName;
config.deviceSecret = deviceSecret;
config.channelHost = productKey + ".iot-as-mqtt." + region + ".aliyuncs.com:1883";
//设备的信息。
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
//报备的设备初始状态。
Map<String, ValueWrapper> propertyValues = new HashMap<String, ValueWrapper>();
params.mqttClientConfig = config;
params.deviceInfo = deviceInfo;
params.propertyValues = propertyValues;
```

- 初始化连接。

```
//连接并设置连接成功以后的回调函数。
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
     @Override
     public void onError(AError aError) {
         System.out.println("Init error:" + aError);
     }
     //初始化成功以后的回调。
     @Override
     public void onInitDone(InitResult initResult) {
         System.out.println("Init done:" + initResult);
     }
 });
```

- 设备发送消息。

  设备端连接物联网平台后，向以上定义的Topic发送消息。需将onInitDone函数内容替换为以下内容：

```
@Override
 public void onInitDone(InitResult initResult) {
     //设置Pub消息的Topic和内容。
     MqttPublishRequest request = new MqttPublishRequest();
     request.topic = "/" + productKey + "/" + deviceName + "/user/devmsg";
     request.qos = 0;
     request.payloadObj = "{\"temperature\":35.0, \"time\":\"sometime\"}";
     //发送消息并设置成功以后的回调。
     LinkKit.getInstance().publish(request, new IConnectSendListener() {
         @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());
         }
     });
 }
```

  服务器收到消息如下：

```
Message
{payload={"temperature":35.0, "time":"sometime"},
topic='/a1uzcH0****/device1/user/devmsg',
messageId='1131755639450642944',
qos=0,
generateTime=1558666546105}
```

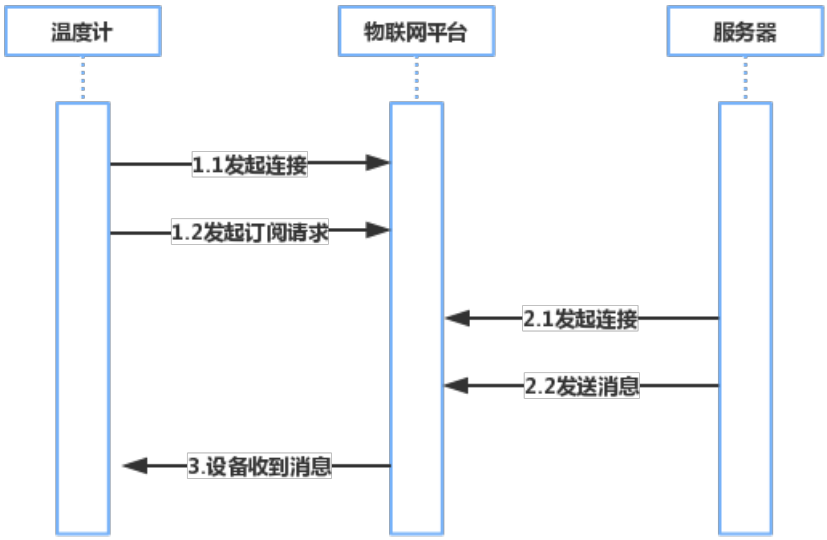
实际业务场景中，您需修改以下参数值。

参数	示例	说明
productKey	a1uzcH0****	设备认证信息，请参见创建产品和设备中的步骤8。
deviceName	device1	
deviceSecret	uwMTmVAMnxB***	
region	cn-shanghai	您的物联网平台设备所在地域代码。地域代码表达方法，请参见地域和可用区。
channelHost	iot-o***.mqtt.iothub.aliyuncs.com:1883	设备接入地址。详细说明，请参见查看实例终端节点。 注意 接入域名若没有携带端口号1883，需添加该端口号。

参数	示例	说明
request.topic	"/" + productKey + "/" + deviceName + "/user/devmsg"	具有发布权限的自定义Topic。
request.payloadObj	"{\"temperature\":35.0,\"time\":\"sometime\"}"	自定义的消息内容。

服务器发送消息给设备

流程图：



- 配置设备端SDK订阅Topic。

  配置设备认证信息、设置初始化连接参数、初始化连接，请参见设备发送消息给服务器中的相应示例代码。

  设备要接收服务器发送的消息，还需订阅消息Topic。

  配置设备端订阅Topic示例如下：

```
//初始化成功以后的回调。
@Override
public void onInitDone(InitResult initResult) {
    //设置订阅的Topic。
    MqttSubscribeRequest request = new MqttSubscribeRequest();
    request.topic = "/" + productKey + "/" + deviceName + "/user/cloudmsg";
    request.isSubscribe = true;
    //发出订阅请求并设置订阅成功或者失败的回调函数。
    LinkKit.getInstance().subscribe(request, new IConnectSubscribeListener() {
        @Override
        public void onSuccess() {
            System.out.println("");
        }
        @Override
        public void onFailure(AError aError) {
        }
    });
    //设置订阅的下行消息到来时的回调函数。
    IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
        //此处定义收到下行消息以后的回调函数。
        @Override
        public void onNotify(String connectId, String topic, AMessage aMessage) {
            System.out.println(
                "received message from " + topic + ":" + new String((byte[])aMessage.getD
ata()));
        }
        @Override
        public boolean shouldHandle(String s, String s1) {
            return false;
        }
        @Override
        public void onConnectStateChange(String s, ConnectState connectState) {
        }
    };
    LinkKit.getInstance().registerOnNotifyListener(notifyListener);
}
```

其中，`request.topic`值需修改为具有订阅权限的自定义Topic。

- 配置云端SDK调用物联网平台接口Pub发布消息。参数说明，请参见Pub，使用说明，请参见Java SDK使用说明。下文代码示例是向物联网平台公共实例下设备发布消息。
  - 设置身份认证信息。

    ```
    String regionId = "***";
    String accessKey = "***";
    String accessSecret = "***";
    final String productKey = "***";
    ```

  - 设置连接参数。

    ```
    //设置client的参数。
    DefaultProfile profile = DefaultProfile.getProfile(regionId, accessKey, accessSecret);
    IAcsClient client = new DefaultAcsClient(profile);
    ```

- 设置消息发布参数。

```
PubRequest request = new PubRequest();
request.setQos(0);
//设置发布消息的Topic。
request.setTopicFullName("/" + productKey + "/" + deviceName + "/user/cloudmsg");
request.setProductKey(productKey);
//设置消息的内容，一定要用Base64编码，否则乱码。
request.setMessageContent(Base64.encode("{\"accuracy\":0.001,\"time\":now}"));
```

- 发送消息。

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

设备端接收到的消息如下：

```
msg = [{"accuracy":0.001,"time":now}]
```

附录：代码示例

> **说明** 实际业务场景中，请按照上文描述，修改代码中相关参数的值。

下载Pub/Sub demo，包含本示例的云端SDK和设备端SDK配置代码Demo。

AMQP客户端接入物联网平台示例，请参见：

- Java SDK接入示例
- .NET SDK接入示例
- Node.js SDK接入示例
- Python 2.7 SDK接入示例
- Python3 SDK接入示例
- PHP SDK接入示例
- Go SDK接入示例

2.2. 远程控制树莓派服务器

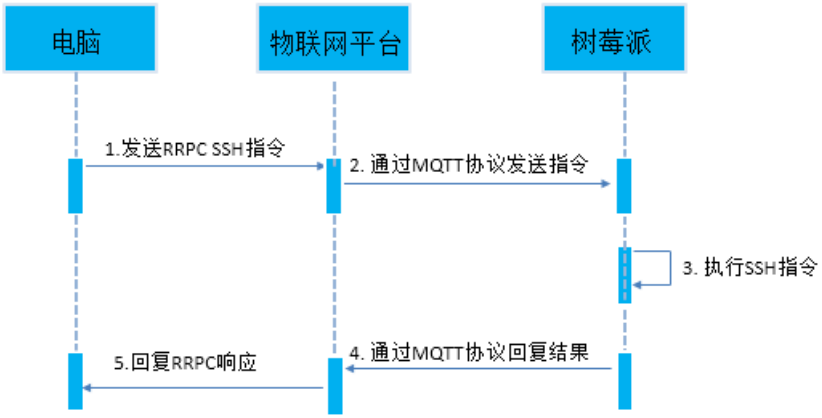
使用阿里云物联网平台可实现伪内网穿透，对无公网IP的树莓派服务器进行远程控制。本文以实现基于树莓派服务器远程控制为例，介绍伪内网穿透的实现流程，并提供开发代码示例。

背景信息

假如您在公司或家里使用树莓派搭建一个服务器，用于执行一些简单的任务，如启动某个脚本，开始下载文件等。但是，如果树莓派没有公网IP，您不在公司或家里的情况下，您就无法控制该服务器。如果使用其他内网穿透工具，也会经常出现断线的情况。为解决以上问题，您可以使用阿里云物联网平台的RRPC（同步远程过程调用）功能结合JSch库来实现对树莓派服务器的远程控制。

> **说明** 本文以公共实例下产品和设备为例，介绍远程控制树莓派服务器的开发方法。

实现远程控制的流程



通过物联网平台远程控制树莓派服务器的流程：

1. 在电脑上调用物联网平台RRPC接口发送SSH指令。
2. 物联网平台接收到指令后，通过MQTT协议将SSH指令下发给树莓派服务器。
3. 服务器执行SSH指令。
4. 服务器将SSH指令执行结果封装成RRPC响应，通过MQTT协议上报到物联网平台。
5. 物联网平台将RRPC响应回复给电脑。

> **说明** RRPC调用的超时限制为5秒。服务器5秒内未收到设备回复，会返回超时错误。如果您发送的指令操作耗时较长，可忽略该超时错误信息。

下载SDK和Demo

实现物联网平台远程控制树莓派，您需先进行服务端SDK和设备端SDK开发。

- 在电脑上安装物联网平台服务端SDK。您可以使用服务端Java SDK Demo来进行服务端开发。
- 在树莓派上安装物联网平台设备端SDK。您可以使用设备端Java SDK Demo来进行设备端开发。

以下章节中，将介绍服务端SDK和设备端SDK的开发示例。

> **说明** 本文示例中提供的代码仅支持一些简单的Linux命令，如uname、touch、mv等，不支持文件编辑等复杂的指令，需要您自行实现。

设备端SDK开发

下载、安装设备端SDK和下载SDK Demo后，您需添加项目依赖和增加以下Java文件。

项目可以导出成JAR包在树莓派上运行。

1. 在*pom.xml*文件中，添加依赖。

```
<!-- 设备端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客户端 -->
<!-- https://mvnrepository.com/artifact/com.jcraft/jsch -->
<dependency>
    <groupId>com.jcraft</groupId>
    <artifactId>jsch</artifactId>
    <version>0.1.55</version>
</dependency>
```

2. 增加*SSHShell.java*文件，用于执行SSH指令。

```
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 passwor
d, 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();
        SSHUserInfo userInfo = new SSHUserInfo();
        try {
            Session session = jsch.getSession(username, host, port);
            session.setPassword(password);
            session.setUserInfo(userInfo);
            session.connect();
            Channel channel = session.openChannel("exec");
            ((ChannelExec) channel).setCommand(command);
            channel.setInputStream(null);
            BufferedReader input = new BufferedReader(new InputStreamReader(channel.get
InputStream()));
            channel.connect();
            String line = null;
            while ((line = input.readLine()) != null) {
                stdout.add(line);
            }
            input.close();
            if (channel.isClosed()) {
                returnCode = channel.getExitStatus();
            }
            channel.disconnect();
            session.disconnect();
        } catch (JSchException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return returnCode;
    }
    public Vector<String> getStdout() {
        return stdout;
    }
}
```

3. 增加*SSHUserInfo.java*文件，用于验证SSH账号密码。

```
public class SSHUserInfo implements UserInfo {
    @Override
    public String getPassphrase() {
        return null;
    }
    @Override
    public String getPassword() {
        return null;
    }
    @Override
    public boolean promptPassphrase(final String arg0) {
        return false;
    }
    @Override
    public boolean promptPassword(final String arg0) {
        return false;
    }
    @Override
    public boolean promptYesNo(final String arg0) {
        if (arg0.contains("The authenticity of host")) {
            return true;
        }
        return false;
    }
    @Override
    public void showMessage(final String arg0) {
    }
}
```

4. 增加*Device.java*文件，用于创建MQTT连接。

```
public class Device {
    /**
     * 建立连接
     *
     * @param productKey 产品key
     * @param deviceName 设备名称
     * @param deviceSecret 设备密钥
     * @throws InterruptedException
     */
    public static void connect(String productKey, String deviceName, String deviceSecre
t) throws InterruptedException {
        // 初始化参数
        LinkKitInitParams params = new LinkKitInitParams();
        // 设置 Mqtt 初始化参数
        IoTMqttClientConfig config = new IoTMqttClientConfig();
        config.productKey = productKey;
        config.deviceName = deviceName;
        config.deviceSecret = deviceSecret;
        params.mqttClientConfig = config;
        // 设置初始化设备证书信息，传入：
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.productKey = productKey;
        deviceInfo.deviceName = deviceName;
```

```
        deviceInfo.deviceSecret = deviceSecret;
        params.deviceInfo = deviceInfo;
        // 初始化
        LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
            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 !!");
            }
        });
        // 确保初始化成功后才执行后面的步骤，可以根据实际情况适当延长这里的延时
        Thread.sleep(2000);
    }
    /**
     * 发布消息
     *
     * @param topic 发送消息的topic
     * @param payload 发送的消息内容
     */
    public static void publish(String topic, String payload) {
        MqttPublishRequest request = new MqttPublishRequest();
        request.topic = topic;
        request.payloadObj = payload;
        request.qos = 0;
        LinkKit.getInstance().getMqttClient().publish(request, new IConnectSendListener
() {
            @Override
            public void onResponse(ARequest aRequest, AResponse aResponse) {
            }
            @Override
            public void onFailure(ARequest aRequest, AError aError) {
            }
        });
    }
    /**
     * 订阅消息
     *
     * @param topic 订阅消息的topic
     */
    public static void subscribe(String topic) {
        MqttSubscribeRequest request = new MqttSubscribeRequest();
        request.topic = topic;
        request.isSubscribe = true;
        LinkKit.getInstance().getMqttClient().subscribe(request, new IConnectSubscribeL
istener() {
            @Override
            public void onSuccess() {
            }
            @Override
            public void onFailure(AError aError) {
            }
        });
```

```
        });
    }
    /**
     * 取消订阅
     *
     * @param topic 取消订阅消息的topic
     */
    public static void unsubscribe(String topic) {
        MqttSubscribeRequest request = new MqttSubscribeRequest();
        request.topic = topic;
        request.isSubscribe = false;
        LinkKit.getInstance().getMqttClient().unsubscribe(request, new IConnectUnscribe
Listener() {
            @Override
            public void onSuccess() {
            }
            @Override
            public void onFailure(AError aError) {
            }
        });
    }
    /**
     * 断开连接
     */
    public static void disconnect() {
        // 反初始化
        LinkKit.getInstance().deinit();
    }
}
```

5. 增加*SSHDevice.java*文件。*SSHDevice.java*包含main方法，用于接收RRPC指令，调用 `SSHShell` 执行SSH指令，返回RRPC响应。*SSHDevice.java*文件中，需要填写设备证书信息（ProductKey、DeviceName和DeviceSecret）和SSH账号密码。

```
public class SSHDevice {
    // ==================需要填写的参数开始==========================
    // 产品productKey
    private static String productKey = "";
    //
    private static String deviceName = "";
    // 设备密钥deviceSecret
    private static String deviceSecret = "";
    // 消息通信的topic，无需创建和定义，直接使用即可
    private static String rrpcTopic = "/sys/" + productKey + "/" + deviceName + "/rrpc/
request/+";
    // ssh 要访问的域名或IP
    private static String host = "127.0.0.1";
    // ssh 用户名
    private static String username = "";
    // ssh 密码
    private static String password = "";
    // ssh 端口号
    private static int port = 22;
    // ==================需要填写的参数结束==========================
    public static void main(String[] args) throws InterruptedException {
```

```
        // 下行数据监听
        registerNotifyListener();
        // 建立连接
        Device.connect(productKey, deviceName, deviceSecret);
        // 订阅topic
        Device.subscribe(rrpcTopic);
    }
    public static void registerNotifyListener() {
        LinkKit.getInstance().registerOnNotifyListener(new IConnectNotifyListener() {
            @Override
            public boolean shouldHandle(String connectId, String topic) {
                // 只处理特定topic的消息
                if (topic.contains("/rrpc/request/")) {
                    return true;
                } else {
                    return false;
                }
            }
            @Override
            public void onNotify(String connectId, String topic, AMessage aMessage) {
                // 接收rrpc请求并回复rrpc响应
                try {
                    // 执行远程命令
                    String payload = new String((byte[]) aMessage.getData(), "UTF-8");
                    SSHShell sshExecutor = new SSHShell(host, username, password, port)
;
                    sshExecutor.execute(payload);
                    // 获取命令回显
                    StringBuffer sb = new StringBuffer();
                    Vector<String> stdout = sshExecutor.getStdout();
                    for (String str : stdout) {
                        sb.append(str);
                        sb.append("\n");
                    }
                    // 回复回显到服务端
                    String response = topic.replace("/request/", "/response/");
                    Device.publish(response, sb.toString());
                } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
                }
            }
            @Override
            public void onConnectStateChange(String connectId, ConnectState connectStat
e) {
            }
        });
    }
}
```

服务端SDK开发

下载、安装服务端SDK和下载SDK Demo后，您需添加项目依赖和增加以下Java文件。

1. 在*pom.xml*文件中，添加依赖。

> 注意 服务端SDK对应最新版版本，请参见Java SDK使用说明。

```
<!-- 服务端SDK -->
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-iot</artifactId>
    <version>7.38.0</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>4.5.6</version>
</dependency>
<!-- commons-codec -->
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>1.8</version>
</dependency>
```

2. 增加*OpenApiClient.java*文件，用于调用物联网平台开放接口。

```
public class OpenApiClient {
    private static DefaultAcsClient client = null;
    public static DefaultAcsClient getClient(String accessKeyID, String accessKeySecret
) {
        if (client != null) {
            return client;
        }
        try {
            IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", accessKey
ID, accessKeySecret);
            DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "Iot", "iot.cn-sha
nghai.aliyuncs.com");
            client = new DefaultAcsClient(profile);
        } catch (Exception e) {
            System.out.println("create OpenAPI Client failed !! exception:" + e.getMess
age());
        }
        return client;
    }
}
```

3. 增加*SSHCommandSender.java*文件。*SSHCommandSender.java*包含main方法，用于发送SSH指令和接收SSH指令响应。*SSHCommandSender.java*中，需要填写您的账号AccessKey信息、设备证书信息（ProductKey和DeviceName）、以及SSH指令。

```
public class SSHCommandSender {
    // ==================需要填写的参数开始==========================
    // 用户账号AccessKey
    private static String accessKeyID = "";
    // 用户账号AccesseKeySecret
    private static String accessKeySecret = "";
    // 产品Key
    private static String productKey = "";
    // 设备名称deviceName
    private static String deviceName = "";
    // ==================需要填写的参数结束==========================
    public static void main(String[] args) throws ServerException, ClientException, Uns
upportedEncodingException {
        // Linux 远程命令
        String payload = "uname -a";
        // 构建RRPC请求
        RRpcRequest request = new RRpcRequest();
        request.setProductKey(productKey);
        request.setDeviceName(deviceName);
        request.setRequestBase64Byte(Base64.encodeBase64String(payload.getBytes()));
        request.setTimeout(5000);
        // 获取服务端请求客户端
        DefaultAcsClient client = OpenApiClient.getClient(accessKeyID, accessKeySecret)
;
        // 发起RRPC请求
        RRpcResponse response = (RRpcResponse) client.getAcsResponse(request);
        // RRPC响应处理
        // response.getSuccess()仅表明RRPC请求发送成功，不代表设备接收成功和响应成功
        // 需要根据RrpcCode来判定，参考文档https://www.alibabacloud.com/help/doc-detail/69
797.htm
        if (response != null && "SUCCESS".equals(response.getRrpcCode())) {
            // 回显
            System.out.println(new String(Base64.decodeBase64(response.getPayloadBase64
Byte()), "UTF-8"));
        } else {
            // 回显失败，打印rrpc code
            System.out.println(response.getRrpcCode());
        }
    }
}
```

2.3. 物模型通信

设备与云端基于Alink协议进行物模型数据通信，包括设备上报属性或事件消息到云端，从云端下发设置属性或调用服务消息到设备。本实践案例提供Java Demo，介绍物模型数据通信代码配置。

前提条件

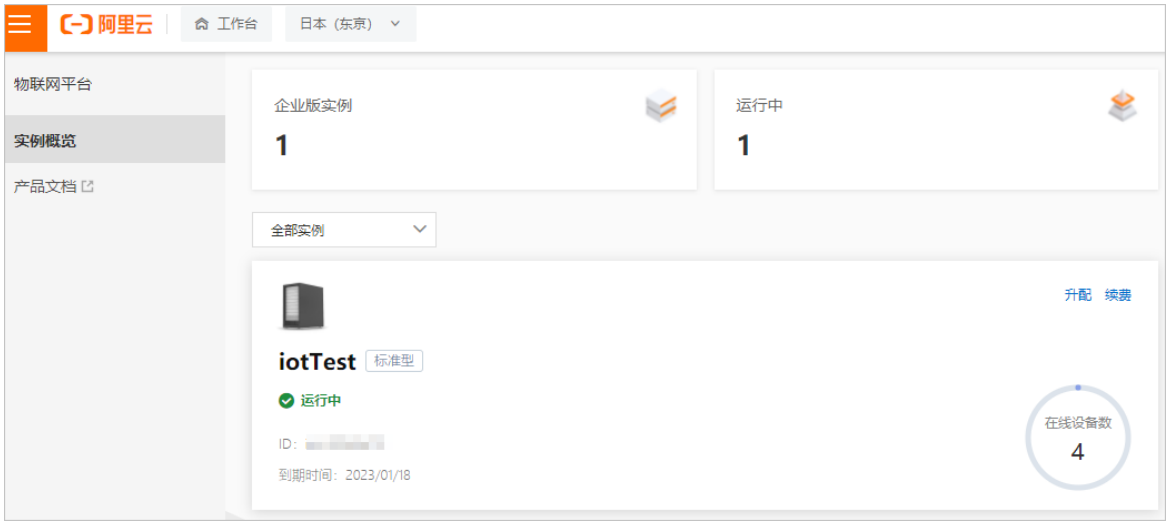
- 已开通物联网平台服务。
- 已安装Java开发环境。

创建产品和设备

首先，需创建产品和设备，为产品定义功能（即物模型）。

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，单击**设备管理 > 产品**。
4. 单击**创建产品**，自定义产品名称，选择**自定义品类**，其他参数使用默认值，然后单击**确认**，完成创建产品。

   详细操作指导，请参见创建产品。
5. 在**产品详情**的**功能定义**页签下，定义物模型。

   本示例中在物模型的**默认模块**中，添加以下属性、服务和事件。

   本文提供了示例的物模型TSL，您可批量导入，请参见批量添加物模型。

默认模块

功能类型	功能名称（全部）	标识符	数据类型	数据定义
事件	离线告警 自定义	Offline_alarm	-	事件类型：告警
服务	modify 自定义	ModifyVehicleInfo	-	调用方式：异步调用
属性	颜色 自定义	hue	int32 (整数型)	取值范围：0 ~ 254
属性	开关 自定义	MicSwitch	bool (布尔型)	布尔值： 0 - 关 1 - 开

6. 在左侧导航栏，单击**设备**，创建设备。

   本示例代码中涉及批量设置设备属性和批量调用设备服务，所以需至少创建两个设备。详细操作指导，请参见批量创建设备。

下载、安装Demo SDK

本示例提供的SDK Demo中包含了服务端SDK Demo和设备端SDK Demo。

1. 单击下载iotx-api-demo，并解压缩。

2. 打开Java开发工具，导入解压缩后的*iotx-api-demo*文件夹。
3. 在*pom.xml*文件中，添加以下Maven依赖，导入阿里云云端SDK和设备端SDK。

```
<!-- https://mvnrepository.com/artifact/com.aliyun/aliyun-java-sdk-iot -->
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-iot</artifactId>
    <version>7.33.0</version>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.5.1</version>
</dependency>
<dependency>
  <groupId>com.aliyun.alink.linksdk</groupId>
  <artifactId>iot-linkkit-java</artifactId>
  <version>1.2.0</version>
  <scope>compile</scope>
</dependency>
```

4. 在*java/src/main/resources/*目录下的*config*文件中，填入初始化信息。

```
user.accessKeyID = <your accessKey ID>
user.accessKeySecret = <your accessKey Secret>
iot.regionId = <regionId>
iot.productCode = Iot
iot.domain = iot.<regionId>.aliyuncs.com
iot.version = 2018-01-20
```

参数	说明
accessKeyID	您的阿里云账号的AccessKey ID。 将光标定位到您的账号头像上，选择**AccessKey管理**，进入**安全信息管理**页，可创建或查看您的AccessKey。
accessKeySecret	您的阿里云账号的AccessKey Secret。查看方法同上AccessKey ID。
regionId	您的物联网设备所属地域ID。地域ID的表达方法，请参见地域和可用区。

设备端SDK上报属性和事件

配置设备端SDK连接物联网平台，上报属性和事件消息。

Demo中，*java/src/main/com.aliyun.iot.api.common.deviceApi*目录下的*ThingTemplate*文件是设备端上报属性和事件的Demo。

- 设置连接信息。

  将代码中productKey、deviceName、deviceSecret和url替换为您的设备证书信息和MQTT接入域名。接入域名获取方法，请参见查看实例终端节点，接入域名必须携带端口1883。

```
public static void main(String[] args) {
        /**
         * 设备证书信息。
         */
        String productKey = "your productKey";
        String deviceName = "your deviceName";
        String deviceSecret = "your deviceSecret";
        /* TODO: 替换为您物联网平台实例的接入地址 */
        String  url = "iot-6d***ql.mqtt.iothub.aliyuncs.com:1883";
        /**
         * mqtt连接信息。
         */
        ThingTemplate manager = new ThingTemplate();
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.productKey = productKey;
        deviceInfo.deviceName = deviceName;
        deviceInfo.deviceSecret = deviceSecret;
        /**
         * 服务器端的Java HTTP客户端使用TSLv1.2。
         */
        System.setProperty("https.protocols", "TLSv2");
        manager.init(deviceInfo, url);
    }
```

- 初始化连接。

```
public void init(final DeviceInfo deviceInfo, String url) {
        LinkKitInitParams params = new LinkKitInitParams();
        /**
         * 设置mqtt初始化参数。
         */
        IoTMqttClientConfig config = new IoTMqttClientConfig();
        config.productKey = deviceInfo.productKey;
        config.deviceName = deviceInfo.deviceName;
        config.deviceSecret = deviceInfo.deviceSecret;
        config.channelHost = url;
        /**
         * 是否接受离线消息。
         * 对应mqtt的cleanSession字段。
         */
        config.receiveOfflineMsg = false;
        params.mqttClientConfig = config;
        ALog.setLevel(LEVEL_DEBUG);
        ALog.i(TAG, "mqtt connetcion info=" + params);
        /**
         * 设置初始化，传入设备证书信息。
         */
        params.deviceInfo = deviceInfo;
        /**建立连接。**/
        LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
            public void onError(AError aError) {
                ALog.e(TAG, "Init Error error=" + aError);
            }
            public void onInitDone(InitResult initResult) {
                ALog.i(TAG, "onInitDone result=" + initResult);
                List<Property> properties =   LinkKit.getInstance().getDeviceThing().getP
roperties();
                ALog.i(TAG, "设备属性列表" + JSON.toJSONString(properties));
                List<Event> getEvents  =   LinkKit.getInstance().getDeviceThing().getEven
ts();
                ALog.i(TAG, "设备事件列表" + JSON.toJSONString(getEvents));
                /*属性上报。TODO：需确保所报的属性，比如MicSwitch，是产品的物模型的一部分．否则
会返回错误 */
                handlePropertySet("MicSwitch", new ValueWrapper.IntValueWrapper(1));
                /* 事件上报．TODO：需确保所报的事件，比如Offline_alarm，是产品的物模型的一部分．
否则会返回错误 */
                Map<String,ValueWrapper> values = new HashMap<>();
                values.put("eventValue",new ValueWrapper.IntValueWrapper(0));
                OutputParams outputParams = new OutputParams(values);
                handleEventSet("Offline_alarm",outputParams);
            }
        });
    }
```

> **说明** 代码中的属性和事件标识符需与物模型中定义的标识符一致。

- 设置设备端上报属性。

```
/**
     * Alink JSON方式设备端上报属性。
     * @param identifier：属性标识符。
     * @param value：上报属性值。
     * @return
     */
    private void handlePropertySet(String identifier, ValueWrapper value ) {
        ALog.i(TAG, "上报属性identity=" + identifier);
        Map<String, ValueWrapper> reportData = new HashMap<>();
        reportData.put(identifier, value);
        LinkKit.getInstance().getDeviceThing().thingPropertyPost(reportData, new IPublish
ResourceListener() {
            public void onSuccess(String s, Object o) {
                // 属性上报成功。
                ALog.i(TAG, "上报成功 onSuccess() called with: s = [" + s + "], o = [" + o
+ "]");
            }
            public void onError(String s, AError aError) {
                // 属性上报失败。
                ALog.i(TAG, "上报失败onError() called with: s = [" + s + "], aError = [" +
JSON.toJSONString(aError) + "]");
            }
        });
    }
```

- 设置设备端上报事件。

```
/**
     * Alink JSON方式设备端上报事件。
     * @param identifyID：事件标识符。
     * @param params：事件上报参数。
     * @return
     */
    private void handleEventSet(String identifyID, OutputParams params ) {
        ALog.i(TAG, "上报事件 identifyID=" + identifyID + "  params=" + JSON.toJSONString(
params));
        LinkKit.getInstance().getDeviceThing().thingEventPost( identifyID,  params, new I
PublishResourceListener() {
            public void onSuccess(String s, Object o) {
                // 事件上报成功。
                ALog.i(TAG, "上报成功 onSuccess() called with: s = [" + s + "], o = [" + o
+ "]");
            }
            public void onError(String s, AError aError) {
                // 事件上报失败。
                ALog.i(TAG, "上报失败onError() called with: s = [" + s + "], aError = [" +
JSON.toJSONString(aError) + "]");
            }
        });
    }
```

云端SDK下发设置属性和调用服务指令

- 初始化SDK客户端。

Demo中，*java/src/main/com.aliyun.iot.client*目录下*IotClient*文件是SDK客户端初始化Demo。

```
public class IotClient {
  private static String accessKeyID;
  private static String accessKeySecret;
  private static String regionId;
  private static String domain;
  private static String version;
  public static DefaultAcsClient getClient() {
    DefaultAcsClient client = null;
    Properties prop = new Properties();
    try {
      prop.load(Object.class.getResourceAsStream("/config.properties"));
      accessKeyID = prop.getProperty("user.accessKeyID");
      accessKeySecret = prop.getProperty("user.accessKeySecret");
      regionId = prop.getProperty("iot.regionId");
            domain = prop.getProperty("iot.domain");
            version = prop.getProperty("iot.version");
      IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyID, accessKey
Secret);
      DefaultProfile.addEndpoint(regionId, regionId, prop.getProperty("iot.productCode"),
          prop.getProperty("iot.domain"));
      // 初始化client。
      client = new DefaultAcsClient(profile);
    } catch (Exception e) {
      LogUtil.print("初始化client失败! exception:" + e.getMessage());
    }
    return client;
  }
    public static String getRegionId() {
        return regionId;
    }
    public static void setRegionId(String regionId) {
        IotClient.regionId = regionId;
    }
    public static String getDomain() {
        return domain;
    }
    public static void setDomain(String domain) {
        IotClient.domain = domain;
    }
    public static String getVersion() {
        return version;
    }
    public static void setVersion(String version) {
        IotClient.version = version;
    }
}
```

- 初始化封装CommonRequest公共类。

  Demo中，*java/src/main/com.aliyun.iot.api.common.openApi*目录下的*AbstractManager*文件是封装云端API的CommonRequest公共类的Demo。

```
public class AbstractManager {
    private static DefaultAcsClient client;
    static {
        client = IotClient.getClient();
    }
    /**
     *  接口请求地址。action：接口名称。
     *  domain：线上地址。
     *  version：接口版本。
     */
    public static CommonRequest executeTests(String action) {
        CommonRequest request = new CommonRequest();
        request.setDomain(IotClient.getDomain());
        request.setMethod(MethodType.POST);
        request.setVersion(IotClient.getVersion());
        request.setAction(action);
        return request;
    }
```

- 配置云端SDK调用物联网平台云端API，下发设置属性和调用服务的指令。

  *java/src/main/com.aliyun.iot.api.common.openApi*目录下的*ThingManagerForPopSDk*是云端SDK调用API设置设备属性和调用设备服务的Demo文件。

  - 调用SetDeviceProperty设置设备属性值。

```
public static void SetDeviceProperty(String InstanceId, String IotId, String ProductKey
, String DeviceName , String Items) {
        SetDevicePropertyResponse response =null;
        SetDevicePropertyRequest request=new SetDevicePropertyRequest();
        request.setDeviceName(DeviceName);
        request.setIotId(IotId);
        request.setItems(Items);
        request.setProductKey(ProductKey);
        request.setIotInstanceId(InstanceId);
        try {
            response = client.getAcsResponse(request);
            if (response.getSuccess() != null && response.getSuccess()) {
                LogUtil.print("设置设备属性成功");
                LogUtil.print(JSON.toJSONString(response));
            } else {
                LogUtil.print("设置设备属性失败");
                LogUtil.error(JSON.toJSONString(response));
            }
        } catch (ClientException e) {
            e.printStackTrace();
            LogUtil.error("设置设备属性失败！" + JSON.toJSONString(response));
        }
    }
```

- 调用SetDevicesProperty批量设置设备属性值。

```
/**
     * 批量设置设备属性。
     *
     * @param ProductKey：要设置属性的设备所隶属的产品Key。
     * @param DeviceNames：要设置属性的设备名称列表。
     * @param Items：要设置的属性信息，组成为key:value，数据格式为JSON String，必须传入。
     *
     * @Des：描述。
     */
    public static void SetDevicesProperty(String InstanceId, String ProductKey, List<St
ring> DeviceNames, String Items) {
        SetDevicesPropertyResponse response = new SetDevicesPropertyResponse();
        SetDevicesPropertyRequest request = new SetDevicesPropertyRequest();
        request.setDeviceNames(DeviceNames);
        request.setItems(Items);
        request.setProductKey(ProductKey);
        request.setIotInstanceId(InstanceId);
        try {
            response = client.getAcsResponse(request);
            if (response.getSuccess() != null && response.getSuccess()) {
                LogUtil.print("批量设置设备属性成功");
                LogUtil.print(JSON.toJSONString(response));
            } else {
                LogUtil.print("批量设置设备属性失败");
                LogUtil.error(JSON.toJSONString(response));
            }
        } catch (ClientException e) {
            e.printStackTrace();
            LogUtil.error("批量设置设备属性失败！" + JSON.toJSONString(response));
        }
    }
```

- 调用InvokeThingService调用设备服务。

```
     /**
     * @param Identifier：服务的Identifier，必须传入。
     * @param Args：要启用服务的入参信息，必须传入。
     */
    public static InvokeThingServiceResponse.Data InvokeThingService(String InstanceId,
String IotId, String ProductKey, String DeviceName,
                                                                     String Identifier,
String Args) {
        InvokeThingServiceResponse response =null;
        InvokeThingServiceRequest request = new InvokeThingServiceRequest();
        request.setArgs(Args);
        request.setDeviceName(DeviceName);
        request.setIotId(IotId);
        request.setIdentifier(Identifier);
        request.setProductKey(ProductKey);
        request.setIotInstanceId(InstanceId);
        try {
            response = client.getAcsResponse(request);
            if (response.getSuccess() != null && response.getSuccess()) {
                LogUtil.print("服务执行成功");
                LogUtil.print(JSON.toJSONString(response));
            } else {
                LogUtil.print("服务执行失败");
                LogUtil.error(JSON.toJSONString(response));
            }
            return response.getData();
        } catch (ClientException e) {
            e.printStackTrace();
            LogUtil.error("服务执行失败！" + JSON.toJSONString(response));
        }
        return null;
    }
```

说明 如果要使用同步调用服务，需在定义物模型时，选择服务的**调用方式**为**同步**；开发设备端时，需编写处理同步调用服务的代码。

- 调用InvokeThingsService批量调用设备服务。

```
     /**
     * @param Identifier：服务的Identifier，必须传入。
     * @param Args：要启用服务的入参信息，必须传入。
     */
    public static void InvokeThingsService(String InstanceId, String IotId, String Prod
uctKey, List<String> DeviceNames,
                                           String Identifier, String Args) {
        InvokeThingsServiceResponse response =null;
        InvokeThingsServiceRequest request = new InvokeThingsServiceRequest();
        request.setArgs(Args);
        request.setIdentifier(Identifier);
        request.setDeviceNames(DeviceNames);
        request.setProductKey(ProductKey);
        request.setIotInstanceId(InstanceId);
        try {
            response = client.getAcsResponse(request);
            if (response.getSuccess() != null && response.getSuccess()) {
                LogUtil.print("批量调用设备服务成功");
                LogUtil.print(JSON.toJSONString(response));
            } else {
                LogUtil.print("批量调用设备服务失败");
                LogUtil.error(JSON.toJSONString(response));
            }
        } catch (ClientException e) {
            e.printStackTrace();
            LogUtil.error("批量调用设备服务失败！" + JSON.toJSONString(response));
        }
    }
```

设置属性和调用服务的请求示例：

```
public static void main(String[] args) {
        /**上线设备的设备名称和所属产品的ProductKey。*/
        String deviceName = "2pxuAQB2I7wGPmqq***";
        String deviceProductkey = "a1QbjI2***";
        /**对于企业版实例和新版公共实例，设置InstanceId为具体实例ID值，您可在物联网平台控制台的实例概览页面，查看当前实例的ID。
         *对于旧版公共实例,设置InstanceId为空值""。
         */
        String InstanceId = "iot-***tl02";
        //1 设置设备的属性。
        SetDeviceProperty(InstanceId, null, deviceProductkey, deviceName,"{\"hue\":0}");
        //2 批量设置设备属性。
        List<String>  deviceNames = new ArrayList<>();
        deviceNames.add(deviceName);
        SetDevicesProperty(InstanceId, deviceProductkey, deviceNames, "{\"hue\":0}");
        //3 调用设备的服务。
        InvokeThingService(InstanceId, null, deviceProductkey, deviceName, "ModifyVehicleInfo", "{}");
        //4 批量调用设备的服务。
        List<String>  deviceNamesService = new ArrayList<>();
        deviceNamesService.add(deviceName);
        InvokeThingsService(InstanceId, null, deviceProductkey, deviceNamesService, "ModifyVehicleInfo", "{}");
    }
```

运行调试

设备端SDK和云端SDK配置完成后，运行各SDK。

查看结果：

- 查看本地日志。

```
2022-04-08 05:29:41.165 - [ThingManagerForPopSDk.java] - SetDeviceProperty(35):设置设备的属性成功
2022-04-08 05:29:41.250 - [ThingManagerForPopSDk.java] - SetDeviceProperty(36):{"code":"","data":{"messageId":"1703383434"},"requestId":"049ED108-B2BB-58F5-8D70-FD3158AF794E","success":true}
2022-04-08 05:29:41.671 - [ThingManagerForPopSDk.java] - SetDevicesProperty(69):批量设置设备属性成功
2022-04-08 05:29:41.675 - [ThingManagerForPopSDk.java] - SetDevicesProperty(70):{"code":"","requestId":"31B3D707-71E8-5F18-A8E8-349CB92C013F","success":true}
2022-04-08 05:29:41.777 - [ThingManagerForPopSDk.java] - InvokeThingService(107):服务执行成功
2022-04-08 05:29:41.784 - [ThingManagerForPopSDk.java] - InvokeThingService(108):{"code":"","data":{"messageId":"1708568778"},"requestId":"63995E46-FE04-5F84-AF40-8F11873B253B","success":true}
2022-04-08 05:29:41.851 - [ThingManagerForPopSDk.java] - InvokeThingsService(148):批量调用设备服务成功
2022-04-08 05:29:41.854 - [ThingManagerForPopSDk.java] - InvokeThingsService(149):{"code":"","requestId":"A9BE6716-B1F5-5916-8DBA-1D17CB651659","success":true}
```

- 在物联网平台控制台，对应设备的**设备详情**页面，单击**默认模块**：
  - **运行状态**页签下，查看设备最后一次上报的属性值和属性数据记录。
  - **事件管理**页签下，查看设备上报的事件记录。
  - **服务调用**页签下，查看云端下发的服务调用记录。

2.4. M2M设备间通信

2.4.1. M2M设备间通信

M2M（即Machine-to-Machine）是一种端对端通信技术。本章节以智能灯和手机App连接为例，分别使用规则引擎数据流转和Topic消息路由来实现M2M设备间通信，主要介绍如何基于物联网平台构建一个M2M设备间通信架构。

智能灯与手机App的连接和通信请求都交由物联网平台承担，您不用担心高并发场景下的稳定通信等技术难点，也不需要购买大量服务器去承载这些请求，您只需要实现自己的业务系统即可。

具体实现过程，请参见以下文档：

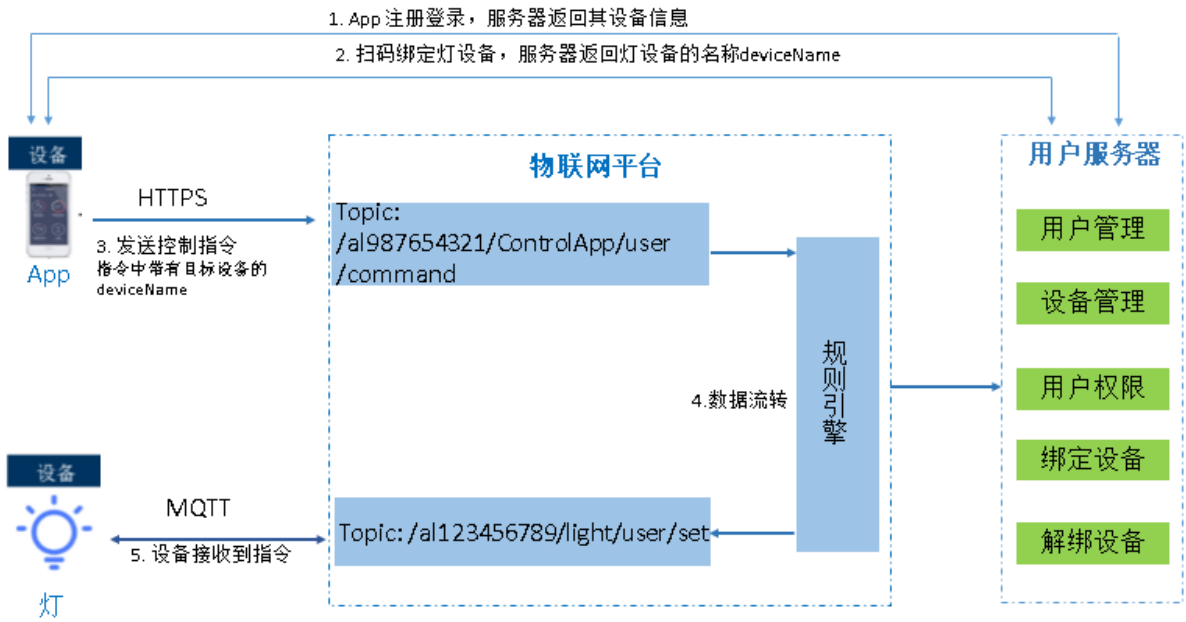
- 基于规则引擎的M2M设备间通信
- 基于Topic消息路由的M2M设备间通信

2.4.2. 基于规则引擎的M2M设备间通信

本文以智能灯和手机App连接为例，基于物联网平台的规则引擎数据流转功能，构建一个M2M设备间通信架构。

背景信息

使用手机App控制智能灯的流程：



操作步骤

1. 在物联网平台控制台，为智能灯设备创建产品和设备，定义功能等。具体操作，请参见创建产品、批量创建设备、单个添加物模型。

   本示例中，智能灯的**ProductKey**为al123456789；**DeviceName**为light。

2. 开发智能灯设备端。

   本示例中，设备与物联网平台间的通信协议为MQTT。

设备端SDK开发详情，请参见设备端Link SDK文档。

3. 在物联网平台，为手机App注册产品和设备。

   本示例中，手机App的**ProductKey**为al987654321；**DeviceName**为ControlApp。

   当手机App用户注册登录时，您的服务器将App的设备信息发送给手机App，让手机App可以作为一个设备连接到物联网平台。

4. 开发手机App。

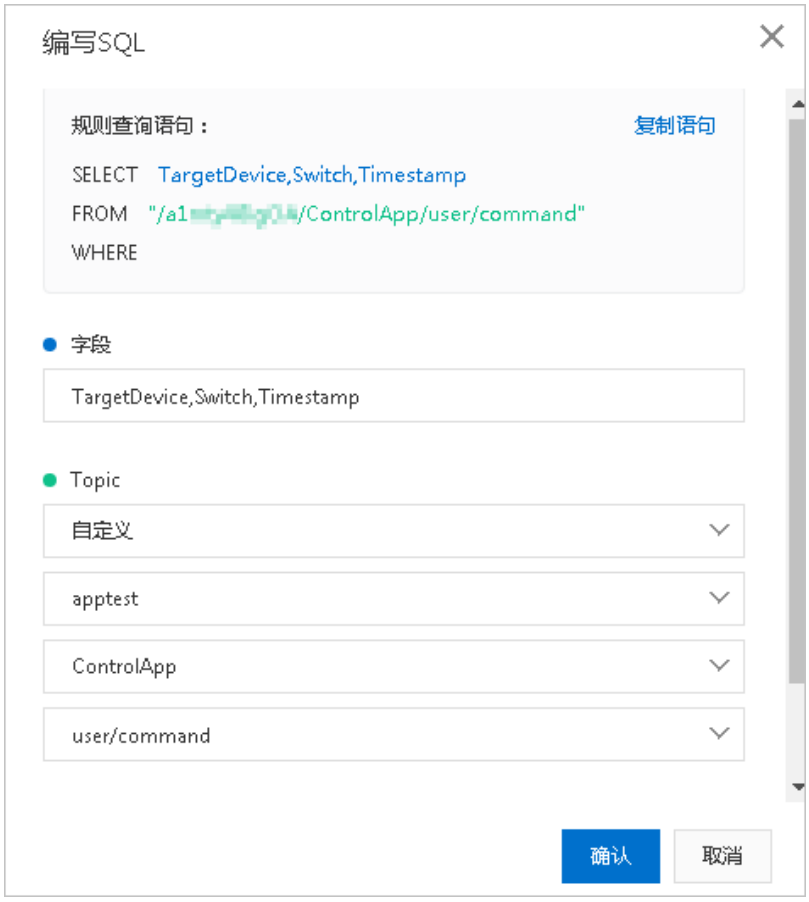
   本示例中，手机App与物联网平台间的通信协议为HTTPS。

   手机App发送的智能灯控制指令payload数据格式如下：

   ```
   {
       "TargetDevice": "light",
       "Switch": "off",
       "Timestamp": 1557750407000
   }
   ```

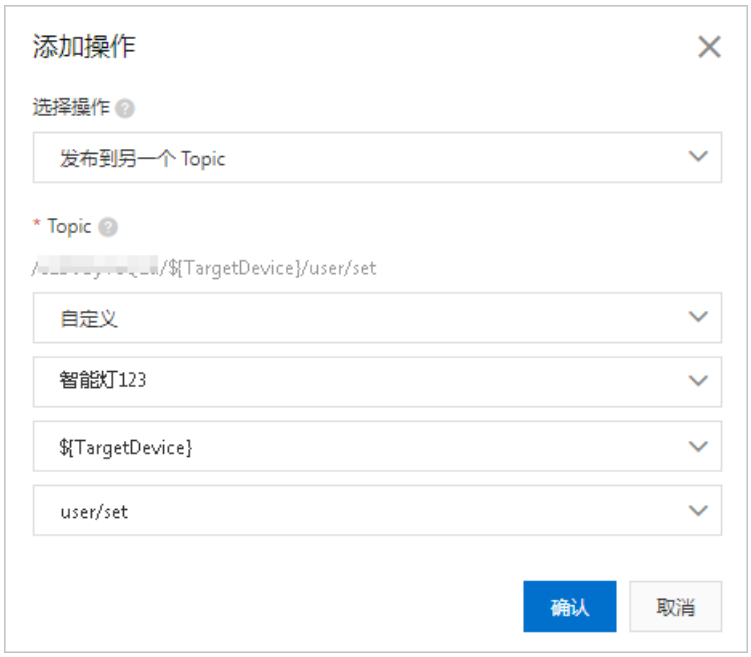
5. 开发智能灯设备端，实现智能灯设备连接物联网平台，接收并执行指令等功能。
6. 设置规则引擎数据流转规则，将手机App发布的指令流转到智能灯的Topic中。
   i. 登录物联网平台控制台，选择**规则引擎 > 云产品流转**。
   ii. 单击**创建规则**，在弹出对话框中，输入规则名称，单击**确认**。
   iii. 在弹出的**创建流转规则成功**对话框，单击**前往编辑**。
   iv. 在**数据流转规则**页面，单击**处理数据**下的**编写SQL**。
   v. 在**编写SQL**对话框，编写处理转发消息内容的SQL，单击**确认**。该SQL将从手机App设备的Topic消息中，筛选出要发送给智能灯的消息字段。

      本示例中，SQL将筛选出消息中的目标设备的名称TargetDevice，消息时间戳Timestamp和Switch三个字段的值。


编写SQL
规则查询语句：
复制语句
SELECT TargetDevice,Switch,Timestamp
FROM "/a1[redacted]/ControlApp/user/command"
WHERE
字段
TargetDevice,Switch,Timestamp
Topic
自定义
apptest
ControlApp
user/command
确认
取消

vi. 在**数据流转规则**页面，单击**转发数据**下的**添加操作**。在弹出对话框中，设置转发消息目的地。将智能灯设备具有订阅权限的Topic作为接收手机App指令的Topic。

> 说明
> - 选择**发布到另一个Topic中**。
> - 指定设备时，需使用转义符 `${TargetDevice}` 通配所有目标智能灯。如本示例中 `${TargetDevice}` 会被转义成智能灯设备名称 `light` 。



7. 手机App用户通过扫码将App与智能灯绑定。

   当App向您的服务器发送绑定某设备的请求后，您的服务器将返回绑定成功的智能灯设备名称deviceName。本示例中，智能灯设备名称为light。

8. 手机App用户通过App发送控制指令。

   i. 手机App向物联网平台中的Topic发送指令，如本示例中，App对应的发送指令Topic： `/a1987654321/ControlApp/user/command` 。

   ii. 物联网平台再根据您定义的数据流转规则，将指令信息发送给智能灯的Topic，如本示例中定义的Topic为： `/a1123456789/light/user/set` 。

   iii. 智能灯接收到指令后，执行相关操作。

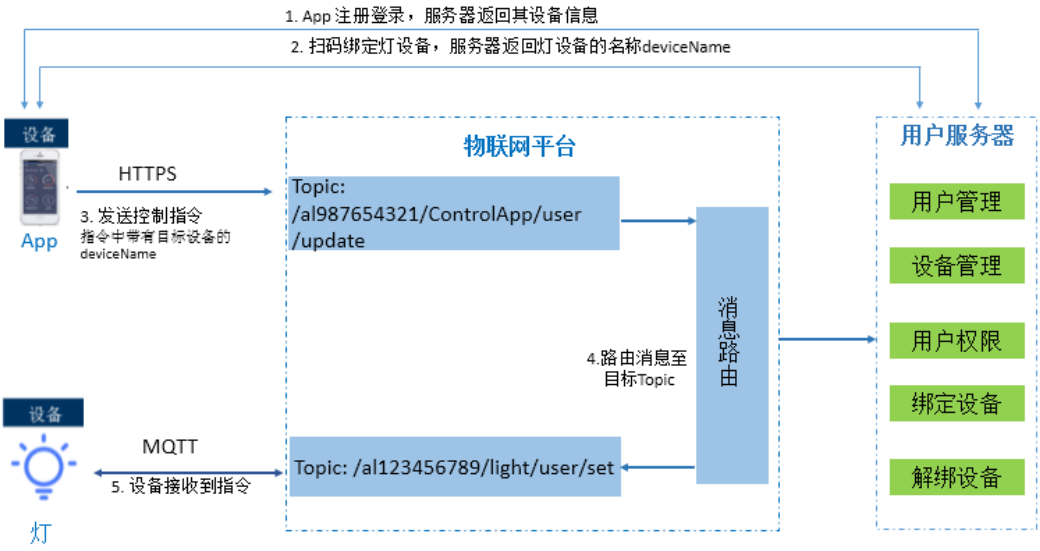
> 说明 手机App也可以向您的服务器发送解绑设备的请求。解绑后，该手机App将不再控制该智能灯。

2.4.3. 基于Topic消息路由的M2M设备间通信

本文以智能灯和手机App连接为例，基于物联网平台的Topic消息路由服务，构建一个M2M设备间通信架构。

背景信息

智能灯控制流程如下图：



操作步骤

1. 在物联网平台控制台，为智能灯设备创建产品和设备，定义功能等。具体操作，请参见创建产品、批量创建设备、单个添加物模型。

   本示例中，智能灯的**ProductKey**为al123456789；**DeviceName**为light。

2. 开发智能灯设备端。

   本示例中，设备与物联网平台间的通信协议为MQTT。

   设备端SDK开发详情，请参见设备端Link SDK文档。

3. 在物联网平台，为手机App注册产品和设备。

   上图示例中，手机App的ProductKey为al987654321；DeviceName为ControlApp。

   当手机App用户注册登录时，服务器将App设备信息发送给手机App。手机App可以作为一个设备连接到物联网平台。

4. 使用服务器，调用云端接口CreateTopicRouteTable，创建App Topic与智能灯Topic之间的消息路由关系。

   - 将入参SrcTopic指定为App的Topic：`/al987654321/ControlApp/user/update`。
   - 将入参DstTopics指定为智能灯的Topic：`/al123456789/light/user/set`。

5. 开发手机App。

   本示例中，手机App与物联网平台间的通信协议为HTTPS。

   手机App发送的智能灯控制指令payload数据格式如下：

   ```
   {
       "TargetDevice": "light",
       "Switch": "off",
       "Timestamp": 1557750407000
   }
   ```

6. 手机App用户通过扫码，将App与智能灯绑定。

   当App向服务器发送绑定设备的请求后，服务器将返回绑定成功的智能灯设备名称deviceName。本示

例中，智能灯设备名称为light。

7. 通过App发送控制指令。
   i. 手机App发送指令到Topic：`/al987654321/ControlApp/user/update`。指令为JSON格式的数据。
   ii. 物联网平台根据已定义的Topic路由关系，将指令信息路由到智能灯设备的Topic：`/al123456789/light/user/set`。
   iii. 智能灯设备接收到指令后，执行相关操作。

> **说明** 可配置手机App向服务器发送解绑请求，触发服务器调用云端接口DeleteTopicRouteTable，删除消息路由关系。路由关系删除后，该手机App将不再控制该智能灯。

2.5. 服务端订阅（MNS）

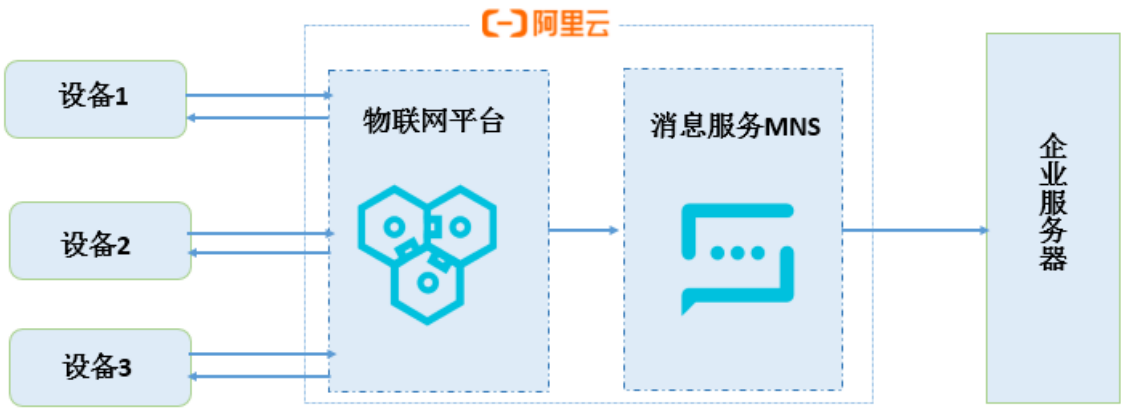
本示例介绍如何配置服务端订阅，将产品下的设备状态变化消息推送到消息服务（MNS）队列中。服务器通过监听MNS队列接收设备状态变化消息。

前提条件

- 开通阿里云产品：
  - 物联网平台
  - 消息服务
- 安装Java开发环境Eclipse。

背景信息

数据流转流程如下图所示。

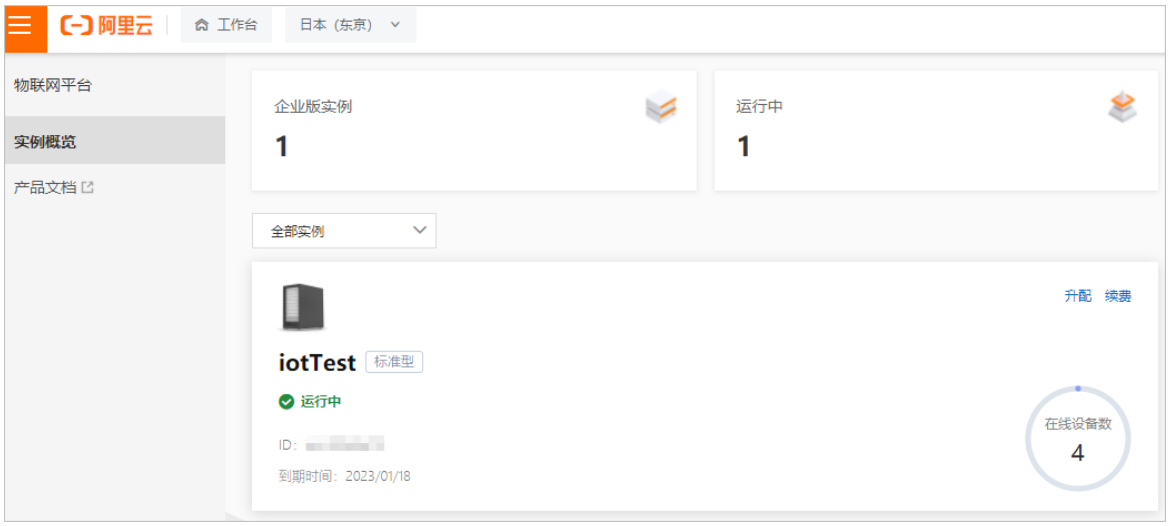


配置服务端订阅

首先在物联网平台控制台创建MNS服务端订阅，选择要订阅的消息类型。

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，选择**设备管理 > 产品**，再单击**创建产品**，创建一个气体监测仪产品。
4. 选择**设备 > 添加设备**，在刚创建的气体监测仪产品下创建设备。

   设备证书信息将会用于设备端SDK开发配置。
5. 在左侧导航栏，选择**规则引擎 > 服务端订阅**，再单击**创建订阅**，创建MNS服务端订阅。详细操作指导，请参见使用MNS服务端订阅。

   > **说明** 首次设置推送到MNS时，需单击提示中的**授权**，进入RAM控制台同意授权IoT访问MNS。

   本示例中，选择了**设备状态变化通知**，即该产品下所有设备的状态变化消息，都会被推送到MNS队列中。

   订阅成功后，物联网平台会在MNS中，自动创建一个接收物联网平台消息的队列。队列名称格式为：`aliyun-iot-${yourProductKey}`。您在配置MNS SDK监听消息时，需填入该队列名称。

   在订阅列表中，单击MNS右侧的图标，可查看MNS队列名称。

配置服务端MNS SDK接收消息

本示例使用MNS Java SDK Demo。

1. 访问MNS Java SDK下载，下载sample包*aliyun-sdk-mns-samples1.1.8*，并解压缩。
2. 打开Eclipse，选择导入工程，导入*aliyun-sdk-mns-samples*文件夹。
3. 在计算机的本地目录*C:\Users\${YourComputerUserName}*下，添加一个PROPERTIES文件*.aliyun-mns*，并在文件中输入MNS访问身份认证信息，格式如下：

```
mns.accountendpoint=http://$your_accountId.mns.$your_regionId.aliyuncs.com
mns.accesskeyid=$your_accesskeyid
mns.accesskeysecret=$your_accesskeysecret
```

参数	说明
accountendpoint	您的MNS服务Endpoint。请在消息服务控制台，选择队列所在地域单击**获取Endpoint**查看。
accesskeyid	您账号的AccessKey ID。 将光标定位到您的账号头像上，选择**accesskeys**，进入**安全信息管理**页，可创建或查看您的AccessKey。
accesskeysecret	您账号的AccessKey Secret。查看方法同上AccessKey ID。

4. 在*src\main\java\com.aliyun.mns.sample.Queue*目录下的*ComsumerDemo*文件中，配置物联网平台自动创建的消息服务队列名称。

```
    public static void main(String[] args) {
        CloudAccount account = new CloudAccount(
                ServiceSettings.getMNSAccessKeyId(),
                ServiceSettings.getMNSAccessKeySecret(),
                ServiceSettings.getMNSAccountEndpoint());
        MNSClient client = account.getMNSClient(); //client初始化
        // 提取消息
        try{
            CloudQueue queue = client.getQueueRef("aliyun-iot-a1eN7La****");// 替换为物联网平台自动创建的队列
            for (int i = 0; i < 10; i++)
            {
                Message popMsg = queue.popMessage(); //长轮询等待时间
                if (popMsg != null){
                    System.out.println("message handle: " + popMsg.getReceiptHandle());
                    System.out.println("message body: " + popMsg.getMessageBodyAsString()); //获取原始消息
                    System.out.println("message id: " + popMsg.getMessageId());
                    System.out.println("message dequeue count:" + popMsg.getDequeueCount());
                    //<<to add your special logic.>>
                    //从队列中删除消息
                    queue.deleteMessage(popMsg.getReceiptHandle());
                    System.out.println("delete message successfully.\n");
                }
            }
        }
```

5. 运行ComsumerDemo。

配置设备端SDK

1. 访问下载设备端SDK，选择Java SDK。
2. 在Java SDK 工程配置文档页底部，单击下载Java SDK Demo，然后解压缩。

3. 在Java开发工具中，导入工程，选择导入*JavaLinkKit Demo*文件夹。
4. 在*device_id*文件中，填入设备证书信息。

```
{
  "productKey": "a1e[illegible]frh",
  "deviceName": "ai[illegible]",
  "deviceSecret": "u[illegible]",
  "subDevice": [
    {
      "productKey": "",
      "deviceName": ""
    },
    {
      "productKey": "",
      "deviceName": ""
    }
  ]
}
```

5. 在*src\devicesdk\demo*目录下的*MqttSample*文件中，填入设备信息，将publish对应的Topic配置为您的设备Topic。

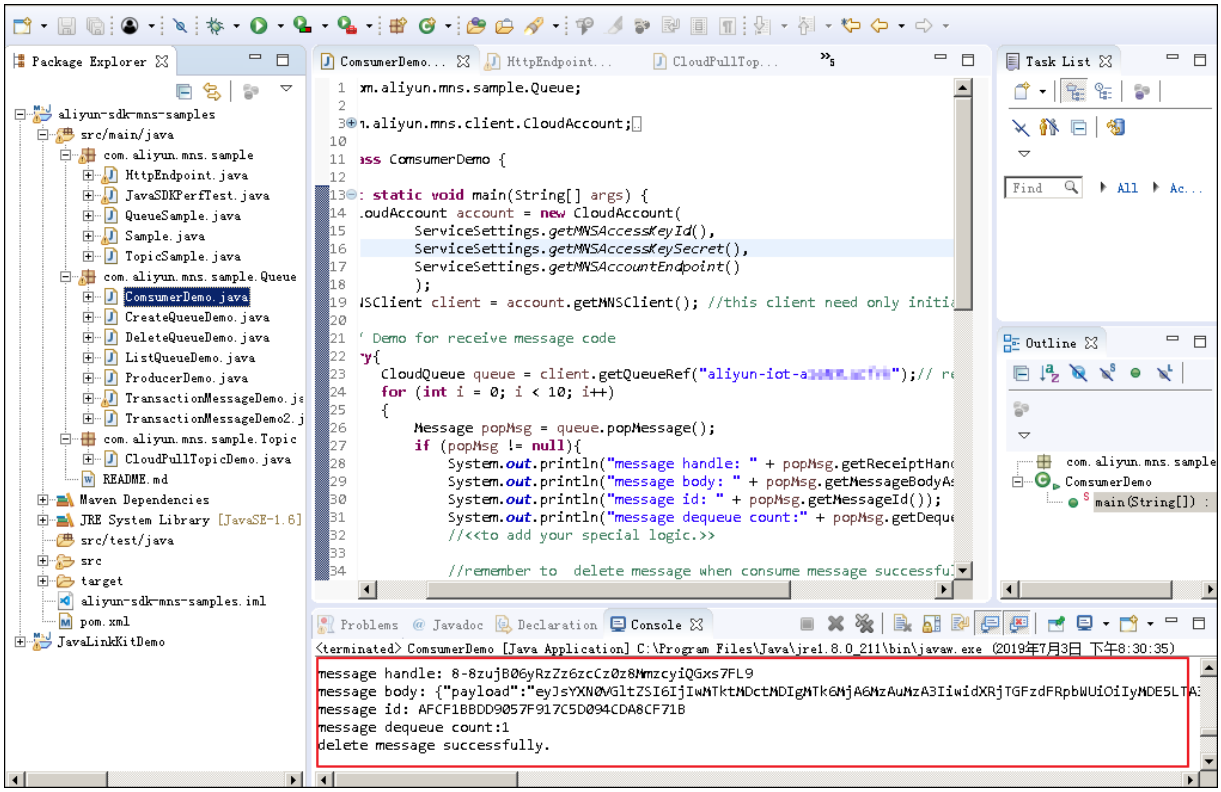
```
public class MqttSample extends BaseSample {
    final static String TAG = "MqttSample";

    public MqttSample(String a[illegible]rfh, String a[illegible]er) {
        super(a[illegible]rfh, a[illegible]er);
    }

    /**
     * 发布接口示例
     */
    public void publish() {
        MqttPublishRequest request = new MqttPublishRequest();
        // topic 用户根据实际场景填写
        request.topic = "/sys/" + productKey + "/" + deviceName + "/thing/deviceinfo/update";
        /**
         * 订阅回复的 replyTopic
         * 如果业务有相应的响应需求，可以设置 replyTopic，且 isRPC=true
         */
```

6. 运行设备连接Demo：MqttSample。

结果验证

设备端SDK运行后，设备上线消息通过服务端订阅发送到消息服务队列中。消息服务SDK从队列中接收消息，并同时将已接收的消息从队列中删除。

下图展示消息服务SDK接收并删除消息。



2.6. 云端解析设备透传数据

在物联网业务场景中，资源受限或配置较低的设备，不适合直接构造JSON数据与物联网平台通信，可将原数据直接透传到物联网平台。物联网平台会调用您提交的数据解析脚本，将设备上行数据解析为物联网平台定义的标准格式（Alink JSON），再进行业务处理。

背景信息

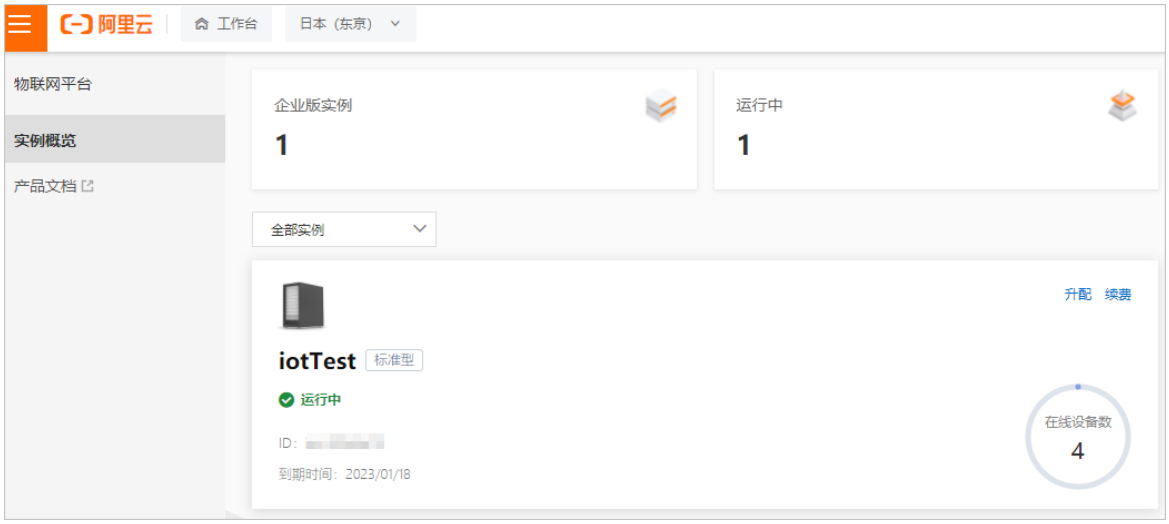
数据解析流程图如下所示。

本文以环境数据采集设备为例，为您介绍数据解析具体操作步骤。

设备端接入物联网平台

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

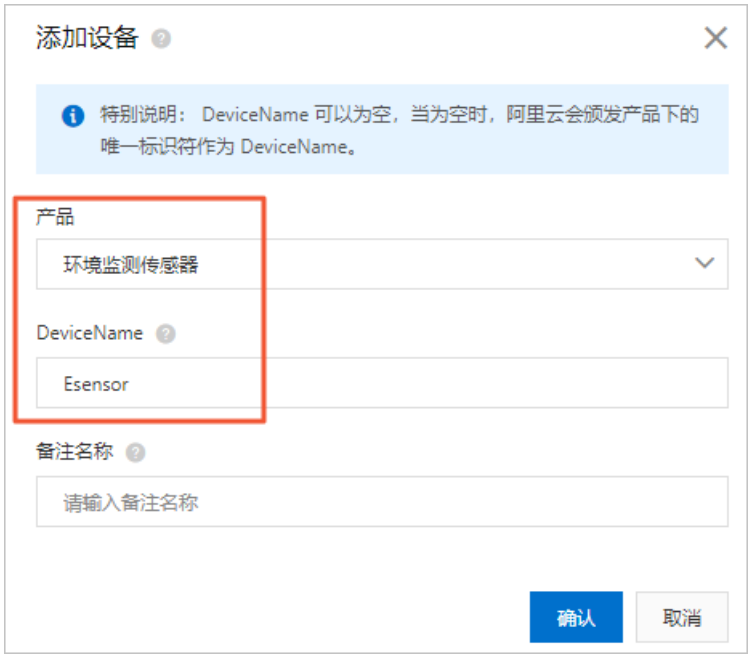
> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，选择**设备管理 > 产品**，单击**创建产品**，创建一个产品：环境监测传感器。
   数据格式选择**透传/自定义**，其他使用默认设置。
4. 产品创建成功后，单击**前往定义物模型**，添加物模型，然后发布上线。
   本文提供了示例的物模型TSL内容，您可批量导入，请参见批量添加物模型。

功能类型	功能名称（全部）	标识符	数据类型	数据定义
属性	温度 自定义	temperature	float (单精度浮点型)	取值范围：-10 ~ 50
属性	湿度 自定义	humidity	int32 (整数型)	取值范围：0 ~ 100
属性	二氧化碳 自定义	co2	int32 (整数型)	取值范围：0 ~ 10000
属性	甲醛 自定义	hcho	float (单精度浮点型)	取值范围：0 ~ 10
属性	PM25 自定义	PM25	int32 (整数型)	取值范围：0 ~ 1000
属性	光照强度 自定义	lightLux	float (单精度浮点型)	取值范围：0 ~ 10000

5. 在左侧导航栏，选择**设备**，单击**添加设备**，在环境监测传感器产品下添加设备：Esensor。



设备创建成功后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

6. 开发设备端，并测试运行。

   本示例使用物联网平台提供的Node.js SDK开发设备，并设置设备端模拟上报ICA标准物模型数据，测试运行设备端SDK。

   设备端开发更多操作说明，请参见Link SDK。

   SDK开发示例代码如下：

```
const mqtt = require('aliyun-iot-mqtt');
// 1. 设备身份信息
var options = {
    productKey: "g4yf***",
    deviceName: "Esensor",
    deviceSecret: "e14f81***",
    host: "iot-***.mqtt.iothub.aliyuncs.com"
};
// 1. 建立连接
const client = mqtt.getAliyunIotMqttClient(options);
// 2. 监听云端指令
client.subscribe(`/${options.productKey}/${options.deviceName}/user/get`)
client.on('message', function(topic, message) {
    console.log("topic " + topic)
    console.log("message " + message)
})
setInterval(function() {
    // 3.上报环境数据
    client.publish(`/sys/${options.productKey}/${options.deviceName}/thing/event/proper
ty/post`, getPostData(), { qos: 0 });
}, 5 * 1000);
function getPostData() {
    const payloadJson = {
        id: Date.now(),
        version: "1.0",
        params: {
            PM25: Math.floor((Math.random() * 1000)),
            temperature: Math.floor((Math.random() * 60) - 10),
            humidity: Math.floor((Math.random() * 100)),
            co2: Math.floor((Math.random() * 10000)),
            hcho: Math.floor((Math.random() * 5)),
            lightLux: Math.floor((Math.random() * 10000))
        },
        method: "thing.event.property.post"
    }
    console.log("payloadJson " + JSON.stringify(payloadJson))
    return JSON.stringify(payloadJson);
}
```

   设备端成功接入物联网平台后，在物联网平台控制台**设备**页，该设备状态显示为**在线**。

单击设备Esensor操作栏的**查看**，单击**物模型数据**。如下图所示，因产品数据格式为**透传/自定义**，模拟上报的标准物模型数据不能在**运行状态**页签显示。

在**监控运维 > 日志服务 > 云端运行日志**中，查询该设备的**设备到云消息**，查看设备模拟上报的标准物模型数据，对应的Hex格式消息内容。

本示例中，Hex格式消息内容为：`0xaa1fc800003710ff0005d76b15001c013400ad04ffff0400ffff18003000ff2e`。

编写数据解析脚本

在物联网平台控制台，编辑、提交脚本，并模拟数据解析。

1. 在物联网平台控制台左侧导航栏，选择**设备管理 > 产品**。
2. 在**产品**页，单击产品对应的**查看**。
3. 在**产品详情**页，单击**数据解析**页签。
4. 在**数据解析**页签下的**编辑脚本**输入框中，输入数据解析脚本。

   根据设备数据协议内容编写解析脚本。本示例中的设备数据消息体结构如下表所示。

Byte	说明	备注
12	PM2.5值低字节	返回：PM2.5值，取值范围0~999ug/m^3。
13	PM2.5值高字节	
14	温度值*10低字节	返回：温度值，取值范围-10°C~50°C。
15	温度值*10高字节	
16	湿度值低字节	返回：湿度值，取值范围0~99%。
17	湿度值高字节	
18	二氧化碳含量低字节	返回：二氧化碳含量，取值范围0~9999mg/m^3。
19	二氧化碳含量高字节	

Byte	说明	备注
22	甲醛含量*100低字节	返回：甲醛含量，取值范围0~9.99。
23	甲醛含量*100高字节	
28	照度值低字节	返回：照度值，单位lux。
29	照度值高字节	

示例中的环境采集设备只有数据上报功能，因此只需要编写上行数据解析函数rawDataToProtocol，无需实现protocolToRawData。

本示例的数据解析脚本如下：

```
var PROPERTY_REPORT_METHOD = 'thing.event.property.post';
//上行数据，自定义格式转物模型JSON格式。
function rawDataToProtocol(bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = new Object();
        //属性上报method。
        jsonMap['method'] = PROPERTY_REPORT_METHOD;
        //协议版本号，固定字段，取值1.0。
        jsonMap['version'] = '1.0';
        //表示该次请求的ID。
        jsonMap['id'] = new Date().getTime();
        var params = {};
        //12、13对应产品属性中PM2.5。
        params['PM25'] = (dataView.getUint8(13)*256+dataView.getUint8(12));
        //14、15对应产品属性中temperature。
        params['temperature'] = (dataView.getUint8(15)*256+dataView.getUint8(14))/10;
        //16、17对应产品属性中humidity。
        params['humidity'] = (dataView.getUint8(17)*256+dataView.getUint8(16));
        //18、19对应产品属性中co2。
        params['co2'] = (dataView.getUint8(19)*256+dataView.getUint8(18));
        //22、23对应产品属性中甲醛hcho。
        params['hcho'] = (dataView.getUint8(23)*256+dataView.getUint8(22))/100;
        //28、29对应产品属性中光照lightLux。
        params['lightLux'] = (dataView.getUint8(29)*256+dataView.getUint8(28));
        jsonMap['params'] = params;
    return jsonMap;
}
//下行指令，物模型JSON格式转自定义格式。
function protocolToRawData(json) {
    var payloadArray = [1];//此设备只有上报数据功能，无法接收云端指令。
    return payloadArray;
}
//将设备自定义Topic数据转换为JSON格式数据。
function transformPayload(topic, rawData) {
    var jsonObj = {}
    return jsonObj;
}
```

5. 测试数据解析。
   i. 选择模拟类型为**设备上报数据**。
   ii. 在**模拟输入**下的输入框中，输入一个模拟数据。

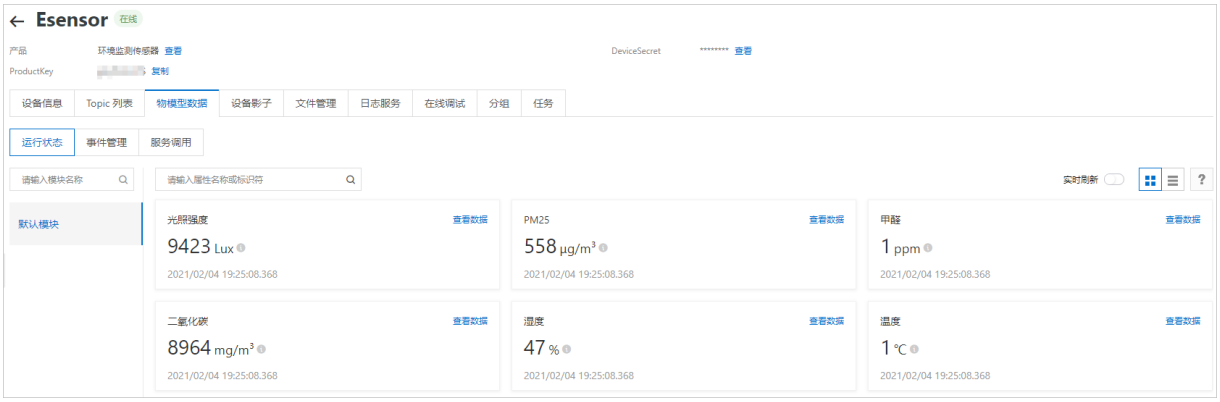
   模拟数据可使用测试运行设备端后，在**日志服务**页，查看到的设备端上报数据的Hex格式内容。例如：`0xaa1fc800003710ff0005d76b15001c013400ad04ffff0400ffff18003000ff2e`。

iii. 单击**执行** 。

右侧**运行结果**栏显示解析结果如下图所示。

6. 确认脚本能正确解析数据后，单击**提交**，将脚本提交到物联网平台。

脚本提交后，可运行设备端SDK脚本调试验证。设备端再向物联网平台上报数据时，物联网平台会调用脚本进行数据解析。解析后的数据将显示在设备对应**设备详情**页的**物模型数据** > **运行状态**页签下。



附：物模型TSL

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "version": "1.0",
    "productKey": "g4***"
  },
  "properties": [
    {
      "identifier": "lightLux",
      "name": "光照强度",
      "accessMode": "rw",
      "required": false,
      "dataType": {
        "type": "float",
```

```
        "specs": {
          "min": "0",
          "max": "10000",
          "unit": "Lux",
          "unitName": "照度",
          "step": "0.1"
        }
      }
    },
    {
      "identifier": "PM25",
      "name": "PM25",
      "accessMode": "rw",
      "required": false,
      "dataType": {
        "type": "int",
        "specs": {
          "min": "0",
          "max": "1000",
          "unit": "μg/m³",
          "unitName": "微克每立方米",
          "step": "1"
        }
      }
    },
    {
      "identifier": "hcho",
      "name": "甲醛",
      "accessMode": "rw",
      "desc": "HCHOValue",
      "required": false,
      "dataType": {
        "type": "float",
        "specs": {
          "min": "0",
          "max": "10",
          "unit": "ppm",
          "unitName": "百万分率",
          "step": "0.1"
        }
      }
    },
    {
      "identifier": "co2",
      "name": "二氧化碳",
      "accessMode": "rw",
      "required": false,
      "dataType": {
        "type": "int",
        "specs": {
          "min": "0",
          "max": "10000",
          "unit": "mg/m³",
          "unitName": "毫克每立方米",
```

```
          "step": "1"
        }
      }
    },
    {
      "identifier": "humidity",
      "name": "湿度",
      "accessMode": "rw",
      "required": false,
      "dataType": {
        "type": "int",
        "specs": {
          "min": "0",
          "max": "100",
          "unit": "%",
          "unitName": "百分比",
          "step": "1"
        }
      }
    },
    {
      "identifier": "temperature",
      "name": "温度",
      "accessMode": "rw",
      "desc": "电机工作温度",
      "required": false,
      "dataType": {
        "type": "float",
        "specs": {
          "min": "-10",
          "max": "50",
          "unit": "℃",
          "step": "0.1"
        }
      }
    }
  ],
  "events": [
    {
      "identifier": "post",
      "name": "post",
      "type": "info",
      "required": true,
      "desc": "属性上报",
      "method": "thing.event.property.post",
      "outputData": [
        {
          "identifier": "lightLux",
          "name": "光照强度",
          "dataType": {
            "type": "float",
            "specs": {
              "min": "0",
              "max": "10000",
```

```
            "unit": "Lux",
            "unitName": "照度",
            "step": "0.1"
          }
        }
      },
      {
        "identifier": "PM25",
        "name": "PM25",
        "dataType": {
          "type": "int",
          "specs": {
            "min": "0",
            "max": "1000",
            "unit": "μg/m³",
            "unitName": "微克每立方米",
            "step": "1"
          }
        }
      },
      {
        "identifier": "hcho",
        "name": "甲醛",
        "dataType": {
          "type": "float",
          "specs": {
            "min": "0",
            "max": "10",
            "unit": "ppm",
            "unitName": "百万分率",
            "step": "0.1"
          }
        }
      },
      {
        "identifier": "co2",
        "name": "二氧化碳",
        "dataType": {
          "type": "int",
          "specs": {
            "min": "0",
            "max": "10000",
            "unit": "mg/m³",
            "unitName": "毫克每立方米",
            "step": "1"
          }
        }
      },
      {
        "identifier": "humidity",
        "name": "湿度",
        "dataType": {
          "type": "int",
          "specs": {
```

```
              "min": "0",
              "max": "100",
              "unit": "%",
              "unitName": "百分比",
              "step": "1"
            }
          }
        },
        {
          "identifier": "temperature",
          "name": "温度",
          "dataType": {
            "type": "float",
            "specs": {
              "min": "-10",
              "max": "50",
              "unit": "℃",
              "step": "0.1"
            }
          }
        }
      ]
    }
  ],
  "services": [
    {
      "identifier": "set",
      "name": "set",
      "required": true,
      "callType": "async",
      "desc": "属性设置",
      "method": "thing.service.property.set",
      "inputData": [
        {
          "identifier": "lightLux",
          "name": "光照强度",
          "dataType": {
            "type": "float",
            "specs": {
              "min": "0",
              "max": "10000",
              "unit": "Lux",
              "unitName": "照度",
              "step": "0.1"
            }
          }
        },
        {
          "identifier": "PM25",
          "name": "PM25",
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "1000"
```

```
            "max": "1000",
            "unit": "μg/m³",
            "unitName": "微克每立方米",
            "step": "1"
          }
        }
      },
      {
        "identifier": "hcho",
        "name": "甲醛",
        "dataType": {
          "type": "float",
          "specs": {
            "min": "0",
            "max": "10",
            "unit": "ppm",
            "unitName": "百万分率",
            "step": "0.1"
          }
        }
      },
      {
        "identifier": "co2",
        "name": "二氧化碳",
        "dataType": {
          "type": "int",
          "specs": {
            "min": "0",
            "max": "10000",
            "unit": "mg/m³",
            "unitName": "毫克每立方米",
            "step": "1"
          }
        }
      },
      {
        "identifier": "humidity",
        "name": "湿度",
        "dataType": {
          "type": "int",
          "specs": {
            "min": "0",
            "max": "100",
            "unit": "%",
            "unitName": "百分比",
            "step": "1"
          }
        }
      },
      {
        "identifier": "temperature",
        "name": "温度",
        "dataType": {
          "type": "float",
          "specs": {
```

```
            "min": "-10",
            "max": "50",
            "unit": "℃",
            "step": "0.1"
          }
        }
      }
    ],
    "outputData": []
  },
  {
    "identifier": "get",
    "name": "get",
    "required": true,
    "callType": "async",
    "desc": "属性获取",
    "method": "thing.service.property.get",
    "inputData": [
      "lightLux",
      "PM25",
      "hcho",
      "co2",
      "humidity",
      "temperature"
    ],
    "outputData": [
      {
        "identifier": "lightLux",
        "name": "光照强度",
        "dataType": {
          "type": "float",
          "specs": {
            "min": "0",
            "max": "10000",
            "unit": "Lux",
            "unitName": "照度",
            "step": "0.1"
          }
        }
      },
      {
        "identifier": "PM25",
        "name": "PM25",
        "dataType": {
          "type": "int",
          "specs": {
            "min": "0",
            "max": "1000",
            "unit": "μg/m³",
            "unitName": "微克每立方米",
            "step": "1"
          }
        }
      },
```

```
        {
          "identifier": "hcho",
          "name": "甲醛",
          "dataType": {
            "type": "float",
            "specs": {
              "min": "0",
              "max": "10",
              "unit": "ppm",
              "unitName": "百万分率",
              "step": "0.1"
            }
          }
        },
        {
          "identifier": "co2",
          "name": "二氧化碳",
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "10000",
              "unit": "mg/m³",
              "unitName": "毫克每立方米",
              "step": "1"
            }
          }
        },
        {
          "identifier": "humidity",
          "name": "湿度",
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "100",
              "unit": "%",
              "unitName": "百分比",
              "step": "1"
            }
          }
        },
        {
          "identifier": "temperature",
          "name": "温度",
          "dataType": {
            "type": "float",
            "specs": {
              "min": "-10",
              "max": "50",
              "unit": "℃",
              "step": "0.1"
            }
          }
```

```
            }
          ]
        }
    ]
}
```

3.设备管理

3.1. 设置期望属性值控制灯泡状态

物联网平台支持设置期望属性值，通过缓存设备属性的期望值，实现云端对设备属性的控制。本文介绍设置期望属性值，实现物联网平台控制灯泡状态的相关操作。

背景信息

灯泡设备接入物联网平台后，如需从物联网平台控制灯泡工作状态（1：打开；0：关闭），需要灯泡一直保持连网在线。实际情况下，灯泡可能无法一直在线。

您可在物联网平台设置设备期望属性值，使其存储在云端。设备在线后，可读取物联网平台存储的期望属性值，来更新自身属性值。然后，设备会将更新后的属性值上报至云端，在物联网平台的设备运行状态中显示。

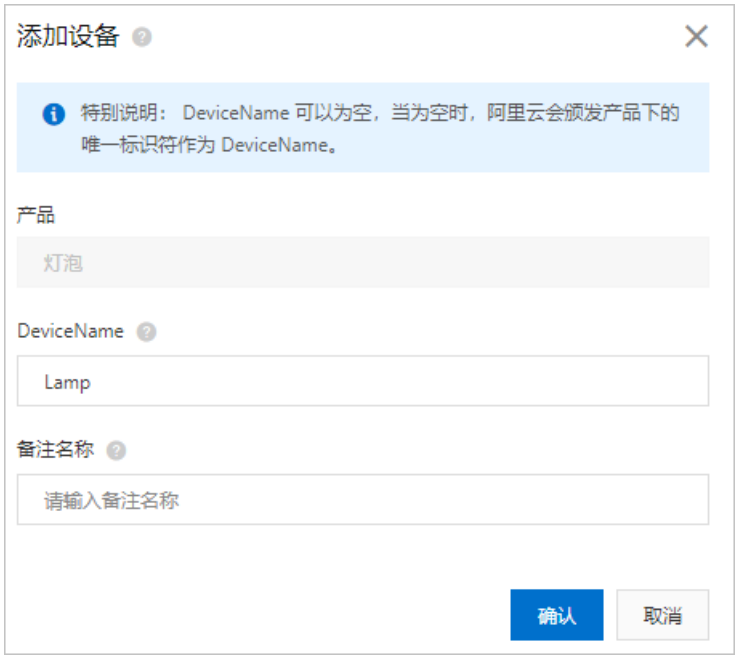
创建产品和设备

1. 登录物联网平台控制台。
2. 在左侧导航栏，选择**设备管理 > 产品**，单击**创建产品**，创建一个产品：灯泡。

3. 产品创建成功后，单击**前往定义物模型**，为产品添加物模型并发布，请参见单个添加物模型。

   如图所示，本文添加属性工作状态（LightStatus）。

4. 在左侧导航栏，选择**设备管理 > 设备**，单击**添加设备**，在灯泡产品下添加设备：Lamp。



设备添加成功后，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。

您可在设备列表，单击Lamp对应的**查看**进入**设备详情**页，查看**运行状态**，设备属性值和期望属性值都为空。此时期望属性值版本为0。

在云端设置、查询期望属性值

您可在云端通过调用API，设置、获取设备最新期望属性值。

具体操作，请参见云端API文档。本文以Java SDK（云端）为例。

- 调用SetDeviceDesiredProperty，设置期望属性值。

```
DefaultProfile profile = DefaultProfile.getProfile(
        "<RegionId>",           // 地域ID
        "<accessKey>",      //阿里云账号的AccessKey ID
        "<accessSecret>"); 阿里云账号AccessKey Secret
IAcsClient client = new DefaultAcsClient(profile);
// 创建API请求并设置参数
SetDeviceDesiredPropertyRequest request = new SetDeviceDesiredPropertyRequest();
request.setIotInstanceId("iot-060***");
request.setDeviceName("Lamp");
request.setProductKey("g4r***");
// 待设置的属性identifier与期望属性值
request.setItems("{\"LightStatus\": 1}");
request.setVersions("{\"LightStatus\": 0}");
// 发起请求并处理应答或异常
try {
    SetDeviceDesiredPropertyResponse response = client.getAcsResponse(request);
    System.out.println(new Gson().toJson(response));
} catch (ServerException e) {
    e.printStackTrace();
} catch (ClientException e) {
    System.out.println("ErrCode:" + e.getErrCode());
    System.out.println("ErrMsg:" + e.getErrMsg());
    System.out.println("RequestId:" + e.getRequestId());
}
```

- 调用QueryDeviceDesiredProperty，查看设备的期望属性值。

```
DefaultProfile profile = DefaultProfile.getProfile(
        "<RegionId>",           // 地域ID
        "<accessKey>",       /阿里云账号的AccessKey ID
        "<accessSecret>"); 阿里云账号Access Key Secret
IAcsClient client = new DefaultAcsClient(profile);
// 创建API请求并设置参数
QueryDeviceDesiredPropertyRequest request = new QueryDeviceDesiredPropertyRequest();
request.setIotInstanceId("iot-060a02fq");
request.setProductKey("g4rmjb5q40O");
request.setDeviceName("Lamp");
// 待查询的属性identifier列表。如不指定则查询所有属性（只读属性除外）的期望属性值。
List<String> identifierList = new ArrayList<String>();
identifierList.add("LightStatus");
request.setIdentifiers(identifierList);
// 发起请求并处理应答或异常
try {
    QueryDeviceDesiredPropertyResponse response = client.getAcsResponse(request);
    System.out.println(new Gson().toJson(response));
} catch (ServerException e) {
    e.printStackTrace();
} catch (ClientException e) {
    System.out.println("ErrCode:" + e.getErrCode());
    System.out.println("ErrMsg:" + e.getErrMsg());
    System.out.println("RequestId:" + e.getRequestId());
}
```

有关如何设置代码中参数，请参见Java SDK使用说明。

在云端设置设备期望属性值后，设备运行状态显示该值。

设备端开发

设备获取期望属性值，有两种场景：

- 灯泡重新上线时，主动获取云端缓存的期望属性值。
- 灯泡正处于上线状态，实时接收云端推送的期望属性值。

设备端开发更多信息，请参见下载设备端SDK。

本文提供了完整的设备端Demo示例，请参见下文附录：设备端Demo代码。

1. 填入设备证书和地域信息。

```
/**
* 设备证书信息
*/
private static String productKey = "******";
private static String deviceName = "********";
private static String deviceSecret = "***************";
/**
* MQTT连接信息
*/
private static String regionId = "******";
```

> 说明
>
> - 设备证书信息，请参见上文创建产品和设备。
> - regionId 为您的服务所在地域对应的Region ID。请在物联网平台控制台左上角，查看您服务所在的地域。表达方法，请参见地域和可用区。

2. 添加以下方法，用于变更实际灯泡的属性，并在属性变更后，主动将信息上报到最新属性值中。

```
/**
 * 真实设备处理属性变更时，在以下两个场下会被调用：
 * 场景1．设备联网后主动获取最新的属性期望值（由设备发起，拉模式）
 * 场景2．设备在线时接收到云端property.set推送的属性期望值（由云端发起，推模式）
 * @param identifier  属性标识符
 * @param value       期望属性值
 * @param needReport  是否通过property.post发送状态上报。
 *                    上面场景2的处理函数中已集成属性上报能力，会将needReport设置为false
 * @return
 */
private boolean handlePropertySet(String identifier, ValueWrapper value, boolean needRe
port) {
    ALog.d(TAG, "真实设备处理属性变更 = [" + identifier + "], value = [" + value + "]");
    // 用户根据实际情况判性是否设置成功 这里测试直接返回成功
    boolean success = true;
    if (needReport) {
        reportProperty(identifier, value);
    }
    return success;
}
private void reportProperty(String identifier, ValueWrapper value){
    if (StringUtils.isEmptyString(identifier) || value == null) {
        return;
    }
    ALog.d(TAG, "上报 属性identity=" + identifier);
    Map<String, ValueWrapper> reportData = new HashMap<>();
    reportData.put(identifier, value);
    LinkKit.getInstance().getDeviceThing().thingPropertyPost(reportData, new IPublishRe
sourceListener() {
        public void onSuccess(String s, Object o) {
            // 属性上报成功
            ALog.d(TAG, "上报成功 onSuccess() called with: s = [" + s + "], o = [" + o +
"]");
        }
        public void onError(String s, AError aError) {
            // 属性上报失败
            ALog.d(TAG, "上报失败onError() called with: s = [" + s + "], aError = [" + J
SON.toJSONString(aError) + "]");
        }
    });
}
```

3. 灯泡在线时，如果云端设置了灯泡的期望属性值，该值将被推送到设备端。灯泡处理消息，改变属性状态。

   如下代码中，将调用 `connectNotifyListener` 处理消息，相关Alink协议，请参见设备上报属性。

   收到异步下行的数据后， `mCommonHandler` 被调用，进而调用 `handlePropertySet` 更新设备的物理属性。

```
/**
 * 注册服务调用（以及属性设置）的响应函数。
 * 云端调用设备的某项服务的时候，设备端需要响应该服务并回复。
 */
public void connectNotifyListener() {
```

```
    List<Service> serviceList = LinkKit.getInstance().getDeviceThing().getServices();
    for (int i = 0; serviceList != null && i < serviceList.size(); i++) {
        Service service = serviceList.get(i);
        LinkKit.getInstance().getDeviceThing().setServiceHandler(service.getIdentifier(
), mCommonHandler);
    }
}
private ITResRequestHandler mCommonHandler = new ITResRequestHandler() {
    public void onProcess(String serviceIdentifier, Object result, ITResResponseCallbac
k itResResponseCallback) {
        ALog.d(TAG, "onProcess() called with: s = [" + serviceIdentifier + "]," +
                " o = [" + result + "], itResResponseCallback = [" + itResResponseCallb
ack + "]");
        ALog.d(TAG, "收到云端异步服务调用 " + serviceIdentifier);
        try {
            if (SERVICE_SET.equals(serviceIdentifier)) {
                Map<String, ValueWrapper> data = (Map<String, ValueWrapper>)((InputPara
ms)result).getData();
                ALog.d(TAG, "收到异步下行数据 " + data);
                // 设置真实设备的属性，然后上报设置完成的属性值
                boolean isSetPropertySuccess =
                        handlePropertySet("LightStatus", data.get("LightStatus"), false
);
                if (isSetPropertySuccess) {
                    if (result instanceof InputParams) {
                        // 响应云端 接收数据成功
                        itResResponseCallback.onComplete(serviceIdentifier, null, null)
;
                    } else {
                        itResResponseCallback.onComplete(serviceIdentifier, null, null)
;
                    }
                } else {
                    AError error = new AError();
                    error.setCode(100);
                    error.setMsg("setPropertyFailed.");
                    itResResponseCallback.onComplete(serviceIdentifier, new ErrorInfo(e
rror), null);
                }
            } else if (SERVICE_GET.equals(serviceIdentifier)) {
            } else {
                // 根据不同的服务做不同的处理，跟具体的服务有关系
                ALog.d(TAG, "根据真实的服务返回服务的值，请参照set示例");
                OutputParams outputParams = new OutputParams();
                // outputParams.put("op", new ValueWrapper.IntValueWrapper(20));
                itResResponseCallback.onComplete(serviceIdentifier, null, outputParams)
;
            }
        } catch (Exception e) {
            e.printStackTrace();
            ALog.d(TAG, "云端返回数据格式异常");
        }
    }
    public void onSuccess(Object o, OutputParams outputParams) {
```

```
        ALog.d(TAG, "onSuccess() called with: o = [" + o + "], outputParams = [" + outp
utParams + "]");
        ALog.d(TAG, "注册服务成功");
    }
    public void onFail(Object o, ErrorInfo errorInfo) {
        ALog.d(TAG, "onFail() called with: o = [" + o + "], errorInfo = [" + errorInfo
+ "]");
        ALog.d(TAG, "注册服务失败");
    }
};
```

4. 灯泡离线后，如果云端设置了灯的期望属性值，该值将被存储在云端。

   灯泡上线后，会主动获取期望属性值，然后调用 `handlePropertySet` 更新实际设备的属性。

```
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
    public void onError(AError aError) {
        ALog.e(TAG, "Init Error error=" + aError);
    }
    public void onInitDone(InitResult initResult) {
        ALog.i(TAG, "onInitDone result=" + initResult);
        connectNotifyListener();
        // 获取云端最新期望属性值
        getDesiredProperty(deviceInfo, Arrays.asList("LightStatus"), new IConnectSendLi
stener() {
            public void onResponse(ARequest aRequest, AResponse aResponse) {
                if(aRequest instanceof MqttPublishRequest && aResponse.data != null) {
                    JSONObject jsonObject = JSONObject.parseObject(aResponse.data.toStr
ing());
                    ALog.i(TAG, "onResponse result=" + jsonObject);
                    JSONObject dataObj = jsonObject.getJSONObject("data");
                    if (dataObj != null) {
                        if (dataObj.getJSONObject("LightStatus") == null) {
                            // 未设置期望值
                        } else {
                            Integer value = dataObj.getJSONObject("LightStatus").getInt
eger("value");
                            handlePropertySet("LightStatus", new ValueWrapper.IntValueW
rapper(value), true);
                        }
                    }
                }
            }
            public void onFailure(ARequest aRequest, AError aError) {
                ALog.d(TAG, "onFailure() called with: aRequest = [" + aRequest + "], aE
rror = [" + aError + "]");
            }
        });
    }
});
private void getDesiredProperty(BaseInfo info, List<String> properties, IConnectSendLis
tener listener) {
    ALog.d(TAG, "getDesiredProperty() called with: info = [" + info + "], listener = ["
+ listener + "]");
    if(info != null && !StringUtils.isEmptyString(info.productKey) && !StringUtils.isEm
```

```
ptyString(info.deviceName)) {
        MqttPublishRequest request = new MqttPublishRequest();
        request.topic = DESIRED_PROPERTY_GET.replace("{productKey}", info.productKey).r
eplace("{deviceName}", info.deviceName);
        request.replyTopic = DESIRED_PROPERTY_GET_REPLY.replace("{productKey}", info.pr
oductKey).replace("{deviceName}", info.deviceName);
        request.isRPC = true;
        RequestModel<List<String>> model = new RequestModel<>();
        model.id = String.valueOf(IDGeneraterUtils.getId());
        model.method = METHOD_GET_DESIRED_PROPERTY;
        model.params = properties;
        model.version = "1.0";
        request.payloadObj = model.toString();
        ALog.d(TAG, "getDesiredProperty: payloadObj=" + request.payloadObj);
        ConnectSDK.getInstance().send(request, listener);
    } else {
        ALog.w(TAG, "getDesiredProperty failed, baseInfo Empty.");
        if(listener != null) {
            AError error = new AError();
            error.setMsg("BaseInfoEmpty.");
            listener.onFailure(null, error);
        }
    }
}
```

验证结果

根据以下场景运行代码，验证灯泡在线、离线状态，可在云端通过设置期望属性值，成功更改设备属性值。

- 设备在线时，云端修改灯泡开关状态，灯泡实时响应状态变化。

- 设备离线后，如果云端修改灯泡开关状态，云端期望属性值与设备的最新属性值不一致。

- 设备重新连网上线后，设备主动拉取期望属性值，设备的最新属性值实现与云端期望属性值的同步。

附录：设备端Demo代码

```
package com.aliyun.alink.devicesdk.demo;
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.aliyun.alink.apiclient.utils.StringUtils;
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.RequestModel;
import com.aliyun.alink.dm.utils.IDGeneraterUtils;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.cmp.api.ConnectSDK;
import com.aliyun.alink.linksdk.cmp.connect.channel.MqttPublishRequest;
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.IConnectSendListener;
import com.aliyun.alink.linksdk.tmp.api.InputParams;
import com.aliyun.alink.linksdk.tmp.api.OutputParams;
import com.aliyun.alink.linksdk.tmp.device.payload.ValueWrapper;
import com.aliyun.alink.linksdk.tmp.devicemodel.Service;
import com.aliyun.alink.linksdk.tmp.listener.IPublishResourceListener;
import com.aliyun.alink.linksdk.tmp.listener.ITResRequestHandler;
import com.aliyun.alink.linksdk.tmp.listener.ITResResponseCallback;
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_SET = "set";
    private final static String SERVICE_GET = "get";
    public static String DESIRED_PROPERTY_GET = "/sys/{productKey}/{deviceName}/thing/prope
rty/desired/get";
    public static String DESIRED_PROPERTY_GET_REPLY = "/sys/{productKey}/{deviceName}/thing
/property/desired/get_reply";
    public static String METHOD_GET_DESIRED_PROPERTY = "thing.property.desired.get";
    public static void main(String[] args) {
        /**
         * 设备证书信息
         */
        String productKey = "****";
        String deviceName = "Lamp";
        String deviceSecret = "****";
        /**
         * mqtt连接信息
         */
        String regionId = "cn-shanghai";
        LampDemo manager = new LampDemo();
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.productKey = productKey;
        deviceInfo.deviceName = deviceName;
        deviceInfo.deviceSecret = deviceSecret;
        manager.init(deviceInfo, regionId);
    }
    public void init(final DeviceInfo deviceInfo, String region) {
        LinkKitInitParams params = new LinkKitInitParams();
```

```
        /**
         * 设置 Mqtt 初始化参数
         */
        IoTMqttClientConfig config = new IoTMqttClientConfig();
        config.productKey = deviceInfo.productKey;
        config.deviceName = deviceInfo.deviceName;
        config.deviceSecret = deviceInfo.deviceSecret;
        config.channelHost = deviceInfo.productKey + ".iot-as-mqtt." + region + ".aliyuncs.
com:1883";
        /**
         * 是否接受离线消息
         * 对应 mqtt 的 cleanSession 字段
         */
        config.receiveOfflineMsg = false;
        params.mqttClientConfig = config;
        /**
         * 设置初始化，传入设备证书信息
         */
        params.deviceInfo = deviceInfo;
        LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
            public void onError(AError aError) {
                ALog.e(TAG, "Init Error error=" + aError);
            }
            public void onInitDone(InitResult initResult) {
                ALog.i(TAG, "onInitDone result=" + initResult);
                connectNotifyListener();
                // 获取云端最新期望属性值
                getDesiredProperty(deviceInfo, Arrays.asList("LightStatus"), new IConnectSe
ndListener() {
                    public void onResponse(ARequest aRequest, AResponse aResponse) {
                        if(aRequest instanceof MqttPublishRequest && aResponse.data != null
) {
                            JSONObject jsonObject = JSONObject.parseObject(aResponse.data.t
oString());
                            ALog.i(TAG, "onResponse result=" + jsonObject);
                            JSONObject dataObj = jsonObject.getJSONObject("data");
                            if (dataObj != null) {
                                if (dataObj.getJSONObject("LightStatus") == null) {
                                    // 未设置期望值
                                } else {
                                    Integer value = dataObj.getJSONObject("LightStatus").ge
tInteger("value");
                                    handlePropertySet("LightStatus", new ValueWrapper.IntVa
lueWrapper(value), true);
                                }
                            }
                        }
                    }
                    public void onFailure(ARequest aRequest, AError aError) {
                        ALog.d(TAG, "onFailure() called with: aRequest = [" + aRequest + "]
, aError = [" + aError + "]");
                    }
                });
            }
```

```
        });
    }
    /**
     * 真实设备处理属性变更，两个场景下会被调用：
     * 场景1. 设备联网后主动获取最新的属性期望值（由设备发起，拉模式）
     * 场景2. 设备在线时接收到云端property.set推送（由云端发起，推模式）
     * @param identifier  属性标识符
     * @param value       期望属性值
     * @param needReport  是否发送property.post状态上报。
     *                    上面场景2的处理函数中已集成属性上报能力，会将needReport设置为false
     * @return
     */
    private boolean handlePropertySet(String identifier, ValueWrapper value, boolean needRe
port) {
        ALog.d(TAG, "真实设备处理属性变更 = [" + identifier + "], value = [" + value + "]");
        // 用户根据实际情况判断属性是否设置成功 这里测试直接返回成功
        boolean success = true;
        if (needReport) {
            reportProperty(identifier, value);
        }
        return success;
    }
    private void reportProperty(String identifier, ValueWrapper value){
        if (StringUtils.isEmptyString(identifier) || value == null) {
            return;
        }
        ALog.d(TAG, "上报 属性identity=" + identifier);
        Map<String, ValueWrapper> reportData = new HashMap<>();
        reportData.put(identifier, value);
        LinkKit.getInstance().getDeviceThing().thingPropertyPost(reportData, new IPublishRe
sourceListener() {
            public void onSuccess(String s, Object o) {
                // 属性上报成功
                ALog.d(TAG, "上报成功 onSuccess() called with: s = [" + s + "], o = [" + o +
"]");
            }
            public void onError(String s, AError aError) {
                // 属性上报失败
                ALog.d(TAG, "上报失败onError() called with: s = [" + s + "], aError = [" + J
SON.toJSONString(aError) + "]");
            }
        });
    }
    /**
     * 注册服务调用（以及属性设置）的响应函数。
     * 云端调用设备的某项服务的时候，设备端需要响应该服务并回复。
     */
    public void connectNotifyListener() {
        List<Service> serviceList = LinkKit.getInstance().getDeviceThing().getServices();
        for (int i = 0; serviceList != null && i < serviceList.size(); i++) {
            Service service = serviceList.get(i);
            LinkKit.getInstance().getDeviceThing().setServiceHandler(service.getIdentifier(
), mCommonHandler);
        }
```

```
    }
    private ITResRequestHandler mCommonHandler = new ITResRequestHandler() {
        public void onProcess(String serviceIdentifier, Object result, ITResResponseCallbac
k itResResponseCallback) {
            ALog.d(TAG, "onProcess() called with: s = [" + serviceIdentifier + "]," +
                    " o = [" + result + "], itResResponseCallback = [" + itResResponseCallb
ack + "]");
            ALog.d(TAG, "收到云端异步服务调用 " + serviceIdentifier);
            try {
                if (SERVICE_SET.equals(serviceIdentifier)) {
                    Map<String, ValueWrapper> data = (Map<String, ValueWrapper>)((InputPara
ms)result).getData();
                    ALog.d(TAG, "收到异步下行数据 " + data);
                    // 设置真实设备的属性，然后上报设置完成的属性值
                    boolean isSetPropertySuccess =
                            handlePropertySet("LightStatus", data.get("LightStatus"), false
);
                    if (isSetPropertySuccess) {
                        if (result instanceof InputParams) {
                            // 响应云端 接收数据成功
                            itResResponseCallback.onComplete(serviceIdentifier, null, null)
;
                        } else {
                            itResResponseCallback.onComplete(serviceIdentifier, null, null)
;
                        }
                    } else {
                        AError error = new AError();
                        error.setCode(100);
                        error.setMsg("setPropertyFailed.");
                        itResResponseCallback.onComplete(serviceIdentifier, new ErrorInfo(e
rror), null);
                    }
                } else if (SERVICE_GET.equals(serviceIdentifier)) {
                } else {
                    // 根据不同的服务做不同的处理，跟具体的服务有关系
                    ALog.d(TAG, "用户根据真实的服务返回服务的值，请参照set示例");
                    OutputParams outputParams = new OutputParams();
                    // outputParams.put("op", new ValueWrapper.IntValueWrapper(20));
                    itResResponseCallback.onComplete(serviceIdentifier, null, outputParams)
;
                }
            } catch (Exception e) {
                e.printStackTrace();
                ALog.d(TAG, "云端返回数据格式异常");
            }
        }
        public void onSuccess(Object o, OutputParams outputParams) {
            ALog.d(TAG, "onSuccess() called with: o = [" + o + "], outputParams = [" + outp
utParams + "]");
            ALog.d(TAG, "注册服务成功");
        }
        public void onFail(Object o, ErrorInfo errorInfo) {
            ALog.d(TAG, "onFail() called with: o = [" + o + "], errorInfo = [" + errorInfo
+ "]");
```

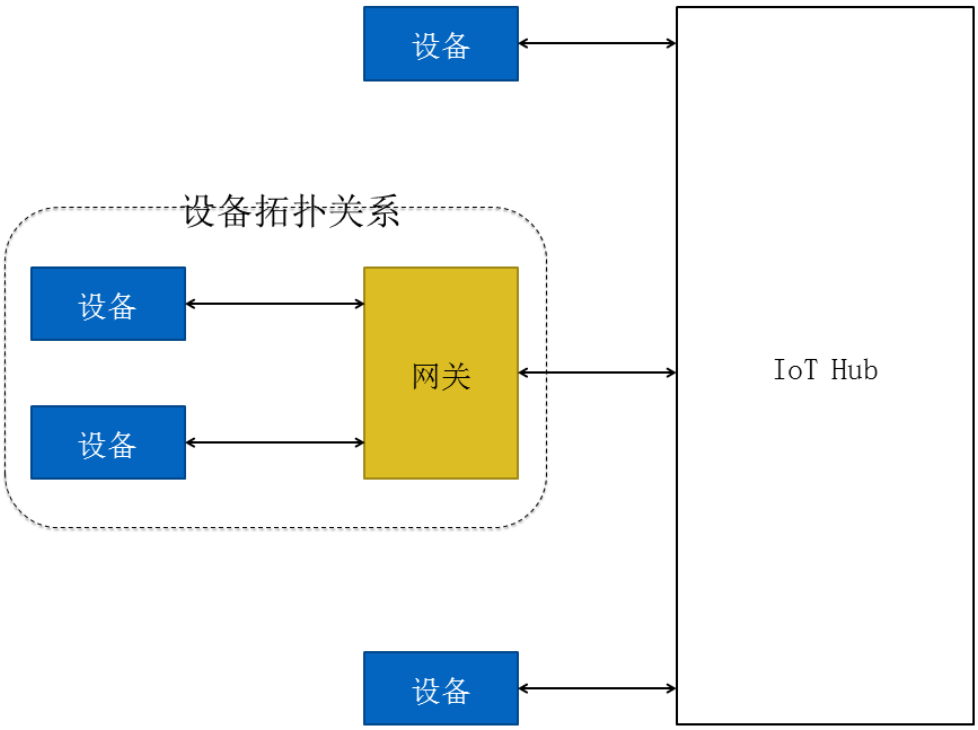
```
            ALog.d(TAG, "注册服务失败");
        }
    };
    private void getDesiredProperty(BaseInfo info, List<String> properties, IConnectSendLis
tener listener) {
        ALog.d(TAG, "getDesiredProperty() called with: info = [" + info + "], listener = ["
+ listener + "]");
        if(info != null && !StringUtils.isEmptyString(info.productKey) && !StringUtils.isEm
ptyString(info.deviceName)) {
            MqttPublishRequest request = new MqttPublishRequest();
            request.topic = DESIRED_PROPERTY_GET.replace("{productKey}", info.productKey).r
eplace("{deviceName}", info.deviceName);
            request.replyTopic = DESIRED_PROPERTY_GET_REPLY.replace("{productKey}", info.pr
oductKey).replace("{deviceName}", info.deviceName);
            request.isRPC = true;
            RequestModel<List<String>> model = new RequestModel<>();
            model.id = String.valueOf(IDGeneraterUtils.getId());
            model.method = METHOD_GET_DESIRED_PROPERTY;
            model.params = properties;
            model.version = "1.0";
            request.payloadObj = model.toString();
            ALog.d(TAG, "getDesiredProperty: payloadObj=" + request.payloadObj);
            ConnectSDK.getInstance().send(request, listener);
        } else {
            ALog.w(TAG, "getDesiredProperty failed, baseInfo Empty.");
            if(listener != null) {
                AError error = new AError();
                error.setMsg("BaseInfoEmpty.");
                listener.onFailure(null, error);
            }
        }
    }
}
```

3.2. 子设备接入物联网平台

3.2.1. 概述

子设备不直接连接物联网平台，而是通过网关接入物联网平台。本示例介绍如何实现子设备通过网关接入物联网平台。

首先，需在物联网平台上创建网关和子设备；然后，开发网关设备端SDK，实现网关直连物联网平台；再由网关向物联网平台上报网关与子设备的拓扑关系；通过网关上报子设备证书（一机一密方式）或者子设备动态注册的认证方式，物联网平台校验子设备的身份和该子设备与网关的拓扑关系。所有校验通过，才会建立子设备逻辑通道，并绑定至网关物理通道上，实现子设备通过网关，与物联网平台建立连接，并进行通信。



本示例仅介绍子设备通过网关接入物联网平台，不涉及物模型数据通信等通信配置。如果您的子设备与物联网平台之间，需进行物模型数据通信，请配置网关代理子设备进行物模型数据通信。配置方法，请参见物模型通信。

> **说明** 网关如何与子设备进行数据传递，需网关厂商自行实现，阿里云不提供代码支持。

相关连接

本示例使用的SDK代码Demo：iotx-api-demo。

实现过程说明：

1. 创建网关和子设备
2. 初始化SDK
3. 网关接入物联网平台
4. 子设备接入物联网平台

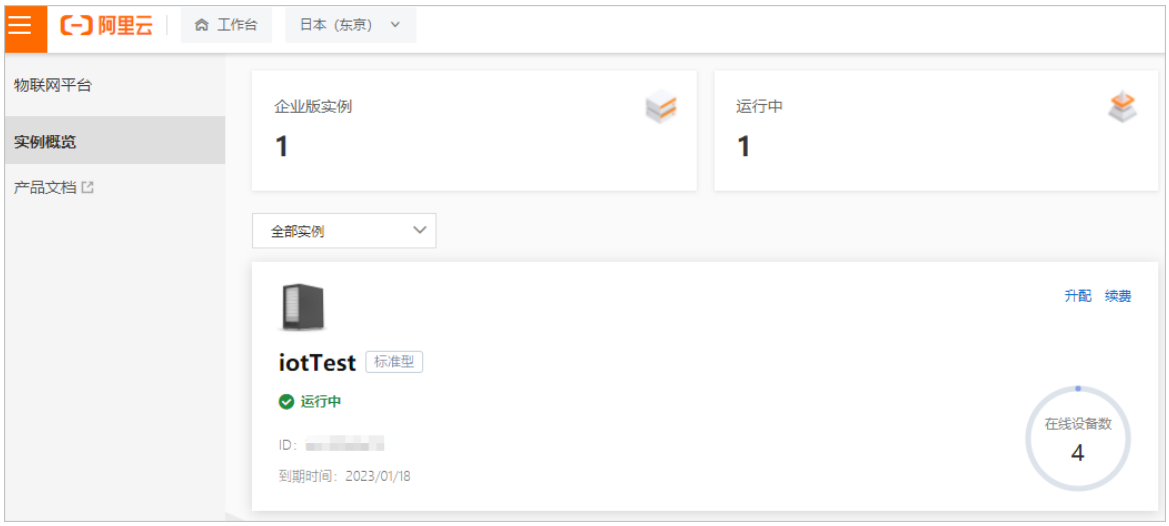
3.2.2. 创建网关和子设备

首先，在物联网平台控制台，分别创建网关产品和设备、子设备所属产品和子设备。

操作步骤

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏，单击**设备管理 > 产品**。
4. 单击**创建产品**，创建两个产品，分别是网关产品和子设备产品。
   - 创建网关产品，具体操作请参见创建产品。

     > 说明 网关产品的**节点类型**需选择为**网关设备**。

   - 创建子设备产品，具体操作请参见创建产品。

     > 说明 **节点类型**需选择为**网关子设备**。

5. 在左侧导航栏，单击**设备**。在**设备**页，分别在网关产品和子设备产品下，创建网关设备和子设备，具体操作请参见创建设备。

后续步骤

初始化SDK

网关接入物联网平台

子设备接入物联网平台

3.2.3. 初始化SDK

本示例中，提供云端和设备端 Java SDK Demo。您需准备Java开发环境，下载SDK Demo，导入项目和初始化SDK。

前提条件

创建网关和子设备

操作步骤

1. 单击下载iotx-api-demo，并解压缩。

SDK Demo中包含了服务端SDK Demo和设备端SDK Demo。

2. 打开Java开发工具，导入解压缩后的iotx-api-demo文件夹。
3. 在*pom.xml*文件中，添加以下依赖，导入阿里云云端SDK和设备端SDK。

```
<!-- https://mvnrepository.com/artifact/com.aliyun/aliyun-java-sdk-iot -->
<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>
<dependency>
  <groupId>com.aliyun.alink.linksdk</groupId>
  <artifactId>iot-linkkit-java</artifactId>
  <version>1.2.0.1</version>
  <scope>compile</scope>
</dependency>
```

4. 在*java/src/main/resources/*目录下的*config*文件中，填入初始化信息。

```
user.accessKeyID = <your accessKey ID>
user.accessKeySecret = <your accessKey Secret>
iot.regionId = <regionId>
iot.productCode = Iot
iot.domain = iot.<regionId>.aliyuncs.com
iot.version = 2018-01-20
```

参数	说明
accessKeyID	您的阿里云账号的AccessKey ID。 将光标定位到您的账号头像上，选择**AccessKey管理**，进入**安全信息管理**页，可创建或查看您的AccessKey。
accessKeySecret	您的阿里云账号的AccessKey Secret。查看方法同上AccessKey ID。
regionId	您的物联网设备所属地域ID。地域ID的表达方法，请参见地域和可用区。

后续步骤

网关接入物联网平台

子设备接入物联网平台

3.2.4. 网关接入物联网平台

配置设备端SDK后，可以实现网关设备连接物联网平台。

前提条件

您已完成以下操作：

- 创建网关和子设备
- 初始化SDK

配置网关设备端SDK

本示例Demo中，*java/src/main/java/com/aliyun/iot/api/common/deviceApi*目录下的*DeviceTopoManager*文件中包含网关接入物联网平台的代码示例。

1. 设置网关连接信息。

```
private static String regionId = "cn-shanghai";
private static final String TAG = "TOPO";
//网关设备。
private static String GWproductKey = "a1BxptK***";
private static String GWdeviceName = "XMtrv3yvftEHAzrTfX1U";
private static String GWdeviceSecret = "19xJNybifnmgcK057vYhazYK4b64****";
public static void main(String[] args) {
    /**
     * mqtt连接信息。
     */
    DeviceTopoManager manager = new DeviceTopoManager();
    /**
     * 服务器端的java http客户端使用TSLv1.2。
     */
    System.setProperty("https.protocols", "TLSv2");
    manager.init();
}
```

2. 建立连接。

```
public void init() {
        LinkKitInitParams params = new LinkKitInitParams();
        /**
         * 设置mqtt初始化参数。
         */
        IoTMqttClientConfig config = new IoTMqttClientConfig();
        config.productKey = GWproductKey;
        config.deviceName = GWdeviceName;
        config.deviceSecret = GWdeviceSecret;
        config.channelHost = GWproductKey + ".iot-as-mqtt." + regionId + ".aliyuncs.com
:1883";
        /**
         * 是否接受离线消息。
         * 对应mqtt的cleanSession字段。
         */
        config.receiveOfflineMsg = false;
        params.mqttClientConfig = config;
        ALog.setLevel(LEVEL_DEBUG);
        ALog.i(TAG, "mqtt connetcion info=" + params);
        /**
         * 设置初始化，传入设备证书信息。
         */
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.productKey = GWproductKey;
        deviceInfo.deviceName = GWdeviceName;
        deviceInfo.deviceSecret = GWdeviceSecret;
        params.deviceInfo = deviceInfo;
        /**建立连接。**/
        LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
            public void onError(AError aError) {
                ALog.e(TAG, "Init Error error=" + aError);
            }
            public void onInitDone(InitResult initResult) {
                ALog.i(TAG, "onInitDone result=" + initResult);
                //获取网关下topo关系，查询网关与子设备是否已经存在topo关系。
                //如果已经存在，则直接上线子设备。
                getGWDeviceTopo();
                //子设备动态注册获取设备deviceSecret，如果设备已知设备证书则忽略此步，直接添加t
opo关系。
                //预注册设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
                gatewaySubDevicRegister();
                //待添加拓扑关系的子设备信息。
                gatewayAddSubDevice();
            }
        });
    }
```

测试

在设备端SDK中，配置完网关设备信息后，测试SDK是否能连接物联网平台。

1. 运行DeviceTopoManager。
2. 在物联网平台控制台左侧导航栏中，选择**设备管理 > 设备**。

3. 在设备列表中，找到网关设备，查看其状态。如果状态显示为**在线**，说明网关设备已连接物联网平台。

后续步骤

子设备接入物联网平台

3.2.5. 子设备接入物联网平台

子设备不直接连接物联网平台，需要通过网关与物联网平台连接。子设备连接网关后，网关查询与当前子设备间的拓扑关系，将子设备的信息上报物联网平台，代理子设备接入物联网平台。

前提条件

您已完成以下操作：

- 创建网关和子设备
- 初始化SDK
- 网关接入物联网平台

背景信息

- 开发子设备

  由于子设备不直接连接物联网平台，所以无需为子设备安装物联网平台设备端SDK。子设备的设备端由厂商自行开发。

- 本示例Demo

  *java/src/main/java/com/aliyun/iot/api/common/deviceApi*目录下的*DeviceTopoManager*文件中包含网关管理拓扑关系、获取子设备证书和子设备上线的代码。

步骤一：网关管理拓扑关系

网关接入物联网平台后，需将拓扑关系同步至物联网平台，才能代理子设备与物联网平台通信。您可以直接在控制台查看、添加拓扑关系，也可以使用示例代码完成这一步。

- 在物联网平台控制台下查看、添加网关与子设备的拓扑关系。
  i. 在左侧导航栏，选择**设备管理 > 设备**，在列表中找到网关设备。
  ii. 单击网关设备对应的**子设备**，进入子设备管理页面。查看网关产品下的子设备信息。
  iii. 单击**添加子设备**，将创建网关和子设备步骤中的子设备添加到网关下。
- 通过以下示例代码查询、添加拓扑关系。

- 查询拓扑关系：

```
     /**
     * 获取网关下topo关系，查询网关与子设备是否已经存在topo关系。
     */
    private void getGWDeviceTopo() {
        LinkKit.getInstance().getGateway().gatewayGetSubDevices(new IConnectSendListene
r() {
            @Override
            public void onResponse(ARequest request, AResponse aResponse) {
                ALog.i(TAG, "获取网关的topo关系成功 : " + JSONObject.toJSONString(aRespon
se));
                // 获取子设备列表结果。
                try {
                    ResponseModel<List<DeviceInfo>> response = JSONObject.parseObject(a
Response.data.toString(), new TypeReference<ResponseModel<List<DeviceInfo>>>() {
                    }.getType());
                    // TODO，根据实际应用场景处理。
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            @Override
            public void onFailure(ARequest request, AError error) {
                ALog.i(TAG, "获取网关的topo关系失败 : " + JSONObject.toJSONString(error))
;
            }
        });
    }
```

- 添加拓扑关系：

> **说明**
>
> - 子设备证书信息的获取方法请参见下一步。
> - 物联网平台系统确认子设备和网关的拓扑关系后，子设备便可上线，复用网关的物理通道与物联网平台进行通信。

```
    /**
     * 待添加拓扑关系的子设备信息。
     */
    private void gatewayAddSubDevice() {
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7SyR****";
        baseInfo1.deviceName = "safa***";
        String deviceSecret = "7lzCJIWHmGFpZpDKbJdVucDHUz6C****";
        LinkKit.getInstance().getGateway().gatewayAddSubDevice(baseInfo1, new ISubDevic
eConnectListener() {
            @Override
            public String getSignMethod() {
                // 使用的签名方法。
                return "hmacsha1";
            }
            @Override
```

```
            @Override
            public String getSignValue() {
                // 获取签名，用户使用deviceSecret获得签名结果。
                Map<String, String> signMap = new HashMap<>();
                signMap.put("productKey", baseInfo1.productKey);
                signMap.put("deviceName", baseInfo1.deviceName);
                //signMap.put("timestamp", String.valueOf(System.currentTimeMillis()));
                signMap.put("clientId", getClientId());
                return SignUtils.hmacSign(signMap, deviceSecret);
            }
            @Override
            public String getClientId() {
                // clientId可为任意值。
                return "id";
            }
            @Override
            public Map<String, Object> getSignExtraData() {
                return null;
            }
            @Override
            public void onConnectResult(boolean isSuccess, ISubDeviceChannel iSubDevice
Channel, AError aError) {
                // 添加结果
                if (isSuccess) {
                    // 子设备添加成功，接下来可以做子设备上线的逻辑
                    ALog.i(TAG, "topo关系添加成功 : " + JSONObject.toJSONString(iSubDevi
ceChannel));
                    //子设备上线
                    gatewaySubDeviceLogin();
                } else {
                    ALog.i(TAG, "topo关系添加失败 : " + JSONObject.toJSONString(aError))
;
                }
            }
            @Override
            public void onDataPush(String s, AMessage aMessage) {
            }
        });
    }
```

步骤二：获取子设备证书

子设备创建成功后，物联网平台会颁发设备证书。网关可通过以下方法，获取子设备证书信息。

- 使用一机一密的认证方式。

  在设备创建成功后，在控制台的设备详情页面，获取ProductKey、DeviceName和DeviceSecret。

  - 在网关与子设备之间定义协议，实现网关发现子设备，获取子设备的设备证书。该协议由网关厂商与子设备厂商自行定义。
  - 网关厂商可以在网关上提供某种配置方式，预置子设备的证书信息。该功能由网关厂商自行实现。

- 使用子设备动态注册的方式。

  由网关向物联网平台上报子设备的ProductKey和DeviceName进行注册。物联网平台校验子设备ProductKey和DeviceName通过后，动态下发子设备的DeviceSecret。

i. 创建子设备时，以设备的SN码或MAC地址作为DeviceName。设备创建成功后，开启产品的动态注册功能。

ii. 开发网关时，实现网关通过某种协议发现子设备，获取子设备的型号（model）和唯一标识（SN码或MAC地址）；并实现子设备型号（model）与阿里云物联网平台ProductKey的映射。

iii. 通过物联网平台的动态注册功能，从物联网平台获取子设备的DeviceSecret。

代码示例：

```
    /**
     * 子设备动态注册获取设备deviceSecret。
     * 在物联网平台上提前创建子设备时，可以使用子设备的MAC地址或SN序列号等作为DeviceName。
     */
    private void gatewaySubDevicRegister() {
        List<BaseInfo> subDevices = new ArrayList<>();
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7SyR***";
        baseInfo1.deviceName = "safasdf";
        subDevices.add(baseInfo1);
        LinkKit.getInstance().getGateway().gatewaySubDevicRegister(subDevices, new IConnectSendListener() {
            @Override
            public void onResponse(ARequest request, AResponse response) {
                ALog.i(TAG, "子设备注册成功 : " + JSONObject.toJSONString(response));
            }
            @Override
            public void onFailure(ARequest request, AError error) {
                ALog.i(TAG, "子设备注册失败 : " + JSONObject.toJSONString(error));
            }
        });
    }
```

关于设备动态注册的详细说明，请参见子设备动态注册。

步骤三：子设备上线

```
    /**
     * 调用子设备上线接口之前，请确保已建立topo关系。网关发现子设备连接之后，需要告知物联网平台子设备上线。
     * 子设备上线之后可以执行子设备的订阅、发布等操作。
     */
    public void gatewaySubDeviceLogin(){
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7SyR****";
        baseInfo1.deviceName = "safasdf";
        LinkKit.getInstance().getGateway().gatewaySubDeviceLogin(baseInfo1, new ISubDeviceActionListener() {
            @Override
            public void onSuccess() {
                // 代理子设备上线成功。
                // 上线之后可订阅、发布消息，并可以删除和禁用子设备。
                // subDevDisable(null);
                // subDevDelete(null);
            }
            @Override
            public void onFailed(AError aError) {
                ALog.d(TAG, "onFailed() called with: aError = [" + aError + "]");
            }
        });
    }
}
```

附录：代码Demo

由网关发现并上报子设备信息，建立子设备与物联网平台的逻辑通道，及子设备复用网关物理通道接入物联网平台的完整示例代码如下：

```
package com.aliyun.iot.api.common.deviceApi;
import com.alibaba.fastjson.JSONObject;
import com.alibaba.fastjson.TypeReference;
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.api.SignUtils;
import com.aliyun.alink.dm.model.ResponseModel;
import com.aliyun.alink.linkkit.api.ILinkKitConnectListener;
import com.aliyun.alink.linkkit.api.IoTMqttClientConfig;
import com.aliyun.alink.linkkit.api.LinkKit;
import com.aliyun.alink.linkkit.api.LinkKitInitParams;
import com.aliyun.alink.linksdk.channel.gateway.api.subdevice.ISubDeviceActionListener;
import com.aliyun.alink.linksdk.channel.gateway.api.subdevice.ISubDeviceChannel;
import com.aliyun.alink.linksdk.channel.gateway.api.subdevice.ISubDeviceConnectListener;
import com.aliyun.alink.linksdk.channel.gateway.api.subdevice.ISubDeviceRemoveListener;
import com.aliyun.alink.linksdk.cmp.core.base.AMessage;
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.IConnectSendListener;
import com.aliyun.alink.linksdk.tools.AError;
import com.aliyun.alink.linksdk.tools.ALog;
import java.util.*;
```

```
import java.util.*;
import static com.aliyun.alink.linksdk.tools.ALog.LEVEL_DEBUG;
public class DeviceTopoManager {
    private static String regionId = "cn-shanghai";
    private static final String TAG = "TOPO";
    //网关设备
    private static String GWproductKey = "a1Bxp*********";
    private static String GWdeviceName = "XMtrv3y**************";
    private static String GWdeviceSecret = "19xJNybifnmgc**************";
    public static void main(String[] args) {
        /**
         * mqtt连接信息
         */
        DeviceTopoManager manager = new DeviceTopoManager();
        /**
         * 服务器端的java http客户端使用TSLv1.2。
         */
        System.setProperty("https.protocols", "TLSv2");
        manager.init();
    }
    public void init() {
        LinkKitInitParams params = new LinkKitInitParams();
        /**
         * 设置mqtt初始化参数。
         */
        IoTMqttClientConfig config = new IoTMqttClientConfig();
        config.productKey = GWproductKey;
        config.deviceName = GWdeviceName;
        config.deviceSecret = GWdeviceSecret;
        config.channelHost = GWproductKey + ".iot-as-mqtt." + regionId + ".aliyuncs.com:188
3";
        /**
         * 是否接受离线消息。
         * 对应mqtt的cleanSession字段。
         */
        config.receiveOfflineMsg = false;
        params.mqttClientConfig = config;
        ALog.setLevel(LEVEL_DEBUG);
        ALog.i(TAG, "mqtt connetcion info=" + params);
        /**
         * 设置初始化，传入网关的设备证书信息。
         */
        DeviceInfo deviceInfo = new DeviceInfo();
        deviceInfo.productKey = GWproductKey;
        deviceInfo.deviceName = GWdeviceName;
        deviceInfo.deviceSecret = GWdeviceSecret;
        params.deviceInfo = deviceInfo;
        /**建立连接**/
        LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
            public void onError(AError aError) {
                ALog.e(TAG, "Init Error error=" + aError);
            }
            public void onInitDone(InitResult initResult) {
                ALog.i(TAG, "onInitDone result=" + initResult);
                //获取网关下拓扑关系，查询网关与子设备是否已经存在拓扑关系。
```

```
                //如果已经存在，则直接上线子设备。
                getGWDeviceTopo();
                //子设备动态注册获取DeviceSecret，如果设备已知设备证书则忽略此步，直接添加拓扑关系。
                //在物联网平台上提前创建设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
                gatewaySubDevicRegister();
                //待添加拓扑关系的子设备信息。
                gatewayAddSubDevice();
            }
        });
    }
    /**
     * 获取网关下拓扑关系，查询网关与子设备是否已经存在拓扑关系。
     */
    private void getGWDeviceTopo() {
        LinkKit.getInstance().getGateway().gatewayGetSubDevices(new IConnectSendListener()
{
            @Override
            public void onResponse(ARequest request, AResponse aResponse) {
                ALog.i(TAG, "获取网关的topo关系成功 : " + JSONObject.toJSONString(aResponse))
;
                // 获取子设备列表结果。
                try {
                    ResponseModel<List<DeviceInfo>> response = JSONObject.parseObject(aResp
onse.data.toString(), new TypeReference<ResponseModel<List<DeviceInfo>>>() {
                    }.getType());
                    // TODO 根据实际应用场景处理。
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            @Override
            public void onFailure(ARequest request, AError error) {
                ALog.i(TAG, "获取网关的topo关系失败 : " + JSONObject.toJSONString(error));
            }
        });
    }
    /**
     * 子设备动态注册获取设备deviceSecret，如果网关已获得子设备证书则忽略此步。
     * 在物联网平台上提前创建设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
     */
    private void gatewaySubDevicRegister() {
        List<BaseInfo> subDevices = new ArrayList<>();
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7SyR**********";
        baseInfo1.deviceName = "test123*********";
        subDevices.add(baseInfo1);
        LinkKit.getInstance().getGateway().gatewaySubDevicRegister(subDevices, new IConnect
SendListener() {
            @Override
            public void onResponse(ARequest request, AResponse response) {
                ALog.i(TAG, "子设备注册成功 : " + JSONObject.toJSONString(response));
            }
            @Override
            public void onFailure(ARequest request, AError error) {
```

```
                ALog.i(TAG, "子设备注册失败 : " + JSONObject.toJSONString(error));
            }
        });
    }
    /**
     * 待添加拓扑关系的子设备信息。
     */
    private void gatewayAddSubDevice() {
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7Sy*************";
        baseInfo1.deviceName = "safasd********";
        String deviceSecret = "7lzCJIWHmGF***************";
        LinkKit.getInstance().getGateway().gatewayAddSubDevice(baseInfo1, new ISubDeviceCon
nectListener() {
            @Override
            public String getSignMethod() {
                // 使用的签名方法
                return "hmacsha1";
            }
            @Override
            public String getSignValue() {
                // 获取签名，用户使用DeviceSecret获得签名结果。
                Map<String, String> signMap = new HashMap<>();
                signMap.put("productKey", baseInfo1.productKey);
                signMap.put("deviceName", baseInfo1.deviceName);
//                signMap.put("timestamp", String.valueOf(System.currentTimeMillis()));
                signMap.put("clientId", getClientId());
                return SignUtils.hmacSign(signMap, deviceSecret);
            }
            @Override
            public String getClientId() {
                // clientId可为任意值。
                return "id";
            }
            @Override
            public Map<String, Object> getSignExtraData() {
                return null;
            }
            @Override
            public void onConnectResult(boolean isSuccess, ISubDeviceChannel iSubDeviceChan
nel, AError aError) {
                // 添加结果
                if (isSuccess) {
                    // 子设备添加成功，接下来可以做子设备上线的逻辑。
                    ALog.i(TAG, "topo关系添加成功 : " + JSONObject.toJSONString(iSubDeviceCha
nnel));
                    //子设备上线
                    gatewaySubDeviceLogin();
                } else {
                    ALog.i(TAG, "topo关系添加失败 : " + JSONObject.toJSONString(aError));
                }
            }
            @Override
            public void onDataPush(String s, AMessage aMessage) {
```

```
            }
        });
    }
    public void  gatewayDeleteSubDevice(){
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7S**************";
        baseInfo1.deviceName = "saf*********";
        LinkKit.getInstance().getGateway().gatewayDeleteSubDevice(baseInfo1, new ISubDevice
RemoveListener() {
            @Override
            public void onSuceess() {
                // 成功删除子设备。删除之前可先做下线操作。
            }
            @Override
            public void onFailed(AError aError) {
                // 删除子设备失败。
            }
        });
    }
    /**
     * 调用子设备上线之前，请确保已建立拓扑关系。网关发现子设备连接后，需要告知物联网平台子设备上线。
     * 子设备上线之后可以执行子设备的订阅、发布等操作。
     */
    public void gatewaySubDeviceLogin(){
        BaseInfo baseInfo1 = new BaseInfo();
        baseInfo1.productKey = "a1j7SyR***********";
        baseInfo1.deviceName = "safa*********";
        LinkKit.getInstance().getGateway().gatewaySubDeviceLogin(baseInfo1, new ISubDeviceA
ctionListener() {
            @Override
            public void onSuccess() {
                // 代理子设备上线成功。
                // 上线之后可订阅、发布消息，并可以删除和禁用子设备。
                // subDevDisable(null);
                // subDevDelete(null);
            }
            @Override
            public void onFailed(AError aError) {
                ALog.d(TAG, "onFailed() called with: aError = [" + aError + "]");
            }
        });
    }
}
```

4.监控运维

4.1. IoT资源监控

4.1.1. 概述

物联网平台支持使用云监控进行事件监控报警和阈值监控报警。事件监控报警基于物联网平台系统限流进行监控和报警；阈值监控报警基于您设置的业务指标数值进行监控和报警。

背景信息

示例场景

某危险品仓储区共布置了5,000个智能温度感应设备。这些设备基于MQTT协议与物联网平台通信。温度每变化0.1℃，温度感应设备将当前温度值发送到云端。根据已配置的数据流转规则，当温度超过25℃后，物联网平台会将温度值流转至函数计算（Function Compute）中进行后续处理。

温度对于仓储区的安全非常重要，控制室需实时获取各检测点的温度变化，并及时作出响应。因此要求温度感应器设备一直在线。如果出现大量设备同时不在线，或设备不断重复上、下线的情况，需对设备和网络进行检修；如果温度在短时间内迅速上升，则可能是出现了安全隐患，需做应急处理。

根据业务需要，需保证各个检测点的温度感应器实时在线，感应器上报的数据都能顺利到达函数计算，和当前账号下物联网平台资源使用情况，以保证不会因为触发了限流，而导致无法及时收到温度数据。

方案设计

使用阿里云云监控监控物联网平台资源，需先在云监控控制台，创建物联网平台相关的报警规则。

需设置以下事件报警：

- 任一设备每分钟最大连接请求数达到上限
- 当前账号每秒最大连接请求数达到上限
- 当前账号每秒发布请求数达到上限
- 任一设备上行消息QPS达到上限
- 当前账号每秒到达规则引擎的请求数达到上限

需设置以下阈值报警：

- 实时在线设备数（MQTT）
- 设备属性上报失败数
- 规则引擎消息流转次数（FC）

规则配置步骤

创建产品和设备

设置报警联系人

创建事件报警规则

创建阈值报警规则

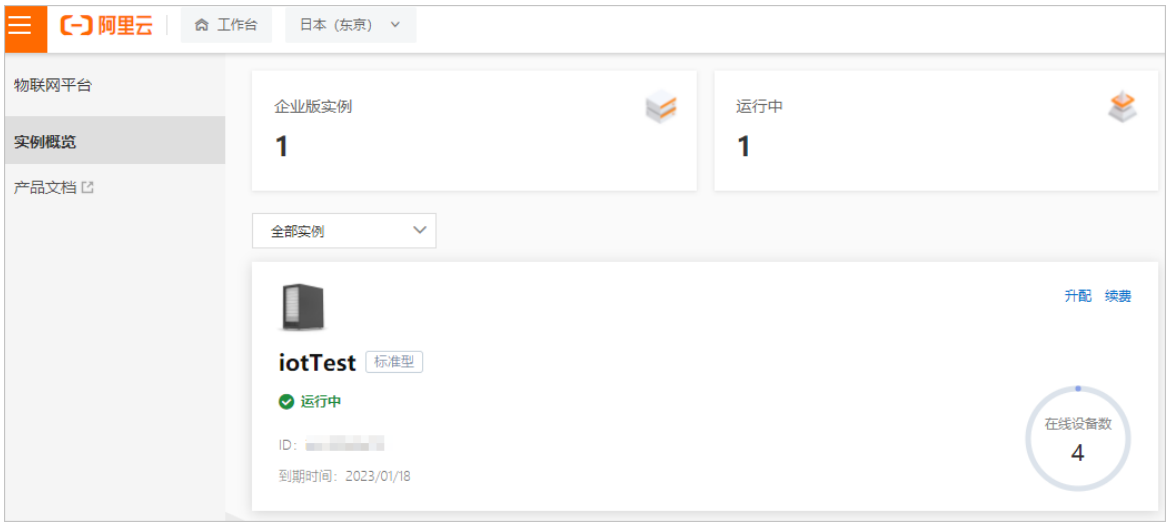
4.1.2. 创建产品和设备

本实践案例以监控建温度感应器为例，因此，您需要先在物联网平台创建温度感应器产品和设备、定义物模型、创建数据流转规则。

操作步骤

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

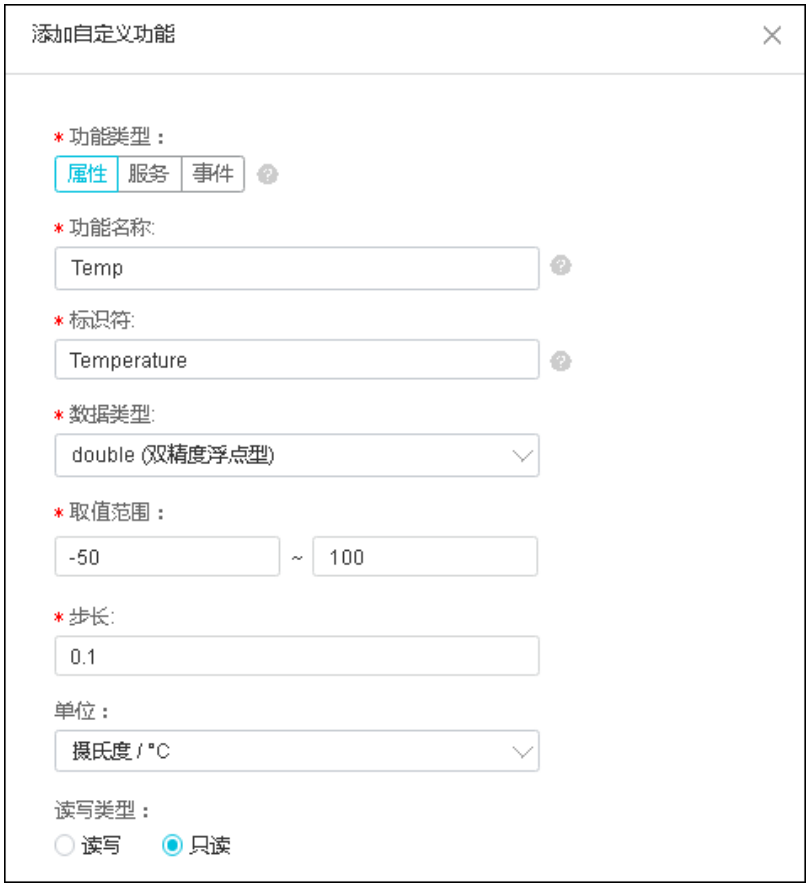
> 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



3. 在左侧导航栏中，选择**设备管理 > 产品**，单击**创建产品**，创建温度感应器产品。具体操作指导，请参见创建产品。

4. 定义功能。

i. 在新建产品的**产品详情**页的**功能定义**页签下，选择**编辑草稿 > 添加自定义功能**，为产品定义温度属性。



ii. 功能添加完成后，单击**发布上线**发布该物模型。

5. 在左侧导航栏中，选择**设备**，单击**批量添加**，批量创建设备。

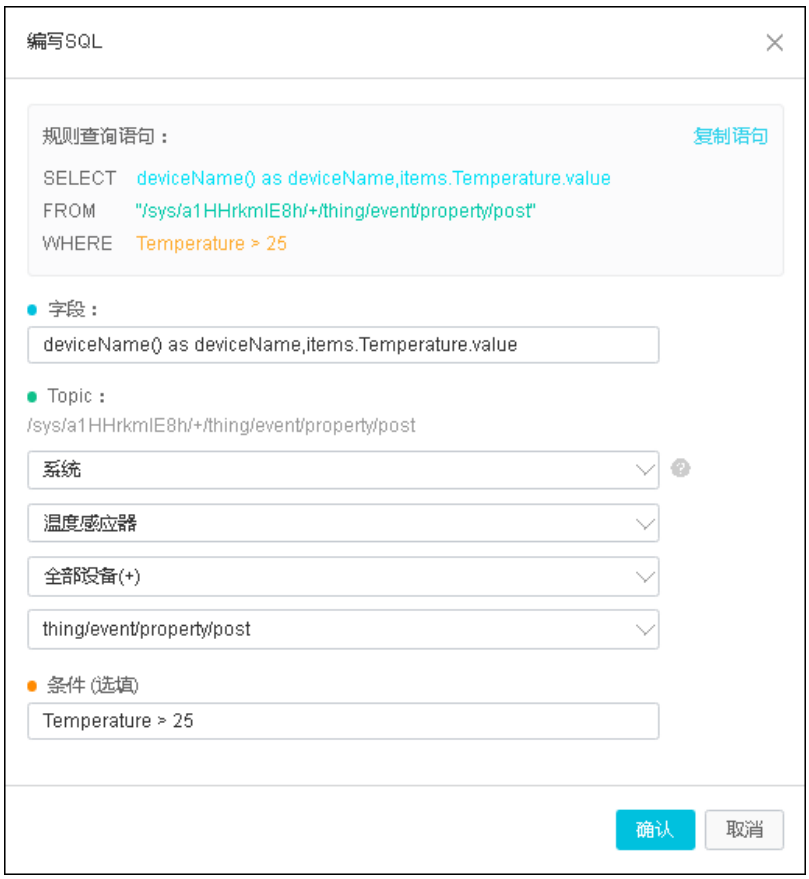
   本示例场景下，共需创建5,000个设备。批量创建设备功能限制一次只能创建最多1,000个设备，因此需进行五次批量创建设备操作。批量创建设备的具体操作指导，请参见批量创建设备。

6. 在左侧导航栏中，选择**规则引擎 > 云产品流转**，创建数据流转规则，以实现将温度感应器上报的温度数据流转到函数计算中。

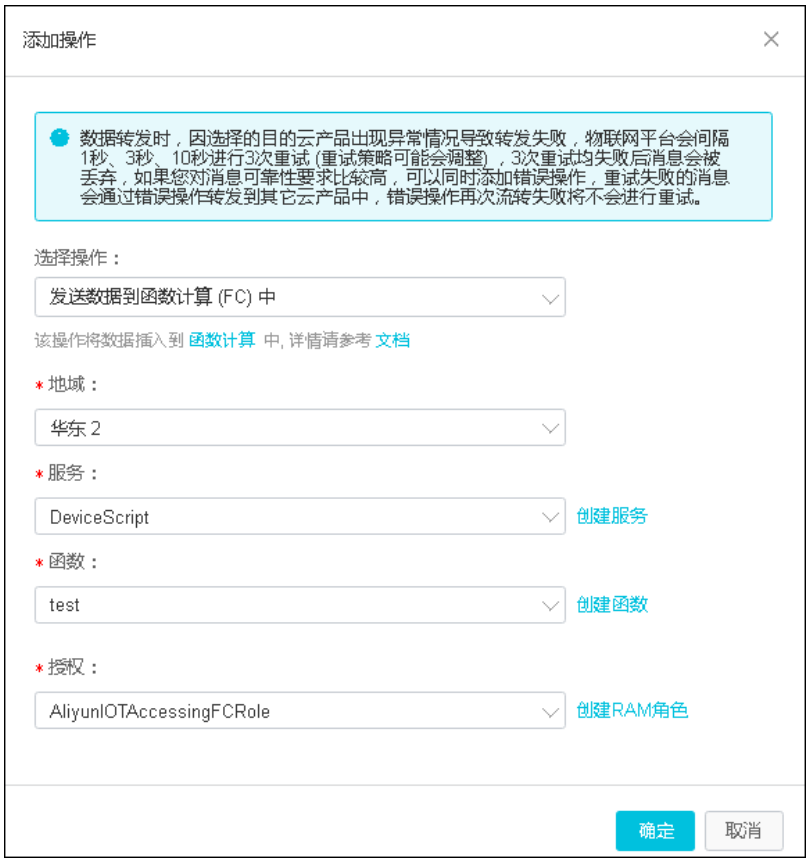
   i. 单击**创建规则**，新建一个数据格式为JSON的规则。

ii. 单击规则对应的**查看**按钮，进入规则详情页，编写处理数据的SQL。

本示例中的SQL表示当温度感应器产品下的设备上报的温度数据高于25℃时，从上报的属性数据中筛选出设备名称（deviceName）和温度（Temperature）两个字段的值。这两个字段将被流转到函数计算中。



iii. 设置数据转发目的地为函数计算。



iv. 规则配置完成后，返回**云产品流转**页，单击规则对应的**启动**按钮，启动规则。

有关数据流转规则更具体的操作指导，请参见设置数据流转规则。

后续步骤

设置报警联系人

创建事件报警规则

创建阈值报警规则

4.1.3. 设置报警联系人

当云监控发现异常时，可将报警信息，通过邮件、钉钉机器人消息等方式，发送到报警联系人组。您需要先创建报警联系人，并把联系人添加到报警联系组。在创建报警规则时，选择相应的报警联系组，才能收到报警通知。

前提条件

如果您想通过钉钉机器人接收报警信息，需先在钉钉应用中，创建钉钉群机器人（智能群助手）。具体操作步骤，请参见通过钉钉群接收报警通知。

操作步骤

1. 创建报警联系人。
2. 创建报警联系组，同时向报警联系组添加报警联系人。

   您也可以向已有的报警联系组添加报警联系人，参见批量添加报警联系人到报警联系组。

后续步骤

创建事件报警规则

创建阈值报警规则

4.1.4. 创建事件报警规则

物联网平台对设备连接请求频率，数据上下行通信频率，消息流转频率等指标都有使用限制约定。使用物联网平台时，一旦触发了使用限制条件，就会被限流，影响业务正常运行。结合云监控的监控和报警功能，您可以及时收到异常报警消息，以便做相应业务调整。

背景信息

物联网平台使用限制，请参见使用限制。

操作步骤

1. 登录云监控控制台。
2. 在左侧导航栏中，选择**报警服务 > 报警规则**。
3. 在**报警规则列表**页，单击**事件报警 > 创建事件报警**。
4. 在右侧弹出的**创建/修改事件报警**对话框中，设置报警规则名称、具体规则和报警通知联系人组合通知方式后，单击**确定**。

   事件报警具体规则详情配置如下表。

参数	说明
报警规则名称	根据提示文案，设置一个合法的规则名称。
事件类型	勾选**系统事件**。
产品类型	选择**物联网平台**。
事件类型	选择**全部类型**。
事件等级	选择**全部级别**。
事件名称	勾选需监控的事件项目。 ○ 任一设备每分钟最大连接请求数达到上限 ○ 当前账号每秒最大连接请求数达到上限 ○ 当前账号每秒发布请求数达到上限 ○ 任一设备上行消息QPS达到上限 ○ 当前账号每秒到达规则引擎的请求数达到上限
资源范围	勾选**全部资源**。
报警方式	勾选**报警通知**，设置通知联系组和通知方式。

5. 调试报警规则。

   i. 在**报警规则列表**中，单击该规则对应的**调试**。

ii. 在右侧弹出的**创建事件调试**对话框中，选择事件名称，修改报警内容数据，单击**确定**，进行事件报警调试。

您选择事件名称后，下方输入框中会出现JSON格式的报警内容格式，您可以根据实际情况，修改报警内容数据。

iii. 通知联系组人员查看是否接收到报警信息。

如钉钉群中，会收到类似如下消息。

后续步骤

创建阈值报警规则

4.1.5. 创建阈值报警规则

云监控阈值报警功能根据您设置的阈值报警规则，监控指定产品下的设备在线数量、物模型通信失败数、规则引擎流转消息次数等，并将报警信息以指定方式发送给指定人员。

操作步骤

1. 登录云监控控制台。
2. 在左侧导航栏中，选择**报警服务 > 报警规则**。
3. 在**报警规则列表**页，单击**阈值报警**页签下的**创建报警规则**。
4. 关联物联网平台资源，指定要监控的产品。

5. 设置报警规则。需根据业务需要，逐一添加报警规则。

本示例中，设置了如下图三个阈值报警规则，并将通道沉默时间设置为10分钟（即如果发出报警10分钟之后，报警还未解除，则重发报警通知）。

上图中的规则说明如下表。

规则名	规则描述	说明
属性上报失败率	每15分钟为一个采样周期，连续三个周期上报属性失败数大于或等于100，则触发报警。	此规则用于监控温度感应器产品下所有设备上报属性的情况。若连续出现大量属性上报不成功，则需进行设备和网络检查。
数据流转次数	每1分钟为一个采样周期，连续三个周期流转至函数计算的次数大于或等于10,000，则触发报警。	根据数据流转规则，设备上报的温度大于25时，规则引擎将数据转发至函数计算。如果流转次数达到10,000，说明大量设备在一分钟内上报了两次以上超过25℃的温度值。需关注温度变化，进行险情排查。
在线设备数量下降	每5分钟为一个采样周期，连续三个周期在线设备数量小于4,800，则触发报警。	连续出现大量设备同时不在线，需检查设备和网络状况。

6. 设置报警通知方式。选择通知对象联系人组和通知方式，其他保持默认即可。

7. 单击**确认**，完成规则创建。

执行结果

阈值报警规则创建成功后，云监控将根据规则持续进行监控。当其中任何一个阈值规则被触发后，云监控将根据通知方式配置发送相应的报警信息。

4.2. 设备OTA固件升级实践

4.2.1. 概述

OTA（Over-the-Air Technology）即空中下载技术，是物联网平台的一项基础功能。您可使用物联网平台的OTA升级功能，对分布在全球各地的IoT设备进行OTA升级。本文以MQTT协议下的设备为例，介绍设备进行OTA升级的过程 。

背景信息

本示例中，使用阿里云提供的设备端4.x版本C Link SDK，开发设备端接入和OTA升级功能，使设备接入物联网平台，并进行OTA升级。

设备进行OTA升级的流程和数据格式说明，请参见设备端OTA升级。本示例介绍为设备进行整包升级的流程，其中升级包模块使用默认（default）模块，升级包仅含一个文件。

OTA升级流程

流程图如下。

流程说明如下表。

序号	说明	相关文档
1	上报当前OTA版本到Topic：`/ota/device/inform/${YourProductKey}/${YourDeviceName}`。 1. 开发设备A，设置OTA固件版本号为1.0.0，激活设备A上线并上报当前固件版本。 2. 开发设备B（与设备A属于同一产品），设置OTA固件版本号为2.0.0，激活设备B上线并上报当前OTA版本。 上报版本消息示例： `{ "id": 1, "params": { "version": "1.0.0" } }`	配置设备端OTA升级
2	在物联网平台为目标产品上传高版本（2.0.0）的OTA升级包，然后向低版本（1.0.0）设备推送升级任务，将设备的固件从低版本（1.0.0）升级到高版本（2.0.0）。	推送OTA升级任务给设备

<table>
<tr><th>序号</th><th>说明</th><th>相关文档</th></tr>
<tr><td>3</td><td>设备端订阅物联网平台推送OTA升级通知消息的Topic：<code>/ota/device/upgrade/${YourProductKey}/${YourDeviceName}</code>。
升级通知消息示例：
<pre>{
    "code":"1000",
    "data":{
        "size":11472299,
        "sign":"83254ac96e141affb8aa42cbfec9****",
        "version":"2.0.0",
        "url":"https://iotx-ota.oss-cn-shanghai.aliyuncs.com/ota/dbab6f742ae389b40db88fc2500b****/ck0q5lyav00003i7hezxe****.zip?Expires=1568951190&OSSAccessKeyId=cS8uRRy54Rsz****&Signature=nk0sogaxtyp7dYvKZnjNQ%2BZ8Q9****",
        "signMethod":"Md5",
        "md5":"83254ac96e141affb8aa42cbfec9****"
    },
    "id":1568864790381,
    "message":"success"
}</pre></td><td rowspan="3">查看设备日志</td></tr>
<tr><td>4</td><td>设备端收到升级通知消息中的升级包URL后，调用SDK提供的API下载升级包，进行本地升级。</td></tr>
<tr><td>5</td><td>上报升级进度到Topic：<code>/ota/device/progress/${YourProductKey}/${YourDeviceName}</code>。
上报进度消息示例：
<pre>{
  "id": 1,
  "params": {
    "step":"1",
    "desc":"******"
  }
}</pre></td></tr>
</table>

序号	说明	相关文档
6	上报升级后的OTA版本到 Topic：`/ota/device/inform/${YourProductKey}/${YourDeviceName}`。 上报版本消息示例： `{ "id": 1, "params": { "version": "2.0.0" } }`	

4.2.2. 配置设备端OTA升级

阿里云物联网平台提供设备端SDK，设备使用SDK与平台建立通信。本文示例使用平台提供的样例程序fota_posix_demo.c，模拟设备接入和OTA升级。

前提条件

已创建产品和设备，并获取设备证书（ProductKey、DeviceName和DeviceSecret）。具体操作，请参见创建产品和创建设备。

本示例需要创建两个设备SDevice1和SDevice2。

背景信息

- 本文使用Linux下的设备端C语言SDK。该SDK的编译环境推荐使用64位的Ubuntu16.04。
- SDK的开发编译环境会用到以下软件：

  make（4.1及以上版本）、gcc（5.4.0及以上版本）。

  可以使用如下命令行安装：

  ```
  sudo apt-get install -y build-essential make gcc
  ```

操作步骤

1. 获取设备端C语言SDK。
   i. 登录物联网平台控制台。
   ii. 在控制台左上方，选择物联网平台所在地域，然后在**实例概览**页面，单击实例。
   iii. 在左侧导航栏单击**文档与工具**，然后在**设备接入SDK**区域的**Link SDK**下，单击**SDK定制**。

iv. 在**高级能力**下，单击**OTA**，其他参数使用默认配置，然后单击**开始生成**，将SDK的ZIP文件保存至本地。

2. 解压本地的C语言SDK文件，修改*/LinkSDK/demos/fota_posix_demo.c*文件中的设备接入信息。

   此处修改为设备**SDevice1**的信息。

```
char *product_key       = "g18***";
char *device_name       = "SDevice1";
char *device_secret     = "cefbebf00***";
...
...
char  *url = "iot-***.mqtt.iothub.aliyuncs.com";
```

参数	示例	说明

参数	示例	说明
url	iot-***.mqtt.iothub.aliyuncs.com	设备的接入域名。 ○ 接入域名格式为 *${YourProductKey}*.iot-as-mqtt.*${YourRegionId}*.aliyuncs.com 。
product_key	g18***	设备认证信息。更多信息，请参见获取设备认证信息。 本例程的身份认证方式为一机一密。
device_name	SDevice1	
device_secret	cefbebf00***	

*fota_posix_demo.c*文件中已提供设备进行OTA升级的代码示例，OTA升级前设备上报的版本号为 1.0.0 。在实际业务中，您需从设备的配置区获取实际的版本号，并执行编写代码。更多信息，请参见示例代码说明。

```
cur_version = "1.0.0";
res = aiot_ota_report_version(ota_handle, cur_version);
if (res < STATE_SUCCESS) {
    printf("aiot_ota_report_version failed: -0x%04X\r\n", -res);
}
```

3. 登录Linux虚拟机，执行以下命令，安装所需软件。

```
sudo apt-get install -y build-essential make gcc
```

4. 将步骤2中的已修改完成的LinkSDK文件，上传至Linux虚拟机的开发环境。
5. 在SDK根目录*/LinkSDK*下，执行make命令，完成样例程序的编译。

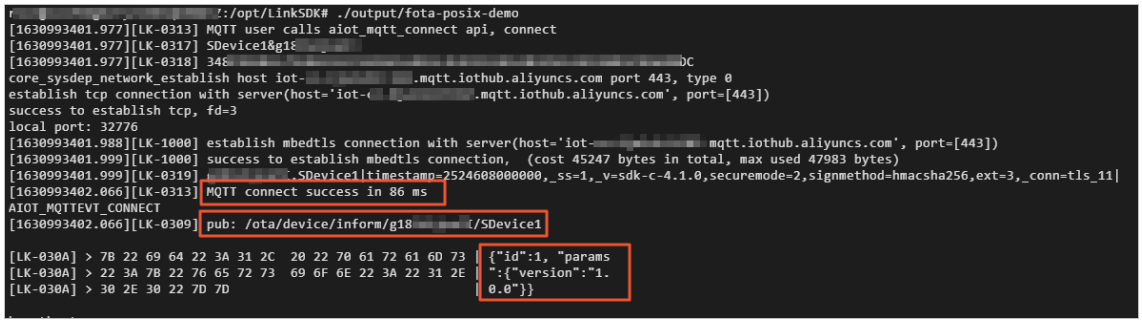
```
make clean
make
```

生成的样例程序*fota-posix-demo*存放在*./output*目录下。

6. 执行以下命令，运行样例程序。

```
./output/fota-posix-demo
```

7. 查看设备运行日志和状态。
   - 设备连接信息和上报版本号日志如下。



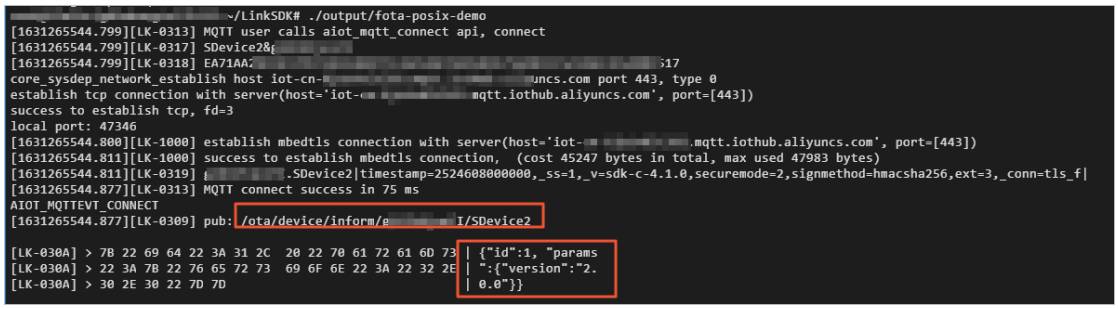
- 返回物联网平台控制台对应实例下，在左侧导航栏选择**设备管理 > 设备**，找到目标设备，查看设备状态。设备状态显示为**在线**，则表示设备与物联网平台成功连接。

8. 参照以上步骤，开发设备**SDevice2**，上报OTA模块版本号2.0.0。
   i. 参照步骤，获取SDK的ZIP文件。
   ii. 解压ZIP文件，在*/LinkSDK/demos/fota_posix_demo.c*文件中，修改**SDevice2**的接入信息和OTA模块版本号。

   代码示例如下：

```
char *product_key       = "g18***";
char *device_name       = "SDevice2";
char *device_secret     = "8b2ada9a0***";
...
...
char  *url = "iot-***.mqtt.iothub.aliyuncs.com";
...
...
cur_version = "2.0.0";
    res = aiot_ota_report_version(ota_handle, cur_version);
    if (res < STATE_SUCCESS) {
        printf("aiot_ota_report_version failed: -0x%04X\r\n", -res);
    }
```

   iii. 参照步骤~步骤，查看设备**SDevice2**成功上报高版本号。



后续步骤

推送OTA升级包到设备端：物联网平台下发OTA升级任务到设备，在线设备获取升级信息，进行OTA升级。

4.2.3. 推送OTA升级包到设备端

本文介绍如何在物联网平台控制台上推送OTA升级包到设备端，流程包括添加升级包、验证升级包和发起批量升级。

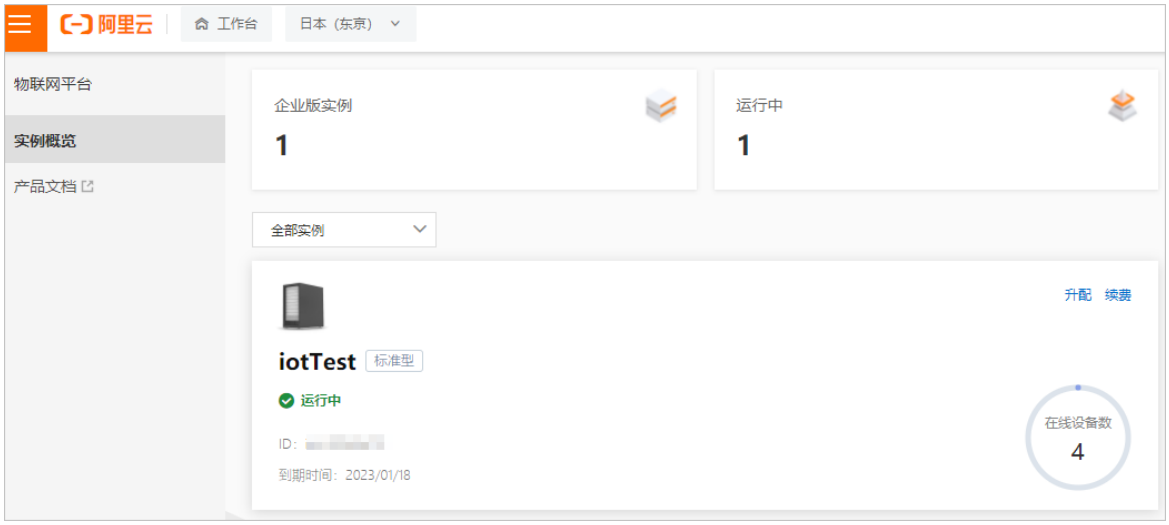
前提条件

设备已接入到物联网平台，并上报当前版本号。具体操作，请参见配置设备端OTA升级。

操作步骤

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > 注意 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**监控运维 > OTA升级**。
4. 在**OTA升级**页面，单击**升级包列表**页签，单击**添加升级包**。
5. 配置升级包信息，上传固件文件作为升级包，单击**确认**。

   部分配置如下表所示，其他参数配置，请参见添加升级包。

参数	配置
升级包类型	整包
升级包模块	default
升级包版本号	2.0.0
签名算法	MD5
升级包是否需要平台验证	是

6. 在升级包列表中，单击升级包对应的**验证**，使用测试设备进行升级包验证。具体配置，请参见验证升级包。

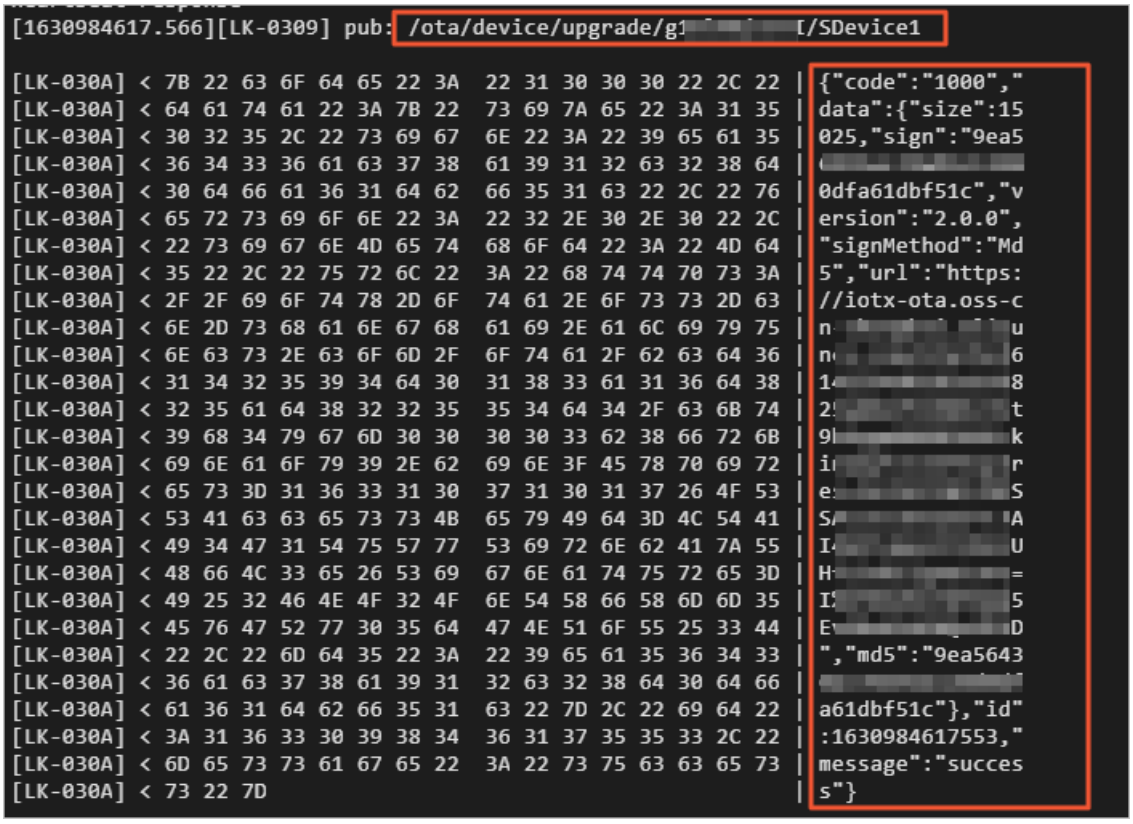
   测试设备升级成功后，验证通过，**批量升级**按钮显示为可用状态。
7. 单击**批量升级**，进行如下配置，向设备推送升级通知。参数说明，请参见发起升级批次任务。
   - 升级范围配置：

- 升级策略配置：

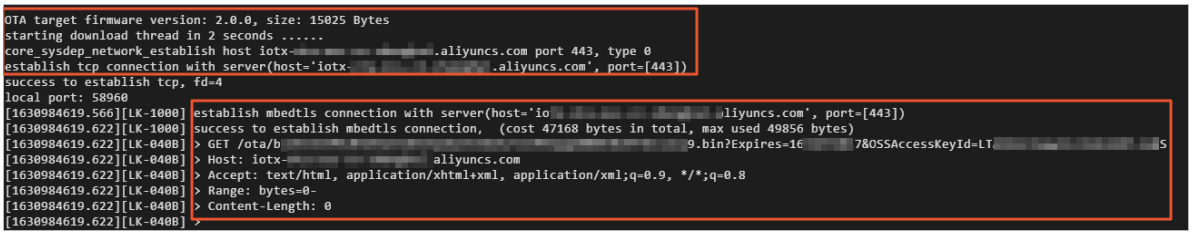
查看设备日志

物联网平台推送OTA升级通知后，可通过设备端日志查看设备端OTA升级情况，包含获取通知信息、处理消息、下载OTA升级包文件、进行升级和上报升级进度的日志。

- 设备端收到的升级通知信息。



- 获取新版本信息，连接OTA升级包下载地址等。



- 下载固件，并上报进度。

```
[1630984619.622][LK-0309] pub: /ota/device/progress/[redacted]I/SDevice1

[LK-030A] > 7B 22 69 64 22 3A 32 2C  20 22 70 61 72 61 6D 73 | {"id":2, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 30 22 2C 22 64 | ":{"step":"0","d
[LK-030A] > 65 73 63 22 3A 22 22 7D  7D                      | esc":""}}

[1630984619.644][LK-040D] < HTTP/1.1 206 Partial Content
[1630984619.644][LK-040D] < Server: AliyunOSS
[1630984619.644][LK-040D] < Date: Tue, 07 Sep 2021 03:16:59 GMT
[1630984619.644][LK-040D] < Content-Type: application/octet-stream
[1630984619.644][LK-040D] < Content-Length: 15025
[1630984619.644][LK-040D] < Connection: keep-alive
[1630984619.644][LK-040D] < x-oss-request-id: 6136D9AB3731FB36345F2350
[1630984619.644][LK-040D] < Content-Range: bytes 0-15024/15025
[1630984619.644][LK-040D] < Accept-Ranges: bytes
[1630984619.644][LK-040D] < ETag: "9EA5[redacted]"
[1630984619.644][LK-040D] < Last-Modified: Tue, 07 Sep 2021 02:47:02 GMT
[1630984619.644][LK-040D] < x-oss-object-type: Normal
[1630984619.644][LK-040D] < x-oss-hash-crc64ecma: 1317[redacted]
[1630984619.644][LK-040D] < x-oss-storage-class: Standard
[1630984619.644][LK-040D] < Content-MD5: r[redacted]
[1630984619.644][LK-040D] < x-oss-server-time: 16
[1630984619.644][LK-040D] <
download 054% done, +8192 bytes
[1630984619.644][LK-0309] pub: /ota/device/progress/g1[redacted]I/SDevice1

[LK-030A] > 7B 22 69 64 22 3A 33 2C  20 22 70 61 72 61 6D 73 | {"id":3, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 35 34 22 2C 22 | ":{"step":"54","
[LK-030A] > 64 65 73 63 22 3A 22 22  7D 7D                   | desc":""}}

[1630984619.644][LK-0901] digest matched
download 100% done, +6833 bytes
[1630984619.644][LK-0309] pub: /ota/device/progress/g[redacted]I/SDevice1

[LK-030A] > 7B 22 69 64 22 3A 34 2C  20 22 70 61 72 61 6D 73 | {"id":4, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 31 30 30 22 2C | ":{"step":"100",
[LK-030A] > 22 64 65 73 63 22 3A 22  22 7D 7D                | "desc":""}}

download completed
```

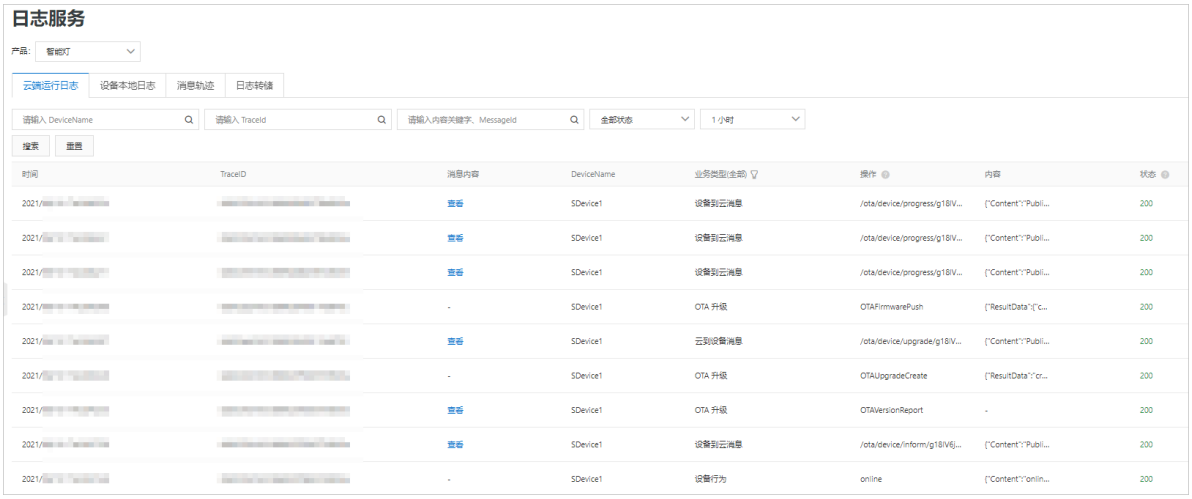
- 设备升级完成，上报新的版本号。

```
[1631267018.622][LK-0309] pub: /ota/device/inform/g[redacted]mTI/SDevice1

[LK-030A] > 7B 22 69 64 22 3A 31 2C  20 22 70 61 72 61 6D 73 | {"id":1, "params
[LK-030A] > 22 3A 7B 22 76 65 72 73  69 6F 6E 22 3A 22 32 2E | ":{"version":"2.
[LK-030A] > 30 2E 30 22 7D 7D                                | 0.0"}}
```

查看物联网平台日志

您可在物联网平台控制台查看设备的升级状态、升级包信息和升级日志等。

- 在**升级包详情**页面，查看升级批次信息和升级状态。具体操作，请参见查看升级情况。
- 在**监控运维 > 日志服务**页面，选择产品后，查看设备在线进行OTA升级时，与物联网平台的通信日志。


日志服务
产品: 智能灯
云端运行日志
设备本地日志
消息轨迹
日志转储
请输入 DeviceName
请输入 TraceId
请输入内容关键字、MessageId
全部状态
1 小时
搜索
重置
时间
TraceID
消息内容
DeviceName
业务类型(全部)
操作
内容
状态
查看
SDevice1
设备到云消息
OTA 升级
云到设备消息
设备行为
OTAFirmwarePush
OTAUpgradeCreate
OTAVersionReport
online
200

5.场景应用

5.1. 通过大数据平台搭建设备监控大屏

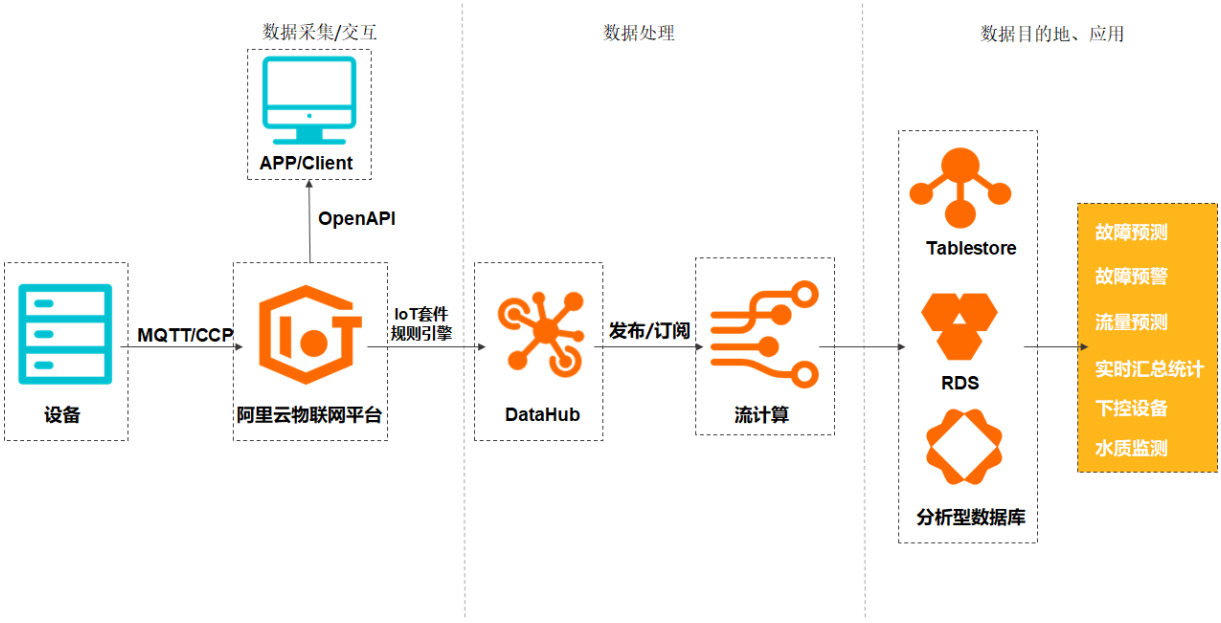
本文介绍如何对接物联网平台和阿里云大数据平台，以实现设备数据分析、统计、计算和可视化实时展示。

前提条件

- 开通、购买相关阿里云产品实例和计算资源。使用阿里云大数据平台处理物联网平台设备相关数据，涉及多个阿里云产品，包括云数据库RDS MySQL版、物联网平台等。
- 了解涉及产品的配置使用方法和注意事项。

背景信息

- 任务场景：在DataV可视化大屏上，实时展示物联网平台某产品下的设备相关数据。
- 实现过程：
  i. 物联网平台采集设备数据。
  ii. 通过规则引擎，物联网平台将一个产品下的设备数据转发至流数据处理平台DataHub中。
  iii. DataHub根据相关配置，将设备数据发送至实时计算平台进行计算处理后，再写入RDS MySQL版数据库中。（若无需计算处理的数据，可通过DataConnector将数据直接从DataHub 平台同步到云数据库RDS MySQL版数据库中。）
  iv. DataV根据配置，以MySQL数据库表作为数据源，实时展示相关设备数据 。



操作步骤

1. 创建一个云数据库RDS版MySQL数据库，用于存储设备数据。

   了解云数据库RDS版，请参见云数据库RDS MySQL版。

   i. 登录云数据库RDS版控制台。

ii. 在**云数据库管理**页，单击**创建实例**，创建一个MySQL类型的数据库实例。

> **说明** RDS for MySQL数据库实例的地域须与物联网平台设备地域和DataHub项目地域保持一致。

iii. 在您的数据库实例列表中，单击该实例对应的**管理**。

iv. 在左侧导航栏中，单击**账号管理**，创建数据库用户账号。

v. 在左侧导航栏中，单击**数据库管理**，创建数据库。

vi. 在左侧导航栏中，单击**数据安全性**，添加数据库白名单。请参见设置IP白名单文档中的设置方法。

vii. 在左侧导航栏中，单击**基本信息**，查看该数据库的信息。

后续步骤中，该数据库信息需配置到DataHub、DataV或阿里实时计算开发平台中，用于同步数据。

viii. 在**基本信息**页上方导航栏中，单击**登录数据库**，输入信息登录数据库。

ix. 创建数据库表。如，表mytable包含两个字段：

mytable

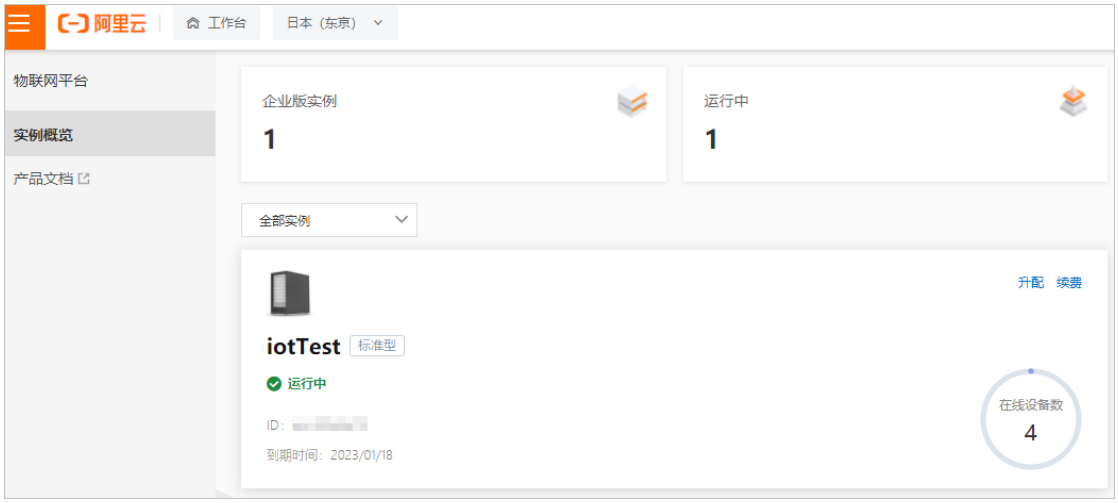
字段名	类型	说明
d_data	varchar(32)	时间。
device_num	int	活跃设备数量。

2. 在物联网平台控制台，创建产品和设备，并配置规则引擎。

i. 登录物联网平台控制台。

ii. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 目前华东2（上海）、日本（东京）地域开通了企业版实例服务。其他地域，请跳过此步骤。



iii. 在左侧导航栏中，单击**设备管理 > 产品**，然后创建产品。如需帮助，请参见创建产品。

iv. 产品创建成功后，进入该产品的**产品详情**页，根据您的业务需要，为产品创建自定义Topic类，定义产品功能（即定义物模型）等。

v. 在左侧导航栏中，单击**设备管理 > 设备**，然后注册设备。

vi. 在左侧导航栏中，单击**规则引擎 > 云产品流转**，创建一条规则。

vii. 在规则列表中，单击规则对应的**查看**按钮。

viii. 在**数据流转规则**页，单击**处理数据**栏的**编写SQL**按钮，为该条规则编写数据处理SQL，并调试SQL语句。

ix. 单击**转发数据**栏对应的**添加操作**按钮，并配置规则动作将设备数据转发至DataHub的Topic中。如需帮助，请参见设置数据流转规则。

x. 规则配置完成后，在**云产品流转**页的规则列表中，单击该规则对应的**启动**按钮，启动规则。

规则启动后，使用模拟设备发送消息，检验设备发送的消息是否成功流转至DataHub中。可以在设备的**日志服务**页查看设备日志；在DataHub控制台对应的Topic中，查看Shards中数据量的变化，并通过数据抽样功能，可以看到具体的消息内容。

3. 开发设备端，连接设备与物联网平台。

本例以Java Link SDK为例开发设备：下载Java SDK Demo。开发方法，请参见工程配置。

设备端SDK开发完成，并将SDK烧录至物理设备中。设备上电联网后，建立与物联网平台之间的连接，便可与物联网平台进行数据交换。数据通过规则引擎转发至DataHub中，再经过实时计算处理后，写入数据库中。

4. 测试。配置完成之后，使用不同的设备登录，并发送消息，大屏展示的活跃设备数会实时改变。

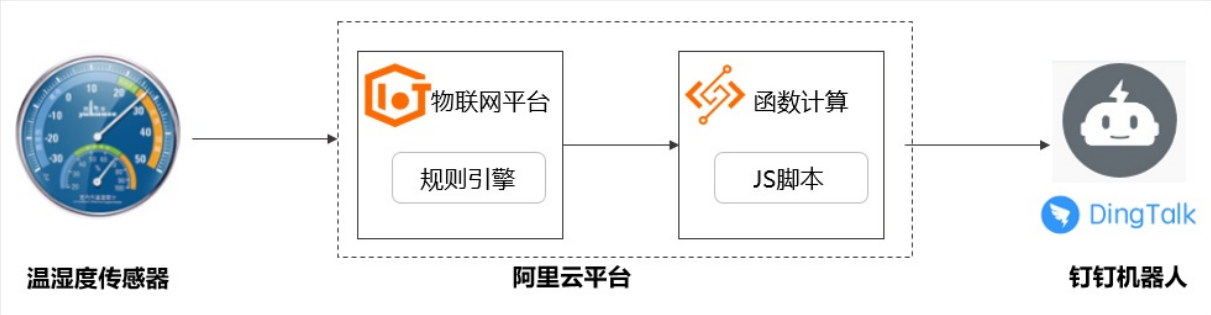
5.2. 推送设备上报数据到钉钉群

本文以温湿度传感器设备为例，介绍通过数据流转规则，将设备上报数据推送到钉钉群的操作步骤。

场景说明

各办公室分布点的温湿度传感器设备上报数据到钉钉群机器人。

数据流转流程图



步骤一：创建产品和设备

1. 登录物联网平台控制台。
2. 在左侧导航栏，选择**设备管理 > 产品**，创建一个直连设备类型的产品：*温湿度传感器*。

   参数设置直接使用默认值。具体操作，请参见创建产品。
3. 单击**前往定义物模型**，在**功能定义**页签，单击**编辑草稿**，然后在**默认模块**，为产品添加自定义功能。

   本文示例为产品添加温度和湿度两个属性，请参见单个添加物模型。

   默认模块

功能类型	功能名称（全部）	标识符	数据类型	数据定义
属性	湿度 自定义	humidity	int32 (整数型)	取值范围：0 ~ 100
属性	温度 自定义	temperature	double (双精度浮点型)	取值范围：-20 ~ 60

4. 在左侧导航栏，选择**设备管理 > 设备**，在温湿度传感器产品下，创建一个具体的设备：*TH_sensor*，请参见单个创建设备。

   创建设备完成后，在弹出的**添加完成**对话框，单击**前往查看**，获取设备证书（ProductKey、DeviceName和DeviceSecret）。设备证书是设备后续与物联网平台交流的重要凭证，请妥善保管。
5. 在**设备列表**页签，单击设备TH_sensor的**查看**，进入**设备详情**页面。在**标签信息**右侧，单击**编辑**，为设备添加标签。

   本文示例添加如下两个标签，请参见标签。

Key	Value	描述
tag	YY小镇X号楼F层00XS	设备所在位置
deviceISN	T20180102X	设备序列号

步骤二：配置函数计算服务

函数计算，是一个事件驱动的全托管计算服务，目前支持的语言Java、Node.js、Python等语言，具体请参见如何使用函数计算。

1. 配置钉钉机器人，获取Webhook地址。
   i. 登录电脑版钉钉。

ii. 选择钉钉群聊天窗口的群设置按钮，选择**智能群助手**。

iii. 单击**添加机器人**，然后单击添加按钮 。

iv. 选择**自定义**，单击**添加**。

v. 设置机器人名字和安全设置，选中**我已阅读并同意《自定义机器人服务及免责条款》**，单击**完成**。

vi. 单击**复制**，保存Webhook地址到本地。

2. 编写函数计算脚本。

本文以Node.js运行环境为例编写函数脚本，从物联网平台获取设备位置、设备编号、实时温度、相对湿度和上报时间数据，按照钉钉消息格式组装，使用HTTPS协议将POST数据推送到钉钉机器人的Webhook接口。

完成编写后，将脚本文件命名为*index.js*，并压缩为*index.zip*文件进行保存。完整代码脚本如下：

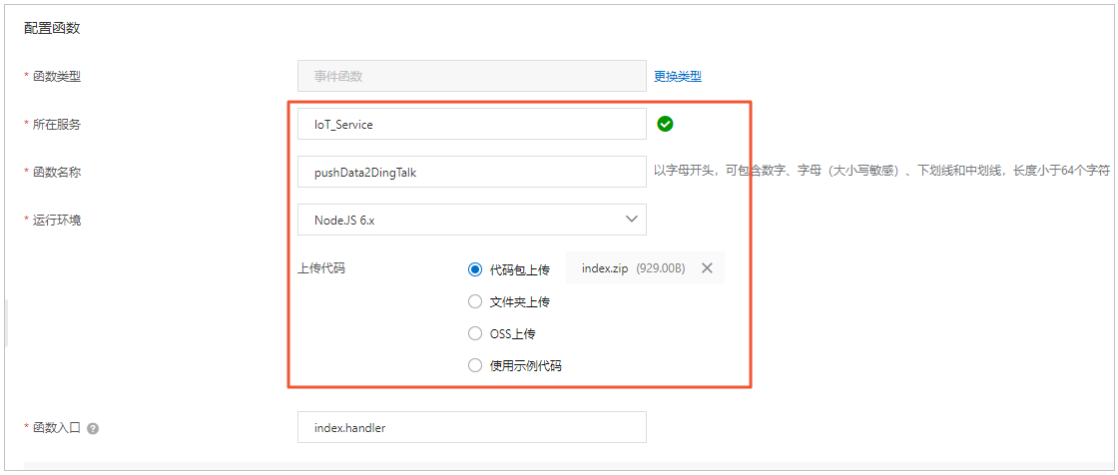
您需将accessToken替换为Webhook地址中access_token的值。

```
const https = require('https');
const accessToken = '填写accessToken，即钉钉机器人Webhook的access_token值';
module.exports.handler = function(event, context, callback) {
var eventJson = JSON.parse(event.toString());
//钉钉消息格式
const postData = JSON.stringify({
"msgtype": "markdown",
"markdown": {
"title": "温湿度传感器",
"text": "#### 温湿度传感器上报\n" +
"> 设备位置：" + eventJson.tag + "\n\n" +
"> 设备编号：" + eventJson.isn+ "\n\n" +
"> 实时温度：" + eventJson.temperature + "℃\n\n" +
"> 相对湿度：" + eventJson.humidity + "%\n\n" +
"> ###### " + eventJson.time + " 发布 by [物联网平台](https://www.aliyun.com/product/iot)
\n"
},
"at": {
"isAtAll": false
}
});
const options = {
hostname: 'oapi.dingtalk.com',
port: 443,
path: '/robot/send?access_token=' + accessToken,
method: 'POST',
headers: {
'Content-Type': 'application/json',
'Content-Length': Buffer.byteLength(postData)
}
};
const req = https.request(options, (res) => {
res.setEncoding('utf8');
res.on('data', (chunk) => {});
res.on('end', () => {
callback(null, 'success');
});
});
// 异常返回
req.on('error', (e) => {
callback(e);
});
// 写入数据
req.write(postData);
req.end();
};
```

3. 创建服务和函数。
   i. 开通阿里云函数计算服务，请参见开通服务。
   ii. 登录函数计算控制台，在左侧导航栏选择**服务及函数**。
   iii. 单击**新增服务**，设置服务名称为*IoT_Service*，然后单击**提交**。
   iv. 在**服务列表**选中**IoT_Service**，然后单击**新增函数**。

v. 将鼠标指针移动到**事件函数**区域，单击**配置部署**。

vi. 配置函数的基础信息，如下表所示，其他参数使用默认设置，然后单击**新建**。



参数名称	说明
函数名称	输入*pushData2DingTalk*。
运行环境	选择**Node.JS 6.x**，**上传代码**为**代码包上传**。单击右侧**上传代码**，上传步骤2中保存的*index.zip*文件。

步骤三：配置数据流转到函数计算中

将TH_sensor上报的温度和湿度等数据转发至函数计算的函数**pushData2DingTalk**中。

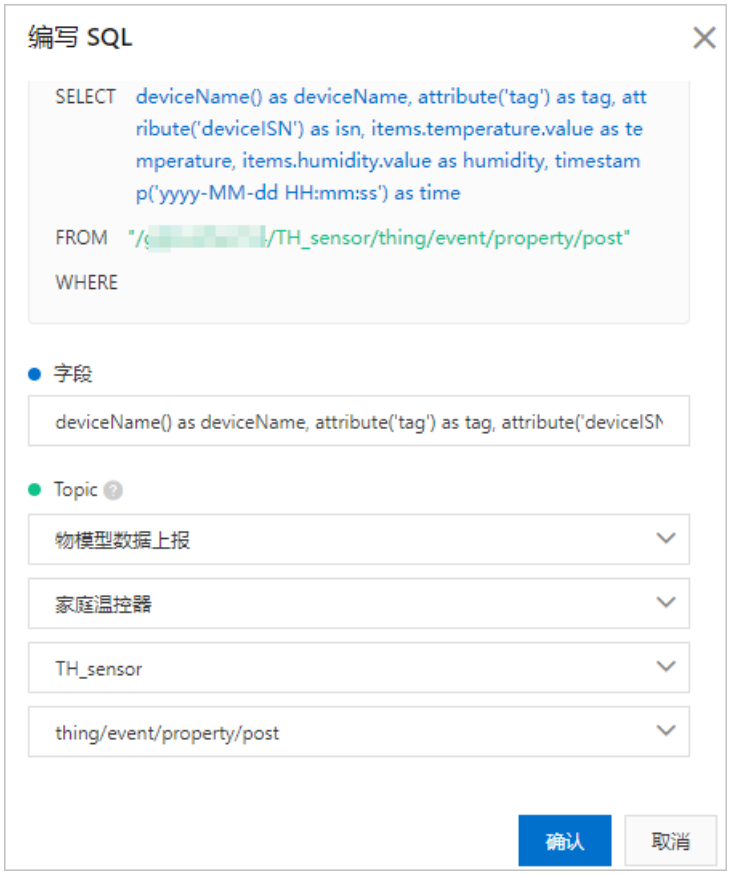
1. 返回物联网平台控制台，在左侧导航栏，选择**规则引擎 > 云产品流转**，单击**创建规则**，输入规则名称：*温湿度数据流转*，单击**确认**。
2. 在**数据流转规则**页面，单击**编写SQL**，编辑处理数据的SQL。

   本文示例中，定义筛选的消息字段包含：

   - 设备信息中的设备名称（deviceName），自定义属性中的标签（tag）和序列号（deviceISN）。
   - 温湿度传感器上报数据消息payload中的温度值（temperature）和湿度值（humidity）。

   具体SQL语句如下：

```
SELECT
deviceName() as deviceName,
attribute('tag') as tag, attribute('deviceISN') as isn,
items.temperature.value as temperature, items.humidity.value as humidity,
timestamp('yyyy-MM-dd HH:mm:ss') as time
FROM
"/g5j3o***/TH_sensorthing/event/property/post"
```



3. 在**数据流转规则**页面，单击**添加操作**，将数据转发到函数计算（FC）。

   本文示例选择已创建的服务**IoT_Service**和函数**pushDat a2DingT alk**，请参见数据转发到函数计算。

4. 在数据流转规则列表中，单击规则**温湿度数据流转**对应的**启动**，启用该规则。

步骤四：接入设备和上报温湿度数据

使用设备证书（ProductKey、DeviceName和DeviceSecret），通过MQTT协议将设备接入物联网平台，并模拟上报温湿度数据。

1. 在Windows系统或Linux系统下载并安装Node.js。本文以windows系统为例。

   ```
   node --version
   ```

2. 在本地计算机创建一个JavaScript文件（例如iot_device.js），用来存放Node.js示例代码。

   Node.js示例代码如下所示：

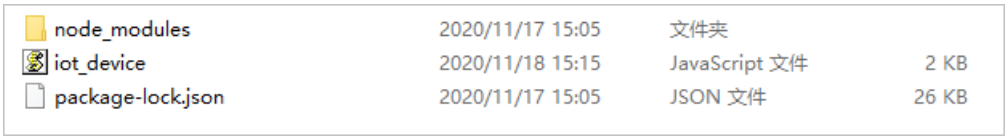
```
const mqtt = require('aliyun-iot-mqtt');
// 1. 设备身份信息
var options = {
    productKey: "a1***",
    deviceName: "TH_sensor",
    deviceSecret: "9JtvE***",
    regionId: 'cn-shanghai'
};
// 1. 建立连接
const client = mqtt.getAliyunIotMqttClient(options);
// 2. 监听云端指令
client.subscribe(`/${options.productKey}/${options.deviceName}/user/get`)
client.on('message', function(topic, message) {
    console.log("topic " + topic)
    console.log("message " + message)
})
setInterval(function() {
    // 3.上报数据
    client.publish(`/sys/${options.productKey}/${options.deviceName}/thing/event/proper
ty/post`, getPostData(), { qos: 0 });
}, 6 * 1000);
function getPostData() {
    const payloadJson = {
        id: Date.now(),
        version: "1.0",
        params: {
            Temperature: Math.floor((Math.random() * 20) + 10),
            Humidity: Math.floor((Math.random() * 20) + 10)
        },
        method: "thing.event.property.post"
    }
    console.log("payloadJson " + JSON.stringify(payloadJson))
    return JSON.stringify(payloadJson);
}
```

参数	示例	说明
productKey	a1***	设备所属产品ProductKey。可在控制台设备详情页查看。
deviceName	TH_sensor	设备名称。可在控制台设备详情页查看。
deviceSecret	9JtvE***	设备密钥。可在控制台设备详情页查看。
regionId	cn-shanghai	MQTT设备所在地域代码。地域代码表达方法，请参见地域和可用区。

3. 打开CMD窗口，使用 `cd` 命令找到iot_device.js文件所在路径，在该路径下使用NPM命令下载阿里云IoT的MQTT库。

```
npm install aliyun-iot-mqtt -S
```

下载后的MQTT库文件如下图所示。



4. 在CMD窗口输入如下命令，运行iot_device.js代码，启动设备，并上报数据。

```
node iot_device.js
```

执行结果

设备接入脚本运行结果如下。

钉钉群机器人接收到消息如下。