阿里云
物联网平台

入门教程

文档版本：20190921

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 快速玩转物联网平台

1.1 概述

本章节帮助您快速了解物联网平台的使用方法。

数据流转过程示意图

准备工作

如按本章介绍的流程进行操作，需做如下准备。

- 开通物联网平台。
- 准备C语言开发环境。示例中，设备端开发是在Linux下，使用阿里云提供的C语言SDK进行开发。
- 准备Java开发环境。示例中，服务端下发指令，使用阿里云提供的云端Java SDK；服务端接收设备消息，使用Java语言的HTTP/2 SDK。

操作步骤

1. #unique_5：在物联网平台上为设备注册一个身份，获取设备证书信息（ProductKey、DeviceName和DeviceSecret）。该证书信息将烧录到设备上，用于设备连接物联网平台时，进行身份认证。
2. 为产品定义物模型：可以从从属性、服务和事件三个维度定义产品功能。物联网平台根据您定义的功能构建出产品的数据模型，用于云端与设备端进行指定数据通信。
3. #unique_7：开发设备端SDK，传入设备的证书信息，使设备端可以连接物联网平台。
4. #unique_8：服务端通过订阅消息类型，接收来自设备的指定类型的消息。
5. #unique_9：调用物联网平台云端API，向设备下发指令。

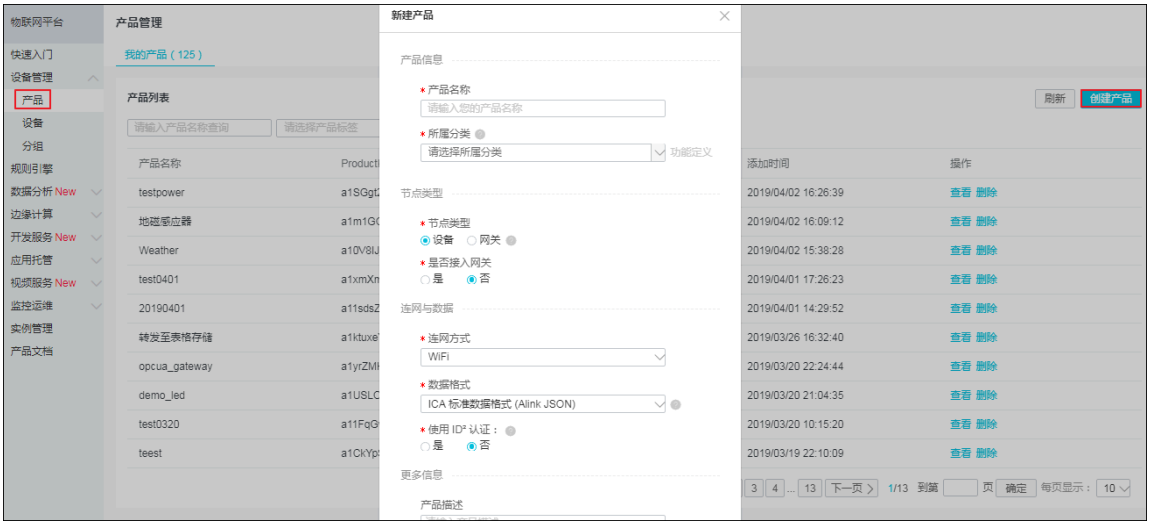
1.2 创建产品与设备

使用物联网平台的第一步是在云端创建产品和对应设备。产品相当于某一类设备的集合，该类设备具有相同的功能，您可以根据产品批量管理对应设备。

操作步骤

1. 登录物联网平台控制台。
2. 创建产品。
   a) 左侧导航栏选择设备管理 > 产品。在产品管理页面，单击创建产品。
   b) 配置参数。

   具体配置细节可参见创建产品文档完成。



   c) 单击完成。

   产品将自动出现在产品列表中。

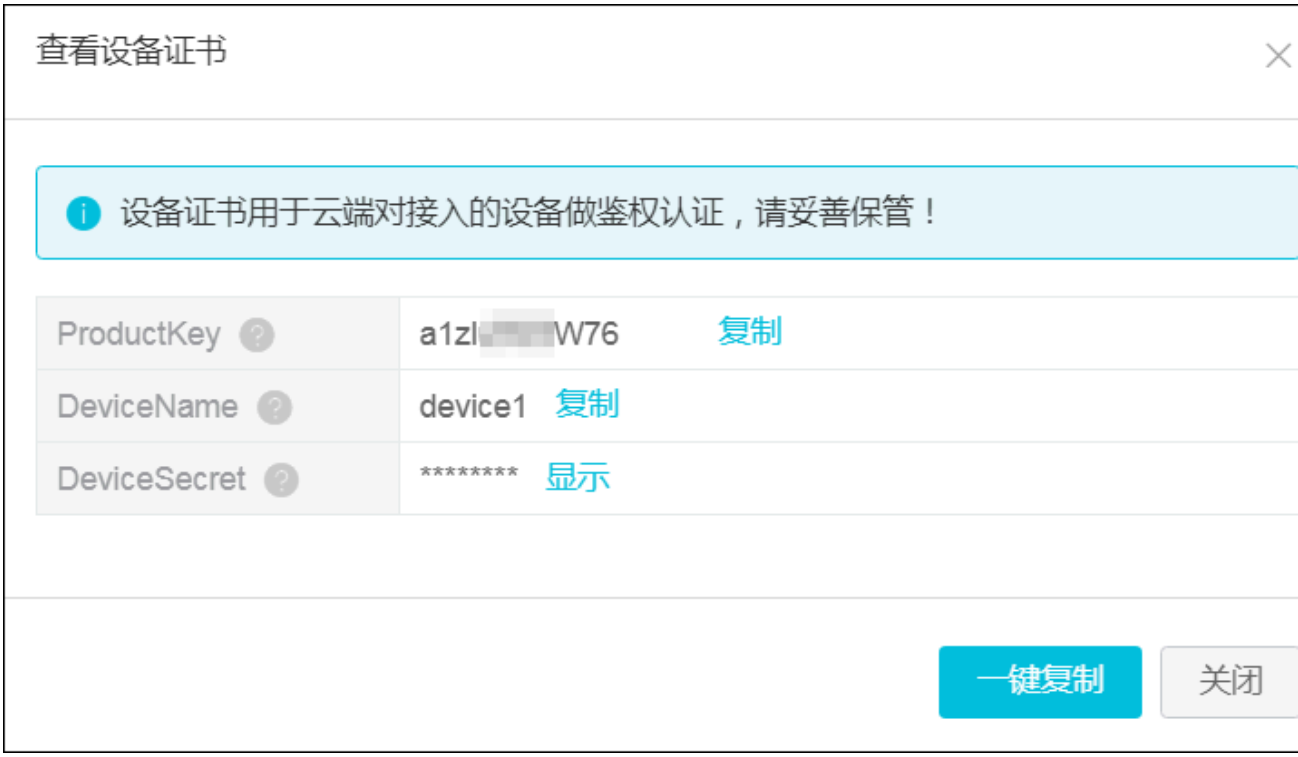
3. 创建设备。

a) 左侧导航栏选中设备管理 > 设备，进入设备管理页面。

b) 单击添加设备。选中上一步创建的产品，输入设备名称（DeviceName），设置设备备注名，单击完成。

c) 记录设备证书信息。

设备证书包含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。



后续步骤

#unique_12。

1.3 为产品定义物模型

物联网平台支持为产品定义物模型，将实际产品抽象成由属性、服务、事件所组成的数据模型，便于云端管理和数据交互。产品创建完成后，您可以为它定义物模型，产品下的设备将自动继承物模型内容。

操作步骤

1. 产品列表中，选中创建的产品，单击查看，进入产品详情页。
2. 单击功能定义。
3. 在自定义功能栏，单击添加功能。

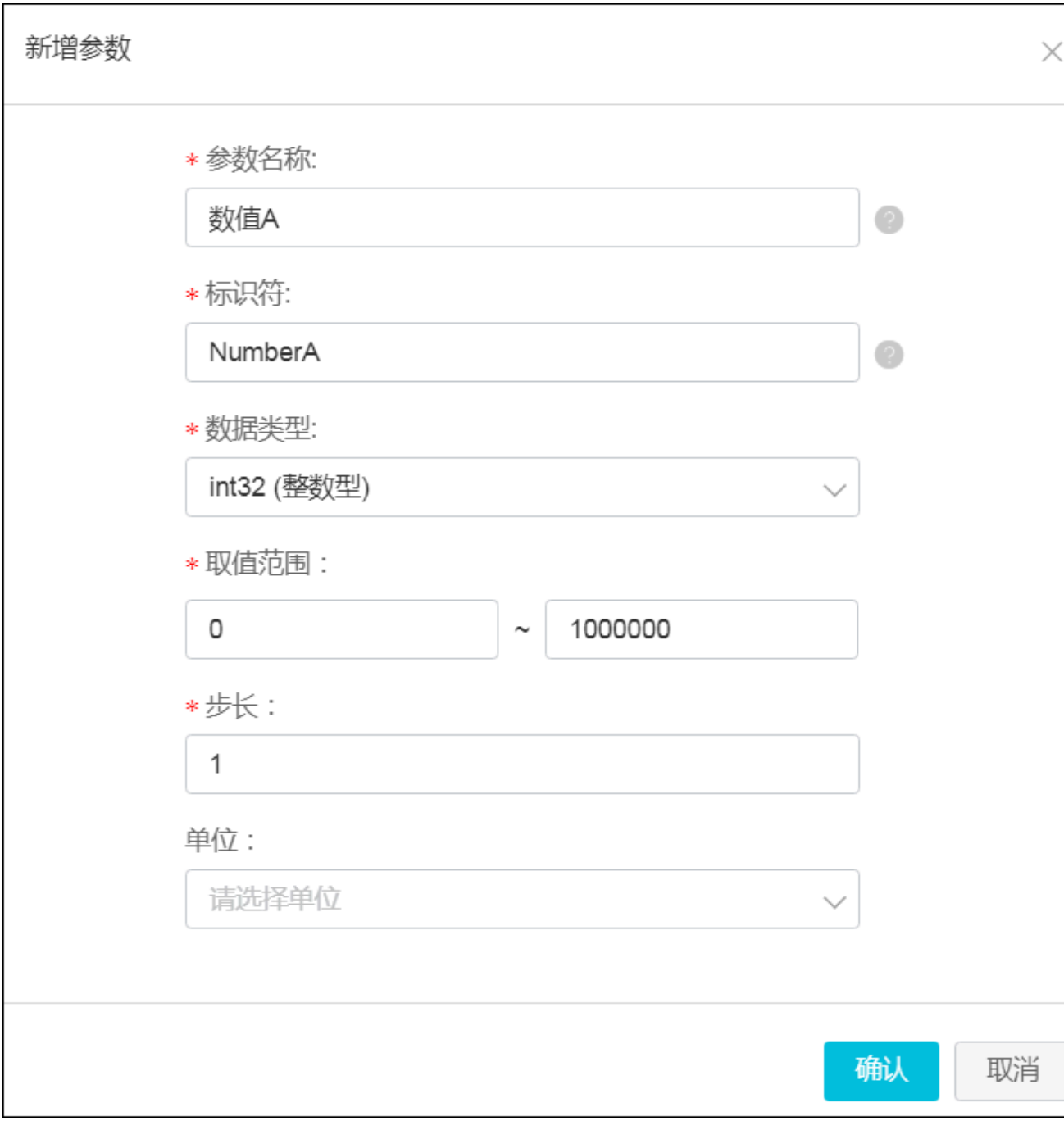
4. 如下图所示，将产品开关定义为属性。完成后单击确认。

5. 如下图所示，将计数器定义为属性。完成后单击确认。

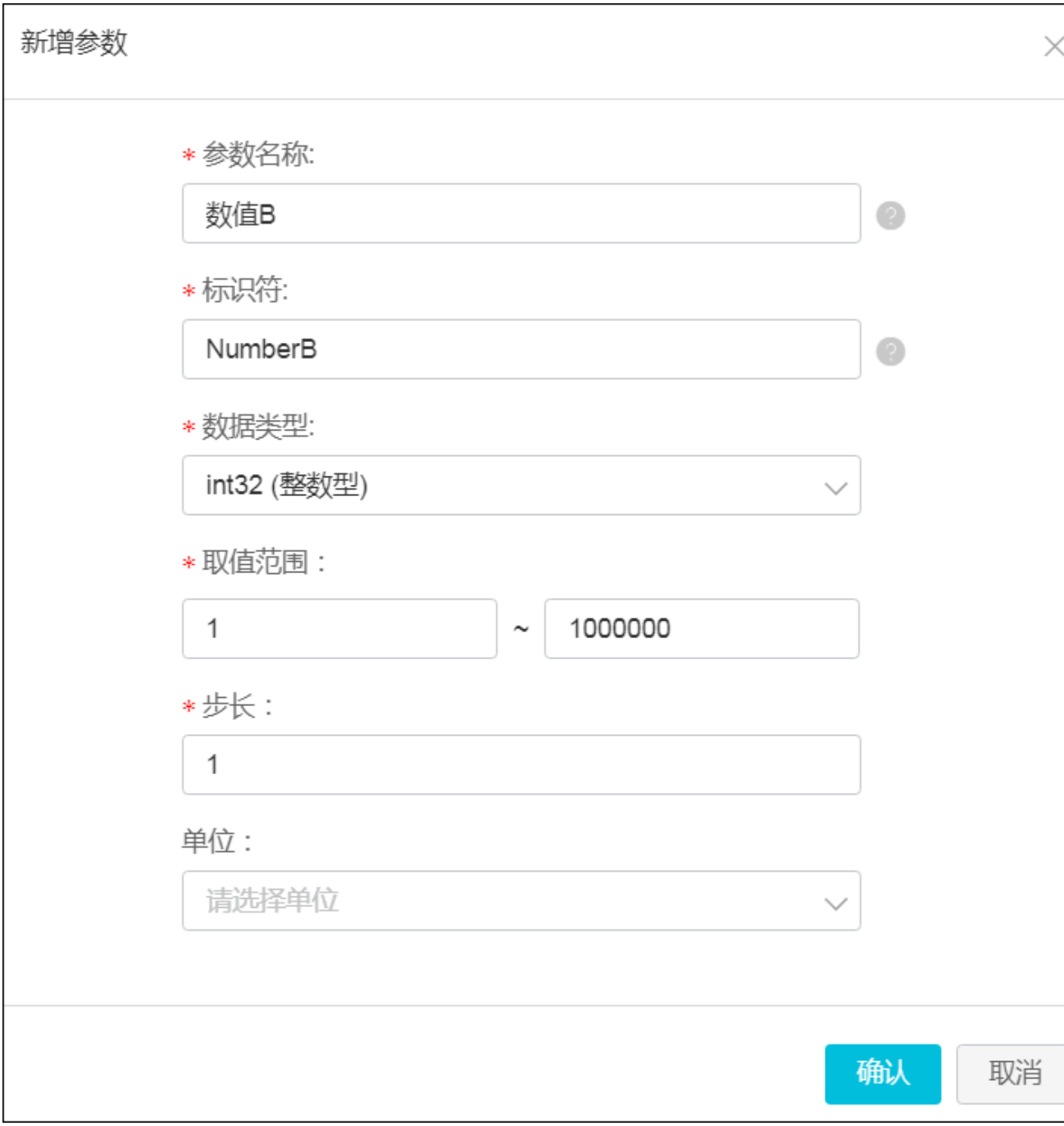

* 功能类型：
属性
服务
事件
* 功能名称:
计数器演示
* 标识符:
Counter
* 数据类型:
int32 (整数型)
* 取值范围：
0
~
99999
* 步长：
1
单位：
请选择单位
读写类型：
读写
只读
描述：
请输入描述
0/100
确认
取消

6. 如下图所示，将数值计算定义为服务。完成后单击确认。

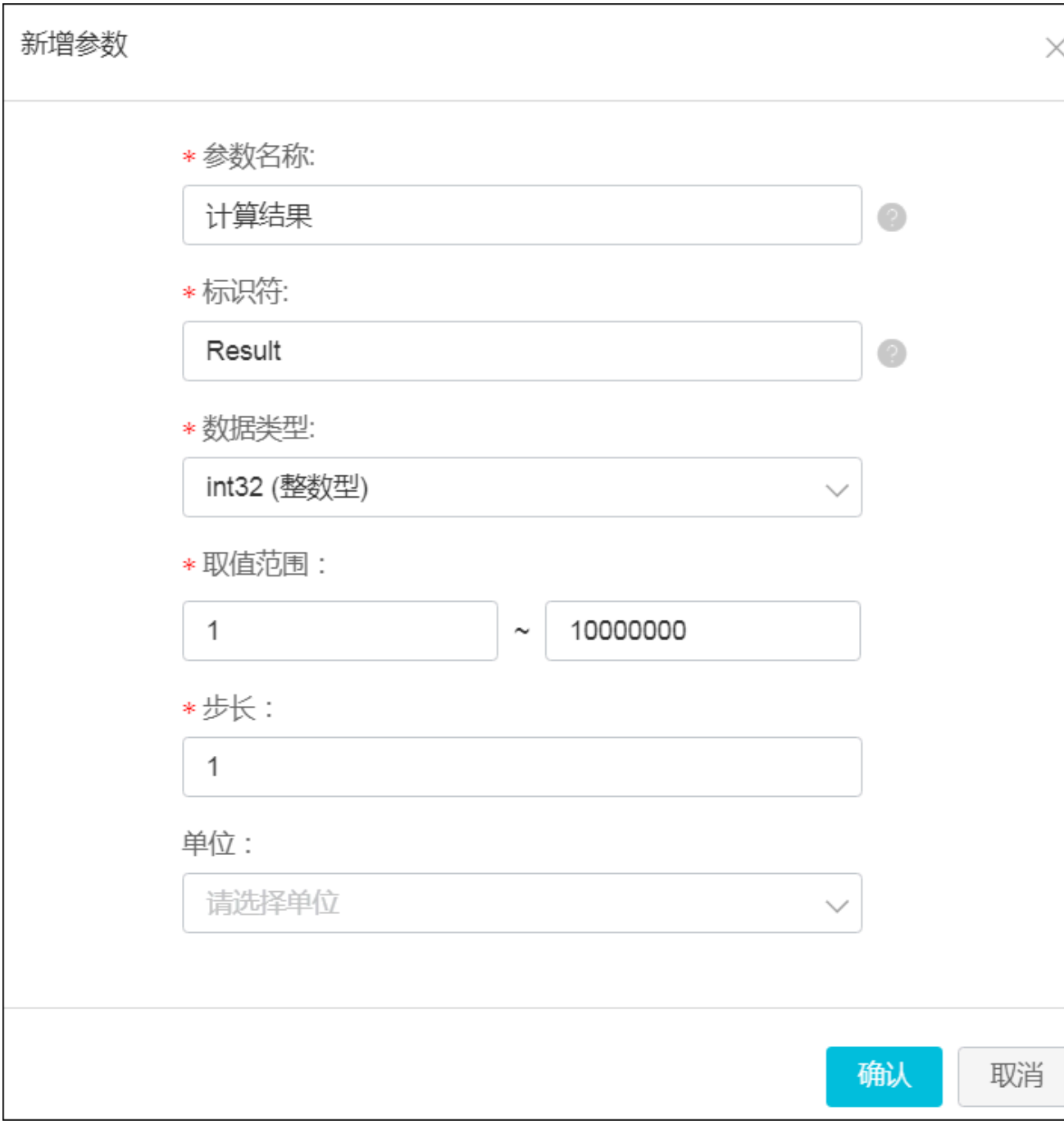
- **输入参数中，数值A定义如下。**



- **输入参数中，数值B定义如下。**

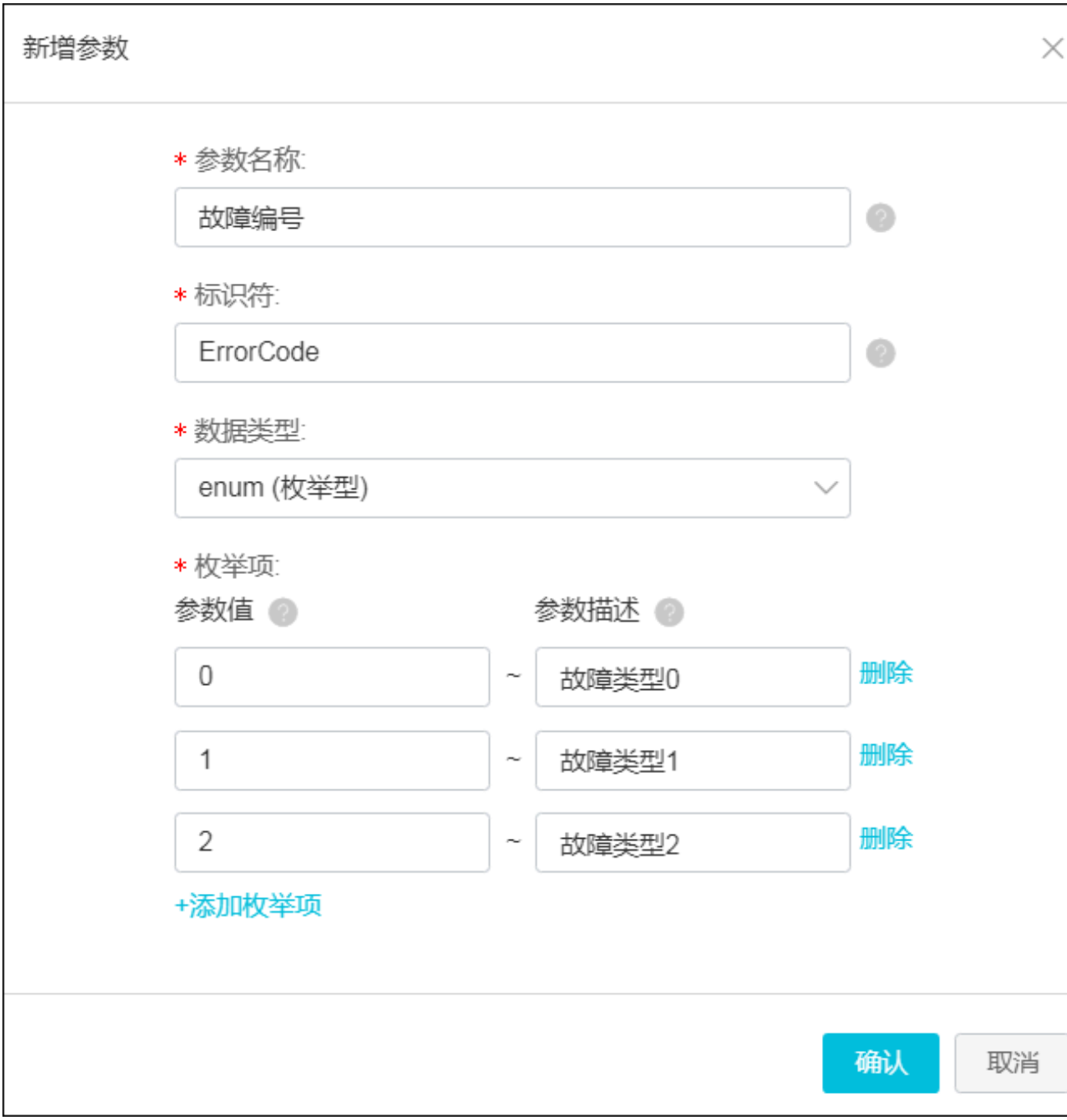

新增参数
* 参数名称:
数值B
* 标识符:
NumberB
* 数据类型:
int32 (整数型)
* 取值范围：
1
~
1000000
* 步长：
1
单位：
请选择单位
确认
取消

- **输出参数为计算结果。**

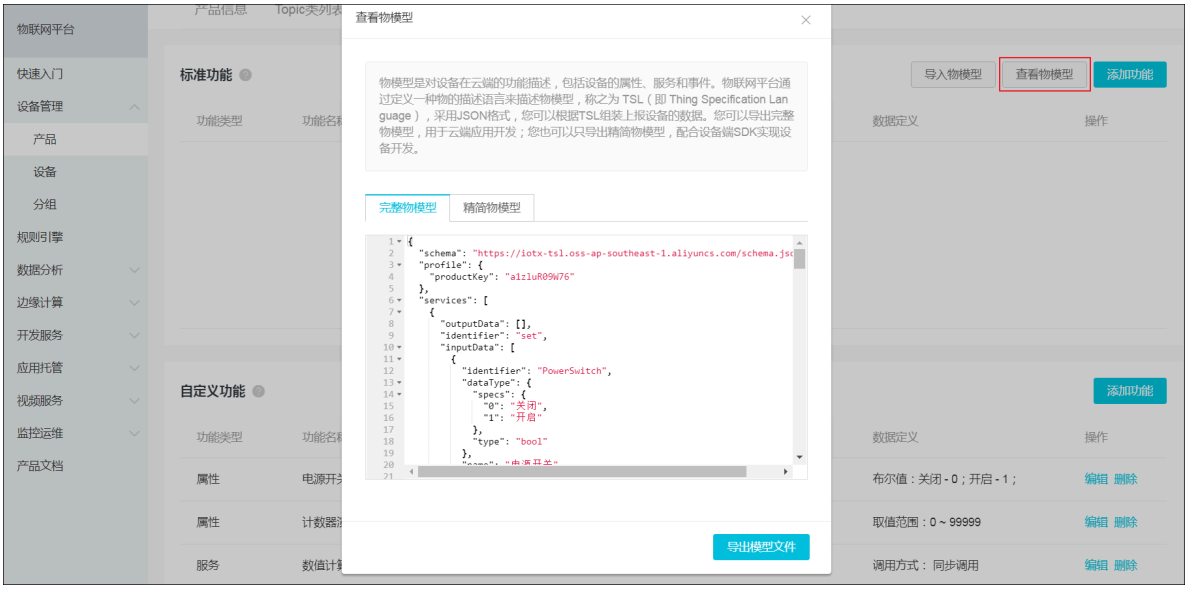

新增参数
* 参数名称:
计算结果
* 标识符:
Result
* 数据类型:
int32 (整数型)
* 取值范围：
1
~
10000000
* 步长：
1
单位：
请选择单位
确认
取消

7. 如下图所示，将故障定义为事件。完成后单击确认。

- 输出参数为故障编号。


新增参数
* 参数名称:
故障编号
* 标识符:
ErrorCode
* 数据类型:
enum (枚举型)
* 枚举项:
参数值
参数描述
0
~
故障类型0
删除
1
~
故障类型1
删除
2
~
故障类型2
删除
+添加枚举项
确认
取消

8. 单击查看物模型，在完整物模型栏下，可看到该产品的完整物模型JSON文件。



后续步骤

#unique_13

1.4 建立设备与平台的连接

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。在这一步里，我们使用平台提供的样例程序`linkkit-example-solo`模拟设备进行开发，实现设备与物联网平台的通信。

背景信息

- 本示例使用Linux下的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、git-2.7.4、gcc-5.4.0、gcov-5.4.0、lcov-1.12、bash-4.3.48、tar-1.28和mingw-5.3.1。

  可以使用如下命令行安装：

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

操作步骤

1. 登录Linux虚拟机。
2. 获取Link Kit SDK。
3. 使用unzip命令解压压缩包。

4. 设备身份信息将通过HAL调用返回给SDK。因此，需要将*wrappers/os/ubuntu/HAL_OS_linux.c*中的设备证书信息修改为#unique_15步骤中创建的设备证书，完成后保存退出。

   如下所示，填入ProductKey、DeviceName和DeviceSecret。设备使用该证书进行身份认证并连接物联网平台。

   > **说明:**
   > 快速入门中使用#unique_16的认证方式，ProductSecret可以不填。

   ```
   #ifdef DEVICE_MODEL_ENABLED
       char _product_key[IOTX_PRODUCT_KEY_LEN + 1]       = "a1zluR09
   ***";
       char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "";
       char _device_name[IOTX_DEVICE_NAME_LEN + 1]       = "device1";
       char _device_secret[IOTX_DEVICE_SECRET_LEN + 1]   = "ynNudfEadS
   doy7RMUJD9WZhEvd7****";
   ```

5. 在SDK根目录，执行make命令，完成样例程序的编译。

   ```
   $ make distclean
   $ make
   ```

   生成的样例程序`linkkit-example-solo`存放在*./output/release/bin*目录下。

6. 运行样例程序。

   ```
   ./output/release/bin/linkkit-example-solo
   ```

   在物联网平台控制台上，设备状态显示为在线，则表示设备与物联网平台成功连接。

   设备上线成功后，会自动向物联网平台上报消息。您可以通过查看日志，获取具体内容。

后续步骤

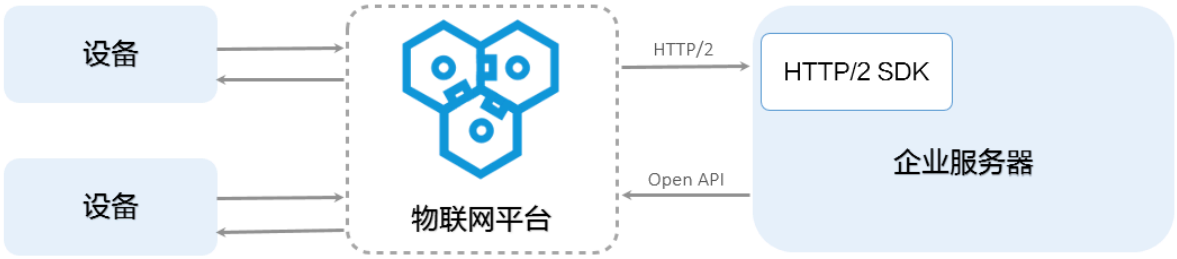
服务端订阅设备消息

1.5 服务端订阅设备消息

设备连接物联网平台后，数据直接上报至物联网平台。平台上的数据可以通过HTTP/2通道流转至您的服务器。这一步中，我们将配置HTTP/2服务端订阅功能。您的服务器可以通过接入HTTP/2 SDK，接收设备数据。

背景信息

服务端订阅设备消息流程图。



操作步骤

1. 在物联网平台控制台中，为产品配置服务端订阅。

   a) 在产品管理页，单击已创建的测试产品对应的查看按钮。

   b) 在产品详情页，单击服务端订阅 > 设置。

   c) 勾选要订阅的消息类型，然后单击保存。

消息类型	说明
设备上报消息	包括设备上报的自定义数据和物模型数据（包括属性上报、事件上报、属性设置响应和服务调用响应）。
设备状态变化通知	指当产品下设备的状态发生变化时，系统发出的通知消息。例如，设备上下线消息。
设备生命周期变更	包括设备创建、删除、禁用和启用的通知消息。
网关发现子设备上报	网关可以将发现的子设备信息上报给物联网平台。需要网关上的应用程序支持。 网关产品特有消息类型。
设备拓扑关系变更	指子设备和网关之间的拓扑关系建立和解除消息。 网关产品特有消息类型。

2. 接入HTTP/2 SDK。

   如果您使用Apache Maven来管理Java项目，需在项目的pom.xml文件中加入以下依赖项。

   > **说明:**
   > 请确保JDK版本为8。HTTP/2更多信息，请参见#unique_19。

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.3</version>
</dependency>

<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.7.1</version>
```

```
</dependency>
```

3. 基于您的阿里云账号AccessKey进行身份认证，建立SDK与物联网平台的连接。

```
// 阿里云accessKey
        String accessKey = "xxxxxxxxxxxxxxx";
        // 阿里云accessSecret
        String accessSecret = "xxxxxxxxxxxxxxx";
        // regionId
        String regionId = "cn-shanghai";
        // 阿里云uid
        String uid = "xxxxxxxxxxxx";
        // endPoint:  https://${uid}.iot-as-http2.${region}.aliyuncs
.com
        String endPoint = "https://" + uid + ".iot-as-http2." +
regionId + ".aliyuncs.com";

        // 连接配置
        Profile profile = Profile.getAccessKeyProfile(endPoint,
regionId, accessKey, accessSecret);

        // 构造客户端
        MessageClient client = MessageClientFactory.messageClient(
profile);

        // 数据接收
        client.connect(messageToken -> {
            Message m = messageToken.getMessage();
            System.out.println("receive message from " + m);
            return MessageCallback.Action.CommitSuccess;
        });
```

参数及获取方式：

参数	获取途径
accessKey	您的账号AccessKey ID。 登录阿里云控制台，将光标移至账号头像上，然后单击accesskeys，跳转至用户信息管理页，即可获取。
accessSecret	您的账号AccessKey Secret。获取方式同accessKey。
uid	您的账号ID。 用主账号登录阿里云控制台，单击账号头像，跳转至账号管理控制台，即可获取账号UID。
regionId	您物联网平台服务所在地域代码。 在物联网平台控制台页，右上方即可查看地域（Region）。RegionId 的表达方法，请参见#unique_20。

4. 确认HTTP/2 SDK可以接收到设备消息。

   若监听成功，可以通过SDK的消息回调获得以下数据。

参数	说明
messageId	物联网平台生成的消息ID。
topic	消息来源Topic。
payload	消息数据。请参见#unique_21。
generateTime	消息生成时间戳，以毫秒表示。
qos	· 0：消息最多投递1次。 · 1：消息最少投递1次。

1.6 设备接收云端指令

设备成功上报消息后，您可以尝试从云端下发指令到设备端。本文档讲解了如何通过云端应用调用SetDeviceProperty接口，设置设备属性值。同时，设备成功接收云端指令。

操作步骤

1. 在Maven项目中加入依赖项，导入阿里云IoT 云端SDK。

   IoT Java SDK的Maven依赖坐标：

```
<!-- https://mvnrepository.com/artifact/com.aliyun/aliyun-java-sdk-
iot -->
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-iot</artifactId>
    <version>6.8.0</version>
</dependency>
```

   阿里云公共包依赖坐标：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.5.1</version>
</dependency>
```

2. 初始化SDK。

   此处地域与产品地域保持一致，使用华东2。

```
String accessKey = "<your accessKey>";
String accessSecret = "<your accessSecret>";
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "Iot", "iot
.cn-shanghai.aliyuncs.com");
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai",
accessKey, accessSecret);
```

```
DefaultAcsClient client = new DefaultAcsClient(profile);
```

3. 云端应用向设备下发属性设置指令，将开关属性设置为1。

   示例如下：

```
SetDevicePropertyRequest request = new SetDevicePropertyRequest();
request.setProductKey("a1zluR09W76");
request.setDeviceName("device1");
JSONObject itemJson = new JSONObject();
itemJson.put("PowerSwitch", 1);
request.setItems(itemJson.toString());

try {
    SetDevicePropertyResponse response = client.getAcsResponse(
request);
    System.out.println(response.getRequestId() + ", success: " +
response.getSuccess());
} catch (ClientException e) {
    e.printStackTrace();
}
```

说明：

属性设置的具体调用方法，请参见SetDeviceProperty。

预期结果

若设备成功接收到云端下发的信息，日志示例如下：

```
< {
<     "method": "thing.service.property.set",
<     "id": "432801169",
<     "params": {
<         "PowerSwitch": 1
<     },
<     "version": "1.0.0"
< }

user_report_reply_event_handler.94: Message Post Reply Received,
Message ID: 646, Code: 200, Reply: {}
user_property_set_event_handler.114: Property Set Received, Request:
 {"PowerSwitch":1}

> {
>     "id": "647",
>     "version": "1.0",
>     "params": {
>         "PowerSwitch": 1
>     },
>     "method": "thing.event.property.post"
> }
```

2 使用MQTT.fx接入物联网平台

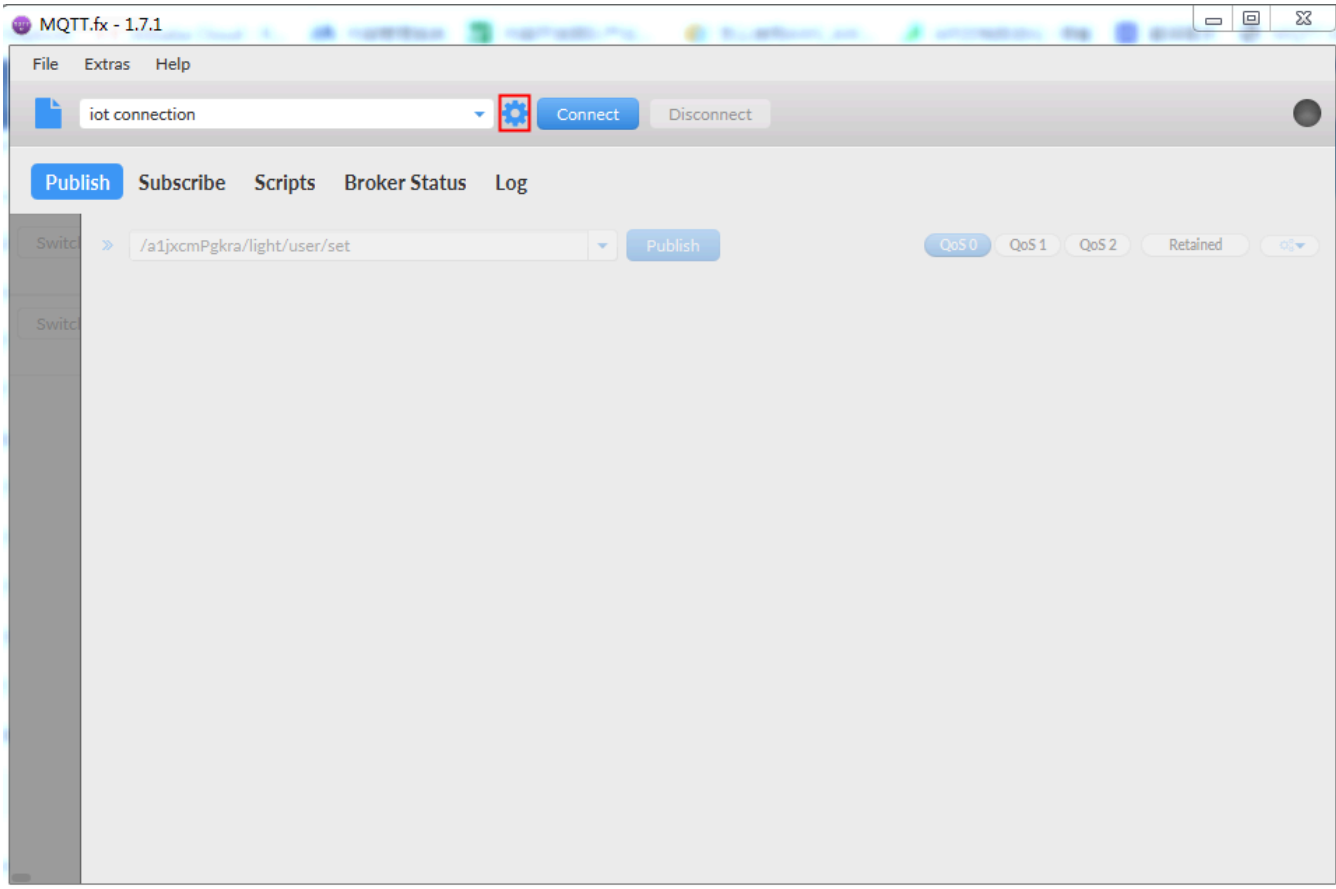
本文档以MQTT.fx为例，介绍使用第三方软件以MQTT协议接入物联网平台。MQTT.fx是一款基于Eclipse Paho，使用Java语言编写的MQTT客户端工具。支持通过Topic订阅和发布消息。

前提条件

已在物联网平台控制台创建产品和设备，并获取设备证书信息（ProductKey、DeviceName和DeviceSerect）。创建产品和设备具体操作细节，请参考创建产品、#unique_25和#unique_26。

MQTT.fx接入

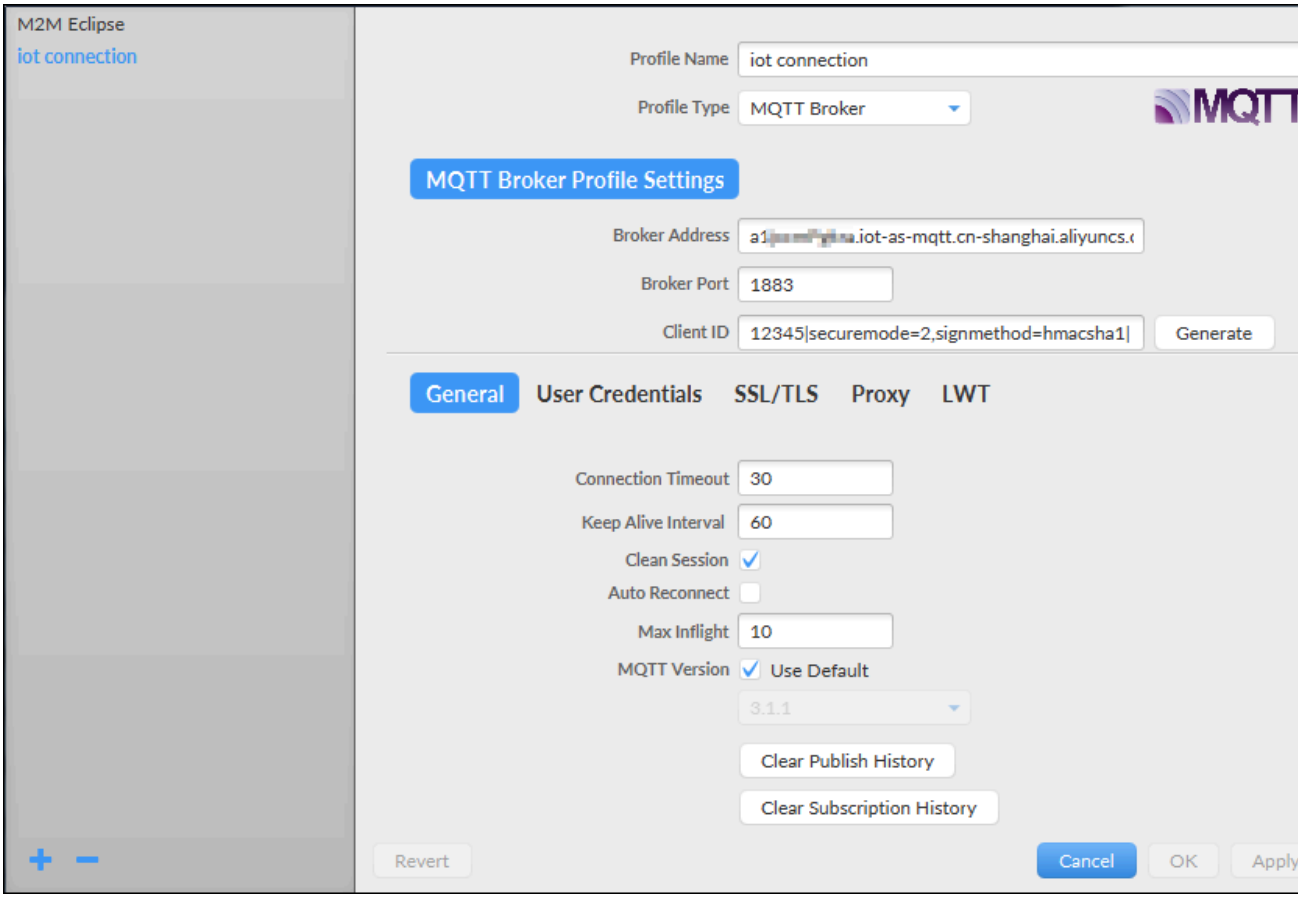
1. 下载并安装MQTT.fx软件。请访问MQTT.fx官网。
2. 打开MQTT.fx软件，单击设置图标。



3. 设置连接参数。物联网平台目前支持两种连接模式，不同模式设置参数不同。

   - TCP直连：Client ID中securemode=3，无需设置SSL/TLS信息。
   - TLS直连：Client ID中securemode=2，需要设置SSL/TLS信息。

   a. 设置基本信息。

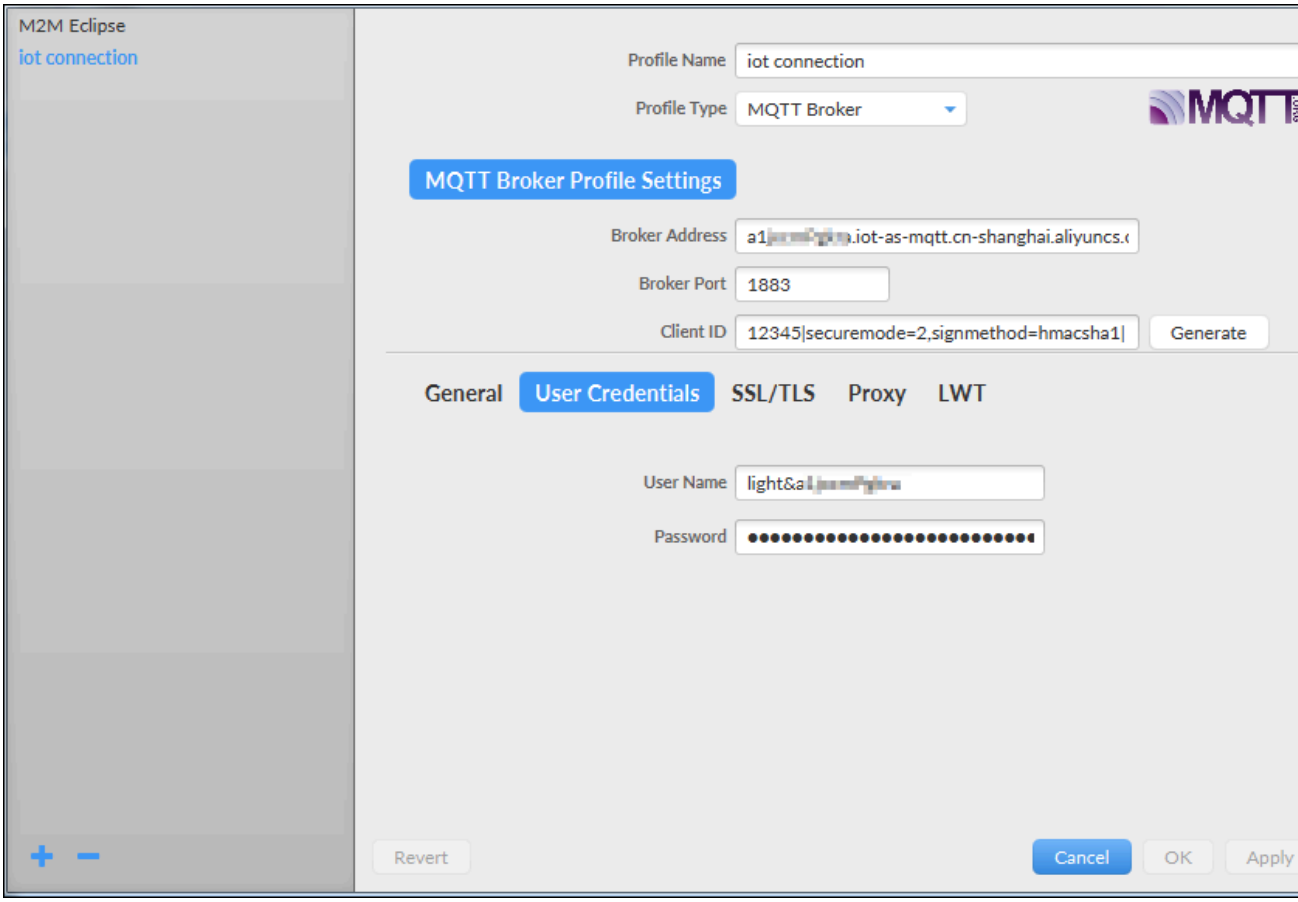


参数	说明
Profile Name	输入您的自定义名称。
Profile Type	选择为 MQTT Broker。
MQTT Broker Profile Settings	
Broker Address	连接域名。 格式：`${YourProductKey}.iot-as-mqtt.${region}.aliyuncs.com`。 其中，*${region}*需替换为您物联网平台服务所在地域的代码。地域代码，请参见#unique_27。如：alxxxxxxxxxx.iot-as-mqtt.cn-shanghai.aliyuncs.com。
Broker Port	设置为1883。

参数	说明
Client ID	填写mqttClientId，用于MQTT的底层协议报文。 格式固定：`${clientId}\|securemode=3,signmethod=hmacsha1\|`。 完整示例：`12345\|securemode=3,signmethod=hmacsha1\|`。 其中， · *${clientId}*为设备的ID信息。可取任意值，长度在64字符以内。建议使用设备的MAC地址或SN码。 · *securemode*为安全模式，TCP直连模式设置为securemode=3，TLS直连为securemode=2。 · *signmethod*为算法类型，支持hmacmd5和hmacsha1。 说明: 输入Client ID信息后，请勿单击Generate。

参数	说明
General General栏目下的设置项可保持系统默认，也可以根据您的具体需求设置。	

b. 单击User Credentials，设置User Name和Password。

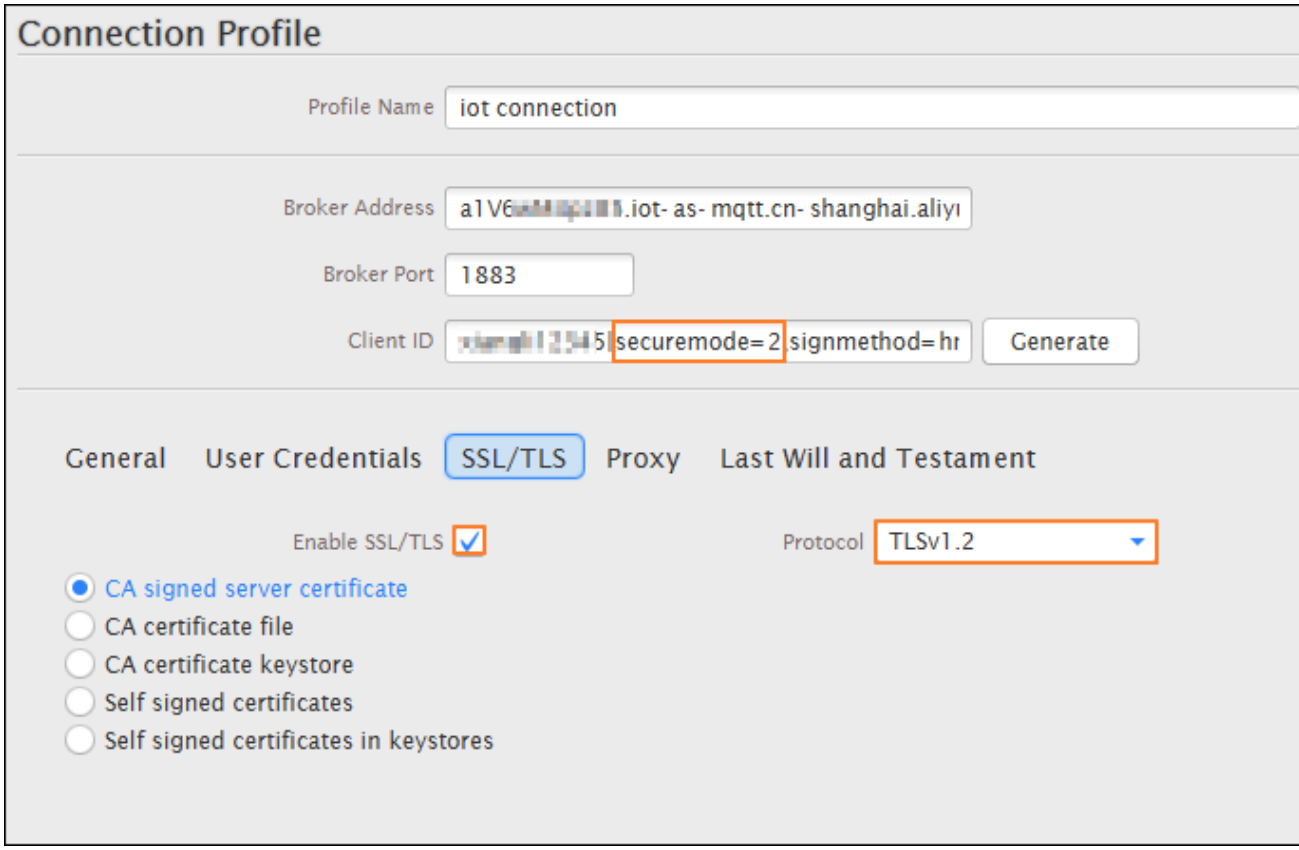


参数	说明
User Name	由设备名DeviceName、符号（&）和产品ProductKey组成。 固定格式：`${YourDeviceName}&${YourPrductKey}`。 完整示例如：device&alxxxxxxxxxx。

参数	说明
Password	密码由参数值拼接加密而成。 您可以使用物联网平台提供的生成工具自动生成Password，也可以手动生成Password。 · 单击下载Password生成小工具。 使用Password生成小工具的输入参数： - productKey：设备所属产品Key。可在控制台设备详情页查看。 - deviceName：设备名称。可在控制台设备详情页查看。 - deviceSecret：设备密钥。可在控制台设备详情页查看。 - timestamp：（可选）时间戳。 - clientId：设备的ID信息，与Client ID中*${clientId}*一致。 - method：选择签名算法类型，与Client ID中*signmethod*确定的加密方法一致。 · 手动生成方法如下： A. 拼接参数。 提交给服务器的clientId、deviceName、productKey和timestamp（timestamp为非必选参数）参数及参数值依次拼接。 本例中拼接结果为：clientId12345deviceNamedeviceproductKeyalxxxxxxxxx B. 加密。 通过Client ID中确定的加密方法，使用设备deviceSecret，将拼接结果加密。 假设设备的deviceSecret值为abc123，加密计算格式为hmacsha1(abc123,clientId12345deviceNamedeviceproductKeyalxxxxxxxxxx)

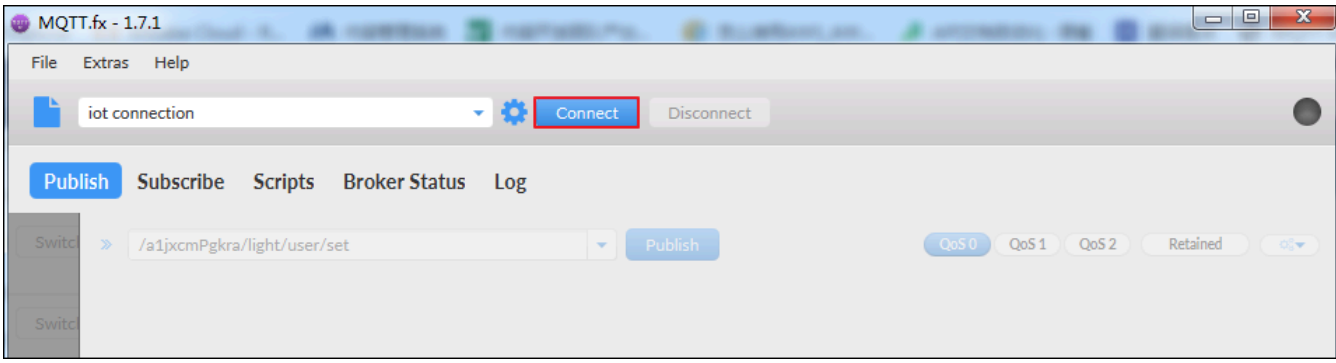
c. （可选）TCP直连模式（即securemode=3）下，无需设置SSL/TLS信息，直接进入下一步。

TLS直连模式（即securemode=2）下，需要选择SSL/TLS，勾选 Enable SSL/TLS，设置Protocol。建议Protocol选择为TLSv1.2。



d. 填写完成后，单击OK。

4. 设置完成后，单击 Connect进行连接。

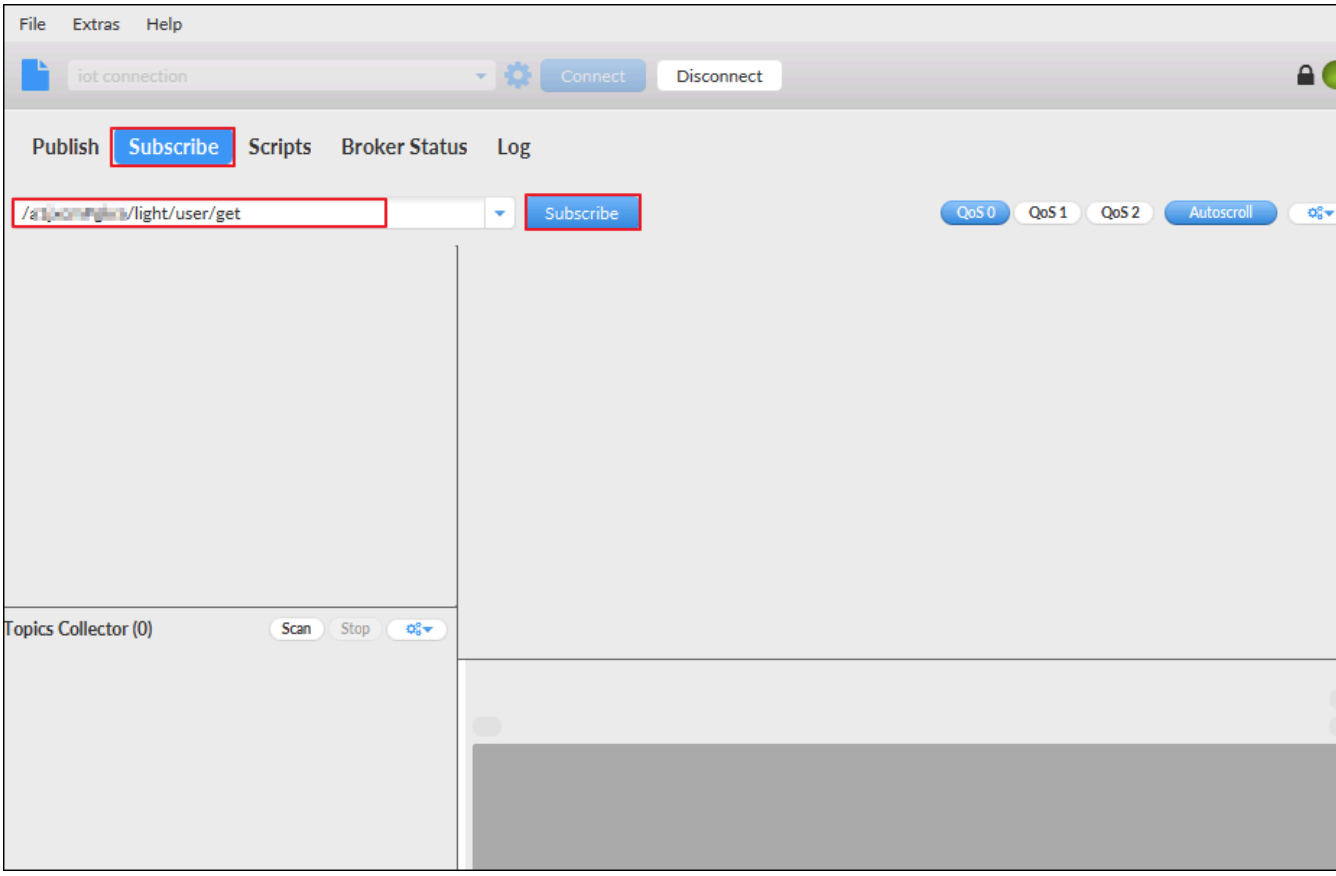


下行通信测试

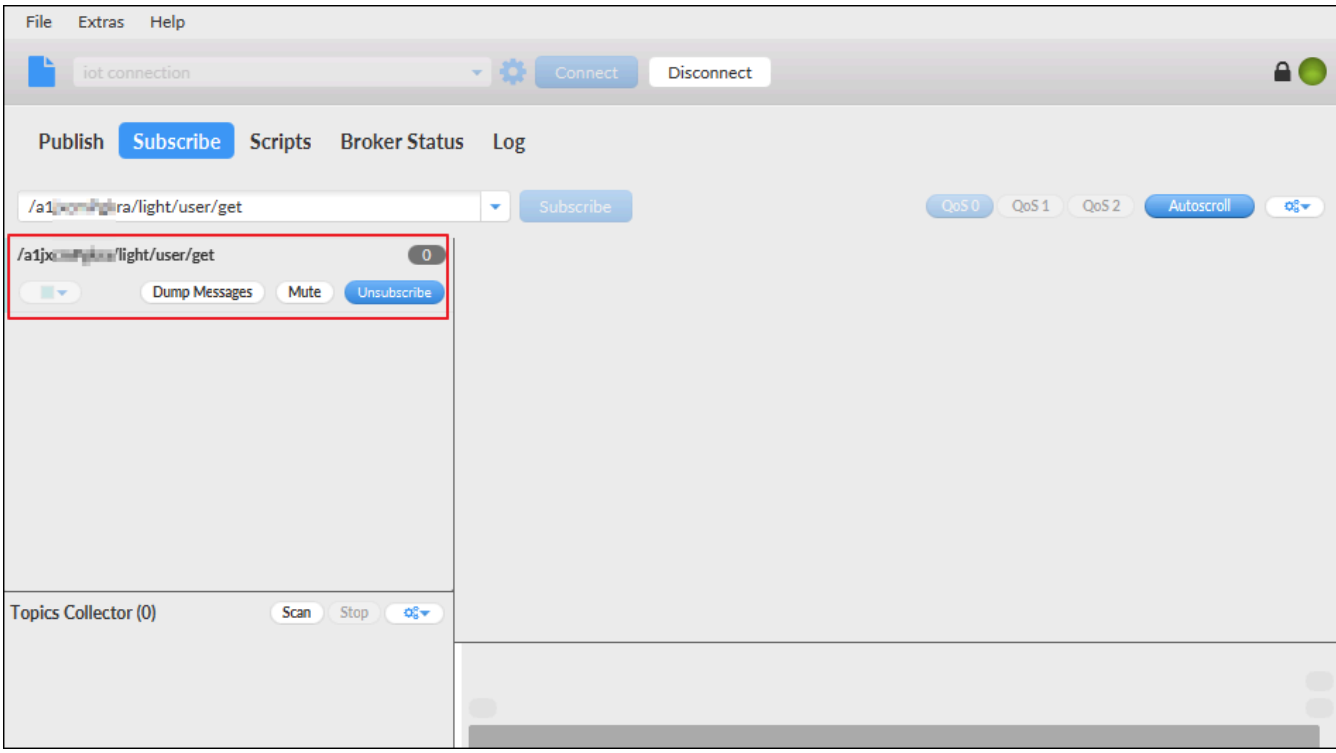
从物联网平台发送消息，在MQTT.fx上接收消息，测试MQTT.fx与物联网平台连接是否成功 。

1. 在MQTT.fx上，单击Subscribe。

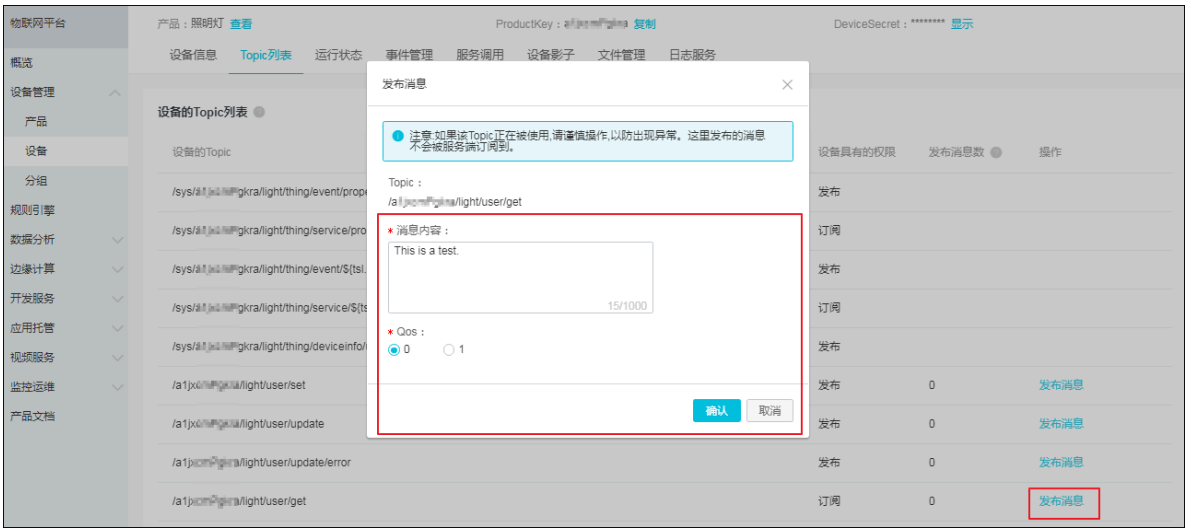
2. 输入一个设备具有订阅权限的Topic，单击Subscribe，订阅这个Topic。



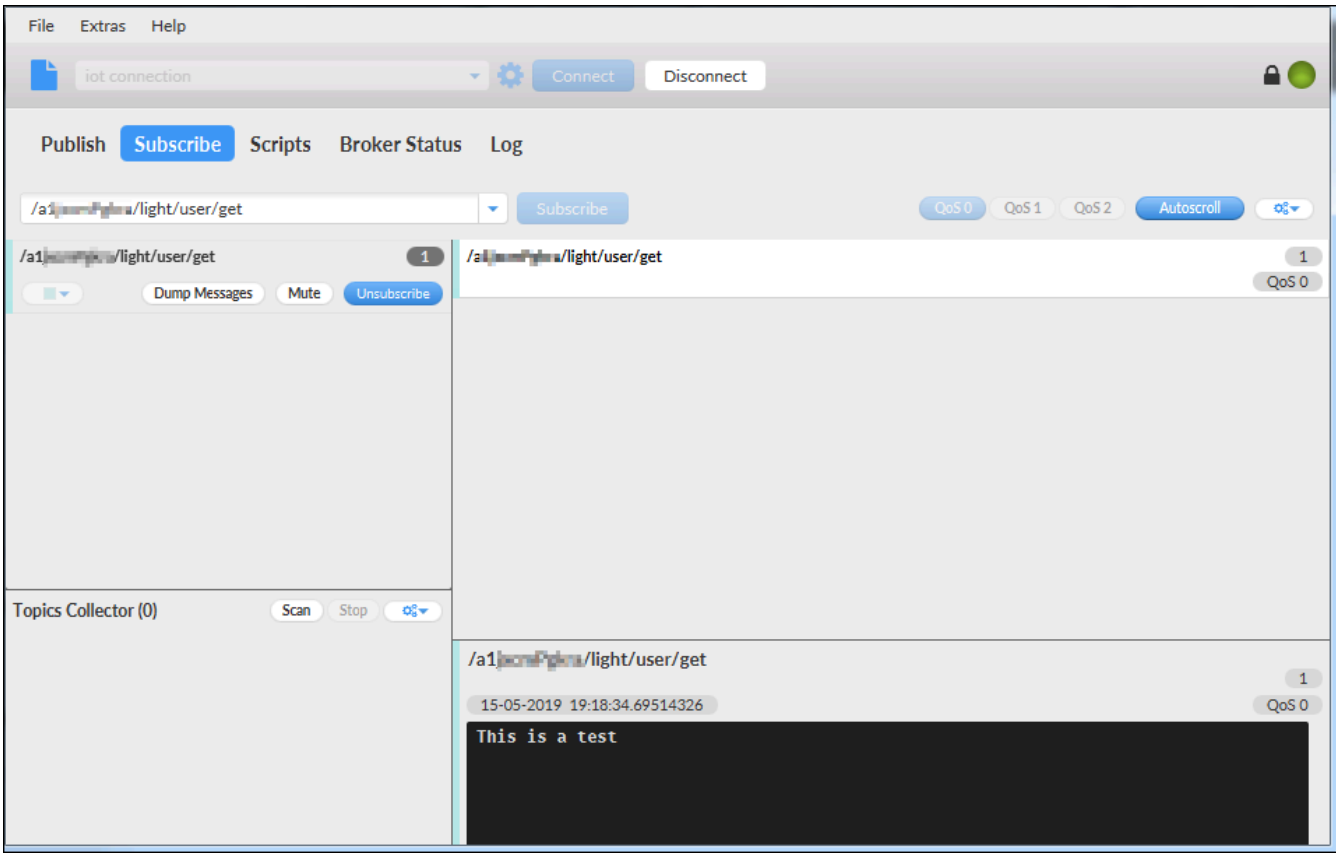
订阅成功后，该Topic将显示在列表中。



3. 在物联网平台控制台中，该设备的设备详情页，Topic列表下，单击已订阅的Topic对应的发布消息操作按钮。
4. 输入消息内容，单击确认。



5. 回到MQTT.fx上，查看是否接收到消息。

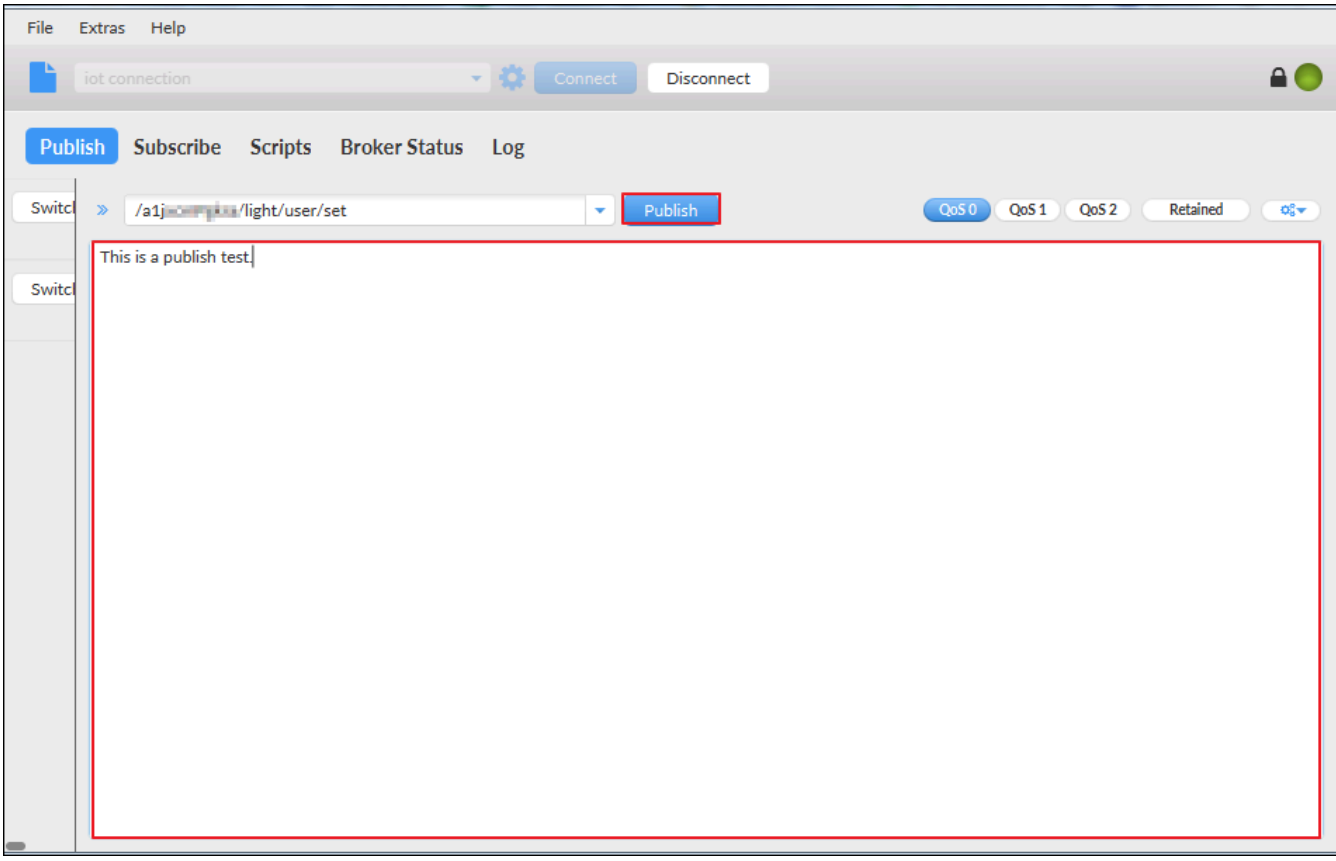


上行通信测试

在MQTT.fx上发送消息，通过查看设备日志，测试MQTT.fx与物联网平台连接是否成功。

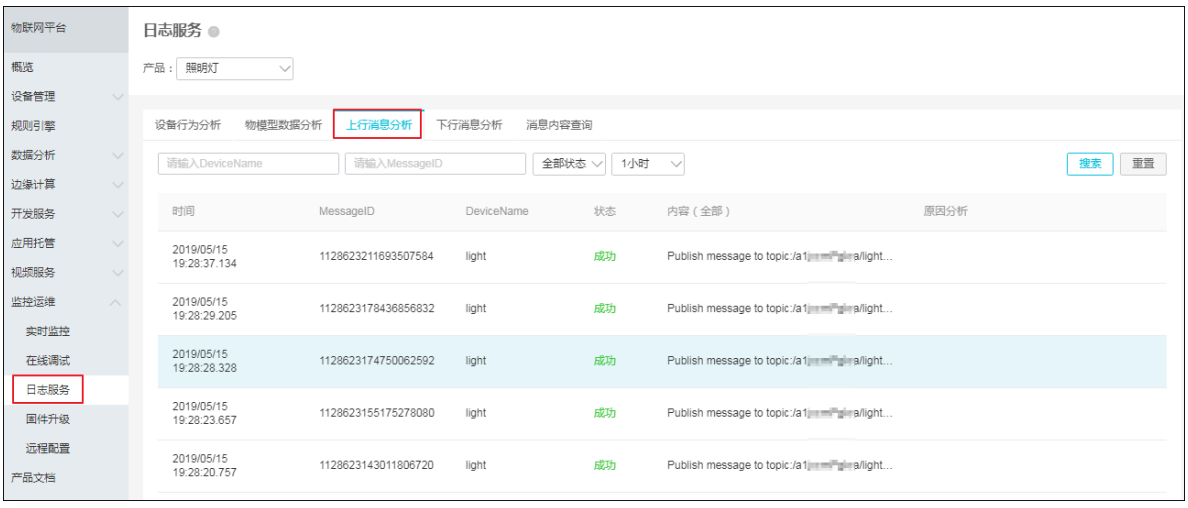
1. 在MQTT.fx上，单击Publish。

2. 输入一个设备具有发布权限的Topic，和要发送的消息内容，单击Publish，向这个Topic推送一条消息。



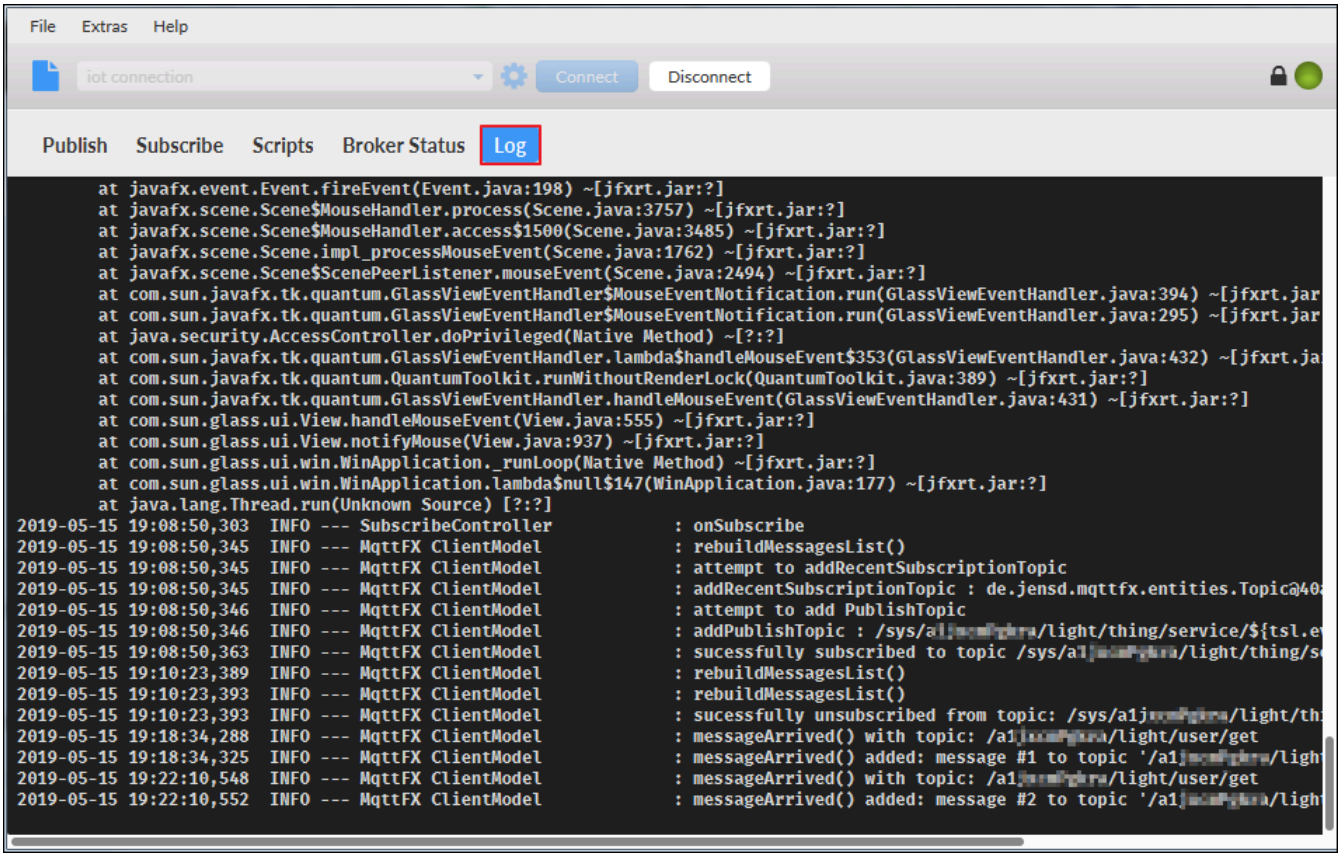
3. 在物联网平台控制台中，该设备的设备详情 > 日志服务 > 上行消息分析栏下，查看上行消息。

您还可以复制MessageID，在消息内容查询中，选择原始数据查看具体消息内容。



查看日志

在MQTT.fx上，单击Log查看操作日志和错误提示日志。


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

3 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HTTP/2 SDK获取设备上报消息；服务端通过调用云端API 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

本文示例中，设备端和云端均使用Java语言的SDK，需先准备Java开发环境。可从Java 官方网站下载、安装Java开发环境。

新建项目，添加以下Maven依赖，导入阿里云设备端SDK和云端SDK。

```
<dependencies>
 <dependency>
     <groupId>com.aliyun.alink.linksdk</groupId>
     <artifactId>iot-linkkit-java</artifactId>
     <version>1.2.0.1</version>
     <scope>compile</scope>
 </dependency>
 < dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-core</artifactId>
      <version>3.7.1</version>
  </dependency>
  <dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-iot</artifactId>
      <version>6.9.0</version>
  </dependency>
  <dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.2</version>
</dependency>
```

```
</dependencies>
```

创建产品和设备

先在物联网平台创建产品，自定义Topic类，定义物模型，设置服务端订阅，并创建设备。

1. 登录物联网平台控制台。
2. 在左侧导航栏，单击设备管理 > 产品。
3. 单击创建产品，创建一个温度计产品。

   详细操作指导，请参见#unique_29。
4. 产品创建成功后，单击该产品对应的查看。
5. 在产品详情页的Topic类列表页签下，增加自定义Topic类。

   详细操作指导，请参见#unique_30。

   本示例中，定义了以下两个Topic类：

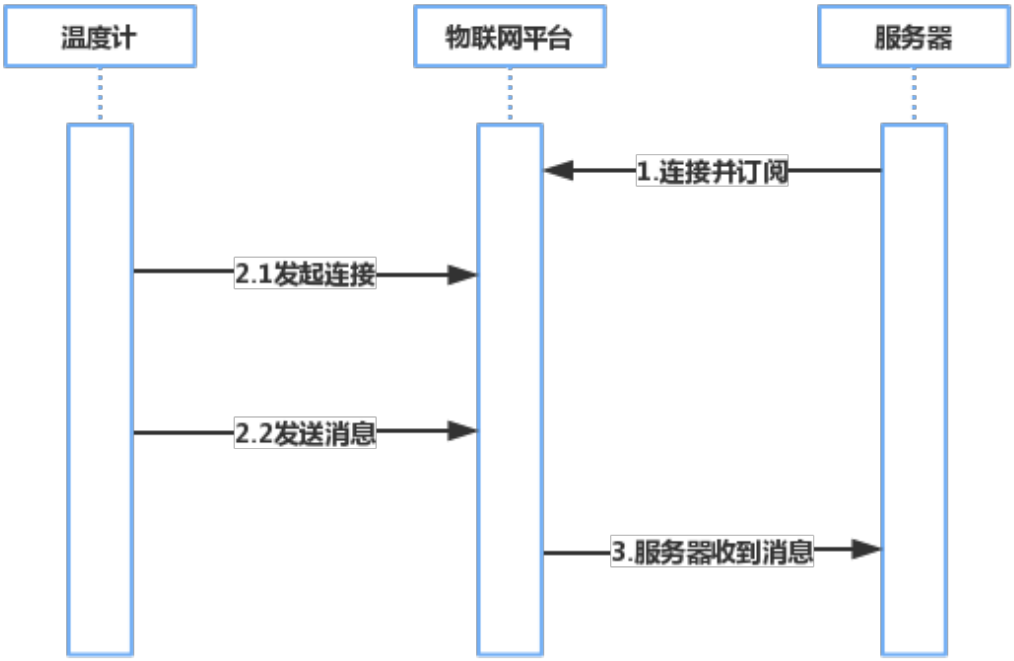
   - 设备发布消息Topic：/${productKey}/${deviceName}/user/devmsg，权限为发布。
   - 设备订阅消息Topic：/${productKey}/${deviceName}/user/cloudmsg，权限为订阅。
6. 在服务端订阅页签下，设置HTTP/2通道服务端订阅，订阅设备上报消息 。

   详细操作指导，请参见服务端订阅#unique_31。

7. 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。详细操作指导，请参见#unique_32。

设备发送消息给服务器

流程图：



- 配置服务端HTTP/2 SDK，设置Topic回调。

  配置服务端订阅后，服务器监听产品下所有设备的全部上报消息。

  - 配置HTTP/2 SDK接入物联网平台 。

```
// 身份认证
String accessKey = "您的阿里云accessKey";
String accessSecret = "您的阿里云accessSecret";
String uid = "您的阿里云账号ID";
String regionId = "您设备所处区域regionId";
String productKey = "您的产品productKey";
String deviceName = "您的设备名字deviceName";
String endPoint = "https://" + uid + ".iot-as-http2." + regionId +
 ".aliyuncs.com";
// 连接配置
Profile profile = Profile.getAccessKeyProfile(endPoint, regionId,
accessKey, accessSecret);

// 构造客户端
MessageClient client = MessageClientFactory.messageClient(profile
);

// 数据接收
client.connect(new MessageCallback() {
    @Override
    public Action consume(MessageToken messageToken) {
        Message m = messageToken.getMessage();
        System.out.println("\ntopic=" + m.getTopic());
        System.out.println("payload=" + new String(m.getPayload
()));
        System.out.println("generateTime=" + m.getGenerateTime());
```

```
            // 此处标记CommitSuccess已消费，IoT平台会删除当前Message，否则会
保留到过期时间
            return MessageCallback.Action.CommitSuccess;
        }
});
```

- 设置消息接收接口。

  本示例中，服务端订阅Topic：/${productKey}/${deviceName}/user/devmsg，因此设置该Topic的回调函数。

```
//定义回调函数
MessageCallback messageCallback = new MessageCallback() {
//获取消息并打印出来，服务器可以在此设置对应的动作
            @Override
            public Action consume(MessageToken messageToken) {
                Message m = messageToken.getMessage();
                log.info("receive : " + new String(messageToken.
getMessage().getPayload()));
                System.out.println(messageToken.getMessage());
                return MessageCallback.Action.CommitSuccess;
            }
        };
        //注册回调函数
        client.setMessageListener("/" + productKey + "/" +
deviceName + "/user/devmsg", messageCallback);
```

更多具体配置详情，请参见HTTP/2 SDK#unique_31。

· 配置设备端SDK发送消息。

  - 配置设备认证信息。

```
final String productKey = "XXXXXX";
final String deviceName = "XXXXXX";
final String deviceSecret = "XXXXXXXXX";
final String region = "XXXXXX";
```

  - 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。

```
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 IConnectSe
ndListener() {
         @Override
         public void onResponse(ARequest aRequest, AResponse
aResponse) {
             System.out.println("onResponse:" + aResponse.getData
());
         }

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

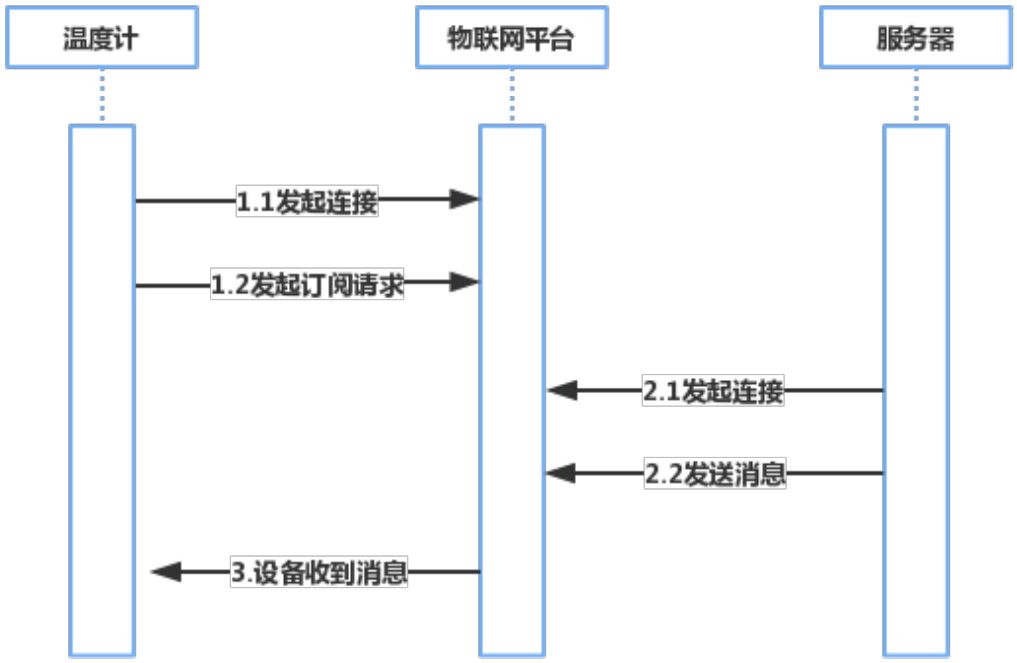
服务器收到消息如下：

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

```
generateTime=1558666546105}
```

服务器发送消息给设备

流程图：



- **配置设备端SDK订阅Topic。**

  设备认证配置、设置初始化连接参数、和初始化连接，请参见上一章节的设备端SDK配置。

  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 IConnectSu
bscribeListener() {
        @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 IConnectNotifyListen
er() {
        //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.getData()));
        }

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

        @Override
        public void onConnectStateChange(String s, ConnectState
connectState) {

        }
    };
    LinkKit.getInstance().registerOnNotifyListener(notifyListener);
}
```

- 配置云端SDK调用云端API #unique_33发布消息。
  - 设置身份认证信息。

    ```
    String regionId = "您设备所处区域regionId";
    String accessKey = "您的阿里云账号accessKey";
    String accessSecret = "您的阿里云账号accessSecret";
    final String productKey = "您的产品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：Demo

单击Pub/SubDemo，下载、查看本示例的完整代码Demo。

4 通过消息队列RocketMQ流转消息

物联网平台将设备上报的数据流转至消息队列RocketMQ的Topic中，然后，RocketMQ再将数据流转到您的服务器。

前提条件

- 已注册阿里云账号。
- 已开通阿里云物联网平台服务。

  如未开通，请登录物联网平台产品页，单击立即开通，根据页面提示，开通服务。
- 已开通消息队列RocketMQ服务。

  如未开通，请登录消息队列RocketMQ产品页，开通服务。

背景信息

建议架构：

方案优势：

通过消息队列RocketMQ消峰去谷，缓冲消息，减轻服务器同时接收大量设备消息的压力。

操作步骤

1. 登录物联网控制台，创建产品和设备。
   a) 左侧菜单栏选择设备管理 > 产品，单击创建产品，创建产品。本示例中，创建了名称为MQ_test的产品，且节点类型选择为设备。
   b) 在产品详情页面，自定义Topic类，用于设备上报数据。本示例中，定义的Topic类：`/${YourProductKey}/${YourDeviceName}/user/data`。
   c) 选择设备管理 > 设备 > 添加设备，创建设备。本示例中，创建了一个名称为MQdevice的设备。

2. 在消息队列RocketMQ控制台，创建Topic和消费者。

a) 登录消息队列RocketMQ控制台。

b) 创建一个实例。

c) 创建Topic。消息类型选择普通消息。

d) 创建Group ID。

e) 创建消息消费者。在服务器SDK中运行如下示例代码，然后，在RocketMQ控制台查看消费者状态，消费者是否处于在线状态，订阅关系是否一致。

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import java.util.Properties;
public class ConsumerTest {
    public static void main(String[] args) {
        Properties properties = new Properties();
        // 您在控制台创建的 Group ID
        properties.put(PropertyKeyConst.GROUP_ID, "XXX");
        // AccessKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.AccessKey, "${AccessKey
}");
        // SecretKey 阿里云身份验证，在阿里云服务器管理控制台创建
        properties.put(PropertyKeyConst.SecretKey, "${SecretKey
}");
        // 设置 TCP 接入域名，到控制台的实例基本信息中查看
        properties.put(PropertyKeyConst.NAMESRV_ADDR,
            "XXX");
        // 集群订阅方式（默认）
        // properties.put(PropertyKeyConst.MessageModel,
PropertyValueConst.CLUSTERING);
        // 广播订阅方式
        // properties.put(PropertyKeyConst.MessageModel,
PropertyValueConst.BROADCASTING);
        Consumer consumer = ONSFactory.createConsumer(properties);
        consumer.subscribe("iot_to_mq", "*", new MessageListener()
 { //订阅多个 Tag
            public Action consume(Message message, ConsumeContext
context) {
                System.out.println("Receive: " + message);
                return Action.CommitMessage;
            }
        });
        consumer.start();
        System.out.println("Consumer Started");
    }
}
```

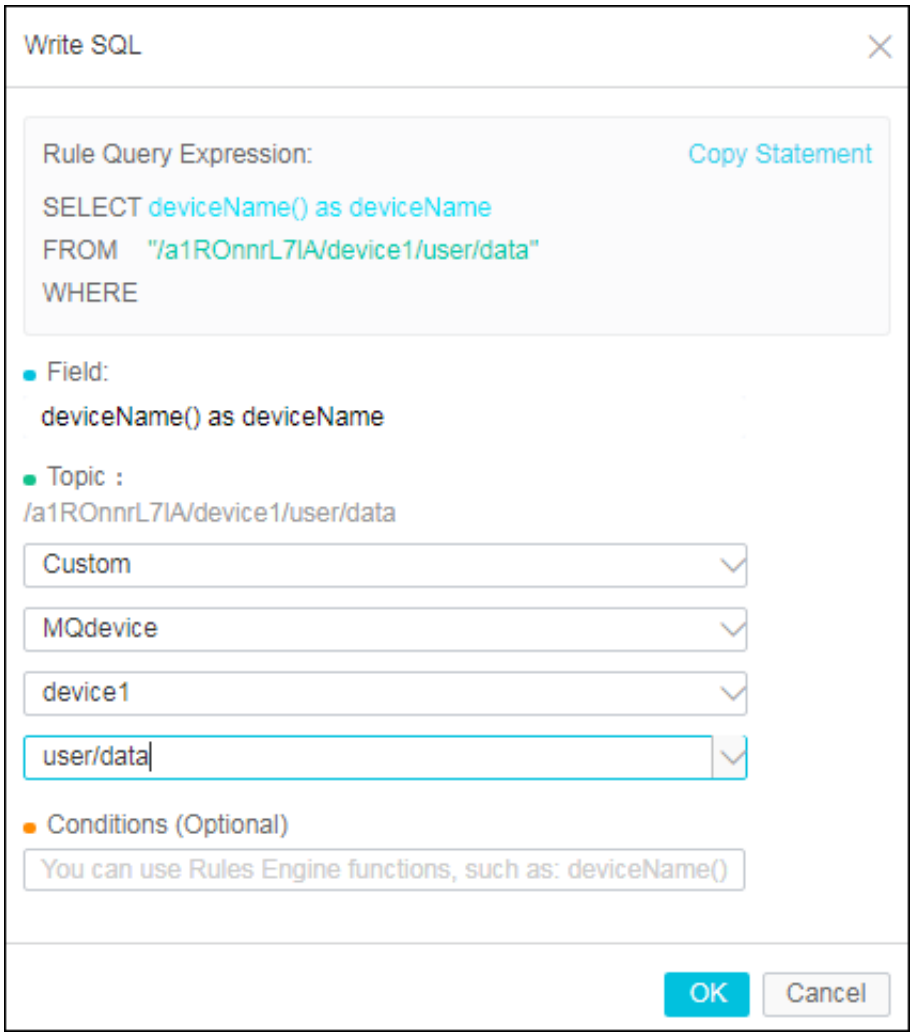
说明:

- 如何创建AccessKey和SecretKey，请参见创建AccessKey。
- RocketMQ详细操作指导，请参考消息队列RocketMQ文档消息队列RocketMQ文档

3. 在物联网平台控制台，设置数据流转规则，将设备上报的数据转发至消息队列（RocketMQ）。

   a) 单击规则引擎 > 创建规则，创建一条数据流转规则。数据格式选择为JSON。

   b) 设置数据处理SQL。



   c) 设置数据转发目的地。

d) 启动规则。

规则启动后，物联网平台会将规则SQL中定义的设备上报消息转发至消息队列（RocketMQ）的Topic中。

4. 使用Java SDK模拟设备，上报消息。

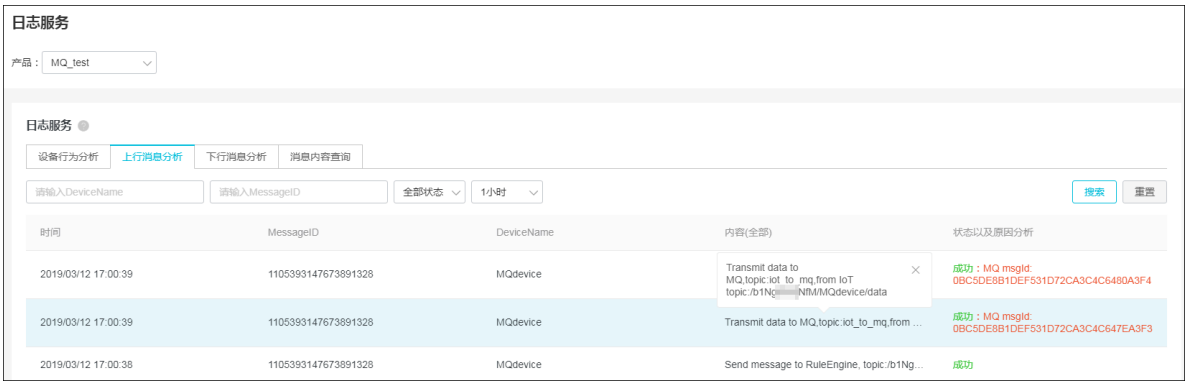
   a) 下载Java SDK Demo。

   b) 输入MQdevice的设备证书信息，包括ProductKey、DeviceName和DeviceSecret。

   c) 修改MQTT Topic为设备上报数据的Topic。本示例中，使用的Topic是`/{YourProductKey}/${YourDeviceName}/user/data`。

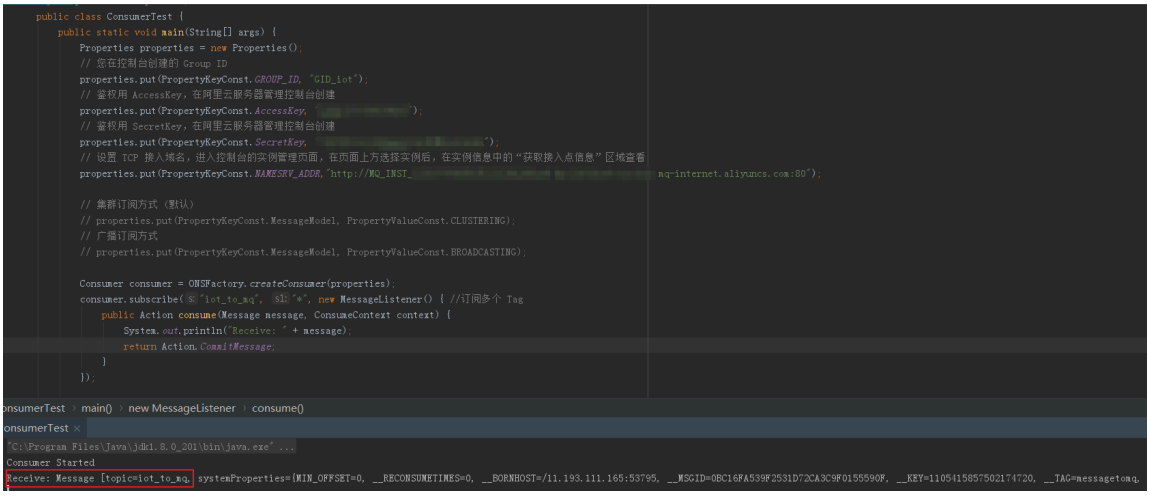
   d) 启动设备。

   在物联网平台控制台监控运维 > 日志服务中，查看该设备的日志信息，发现设备数据成功转发至RocketMQ。



5. 在RocketMQ控制台查看消息。

   a) 在本地运行订阅消息队列RocketMQ资源的代码。



   b) 在消息队列RocketMQ控制台，消息查询页面，按Topic或者Message ID查询消息，验证消息是否成功流转至消息队列RocketMQ中。

   RocketMQ接收到的消息类型：

```
{"deviceName()":"MQdevice"}
```