物联网开发平台

入门指引

腾讯云

文档目录

入门指引
公共实例快速入门
企业实例快速入门
企业实例入门概述
创建企业实例及产品设备
设备接入及上报数据
平台下发控制指令
平台转发消息至用户业务系统
快速体验平台
MQTT.fx 快速接入物联网开发平台
使用MQTT物模型接入平台
使用自定义透传上下行消息
云端控制设备入门
平台转发消息至用户 HTTP 服务

入门指引
公共实例快速入门

最近更新时间：2025-05-28 10:22:32

操作场景

在物联网开发平台创建智能灯产品和设备，并连接物联网开发平台，通过腾讯连连小程序绑定设备，进行远程控制灯的亮度、颜色、开关，并实时获取智能灯上报的数据。

前提条件

为了通过下面的步骤快速理解该业务场景，需要做好以下准备工作：

- 拥有一台物理或虚拟的 Linux 环境，可以编译、运行 light_demo 程序。
- light_demo 在 Linux 环境下测试和验证，主要基于 Ubuntu 16.04 版本，gcc-5.4（建议至少 gcc-4.7+）。

> **注意：**
> 物联网开发平台从2024年6月20日起，新注册物联网开发平台的用户需购买公共实例激活码才可使用公共实例，在此时间之前注册的用户并已开通公共实例的用户则不受影响，依然享有免费额度。若您需要使用公共实例，您可直接在 线上购买，单击“公共实例”，根据您的量产设备数量购买激活码数量。若用户有商业化的设备接入与量产需求，也可以通过接口的商务联系我们。

操作步骤

控制台操作

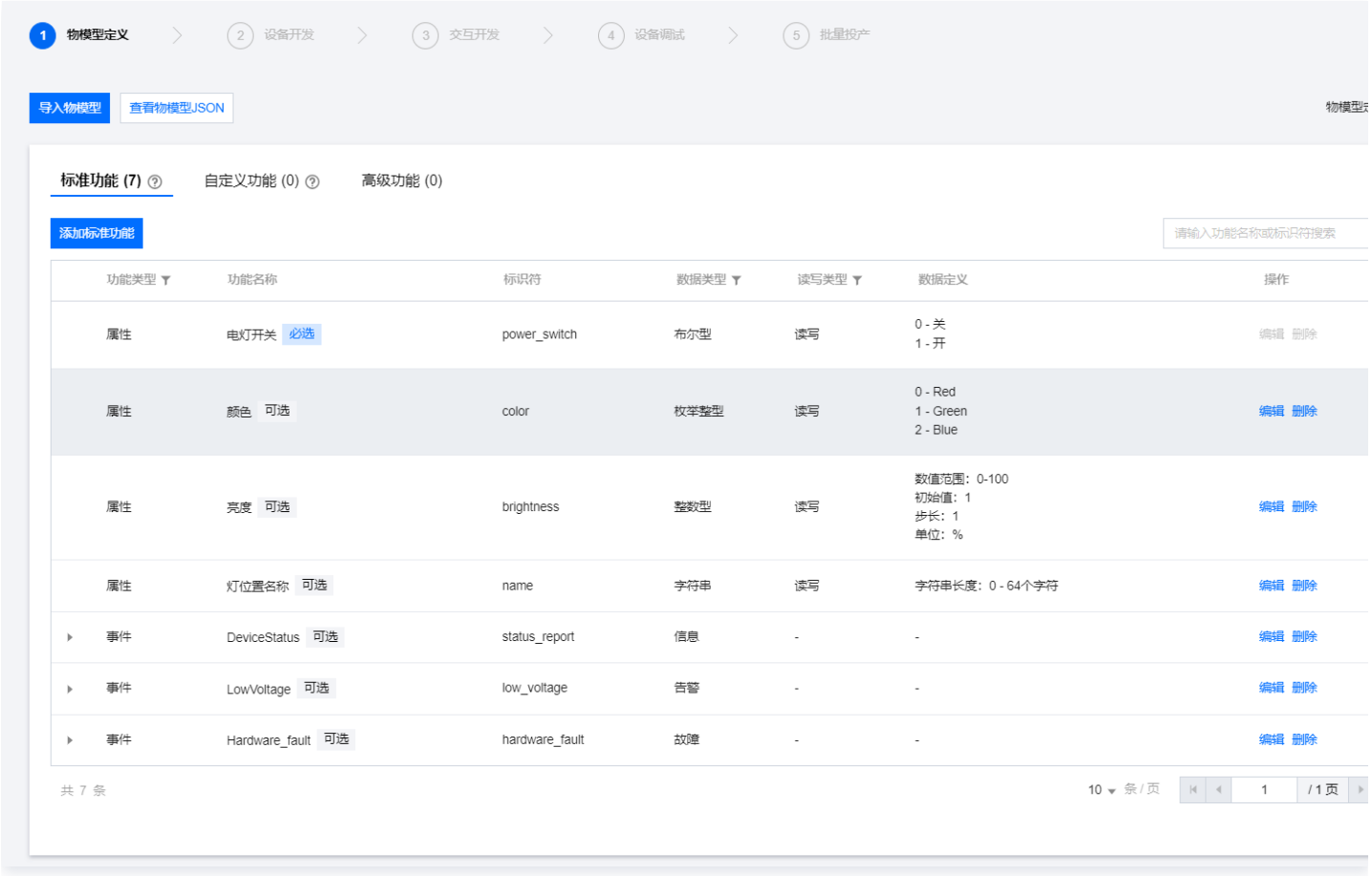
新建产品

1. 登录 物联网开发平台，选择**公共实例**。
2. 选择产品开发菜单，单击**新建产品**进入产品开发页面。
3. 在新建产品页面，填写产品基本信息。
   - 产品名称：输入“智能灯”或其他产品名称。
   - 产品品类：选择**智慧城市 > 公共事业 > 路灯照明**。
   - 设备类型：选择**设备**。
   - 通信方式：选择**Wi-Fi**。

4. 产品信息填写完成后，单击**新建产品**，即可完成新建产品。
5. 产品新建成功后，您可在产品列表页查看到“智能灯”。

定义产品物模型

选择“智能灯”类型后，系统会自动生成标准功能。



创建设备

在设备调试页面中，单击**新建设备**，设备名称是：dev001。

下载 Demo 程序

下载 lightdemo 例程

首先从 GitHub 下载，或执行下面的 git 命令。

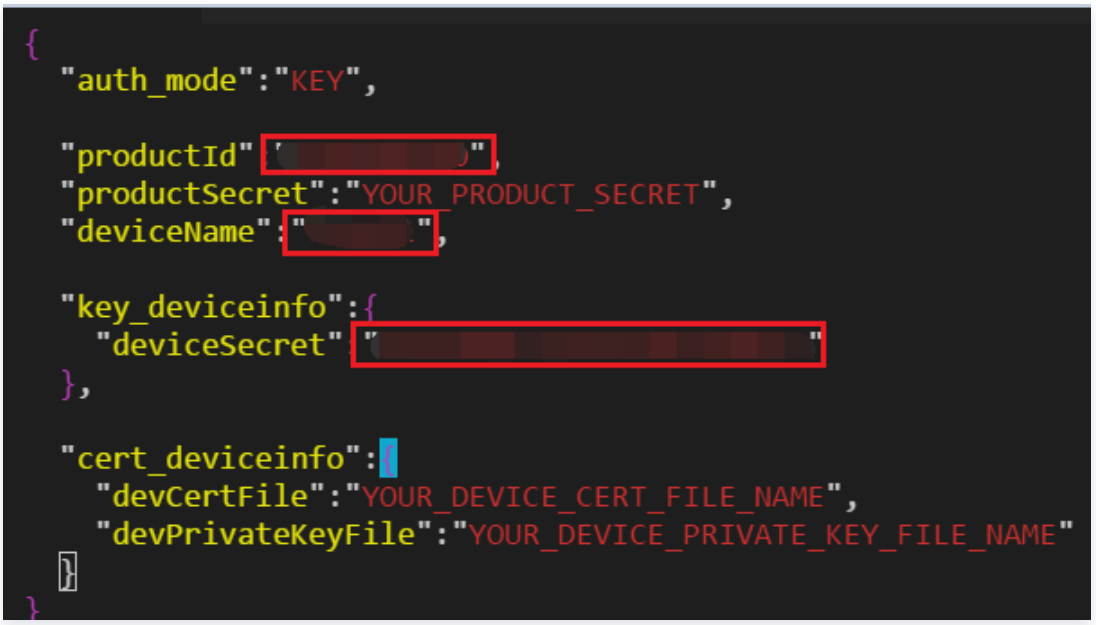
```
git clone https://github.com/tencentyun/qcloud-iot-explorer-sdk-embedded-c.git
```

修改 Demo 程序

上述 git 命令执行成功后，会生成一个 qcloud-iot-sdk-embedded-c 目录。

1. 进入 qcloud-iot-explorer-sdk-embedded-c 目录。
2. 修改该目录下的 device_info.json 文件。

```
> vi  device_info.json
```



3. 将上图红色线框中的数据分别替换为控制台产品在设备调试阶段的参数信息，单击选择**设备名称**进入“设备详情页”中，查看参数并保存。
   - 产品 ID： 将控制台的产品 ID，复制到上图 productId。
   - 设备名称： 将控制台的设备名称，复制到上图 deviceName。
   - 设备密钥：将控制台的设备密钥，复制到上图 deviceSecret。

编译

1. 上述配置信息修改完成后，即可编译。
2. 在 qcloud-iot-sdk-embedded-c 目录下执行以下命令进行编译。

```
./cmake_build.sh
```

3. 编译成功后，会在 output/release/bin 目录下生成 light_data_template_sample 执行文件。

运行 Demo 程序

1. 进入 `output/release/bin` 目录。
2. 输入 `./light_data_template_sample` 。
3. 运行成功后，系统输出示例如下：

```
INF|2019-05-07 21:51:33|device.c|iot_device_info_set(65): SDK_Ver: 3.0.0,
Product_ID: BKDDAHRGRX, Device_Name: dev001
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(204): Connecting to
/BKDDAHRGRX.iotcloud.tencentdevices.com/8883...
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(209): Setting up the
SSL/TLS structure...
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(251): Performing the
SSL/TLS handshake...
INF|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(269): connected with
/BKDDAHRGRX.iotcloud.tencentdevices.com/8883...
INF|2019-05-07 21:51:33|mqtt_client.c|IOT_MQTT_Construct(115): mqtt connect
with id: ZPEm9 success
DBG|2019-05-07 21:51:33|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(129):
topicName=$template/operation/result/BKDDAHRGRX/dev001|packet_id=35313|pUserdat
a=(null)
DBG|2019-05-07 21:51:33|shadow_client.c|_shadow_event_handler(63): shadow
subscribe success, packet-id=35313
INF|2019-05-07 21:51:33|light_data_template_sample.c|event_handler(222):
subscribe success, packet-id=35313
INF|2019-05-07 21:51:33|shadow_client.c|IOT_Shadow_Construct(173): Sync device
data successfully
INF|2019-05-07 21:51:33|light_data_template_sample.c|main(496): Cloud Device
Construct Success
DBG|2019-05-07 21:51:33|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(129):
topicName=$thing/down/event/BKDDAHRGRX/dev001|packet_id=35314|pUserdata=(null)
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=power_switch registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=color registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=brightness registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=name registered.
INF|2019-05-07 21:51:33|light_data_template_sample.c|main(517): Register data
template propertys Success
DBG|2019-05-07 21:51:33|shadow_client.c|IOT_Shadow_Get(384): GET Request
Document: {"clientToken":"BKDDAHRGRX-0"}
DBG|2019-05-07 21:51:33|mqtt_client_publish.c|qcloud_iot_mqtt_publish(337):
publish packetID=0|topicName=$template/operation/BKDDAHRGRX/dev001|payload=
{"type":"get", "clientToken":"BKDDAHRGRX-0"}
DBG|2019-05-07 21:51:33|shadow_client.c|_shadow_event_handler(63): shadow
subscribe success, packet-id=35314
INF|2019-05-07 21:51:33|light_data_template_sample.c|event_handler(222):
subscribe success, packet-id=35314
```

```
DBG|2019-05-07 21:51:33|shadow_client.c|_update_ack_cb(114): requestAck=0
DBG|2019-05-07 21:51:33|shadow_client.c|_update_ack_cb(117): Received Json
Document={"clientToken":"BKDDAHRGRX-0","payload":{"metadata":{"reported":
{}},"state":{"reported":
{}},"timestamp":1557236942,"version":0},"result":0,"timestamp":1557237093,"type
":"get"}
DBG|2019-05-07 21:51:34|light_data_template_sample.c|main(602): cycle report:
{"version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
DBG|2019-05-07 21:51:34|shadow_client.c|IOT_Shadow_Update(318): UPDATE Request
Document: {"version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
DBG|2019-05-07 21:51:34|mqtt_client_publish.c|qcloud_iot_mqtt_publish(337):
publish packetID=0|topicName=$template/operation/BKDDAHRGRX/dev001|payload=
{"type":"update", "version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
INF|2019-05-07 21:51:34|light_data_template_sample.c|main(607): shadow
update(reported) success
INF|2019-05-07
21:51:35|light_data_template_sample.c|OnShadowUpdateCallback(352): recv shadow
update response, response ack: 0
DBG|2019-05-07 21:51:40|light_data_template_sample.c|main(602): cycle report:
{"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
DBG|2019-05-07 21:51:40|shadow_client.c|IOT_Shadow_Update(318): UPDATE Request
Document: {"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
```

4. Light Demo 程序定时会上报数据到开发平台，数据格式如下：

```
{"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
```

5. 继续保持 Light Demo 程序处于运行状态，然后前往控制台查看该设备的数据。

查看设备状态

1. 保持 light Demo 程序为运行状态。
2. 进入**控制台 > 产品开发 > 设备调试**，可查看到设备 “dev001” 的状态为“上线”状态，表示 Demo 程序已成功连接上开发平台。
3. 单击**查看**，可进入设备详情页。

4. 单击**设备属性**，可查询设备上报到开发平台的最新数据及历史数据。
   - 当前上报数据的最新值：会显示设备上报的最新数据。
   - 当前上报数据的更新时间：显示数据的更新时间。

5. 单击**查看**，可查看某个属性的历史上报数据。

查看设备通信日志

单击**设备日志**，可查询该设备某段时间范围的所有上下行数据。

- 上行：上行指设备端上报到开发平台的数据。
- 下行：下行指从开发平台下发到设备的数据。

在线调试

1. 当 Light Demo 成功连接到物联网开发平台后，您可在控制台**设备调试**列表，单击**调试**，进入在线调试。

2. 将亮度设置为68，颜色设置为“Red”，单击**发送**。
3. 查看 Light Demo 程序，可查看到成功接收到下发的数据。

```
lighting  ][color: RED ][brightness:||||||||||||-------][dev001]
```

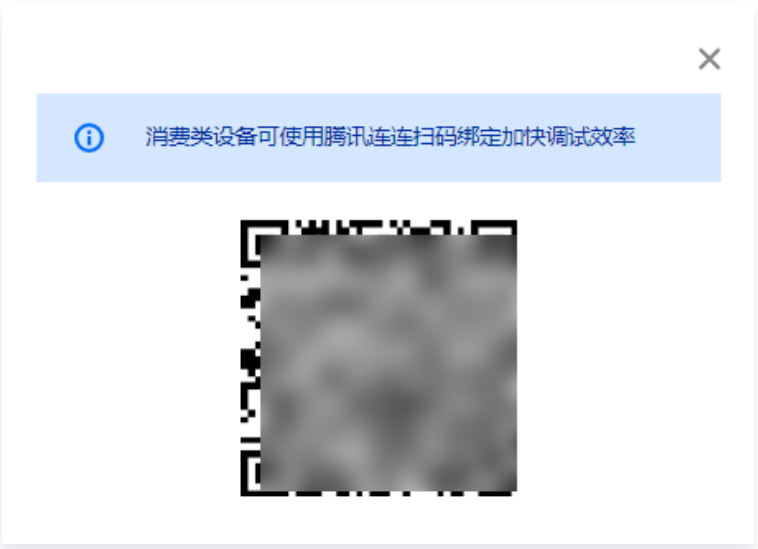
4. 通信日志会显示如下日志，表示成功下发了指令到设备端。

```
{
"method": "control",
"clientToken": "123",
"params": {
    "power_switch": 1,
    "color": 1,
    "brightness": 68
}
}
```

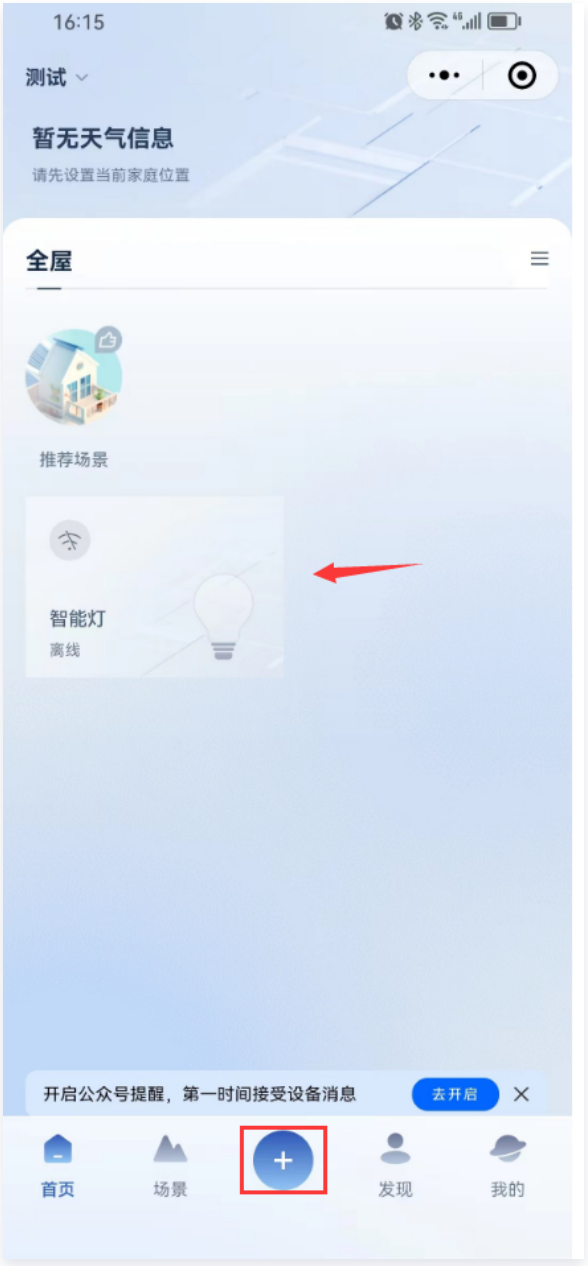
5. 查看通信日志，即可查看到设备成功接收到下行指令，并上报最新数据到开发平台的详细日志。

腾讯连连小程序绑定设备

1. 点击刚创建的“智能灯”产品，在设备列表中，找到刚创建的设备。
2. 单击右侧的**二维码**，展示用于腾讯连连小程序绑定的二维码。



3. 打开腾讯连连微信小程序，通过**扫一扫**添加此设备，添加成功后，在设备列表中将看到此设备。



4. 点击“智能灯”设备卡片，进入智能灯操作界面，可查看设备上报的数据和控制设备。


16:19
智能灯
电灯开关
关
电灯开关
亮度
1%
Red
Green
Blue
色温
0%

企业实例快速入门
企业实例入门概述

最近更新时间：2024-11-27 17:34:32

企业实例通常为产业类的物联网解决方案提供设备接入、消息通信、设备管理等能力。本文将以车载设备为例帮助用户快速了解企业实例的基本功能，如何将车载设备接入企业实例，车载设备如何上报消息，如何通过平台下发控制消息到设备，以及如何将设备消息转发至用户的业务系统。

前提条件

1. 注册腾讯云 账号，并完成 实名认证 。
2. 开通 物联网开发平台 。

操作步骤

用户可按如下指引快速了解企业实例。

1. 创建企业实例及产品设备：如何开通企业实例及如何创建产品与设备。
2. 设备接入及上报数据：如何通过模拟程序接入平台并上报数据。
3. 平台下发控制指令：如何通过平台控制设备。
4. 平台转发消息至用户业务系统：用户的业务系统如何接收物联网开发平台采集的设备数据和状态。

创建企业实例及产品设备

最近更新时间：2024-11-27 17:34:32

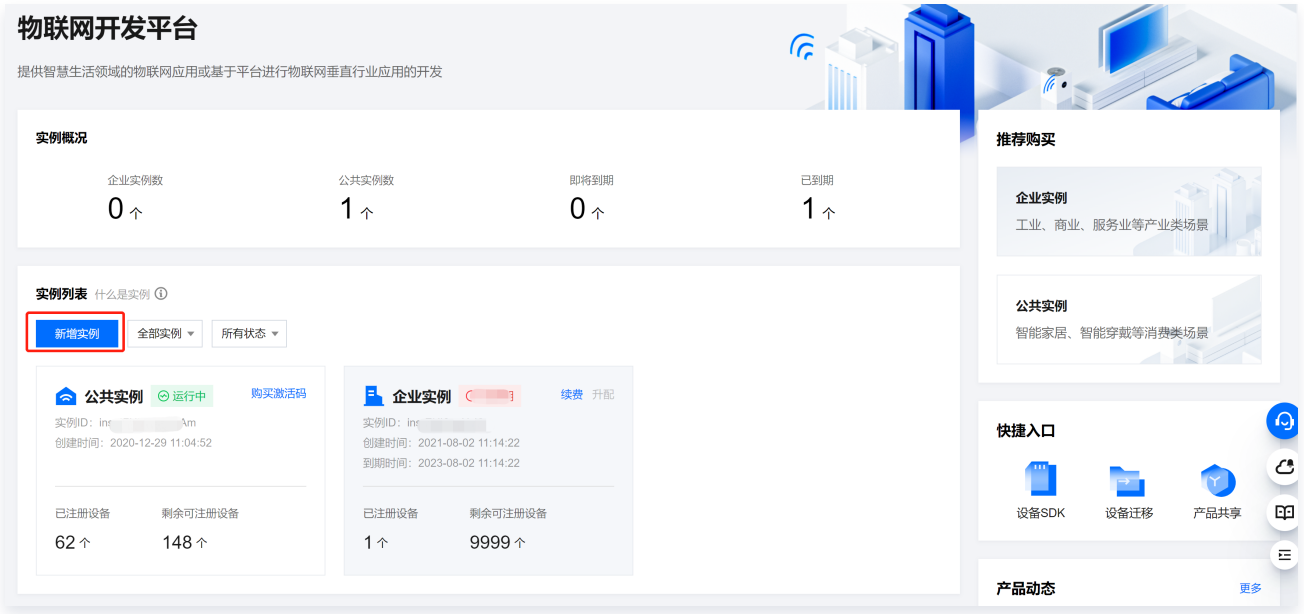
本文介绍如何开通企业实例，如何在企业实例下创建产品与设备，为设备接入与设备上报数据做准备。

操作步骤

购买企业实例

1. 进入物联网开发平台的产品介绍页或者进入物联网开发平台控制台的实例列表页都可购买企业实例。
   - 在 物联网开发平台 IoT Explorer 产品介绍页面，单击**立即选购**。

   - 登录 物联网开发平台控制台，单击导航栏左侧菜单**实例管理**进入实例列表页。单击实例列表的**新增实例**进入实例购买页。



2. 在实例购买页面，根据您的业务需要选择实例配置规格，单击**立即购买**进入商品信息确认页面。

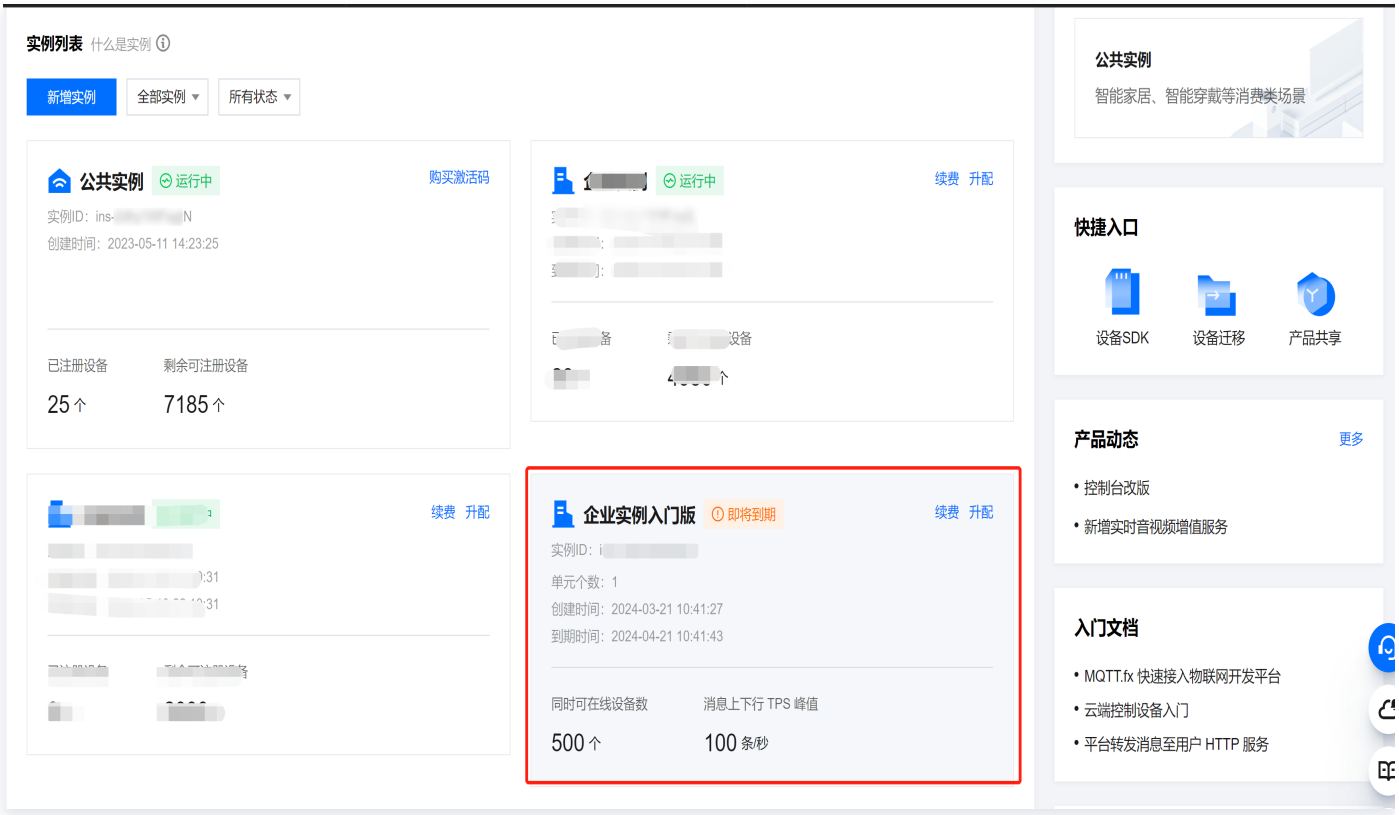
- **实例类型：选择“企业实例”。**
- **实例单元规格：选择对应的规格版，根据实际业务需求选择“入门版”、“标准版”或者“高性能版”。**
- **实例单元数：默认选择1，当选择其他单元数，下方的“实例规格说明”会展示对应单元数的实例规格。**
- **时长：该实例的使用时长，可先选择1个月。**

3. 在订单确认信息页面核对订单信息，确认无误后单击去支付 > 确认支付即可完成实例购买。

> **说明:**
> 如您有腾讯云相关代金券，您可勾选“使用代金券”在结算时进行抵扣。

查看企业实例

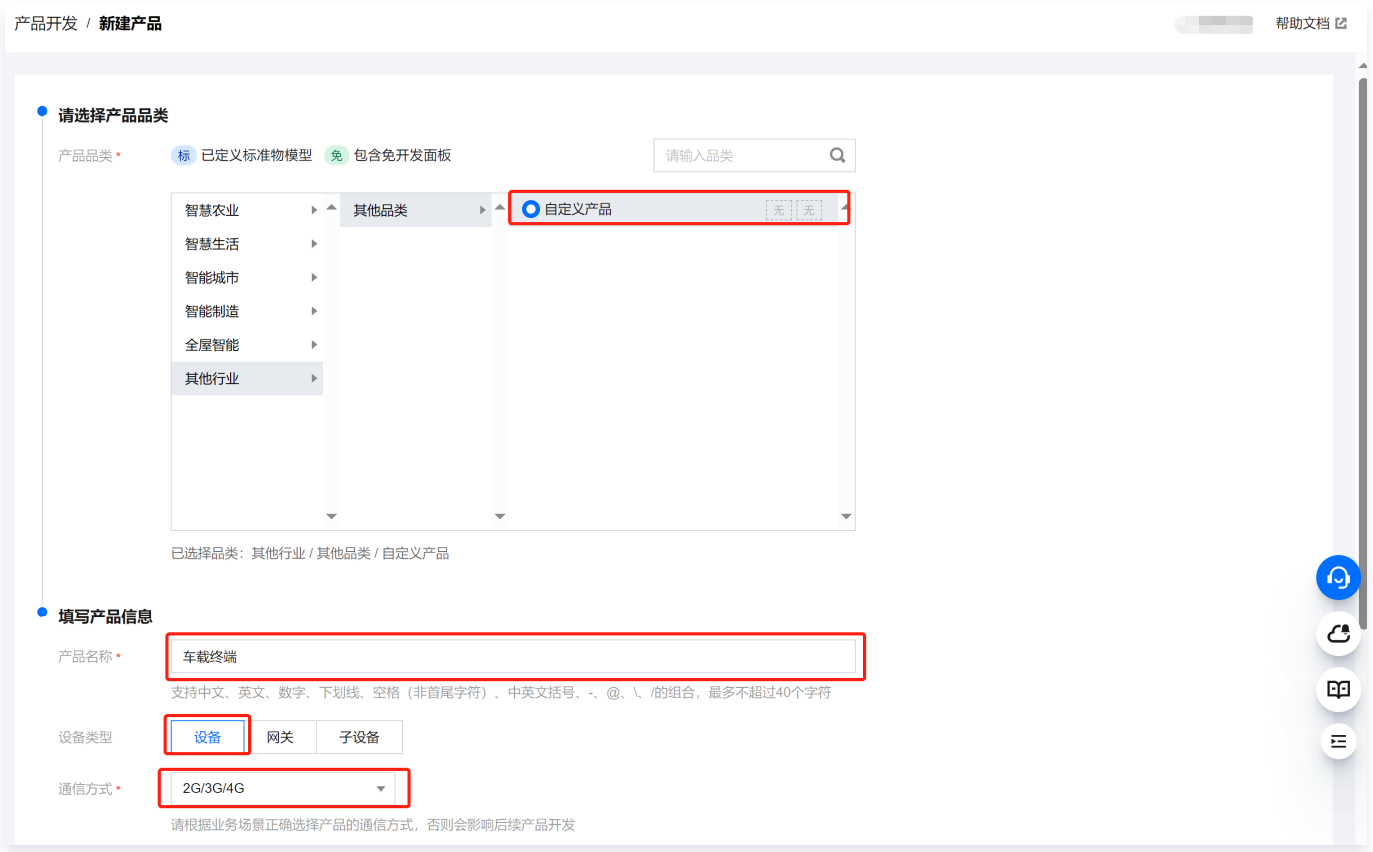
1. 购买成功后，在物联网开发平台的实例列表将查看到购买成功的企业实例。点击企业实例，如下图红色线框中会显示所购买的企业实例信息，查看系统自动分配的实例 ID，单元数等实例规格。



2. 单击企业实例入门版，则进入该企业实例信息概览页，在概览页用户可通过资源状态查看所购买实例的规格信息。

创建产品

1. 在实例列表页单击对应企业实例，进入实例内页，单击左侧菜单的**产品开发**，进入产品列表。
2. 单击**新建产品**，显示新建产品表单界面。



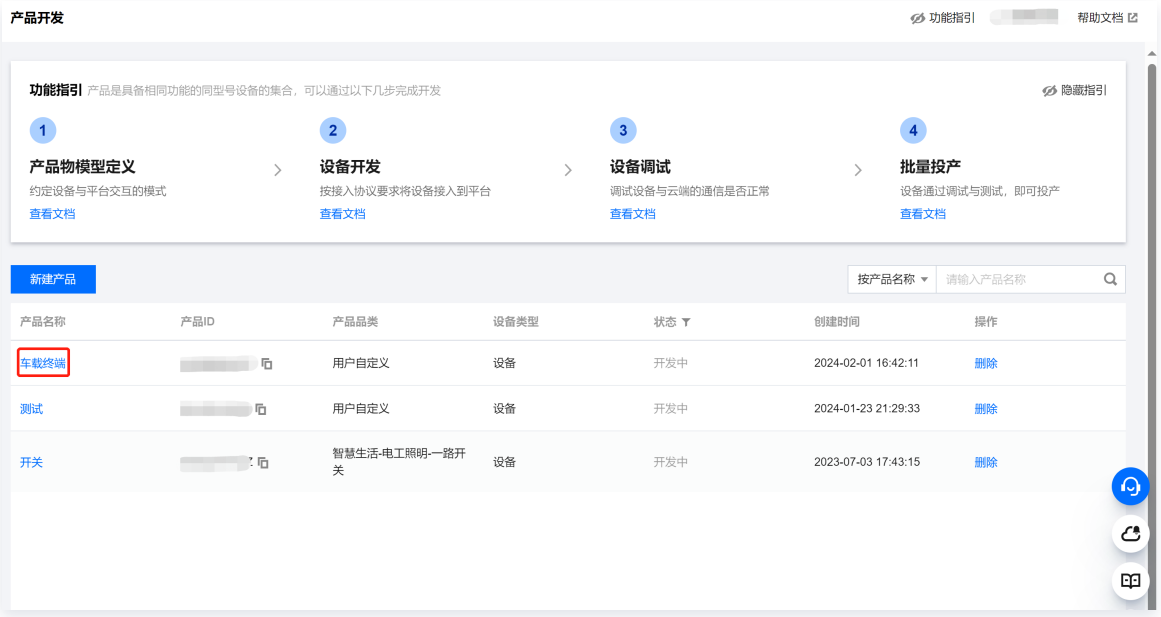
产品信息设置如下：

- 产品品类：选择**其他行业 > 其他品类 > 自定义品类**。
- 产品名称：输入“车载终端”或其他名称。
- 设备类型：选择“设备”。
- 通信方式：选择“2G/3G/4G”，表示设备是通过蜂窝网络接入物联网平台。
- 数据协议：选择“物模型”。

3. 单击"**新建产品**"按钮，系统将为用户创建产品。

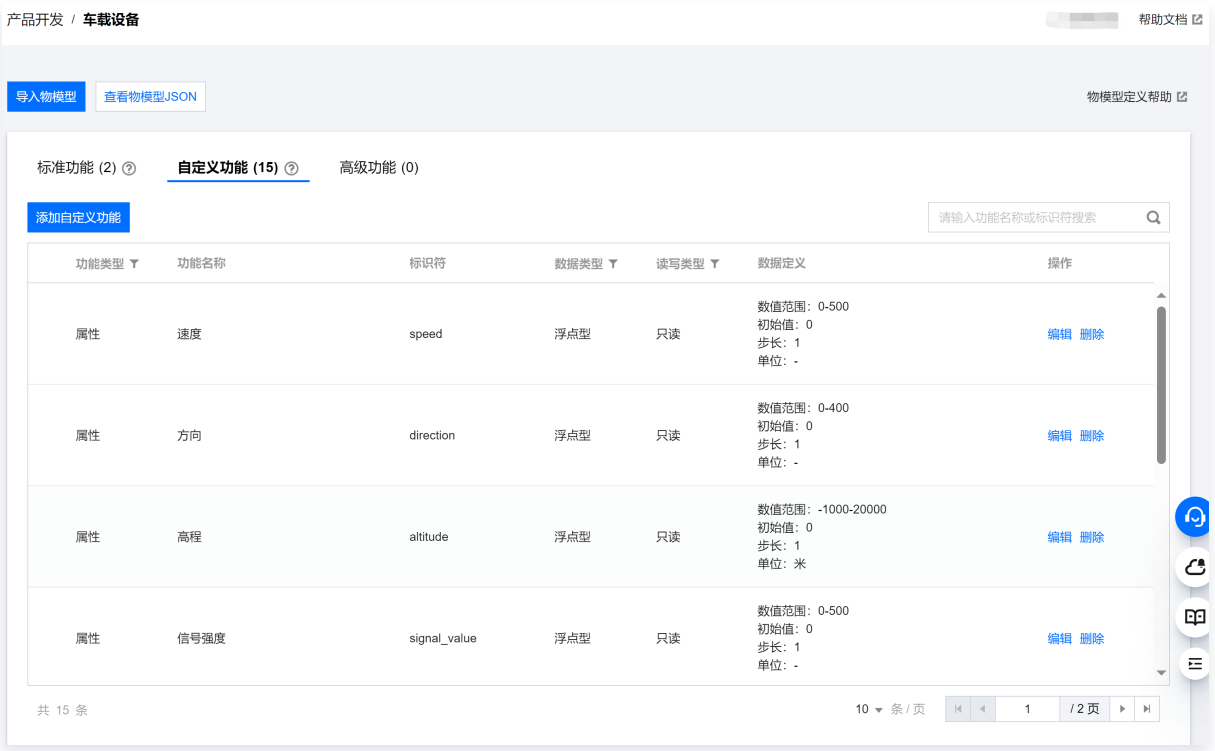
物模型定义

1. 选择已经创建成功的产品，点击**车载终端**，进入产品详情页。



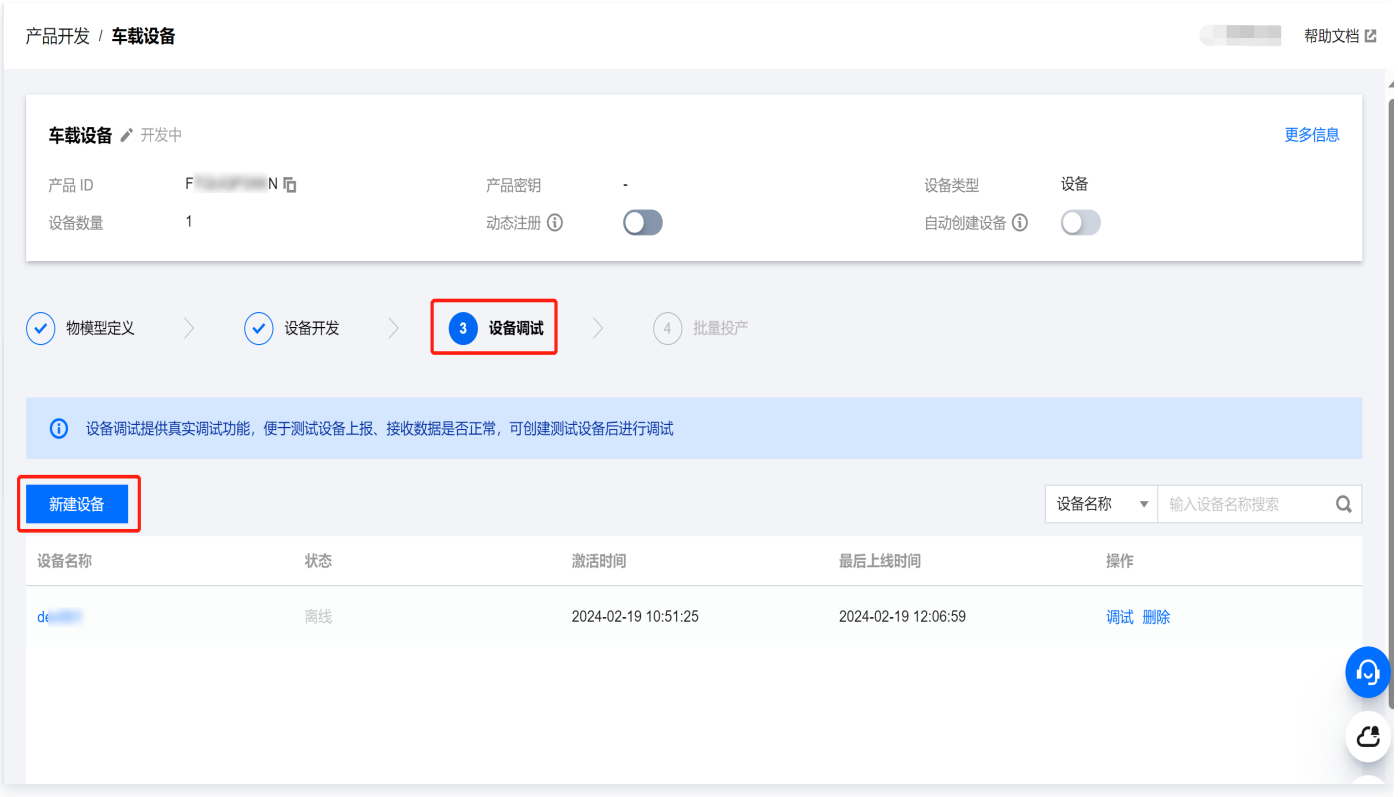
2. 单击**导入物模型**，点击 下载 tbox.json 来下载tbox.json文件。使用文本编辑器打开 tbox.json，复制 tbox.json 文件内容到下图的文本框中，单击**导入**。系统将成功导入物模型定义。

3. 物模型导入成功后，可以查看该产品已从 JSON 文件中成功导入的物模型。



创建设备

1. 单击**设备调试** > **新建设备**按钮。



2. 在弹出的新建设备窗口中，输入设备名称 dev001，单击**保存**按钮。

3. 系统成功创建设备后，自动生成其设备密钥。

设备接入及上报数据

最近更新时间：2024-11-27 17:34:32

使用 Windows 模拟程序上报数据，通过该模拟程序模拟车载设备连接物联网平台，并上报车载设备采集的相关数据到平台。用户可以快速体验设备如何接入及上报物模型数据至平台，以及如何在控制台查看上报的设备数据。

前提条件

1. 已开通企业实例。
2. 已按 创建企业实例及产品设备 指引在用户自有账号下创建完产品、导入对应的物模型以及创建完一个设备。

> **说明：**
> 使用本文的 Windows 设备模拟程序，必须要按指引导入对应的物模型格式，用户也可自由定义物模型，该模拟程序可根据根据 JSON 文件中的物模型定义随机上报物模型属性、事件数据至平台。

操作步骤

下载 MQTT 客户端模拟程序

1. 若用户系统为 Windows 系统且版本为 Win10 及以上，则可以下载平台提供的模拟程序到 PC 端。
   点击下载 mqtt 客户端模拟程序 。
2. 下载模拟程序到用户 Windows 本地磁盘并解压缩。

运行模拟程序上报数据

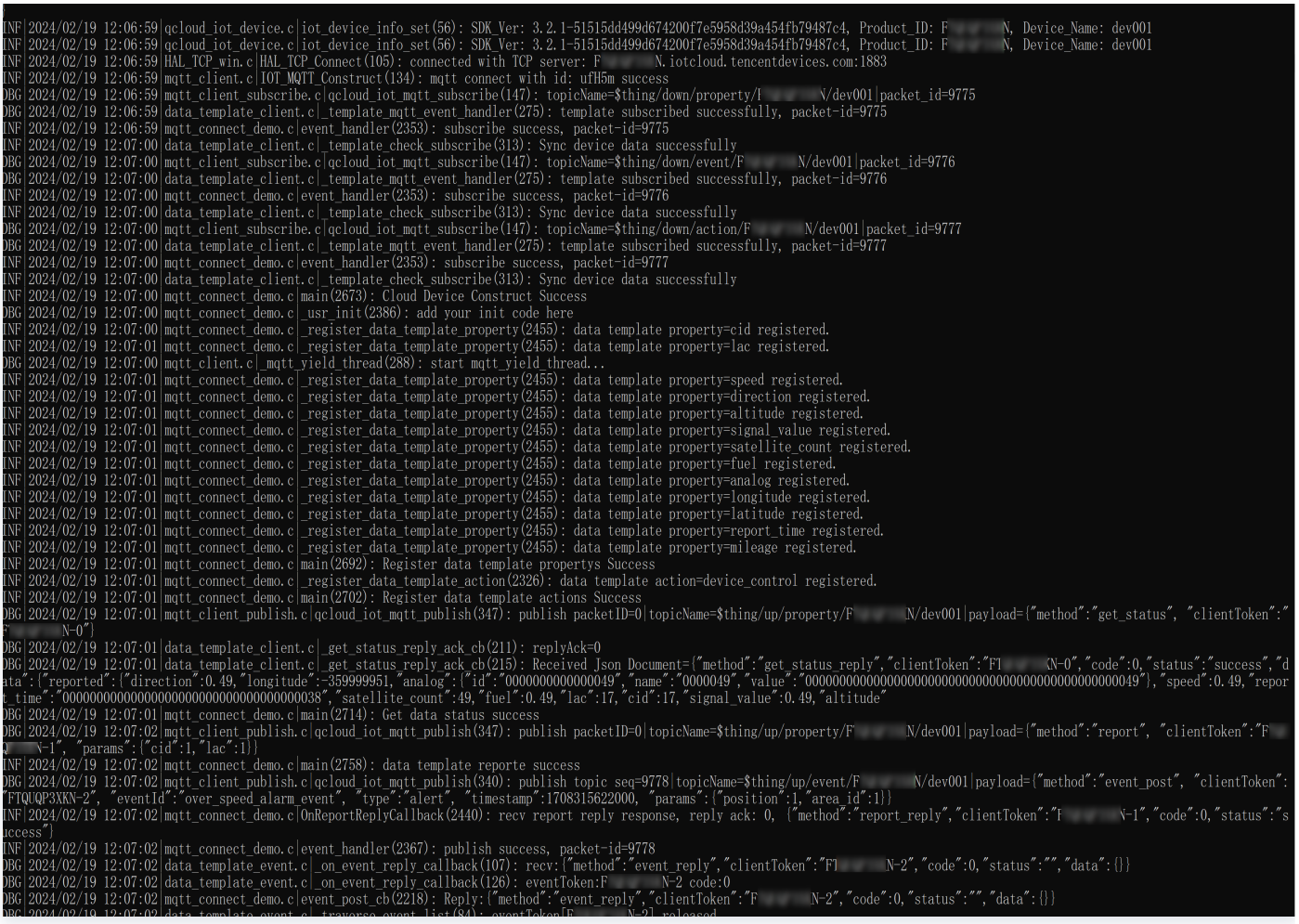
1. 进入 Windows 终端命令行，进入模拟程序解压目录。假设用户解压目录为 e:\mqtt_client。注意，解压后将看到 mqtt_connect.exe 和 tbox.json 两个文件。
2. 运行以下命令。

```
e:\mqtt_client>mqtt_connect.exe -s tbox.json -i 用户产品id -n 用户创建的DeviceName -k 设备密钥 -d 用户产品id.iotcloud.tencentdevices.com -l 50
```

3. 上述命令行中的用户产品id、用户创建的DeviceName、设备密钥分别从控制台上创建的设备信息页中复制粘贴。如下图所示：

命令行参数	描述
-s	创建产品导出的物模型 JSON 文件，模拟程序将根据该 JSON 文件定义的属性、事件随机上报数据至平台。
-i	指定产品 ID，从控制台创建的产品 ID。
-n	指定设备名称。注意，动态注册一般是由设备端随机生成设备名称，无需在控制台创建设备。
-k	指设备密钥，从控制台所创建的设备信息页中获取设备密钥。
-d	MQTT 服务器地址，替换为自己的产品 ID，平台 MQTT 服务地址为 “产品 ID.iotcloud.tencentdevices.com”。
-l	指定循环次数，循环上报数据次数达到后模拟程序自动关闭 MQTT 连接并退出。

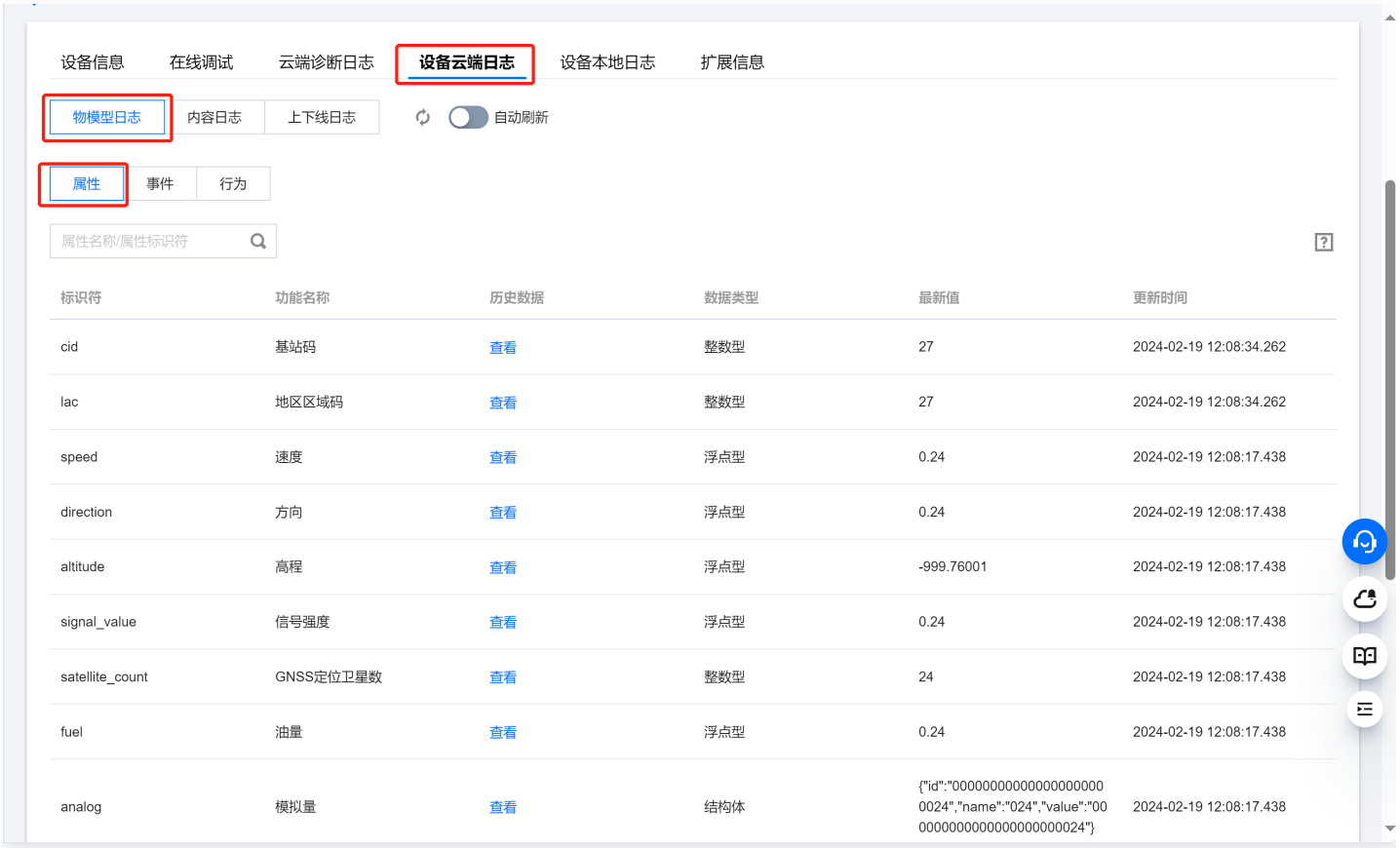
4. 运行成功后，模拟程序将打印如下日志，用户无需关闭窗口，模拟程序将会随机上报数据至平台。



控制台查看设备上报数据

查看物模型日志数据

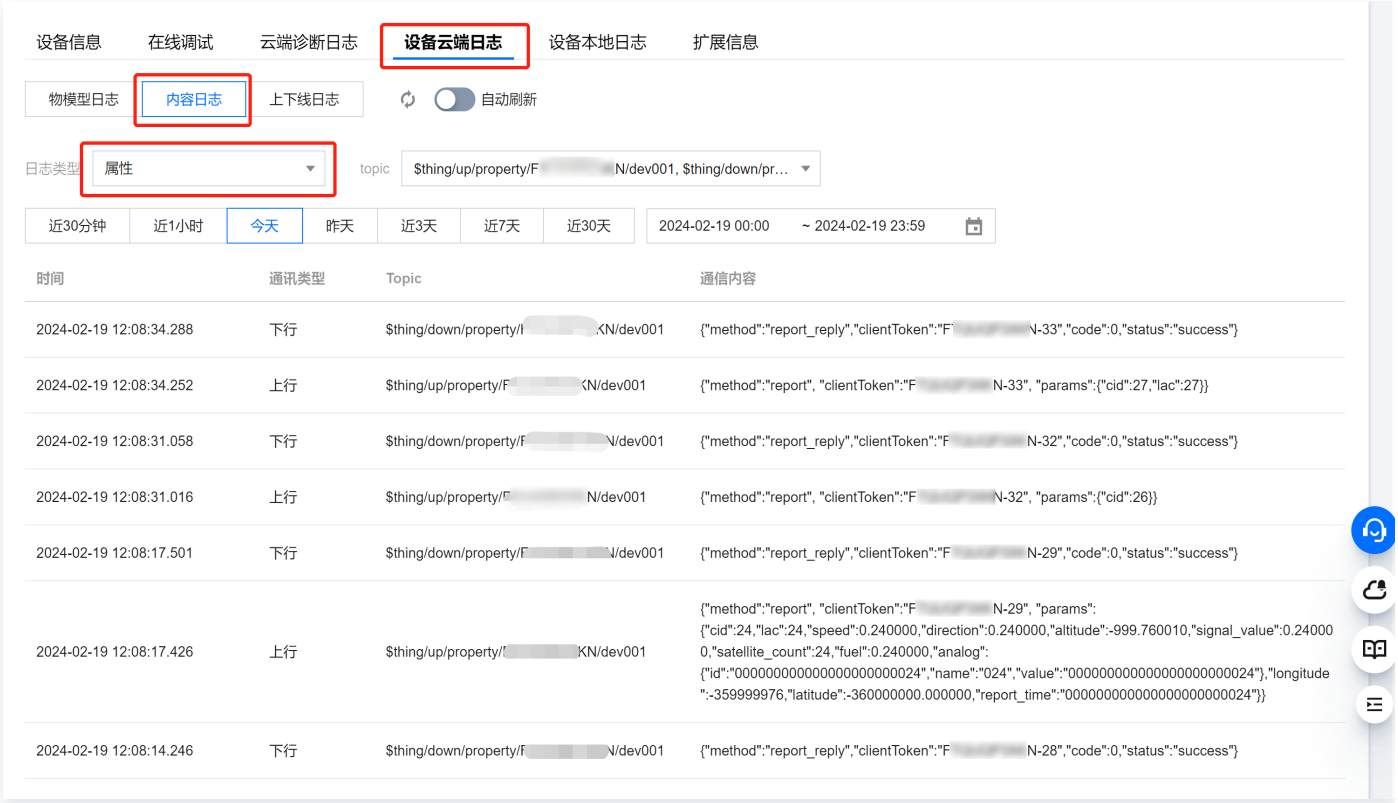
1. 进入该模拟设备控制台页面，进入**设备云端日志 > 物模型日志 > 属性**。
2. 当模拟程序在运行时，将会在如下页面查看到模拟程序上报的最新属性值以及对应的更新时间。



标识符	功能名称	历史数据	数据类型	最新值	更新时间
cid	基站码	查看	整数型	27	2024-02-19 12:08:34.262
lac	地区区域码	查看	整数型	27	2024-02-19 12:08:34.262
speed	速度	查看	浮点型	0.24	2024-02-19 12:08:17.438
direction	方向	查看	浮点型	0.24	2024-02-19 12:08:17.438
altitude	高程	查看	浮点型	-999.76001	2024-02-19 12:08:17.438
signal_value	信号强度	查看	浮点型	0.24	2024-02-19 12:08:17.438
satellite_count	GNSS定位卫星数	查看	整数型	24	2024-02-19 12:08:17.438
fuel	油量	查看	浮点型	0.24	2024-02-19 12:08:17.438
analog	模拟量	查看	结构体	{"id":"0000000000000000000024","name":"024","value":"000000000000000000000024"}	2024-02-19 12:08:17.438

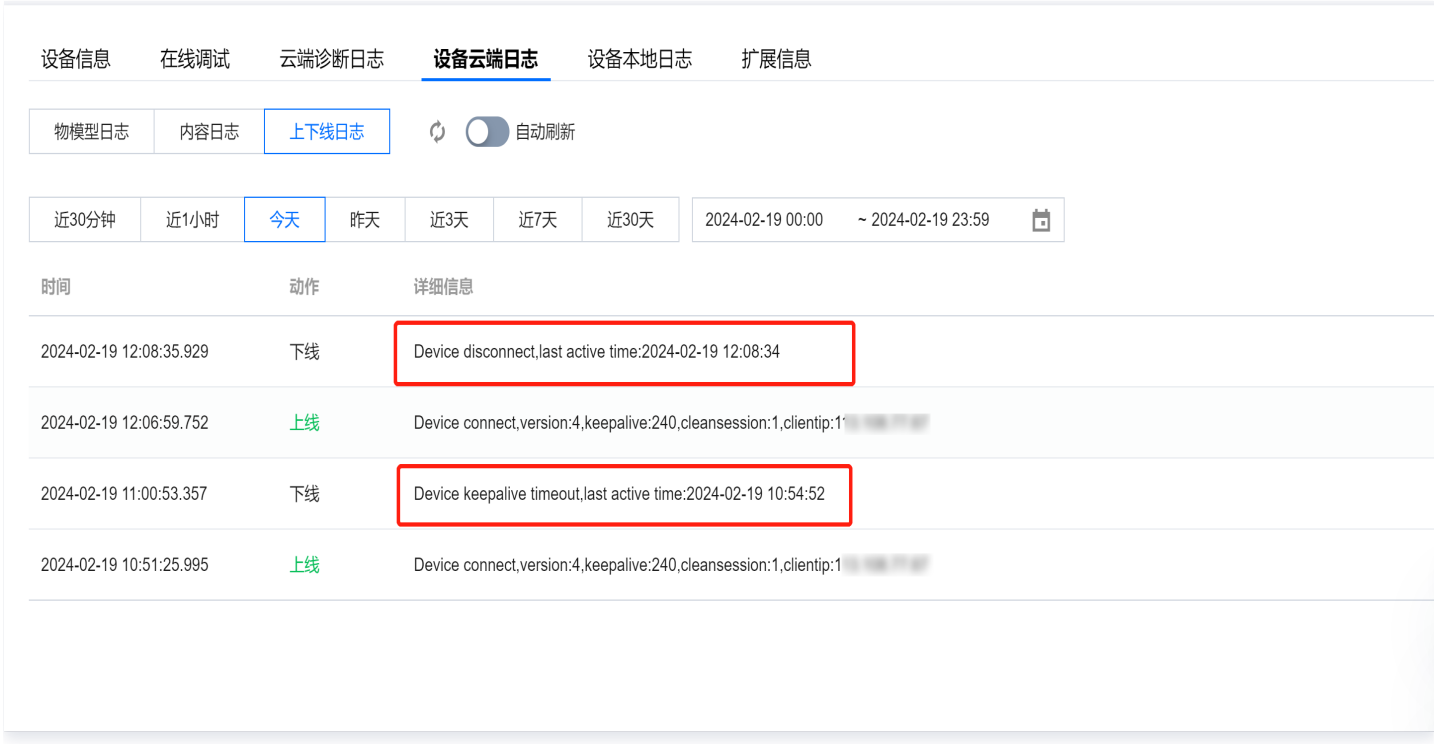
查看内容日志

1. 进入该模拟设备控制台页面，进入**设备云端日志 > 内容日志 > 属性**。
2. 当模拟程序在运行后并成功上报数据后，将会在如下页面查看到模拟程序上行的 JSON 报文以及平台返回至设备的下行 JSON 报文。详细的物模型协议格式可参考 物模型协议 。



查看设备上线/离线日志

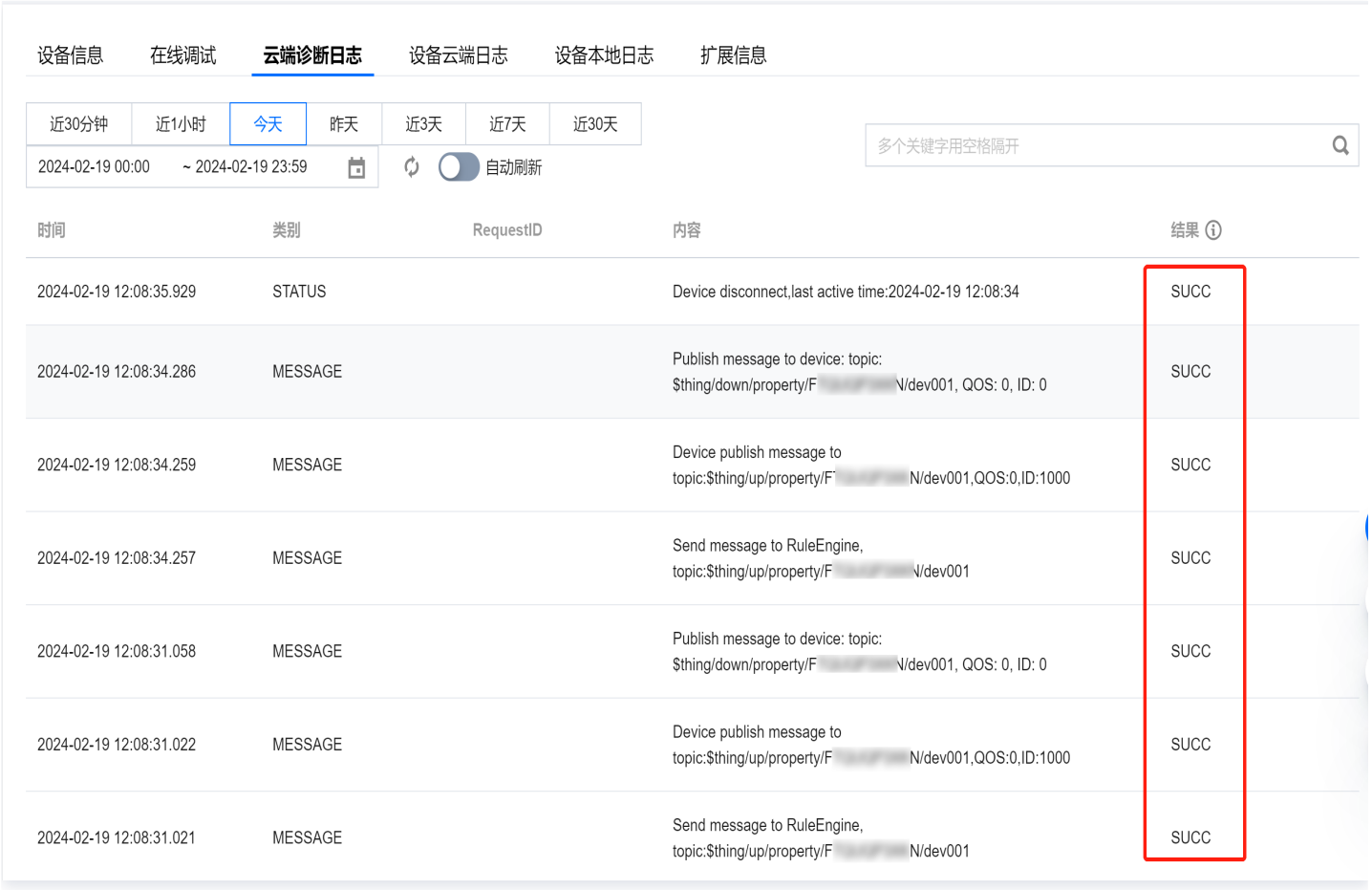
1. 进入该模拟设备控制台页面，进入**设备云端日志 >上下线日志**，可查询设备上下线行为以及下线原因。
2. 当模拟程序在运行后并成功上报数据后，将会在如下页面查看到设备连接到平台的事件，动作类型为“上线”；若模拟程序主动关闭或者因心跳超时，则动作类型都显示为“下线”，具体的原因如下图红色线框显示。



控制台查看设备轨迹日志

平台提供了云端轨迹日志查询功能，当设备出现无法上报数据或无法接收控制指令或规则引擎无法转发消息时，便于用户分析哪个环节出现问题，用于定位是否是设备侧或用户业务系统相关问题。

1. 进入该模拟设备控制台页面，进入**设备云端日志 > 云端诊断日志**，选择合适的时间范围。
2. 当模拟程序在运行后并成功上报数据后，将会在如下页面查看到模拟设备在平台的端到端行为轨迹。
3. 若某个行为出现失败，失败的原因分析可参考 异常原因 。



平台下发控制指令

最近更新时间：2024-11-27 17:34:32

车载设备接入到平台后，一般会通过用户的业务系统发起控制设备的操作。例如远程设置车载设备的某个参数，可以通过物联网开发平台提供的云 API 或在线调试功能下发控制消息至设备侧。

前提条件

体验下发控制指令需要提前做好以下准备工作：

- 按企业实例快速入门创建对应产品，并导入物模型。
- 能够成功使用 Windows 模拟程序模拟设备连接平台。

使用 API Explorer 控制设备

1. 登录 物联网开发平台 后，访问 设备远程控制 API ，可查看平台提供的 API 服务，单击“**点击调试**”，可进入 API Explorer 在线 API 调试工具。
2. 进入设备远程控制 API 调试页面，输入必选参数。然后单击右侧区域的“**发送请求**”按钮。API Explorer 将返回响应结果。

> ⚠ **注意：**
> 在发送请求前，请确保被控设备已成功连接平台。

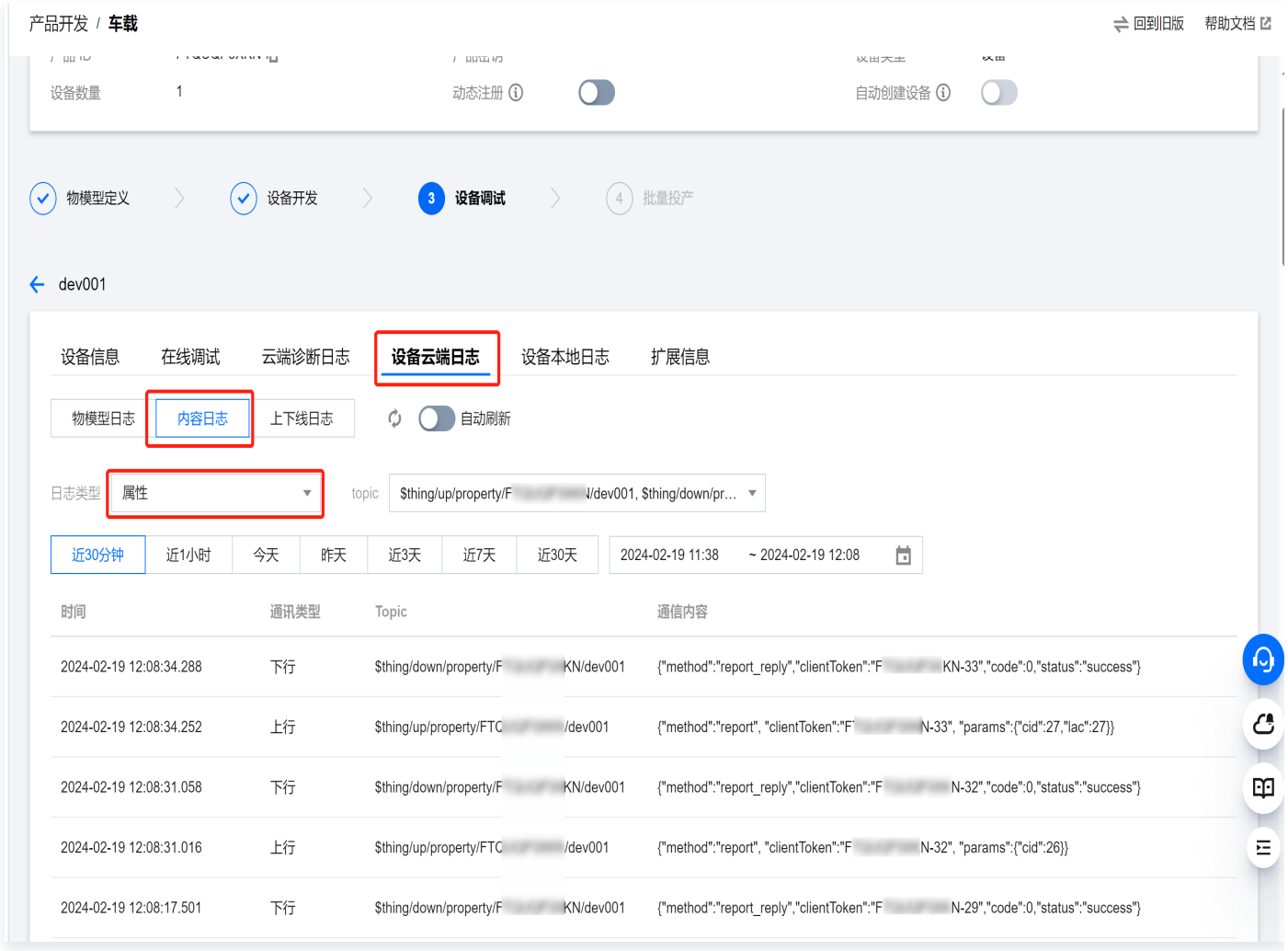
   - 输入参数 Region：选择**华南地区（广州）ap-guangzhou**。
   - 输入参数 ProductId：请输入按企业实例快速入门指引，在平台生成的对应产品 ID。
   - 输入参数 DeviceName：请输入连接到平台的 DeviceName。
   - 输入参数 Data：请输入物模型 JSON，例如{"win_switch":1}

3. 若输入参数输入正确，而且远控的设备已连接物联网开发平台，即为在线状态，API Explorer显示的相应结果为如下报文。

```
{
    "Response": {
        "Data": "",
        "RequestId": "519486c9-d37c-4c3f-91e4-5261866e3e24",
        "Result": "{\"Sent\":1,\"pushResult\":0}"
    }
}
```

控制台查看下行控制消息

1. 进入控制台**产品开发**，选择目标产品进入**设备调试**，单击模拟接入的设备名称。
2. 选择**设备云端日志 > 内容日志 > 属性**，筛选设备下行时间范围。



3. 搜索“win_switch”，设备远程控制 API 执行成功后，在控制台可以查询到对应的下行消息记录。

2024-02-19 12:08:17.501	下行	$thing/down/property/F[redacted]N/dev001	{"method":"report_reply","clientToken":"F[redacted]N-29","code":0,"status":"success"}
2024-02-19 12:08:17.426	上行	$thing/up/property/F[redacted]N/dev001	{"method":"report", "clientToken":"F[redacted]N-29", "params": {"cid":24,"lac":24,"speed":0.240000,"direction":0.240000,"altitude":-999.760010,"signal_value":0.240000,"satellite_count":24,"fuel":0.240000,"analog": {"id":"00000000000000000000000024","name":"024","value":"00000000000000000000000024"},"longitude":-359999976,"latitude":-360000000.000000,"report_time":"00000000000000000000000024"}}
2024-02-19 12:08:14.246	下行	$thing/down/property/F[redacted]N/dev001	{"method":"report_reply","clientToken":"F[redacted]N-28","code":0,"status":"success"}
2024-02-19 12:08:14.205	上行	$thing/up/property/F[redacted]N/dev001	{"method":"report", "clientToken":"F[redacted]N-28", "params": {"cid":23,"lac":23,"speed":0.230000,"direction":0.230000,"altitude":-999.770020,"signal_value":0.230000,"satellite_count":23,"fuel":0.230000,"analog": {"id":"00000000000000000000000023","name":"23","value":"00000000000000000000000023"},"longitude":-359999977,"latitude":-360000000.000000}}
2024-02-19 12:08:11.624	下行	$thing/down/property/F[redacted]N/dev001	{"method":"control","clientToken":"v2530232175iDTeC::519486c9-d37c-4c3f-91e4-5261866e3e24","params":{"win_switch":1}}
2024-02-19 12:08:11.032	下行	$thing/down/property/F[redacted]N/dev001	{"method":"report_reply","clientToken":"F[redacted]N-27","code":0,"status":"success"}
2024-02-19 12:08:10.992	上行	$thing/up/property/F[redacted]N/dev001	{"method":"report", "clientToken":"F[redacted]N-27", "params": {"cid":22,"lac":22,"speed":0.220000,"direction":0.220000,"altitude":-999.780029,"signal_value":0.220000,"satellite_count":22,"fuel":0.220000,"analog": {"id":"00000000000000000000000022","name":"2","value":"00000000000000000000000022"},"longitude":-359999978}}

平台转发消息至用户业务系统

最近更新时间：2024-11-27 17:34:32

操作场景

物联网解决方案通常需要实时获取设备上报到物联网平台的数据、状态等信息，再结合各自场景的业务数据来完成整套物联网解决方案的闭环业务流程。本文档主要介绍如何将采集的车载设备数据使用平台的规则引擎消息转发至用户自建 HTTP 服务的能力，将设备数据、状态转发到用户自建的 HTTP 服务，用户可根据此文档中的代码示例快速了解并构建自己的 HTTP 服务获取设备数据。

前提条件

1. 已开通企业实例，模拟程序已接入平台并上报数据至平台调试通过。
2. 用户需提前注册开通 腾讯云 SCF 产品，通过 SCF 产品托管 HTTP 服务，将对应的 HTTP 服务接收地址准备好配置到规则引擎转发服务中。

> **说明：**
> 用户基于腾讯云SCF产品部署好HTTP服务后，需要确保模拟程序连接平台并持续上报数据，HTTP服务才能打印接收到的设备数据。

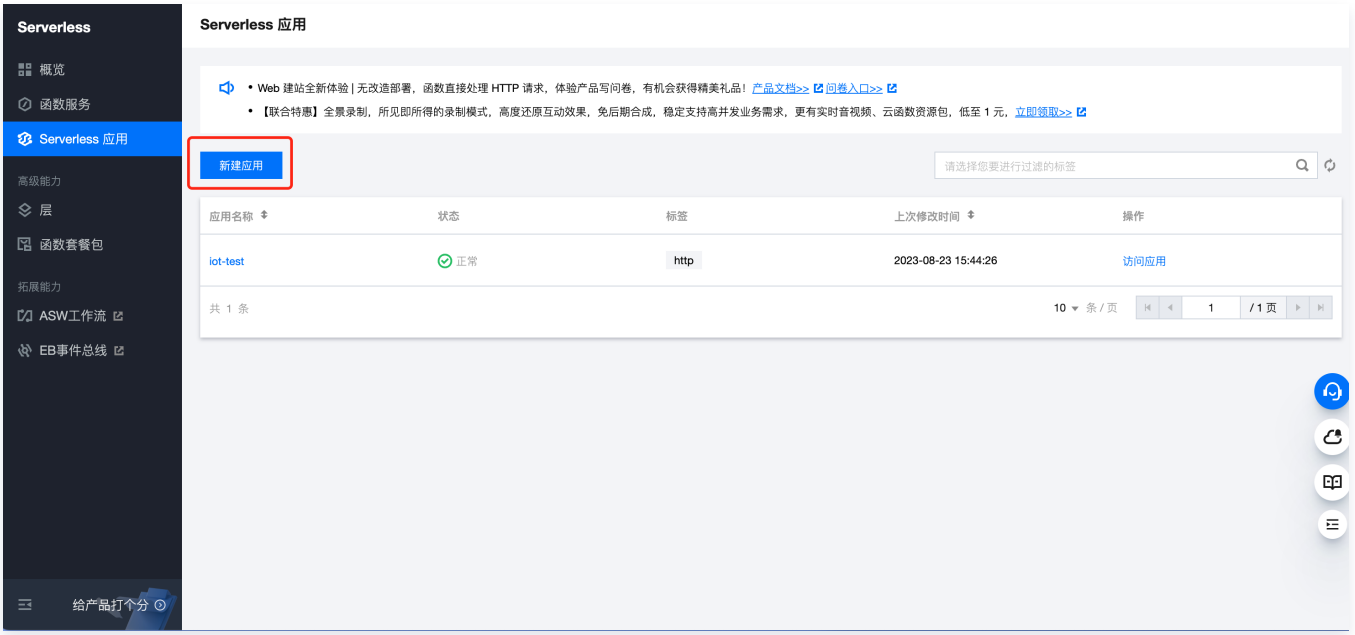
操作步骤

使用 SCF 部署 HTTP 服务

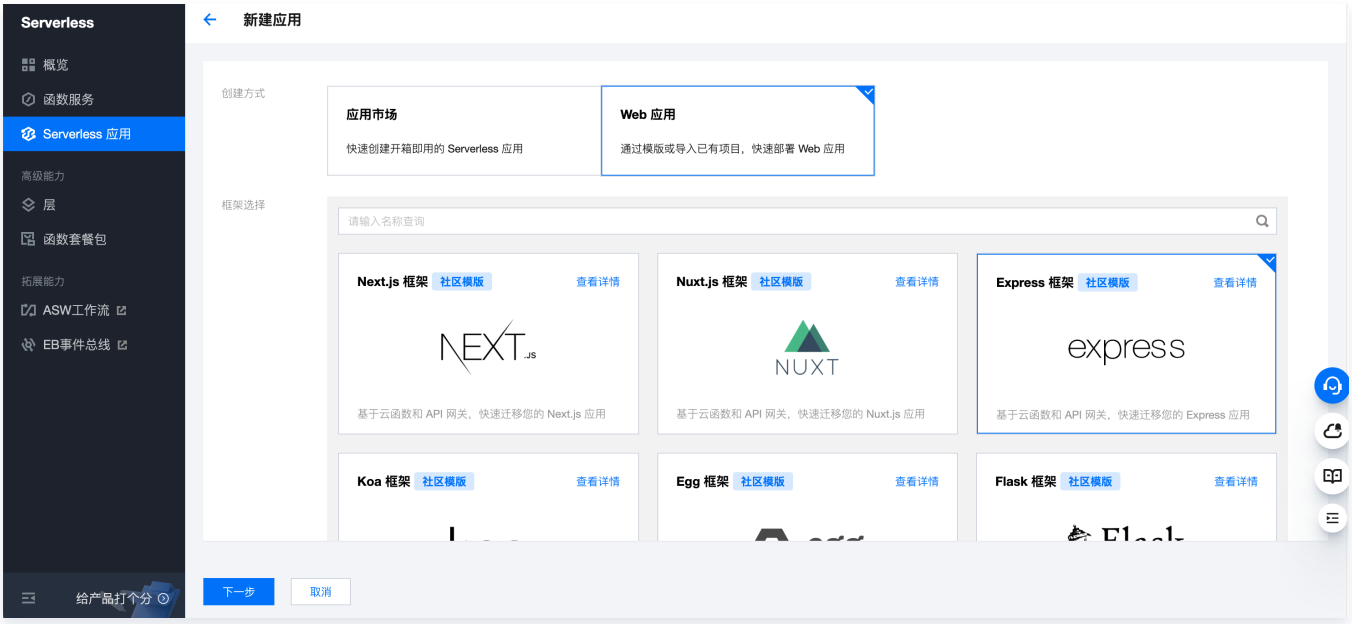
本文档使用腾讯云云函数（Serverless Cloud Function，SCF）产品快速搭建 Web 服务，来接收物联网平台规则引擎处理后的数据。本示例只展示如何接收物联网平台的数据，进一步使用数据需要用户根据实际业务场景去处理。

在 SCF 平台基于 Express 快速搭建 Node 服务示例

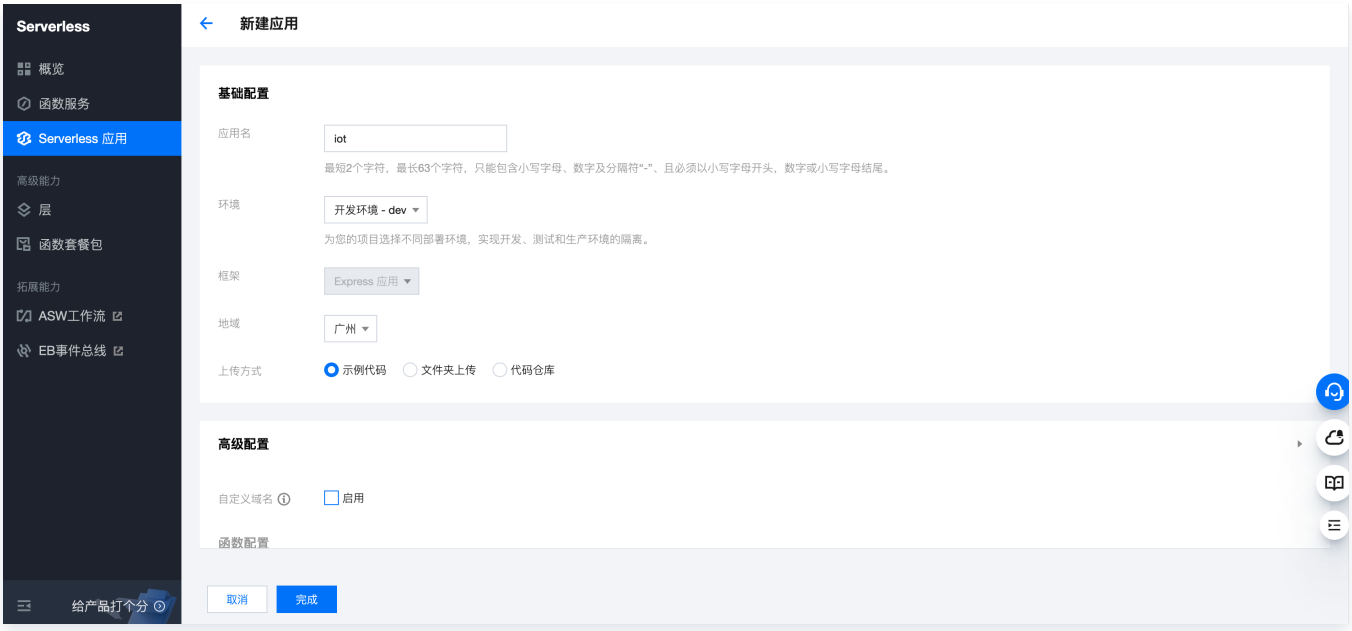
1. 登录 SCF 控制台，选择 Serveless 应用，单击新建应用。



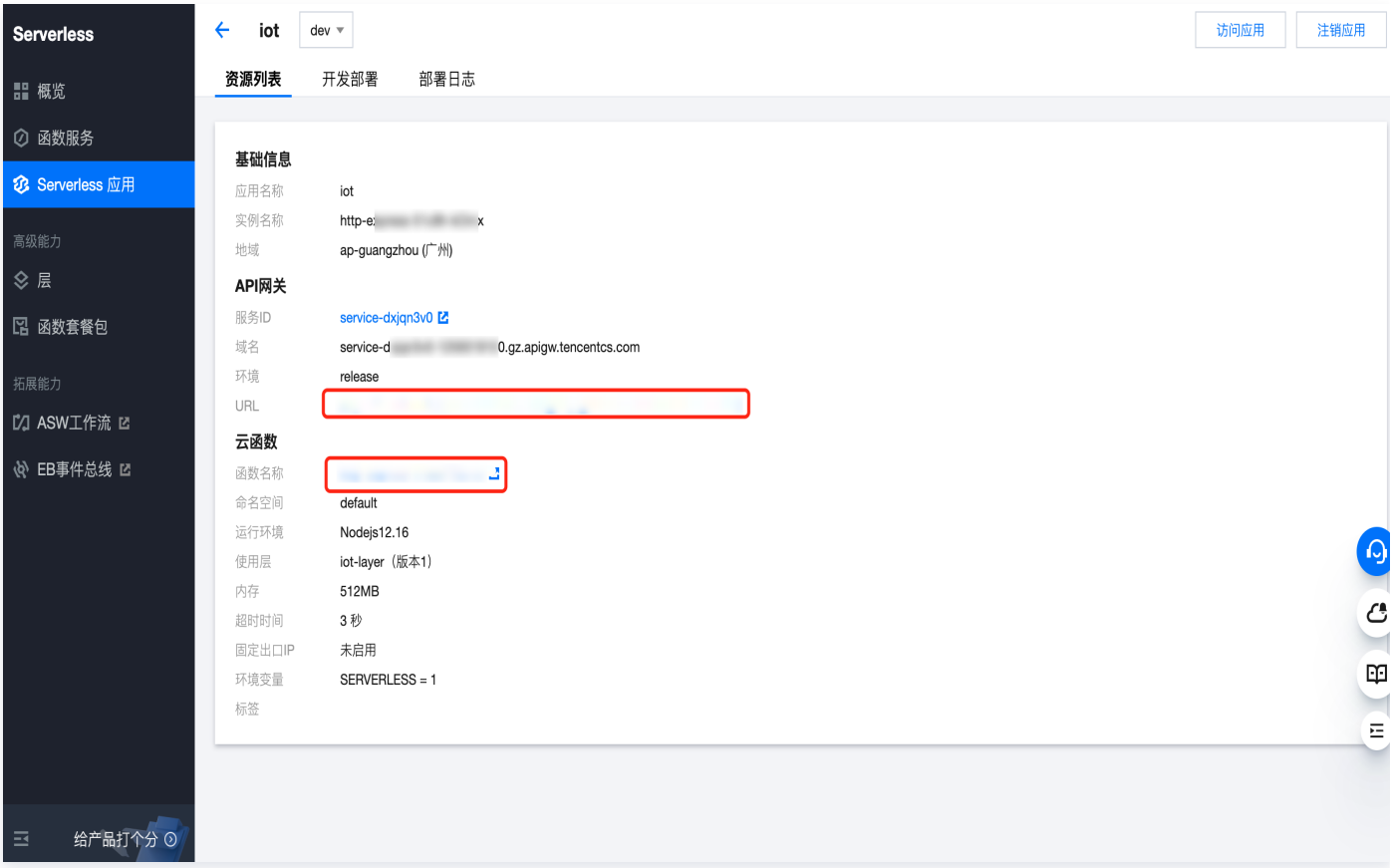
2. 选择 Web 应用下的 Express 框架模板，单击**下一步**。



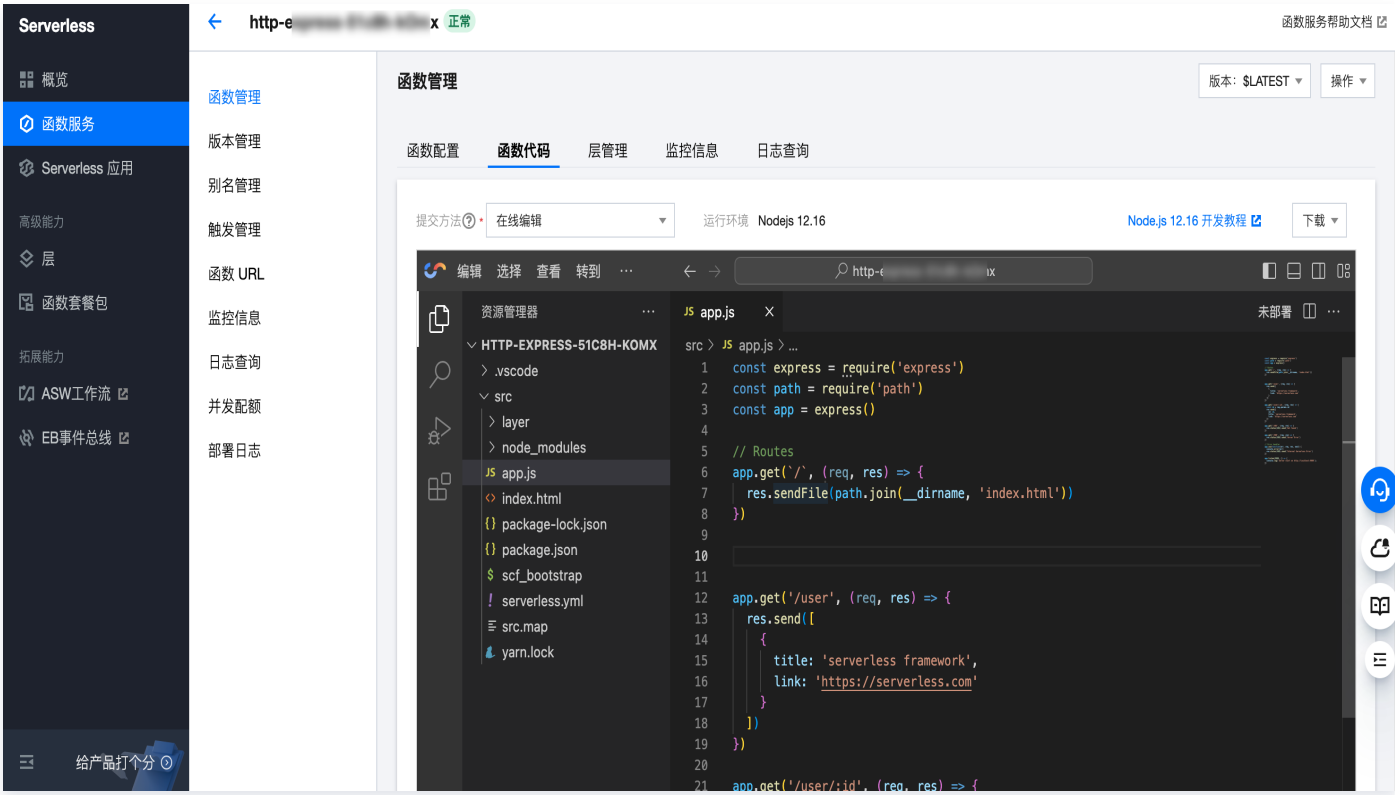
3. 按下图所示输入应用名，选择环境、地域，上传方式。配置结束后单击**完成**。



4. 等待部署完成后进入应用，在 API 网关中可以看到 URL。此 URL 就是您要在规则引擎中填写的 API 地址，假设此 URL 地址为：`https://iot-api/`，那么在本示例中您的 API 地址为 URL 加上接口路由，即：`https://iot-api/test`。单击函数名称，进入函数服务。



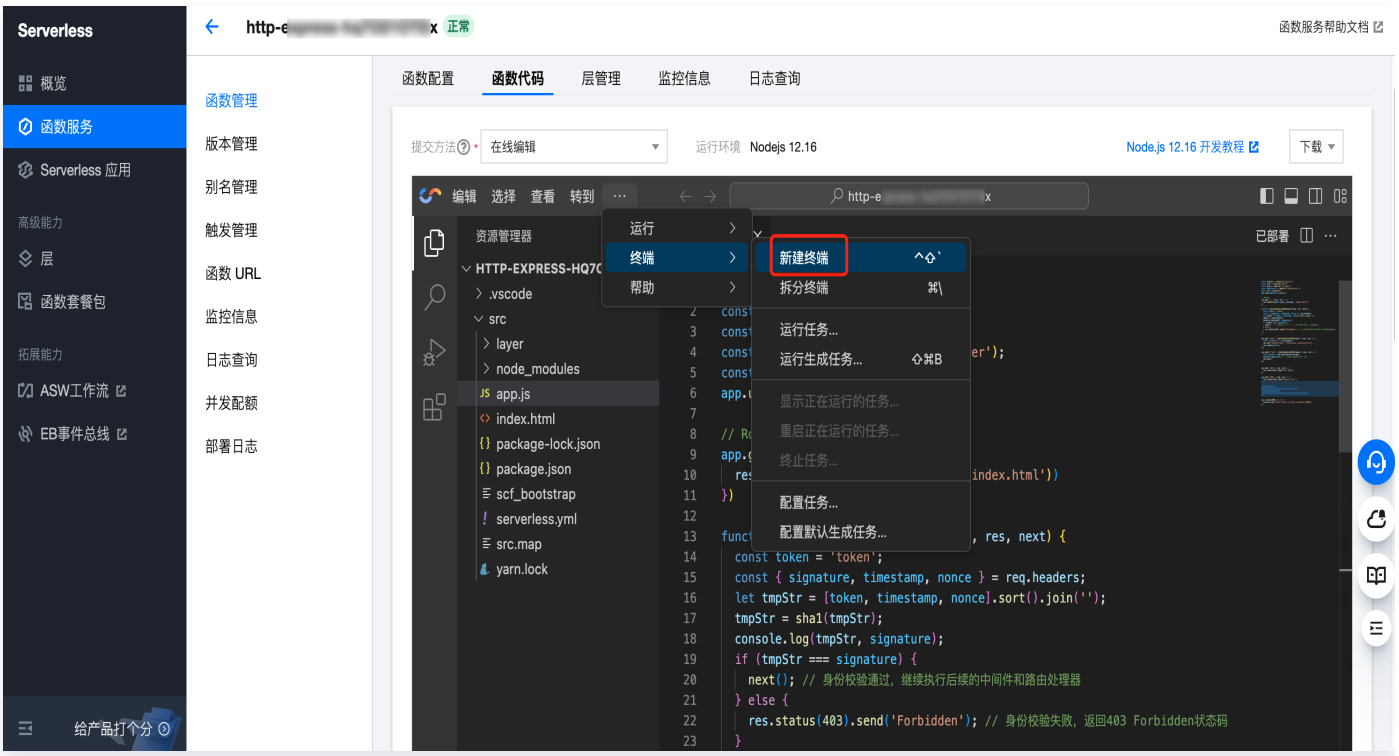
5. 在函数服务中，可以看到如下图所示的对应的代码模板。单击 **app.js**，复制示例代码到 app.js 文件中，注意无需全覆盖。

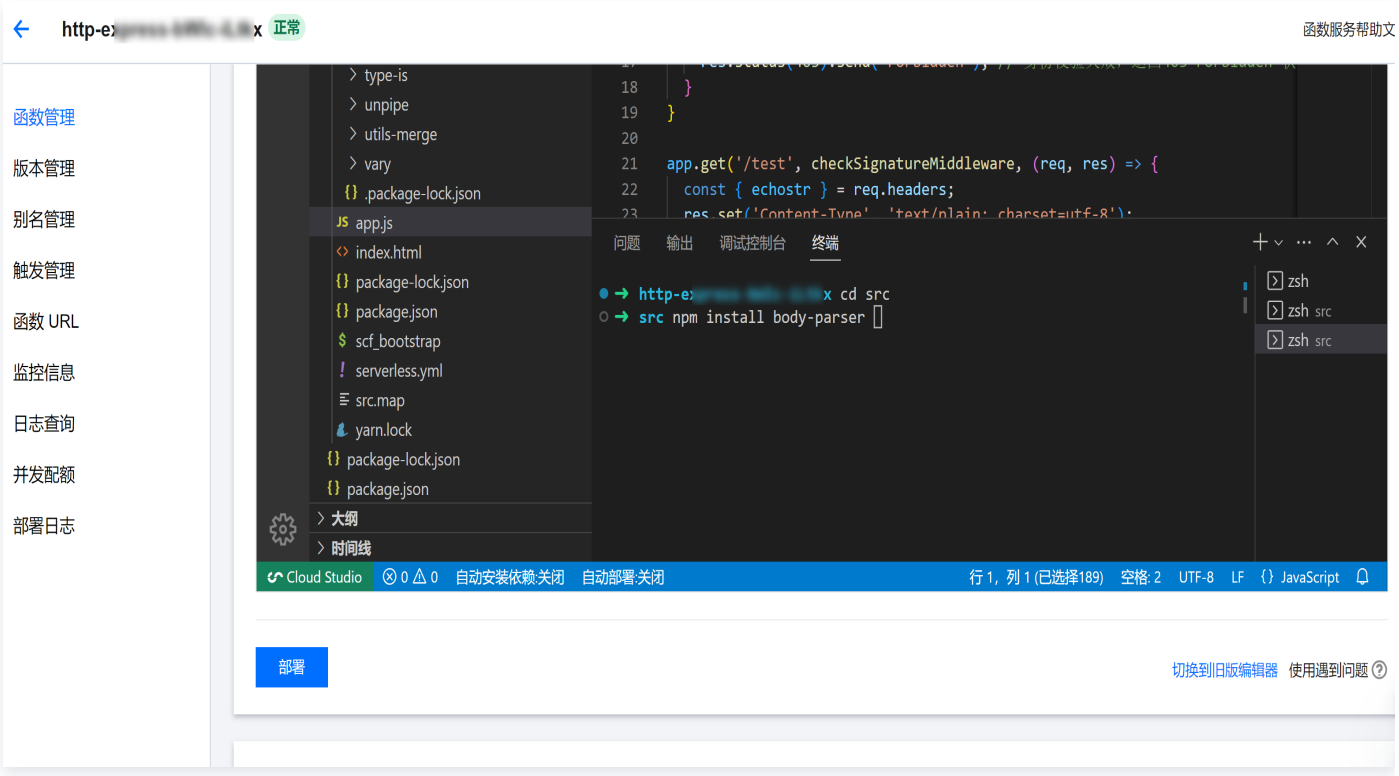


需要将下述代码复制到 app.js 的代码，请覆盖原 app.js 中的 const 开头的3行代码，原 app.js 中的 app.get 方法无需覆盖。

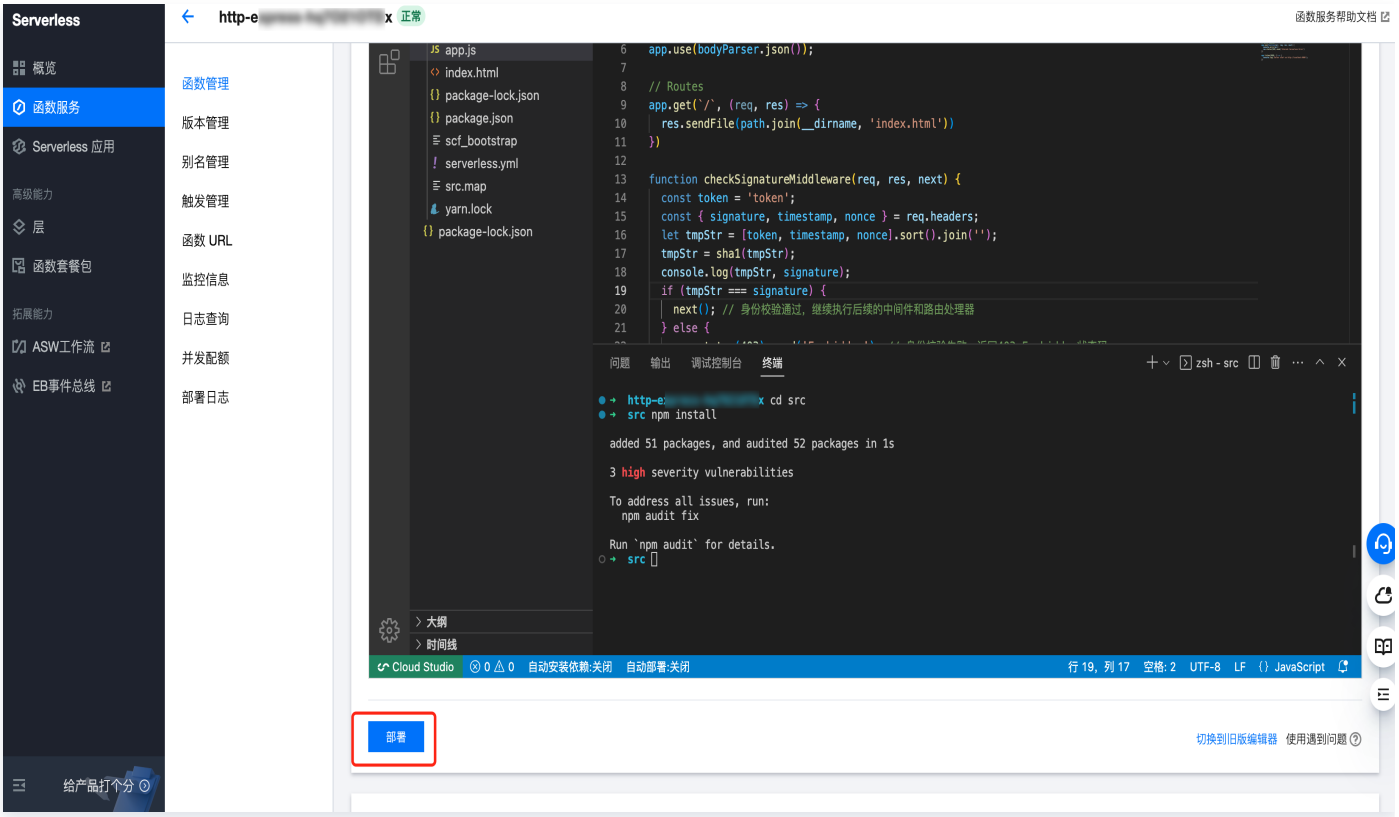
```
const express = require('express')
const path = require('path')
const bodyParser = require('body-parser');
const sha1 = require('js-sha1');
const app = express()
app.use(bodyParser.json());

// 鉴权信息校验的代码在上文已经解释过，这里使用Middleware来简化代码
function checkSignatureMiddleware(req, res, next) {
  const token = 'test';
  const { signature, timestamp, nonce } = req.headers;
  let tmpStr = [token, timestamp, nonce].sort().join('');
  tmpStr = sha1(tmpStr);
  if (tmpStr === signature) {
    next(); // 身份校验通过，继续执行后续的中间件和路由处理器
  } else {
    res.status(403).send('Forbidden'); // 身份校验失败，返回403 Forbidden 状态码
  }
}

app.get('/test', checkSignatureMiddleware, (req, res) => {
  const { echostr } = req.headers;
  res.set('Content-Type', 'text/plain; charset=utf-8');
  res.send(echostr);
});

// 注意接收规则引擎转发的数据接口为 POST 类型
app.post('/test', checkSignatureMiddleware, (req, res) => {
  const params = req.body.payload.params;
  console.log(params); // { body_temperature: 36 }
  res.end();
});
```

6. 上述步骤完成后，需要引入两个依赖包，分别是 body-parser 与 js-sha1。在代码编辑器的上方选择**终端 > 新建终端**，然后在底部终端中输入命令：`cd src` 进入 src 目录，接着分别输入命令 `npm install body-parser` 和 `npm install js-sha1` 安装两个依赖包。





7. 等待依赖安装完成之后，单击下图红色线框中的**部署**，即可成功部署 Web 服务，SCF 会自动启动该 Web 服务，约5秒 – 10秒后系统会提示“部署成功”。



控制台配置规则引擎

创建规则引擎

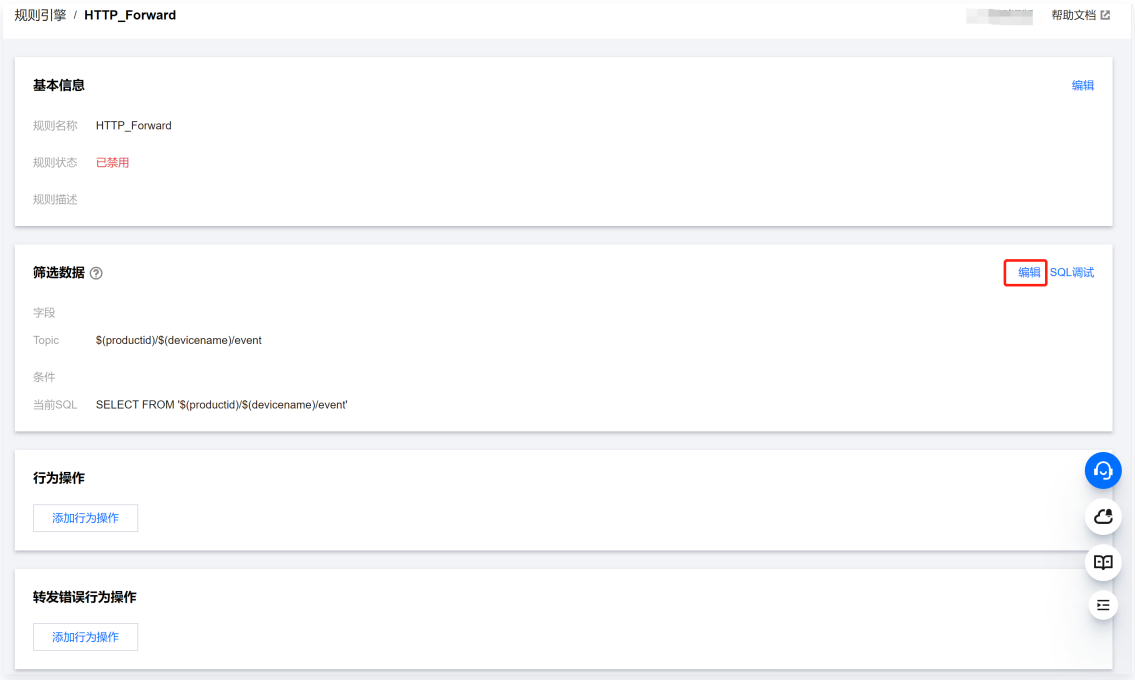
1. 登录 物联网开发平台控制台，选择用户购买的**企业实例**。
2. 单击左侧菜单**数据流转 > 规则引擎**，再次单击规则引擎列表页的**新建规则**。
   - 规则名称：输入“HTTP_Forward”或其他内容。
   - 规则描述：输入转发某类产品下设备数据的备注信息。

3. 规则基本信息填写完成后，单击**确定**，即可完成规则的创建。

配置规则引擎

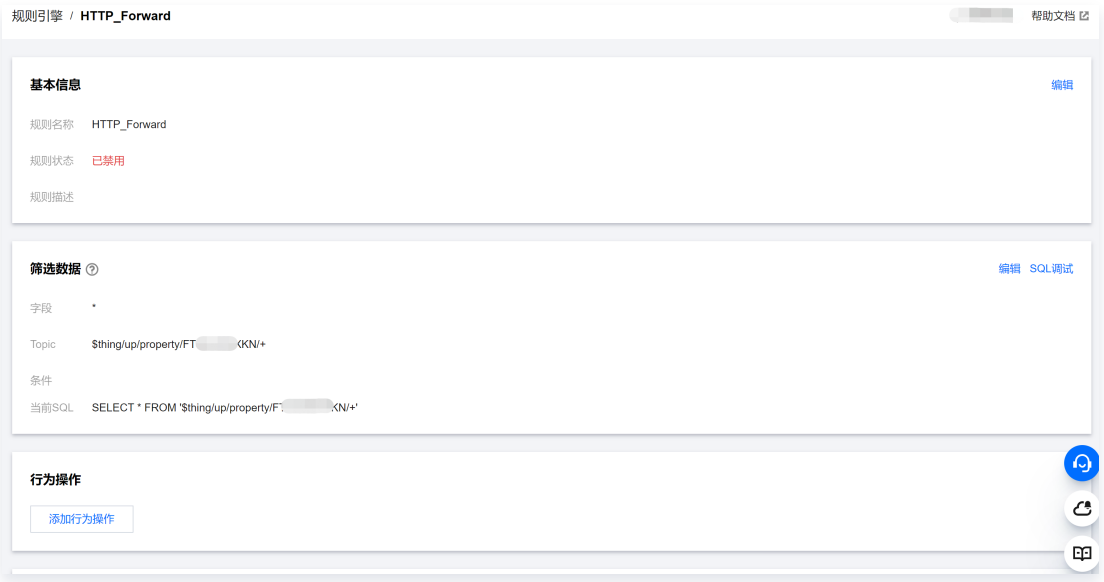
筛选数据

1. 在规则引擎列表选择刚创建成功的规则 **HTTP_Forward**，单击规则名称或操作列的**管理**，进入规则引擎配置页面。
2. 单击下图**筛选数据**右侧的**编辑**，选择要转发的设备数据源。



3. 在编辑规则页面，输入或选择如下信息。
   - 字段：输入"*"，表示将设备上报的所有 JSON 数据进行转发。
   - Topic：首先选择对应的产品。选择入门文档中所创建的产品**车载设备**；第二步选择设备，请选择**全部设备**，表示规格引擎将转发该产品下所有的设备；最后一步选择转发的数据，请选择**物模型属性上报**。

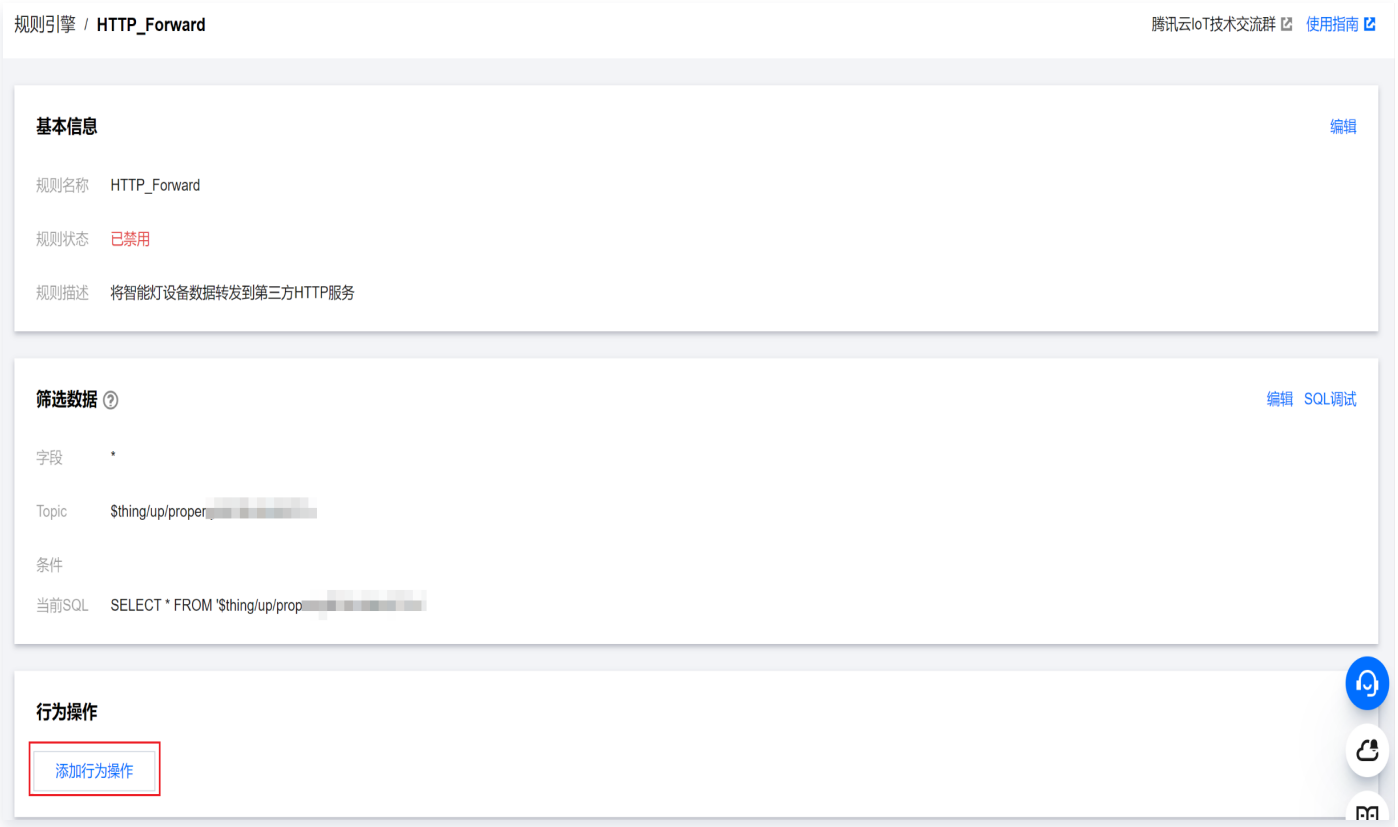
4. 单击**确定**按钮，将返回配置规则页面，系统自动生成如下图所示对应的 TOPIC 及筛选数据 SQL。



配置规则引擎

行为操作配置

1. 在规则引擎列表选择刚创建成功的规则 **HTTP_Forward**，单击规则名称或操作列的**管理**链接，进入规则引擎配置页面。

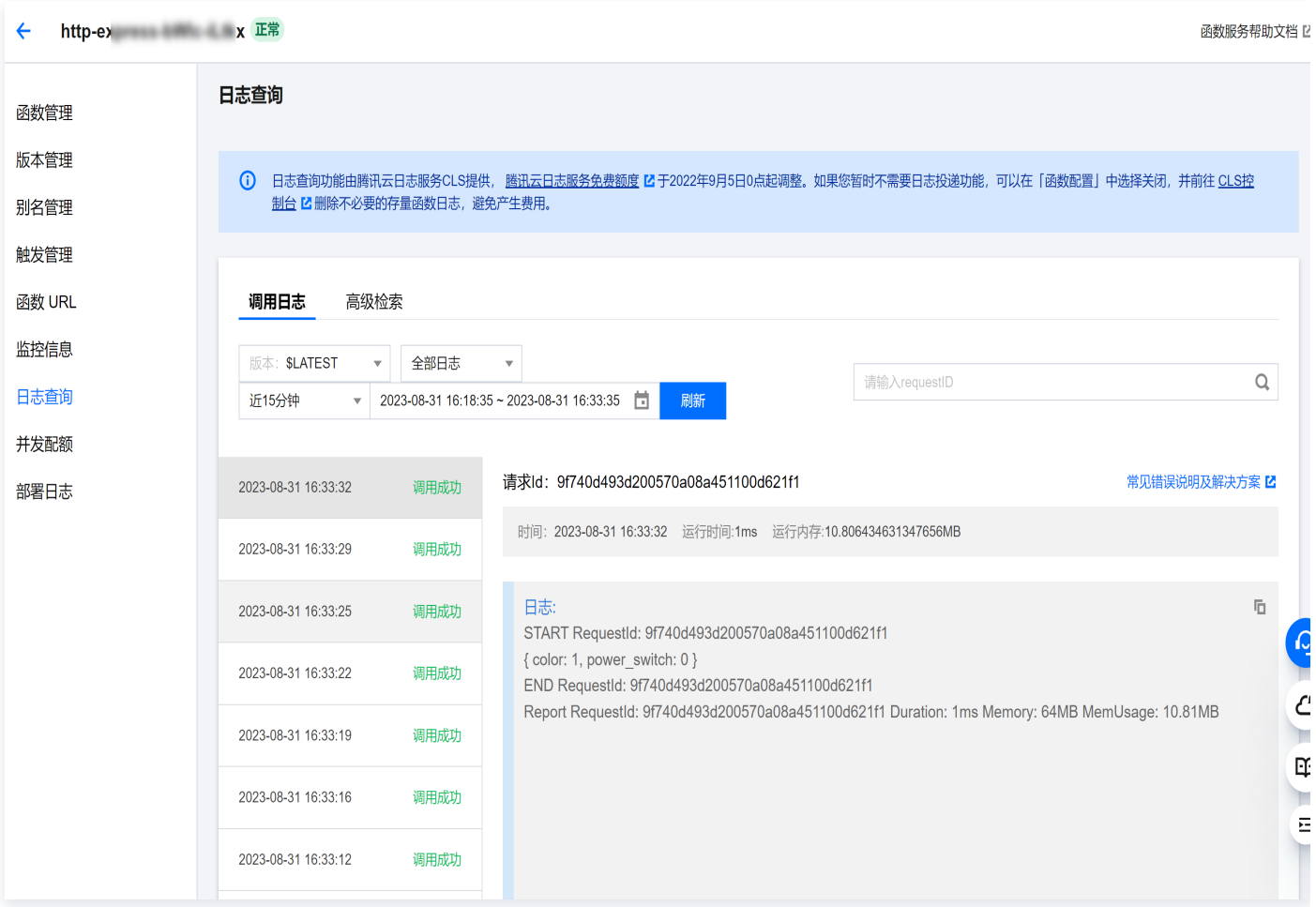


2. 下图行为类型选择“**数据转发到第三方服务（Forward）**”、选择 API 地址类型选择“**使用已有 HTTP 服务地址**”。HTTP 地址则为 **此步骤** 中复制 SCF 自动生成的 URL 地址，并在最后加上 `/test` 。类似 `https://service.*****.apigw.tencenttcs.com/release/test` 。

3. 请勾选 “增加鉴权 Token”，且输入 test ，该 Token 必须与 SCF 函数服务中 app.js 里定义的 Token 值一致。
4. 单击**保存**，完成规则行为配置。
5. 返回规则引擎列表，将该规则的状态设置为 **启用**。

模拟验证

1. 参见 “设备接入及上报数据”，使用模拟程序上报数据。注意上报的数据相关产品需要与规则引擎筛选的目标产品、设备保持一致。若不一致 SCF 日志查询将无法打印输出数据。
2. 进入 SCF 的函数服务，单击**日志查询**，可查看到 SCF 打印的日志，可完整打印出设备上报的物模型属性。



快速体验平台
MQTT.fx 快速接入物联网开发平台

最近更新时间：2024-10-13 10:58:21

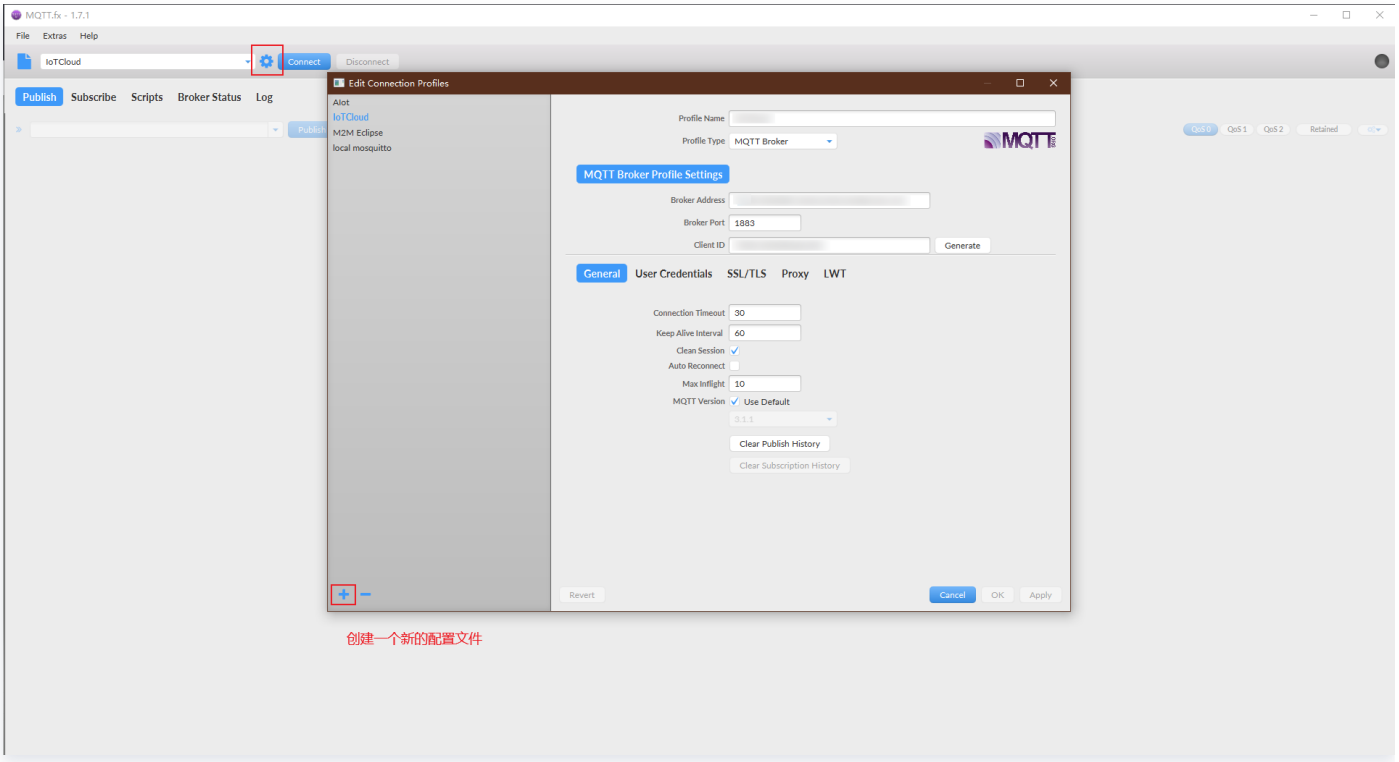
操作场景

MQTT.fx 是目前主流的 MQTT 桌面客户端，它支持 Windows、Mac、Linux 操作系统，可以快速验证是否可与物联网开发平台（简称平台）进行连接，并发布或订阅消息。更多 MQTT 协议介绍请参见 MQTT 协议介绍 。
本文档主要介绍如何使用 MQTT.fx 将设备连接到腾讯云物联网开发平台，如何通过 MQTT.fx 配置 MQTT Client ID、User Name 及 Password 等参数，如何向平台发布消息并订阅消息。本文以 MQTT.fx 1.7.1 for Windows 版本为例，不限于 V1.7.1，最新 MQTT.fx 5.3 同样支持。

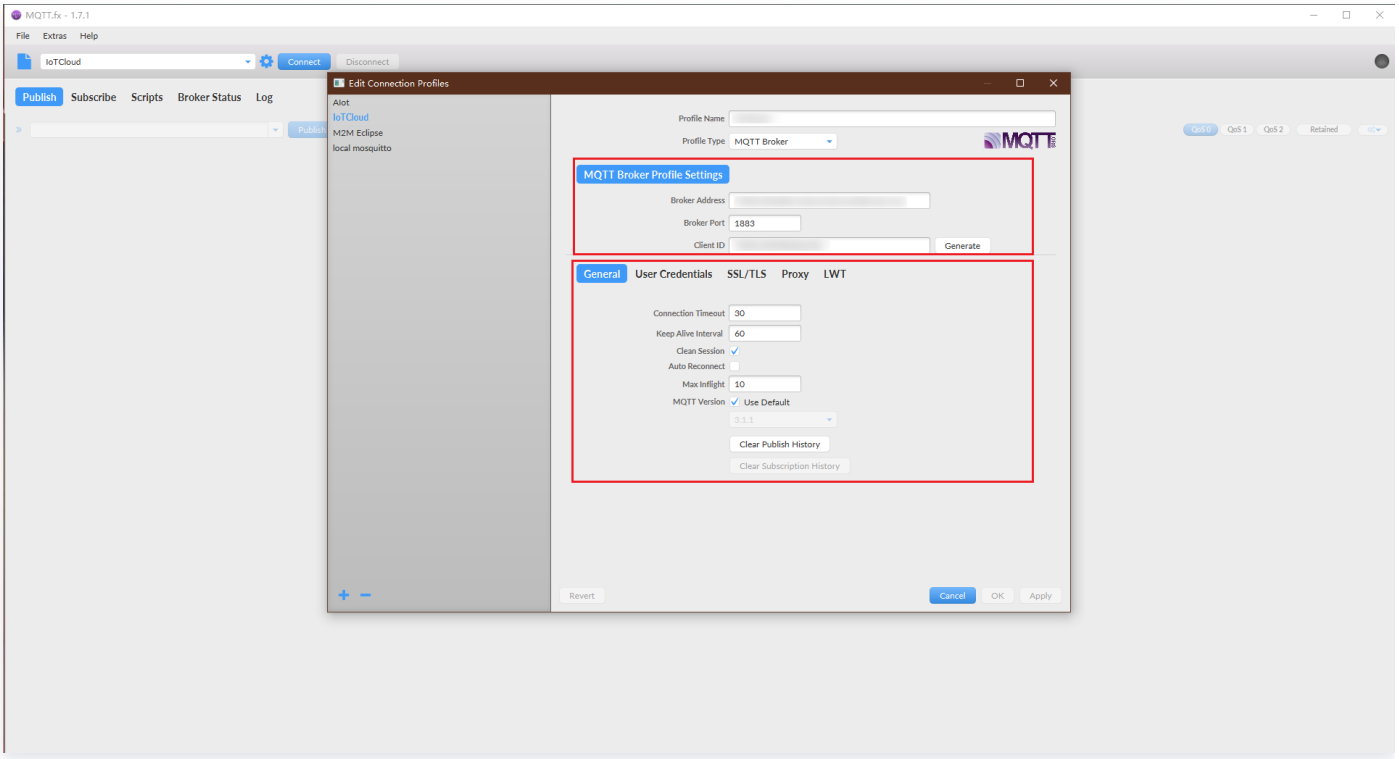
操作步骤

下载 MQTT.fx 并连接平台

1. 下载并安装 MQTT.fx 客户端。
2. 打开 MQTT.fx 客户端程序，单击**设置**。
3. 进入设置页面，并单击“+”，创建一个新的配置文件，输入自定义名称 Profile Name，Profile Type 选择 MQTT Broker。



4. 填写 MQTT Broker Profile Settings 和 General 相关信息。



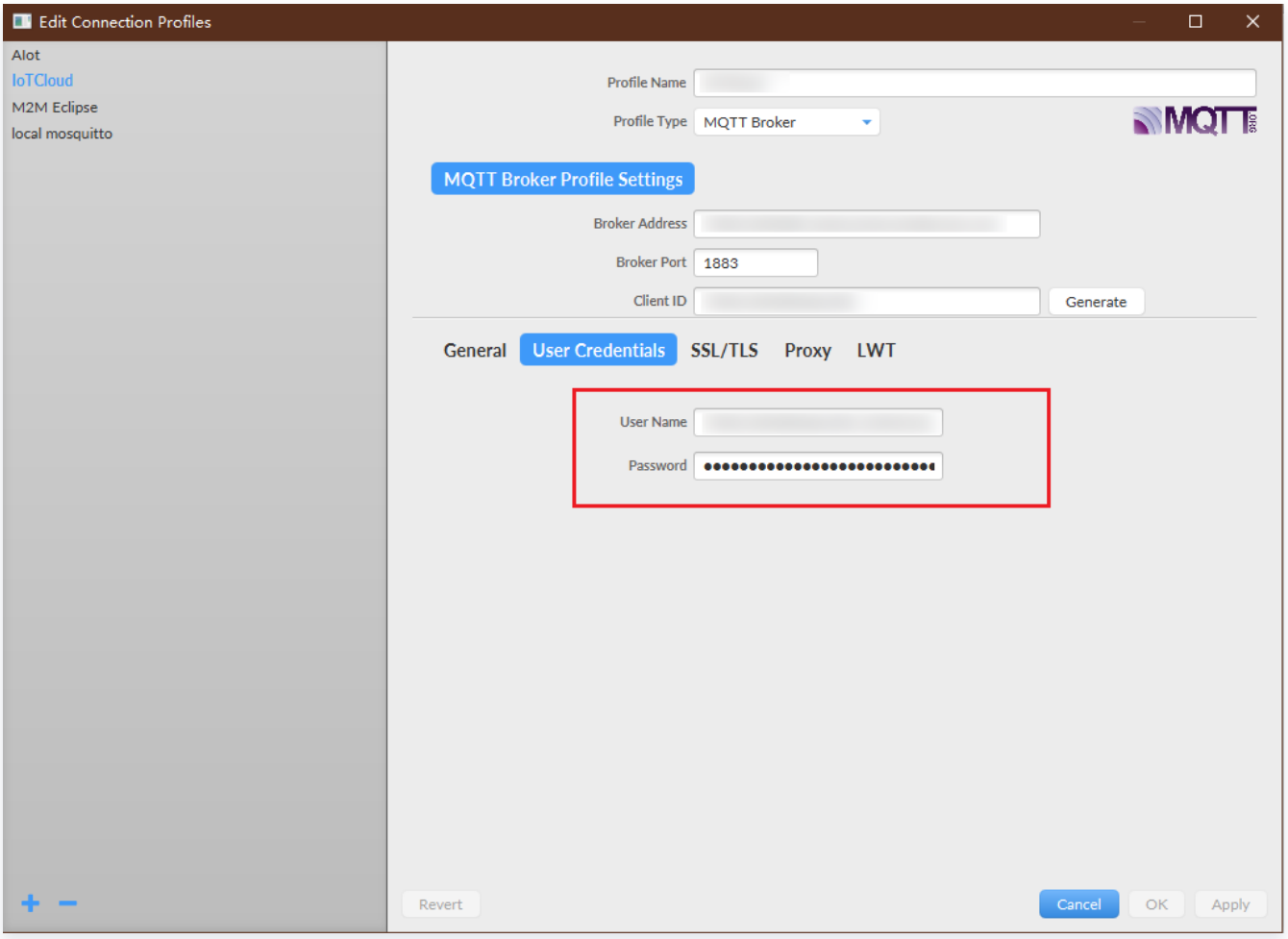
参数说明

参数	说明
Profile Name	用户自定义名称。
Broker Address	MQTT 服务器连接地址，广州域设备填入：PRODUCT_ID.iotcloud.tencentdevices.com，这里 PRODUCT_ID 为变量参数，用户需填入创建产品时自动生成的产品 ID，例如 T****DS8G.iotcloud.tencentdevices.com。
Broker Port	MQTT 服务器连接端口，填入：1883。本文主要针对密钥认证类型的产品，端口必须是1883，如果您想通过8883接口接入，建议使用证书认证型产品自行接入。
Client ID	MQTT 协议字段，按照物联网通信约束填入：产品 ID + 设备名，如："TXXXXDS8Gdev001 "，TXXXXDS8G 是产品 ID，dev001 是设备名称。
Connection Timeout	连接超时时间（秒）。
Keep Alive Interval	心跳间隔时间（秒）。
Auto Reconnect	断网自动重连。

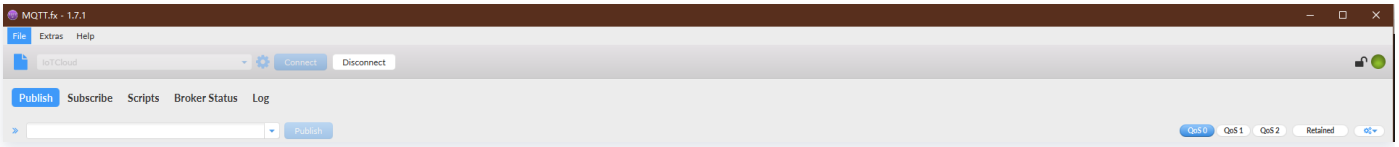
5. 单击 **User Credentials**，填写 User Name 和 Password 。
   - **User Name**：MQTT 协议字段，按照物联网通信约束填入：产品 ID + 设备名 + SDKAppID + connid+expiry。创建完产品即可在产品列表页和产品详情页查看 ProductID，例

如："TO****DS8Gdev001;12010126;E4F3Q;1591948593"，仅替换示例中的产品 ID + 设备名即可，后面的三个参数本身由平台提供的设备接入 SDK 自动生成，也可由平台提供的 生成小工具 自动生成。

- **Password**：Password 必须填写，用户可以使用平台提供的 生成小工具 自动生成 Password，也可以按照文档 手动生成 Password 。



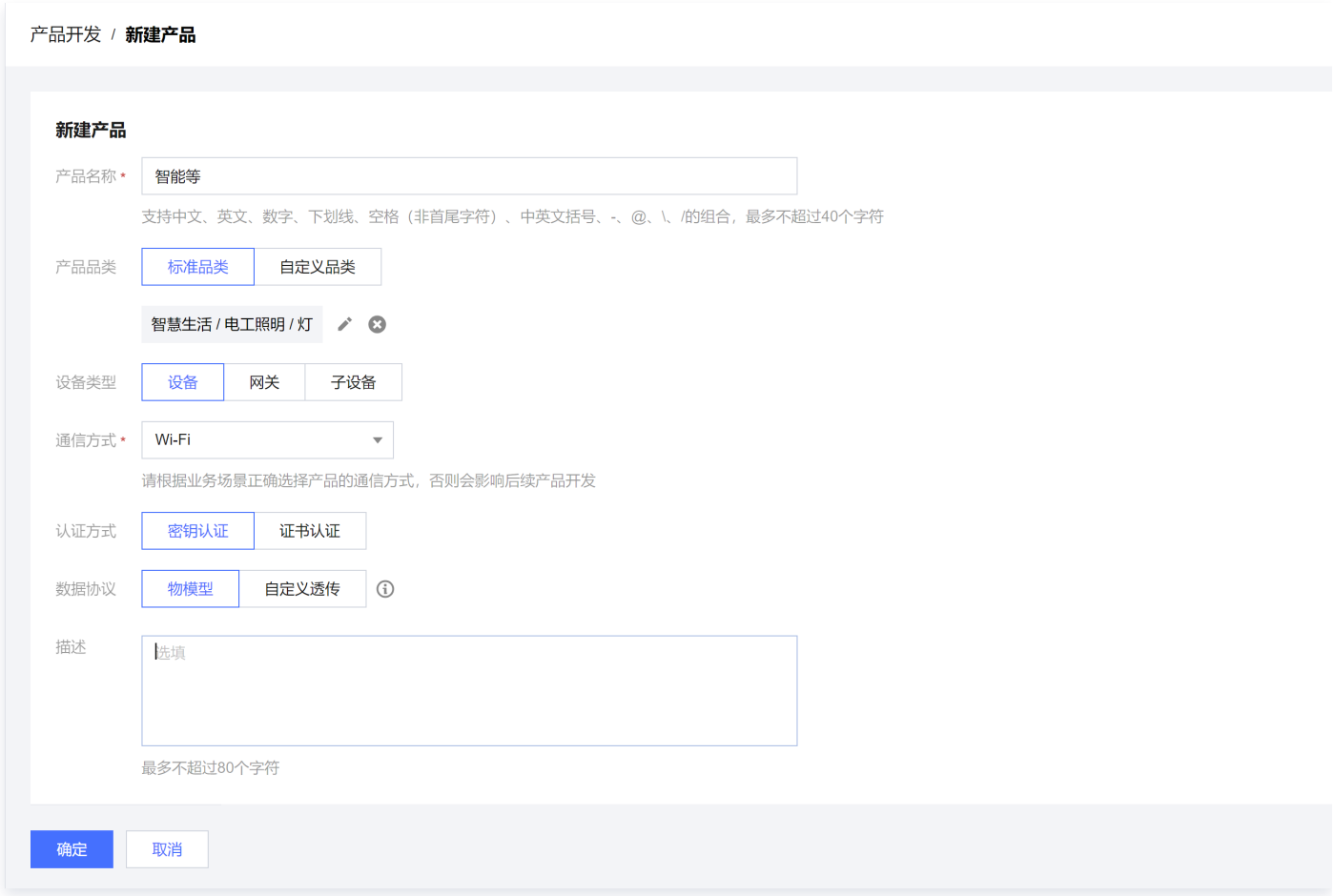
6. 完成以上步骤设置后，单击 **Apply** 和 **OK** 进行保存，并在配置文件框中选择刚才创建的文件名，单击 **Connect**。
7. 当右上角圆形图标为绿色时，说明已成功连接物联网开发平台，即可进行发布和订阅等操作。



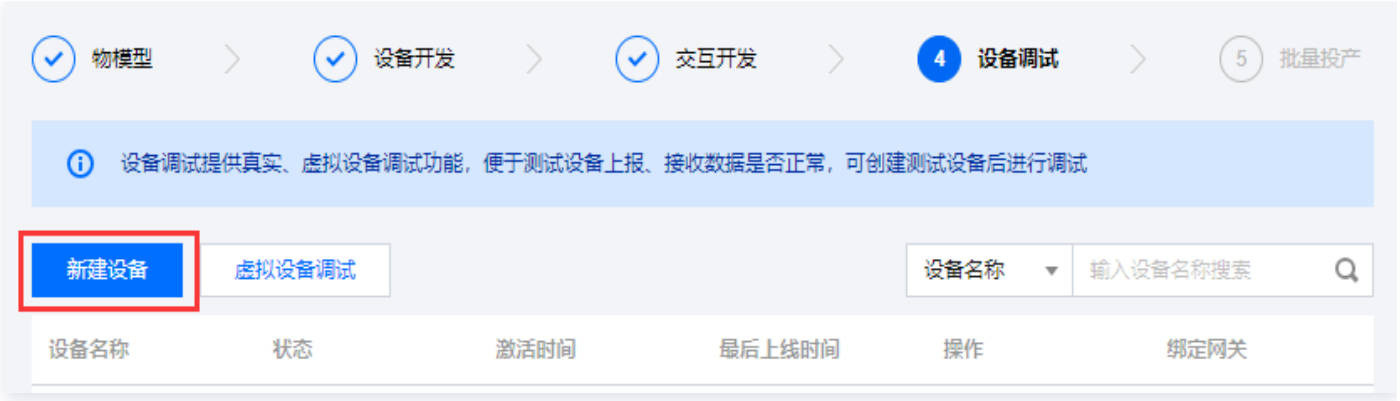
生成 UserName 与 Password

1. 进入 控制台，单击“公共实例”，创建项目后，单击项目名称进入产品开发页，单击**新建产品**，输入产品信息。

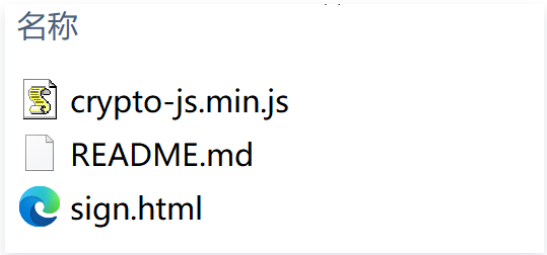
> **注意：**
> 产品品类需选择“智慧生活/电工照明/灯”，系统会自动创建物模型属性、事件等。



2. 单击刚创建的“智能灯”产品，选择第四步**设备调试**，单击**新建设备**创建设备。

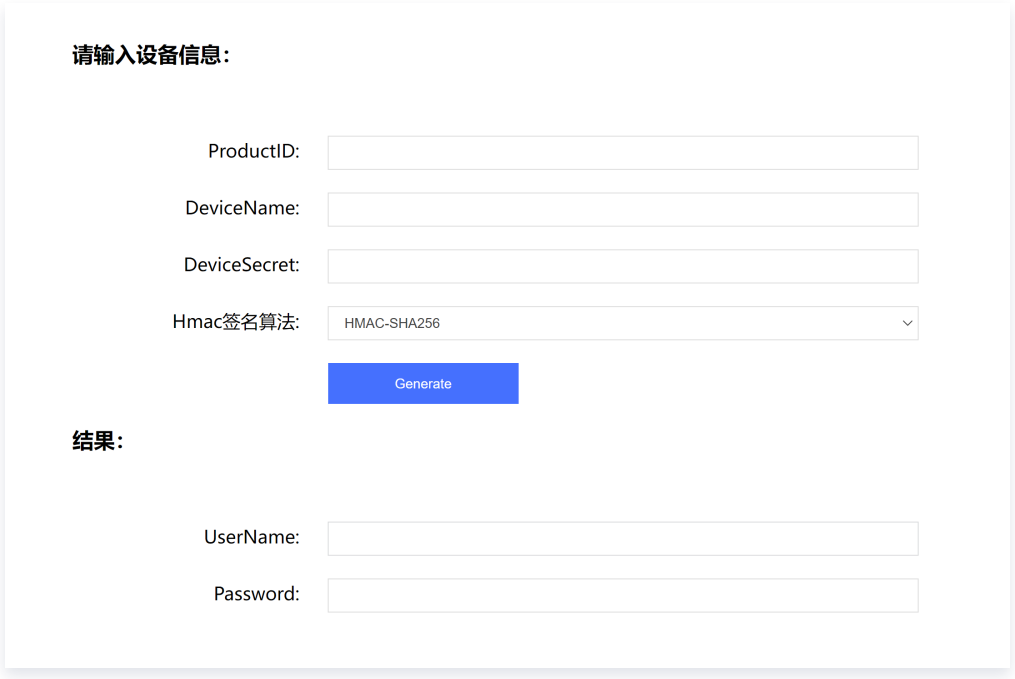


3. 下载网页小工具 生成小工具 并解压缩后，在目录中可查看到以下3个文件。



4. 单击平台**设备调试**中创建的设备名称，获取设备的三元组信息“产品 ID”、“设备名称”和“设备密钥”。

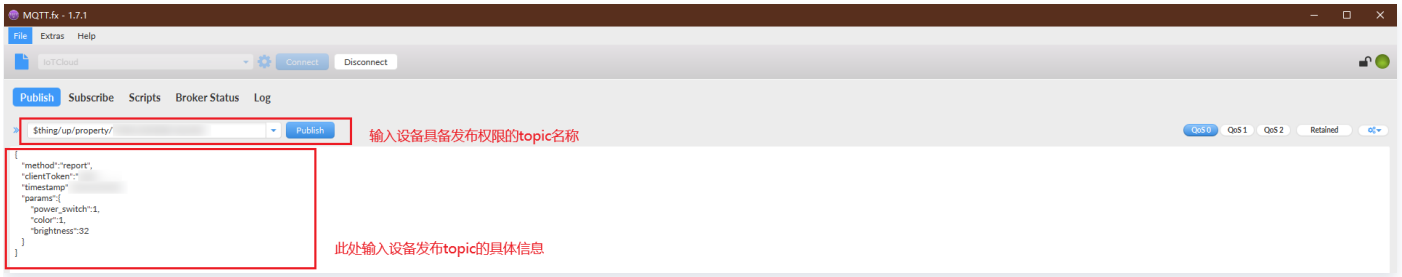
5. 打开 Chrome 浏览器，并打开 `sign.html` 文件，显示如下图。将上一步中“产品 ID”、“设备名称”、“设备密钥”信息分别复制到对应的 ProductID、DeviceName、DeviceSecret 文本框中，签名算法默认选择 HMAC-SHA256，单击 **Generate**，网页工具自动生成 UserName 与 Password。



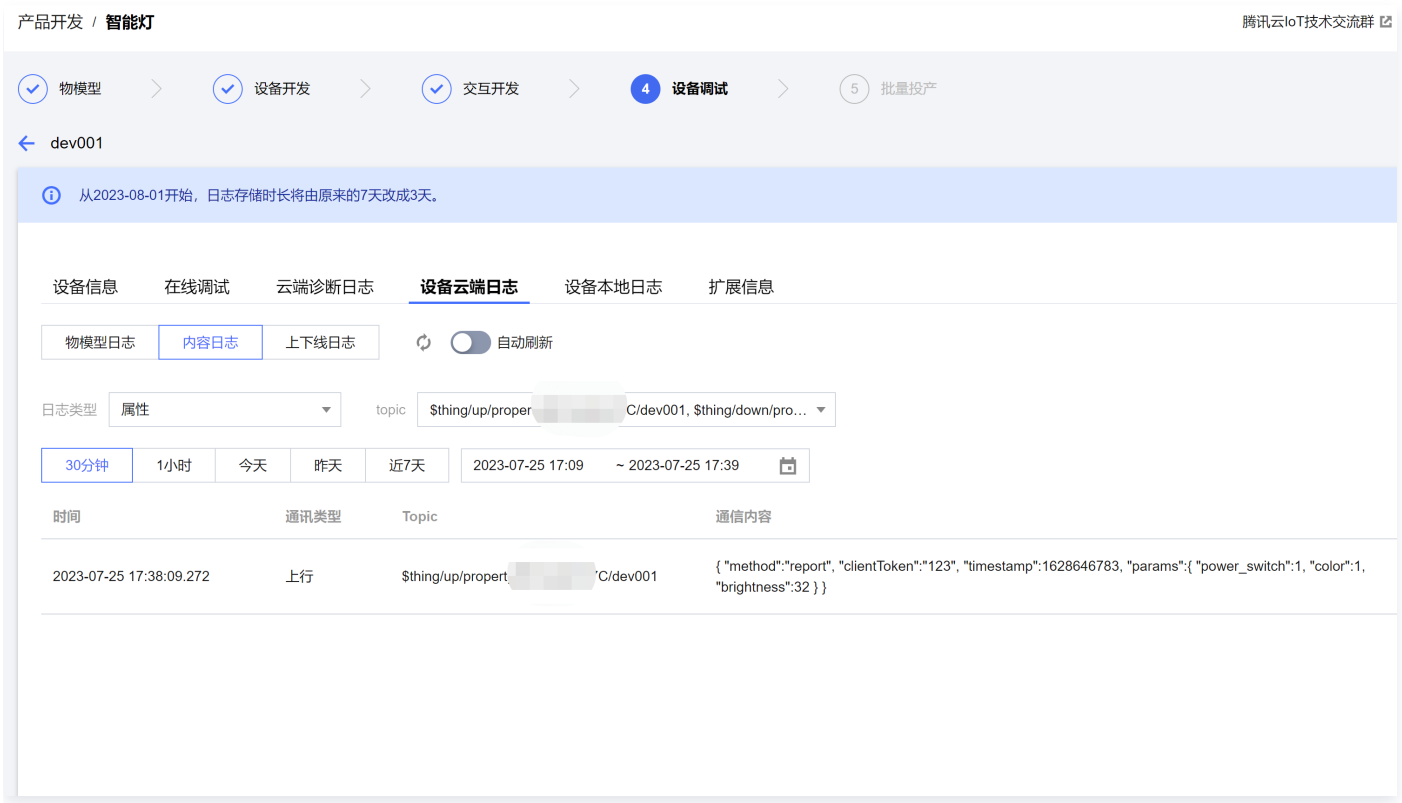
6. 复制自动生成的 User Name、Password 到 MQTT.fx 的 **User Credentials** 区域对应的文本框。

设备发布物模型消息

选择客户端 Publish Tab，输入主题名称、Qos 等级，单击 **Publish** 进行发布。
示例 `Topic：$thing/up/property/ProductID/DeviceName`（设备上报数据到平台的物模型 Topic，实际体验需要将 ProductID 与 DeviceName 替换成用户在平台创建的对应内容）。发布消息数据格式规范及示例，可参见文档 物模型协议。



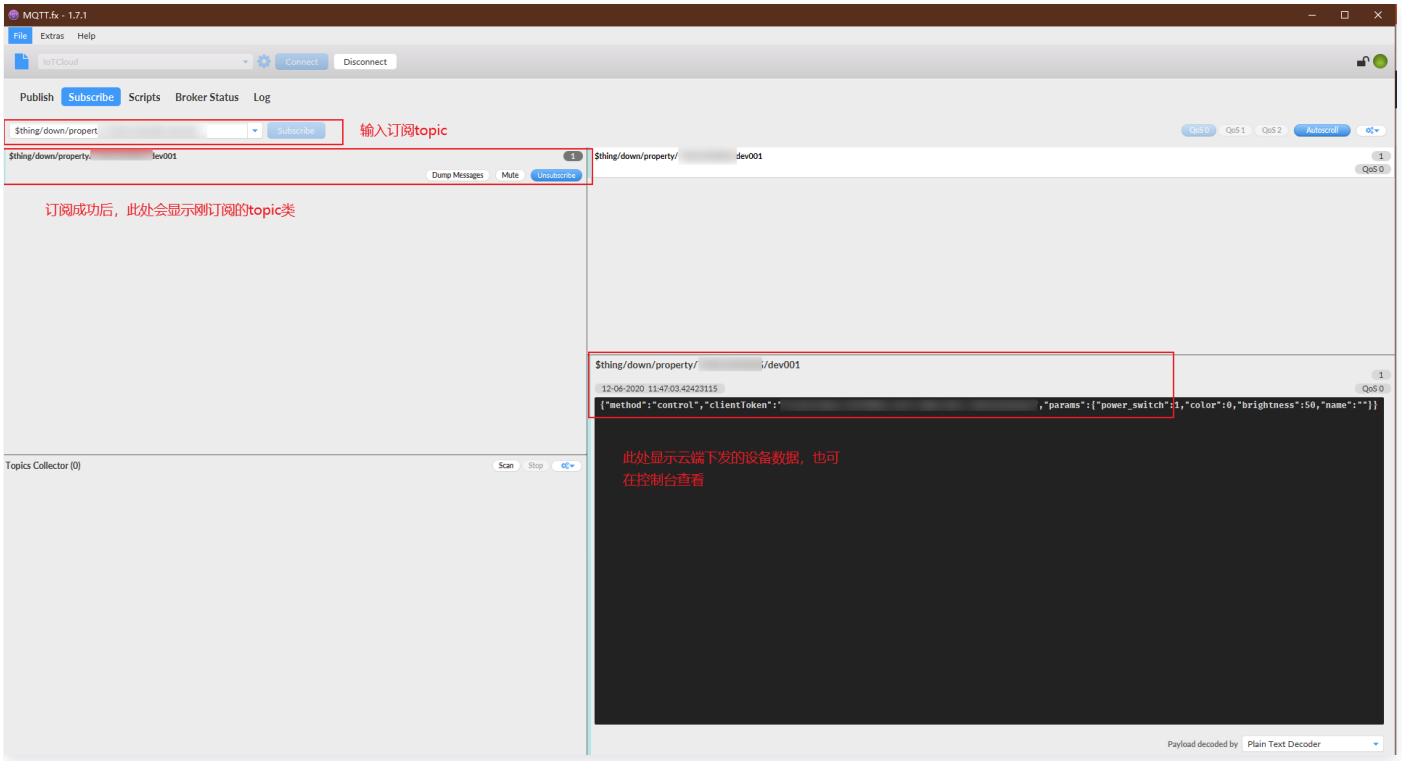
发布结果可通过 控制台 中对应的**设备云端日志**查询到上报记录。



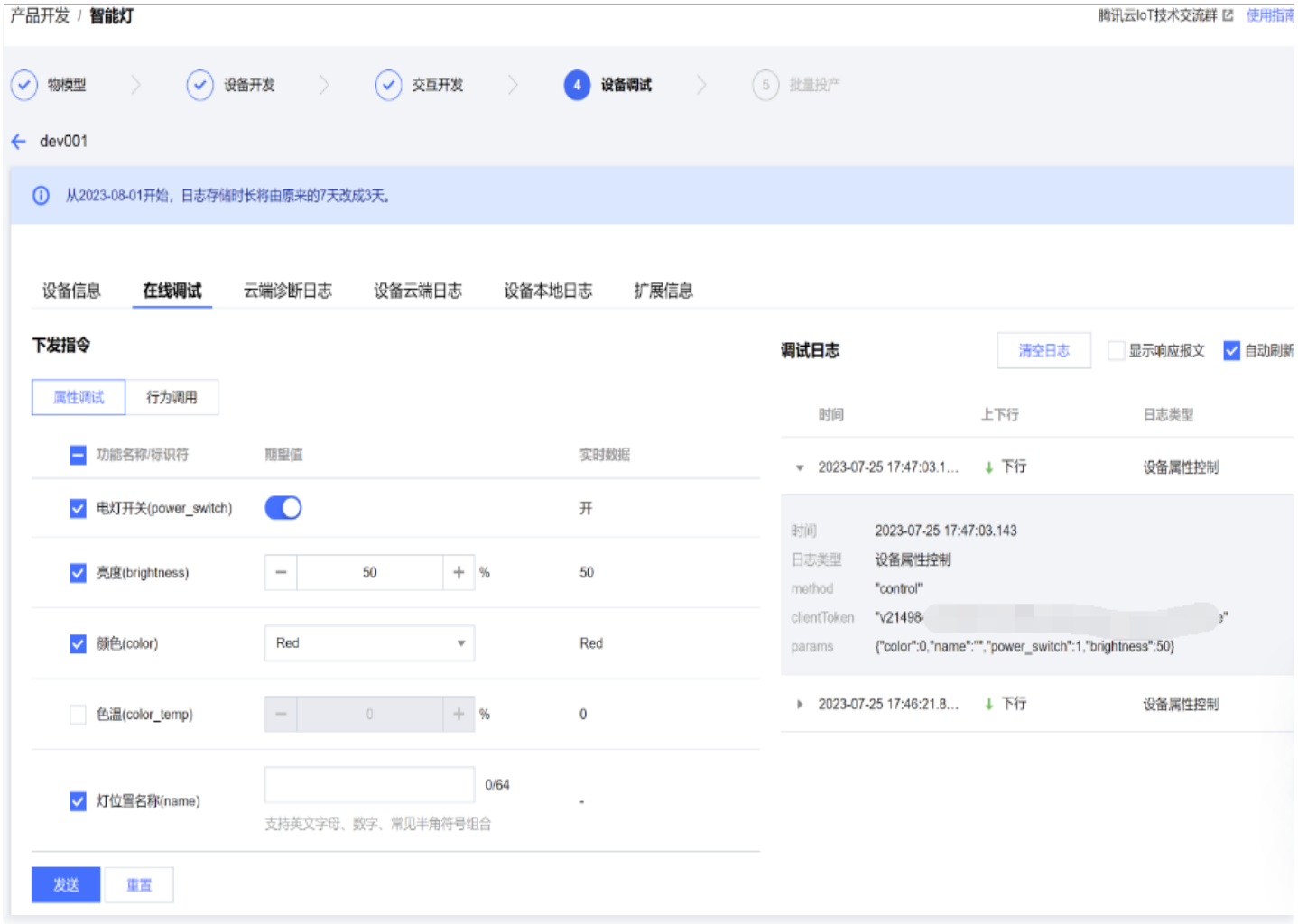
设备订阅物模型消息

选择客户端 Subscribe Tab，输入订阅主题 Topic 名称、Qos 等级，单击 **Subscribe** 进行主题订阅。订阅结果可通过 控制台 的设备日志查询。
示例 `Topic：$thing/down/property/ProductID/DeviceName`（设备订阅物模型 Topic，常用于云端通过物模型下发控

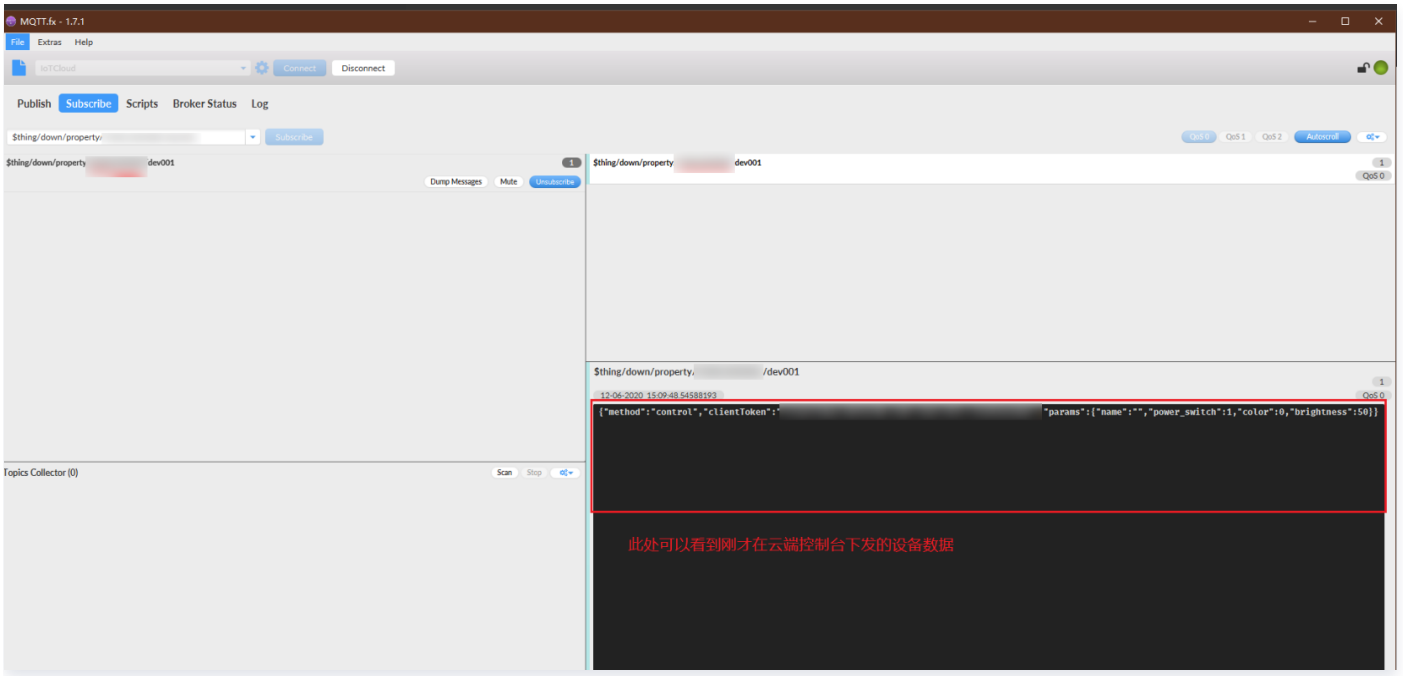
制报文至设备端，实际体验需要将 ProductID 与 DeviceName 替换成用户在平台创建的对应内容）。



进入设备**在线调试**，设置开关，亮度后单击**发送**，平台将下发控制指令。

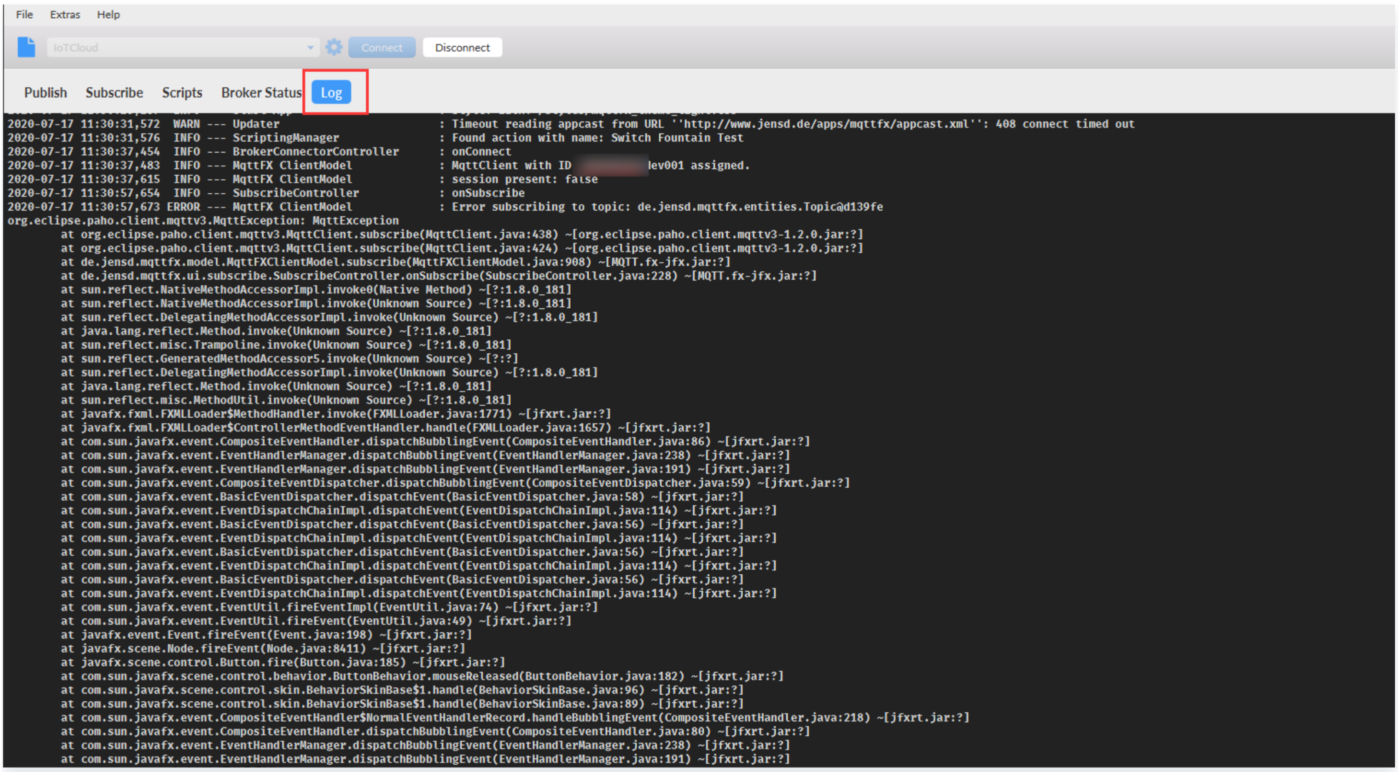


之后回到 MQTT.fx 窗口，可以看到云端刚才下发的控制报文数据：



查看日志

在 MQTT.fx 上，单击 Log 查看操作日志和错误提示日志。



使用MQTT物模型接入平台

最近更新时间：2024-08-14 17:01:21

操作场景

物联网开发平台提供了基于MQTT协议的物模型接入方式，假设一款智能灯接入到物联网开发平台，通过物联网开发平台可以远程控制灯的亮度、颜色、开关，并实时获取智能灯上报到开发平台的数据。本文档主要指导您如何在物联网开发平台控制台接入智能灯。

前提条件

为了通过下面的步骤快速理解该业务场景，需要做好以下准备工作：

- 申请物联网开发平台服务。
- 拥有一台物理或虚拟的 Linux 环境，可以编译、运行 light_demo 程序。
- light_demo 在 Linux 环境下测试和验证，主要基于 Ubuntu 16.04 版本，gcc-5.4（建议至少 gcc-4.7+）。

操作步骤

控制台操作

创建项目

1. 登录 物联网开发平台控制台，选择平台默认开通的**公共实例**或用户购买的**企业实例**。
2. 进入项目列表页面，单击**新建项目**。
   - 项目名称：输入“智能灯演示”或其它名称。
   - 项目描述：按照实际需求填写项目描述。

3. 项目基本信息填写完成后，单击**保存**，即可完成新建项目。
4. 项目新建成功后，即可新建产品。

新建产品

1. 进入该项目的产品列表页面，单击**新建产品**。

2. 在新建产品页面，填写产品基本信息。
   - 产品名称：输入“智能灯”或其它产品名称。
   - 产品品类：选择“智慧城市” > “公共事业” > “路灯照明”。
   - 设备类型：选择“设备”。
   - 认证方式：选择“密钥认证”。
   - 通信方式：按需选择。
   - 其它都为默认选项。

3. 产品信息填写完成后，单击**保存**，即可完成新建产品。
4. 产品新建成功后，您可在产品列表页查看到“智能灯”。

定义产品物模型

选择“智能灯”类型后，系统会自动生成标准功能。

创建设备

在设备调试页面中，单击**新建设备**，设备名为 dev001。

下载 Demo 程序

下载 lightdemo 例程

首先从 GitHub 下载，或执行下面的 git 命令。

```
git clone https://github.com/tencentyun/qcloud-iot-explorer-sdk-embedded-c.git
```

修改 Demo 程序

上述 git 命令执行成功后，会生成一个 `qcloud-iot-sdk-embedded-c` 目录。

1. 进入 `qcloud-iot-explorer-sdk-embedded-c` 目录。
2. 修改该目录下的 device_info.json 文件。

```
> vi  device_info.json
```

```
{
  "auth_mode":"KEY",

  "productId":"[redacted]",
  "productSecret":"YOUR_PRODUCT_SECRET",
  "deviceName":"[redacted]",

  "key_deviceinfo":{
    "deviceSecret":"[redacted]"
  },

  "cert_deviceinfo":{
    "devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
    "devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
  }
}
```

3. 将上图红色线框中的数据分别替换为控制台产品在设备调试阶段，单击选择**设备名称**进入“设备详情页”中的参数并保存。
   - 产品 ID： 将控制台的产品 ID，复制到上图 productId。
   - 设备名称： 将控制台的设备名称，复制到上图 deviceName。
   - 设备密钥：将控制台的设备密钥，复制到上图 deviceSecret。

编译

1. 上述配置信息修改完成后，即可编译。
2. 在 qcloud-iot-sdk-embedded-c 目录下执行以下命令进行编译。

```
./cmake_build.sh
```

3. 编译成功后，会在 output/release/bin 目录下生成 light_data_template_sample 执行文件。

运行 Demo 程序

1. 进入 output/release/bin 目录。
2. 输入 ./light_data_template_sample。
3. 运行成功后，系统输出示例如下：

```
INF|2019-05-07 21:51:33|device.c|iot_device_info_set(65): SDK_Ver: 3.0.0,
Product_ID: BKDDAHRGRX, Device_Name: dev001
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(204): Connecting to
/BKDDAHRGRX.iotcloud.tencentdevices.com/8883...
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(209): Setting up the
SSL/TLS structure...
DBG|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(251): Performing the
SSL/TLS handshake...
INF|2019-05-07 21:51:33|HAL_TLS_mbedtls.c|HAL_TLS_Connect(269): connected with
/BKDDAHRGRX.iotcloud.tencentdevices.com/8883...
```

```
INF|2019-05-07 21:51:33|mqtt_client.c|IOT_MQTT_Construct(115): mqtt connect
with id: ZPEm9 success
DBG|2019-05-07 21:51:33|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(129):
topicName=$template/operation/result/BKDDAHRGRX/dev001|packet_id=35313|pUserdat
a=(null)
DBG|2019-05-07 21:51:33|shadow_client.c|_shadow_event_handler(63): shadow
subscribe success, packet-id=35313
INF|2019-05-07 21:51:33|light_data_template_sample.c|event_handler(222):
subscribe success, packet-id=35313
INF|2019-05-07 21:51:33|shadow_client.c|IOT_Shadow_Construct(173): Sync device
data successfully
INF|2019-05-07 21:51:33|light_data_template_sample.c|main(496): Cloud Device
Construct Success
DBG|2019-05-07 21:51:33|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(129):
topicName=$thing/down/event/BKDDAHRGRX/dev001|packet_id=35314|pUserdata=(null)
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=power_switch registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=color registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=brightness registered.
INF|2019-05-07
21:51:33|light_data_template_sample.c|_register_data_template_property(370):
data template property=name registered.
INF|2019-05-07 21:51:33|light_data_template_sample.c|main(517): Register data
template propertys Success
DBG|2019-05-07 21:51:33|shadow_client.c|IOT_Shadow_Get(384): GET Request
Document: {"clientToken":"BKDDAHRGRX-0"}
DBG|2019-05-07 21:51:33|mqtt_client_publish.c|qcloud_iot_mqtt_publish(337):
publish packetID=0|topicName=$template/operation/BKDDAHRGRX/dev001|payload=
{"type":"get", "clientToken":"BKDDAHRGRX-0"}
DBG|2019-05-07 21:51:33|shadow_client.c|_shadow_event_handler(63): shadow
subscribe success, packet-id=35314
INF|2019-05-07 21:51:33|light_data_template_sample.c|event_handler(222):
subscribe success, packet-id=35314
DBG|2019-05-07 21:51:33|shadow_client.c|_update_ack_cb(114): requestAck=0
DBG|2019-05-07 21:51:33|shadow_client.c|_update_ack_cb(117): Received Json
Document={"clientToken":"BKDDAHRGRX-0","payload":{"metadata":{"reported":
{}},"state":{"reported":
{}},"timestamp":1557236942,"version":0},"result":0,"timestamp":1557237093,"type
":"get"}
DBG|2019-05-07 21:51:34|light_data_template_sample.c|main(602): cycle report:
{"version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
```

```
DBG|2019-05-07 21:51:34|shadow_client.c|IOT_Shadow_Update(318): UPDATE Request
Document: {"version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
DBG|2019-05-07 21:51:34|mqtt_client_publish.c|qcloud_iot_mqtt_publish(337):
publish packetID=0|topicName=$template/operation/BKDDAHRGRX/dev001|payload=
{"type":"update", "version":0, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-1"}
INF|2019-05-07 21:51:34|light_data_template_sample.c|main(607): shadow
update(reported) success
INF|2019-05-07
21:51:35|light_data_template_sample.c|OnShadowUpdateCallback(352): recv shadow
update response, response ack: 0
DBG|2019-05-07 21:51:40|light_data_template_sample.c|main(602): cycle report:
{"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
DBG|2019-05-07 21:51:40|shadow_client.c|IOT_Shadow_Update(318): UPDATE Request
Document: {"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
```

4. Light Demo 程序定时会上报数据到开发平台，数据格式如下：

```
{"version":1, "state":{"reported":
{"power_switch":0,"color":0,"brightness":0.000000,"name":"dev001"}},
"clientToken":"BKDDAHRGRX-2"}
```

5. 继续保持 Light Demo 程序处于运行状态，然后前往控制台查看该设备的数据。

查看设备状态

1. 保持 light Demo 程序为运行状态。
2. 进入**控制台 > 产品开发 > 设备调试**，可查看到设备 “dev001” 的状态为“上线”状态，表示 Demo 程序已成功连接上开发平台。
3. 单击**查看**，可进入设备详情页。
4. 单击**设备属性**，可查询设备上报到开发平台的最新数据及历史数据。
   - 当前上报数据的最新值：会显示设备上报的最新数据。
   - 当前上报数据的更新时间：显示数据的更新时间。

5. 单击**查看**，可查看某个属性的历史上报数据。

查看设备通信日志

1. 单击**设备日志**，可查询该设备某段时间范围的所有上下行数据。
   - 上行：上行指设备端上报到开发平台的数据。
   - 下行：下行指从开发平台下发到设备的数据。

在线调试

1. 当 Light Demo 成功连接到物联网开发平台后，您可在控制台**设备调试**列表，单击**调试**，进入在线调试。

2. 将亮度设置为68，颜色设置为“Red”，单击**发送**。
3. 查看 Light Demo 程序，可查看到成功接收到下发的数据。

```
lighting  ][color: RED ][brightness:||||||||||||-------][dev001]
```

4. 通信日志会显示如下日志，表示成功下发了指令到设备端。

```
{
"method": "control",
"clientToken": "123",
"params": {
    "power_switch": 1,
```

```
    "color": 1,
    "brightness": 68
}
}
```

5. 查看通信日志，即可查看到设备成功接收到下行指令，并上报最新数据到开发平台的详细日志。

使用自定义透传上下行消息

最近更新时间：2024-09-30 17:54:51

操作场景

MQTT 自定义透传接入平台主要用于设备通过 DTU 或边缘网关接入物联网平台的情况。当设备无法直接使用物联网开发平台的物模型业务格式时，可以选择使用 MQTT 透传报文的模式。在这种应用场景下，可以利用平台支持的 MQTT 自定义透传报文能力将设备数据上传到云端。通过本文档可以快速了解如何使用平台将自定义透传协议类的设备上云，以及如何通过云 API 远程控制使用自定义透传协议的设备。

前提条件

为了快速了解该业务场景，需要提前做好以下准备工作：

- 申请物联网开发平台服务。直接 注册腾讯云账号 即可立即开通物联网开发平台。
- 拥有一台物理或虚拟的 Linux 环境，可以编译、运行 C SDK 中的 raw_data_mqtt_sample 程序。
- raw_data_mqtt_sample 在 Linux 环境下测试和验证，主要基于 Ubuntu 16.04 版本，gcc-5.4（建议至少 gcc-4.7+）。

操作步骤

创建项目

1. 登录 物联网开发平台控制台，单击“公共实例”框进入项目列表页面。
2. 单击**新建项目**，在新建项目页面，填写项目基本信息。

   - 项目名称：输入“自定义透传设备上云”或其他名称。
   - 项目描述：按照实际需求填写项目描述。
3. 项目基本信息填写完成后，单击**保存**，即可完成新建项目。
4. 项目新建成功后，即可新建产品。

新建产品

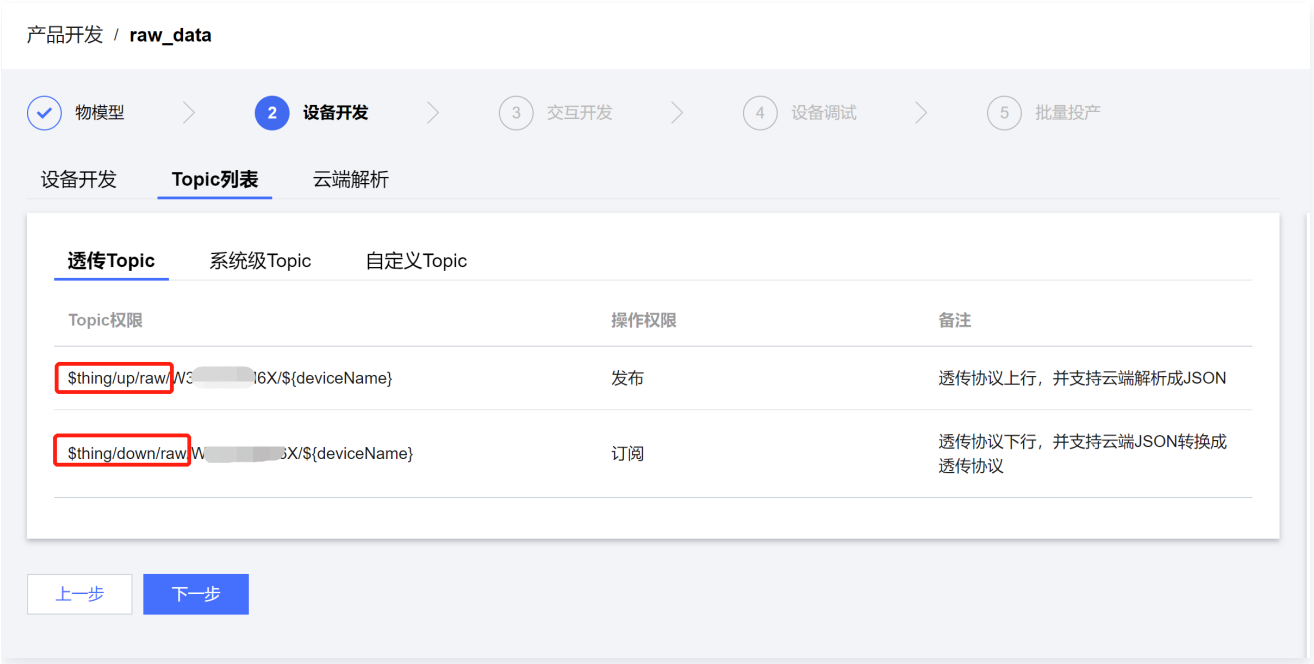
1. 点击创建成功的项目“自定义透传设备上云”，单击**产品开发**，进入该项目的产品列表页面，单击**新建产品**。
2. 在新建产品页面，填写产品基本信息。

- 产品名称：输入“raw_data”或其他产品名称。
- 产品品类：选择标准品类，并选择“智慧生活->电工照明->灯”。
- 设备类型：选择“设备”。
- 认证方式：选择“密钥认证”。
- 通信方式：默认 Wi-Fi。
- 数据协议：选择“自定义透传”
- 描述：根据需要进行填写。

3. 产品信息填写完成后，单击**保存**，即可完成新建产品。
4. 产品新建成功后，您可在产品列表页查看到“raw_data”。

设备开发

单击**设备开发 > Topic 列表 > 透传 Topic**，当产品的数据协议为“自定义透传”类型时，设备开发界面如下图所示。



说明：

- 平台默认创建了 Topic，其中上图红色线框为使用自定义透传协议的系统约定的上行、下行 Topic。
- 平台支持用户使用自定义 Topic 自由定义，以 $thing 开头的 Topic 支持云端脚本解析能力，若需要通过平台云端 JS 脚本解析自定义透传数据为物模型格式，则必须使用 $thing/up/raw 开头的 Topic。

创建设备

1. 单击**设备调试 > 新建设备**，输入设备名称为“dev001”，单击**保存**。

2. 单击**设备名称**“dev001”，进入设备详情页，查看产品 ID、设备名称和设备密钥等信息。此部分信息将会在编译 raw_data_mqtt_sample 时需要使用。

下载 Demo 程序

下载 raw_data_mqtt_sample 例程

进入 Linux 主机（建议使用腾讯云 CVM），下载 SDK 的方式有以下两种：

- 从 GitHub 下载 SDK 直接上传至云主机。
- 执行以下 git 命令获取。若 git clone 执行失败则需要设置云主机安全组的入站、出站规则是否正确。

```
git clone https://github.com/tencentyun/qcloud-iot-explorer-sdk-embedded-c.git
```

编译

1. 在 qcloud-iot-sdk-embedded-c 目录下执行以下命令进行编译。

```
./cmake_build.sh
```

2. 编译成功后，可在 output/release/bin 目录下生成 raw_data_mqtt_sample 执行文件及 device_info.json 文件。若编译失败，需要检查 gcc 版本是否是8.5以上。

修改配置文件

1. 进入 output/release/bin 目录。
2. 输入以下命令，修改该目录下的 device_info.json 文件。

```
vi  device_info.json
```

3. 将红色线框中的数据分别替换为控制台创建的设备“dev001”对应的产品 ID、设备名称、设备密钥信息。

```
{
  "auth_mode":"KEY",

  "productId":"[redacted]",
  "productSecret":"YOUR_PRODUCT_SECRET",
  "deviceName":"[redacted]",

  "key_deviceinfo":{
    "deviceSecret":"[redacted]"
  },

  "cert_deviceinfo":{
    "devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
    "devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
  }
}
```

   - 产品 ID： 设备“dev001”的产品 ID，复制到上图 productId。
   - 设备名称： 设备“dev001”的设备名称，复制到上图 deviceName。
   - 设备密钥：设备“dev001”的设备密钥，复制到上图 deviceSecret。
4. 保存 device_info.json 文件即可。

运行 Demo 程序

1. 进入 output/release/bin 目录。
2. 输入 ./raw_data_mqtt_sample 并执行。
3. 运行成功后，输出示例如下图红框所示。

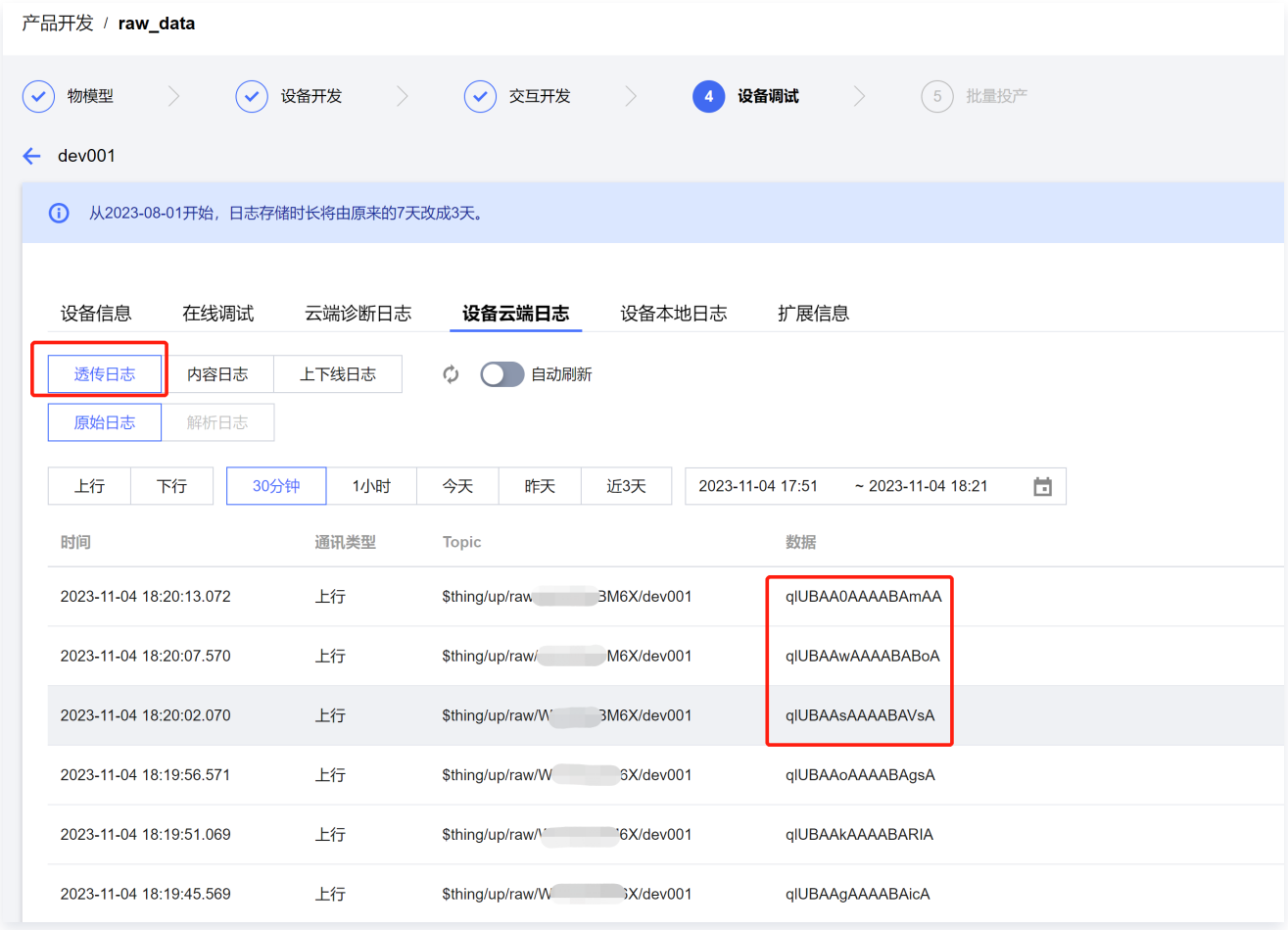
```
DBG|2021-02-19 10:42:50|HAL_TLS_mbedtls.c|HAL_TLS_Connect(223): Setting up the SSL/TLS structure...
DBG|2021-02-19 10:42:50|HAL_TLS_mbedtls.c|HAL_TLS_Connect(265): Performing the SSL/TLS handshake...
DBG|2021-02-19 10:42:50|HAL_TLS_mbedtls.c|HAL_TLS_Connect(266): Connecting to /ALMUQ6FAB3.iotcloud.tencentdevices.com/8883...
DBG|2021-02-19 10:42:50|HAL_TLS_mbedtls.c|HAL_TLS_Connect(288): connected with /ALMUQ6FAB3.iotcloud.tencentdevices.com/8883...
INF|2021-02-19 10:42:50|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: qkkSg success
INF|2021-02-19 10:42:50|raw_data_sample.c|main(308): Cloud Device Construct Success
DBG|2021-02-19 10:42:50|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(147): topicName=$thing/down/raw/ALMUQ6FAB3/dev001|packet_id=57840
INF|2021-02-19 10:42:50|raw_data_sample.c|_mqtt_event_handler(96): subscribe success, packet-id=57840
DBG|2021-02-19 10:42:50|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:

 AA 55 01 00 00 00 00 00 01 01 1E 00
DBG|2021-02-19 10:42:50|mqtt_client_publish.c|qcloud_iot_mqtt_publish(345): publish packetID=0|topicName=$thing/up/raw/ALMUQ6FAB3/dev001|payload=
INF|2021-02-19 10:42:51|mqtt_client_connect.c|qcloud_iot_mqtt_disconnect(474): mqtt disconnect!
INF|2021-02-19 10:42:51|mqtt_client.c|IOT_MQTT_Destroy(189): mqtt release!
```

4. Demo 程序会上报数据到平台，上报数据为下方16进制数据。

```
AA 55 01 00 00 00 00 00 01 01 1E 00
```

查看设备上报数据

1. 进入 物联网开发平台控制台 > **产品开发** > **设备调试**，单击**设备名称** "dev001" 进入设备详情页。
2. 单击**设备云端日志**，系统会显示“透传日志”，即可在控制台查看该设备上报的数据。



3. 上图 Topic 列为设备向云端发布消息时的 Topic，数据列红色线框中的数据为平台 base64 编码后的数据。

使用自定义透传模式远程控制设备

1. 当需要从云端控制使用自定义透传协议上云的设备时，可参考 设备透传指令控制 云 API 直接使用 在线调试 工具进行调试。
2. API Explorer 调试工具如下图所示。

3. 在发送透传报文指令至设备端前，需运行 `./raw_data_mqtt_sample -l 10` 命令连接到平台，保持连接以便接收平台下发的指令报文。
4. 当看到 demo 程序接收到云端下发的指令报文时，如下图红色线框中的报文为16进制。

```
 AA 55 01 00 26 00 00 00 01 01 38 00
DBG|2023-11-04 18:40:22|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish packetID=0|topicName=$thing/up/raw/W3VZ6JBM6X/dev001|payload=
DBG|2023-11-04 18:40:28|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:

 AA 55 01 00 27 00 00 00 01 02 3C 00
DBG|2023-11-04 18:40:28|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish packetID=0|topicName=$thing/up/raw/W3VZ6JBM6X/dev001|payload=
DBG|2023-11-04 18:40:33|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:

 AA 55 01 00 28 00 00 00 01 02 40 00
DBG|2023-11-04 18:40:33|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish packetID=0|topicName=$thing/up/raw/W3VZ6JBM6X/dev001|payload=
INF|2023-11-04 18:40:33|raw_data_sample.c|on_raw_data_message_callback(235): Receive Message With topicName:$thing/down/raw/W 6X/dev001, payloadlen:12
DBG|2023-11-04 18:40:33|raw_data_sample.c|on_raw_data_message_callback(238): raw_data reveived dump:

 AA 55 01 00 08 00 00 00 01 02 27 00
DBG|2023-11-04 18:40:39|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:

 AA 55 01 00 29 00 00 00 01 02 0A 00
DBG|2023-11-04 18:40:39|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish packetID=0|topicName=$thing/up/raw/W3VZ6JBM6X/dev001|payload=
DBG|2023-11-04 18:40:44|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:

 AA 55 01 00 2A 00 00 00 01 02 5C 00
DBG|2023-11-04 18:40:44|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish packetID=0|topicName=$thing/up/raw/W3VZ6JBM6X/dev001|payload=
DBG|2023-11-04 18:40:50|raw_data_sample.c|_publish_raw_data_msg(223): raw_data published dump:
```

5. 控制台的设备云端日志会显示如下“下行”报文。


设备信息
在线调试
云端诊断日志
设备云端日志
设备本地日志
扩展信息
透传日志
内容日志
上下线日志
自动刷新
原始日志
解析日志
上行
下行
30分钟
1小时
今天
昨天
近3天
2023-11-04 18:11 ~ 2023-11-04 18:41
时间 通讯类型 Topic 数据
2023-11-04 18:40:55.620 上行 $thing/up/raw/W3 /dev001 qlUBACwAAAABABkA
2023-11-04 18:40:50.117 上行 $thing/up/raw/W3 /dev001 qlUBACsAAAABASsA
2023-11-04 18:40:44.617 上行 $thing/up/raw/W3 /dev001 qlUBACoAAAABAlwA
2023-11-04 18:40:39.118 上行 $thing/up/raw/W3 /dev001 qlUBACkAAAABAgoA
2023-11-04 18:40:33.612 上行 $thing/up/raw/W3 /dev001 qlUBACgAAAABAkAA
2023-11-04 18:40:32.534 下行 $thing/down/raw/W3 /dev001 qlUBAAgAAAABAicA
2023-11-04 18:40:28.110 上行 $thing/up/raw/W3 /dev001 qlUBACcAAAABAjwA

云端控制设备入门

最近更新时间：2024-10-03 11:18:11

操作场景

设备接入到物联网开发平台后，一般会通过用户的业务系统发起控制设备的操作。例如，智能公寓场景 App 远程为房间门锁设置密码，可以通过物联网开发平台提供的云 API 或在线调试功能下发消息至设备侧，以实现用户业务系统通过物联网平台控制、管理设备。

前提条件

为了快速体验控制设备业务功能，需要提前做好以下准备工作：

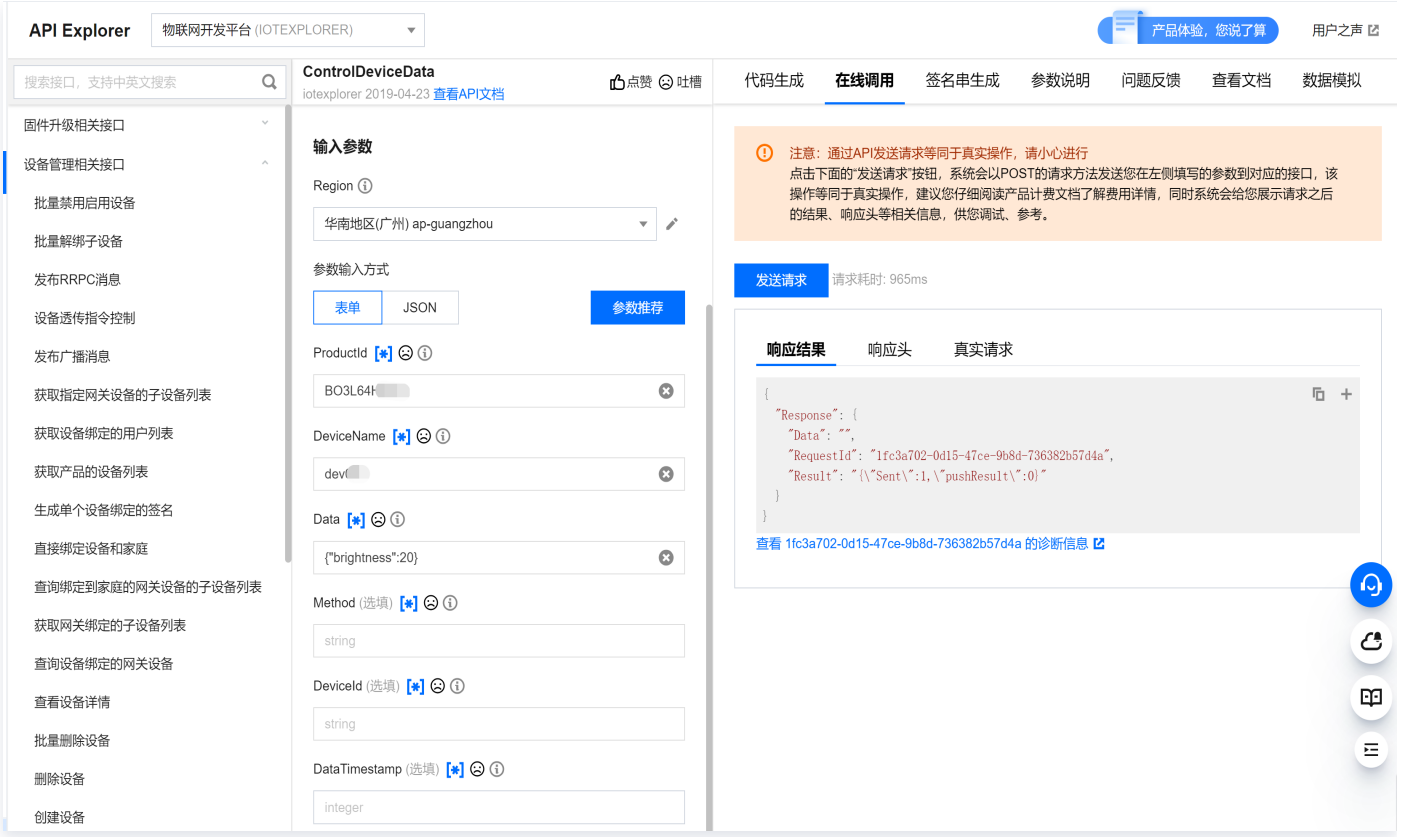
- 注册腾讯云账号，申请 物联网开发平台服务 。
- 参见 MQTT.fx 快速接入物联网开发平台，能够成功模拟设备连接平台，并能订阅物模型消息用于接收远控 API 下发的消息。

使用 API Explorer 控制设备

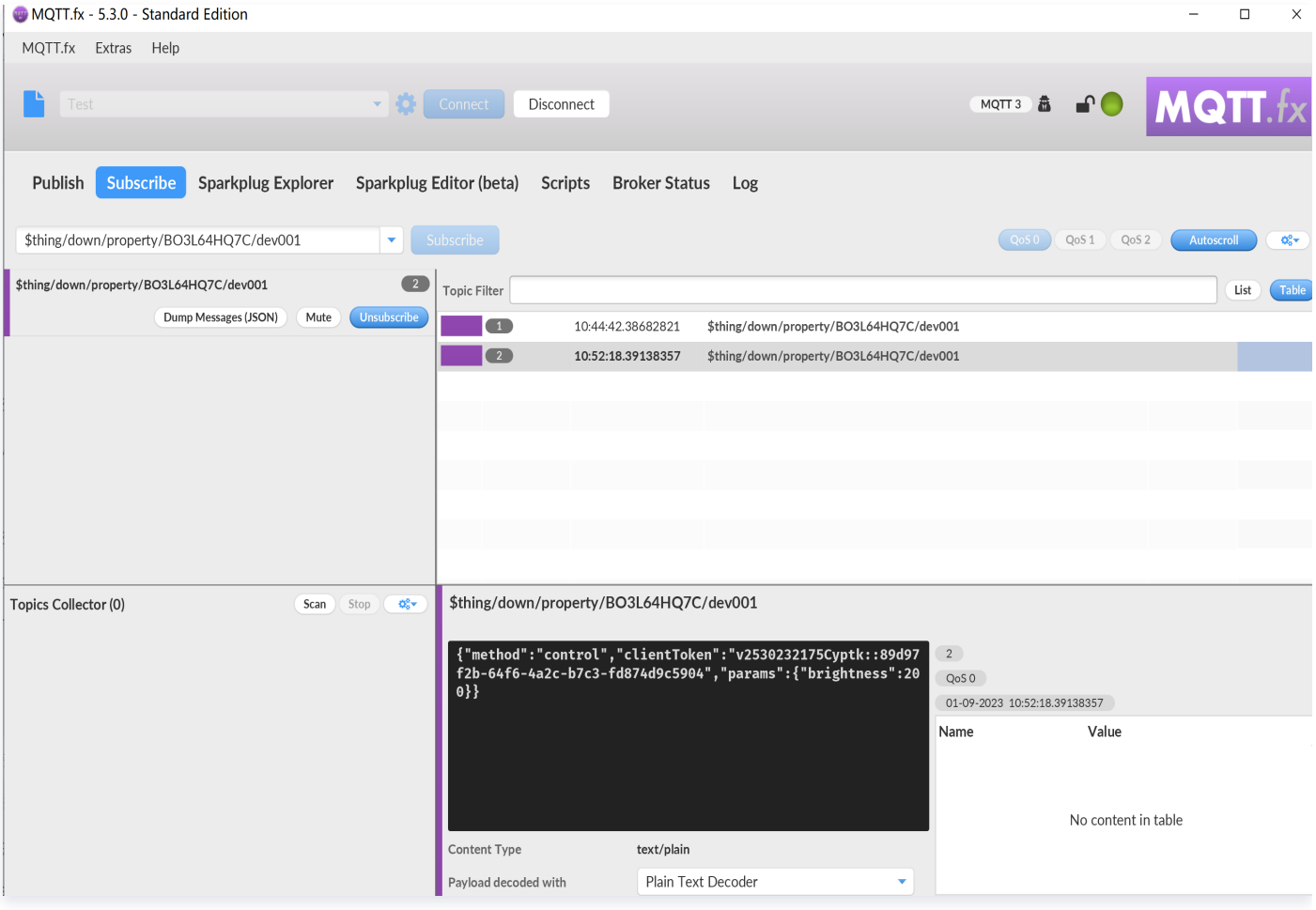
1. 登录 物联网开发平台 后，访问 设备远程控制 API ，可查看平台提供的远控 API 服务，单击**点击调试**，可进入 API Explorer 在线 API 调试工具。
2. 进入设备远程控制 API 调试页面，输入必选参数。然后单击右侧区域的“**发送请求**”按钮。API Explorer 将返回响应结果。

> ⚠ **注意：**
> 在发送请求前，被控设备已成功连接平台，并且成功订阅了物模型下行 Topic。

   - 输入参数 Region：选择**华南地区（广州）ap-guangzhou**。
   - 输入参数 ProductId：请输入 MQTT.fx 工具或其他模拟工具或真实物理设备连接到平台的产品 ID。
   - 输入参数 DeviceName：请输入连接到平台的 DeviceName。
   - 输入参数 Data：请输入物模型 JSON，例如{"brightness":20} 。



3. 若输入参数输入正确，而且远控的设备已连接物联网开发平台，并成功订阅物模型属性下行 Topic，MQTT.fx 将会收到平台下发的物模型消息。



平台转发消息至用户 HTTP 服务

最近更新时间：2024-11-05 15:02:52

操作场景

物联网解决方案通常需要实时获取设备上报到物联网平台的数据、状态等信息，再结合各自场景的业务数据来完成整套物联网解决方案的闭环业务流程。本文档主要介绍如何使用平台的规则引擎消息转发至用户自建 HTTP 服务的能力，将设备数据、状态转发到用户自建的 HTTP 服务，用户可根据此文档中的代码示例快速了解并构建自己的 HTTP 服务。

准备工作

为了快速体验该业务功能，需要提前做好以下准备工作：

- 注册腾讯云账号，申请 物联网开发平台服务 。
- 参见 MQTT.fx 快速接入物联网开发平台，能够成功模拟设备连接平台，并能发布物模型消息至平台。
- 用户需提前注册开通 腾讯云 SCF 产品，通过 SCF 产品托管 HTTP 服务，将对应的 HTTP 服务接收地址准备好配置到规则引擎转发服务中。

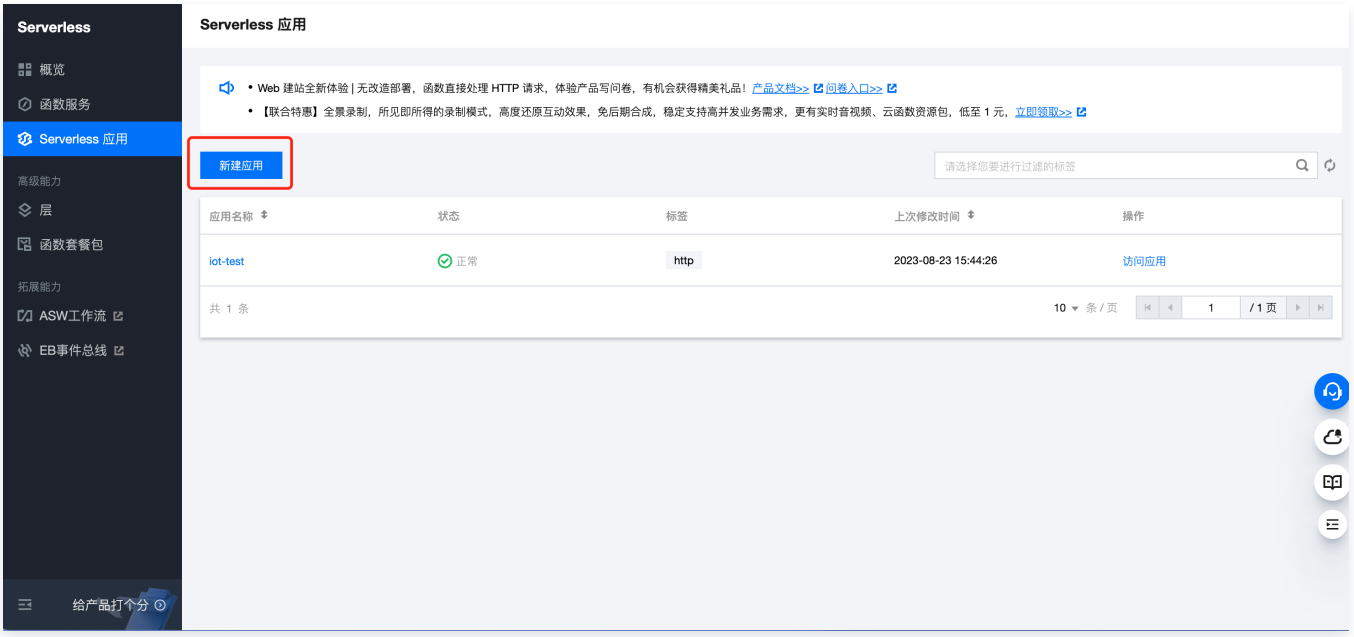
操作步骤

使用 SCF 部署 HTTP 服务

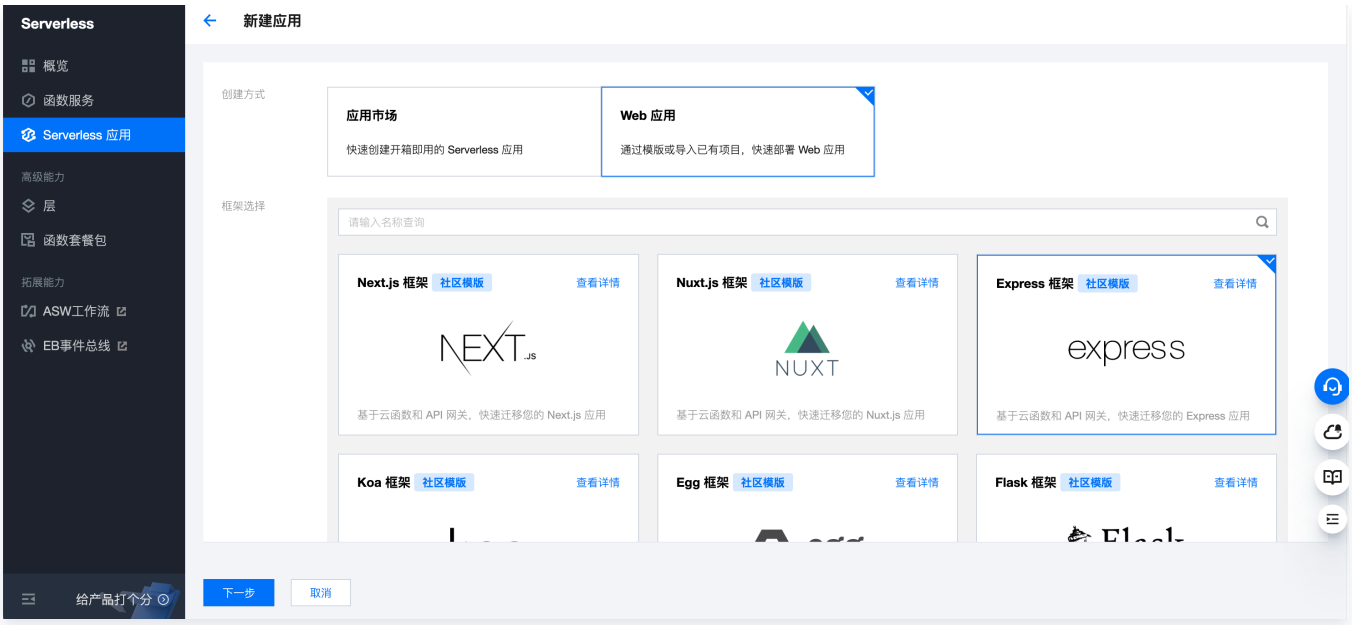
本文档使用腾讯云云函数（Serverless Cloud Function，SCF）产品快速搭建 Web 服务，来接收物联网平台规则引擎处理后的数据。本示例只展示如何接收物联网平台的数据，进一步使用数据需要用户根据实际业务场景去处理。

在 SCF 平台基于 Express 快速搭建 Node 服务示例

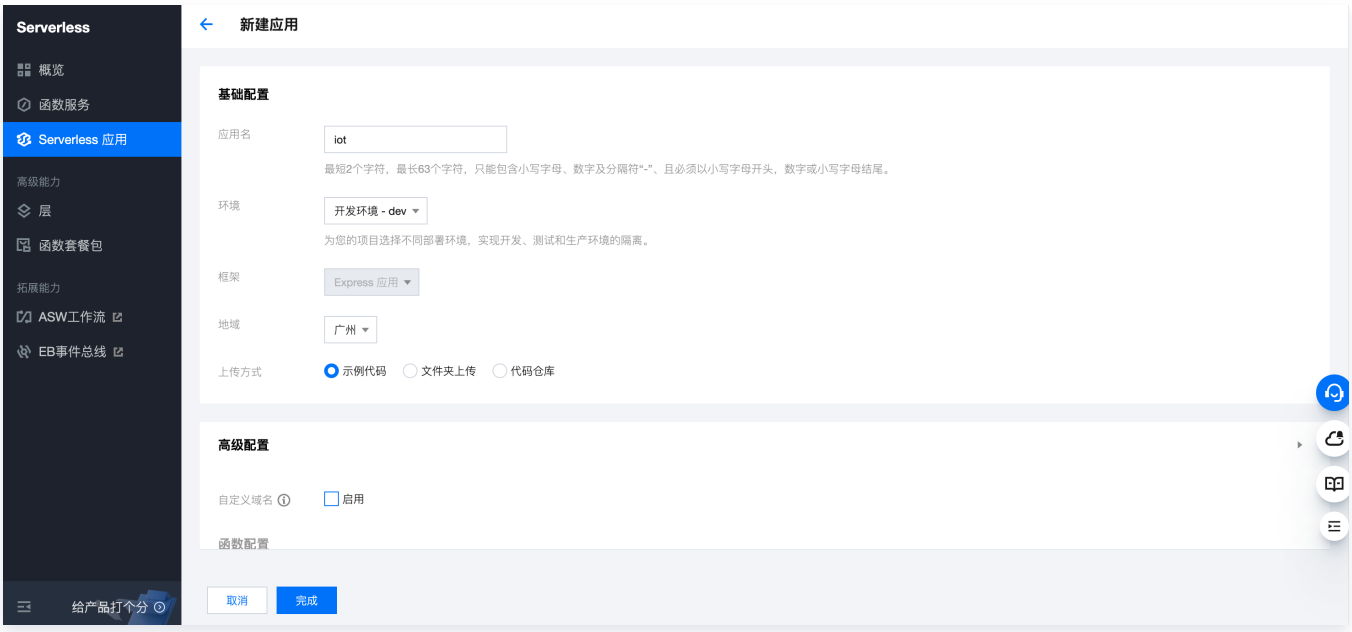
1. 登录 SCF 控制台，选择 **Serveless 应用**，单击**新建应用**。



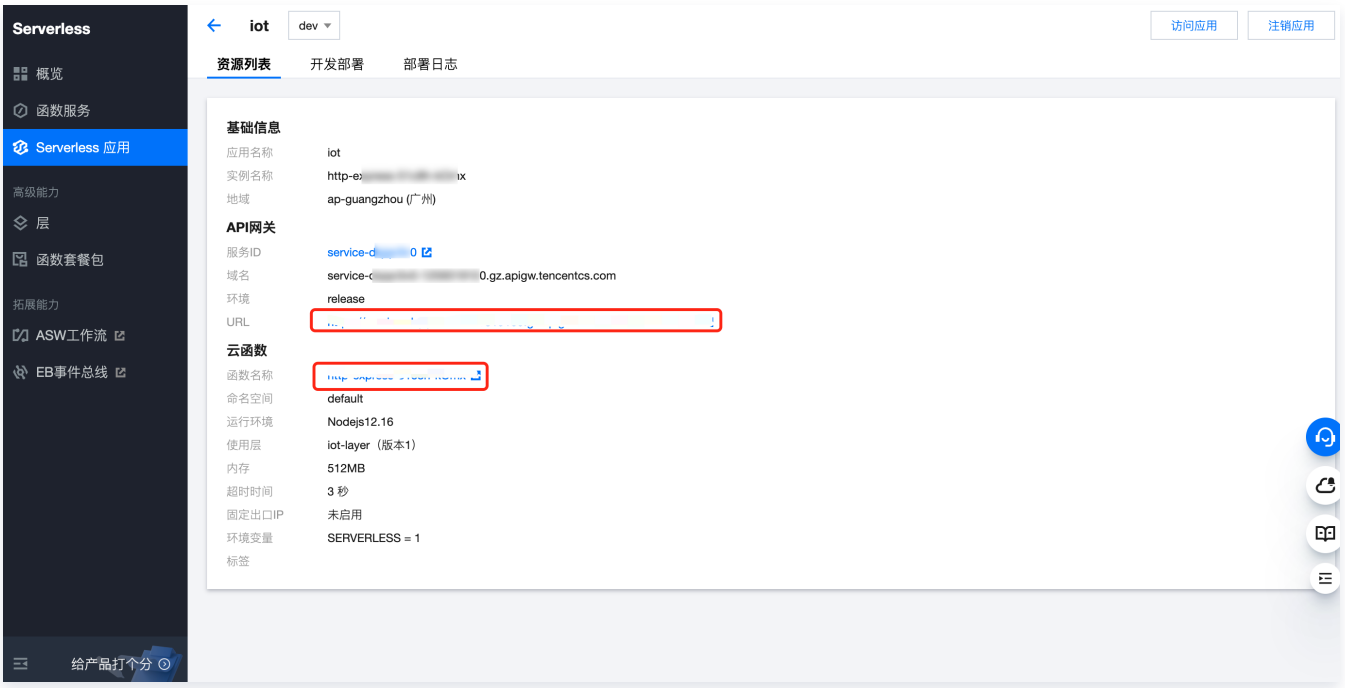
2. 选择 Web 应用下的 Express 框架模板，单击**下一步**。



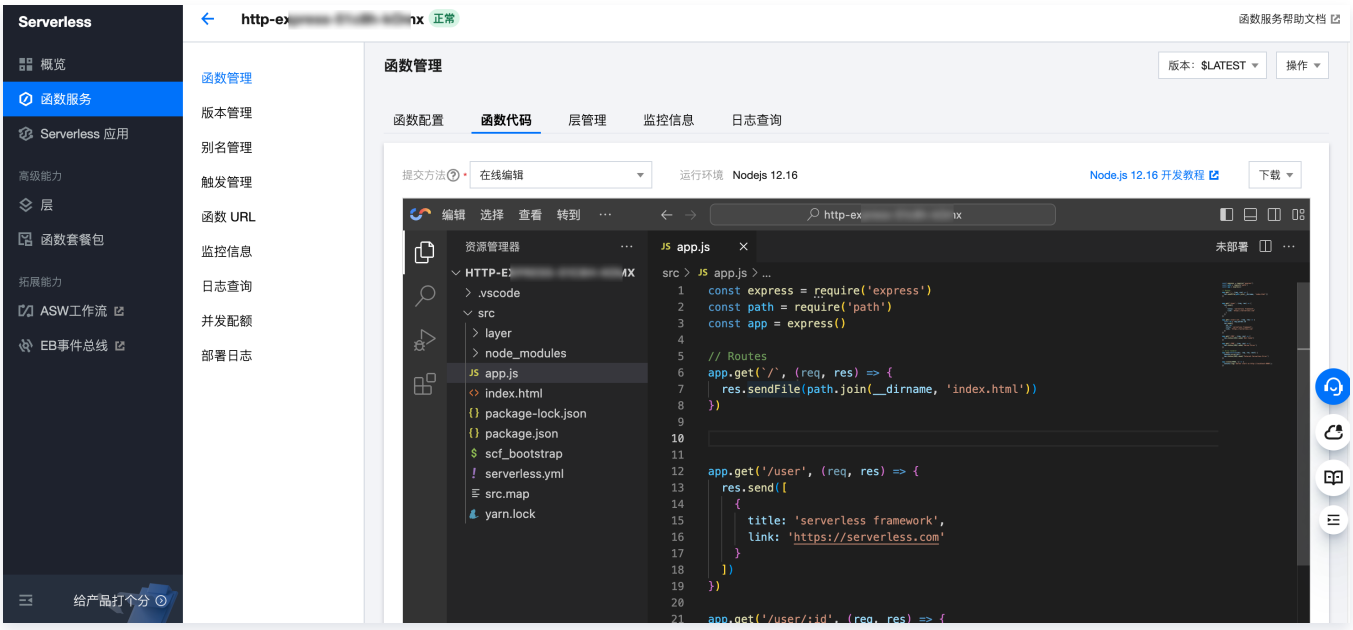
3. 按下图所示输入应用名，选择环境、地域，上传方式。配置结束后单击**完成**。



4. 等待部署完成后进入应用，在 API 网关中可以看到 URL。此 URL 就是您要在规则引擎中填写的 API 地址，假设此 URL 地址为：`https://iot-api/`，那么在本示例中您的 API 地址为 URL 加上接口路由，即：`https://iot-api/test`。单击函数名称，进入函数服务。



5. 在函数服务中，可以看到如下图所示的对应的代码模板。单击 **app.js**，复制示例代码到 app.js 文件中，注意无需全覆盖。

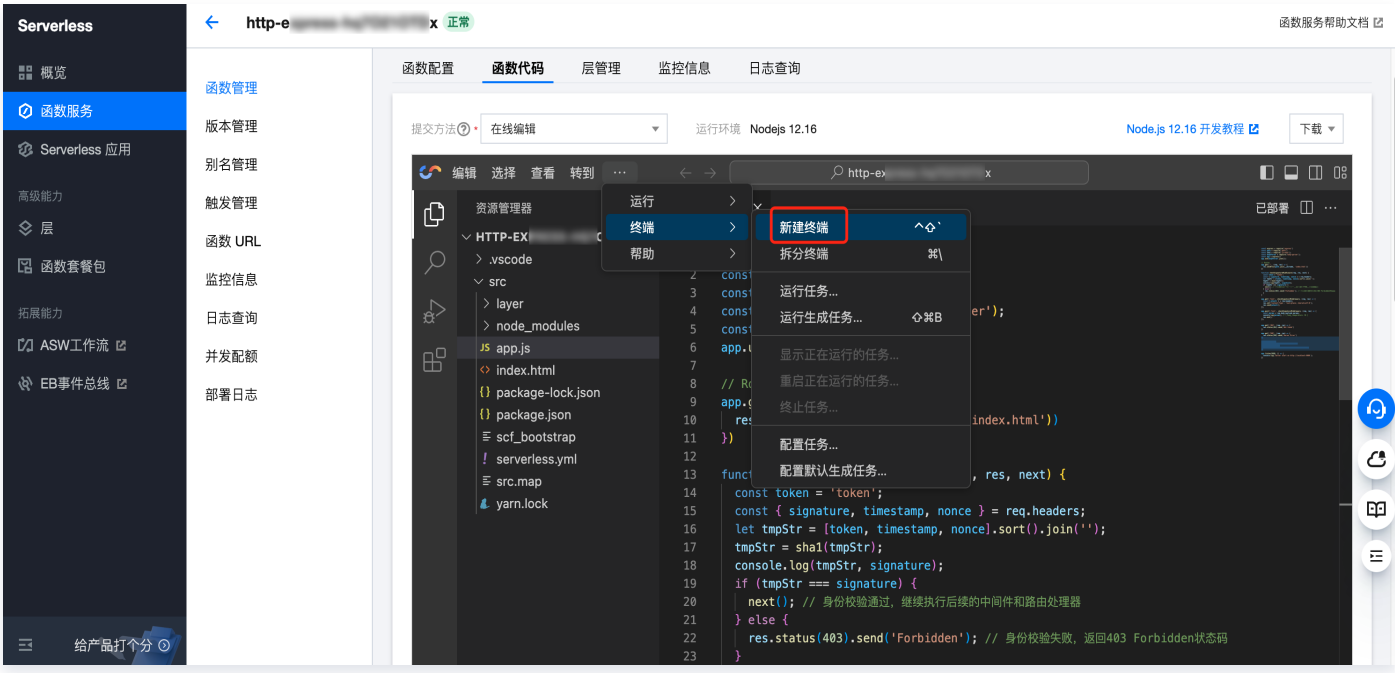


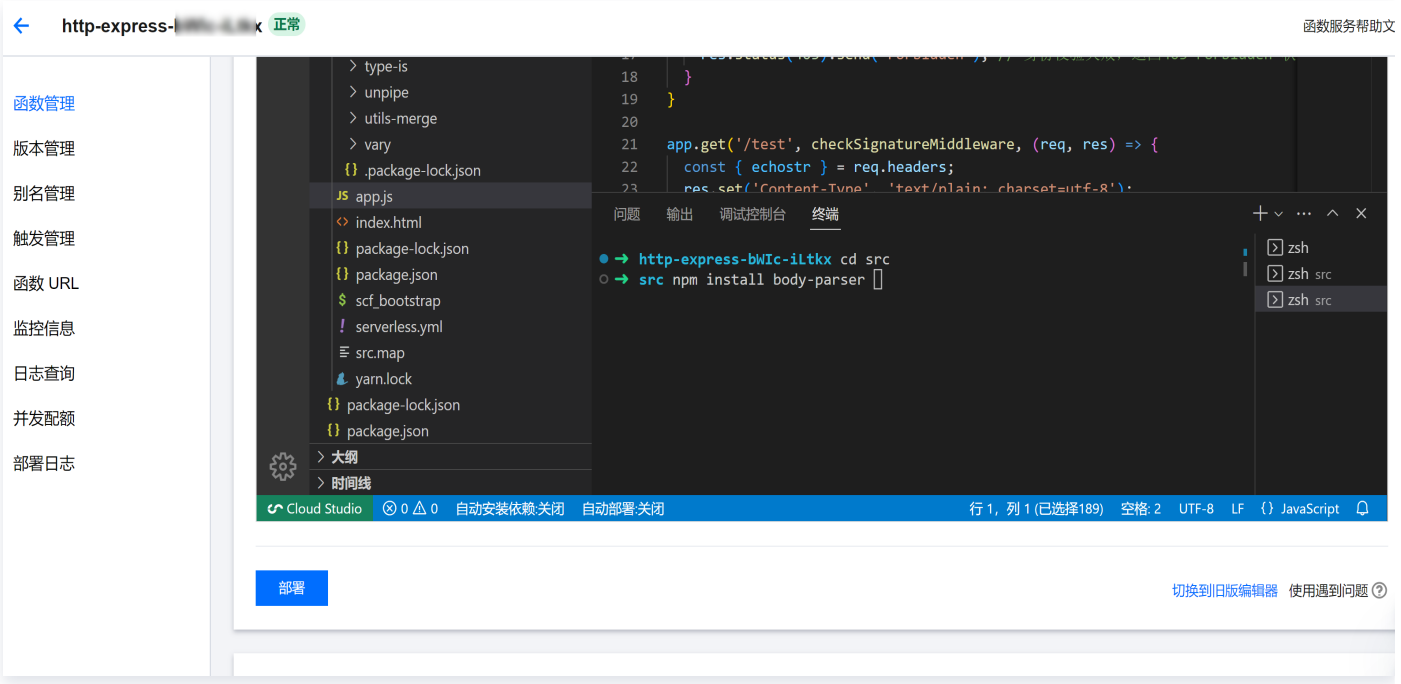
需要将下述代码复制到 app.js 的代码，请覆盖原 app.js 中的 const 开头的3行代码，原 app.js 中的 app.get 方法无需覆盖。

```
const express = require('express')
const path = require('path')
const bodyParser = require('body-parser');
const sha1 = require('js-sha1');
const app = express()
app.use(bodyParser.json());

// 鉴权信息校验的代码在上文已经解释过，这里使用Middleware来简化代码
function checkSignatureMiddleware(req, res, next) {
```

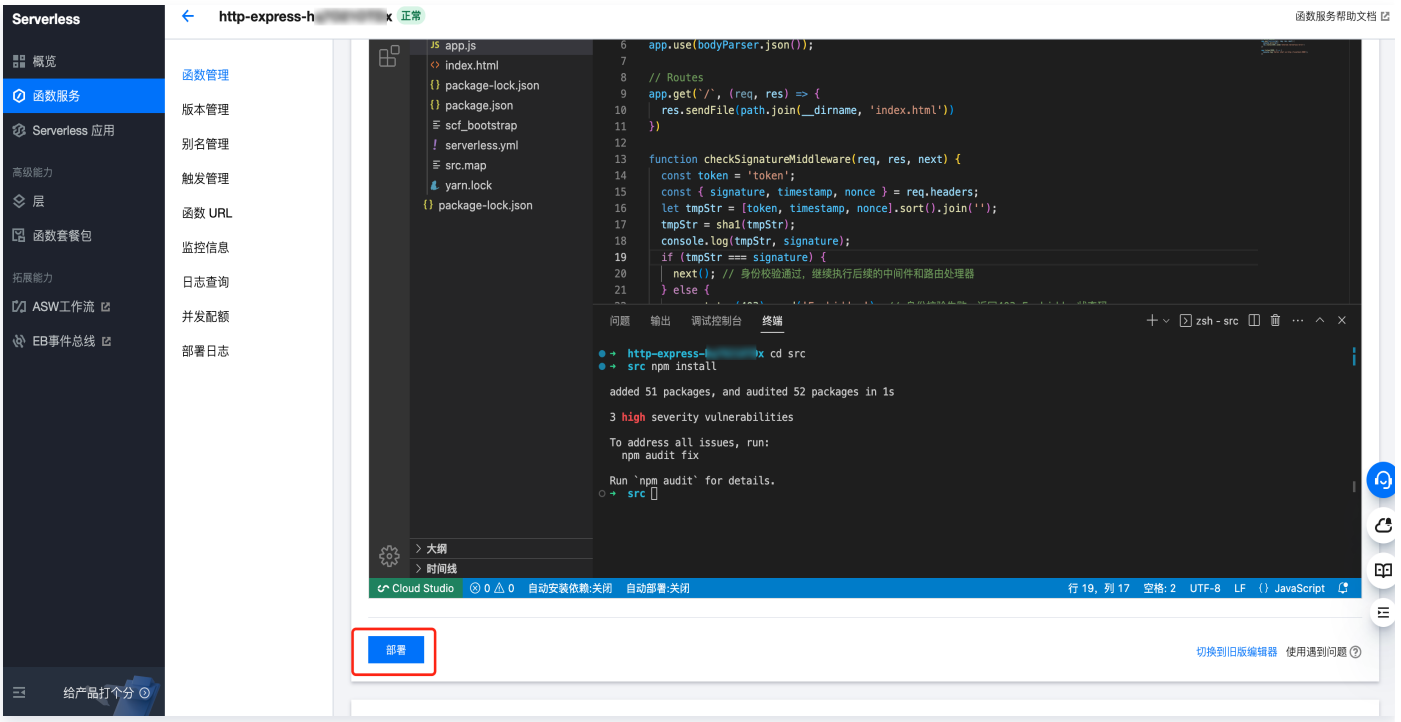
```
  const token = 'test';
  const { signature, timestamp, nonce } = req.headers;
  let tmpStr = [token, timestamp, nonce].sort().join('');
  tmpStr = sha1(tmpStr);
  if (tmpStr === signature) {
    next(); // 身份校验通过，继续执行后续的中间件和路由处理器
  } else {
    res.status(403).send('Forbidden'); // 身份校验失败，返回403 Forbidden 状态码
  }
}

app.get('/test', checkSignatureMiddleware, (req, res) => {
  const { echostr } = req.headers;
  res.set('Content-Type', 'text/plain; charset=utf-8');
  res.send(echostr);
});

// 注意接收规则引擎转发的数据接口为 POST 类型
app.post('/test', checkSignatureMiddleware, (req, res) => {
  const params = req.body.payload.params;
  console.log(params); // { body_temperature: 36 }
  res.end();
});
```

6. 上述步骤完成后，需要引入两个依赖包，分别是 body-parser 与 js-sha1。在代码编辑器的上方选择**终端 > 新建终端**，然后在底部终端中输入命令：`cd src` 进入 src 目录，接着分别输入命令 `npm install body-parser` 和 `npm install js-sha1` 安装两个依赖包。





7. 等待依赖安装完成之后，单击下图红色线框中的**部署**，即可成功部署 Web 服务，SCF 会自动启动该 Web 服务，约5秒 – 10秒后系统会提示“部署成功”。



控制台操作

创建规则引擎

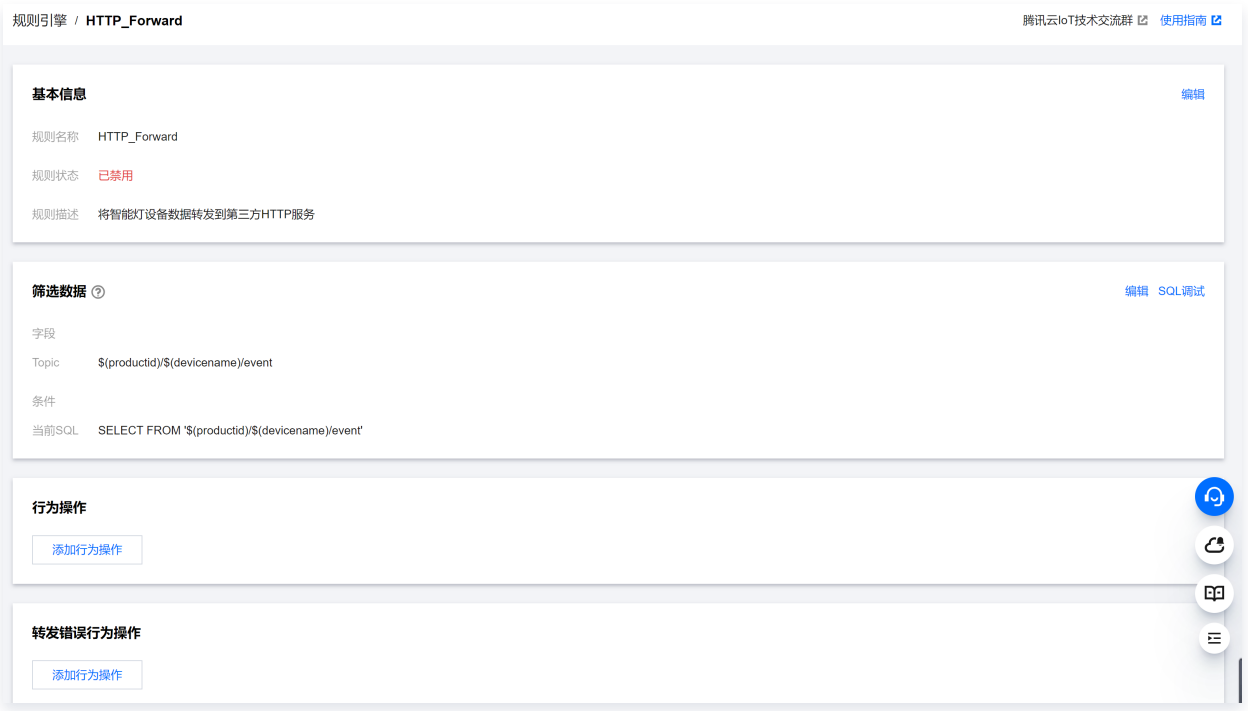
1. 登录 物联网开发平台控制台，选择平台默认开通的**公共实例**或用户购买的**企业实例**。
2. 进入某个项目，单击左侧菜单**基础服务 > 规则引擎**，再次单击规则引擎列表页的**新建规则**。
   - 规则名称：输入“HTTP_Forward”或其他内容。
   - 规则描述：输入转发某类产品下设备数据的备注信息。

3. 规则基本信息填写完成后，单击**确定**，即可完成规则的创建。

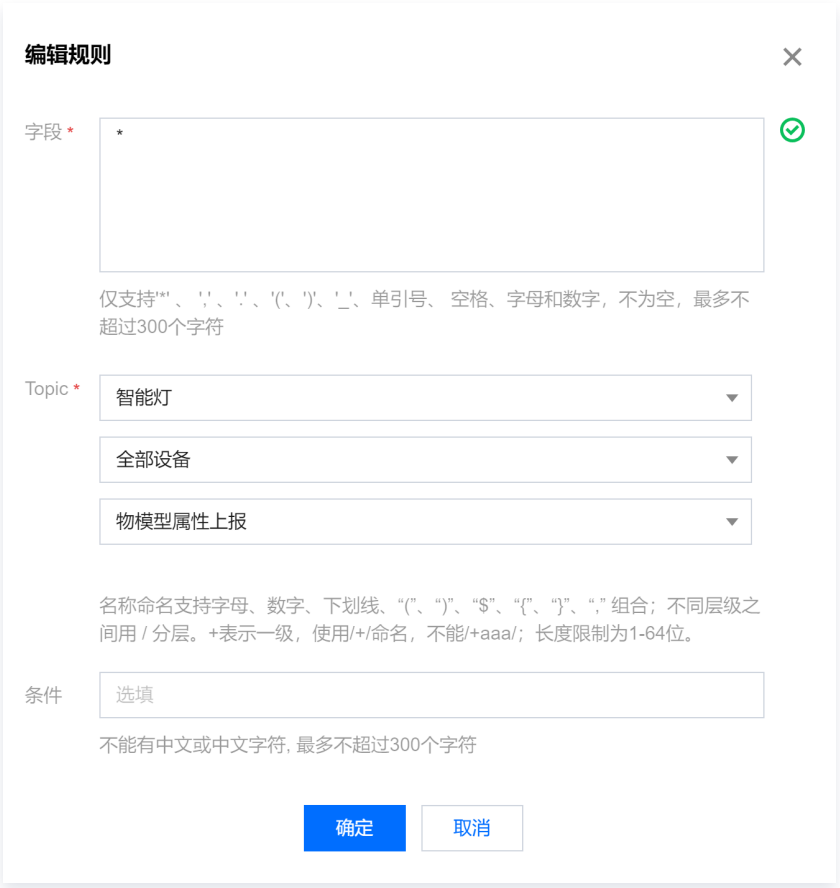
配置规则引擎

筛选数据

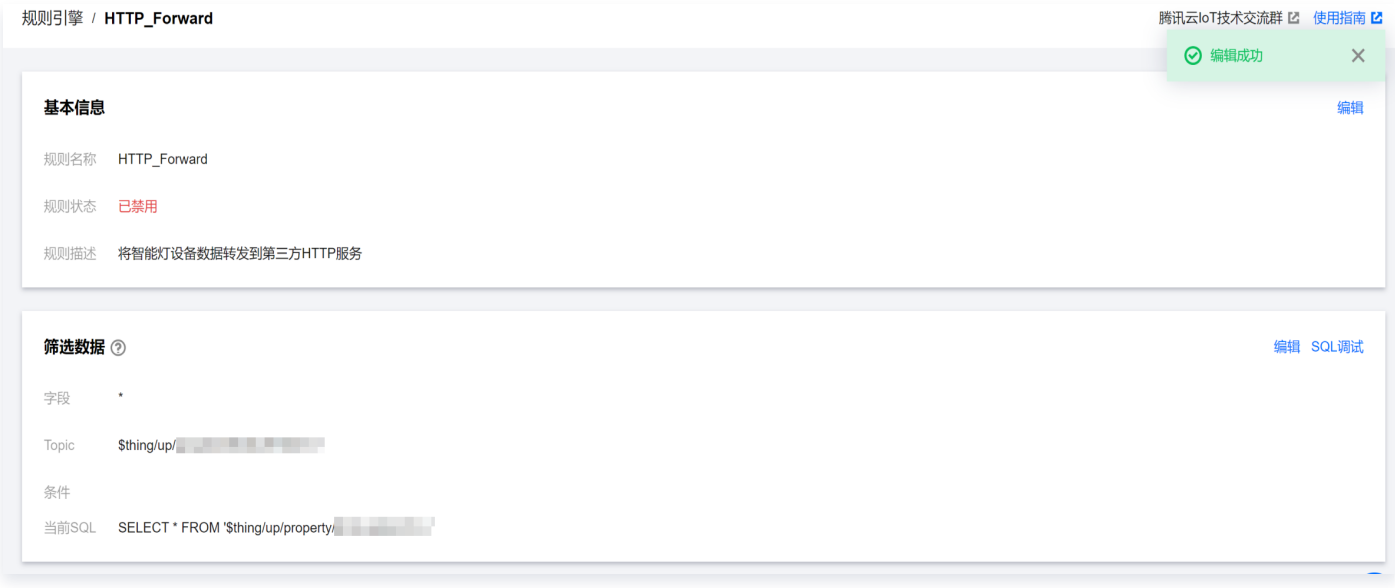
1. 在规则引擎列表选择刚创建成功的规则 **HTTP_Forward**，单击规则名称或操作列的**管理**，进入规则引擎配置页面。
2. 单击下图**筛选数据**右侧的**编辑**，选择要转发的设备数据源。



3. 在编辑规则页面，输入或选择如下信息。
   - 字段：输入"*"，表示将设备上报的所有 JSON 数据进行转发。
   - Topic：首先选择对应的产品。建议选择“MQTT.fx 快速接入平台”中所创建的产品**智能灯**；第二步选择设备，请选择**全部设备**，表示规格引擎将转发该产品下所有的设备；最后一步选择转发的数据，请选择**物模型属性上报**。



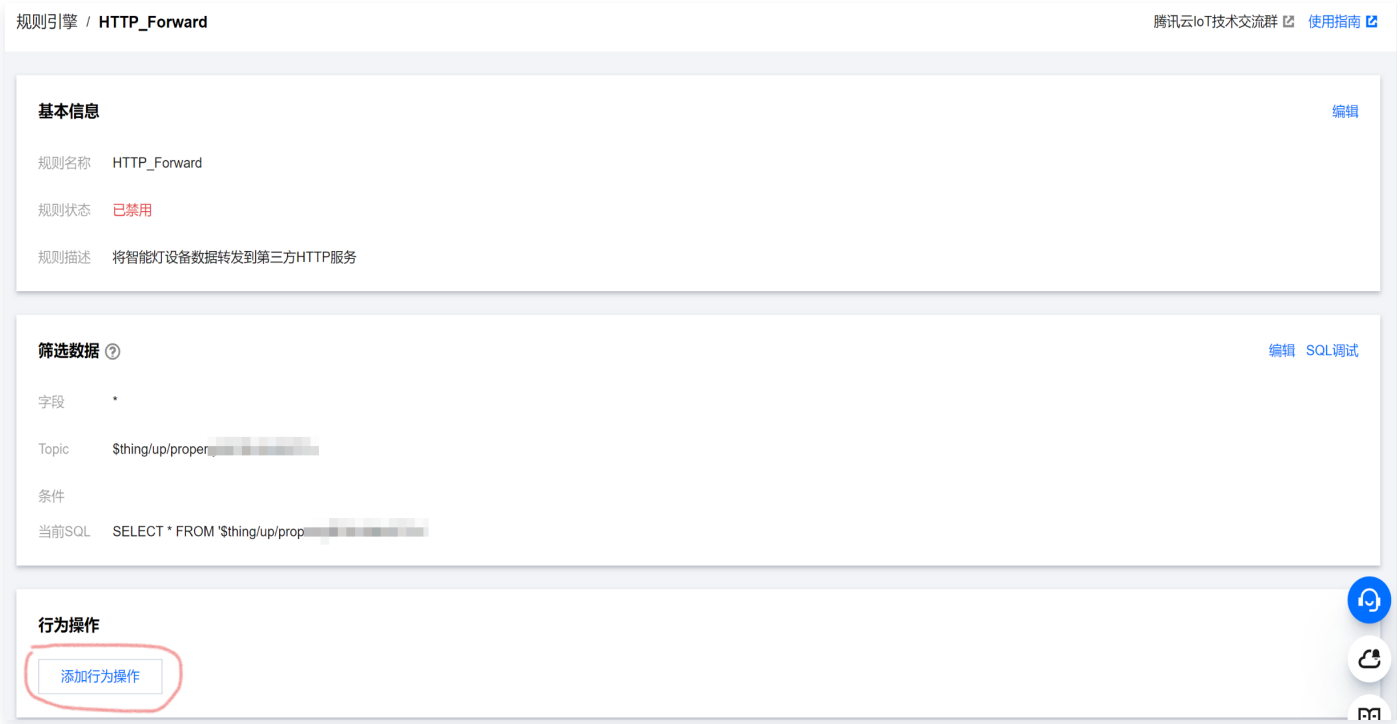
4. 单击**确定**按钮，将返回配置规则页面，系统自动生成如下图所示对应的 TOPIC 及筛选数据 SQL。



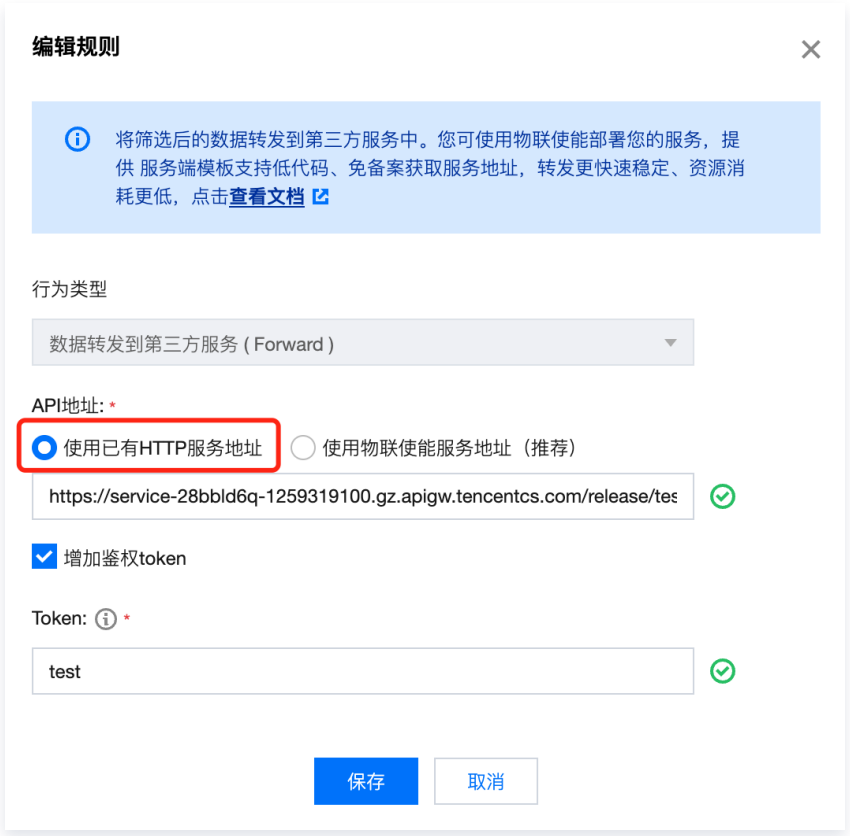
配置规则引擎

行为操作配置

1. 在规则引擎列表选择刚创建成功的规则 HTTP_Forward，单击规则名称或操作列的**管理**链接，进入规则引擎配置页面。



2. 下图行为类型选择“**数据转发到第三方服务（Forward）**”、选择 API 地址类型选择“**使用已有 HTTP 服务地址**”。HTTP 地址则为 **此步骤** 中复制 SCF 自动生成的 URL 地址，并在最后加上 `/test` 。类似 `https://service.*****.apigw.tencenttcs.com/release/test` 。



3. 请勾选 “增加鉴权 Token”，且输入 test ，该 Token 必须与 SCF 函数服务中 app.js 里定义的 Token 值一致。

4. 单击**保存**，完成规则行为配置。
5. 返回规则引擎列表，将该规则的状态设置为**启用**。

模拟验证

1. 参见 MQTT.fx 快速接入物联网开发平台，模拟设备上报物模型消息。注意上报的数据相关产品需要与规则引擎删选的目标产品、设备保持一致。若不一致 SCF 日志查询将无法打印输出数据。
2. 进入 SCF 的函数服务，单击**日志查询**，可查看到 SCF 打印的日志，可完整打印出设备上报的物模型属性。