物联网开发平台

通用功能（设备端）

腾讯云

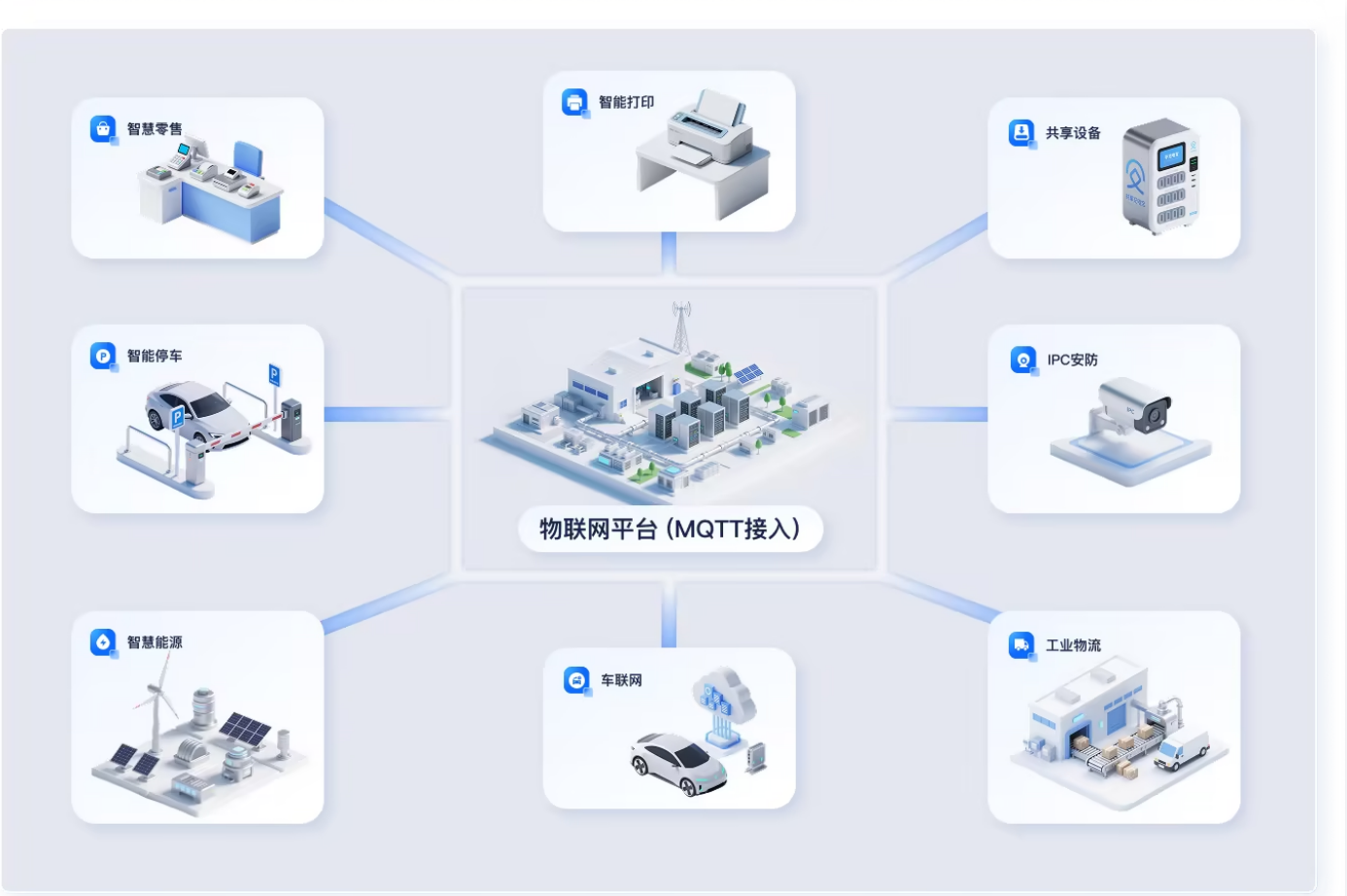
文档目录

通用功能（设备端）
功能介绍

最近更新时间：2026-07-30 12:00:36

功能概述

设备接入（通用功能）指平台提供的基于 MQTT 协议的设备接入、消息通信等能力，助力各场景下物联网解决方案的设备快速接入平台，简化用户设备管理成本，加快物联网解决方案的落地。



计费说明

设备接入分为两种计费模式，分别对应物联网行业两种典型场景，具体可参见 实例概览 。其中企业实例对应的计费方式请参考 企业实例计费说明 ，消费实例对应的计费方式参考 消费实例计费说明 。

应用场景

企业实例应用场景

应用场景	说明

车联网	支持各类车辆的海量连接与实时消息通信业务场景。例如，自动驾驶、物流货运车辆管理、前装 TBox 或车载终端、后装 OBD 车联等场景。平台提供海量连接及高并发消息通信能力以满足车辆在各场景下的瞬时高并发消息吞吐需求。
商业服务	支持充电桩、充电宝、自助零售、停车、快递柜服务等无人值守商业场景。提供高可靠的设备连接与低延迟的消息通信服务，用户只需聚焦自研业务平台的开发运维及业务运营。腾讯云物联网开发平台可以降低用户的开发及运维投入成本，并可充分保障商业服务所依赖的消息可靠通信。
智慧能源	支持各类 DTU、边缘网关采集水、电、燃气、光、储、充等各类能源类设备数据与高可靠传输，助力企业实现各场景能源管理解决方案。
资产管理	支持各行业企业设备资产的数据采集、远程运维、健康度监测、位置追踪、OTA 等服务，降低企业设备管理成本，提高设备资产运维效率。
金融支付	支持金融支付场景下的收款语音播报、订单小票打印等设备的海量设备接入与消息通信服务，保障金融支付场景下的高可靠消息通信需求。

消费实例应用场景

应用场景	说明
大小家电	支持各类大小家电的设备接入、管理。例如空调、热水器、洗衣机、温控、热泵等各类形态的大小家电设备智能化场景。
全屋智能	支持全屋各类传感器、中控网关、电工照明。
家庭安防	支持各类家庭安防监控 IPC 设备基于 MQTT 协议的设备接入、消息通信、OTA 等能力。

开通服务

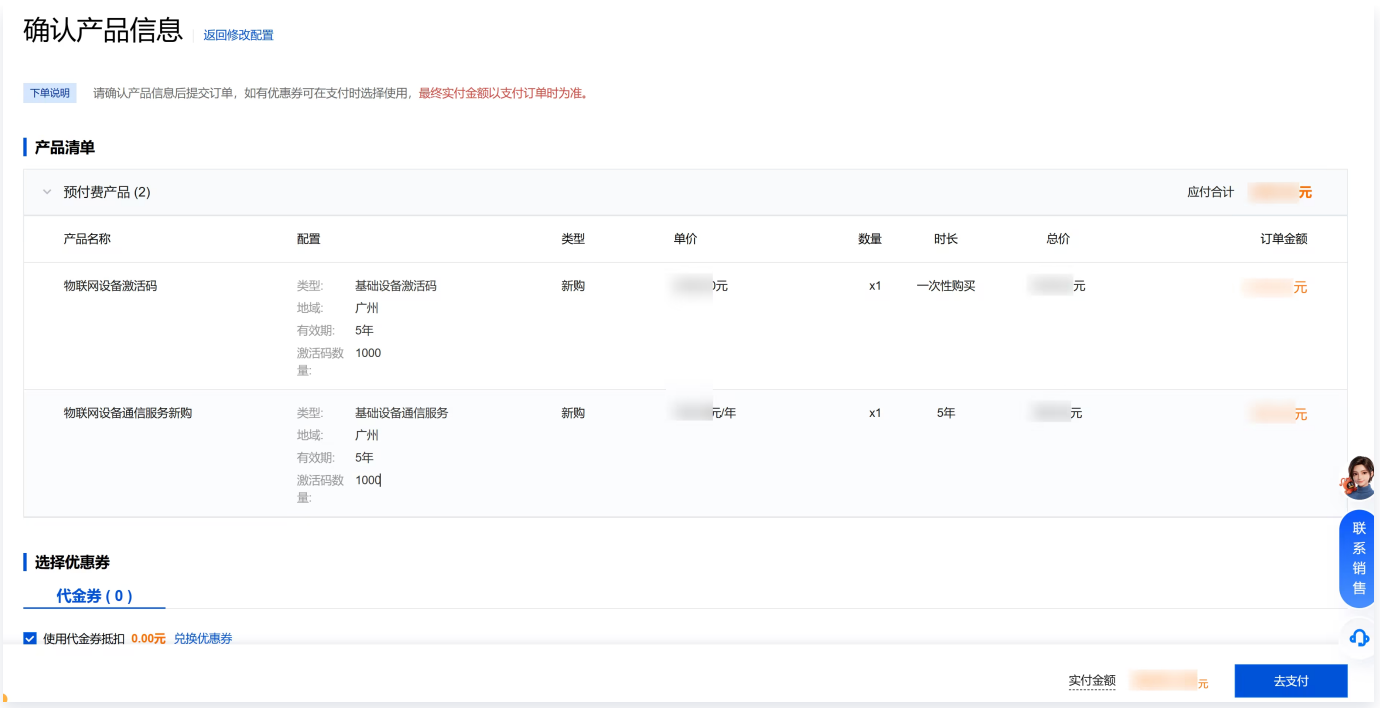
最近更新时间：2026-07-30 12:00:36

开通消费实例正式版

购买消费实例

1. 在购买基础设备接入服务之前，您需要先 注册腾讯云 账号，并完成 实名认证 。
2. 进入 物联网开发平台购买页 ，实例类型选择**消费实例**，服务类型选择“基础设备接入服务”。
3. 生效地域：国内设备选择广州区域，出海设备选择新加坡；有效期默认规格5年。
4. 激活码数量：输入您要量产的设备数量，一个物理设备默认烧录一个激活码。
5. 勾选我已阅读并同意《腾讯云服务协议》，单击**立即购买**。

6. 单击立即购买后进入订单确认，基础设备接入服务分别对应“基础设备激活码”和“基础设备通信服务”。



7. 确认后单击**去支付**进行付款，支付成功后系统将您支付订单的激活码数量累加到对应激活码规格的总可用数量。

查看激活码用量

支付完基础设备接入服务后，用户可进入 物联网开发平台控制台 的**消费实例**，选择左侧导航栏的**用量统计 > 激活码概览**可查看支付成功的订单对应的激活码用量。

创建产品

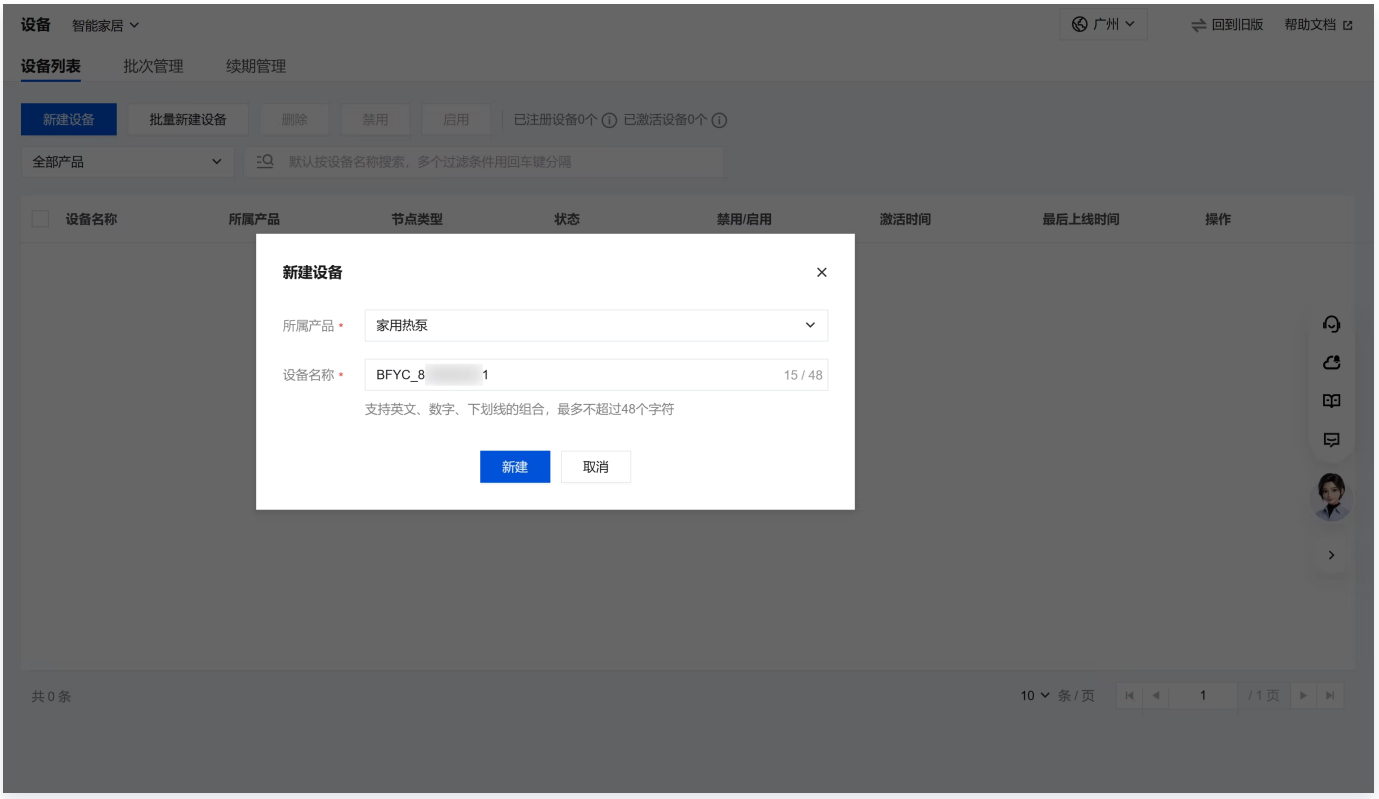
1. 登录 物联网开发平台控制台 ，进入**实例管理**页面，单击**消费实例**的卡片进入详情页。
2. 选择消费实例左侧导航栏的**产品**，然后单击**创建产品**。

3. 在创建产品页面中，输入“产品名称”，设备类型必须是“**基础设备**”，信息配置好后，单击**创建**。

创建设备

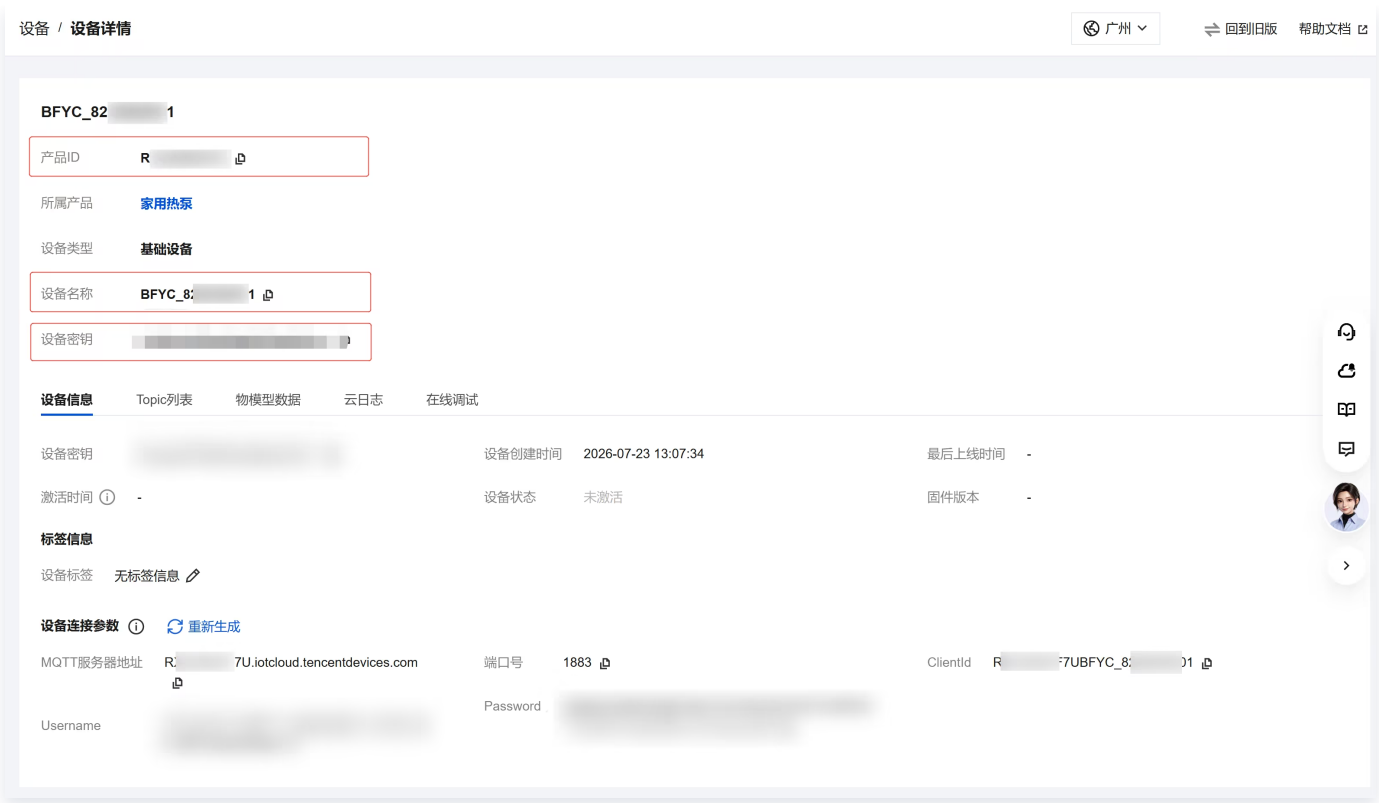
1. 选择消费实例左侧导航栏的**设备**，然后单击**新建设备**。

2. 所属产品选择前述步骤已创建成功的产品，输入设备名称并单击**新建**。



3. 单击创建成功的设备，进入设备详情页获取设备三元组信息。

> **说明：**
> 产品 ID、设备名称、设备密钥三元组信息作为快速接入的设备身份凭证。


设备 / 设备详情
广州
回到旧版
帮助文档
BFYC_82 1
产品ID
所属产品
家用热泵
设备类型
基础设备
设备名称
设备密钥
设备信息
Topic列表
物模型数据
云日志
在线调试
设备密钥
设备创建时间
2026-07-23 13:07:34
最后上线时间
激活时间
设备状态
未激活
固件版本
标签信息
设备标签
无标签信息
设备连接参数
重新生成
MQTT服务器地址
7U.iotcloud.tencentdevices.com
端口号
1883
ClientId
Username
Password

快速接入

最近更新时间：2026-07-30 12:00:37

本文将介绍如何快速接入腾讯云物联网设备端 SDK，跑通**登录**功能。

前提条件

开通服务

在使用 IoT 设备端 SDK 前，您需要前往控制台，创建产品并在产品下创建设备。具体步骤参见 开通服务 。
开通服务后，请记录以下设备三元组信息，在后续 登录 步骤中会用到：

- **ProductId**（产品 ID）
- **DeviceName**（设备名称）
- **DeviceSecret**（设备密钥）

> ⚠ **注意：**
> 设备三元组是设备登录平台的唯一身份凭证，请妥善保管 DeviceSecret，避免硬编码提交到公共代码仓库。

环境准备

请根据您实际对接的设备芯片平台，准备对应的编译环境。本文以 Linux 本机环境为例演示接入流程，帮助您快速跑通登录功能；正式对接您的设备时，替换为目标芯片平台的交叉编译工具链即可，示例代码无需改动。

Linux 本机（快速体验）

环境项	要求
操作系统	x86_64 Linux（Ubuntu 18.04+ / CentOS 7+ 等主流发行版）。
编译工具	CMake 3.10 及以上、GCC / Clang（支持 C99）。
系统依赖库	`pthread`、`m`（数学库）、`dl`、`z`（zlib）。

嵌入式设备（交叉编译）

环境项	要求

交叉编译工具链	与目标芯片架构匹配的 GCC / Clang 交叉工具链（如 `arm-linux-gnueabihf-gcc`、`aarch64-none-linux-gnu-gcc` 等），具体工具链由芯片厂商 SDK 或 FAE 提供。
编译工具	CMake 3.21 及以上（交叉编译需要通过 `-DCMAKE_TOOLCHAIN_FILE` 指定工具链配置文件）。
sysroot	部分工具链已自带默认 sysroot，无需单独指定；如需自定义，请参考芯片厂商 SDK 文档。
系统依赖库	与目标架构匹配的 `pthread`、`m`、`dl`、`z` 等库版本，通常随交叉工具链一并提供。

说明：
不同芯片平台的交叉编译工具链配置差异较大，具体的工具链路径、`-DCMAKE_TOOLCHAIN_FILE` 配置方式请参考对应芯片平台 SDK 包内的 README 说明。

获取 SDK

前往 SDK & Demo 页面下载 SDK 包并解压。解压后得到以下内容：

```
tc_iot_sdk/
├── include/              # 全量 public API 头文件（tc_iot.h 等）
└── lib/
    └── libtc_iot_sdk.a     # SDK 静态库
```

创建项目

1. 新建项目目录，并将上一步下载解压得到的 `tc_iot_sdk/` 目录拷贝到项目根目录下。

```
mkdir -p my_iot_project && cd my_iot_project
cp -r /path/to/tc_iot_sdk .
```

2. 项目目录结构如下：

```
my_iot_project/
├── CMakeLists.txt
├── main.c
└── tc_iot_sdk/
```

```
    ├── include/
    └── lib/libtc_iot_sdk.a
```

集成并引入 SDK

步骤 1：配置 CMakeLists.txt

在项目根目录创建 `CMakeLists.txt`，添加头文件搜索路径与静态库链接：

```
cmake_minimum_required(VERSION 3.10)
project(iot_quickstart_demo C)

set(CMAKE_C_STANDARD 99)

set(SDK_DIR ${CMAKE_CURRENT_SOURCE_DIR}/tc_iot_sdk)

add_executable(iot_quickstart_demo main.c)

target_include_directories(iot_quickstart_demo PRIVATE
    ${SDK_DIR}/include)

target_link_libraries(iot_quickstart_demo PRIVATE
    -Wl,--start-group ${SDK_DIR}/lib/libtc_iot_sdk.a -Wl,--end-group
    pthread m dl z)
```

步骤 2：引入头文件

在 `main.c` 中引入设备端登录相关头文件：

```
#include "tc_iot.h"      // 初始化 / 登录 / 登出
```

接入步骤

本节介绍如何调用 SDK 完成登录。

步骤 1：初始化 SDK

调用 `tc_iot_init` 完成 SDK 全局初始化：

```
#include <string.h>
#include "tc_iot.h"
```

```
tc_iot_config_s config;
memset(&config, 0, sizeof(config));
config.storage_path = "./";
config.log_level = TC_IOT_LOG_LEVEL_INFO;
config.on_log = NULL;
config.on_device_event = NULL;

tc_iot_error_e rc = tc_iot_init(&config);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_init failed: %d\n", rc);
    return -1;
}
```

步骤 2：登录

调用 `tc_iot_login` 使用设备三元组完成登录鉴权，登录结果通过回调异步返回：

参数	类型	说明
`product_id`	`const char *`	产品 ID。
`device_id`	`const char *`	设备名称。
`device_secret`	`const char *`	设备密钥。
`region`	`const char *`	接入地域，如 `ap-guangzhou`。

```
static void on_login(tc_iot_error_e error_code, const char
*error_message) {
    if (error_code == TC_IOT_ERR_SUCCESS) {
        printf("[demo] login success\n");
    } else {
        printf("[demo] login failed: %d, msg=%s\n", error_code,
               error_message ? error_message : "");
    }
}

tc_iot_device_info_s device_info;
device_info.product_id = "YOUR_PRODUCT_ID";
device_info.device_id = "YOUR_DEVICE_NAME";
```

```
device_info.device_secret = "YOUR_DEVICE_SECRET";
device_info.region = "ap-guangzhou";

rc = tc_iot_login(&device_info, on_login);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_login call failed: %d\n", rc);
}
```

完整示例代码

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "tc_iot.h"

static volatile int s_login_done = 0;
static tc_iot_error_e s_login_rc = TC_IOT_ERR_SUCCESS;

static void on_login(tc_iot_error_e error_code, const char
*error_message) {
    s_login_rc = error_code;
    s_login_done = 1;
    if (error_code != TC_IOT_ERR_SUCCESS) {
        printf("[demo] login failed: %d, msg=%s\n", error_code,
               error_message ? error_message : "");
    } else {
        printf("[demo] login success\n");
    }
}

int main(int argc, char **argv) {
    if (argc < 7) {
        printf("Usage: %s -p <product_id> -d <device_id> -s
<device_secret>\n", argv[0]);
        return 0;
    }

    const char *product_id = NULL;
    const char *device_id = NULL;
```

```
    const char *device_secret = NULL;
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-p") == 0 && i + 1 < argc) {
            product_id = argv[++i];
        } else if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
            device_id = argv[++i];
        } else if (strcmp(argv[i], "-s") == 0 && i + 1 < argc) {
            device_secret = argv[++i];
        }
    }
    if (!product_id || !device_id || !device_secret) {
        printf("device info error\n");
        return 1;
    }

    // 1. 初始化 SDK
    tc_iot_config_s config;
    memset(&config, 0, sizeof(config));
    config.storage_path = "./";
    config.log_level = TC_IOT_LOG_LEVEL_INFO;

    tc_iot_error_e rc = tc_iot_init(&config);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_init failed: %d\n", rc);
        return 1;
    }

    // 2. 登录
    tc_iot_device_info_s device_info;
    device_info.product_id = product_id;
    device_info.device_id = device_id;
    device_info.device_secret = device_secret;
    device_info.region = "ap-guangzhou";

    rc = tc_iot_login(&device_info, on_login);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_login call failed: %d\n", rc);
        tc_iot_deinit();
        return 1;
    }
```

```
    int waited_ms = 0;
    while (!s_login_done && waited_ms < 5000) {
        usleep(100 * 1000);
        waited_ms += 100;
    }
    if (!s_login_done || s_login_rc != TC_IOT_ERR_SUCCESS) {
        printf("login timeout or failed\n");
        tc_iot_deinit();
        return 1;
    }

    // 3. 释放资源
    tc_iot_logout();
    tc_iot_deinit();
    printf("[demo] main exit\n");
    return 0;
}
```

编译并运行

1. 在项目根目录执行 CMake 编译。

```
cmake -B build
cmake --build build -j
```

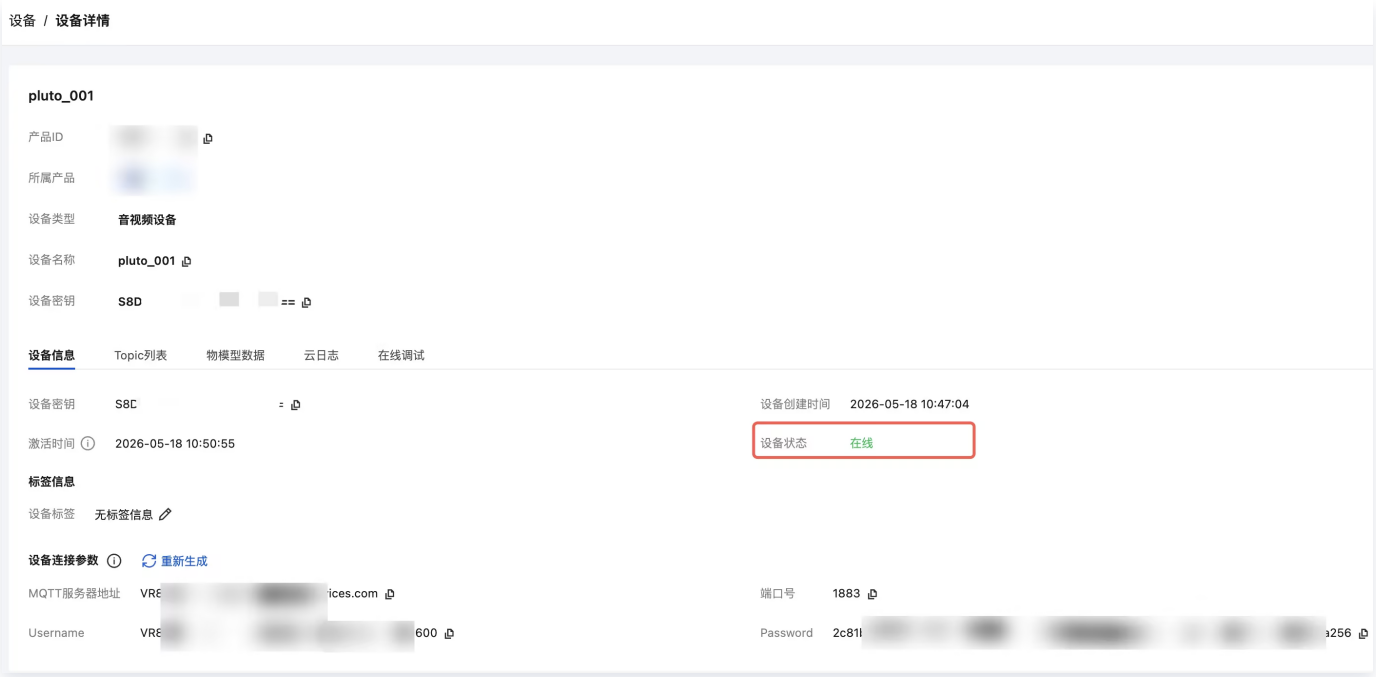
2. 运行可执行文件，传入设备三元组。

```
./build/iot_quickstart_demo -p YOUR_PRODUCT_ID -d YOUR_DEVICE_NAME -s
'YOUR_DEVICE_SECRET'
```

3. 观察日志输出，出现以下内容代表登录成功。

```
./build/iot_quickstart_demo -p YOUR_PRODUCT_ID -d YOUR_DEVICE_NAME -s 'YOUR_DEVICE_SECRET'
[IOT_SDK] 2026-07-14 03:30:28.691 I 2321809-3131684416|tc_iot.c:117|_do_iot_init|-storage_path=./ sdk_version=2.5.0 platform=linux trtc=c av=1 cs=1 talk=1
[IOT_SDK] 2026-07-14 03:30:28.691 I 2321809-3131684416|tc_iot_impl.c:181|tc_iot_impl_login|-product_id=YOUR_PRODUCT_ID|device_id=YOUR_DEVICE_NAME
[IOT_SDK] 2026-07-14 03:30:28.942 I 2321809-3131684416|mqtt_transport.c:418|_mqtt_on_tls_connected|-TLS connected
[IOT_SDK] 2026-07-14 03:30:29.141 I 2321809-3131684416|mqtt_connection.c:441|_mqtt_handle_connection_complete|-connected to YOUR_PRODUCT_ID.iotcloud.tencentdevices.com:8883
[IOT_SDK] 2026-07-14 03:30:29.141 I 2321809-3131684416|tc_iot_impl.c:276|_on_connect_result|-mqtt connect success|cost_time=450|product_id=YOUR_PRODUCT_ID|device_id=YOUR_DEVICE_NAME|retry_index=1
[demo] login success
[IOT_SDK] 2026-07-14 03:30:29.188 I 2321809-3131684416|mqtt_connection.c:1029|_mqtt_on_suback_packet_received|-subscribed: topic=$sys/operation/result/YOUR_PRODUCT_ID/YOUR_DEVICE_NAME
tc_iot_av_call failed: -4
[IOT_SDK] 2026-07-14 03:30:29.235 I 2321809-3131684416|mqtt_connection.c:1029|_mqtt_on_suback_packet_received|-subscribed: topic=$opt/operation/result/YOUR_PRODUCT_ID/YOUR_DEVICE_NAME
[IOT_SDK] 2026-07-14 03:30:29.284 I 2321809-3131684416|ntp_time.c:446|_on_receive_ntp_time_response|-ntp sync success|accurate_utc_ms:1783999829261|recv_tick_ms:16291676135|new_offset_ms=1767708153126|
now_utc_ms:1783999829261
[IOT_SDK] 2026-07-14 03:30:29.369 I 2321809-3131684416|mqtt_connection.c:1011|_mqtt_on_puback_packet_received|-puback received: packet_id=4 topic=$opt/operation/YOUR_PRODUCT_ID/YOUR_DEVICE_NAME
[IOT_SDK] 2026-07-14 03:30:29.369 I 2321809-3131684416|mqtt_connection.c:1029|_mqtt_on_suback_packet_received|-subscribed: topic=$trtc/down/service/YOUR_PRODUCT_ID/YOUR_DEVICE_NAME
[IOT_SDK] 2026-07-14 03:30:30.123 I 2321809-3131684416|server_config.c:274|_on_down_message|-received down message|topic=$opt/operation/result/YOUR_PRODUCT_ID/YOUR_DEVICE_NAME|size=164|payload=
{"type":"publish_server_config","clientToken":"7-1783999829235","timestamp":1783999829,"seq":2,"version":5,"param":{"disable_log_report":0,"disable_data_report":0}}
[IOT_SDK] 2026-07-14 03:30:32.202 I 2321809-3131684416|tc_iot_impl.c:396|tc_iot_impl_logout|-product_id=YOUR_PRODUCT_ID|device_id=YOUR_DEVICE_NAME
[IOT_SDK] 2026-07-14 03:30:32.202 I 2321809-3131684416|mqtt_connection.c:689|mqtt_connection_disconnect|-disconnecting
[IOT_SDK] 2026-07-14 03:30:32.202 I 2321809-3131684416|tc_iot_impl.c:421|tc_iot_impl_logout|-tc_iot_impl_logout done
[IOT_SDK] 2026-07-14 03:30:32.202 I 2321809-3131684416|tc_iot.c:183|_do_iot_deinit|-_do_iot_deinit
[demo] main exit
```

您可以进入 物联网开发平台控制台，在实例下的设备详情页面查看设备状态。



常见问题

现象	排查建议
`tc_iot_login` 长时间无回调 / 超时	检查设备三元组是否正确、网络是否可达、本机系统时间是否准确。
编译报错找不到头文件	确认 `CMakeLists.txt` 中 `target_include_directories` 正确指向 `tc_iot_sdk/include`。
链接报错找不到符号	确认链接顺序使用了 `-Wl,--start-group ... -Wl,--end-group` 包裹静态库，且补齐了 `pthread m dl z` 系统库。

联系我们

如果您在接入或使用过程中有任何疑问或者建议，欢迎 联系我们 提交反馈。

物模型通信

最近更新时间：2026-07-30 12:00:37

本文将介绍如何基于腾讯云物联网设备端 SDK，接入物模型（Data Model）能力，演示属性的上报与控制、事件的上报、行为的下发与回复等核心流程。

关键概念

物模型（Data Model）是物理世界中的设备在云端的数字化抽象。它将设备能力拆解为三个维度：属性（Property）、事件（Event）、行为（Action），让云端、应用端可以用统一的语言理解设备。

属性（Property）

设备运行时的状态或配置参数，既可被云端读取，也可被云端设置，是双向的。例如灯的开关、当前温度、亮度档位。

事件（Event）

设备主动上报到云端的信息，常用于告警、故障、运行状态变化等不可预期的场景，由设备单向推送。

行为（Action）

由云端下发给设备执行的命令，设备接收后需执行并回复结果。行为通常带有输入参数（云端下传）和输出参数（设备回传）。

data_id

属性、事件参数、行为参数的标识符，在同一物模型下唯一，调用 SDK 时必须与云端定义保持一致。

前提条件

在接入物模型通信功能前，请您先完成 快速接入 文档中跑通 `tc_iot_login` 登录功能。如果您已经完成，可以跳过该操作。

接入步骤

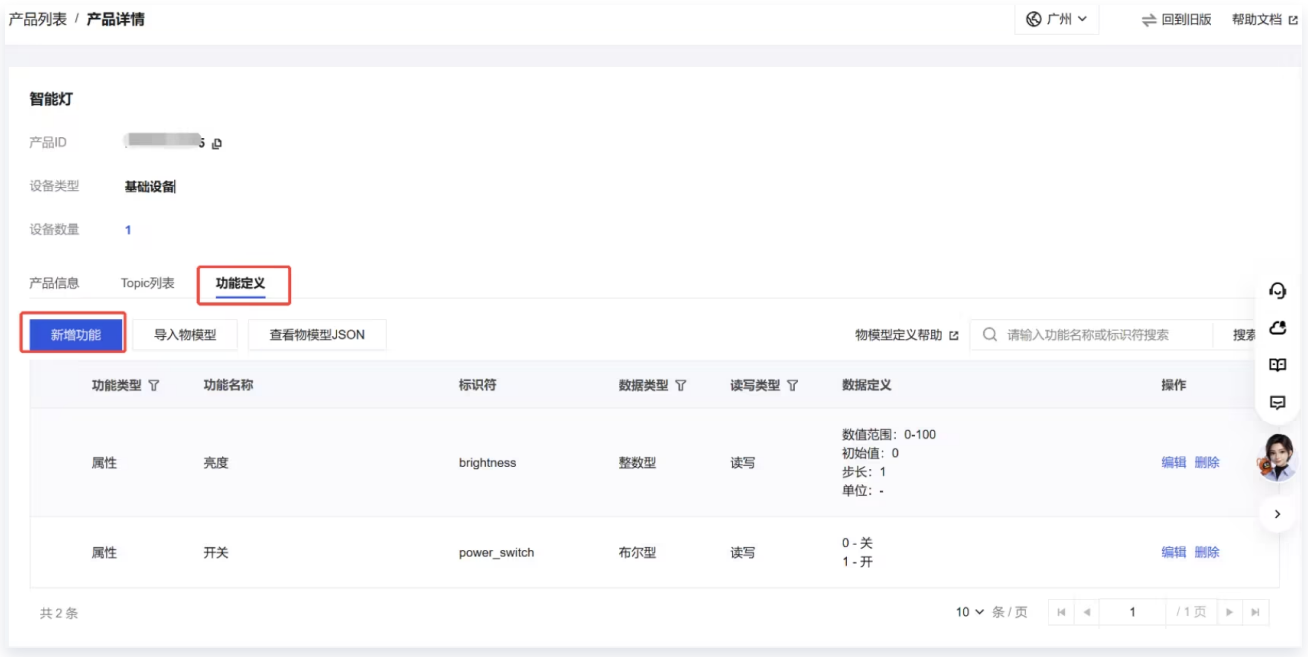
本节介绍如何调用 SDK 完成物模型的核心调用：上报属性、上报事件、接收云端对属性的设置、接收云端下发的行为并回复结果。

步骤 1：控制台定义物模型

控制台必须定义物模型，设备才可以上报物模型数据到服务器，上报未定义的物模型数据，服务器会拒绝。

1. 登录 物联网开发平台控制台，在产品列表中单击创建的产品名称进入详情。此处以“**智能灯**”产品为例。

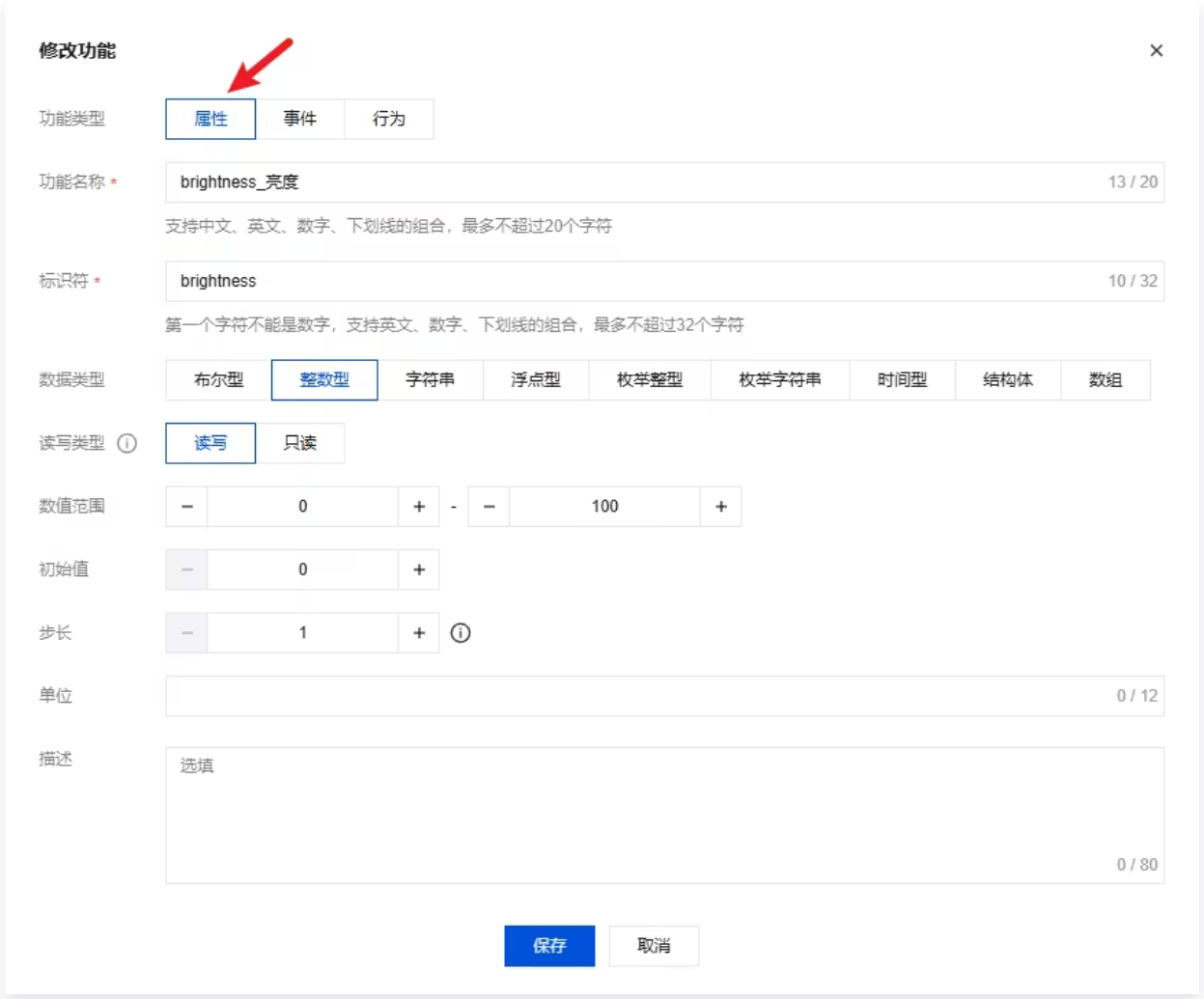
2. 选择**功能定义**，单击**新增功能**或**导入物模型**，成功后会自动生成对应功能类型。
   - 新增功能：在当前物模型基础上新增一条物模型属性，事件，或行为。
   - 导入物模型：将同类产品的物模型配置文件导入当前产品。



新增功能可以选择新增属性、事件或行为。

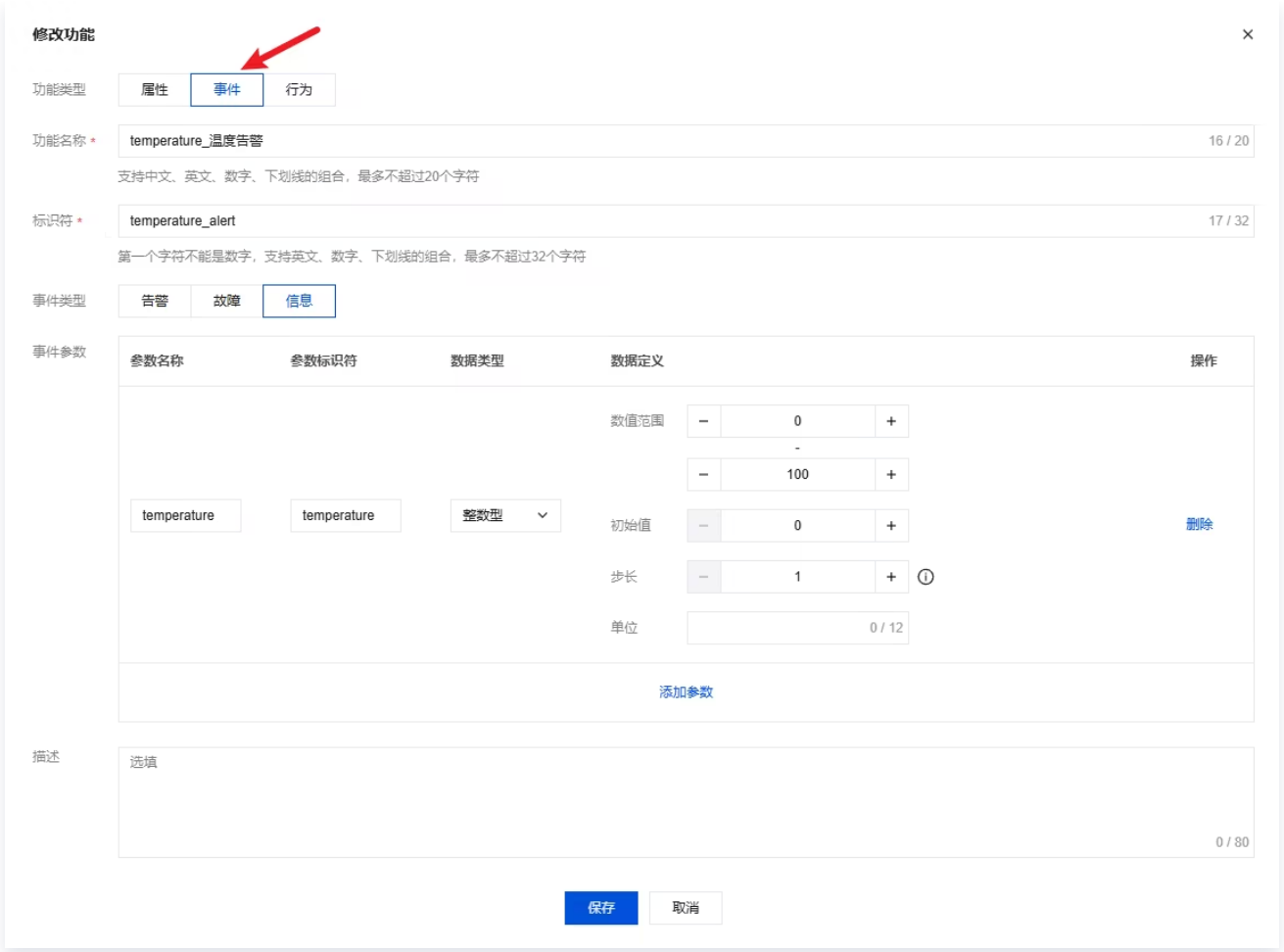
新增属性

界面如下图：


修改功能
功能类型
属性
事件
行为
功能名称 *
brightness_亮度
13 / 20
支持中文、英文、数字、下划线的组合，最多不超过20个字符
标识符 *
brightness
10 / 32
第一个字符不能是数字，支持英文、数字、下划线的组合，最多不超过32个字符
数据类型
布尔型
整数型
字符串
浮点型
枚举整型
枚举字符串
时间型
结构体
数组
读写类型
读写
只读
数值范围
0
100
初始值
0
步长
1
单位
0 / 12
描述
选填
0 / 80
保存
取消

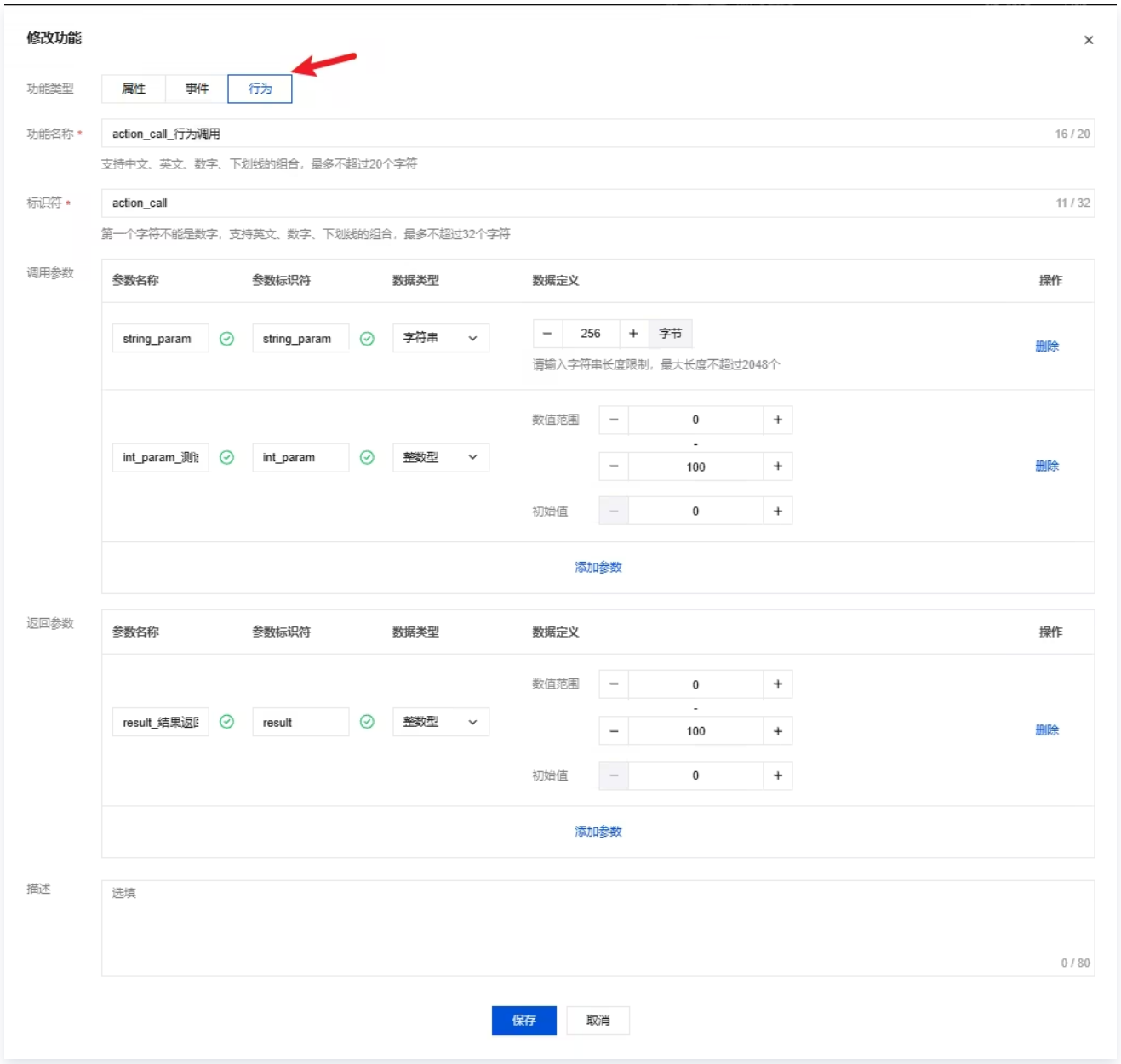
新增事件

界面如下图：


修改功能
功能类型
属性
事件
行为
功能名称 *
temperature_温度告警
16 / 20
支持中文、英文、数字、下划线的组合，最多不超过20个字符
标识符 *
temperature_alert
17 / 32
第一个字符不能是数字，支持英文、数字、下划线的组合，最多不超过32个字符
事件类型
告警
故障
信息
事件参数
参数名称
参数标识符
数据类型
数据定义
操作
temperature
temperature
整数型
数值范围
0
-
100
初始值
0
步长
1
单位
0 / 12
删除
添加参数
描述
选填
0 / 80
保存
取消

新增行为

界面如下图：



步骤 2：设备初始化物模型模块

登录成功后，调用 `tc_iot_data_model_init` 注册 `tc_iot_data_model_callback_t` 回调函数，物模型的回调如下：

回调	说明
`on_report_property_result_cb`	属性上报后，收到服务器响应或设备发送超时后触发，属性上报结果通过 `data_model_report_result_t` 告知用户。
`on_receive_property_changed_cb`	云端修改属性值时，触发设备回调，云端修改后的属性值通过回调参数 `data_model_property` 告知用户。

on_report_event_result_cb	事件上报后，收到服务器响应或网络超时后触发，事件上报结果通过 data_model_report_result_t 告知用户。
on_receive_new_action_cb	云端调用行为时，触发设备回调，云端下发的行为输入参数通过回调参数 data_model_action 告知用户。

调用 tc_iot_data_model_init 设置回调，初始化物模型模块。

```
static void on_report_property_result_cb(char *property_id, void
*user_data,
                                         data_model_report_result_t
result_code);
static void on_receive_property_changed_cb(char *property_id,
                                           tc_iot_data_model_property_t
*data_model_property);
static void on_report_event_result_cb(char *event_id, void *user_data,
                                      data_model_report_result_t
result_code);
static int  on_receive_new_action_cb(char *action_id,
                                      tc_iot_data_model_action_t
*data_model_action);


tc_iot_data_model_callback_t callback;
memset(&callback, 0, sizeof(callback));
callback.on_report_property_result_cb   = on_report_property_result_cb;
callback.on_receive_property_changed_cb =
on_receive_property_changed_cb;
callback.on_report_event_result_cb      = on_report_event_result_cb;
callback.on_receive_new_action_cb       = on_receive_new_action_cb;


tc_iot_error_e dm_rc = tc_iot_data_model_init(callback);
if (dm_rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_data_model_init failed: %d\n", dm_rc);
}
```

属性和事件上报结果 data_model_report_result_t 取值和说明：

取值	说明
DATA_MODEL_REPORT_SUCCESS	上报成功，云端已确认。
DATA_MODEL_REPORT_REJECTED	上报被云端拒绝。
DATA_MODEL_REPORT_NO_RESPONSE	上报无响应（通常为超时）。
DATA_MODEL_REPORT_LOCAL_TIMEOUT	本地超时。

步骤 3：设备上报属性

调用 `tc_iot_data_model_report_property` 上报一条或多条属性到云端。属性支持以下数据类型（data_model_data_type_t）：

取值	说明
DATA_MODEL_DATA_TYPE_BOOL	布尔值。
DATA_MODEL_DATA_TYPE_INT	32 位整数。
DATA_MODEL_DATA_TYPE_FLOAT	双精度浮点。
DATA_MODEL_DATA_TYPE_STRING	字符串。
DATA_MODEL_DATA_TYPE_TIME	32 位无符号时间戳（秒）。
DATA_MODEL_DATA_TYPE_ENUM	32 位无符号枚举。

上报一个布尔属性（灯的开关）：

```
#include "tc_iot_data_model.h"

tc_iot_data_model_property_t property[1];
memset(property, 0, sizeof(property));
```

```
char power_switch_id[] = "power_switch";
property[0].property.data_id = power_switch_id;
property[0].property.data_type = DATA_MODEL_DATA_TYPE_BOOL;
property[0].property.data_value.value_bool = true;

rc = tc_iot_data_model_report_property(property, 1, NULL);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_data_model_report_property failed: %d\n", rc);
}
```

上报多种不同类型的属性：

```
tc_iot_data_model_property_t property[4];
memset(property, 0, sizeof(property));

char id_power[]  = "power_switch";
char id_bright[] = "brightness";
char id_color[]  = "color";
char id_name[]   = "device_name";

property[0].property.data_id   = id_power;
property[0].property.data_type = DATA_MODEL_DATA_TYPE_BOOL;
property[0].property.data_value.value_bool = true;

property[1].property.data_id   = id_bright;
property[1].property.data_type = DATA_MODEL_DATA_TYPE_INT;
property[1].property.data_value.value_int = 80;

property[2].property.data_id   = id_color;
property[2].property.data_type = DATA_MODEL_DATA_TYPE_ENUM;
property[2].property.data_value.value_enum = 2;

property[3].property.data_id   = id_name;
property[3].property.data_type = DATA_MODEL_DATA_TYPE_STRING;
property[3].property.data_value.value_string = "test_device_001";

rc = tc_iot_data_model_report_property(property, 4, NULL);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_data_model_report_property failed: %d\n", rc);
```

```
}
```

说明：

- tc_iot_data_model_report_property 是异步接口，返回值仅表示参数是否合法以及消息是否成功入队；真正的上报结果通过 on_report_property_result_cb 异步回调返回。
- data_id 必须与控制台物模型中定义的属性标识符完全一致，否则云端会返回错误码。
- data_type 与 data_value 的 union 字段必须一一对应（例如 data_type=INT 时只能写 value_int）。

步骤 4：设备上报事件

调用 `tc_iot_data_model_report_event` 上报一个或多个事件到云端。事件类型（data_model_event_type_t）：

取值	说明
DATA_MODEL_EVENT_TYPE_INFO	普通信息事件，例如设备上线、心跳等。
DATA_MODEL_EVENT_TYPE_ALERT	告警事件，例如温度过高、门被打开。
DATA_MODEL_EVENT_TYPE_FAULT	故障事件，例如传感器异常。

上报一个告警事件（带参数）：

```
#include "tc_iot_data_model.h"

tc_iot_data_model_event_t event[1];
memset(event, 0, sizeof(event));

char event_id[] = "temperature_alert";
event[0].event_id = event_id;
event[0].event_type = DATA_MODEL_EVENT_TYPE_ALERT;

data_model_data_item_t event_data[1];
memset(event_data, 0, sizeof(event_data));

char temp_id[] = "temperature";
event_data[0].data_id = temp_id;
```

```
event_data[0].data_type = DATA_MODEL_DATA_TYPE_INT;
event_data[0].data_value.value_int = 85;

event[0].event_data_list = event_data;
event[0].event_data_num  = 1;

rc = tc_iot_data_model_report_event(event, 1, NULL);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_data_model_report_event failed: %d\n", rc);
}
```

上报一个故障事件（不带参数）：

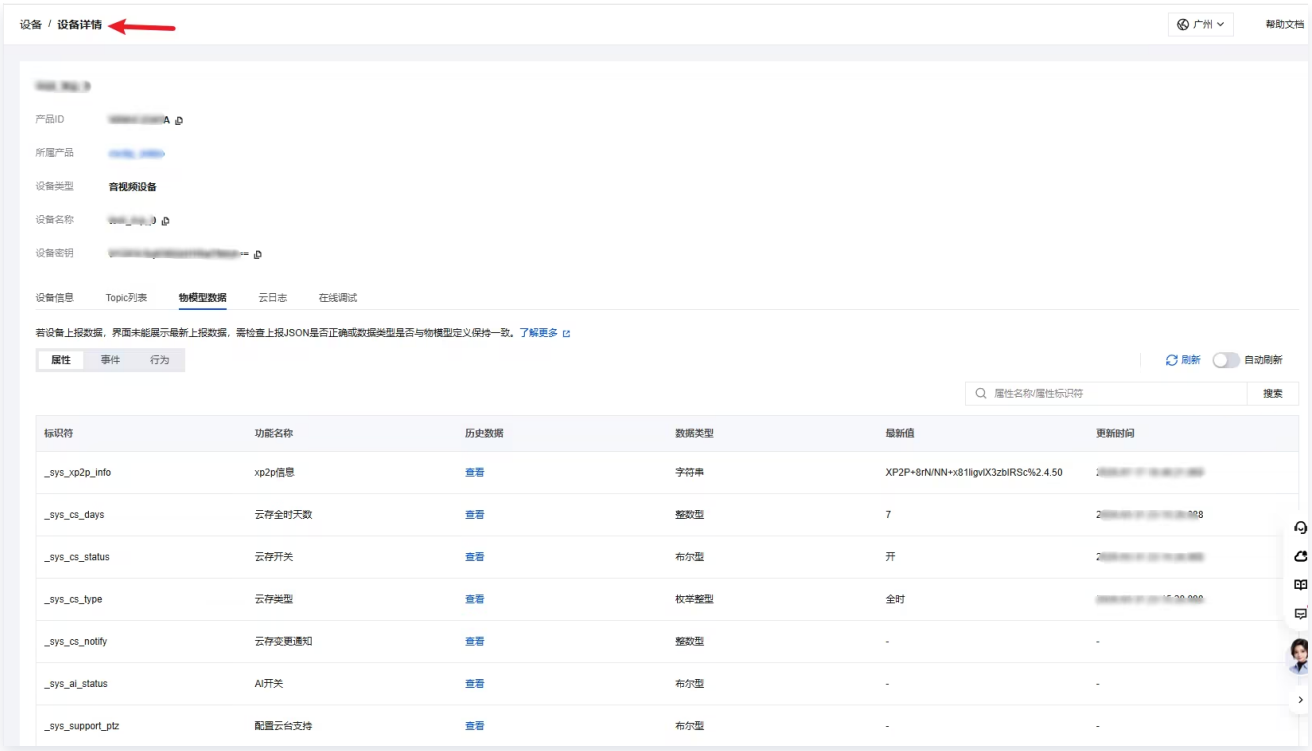
```
tc_iot_data_model_event_t fault[1];
memset(fault, 0, sizeof(fault));

fault[0].event_id = "device_error";
fault[0].event_type = DATA_MODEL_EVENT_TYPE_FAULT;
fault[0].event_data_num = 0;
fault[0].event_data_list = NULL;

rc = tc_iot_data_model_report_event(fault, 1, NULL);
if (rc != TC_IOT_ERR_SUCCESS) {
    printf("tc_iot_data_model_report_event failed: %d\n", rc);
}
```

说明：

- 事件的 event_data_list 中的 data_id 同样必须与云端物模型中该事件定义的参数标识符一致。
- 事件如果没有参数，可以将 event_data_num 置为 0、event_data_list 置为 NULL。

步骤 5：控制台查看设备上报的属性和事件

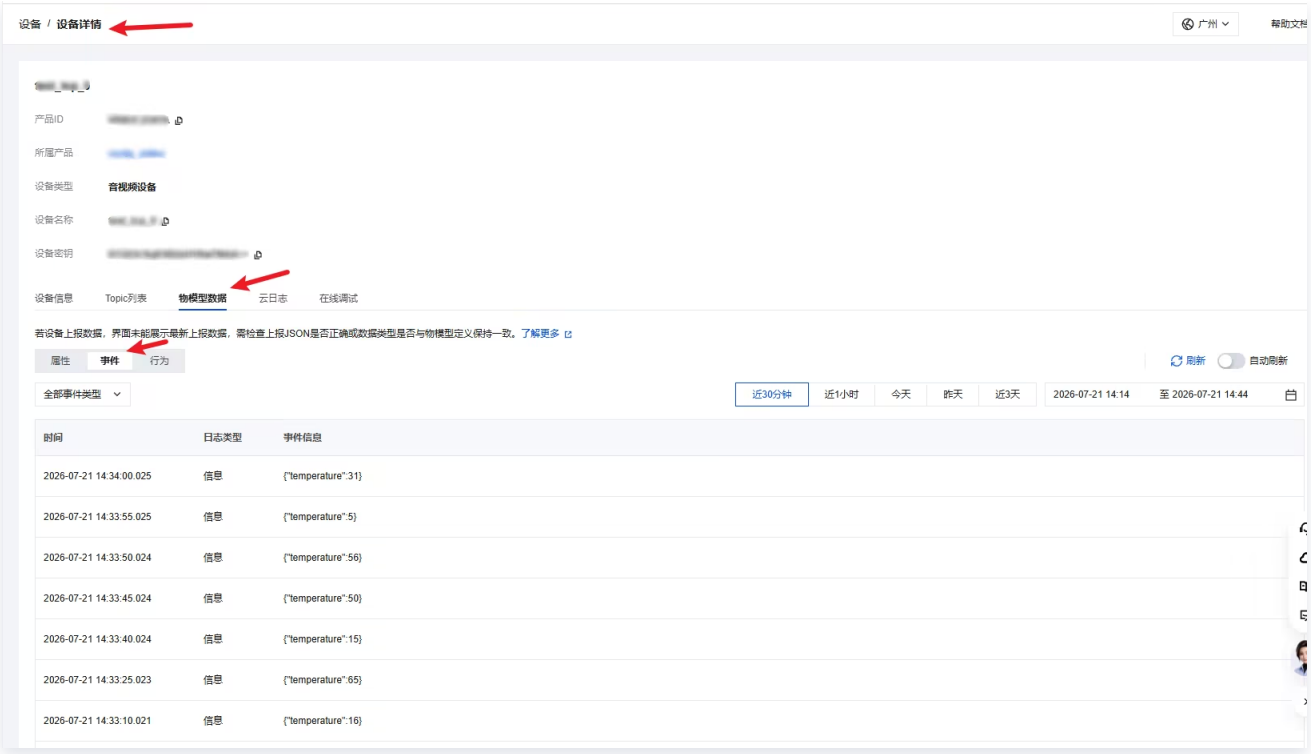
1. 登录 物联网开发平台控制台，在**设备**列表单击设备名称进入详情页面，选择**物模型数据**即可查看设备上报的属性和事件，



2. 选择需要查看的属性，单击**查看**。

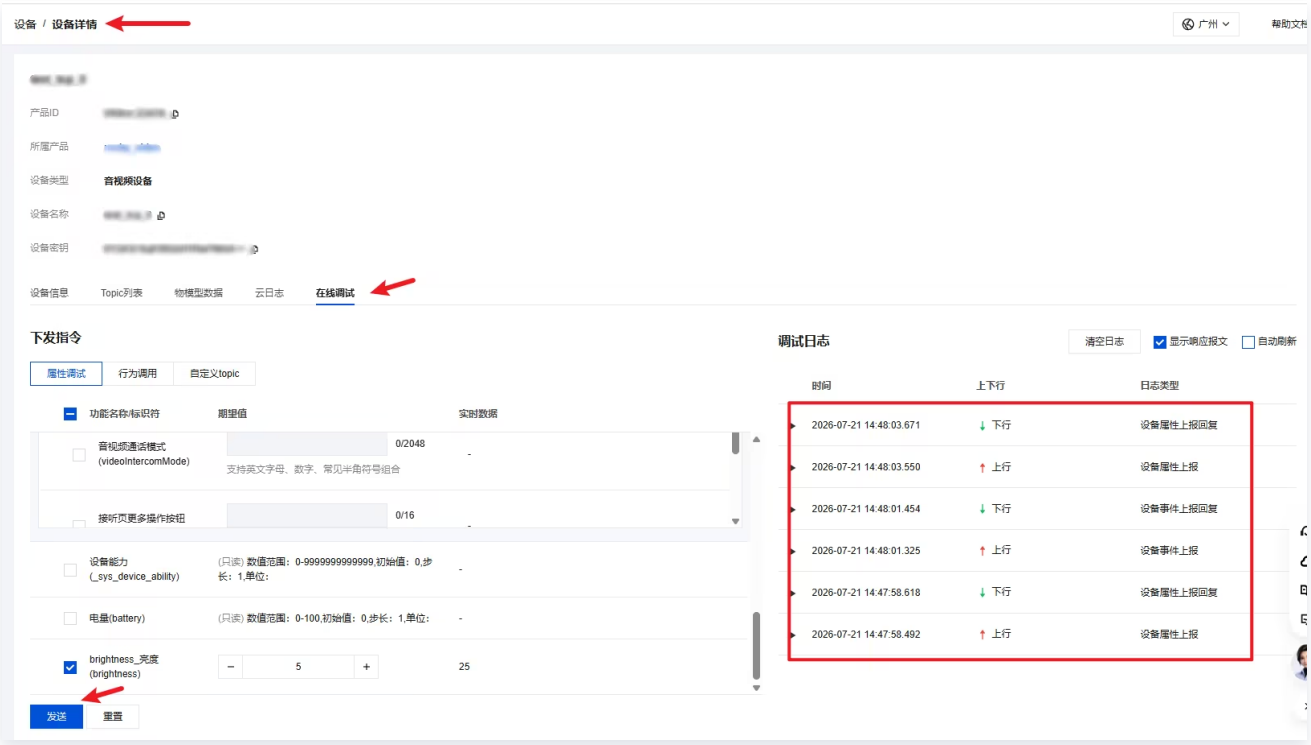
可以查看属性的变化曲线，以及每次上报的属性值。

3. 在设备详情页选择**物模型数据**后，单击**事件**，即可查看设备上报的所有事件数据。

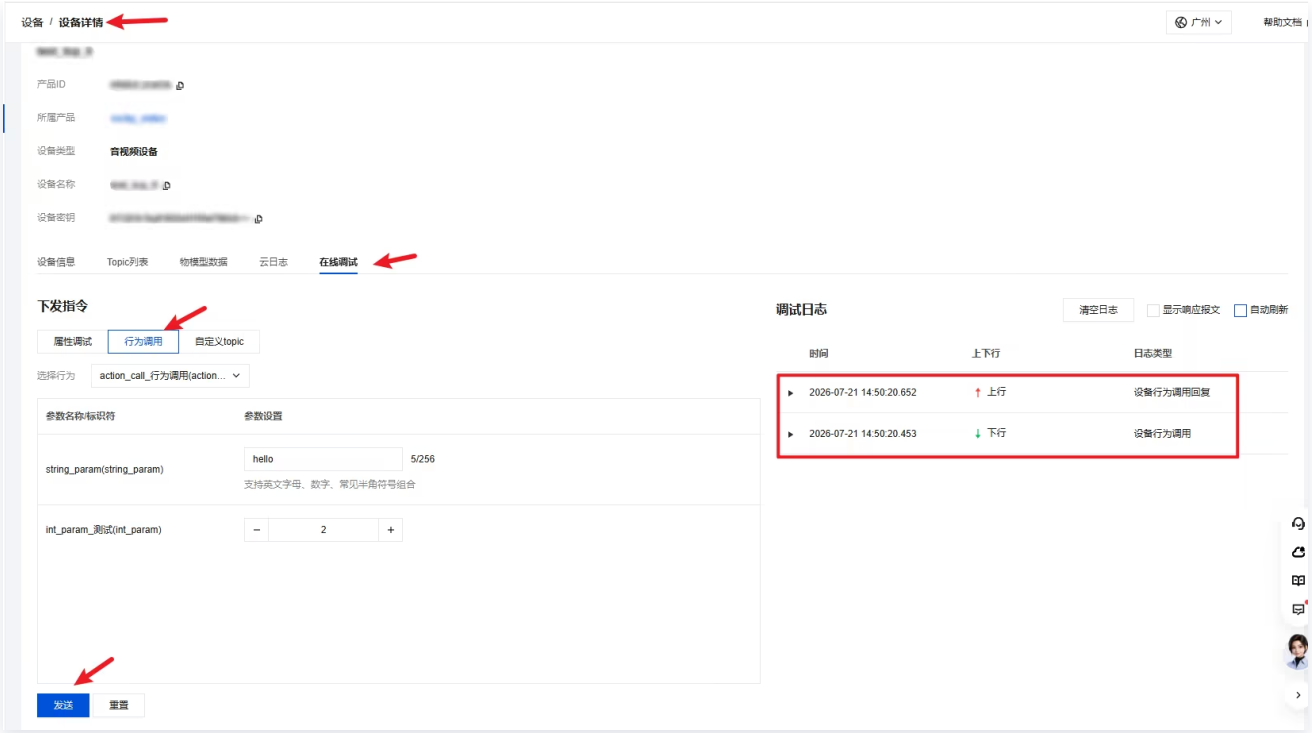


步骤 6：控制台修改属性和调用行为

1. 登录 物联网开发平台控制台 ，在**设备**列表单击设备名称进入详情页面。
2. 选择**在线调试** > **属性调试**，勾选要修改的属性，填写修改后的数据，单击**发送**即可



3. 选择**在线调试** > **行为调用**，在左侧设置行为参数并发送，右侧会显示行为调用时间，以及设备对行为调用的响应报文和响应时间。



步骤 7：设备端接收属性变更

当云端修改了设备的某个属性时，SDK 会通过 on_receive_property_changed_cb 回调通知上层。根据回调参数 tc_iot_data_model_property_t 确定属性名称，属性数据类型，属性数据值。

```
static void on_receive_property_changed_cb(char *property_id,
                                            tc_iot_data_model_property_t
*data_model_property) {
    if (property_id == NULL || data_model_property == NULL) {
        return;
    }

    data_model_data_item_t *item = &data_model_property->property;
    printf("[demo] property changed: id=%s, type=%d\n", item->data_id,
(int)item->data_type);

    switch (item->data_type) {
        case DATA_MODEL_DATA_TYPE_BOOL:
            // 根据 item->data_value.value_bool 调整设备状态
            break;
        case DATA_MODEL_DATA_TYPE_INT:
            // 根据 item->data_value.value_int 调整设备状态
            break;
        case DATA_MODEL_DATA_TYPE_FLOAT:
```

```
            // 根据 item->data_value.value_float 调整设备状态
            break;
        case DATA_MODEL_DATA_TYPE_STRING:
            // 根据 item->data_value.value_string 调整设备状态
            break;
        case DATA_MODEL_DATA_TYPE_ENUM:
            // 根据 item->data_value.value_enum 调整设备状态
            break;
        case DATA_MODEL_DATA_TYPE_TIME:
            // 根据 item->data_value.value_time 调整设备状态
            break;
        default:
            break;
    }
}
```

说明：
on_receive_property_changed_cb 触发后，设备如果修改了属性值，可以调用 tc_iot_data_model_report_property 将最新的属性值上报云端。

步骤 8：设备端接收行为调用

当云端调用设备的行为时，SDK 会通过 on_receive_new_action_cb 回调通知上层。
回调返回 int：0 表示执行成功，非0表示执行失败。
如果行为定义了输出参数，回调里需要自行给 action_output_data_list / action_output_data_num 赋值，SDK 会自动把结果回传给云端。

```
static int on_receive_new_action_cb(char *action_id,
tc_iot_data_model_action_t *data_model_action) {
    if (action_id == NULL || data_model_action == NULL) {
        return -1;
    }

    printf("[demo] receive new action: id=%s, token=%s, timestamp=%u,
input_num=%d\n",
           action_id,
           data_model_action->token ? data_model_action->token : "",
           (unsigned)data_model_action->timestamp,
```

```
            data_model_action->action_input_data_num);

    // 1. 解析输入参数
    for (int i = 0; i < data_model_action->action_input_data_num; i++) {
        data_model_data_item_t *input = &data_model_action->action_input_data_list[i];
        printf("    input[%d]: id=%s, type=%d\n", i, input->data_id, (int)input->data_type);
        // 根据业务需要读取 input->data_value ...
    }

    // 2. 执行行为逻辑（同步执行，避免在回调里做长时间阻塞）

    // 3. 填充输出参数（按云端定义的输出参数顺序和类型）
    int output_num = data_model_action->action_input_data_num;  // 示例：与输入参数个数保持一致
    data_model_action->action_output_data_num = output_num;
    data_model_action->action_output_data_list = NULL;
    if (output_num > 0) {
        data_model_action->action_output_data_list =
            (data_model_data_item_t *)calloc(output_num, sizeof(data_model_data_item_t));
        if (data_model_action->action_output_data_list == NULL) {
            return -2;  // 内存不足，回复云端失败
        }
        for (int i = 0; i < output_num; i++) {
            data_model_data_item_t *input  = &data_model_action->action_input_data_list[i];
            data_model_data_item_t *output = &data_model_action->action_output_data_list[i];

            output->data_id = input->data_id;        // 与云端约定好的输出参数标识
            output->data_type = input->data_type;    // 与云端约定好的输出参数类型
            if (input->data_type == DATA_MODEL_DATA_TYPE_INT) {
                output->data_value.value_int = input->data_value.value_int + 1;
            } else {
                output->data_value = input->data_value;
```

```
            }
        }
    }

    // 4. 返回 0：执行成功；返回非 0：执行失败，云端会收到对应错误码
    return 0;
}
```

说明：

- 如果行为没有输出参数，把 action_output_data_num 置为 0、action_output_data_list 置为 NULL 即可。
- 返回 0 表示执行成功，云端收到的 action_reply 消息中 code=0、status=success；返回非 0 时 code=返回值、status=action execution failed。

完整示例代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include "tc_iot.h"
#include "tc_iot_data_model.h"

static volatile int s_login_done = 0;
static tc_iot_error_e s_login_rc = TC_IOT_ERR_SUCCESS;

// 登录回调
static void on_login(tc_iot_error_e error_code, const char
*error_message) {
    s_login_rc = error_code;
    s_login_done = 1;
    if (error_code == TC_IOT_ERR_SUCCESS) {
        printf("[demo] login success\n");
    } else {
        printf("[demo] login failed: %d, msg=%s\n", error_code,
               error_message ? error_message : "");
```

```
    }
}

// 属性上报结果回调
static void on_report_property_result_cb(char *property_id, void
*user_data,
                                         data_model_report_result_t
result_code) {
    printf("[demo] report property result: id=%s, code=%d\n",
           property_id ? property_id : "(null)", (int)result_code);
}

// 事件上报结果回调
static void on_report_event_result_cb(char *event_id, void *user_data,
                                      data_model_report_result_t
result_code) {
    printf("[demo] report event result: id=%s, code=%d\n",
           event_id ? event_id : "(null)", (int)result_code);
}

// 云端修改属性
static void on_receive_property_changed_cb(char *property_id,
                                           tc_iot_data_model_property_t
*data_model_property) {
    if (!property_id || !data_model_property) {
        return;
    }
    data_model_data_item_t *item = &data_model_property->property;
    printf("[demo] property changed: id=%s, type=%d\n", item->data_id,
(int)item->data_type);
    // 按 item->data_type 解析 item->data_value 并更新本地状态 ...
}

// 云端下发行为
static int on_receive_new_action_cb(char *action_id,
tc_iot_data_model_action_t *data_model_action) {
    if (!action_id || !data_model_action) {
        return -1;
    }
    printf("[demo] receive new action: id=%s, input_num=%d\n",
```

```
            action_id, data_model_action->action_input_data_num);

    // 解析输入参数并执行业务逻辑（示例：直接回传）
    int n = data_model_action->action_input_data_num;
    data_model_action->action_output_data_num = n;
    data_model_action->action_output_data_list = NULL;
    if (n > 0) {
        data_model_action->action_output_data_list =
            (data_model_data_item_t *)calloc(n,
sizeof(data_model_data_item_t));
        if (!data_model_action->action_output_data_list) {
            return -2;
        }
        for (int i = 0; i < n; i++) {
            data_model_data_item_t *in  = &data_model_action-
>action_input_data_list[i];
            data_model_data_item_t *out = &data_model_action-
>action_output_data_list[i];
            out->data_id   = in->data_id;
            out->data_type = in->data_type;
            out->data_value = in->data_value;  // 示例：原样回传
        }
    }
    return 0;
}

int main(int argc, char **argv) {
    if (argc < 7) {
        printf("Usage: %s -p <product_id> -d <device_id> -s
<device_secret>\n", argv[0]);
        return 0;
    }

    const char *product_id = NULL;
    const char *device_id = NULL;
    const char *device_secret = NULL;
    for (int i = 1; i < argc; i++) {
        if (strcmp(argv[i], "-p") == 0 && i + 1 < argc) {
            product_id = argv[++i];
        } else if (strcmp(argv[i], "-d") == 0 && i + 1 < argc) {
```

```
            device_id = argv[++i];
        } else if (strcmp(argv[i], "-s") == 0 && i + 1 < argc) {
            device_secret = argv[++i];
        }
    }
    if (!product_id || !device_id || !device_secret) {
        printf("device info error\n");
        return 1;
    }

    // 1. 初始化 SDK
    tc_iot_config_s config;
    memset(&config, 0, sizeof(config));
    config.storage_path = "./";
    config.log_level = TC_IOT_LOG_LEVEL_INFO;

    tc_iot_error_e rc = tc_iot_init(&config);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_init failed: %d\n", rc);
        return 1;
    }

    // 2. 登录
    tc_iot_device_info_s device_info;
    device_info.product_id    = product_id;
    device_info.device_id     = device_id;
    device_info.device_secret = device_secret;
    device_info.region        = "ap-guangzhou";

    rc = tc_iot_login(&device_info, on_login);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_login call failed: %d\n", rc);
        tc_iot_deinit();
        return 1;
    }

    int waited_ms = 0;
    while (!s_login_done && waited_ms < 5000) {
        usleep(100 * 1000);
        waited_ms += 100;
```

```
    }
    if (!s_login_done || s_login_rc != TC_IOT_ERR_SUCCESS) {
        printf("login timeout or failed\n");
        tc_iot_deinit();
        return 1;
    }

    // 3. 初始化物模型模块
    tc_iot_data_model_callback_t callback = {
        .on_report_property_result_cb    = on_report_property_result_cb,
        .on_receive_property_changed_cb =
on_receive_property_changed_cb,
        .on_report_event_result_cb       = on_report_event_result_cb,
        .on_receive_new_action_cb        = on_receive_new_action_cb,
    };
    rc = tc_iot_data_model_init(callback);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_data_model_init failed: %d\n", rc);
        tc_iot_logout();
        tc_iot_deinit();
        return 1;
    }

    // 4. 上报一个布尔属性: power_switch = true
    tc_iot_data_model_property_t property[1];
    memset(property, 0, sizeof(property));
    char id_power[] = "power_switch";
    property[0].property.data_id    = id_power;
    property[0].property.data_type = DATA_MODEL_DATA_TYPE_BOOL;
    property[0].property.data_value.value_bool = true;

    rc = tc_iot_data_model_report_property(property, 1, NULL);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_data_model_report_property failed: %d\n", rc);
    }

    // 5. 上报一个告警事件: temperature_alert, 参数 temperature = 85
    tc_iot_data_model_event_t event[1];
    memset(event, 0, sizeof(event));
    char id_event[] = "temperature_alert";
```

```
    event[0].event_id   = id_event;
    event[0].event_type = DATA_MODEL_EVENT_TYPE_ALERT;

    data_model_data_item_t event_data[1];
    memset(event_data, 0, sizeof(event_data));
    char id_temp[] = "temperature";
    event_data[0].data_id   = id_temp;
    event_data[0].data_type = DATA_MODEL_DATA_TYPE_INT;
    event_data[0].data_value.value_int = 85;

    event[0].event_data_list = event_data;
    event[0].event_data_num  = 1;

    rc = tc_iot_data_model_report_event(event, 1, NULL);
    if (rc != TC_IOT_ERR_SUCCESS) {
        printf("tc_iot_data_model_report_event failed: %d\n", rc);
    }

    // 6. 等待云端交互（属性修改、行为调用等），观察回调打印
    sleep(10);

    // 7. 释放资源
    tc_iot_data_model_deinit();
    tc_iot_logout();
    tc_iot_deinit();
    printf("[demo] main exit\n");
    return 0;
}
```

编译并运行

1. 在项目根目录执行 CMake 编译。

```
cmake -B build
cmake --build build -j
```

2. 运行可执行文件，传入设备三元组。

```
./build/iot_quickstart_demo -p YOUR_PRODUCT_ID -d YOUR_DEVICE_NAME -s
'YOUR_DEVICE_SECRET'
```

3. 观察日志输出，出现以下内容代表登录与信令发送均成功。

```
[demo] login success
[demo] report property result: id=power_switch, code=0
[demo] report event result: id=temperature_alert, code=0
```

云端在控制台对设备发起一次行为调用（如 set_brightness），设备端会打印：

```
[demo] receive new action: id=set_brightness, input_num=1 input[0]:
id=brightness, type=2
```

云端在控制台修改一次属性（如把 brightness 改为 90），设备端会打印：

```
[demo] property changed: id=brightness, type=2
```

常见问题

现象	排查建议
`tc_iot_data_model_init` 返回 TC_IOT_ERR_NOT_INITIALIZED	确认在调用前已成功调用 tc_iot_init 和 tc_iot_login，且 on_login 回调返回 TC_IOT_ERR_SUCCESS。
`tc_iot_data_model_init` 返回 TC_IOT_ERR_ALREADY_INITIALIZED	物模型模块只允许初始化一次，确认没有重复调用，或先调用 tc_iot_data_model_deinit 再重新 init。
tc_iot_data_model_report_property 返回 TC_IOT_ERR_INVALID_ARGUMENT	检查 property 指针是否为空、property_num 是否 ≤ 0。
云端收不到上报的属性 / 事件	检查 data_id、event_id 是否与控制台物模型定义完全一致；data_type 与 data_value 的 union 字段是否对应正确。

on_report_property_result_cb 回调里 result_code ≠ DATA_MODEL_REPORT_SUCCESS	查看 SDK 日志中的错误码；常见原因有网络抖动、MQTT 断连、设备未登录。
云端行为调用,设备执行后并响应后，云端没有收到结果 / 一直 pending	确认 on_receive_new_action_cb 在合理时间内返回（0 表示成功）；回调中不要做阻塞或 sleep；输出参数按云端约定填写。
编译报错找不到头文件	确认 CMakeLists.txt 中 target_include_directories 正确指向 tc_iot_sdk/include。

联系我们

如果您在接入或使用过程中有任何疑问或者建议，欢迎 联系我们 提交反馈。