物联网开发平台

旧版文档

腾讯云

文档目录

旧版文档
旧控制台操作指南
开发中心
项目管理

最近更新时间：2023-06-01 16:25:24

操作场景

项目是为了用户面对不同的客户、不同的产品迭代或不同的项目角色而设计的一种隔离机制，便于用户清晰管理物联网项目，并能灵活地配置项目权限。

- 项目下可以建立多个产品与应用，应用默认有权限访问该项目下的产品。
- 每个项目会有自己的唯一 ID。数据会根据项目进行隔离，以确保您客户的数据安全。
- 项目删除后，该项目所属产品等数据都将被删除且不能恢复。
- 开发平台提供资源级的权限控制，可为不同的子用户分配项目级、产品级的权限控制。

操作步骤

新建项目

1. 登录 物联网开发平台 ，选择**公共实例**或您购买的**标准企业实例**。
2. 进入项目列表页面，单击**新建项目**
   - **项目名称**：根据实际业务输入便于识别的项目名称。
   - **项目描述**：输入项目的备注信息。

3. 单击**保存**，即可新建项目。

修改项目

1. 选择需要修改的项目，单击项目右侧**编辑**进入项目详情页。

2. 支持修改项目名称与项目描述，单击**保存**后，返回项目列表页。

删除项目

> **注意：**
> 为了防止误操作删除数据影响您的业务，若该项目下还有产品数据，则不允许删除项目。

1. 当您无需该项目时，您可以在该项目的右侧，单击**删除**。

2. 确认删除后，系统将删除该项目。

产品开发
产品定义

最近更新时间：2024-12-23 16:49:12

操作场景

用户成功注册腾讯云账号后，通过物联网开发平台将设备对接到腾讯云物联网平台时，需要创建项目，并在项目下创建产品、定义产品的数据模板。本文档主要介绍如何使用开发平台创建项目并进行产品定义。

操作说明

新建项目

> **说明：**
> 项目是为了方便灵活地管理不同项目权限而设置的，如一个项目涉及多方合作，需要将设备端、应用端开发人员的腾讯云账号都添加到该项目下，待某合作方退出，则可将该账号从项目中删除。
> - 项目下可以建立多个产品与应用，应用默认有权限访问该项目下的产品。
> - 每个项目会有自己的唯一 ID。数据会根据项目进行隔离，以确保您的客户的数据安全。
> - 项目删除后，该项目所属产品等数据都将被删除且不能恢复。

1. 登录 物联网开发平台，选择**公共实例**或您购买的**标准企业实例**。
2. 进入项目列表页面，单击**新建项目**。
   - **项目名称**：根据实际业务输入便于识别的项目名称。
   - **项目描述**：输入项目的备注信息。
3. 单击**保存**，则项目创建成功。

删除项目

> **注意：**
> 为了防止误操作删除数据影响您的业务，若该项目下还有产品数据，则不允许直接删除项目，需删除产品后才允许删除项目。

1. 完成新项目创建后，您可以在项目列表页面查看创建的项目。
2. 当您无需该项目时，您可以在该项目的右侧，单击**删除**并确认即可。

创建产品

产品相当于某一类设备的集合，用户通过产品管理其下的所有设备。

1. 单击创建的项目进入产品开发中心，单击**新建产品**，定义您的产品。
2. 根据页面提示填写产品基本信息，然后单击**保存**。

产品基本信息设置如下：

- 产品名称：名称为中文、字母、数字、下划线的组合，1位 – 20位且不能为空。
- 产品品类：选择您所创建产品的所属品类，不同类型产品的属性、事件等数据模板会有所不同。详情请参见 数据模板 。
- 设备类型：设备类型分为设备、网关、子设备3类，详情如下：
  - 设备：此类设备可直接接入物联网开发平台且无挂载子设备。
  - 网关：此类设备可直接接入物联网开发平台，并且可接受子设备加入局域网络。可以维持子设备的拓扑关系，将与子设备的拓扑关系同步到云端。
  - 子设备：此类设备必须依托网关设备才可与物联网开发平台进行通信，例如 Zigbee、蓝牙、RF433 等设备。网关与子设备的说明，请参考文档 网关子设备 。
- 认证方式：物联网开发平台提供两种认证方式用于设备与平台之间鉴权认证。
  - 证书认证：在创建设备时，平台将为设备生成一个证书文件和一个私钥文件，实现设备与云之间的双向认证。
  - 密钥认证：在创建设备时，使用平台为设备随机生成的 PSK。

- 通信方式：您可以选择 Wi-Fi、移动蜂窝（2G/3G/4G）、5G、BLE、LoRaWAN 等其它通信方式。
- 接入网关协议（设备类型选择为子设备时可选）：表示该产品下的设备作为子设备与网关的通讯协议类型。
  - Zigbee：表示子设备和网关间的通讯协议为ZigBee。
  - BLE：表示子设备和网关间的通讯协议为BLE。
  - 433：表示子设备和网关间的通讯协议为433。
  - 自定义：表示子设备和网关间的通讯协议为其它标准或私有协议。
- 数据协议：默认采用物模型的数据协议，您也可以自定义协议进行透传。
- 描述：字数不能超过80个，您可以根据需要选填。

删除产品

> **注意：**
> 为了防止误操作删除产品影响您的业务，若该产品下还有设备，则不允许删除该产品。

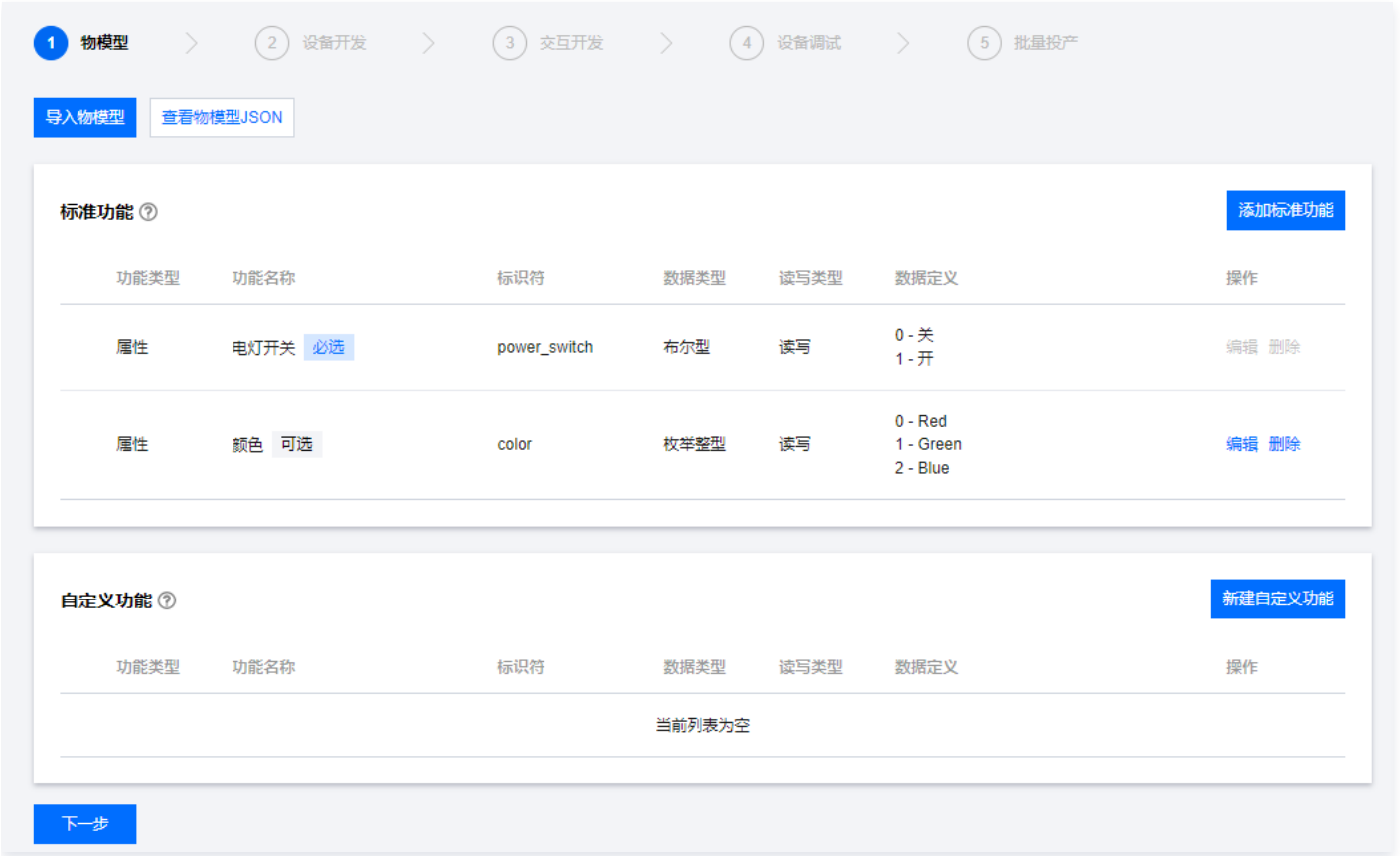
1. 完成新产品创建后，您可以在产品列表页面查看创建的产品。
2. 当您无需该产品时，您可以在该产品的右上方，单击**删除**并确认即可。

物模型

最近更新时间：2023-06-01 16:25:24

简介

物模型通过将物理实体设备进行数字化描述，构建其数字模型。在物联网开发平台定义物模型即定义产品功能。完成功能定义后，系统将自动生成该产品的物模型。
单击已创建的产品，进入产品开发环节，产品开发第一步即定义物模型。



功能类型

产品的功能包括标准功能和自定义功能。

功能类型	功能描述
标准功能	该产品类别下提供的常用功能，默认已创建。分为必选属性和可选属性，必选属性不可删除，其中部分属性可编辑；可选属性则可删除、可编辑。
自定义功能	如果标准功能无法满足您的需求，您可以自定义功能。可由开发者自由删除和编辑。 注意：**已发布的产品不能编辑、删除属性及事件**。

功能类型包含三元素：属性、事件和行为。

功能元素	功能描述	功能标识符
属性	用于描述设备的实时状态，支持读取和设置，如模式、亮度、开关等。 包括以下六种基本数据类型： • 布尔型：非真即假的二值型变量。例如，开关功能。 • 整数型：可用于线性调节的整数变量。例如，空调的温度。 • 字符型：以字符串形式表达的功能点，例如，灯的位置。 • 浮点型：精度为浮点型的功能点。例如，压力值的范围：0.0 – 24.0。 • 枚举型：自定义的有限集合值。例如，灯的颜色：白色、红色、黄色等。	PropertiesId

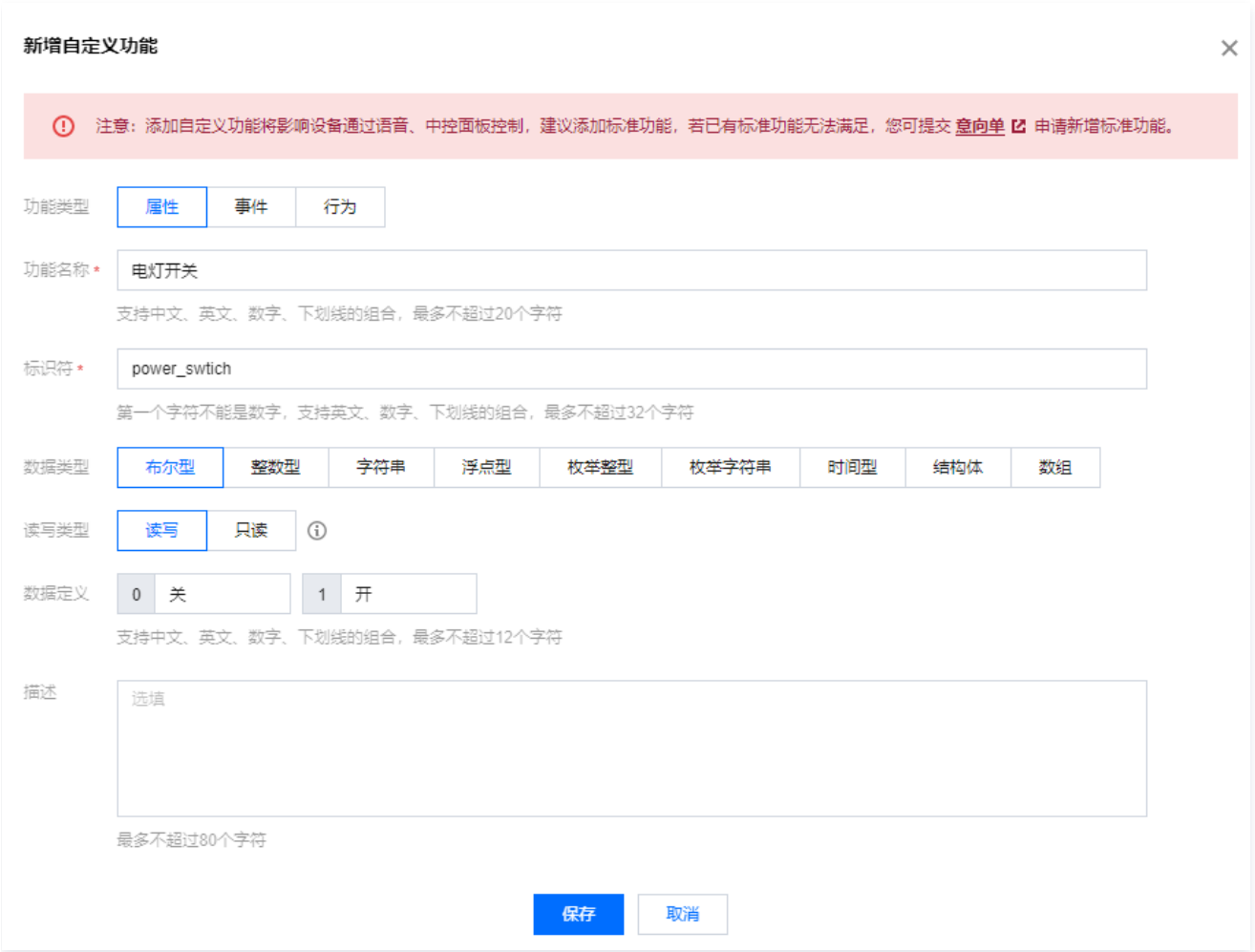
	• 时间型：string 类型的 UTC 时间戳（毫秒）。	
事件	用于描述设备运行时的事件，包括告警、信息和故障三种事件类型，事件型功能属性可以添加具体的事件参数，这些参数可以由属性中六种基本数据类型组成。可添加多个输出参数，例如环境传感器检测到空气质量很差，空调异常告警等。	EventId
行为	用于实现更复杂的业务逻辑，可添加多个调用参数和返回参数。行为的输入参数和输出参数可添加属性中六种基本数据类型，用于让设备执行某项特定的任务。例如，开锁动作需要知道是哪个用户在什么时间开锁，锁的状态如何等。	ActionId

功能示例

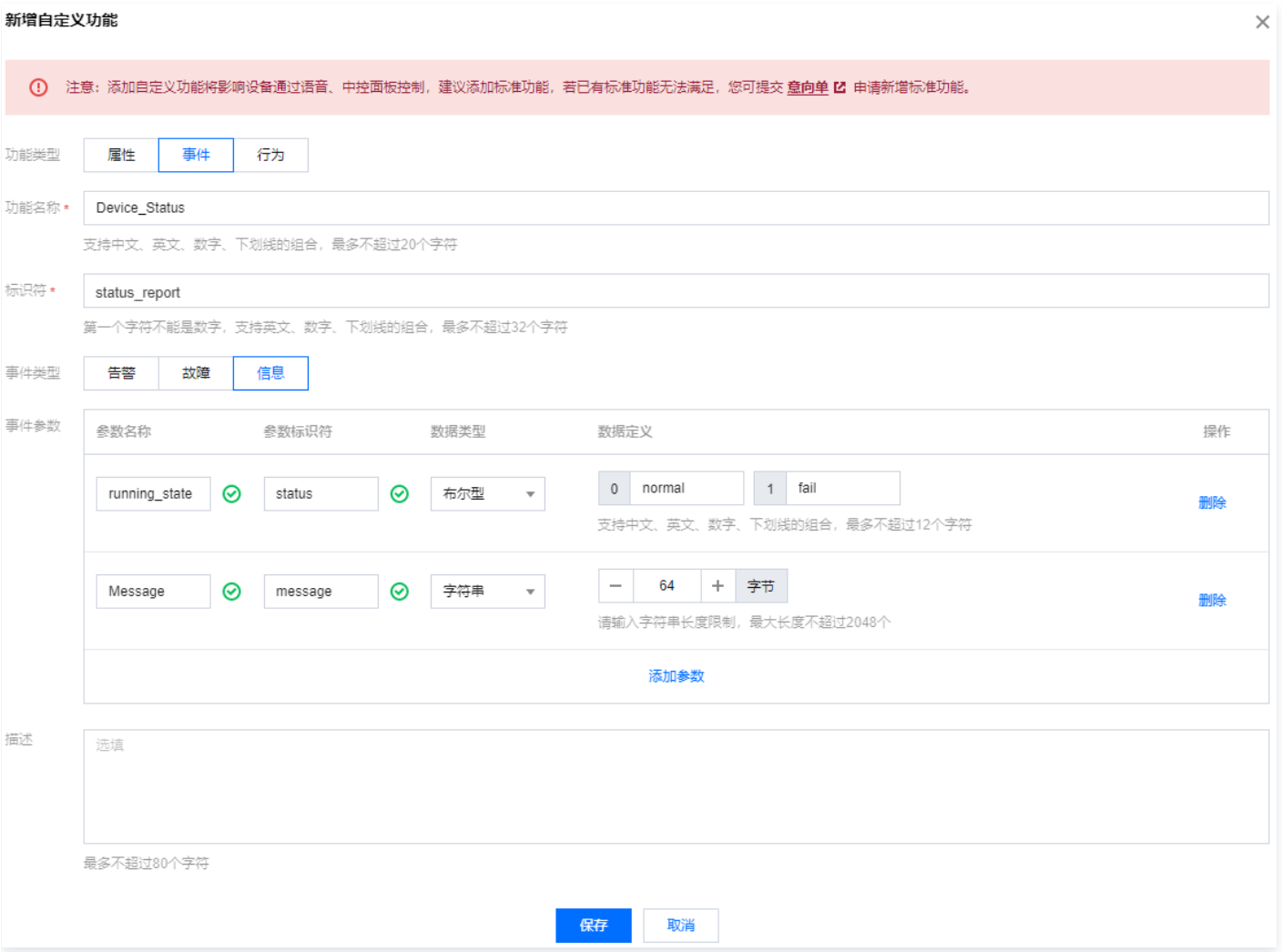
属性、事件功能在用户创建的产品已包含一定标准功能，若用户需要根据业务场景新增，也可在自定义功能栏中新增。

> ⚠ **注意：**
> **添加自定义功能将影响设备通过语音、中控面板控制，建议添加标准功能，若已有标准功能无法满足，您可提交 意向单 申请新增标准功能。**

下文提供了部分自定义功能示例：
进入控制台目标产品物模型页，单击添加自定义功能，进入对应设置界面：
属性示例：

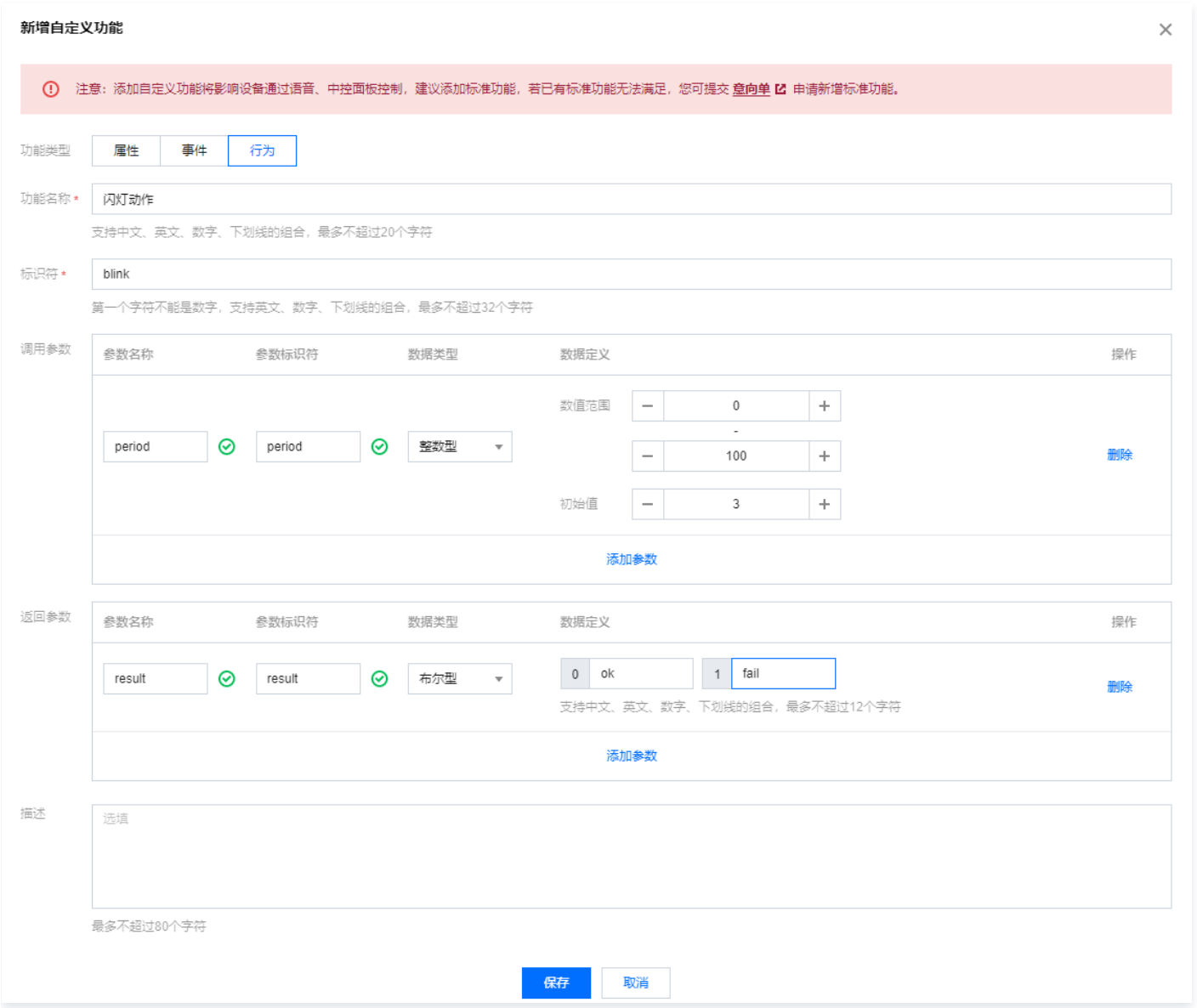


事件示例：
选择事件功能，EventId 为用户自定义的事件标识符。描述内容为选填，可填写对事件功能的描述，描述功能场景，在什么情况下触发事件等。

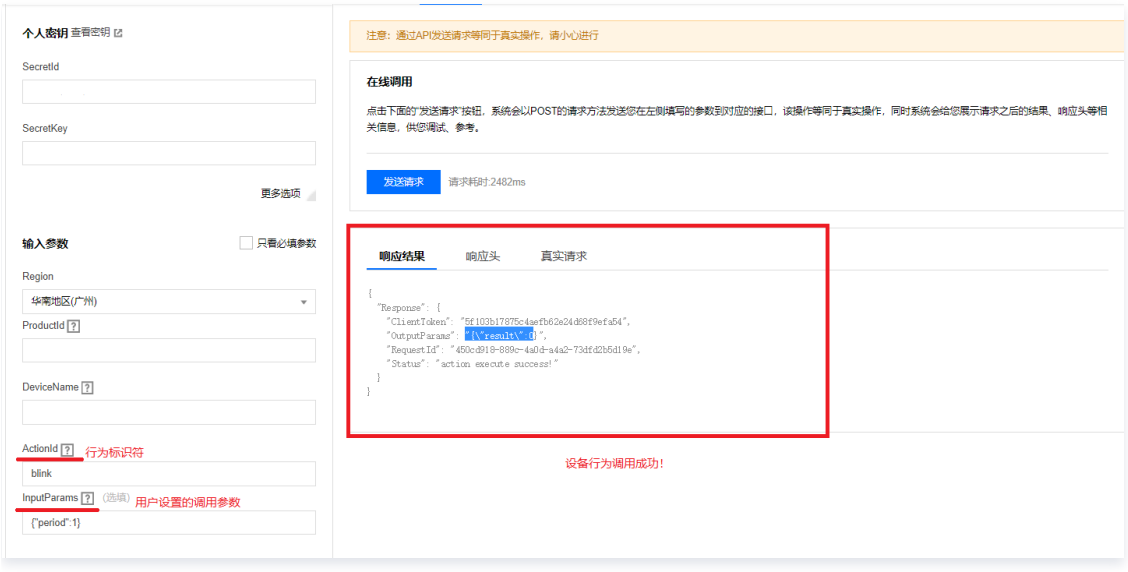


行为示例：

选择行为功能，根据用户场景设置行为信息，ActionId 为用户自定义的行为标识符。如图，设置闪烁行为功能，将行为标识符设置为 blink（ActionId），设置调用参数和返回参数，实现闪灯动作。



行为功能设置完毕后，可在腾讯云提供的开发者资源 API Explorer 3.0 中实现设备行为调用，API 调用时，填入对应的 ActionId 和输入参数。



设备端收到 Action 消息后，配合实现对应的动作，C-SDK 提供数据模板的自动代码生成及属性、事件、动作的响应框架。

> **说明：**
> 数据模板，它是一个 JSON 格式的文件，使用数据模板协议，用户的设备需按数据模板定义要求传输设备数据到云端，并可使用基于数据模板的诸多业务功能。

数据模板格式参考

以下为智能高级路灯的数据模板字段描述说明，包括各种数据类型和事件类型。示例代码如下：

```
{
  "version": "1.0",
  "profile": {
    "ProductId": "2300UMK31M",
    "CategoryId": "3"
  },
  "properties": [
    {
      "id": "power_switch",
      "name": "电灯开关",
      "desc": "控制电灯开灭",
      "required": true,
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      }
    },
    {
      "id": "color",
      "name": "颜色",
      "desc": "灯光颜色",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "0": "Red",
          "1": "Green",
          "2": "Blue"
        }
      }
    },
    {
      "id": "brightness",
      "name": "亮度",
      "desc": "灯光亮度",
      "mode": "rw",
      "define": {
        "type": "int",
        "unit": "%",
        "step": "1",
        "min": "0",
        "max": "100",
        "start": "1"
      }
    },
    {
      "id": "name",
```

```
      "name": "灯位置名称",
      "desc": "灯位置名称：书房、客厅等",
      "mode": "rw",
      "required": false,
      "define": {
        "type": "string",
        "min": "0",
        "max": "64"
      }
    }
  ],
  "events": [
    {
      "id": "status_report",
      "name": "DeviceStatus",
      "desc": "Report the device status",
      "type": "info",
      "required": false,
      "params": [
        {
          "id": "status",
          "name": "running_state",
          "desc": "Report current device running state",
          "define": {
            "type": "bool",
            "mapping": {
              "0": "normal",
              "1": "fault"
            }
          }
        },
        {
          "id": "message",
          "name": "Message",
          "desc": "Some extra message",
          "define": {
            "type": "string",
            "min": "0",
            "max": "64"
          }
        }
      ]
    },
    {
      "id": "low_voltage",
      "name": "LowVoltage",
      "desc": "Alert for device voltage is low",
      "type": "alert",
      "required": false,
      "params": [
        {
          "id": "voltage",
          "name": "Voltage",
          "desc": "Current voltage",
          "define": {
            "type": "float",
            "unit": "V",
            "step": "1",
            "min": "0.0",
            "max": "24.0",
            "start": "1"
```

```
          }
        }
      ]
    },
    {
      "id": "hardware_fault",
      "name": "Hardware_fault",
      "desc": "Report hardware fault",
      "type": "fault",
      "required": false,
      "params": [
        {
          "id": "name",
          "name": "Name",
          "desc": "Name like: memory,tf card, censors ...",
          "define": {
            "type": "string",
            "min": "0",
            "max": "64"
          }
        },
        {
          "id": "error_code",
          "name": "Error_Code",
          "desc": "Error code for fault",
          "define": {
            "type": "int",
            "unit": "",
            "step": "1",
            "min": "0",
            "max": "2000",
            "start": "1"
          }
        }
      ]
    }
  ],
  "actions": [
    {
      "id": "unlock",
      "name": "开灯行为检测",
      "desc": "用于描述开灯的动作",
      "input": [
        {
          "id": "open",
          "name": "开关",
          "define": {
            "type": "bool",
            "mapping": {
              "0": "关",
              "1": "开"
            }
          }
        },
        {
          "id": "user",
          "name": "用户",
          "define": {
            "type": "string",
            "min": "0",
            "max": "2048"
```

```
          }
        }
      ],
      "output": [
        {
          "id": "user",
          "name": "用户",
          "define": {
            "type": "string",
            "min": "0",
            "max": "2048"
          }
        },
        {
          "id": "time",
          "name": "开灯时间",
          "define": {
            "type": "timestamp"
          }
        },
        {
          "id": "state",
          "name": "灯的状态",
          "define": {
            "type": "bool",
            "mapping": {
              "0": "关",
              "1": "开"
            }
          }
        }
      ],
      "required": false
    }
  ]
}
```

设备开发

最近更新时间：2024-12-23 16:49:12

操作场景

用户定义完产品与物模型后，需要按接入协议要求将设备接入到平台。本文档主要介绍如何使用开发平台进行设备开发。

设备开发说明

用户在 物联网开发平台 创建完产品并定义完产品的数据模板后，可单击**设备开发**，开发平台目前提供三种开发方式。

- 基于模组开发：满足 MCU 以串口通信方式，并通过通信模组与云端通信的场景。
- 基于 SDK 开发：满足直接集成 C SDK 的接入场景。
- 基于 OS 开发：满足基于物联网操作系统集成 C SDK 的接入场景。

基于模组开发

1. 如果您的设备需要通过通信模组连接开发平台，则单击**基于模组开发**。

2. 系统显示模组选择窗口，您需要根据您的业务需求选择合适的通信模组。包括模组品牌与模组通信类型，选择合适的模组后可单击**确定**。

3. 选择模组后，可单击**重新选择**更换模组，也可以单击**查看详情**了解模组的详细参数，还可以单击**采购咨询**去模组公司采购。

4. 嵌入式开发。
   - 通过模组对接的设备，如果定义了数据模板，则平台提供了 MCU SDK 代码自动生成的功能，MCU SDK 代码用于加快 MCU 如何对接通信模组。
   - 单击 **MCU SDK 代码**，开发平台会生成一个压缩文件，您下载后即可遵循开发指引将设备对接到开发平台。
   - 如何基于下载的 MCU SDK 代码进行 MCU 开发，详情请参见 设备开发指南 相关文档。

5. 下载 AT 指令协议
   单击 AT 指令协议 可了解腾讯云 IoT AT 指令协议。

基于 OS 开发

1. 如果您的设备所运行的物联网操作系统为 SDK 已经对接支持的 OS 类型，可以单击**基于 OS 开发** 查看基于对应物联网操作系统接入平台的开发指南。
2. 嵌入式开发。
   - 数据模板配置文件生成：如果已创建所定义产品的数据模板及事件，您可根据指南文档了解如何将数据模板生成模板代码、如何基于生成的数据模板配置文件以及数据模板示例进行业务逻辑开发。
   - OS 代码下载：提供腾讯物联网终端操作系统 TencentOS tiny 和 RT-Thread 的下载路径。
   - 开发指引：提供基于不同物联网操作系统接入腾讯云物联网开发平台的开发指南。

交互开发

最近更新时间：2024-09-30 17:54:51

操作场景

用户可通过官方小程序或自主品牌小程序与设备互动，物联网开发平台提供了交互开发配置服务，简化了小程序的开发难度。您可以通过简单的配置，实现小程序与平台的数据通信，快速拥有移动应用端的能力。

前提条件

已完成 设备开发 阶段工作。

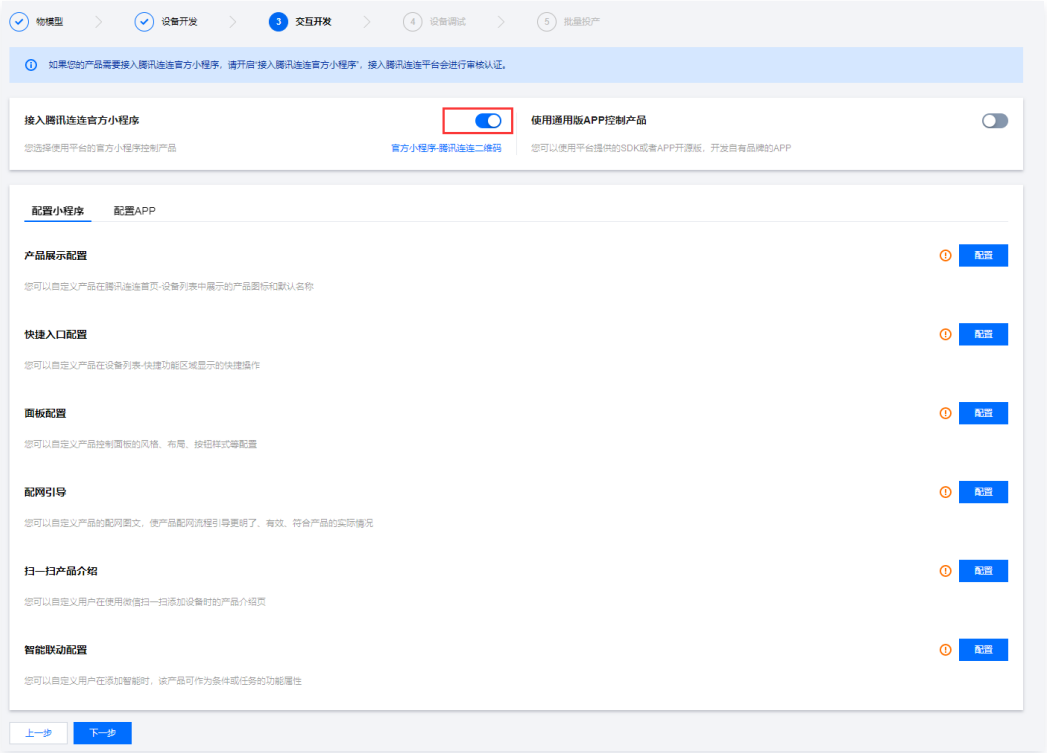
操作步骤

控制产品方式

物联网开发平台提供小程序和 App 两种应用端的形式，并且支持官方公版小程序、自主品牌小程序、通用版 App、自主品牌 App 四种应用类型的交互方式配置。进入交互开发页面后，您可以根据您的业务需求，通过开关按钮选择使用上述应用类型进行交互开发配置。

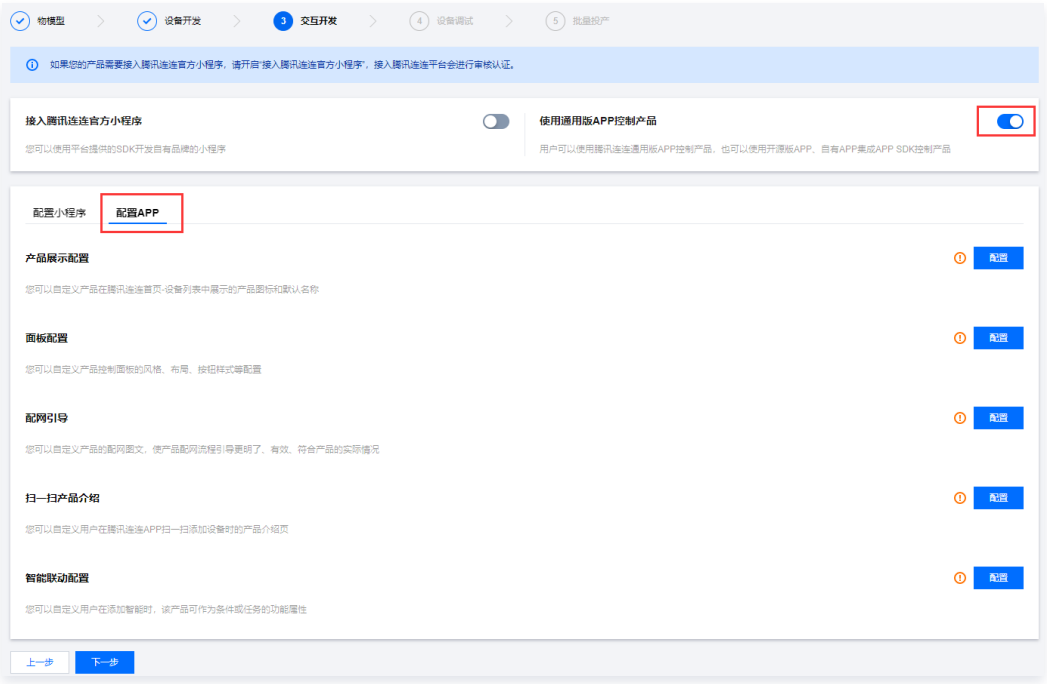
> **说明：**
> 如果您的产品需要接入腾讯连连官方小程序，请开启“接入腾讯连连官方小程序”，接入腾讯连连平台会进行审核认证。

- 官方公版小程序：使用腾讯连连官方小程序控制产品，则开启**接入腾讯连连官方小程序**开关并进行相应配置。



- 自主品牌小程序：使用物联网开发平台小程序 SDK 开发自主品牌的小程序，则关闭**接入腾讯连连官方小程序**开关并进行相应配置。

- 通用版 App：使用腾讯连连通用版 App 控制产品，则打开**使用通用版 App 控制产品**并切换至**配置 App** 进行相应配置。



- 自主品牌 App：使用开源版 App 和 App SDK 开发自主品牌的 App，则关闭**使用通用版 App 控制产品**并切换至**配置 App** 进行相应配置。

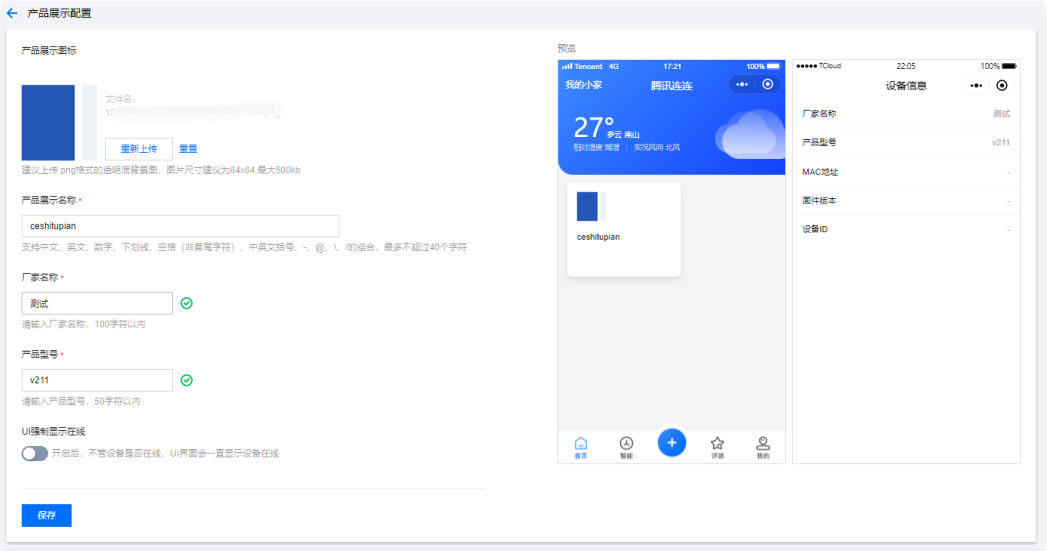
说明：

> 小程序与 App 的配置相互独立。如果您的业务既使用小程序应用，也使用 App 应用，请在**配置小程序**页与**配置 App** 页分别进行配置，相应的配置将分别在您的小程序和 App 中生效。

本实例为您介绍使用官方小程序控制产品的交互开发配置过程：

产品展示配置

1. 单击**产品展示配置**项后的**配置**。
2. 进入产品展示配置页，展示默认产品图片，如需更换图片，可单击**重新上传**更换自定义产品图片。



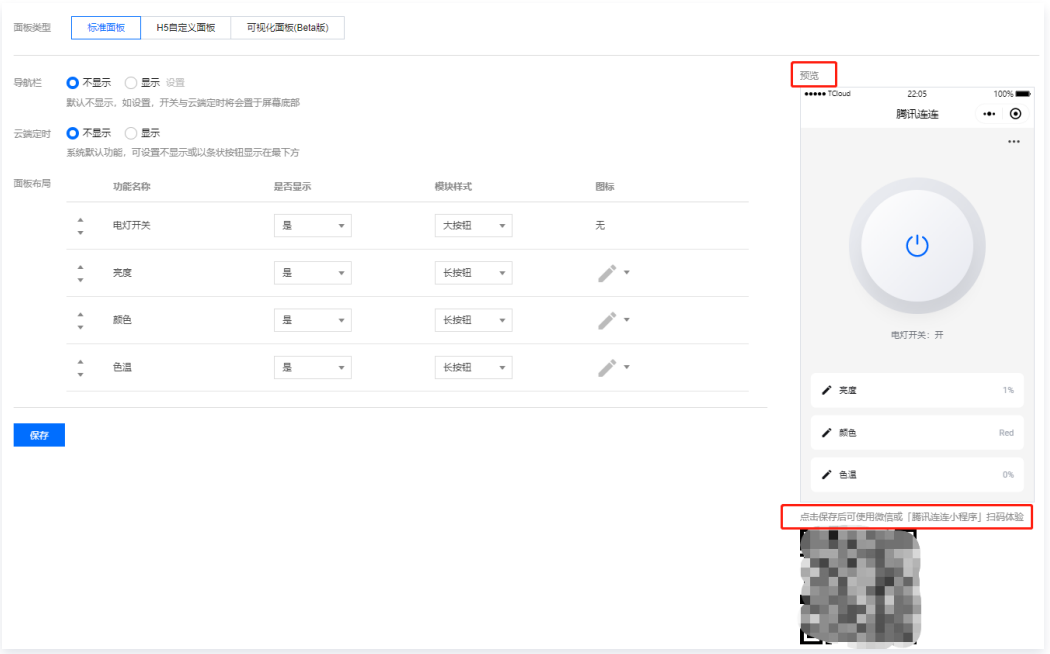
3. 填写厂家名称及产品型号。
4. 按实际需求选择是否开启 **UI强制显示在线**。开启后，不管设备是否在线，UI 界面会一直显示设备在线。
5. 单击**保存**完成自定义产品图片配置。
6. 保存成功后，刷新腾讯连连小程序绑定列表，即可查看最新自定义上传的 icon。

面板配置

通过面板配置，您可以自定义产品控制面板的风格、布局、按钮样式等配置。可选择标准面板或者H5自定义面板，标准控制面板无需进行二次开发，只需要在云端配置后即可在小程序生效。若对面板有个性化需求，可根据 H5 自定义面板开发 文档进行开发。

1. 单击面板配置栏中的**配置**。
2. 进入面板配置页面，选择标准面板。
   - 导航栏：可设置是否显示导航栏，并支持对导航栏进行设置。
   - 云端定时：可以设置是否展示云端定时按钮，若展示，则以条状展示在最下方。
   - 面板布局：支持自定义面板内按钮的布局，包括按钮样式、位置、图标。

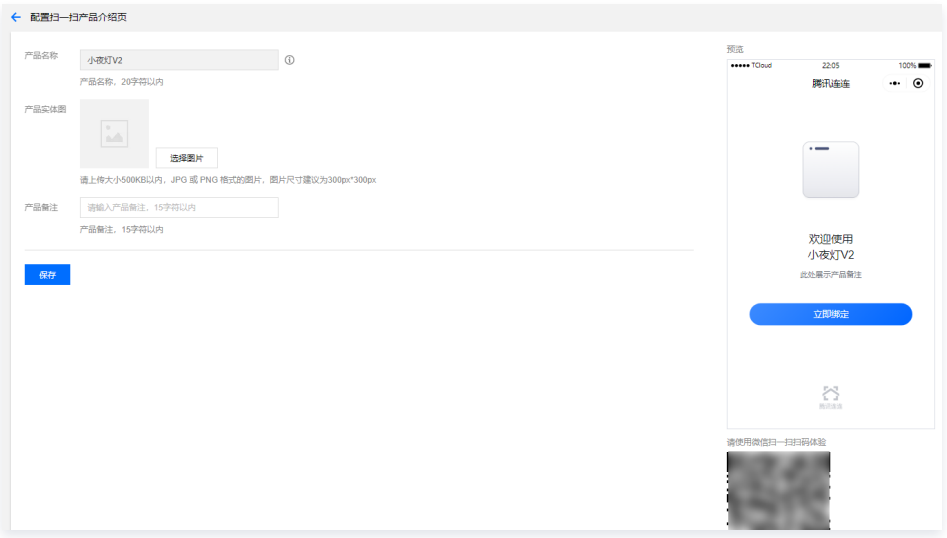
3. 单击**保存**，保存当前的面板配置。
   - 预览：对面板的配置，可在右方的预览中进行实时性的预览，查看配置的效果。
   - 扫码体验：您可以使用微信或“腾讯连连小程序”扫码体验，在小程序端查看面板配置效果。



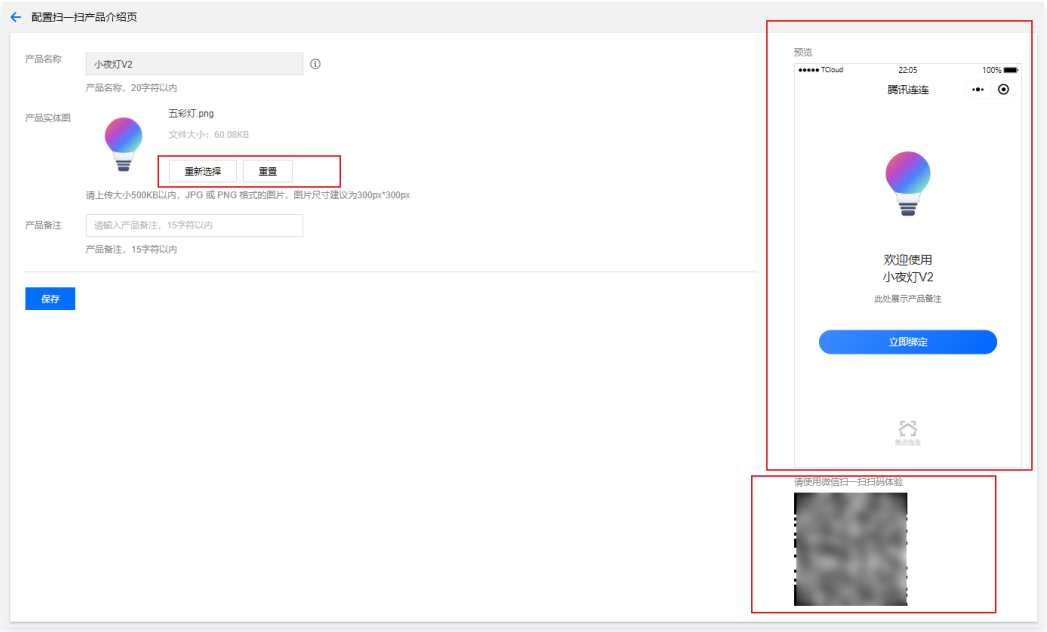
扫一扫产品介绍

通过扫一扫产品介绍页配置，您可以自定义用户在使用微信扫一扫添加设备时的产品介绍页，充分体现产品的特性，引导用户完成设备的添加，从而提升用户的产品体验。

1. 单击扫一扫产品介绍中的**配置**，进入扫一扫产品介绍配置页面。



2. 您可以自定义扫一扫产品介绍页的产品名称、产品实体图及产品备注内容。



- 产品名称：默认使用创建产品时的产品名称，如果需要修改产品名称，请前往该产品的数据模板页面处修改。
- 产品实体图：您可以根据图片尺寸建议格式上传产品的实体图，若不符合要求将无法上传。
- 产品备注：您可以自定义产品备注内容，如产品的简要介绍、绑定设备需注意的事项等。

3. 产品介绍页配置完成后，单击**保存**，保存当前扫一扫产品介绍页配置信息后，即可使用微信扫一扫扫码体验。

配网引导

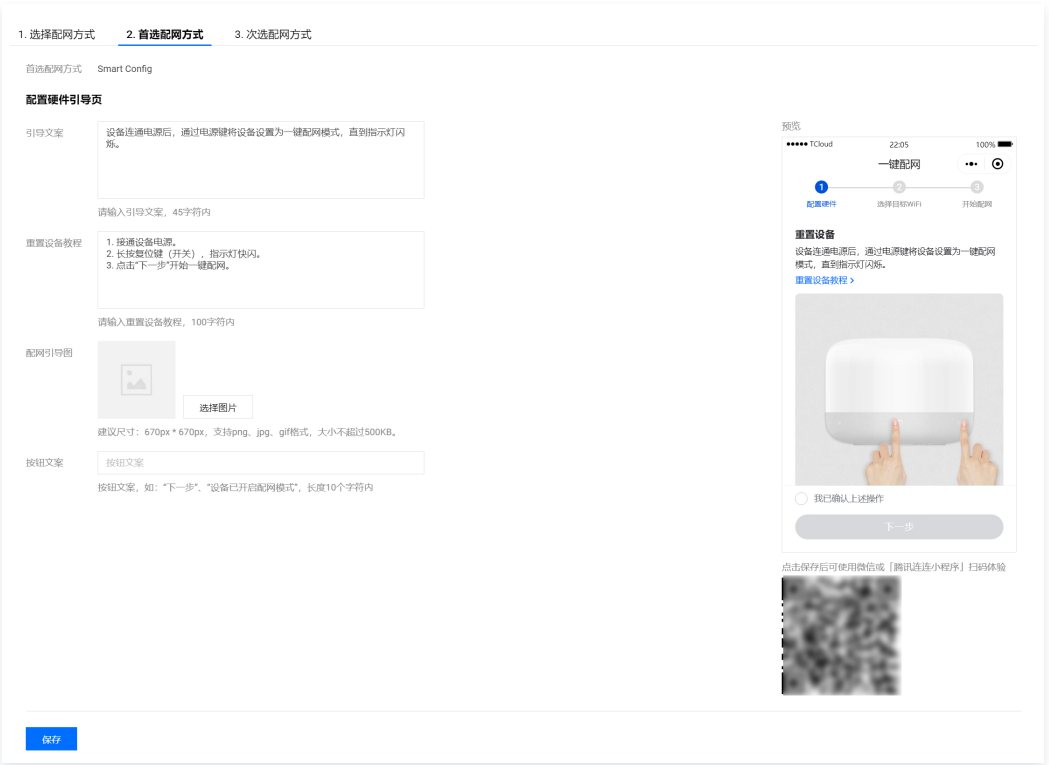
通过配网引导页配置，您可以自定义用户在进行设备配网流程时的配网交互页面，使得配网的引导更加符合产品的实际情况。

1. 单击配网引导栏中的**配置**。
2. 进入选择配网方式页。首先选择设备实际使用的芯片类型，因为不同芯片方案所支持的配网方式不同。然后选择您希望定义的首选配网方式和次选配网方式，确认配网方式后，单击**保存**即可生效。

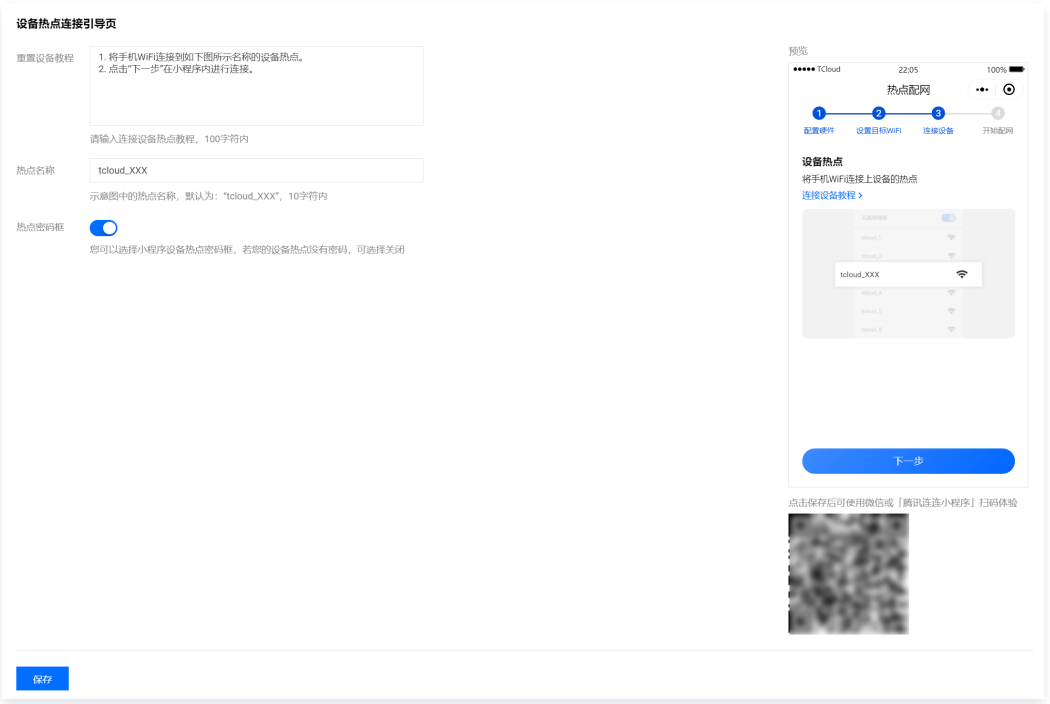
说明：

- **首选配网方式：腾讯连连移动端服务默认的配网方式。**
- **次选配网方式：当首选配网方式未成功时可切换使用的配网方式。**

3. 进入首选配网方式或选配网方式的配置硬件引导页，支持用户自行配置相应的配网图文信息，可在右方的预览中对配置进行实时性的预览，查看配置的效果。



4. 当选择配网模式为 Soft Ap 时，还可对设备热点连接引导页的文案进行配置。

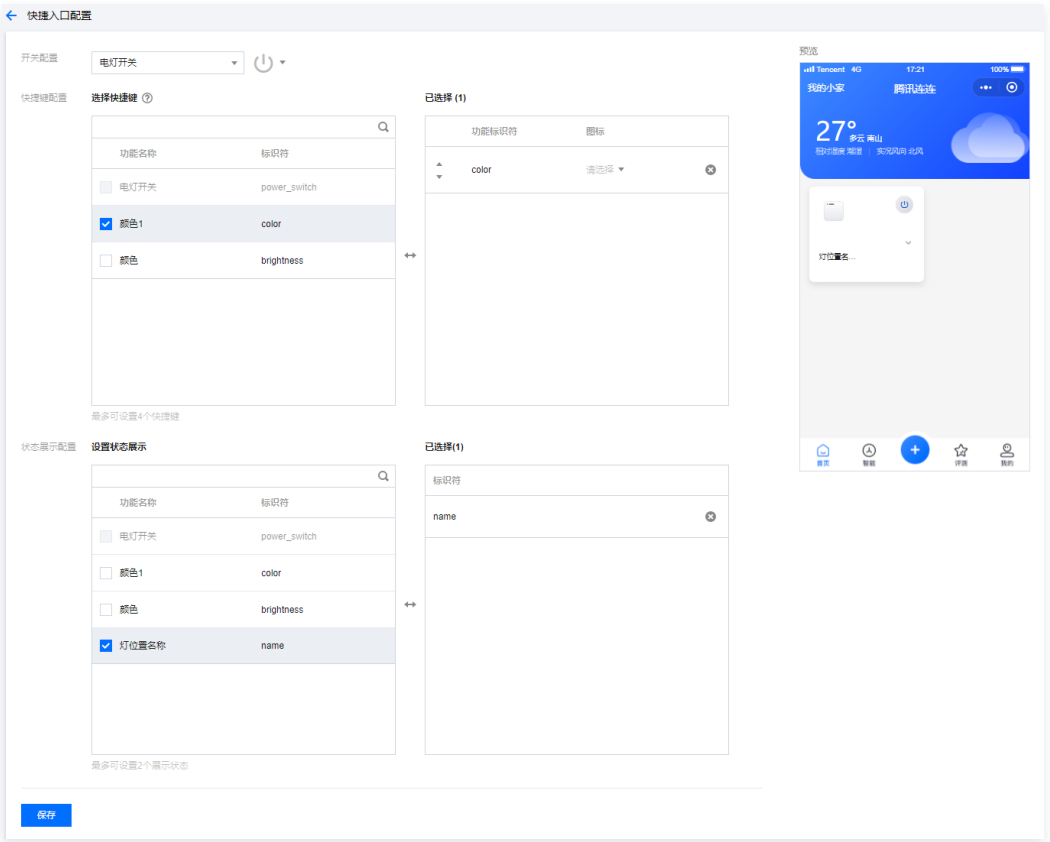


5. 配网引导页面配置完成后，单击**保存**按钮，保存当前的配网引导页面配置信息，即可使用微信或“腾讯连连小程序”扫码体验。

快捷入口配置

通过快捷入口配置，您可以自定义该产品在官方小程序的快捷操作功能，用户可在设备列表页进行快捷控制，无需进入设备操控面板内进行控制。

1. 单击快捷入口配置栏中的**配置**。
2. 进入快捷入口配置页后，您可以自行配置快捷操作的功能。



- 支持设置一个布尔型功能按钮，在产品列表直接对该功能进行控制。
- 快捷键配置支持选择四个功能，作为快捷操作按钮，可以设置按钮的图标与顺序。

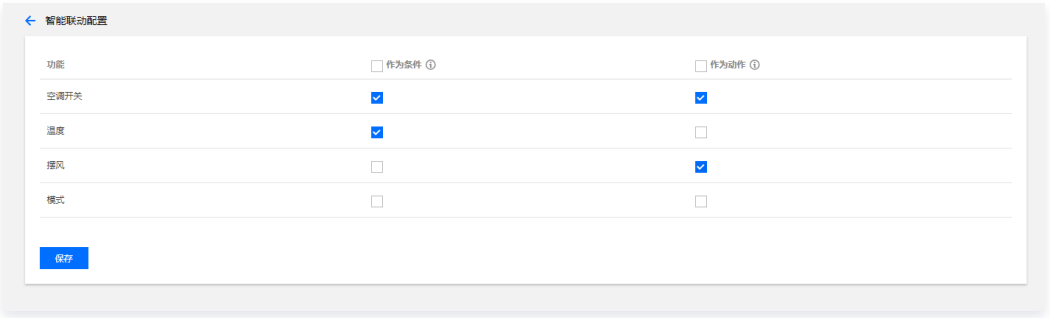
- 展示状态最多可设置2个。

3. 预览：对于快捷入口的配置，可在右方的预览中进行实时性的预览，查看配置的效果。
4. 单击**保存**按钮，保存当前的快捷入口配置。

智能联动配置

通过智能联动配置，您可以自定义用户在小程序添加智能时，该产品可作为条件或任务的功能属性。

1. 单击智能联动配置栏中的**配置**。
2. 进入智能联动配置页面后，您可以通过勾选对应功能自定义该产品可作为条件或任务的功能属性。



- 作为条件：选择某功能作为条件后，该功能将会在腾讯连连-智能页面添加智能时，作为自动智能的触发条件让用户设置。
- 作为动作：选择某功能作为任务后，该功能将会在腾讯连连-智能页面添加智能时，作为手动智能和自动智能的设备触发任务让用户设置。

3. 单击**保存**按钮，保存当前的智能联动配置。

设备调试

最近更新时间：2024-09-30 17:54:51

操作场景

设备开发完成后，需要进入设备调试阶段，调试设备与云端的通信是否正常。设备调试提供了真实设备在线调试及虚拟设备调试，并可通过控制台查询设备上报的当前数据、历史通信日志、事件及上下线记录等。本文档主要介绍如何进行设备调试。

操作步骤

新建设备

1. 设备开发完成后，单击**设备调试**，进入设备调试环节。
2. 选择**新建设备**，如下图输入设备名，单击**保存**，即可创建设备。

3. 创建成功后，将会在“设备调试”列表页中查看到新建成功的设备。

查看设备信息

1. 新建设备成功后，需要查询设备信息，获取重要参数及进行设备调试。
2. 单击设备列表的设备名称，即可查看设备的信息及设备上报云端的状态、上下行日志等信息。
3. 在产品开发中，单击**二维码**可以快速绑定真实设备，帮助开发者降低开发难度，量产后为了安全性，将会关闭二维码入口。

4. 设备查看打开后，即可查看设备的所有信息。
   - 设备密钥：使用密钥认证需要将此信息烧录到设备端。
   - 产品 ID：唯一标识，需要烧录到设备端。
   - 激活时间：设备第一次连接开发平台的时间。
   - 最后上线时间：设备最后一次连接开发平台的时间。
   - 设备状态：如果设备在线，则显示“在线”，如果设备离线，则显示“离线”，如果设备从未连接开发平台，则显示“未激活”。

设备属性

1. 查看设备信息：单击查看设备信息。
2. 列表中将该设备的数据模板的功能项列出。
   - 变量标识符：对应该设备的数据模板中的标识符。
   - 变量名称：对应数据模板中的“功能名称”。
   - 历史数据：单击**查看**，即可查询该功能项的历史上报数据。
   - 变量类型：对应数据模板中的“数据类型”。
   - 最新值：当设备在向云端上报数据时，只要某个功能的最新上报值发生变化，最新值列都会立刻显示设备上报的最新值。
   - 更新时间：指最新值的变化时间。一般是设备上报该功能的发生时间。
3. 查看某个功能的历史上报数据。按时间展示该功能上报到云端的历史数据，验证上报的数据是否正确。

设备日志

单击**设备日志**，即可查看该设备上行到云端，并从云端接收的信息。

- 上行：上行表示设备端向云端上报的数据。
- 下行：下行表示云端向设备端发送的数据。

设备事件

单击**设备事件**，即可查看该设备上报到云端的事件信息。

- 事件的定义：在数据模板中定义管理。
- 事件类型：系统将事件类型分为三种，分别是告警、故障、信息。

设备行为

单击**设备行为**，即可查看该设备的行为信息。

- 行为的定义：在数据模板中的管理。
- 行为描述：用于描述复杂的业务逻辑，可添加多个调用参数和返回参数，可用于让设备执行某项特定的任务，例如，开锁动作需要知道是哪个用户在什么时间开锁，锁的状态如何等。

设备上下线日志

单击**设备上下线日志**，即可查看该设备连接云端与断开连接的日志记录。

真实设备在线调试

1. 当您的真实设备已成功对接到开发平台后，则可以使用在线调试对真实设备进行数据收发的测试。
2. 单击**在线调试**，即可进入在线调试功能。前提是真实设备已开启并成功连接到开发平台。
3. 在线调试左侧的操控面板是根据设备所属产品的数据模板自动生成，设置需要下发的数据后，单击**发送**后，系统会自动触发控制指令到设备端。
4. 设备端接收到指令后，会立刻返回数据到云端并显示在右侧的文本框中。

云端诊断日志错误信息

最近更新时间：2022-06-10 14:08:32

概念介绍

云端诊断日志功能用于查看设备与云端交互的通信交互日志，帮助用户快速诊断在调试过程中设备出现的异常错误，如订阅 topic 无权限、上行消息发布失败、规则引擎转发第三方服务失败等错误异常事件定位。
该文档用于在与云端消息通信错误时，根据错误内容定位原因，并寻求解决方案。

消息错误日志

设备上下线相关

content	result	errcode	描述
Device connect	FAIL, system error	Dev_Conn_System_Err	设备上线失败，系统错误。

发布、订阅 Topic 相关

content	result	errcode	描述
Device subscribe topic: {}	FAIL, unauthorized operation	Subscribe_Topic_Unauthorized	订阅失败，无 topic 订阅权限。
Device subscribe topic: {}	FAIL, system error	Subscribe_System_Err	订阅失败，系统错误。
Device unsubscribe topic: {}	FAIL, system error	UnSubscribe_System_Err	退订失败，系统错误。

设备消息相关

content	result	errcode	errcode 释义
Device publish message to topic:{}, QOS:{}	FAIL, unauthorized operation	Dev_Pub_Unauthorized	发布失败，无 Topic 发布权限。
	FAIL, reach max limit with {}	Dev_Pub_Reach_Max_Limit	发布失败，publish 超过频率限制。
	FAIL, payload too long({} > {})	Dev_Pub_Payload_TooLong	发布失败，payload 超过长度限制。
	FAIL, system error	Dev_Pub_System_Err	发布失败，系统错误。
Publish message to device: topic: {}, QOS:{}{，业务错误说明}	FAIL,no subcriber	Pub_To_Dev_No_Subscriber	发送失败，没有订阅者。
	FAIL, too many offline message	Pub_To_Dev_Offline_Msg_Exceed_Limit	发送失败，离线消息存储满。
	FAIL, offline message payload exceed limit	Pub_To_Dev_Payload_Too_Large	发送失败，payload 大小超过限制。
	FAIL, system error	Pub_To_Dev_System_Err	发送失败，系统错误。

规则引擎相关

content	result	errcode	errcode释义
Send message to RuleEngine, topic:{}	FAIL, system error	Msg_Send_To_Rule_System_Err	转发失败，系统错误。
MQ:forward CMQ, type:{}, CMQ{queue: {}, region: {}}	FAIL, queue name is not existed, or deleted	MQ_Queue_NotExist_Or_Deleted	转发失败，队列不存在，或者队列已经被删

			除了。
	FAIL, exceed maximum message size	MQ_Exceed_Max_Msg_Size	转发失败，存在至少一条消息达到了最大消息大小限制。
	FAIL, reach maximum retention number of message	MQ_Reach_Max_Retention_Num	转发失败，达到队列的最大消息堆积数。
	FAIL, unexpected error: {}	MQ_Forward_CMQ_Unexpected_Err	转发失败，腾讯云接口公共错误。
	FAIL, system error	MQ_Forward_CMQ_System_Err	转发失败，系统错误。
MQ forward CKafka, type:{}, Ckafka{instance:{}, topic: {}, region: {}}	FAIL, unexpected error: {}	MQ_Fowward_Kafka_Unexpected_Err	转发失败，Kafka 错误信息。
	FAIL, system error	MQ_Fowward_Kafka_System_Err	转发失败，系统错误。
RuleEngine republish message, source topic:{}, destination topic: {}	FAIL, no such field({}) in payload	Payload_No_Field	转发失败，payload 没有对应的字段。
	FAIL, system error	Rule_Repub_System_Err	转发失败，系统错误。
RuleEngine forward third-party server,topic:{}, url:{}	FAIL, url server timeout	Forward_Third_Not_Responding	转发失败，第三方服务器无响应。
	FAIL, system error	Forward_Third_System_Err	转发失败，系统错误。
RuleEngine forward CKafka, topic:{}, Ckafka.instance:{}, topic: {}, region: {},retry times: {}"	FAIL,unexpected error: {}	Rule_Forward_Kafka_Unexpected_Err	转发失败，Kafka 错误信息。
RuleEngine forward CMQ Topic, IOT topic:{}, CMQ.topic: {}, region:{}	FAIL,system error:{}	MQ_Queue_NotExist_Or_Deleted	转发失败，队列不存在，或者已经被删除了。
		MQ_Exceed_Max_Msg_Size	转发失败，消息超过最大值限制。
		MQ_Reach_Max_Retention_Num	转发失败，达到最大保持数目。
		Rule_Forward_CMQ_Topic_No_Subscription	转发失败，本主题没有订阅者。
		Rule_Forward_CMQ_Topic_Unexpected_Err	转发失败，未知错误。
RuleEngine forward CMQ Queue, IOT topic:{}, CMQ.topic: {}, region:{}	FAIL,system error:{}	MQ_Queue_NotExist_Or_Deleted	转发失败，队列不存在，或者已经被删除了。
		MQ_Exceed_Max_Msg_Size	转发失败，消息超过最大值限制。
		MQ_Reach_Max_Retention_Num	转发失败，达到最大保持数目。
		Rule_Forward_CMQ_Topic_No_Subscription	转发失败，本主题没有订阅者。
		Rule_Forward_CMQ_Topic_Unexpected_Err	转发失败，未知错误。

RuleEngine forward CTSDB. topic:{}, CTSDB.instanceid:{}, region:{}, metric:{}	FAIL,system error:{}	Rule_Forward_Ctsdb_System_Err	转发失败，系统错误。
RuleEngine forward Mongo, topic:{}, Mongo.instanceid:{}, database:{}, collection:{}	FAIL,system error:{}	Rule_Forward_MongoDB_System_Err	转发失败，系统错误。
RuleEngine forward TCB Func,get role error topic:{}, TCB.envID:{}, functionName:{}	FAIL,system error:{}	Rule_Forward_TCB_Func_System_err	转发失败，系统错误。
RuleEngine forward Mysql.IOT topic:{}. info:{}	FAIL,system error:{}	Rule_Forward_Mysql_System_Err	转发失败，系统错误。
RuleEngine forward TDMQ.IOT topic:{}. info:{}	FAIL,system error:{}	Rule_Forward_TDMQ_System_Err	转发失败，系统错误。
Process sql: {}	FAIL,unexpected error:{}	Rule_Process_SQL_Err	执行失败。
Payload is not JSON fmt	FAIL	Payload_Not_JSON	消息负载不是 JSON 格式。

批量投产

最近更新时间：2022-06-10 14:08:42

操作场景

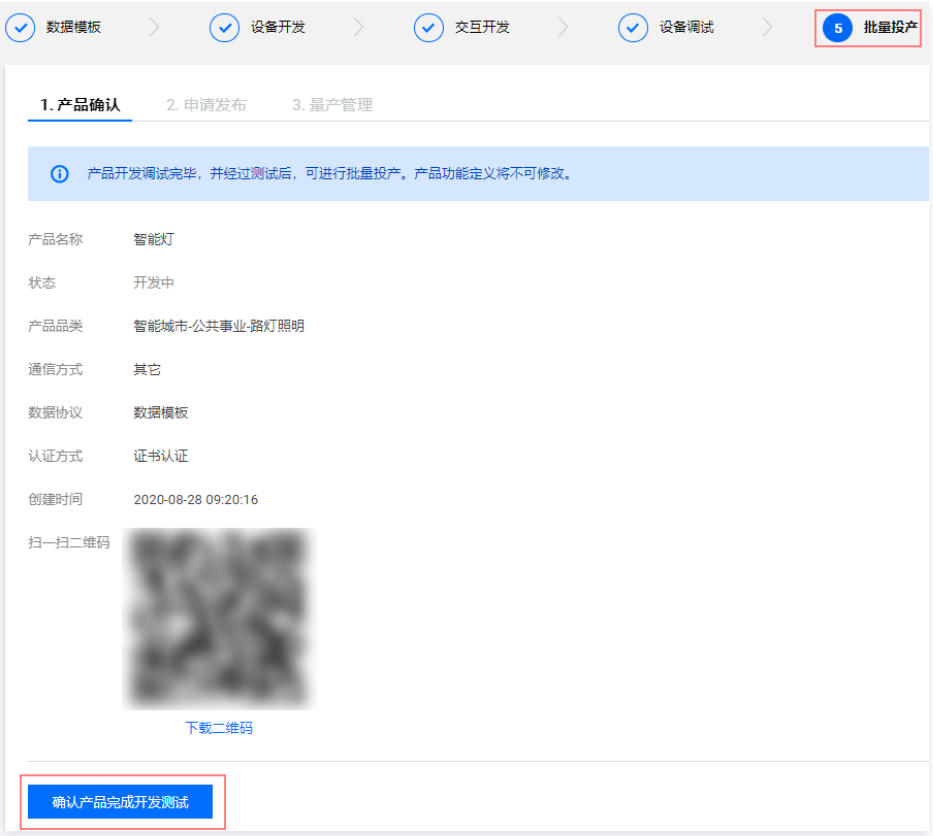
设备调试完成，并经过必要的测试后，即可进入投产阶段，开发平台会将申请发布后产品的数据模板设置为不可修改状态，防止在开发平台所做的修改导致实际设备运行出现问题。本文档主要介绍如何进行批量投产。

前提条件

设备调试完成，并经过小批量设备的测试后，用户即可进入批量投产环节。

操作步骤

1. 登录 物联网开发平台控制台，选择已创建的项目进入，单击左侧菜单栏**产品开发**进入产品列表页。
2. 选择对应的产品单击进入产品详情页，单击**批量投产**，进入批量投产环节。
3. 单击**确认产品完成开发测试**，进入申请发布环节。

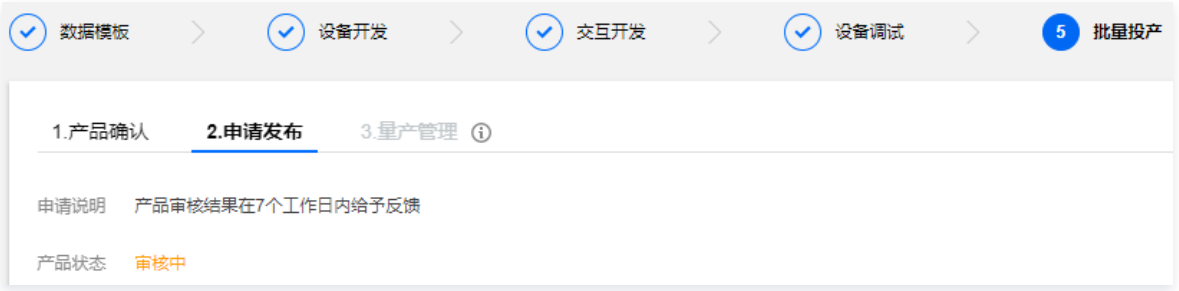


4. 在申请发布环节，需要输入相关产品信息进行申请。
   - **厂家名称：**产品厂家的官方名称。
   - **产品型号：**输入产品的具体型号。
   - **选择照片：**上传一张产品的真实图片。
   - **上传测试报告：** 按要求编写测试报告并上传。

注意：

当申请发布后，该产品的数据模板将不允许再修改。如果您的产品在市场上获取反馈后，因业务需要必须要增加新的功能，则可通过创建一个新的产品来解决。

5. 单击**申请发布**，则产品的状态修改为“审核中”。（审核未通过时，将回到“开发中”状态，您可以重新申请发布。）



6. 若您的产品已开发完成并通过审核，状态将变为“已发布”，您可以在量产管理内进行投产。

产品共享

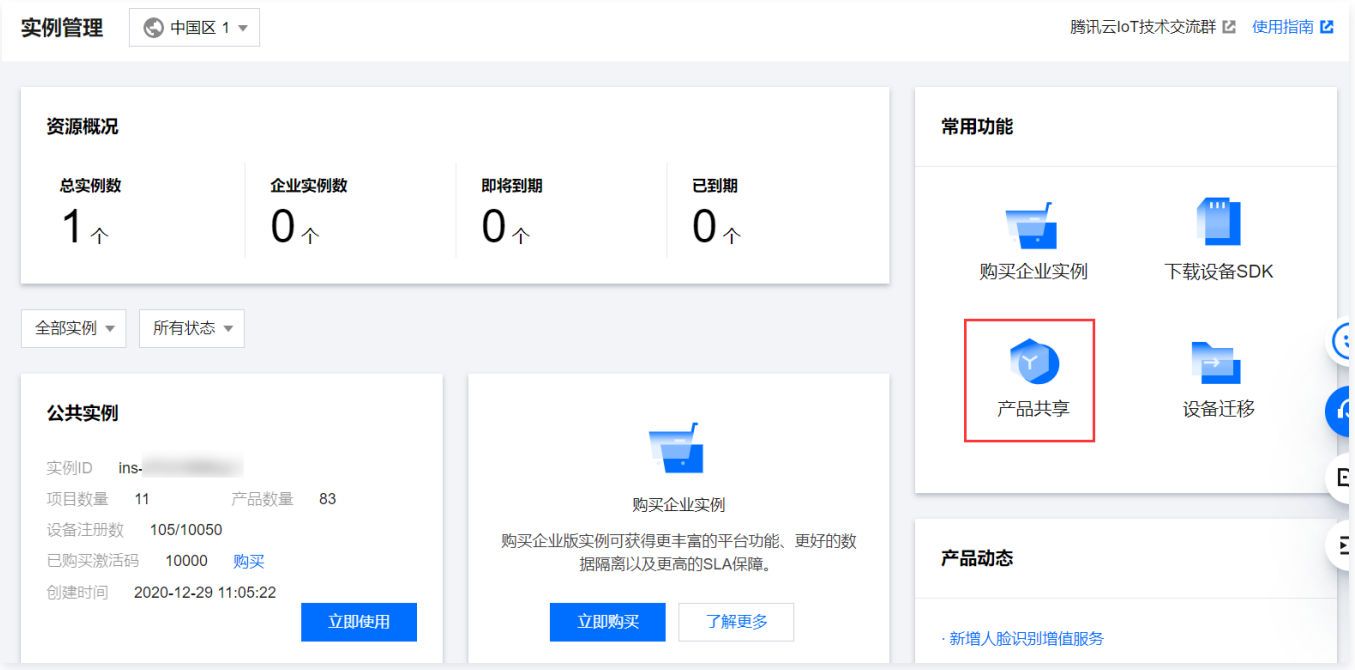
最近更新时间：2024-09-30 17:54:51

应用场景

产品共享功能用于账号间共享产品，方便自主小程序(或APP)开发者跨账号绑定其他账号下创建的设备，与合作伙伴构建生态联盟。
本文档为您介绍产品共享功能的使用方法，帮助您快速使用此功能。

操作步骤

1. 进入实例管理页面，单击**产品共享**按钮：



2. 自主小程序开发者，可以通过**申请共享**向设备制造商申请产品共享：

3. 自主小程序开发者申请产品共享后，设备制造商会接收**审批任务**，审批通过的产品才能被共享出去：

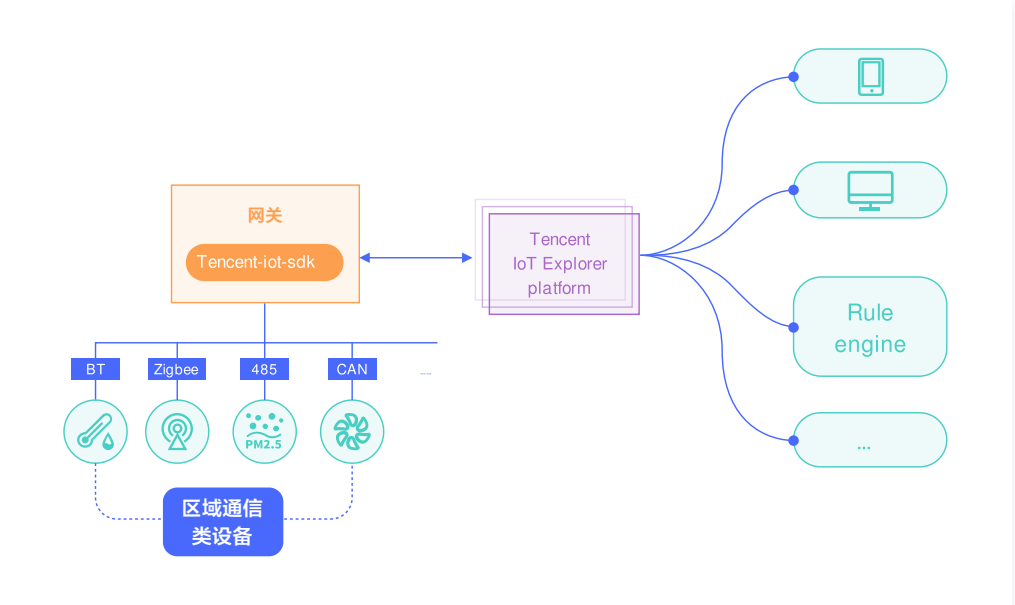
4. **自主小程序开发者和设备制造商都可以解除共享产品，产品解除共享后，自主小程序就无法再跨账号绑定此产品。**
5. **产品共享成功后，在自主小程序关联产品页面中，会显示跨账号产品列表，您可以根据您的自主小程序需要，是否关联此产品，关联后，您的应用就可以绑定此产品下的设备：**

网关设备接入
网关子设备产品管理

最近更新时间：2022-06-10 14:09:03

操作场景

对于 BLE、Zigbee 和485等不具备直接访问网络能力的设备，需要先接入网关，然后通过网关代理，间接实现设备接入腾讯物联网开发平台 IoT Explorer，具体流程框架图如下：



本文档将指导您如何在平台管理网关与子产品的关联绑定关系以及网关设备与子设备的设备级拓扑关系。

操作步骤

网关下子产品管理

创建网关产品

1. 登录 物联网开发平台控制台，单击某个实例，进入项目列表页。
2. 选择某个项目进入详情页，单击**新建产品**。

3. 进入新建产品页面，填写相关信息。

- 产品名称：名称为中文、字母、数字、下划线的组合，1位 – 20位字符且不能为空。
- 产品类型：选择您所创建产品的所属品类，不同类型产品的数据模板会有所不同。也可以选择自定义产品的数据模板。
- 设备类型：选择网关类型。
- 认证方式：物联网开发平台提供两种认证方式用于设备与平台之间鉴权认证。可以根据需要进行选择证书认证或密钥认证。
- 通信方式：您可以选择 Wi-Fi、移动蜂窝（2G/3G/4G）等其他通信方式。
- 数据协议：目前网关类型产品仅支持数据模板协议。
- 描述：字数不能超过80个，您可以根据需要选填。

4. 信息填写完成后，单击**保存**。

创建子产品

单击**新建产品**按钮，设备类型选择子设备，本示例创建子产品：产品品类为灯、密钥认证、与网关通信方式为自定义协议、数据协议选择为物模型。

绑定子产品

> **说明：**
> 网关产品需要与子产品建立绑定后，网关设备才能够与子设备建立绑定并代理通信。

1. 选择产品列表下网关产品的**子产品管理**，进入子产品绑定页面。
2. 单击**添加子产品**，勾选您需要绑定的产品后，单击**添加**，完成绑定。

解绑子产品

解绑子产品需要先解绑网关设备下的子产品设备，解绑子产品之后，网关下对应子产品的权限也将被清除。

1. 网关不再需要绑定的子产品，您可以在**子产品管理**页面进行解绑。
2. 勾选对应子产品，单击**批量解绑**，即可完成解绑子产品。

子设备管理

添加网关设备

1. 单击产品列表对应的网关产品，进入产品开发页面。
2. 单击第4步进入设备调试，为网关产品创建设备。

3. 单击**新建设备**按钮，创建网关设备 gateway001。

添加子设备

在子产品的设备调试页面，创建一个子设备 sub_dev001。

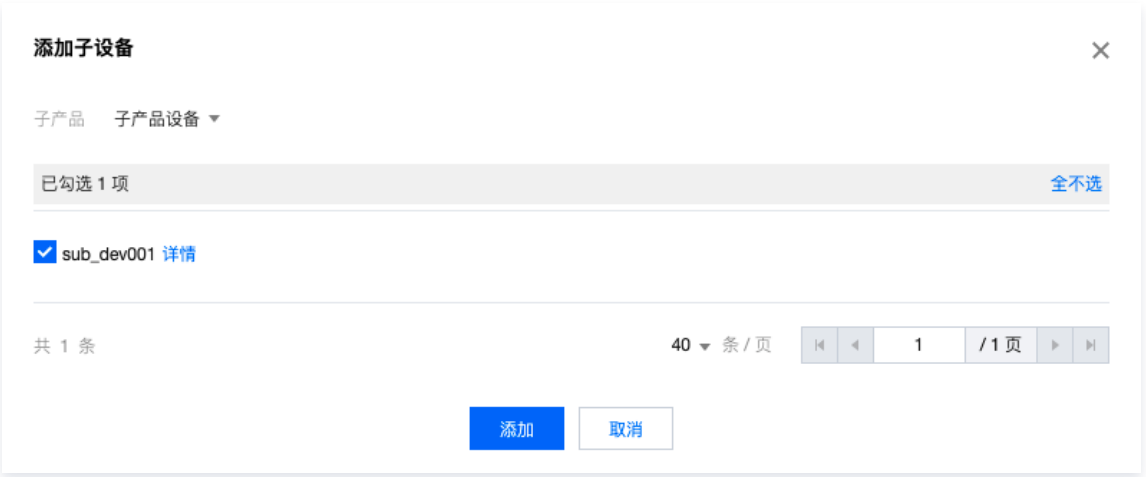
绑定子设备

说明：

- 子设备是通过网关产品代理实现子设备的鉴权和通信，网关能够代理子设备鉴权和通信的前提是在平台实现了网关和子设备的绑定操作。
- 子设备是无法直连物联网平台的产品，由网关设备代理连接，所以子设备的认证方式不影响连接，由网关设备负责认证接入。

1. 选择对应的网关设备，单击**子设备管理**添加子设备。

2. 勾选需要绑定的子设备，单击**添加**，完成绑定。



解绑子设备

选择产品列表下**设备列表**，勾选对应子设备，单击**解绑**进行解绑操作。

后续步骤

网关子设备快速入门

结合 C-SDK 的 gateway_sample 快速体验网关设备代理子设备上下线，子设备基于数据模板协议发送和接收消息。同时如何进行小程序端开发，及小程序侧添加网关子设备交互流程。请参考文档 网关子设备快速入门 。

网关子设备快速入门

最近更新时间：2024-08-26 14:55:51

操作场景

若需要将网关与子设备类型设备接入到物联网开发平台，网关设备直连云平台，子设备通过网关代理绑定、上下线、通信的方式与平台进行通信，从而实时控制网关与子设备。本文档主要帮助您快速熟悉网关与子设备的对接全流程。

前提条件

为了通过下面的步骤快速理解网关子设备类型设备的对接流程，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、运行示例程序。
- 示例程序在 Linux 环境下测试和验证，主要基于 Ubuntu16.04 版本，gcc-5.4（建议至少 gcc-4.7+）。

操作步骤

步骤一：控制台操作

创建网关子设备产品，并将子设备绑定到网关设备下：

该部分操作可参考 网关子设备产品 。

步骤二：运行设备端程序

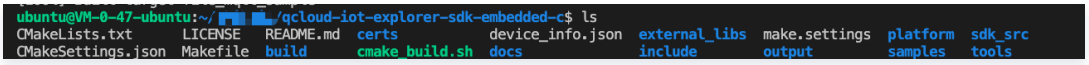
2.1 下载 demo 程序

从 GitHub 下载，或执行下面的 git 命令。

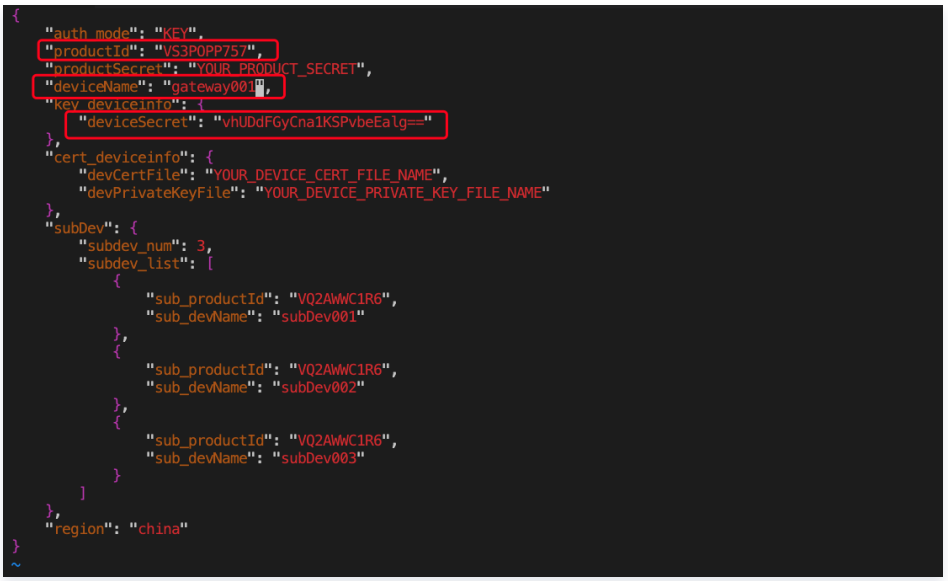
```
git clone https://github.com/tencentyun/qcloud-iot-explorer-sdk-embedded-c.git
```

2.2 代码编译

1. 进入下载的 demo 程序目录下。



2. 在 `device_info.json` 中填入网关的设备信息，包括： `productId` (对应着控制台的产品ID)、 `deviceName` (对应着控制台的设备名称)、 `key_deviceinfo->deviceSecret` (对应着控制台的设备密钥)。



3. 在CMakeLists.txt中把FEATURE_GATEWAY_ENABLED这个参数设置为ON，然后执行 `./cmake_build.sh` ，即可在 `output/release/bin` 下面生成 `gateway_sim_sample` 和 `subdev_sim_sample` 。

2.3 运行程序

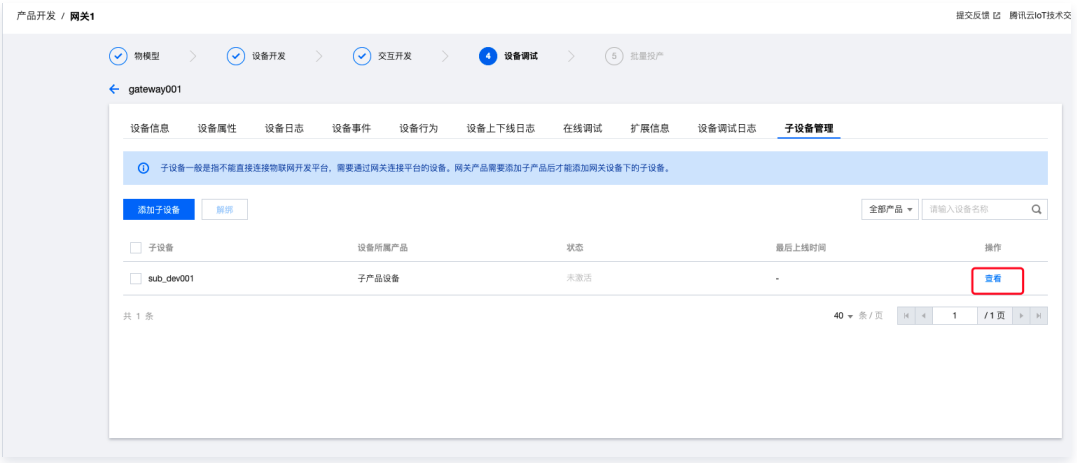
1. 运行 gateway_sim_sample:

```
./output/release/bin/gateway_sim_sample
```

日志输出应该如下：

```
INF|2021-07-06 15:42:58|qcloud_iot_device.c|iot_device_info_set(56): SDK_Ver: 3.1.6, Product_ID:
XKVP9QORAR, Device_Name: test001
DBG|2021-07-06 15:42:58|HAL_TLS_mbedtls.c|HAL_TLS_Connect(224): Setting up the SSL/TLS structure...
DBG|2021-07-06 15:42:58|HAL_TLS_mbedtls.c|HAL_TLS_Connect(266): Performing the SSL/TLS handshake...
DBG|2021-07-06 15:42:58|HAL_TLS_mbedtls.c|HAL_TLS_Connect(267): Connecting to
/XKVP9QORAR.iotcloud.tencentdevices.com/8883...
DBG|2021-07-06 15:42:58|HAL_TLS_mbedtls.c|HAL_TLS_Connect(289): connected with
/XKVP9QORAR.iotcloud.tencentdevices.com/8883...
INF|2021-07-06 15:42:58|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: yfSdT success   //
网关连接平台成功
DBG|2021-07-06 15:42:58|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(147):
topicName=$gateway/operation/result/XKVP9QORAR/test001|packet_id=4444 // 订阅网关拓扑关系管理topic
DBG|2021-07-06 15:42:58|gateway_api.c|_gateway_event_handler(80): gateway sub|unsub(3) success, packet-
id=4444 // 订阅成功
```

2. 获取子设备三元组信息,进入网关设备的子设备管理界面，单击**查看**按钮：



进入子设备 > 设备信息页面：

3. 在另外一个控制台内运行 `subdev_sim_sample` ，运行的格式为 `./output/release/bin/subdev_sim_sample -p ProductID -d DeviceName` ，运行之后，子设备会和 `gateway_sim_sample` 建立连接，并通知网关设备子设备上线，然后会循环上报属性。

> **说明：**

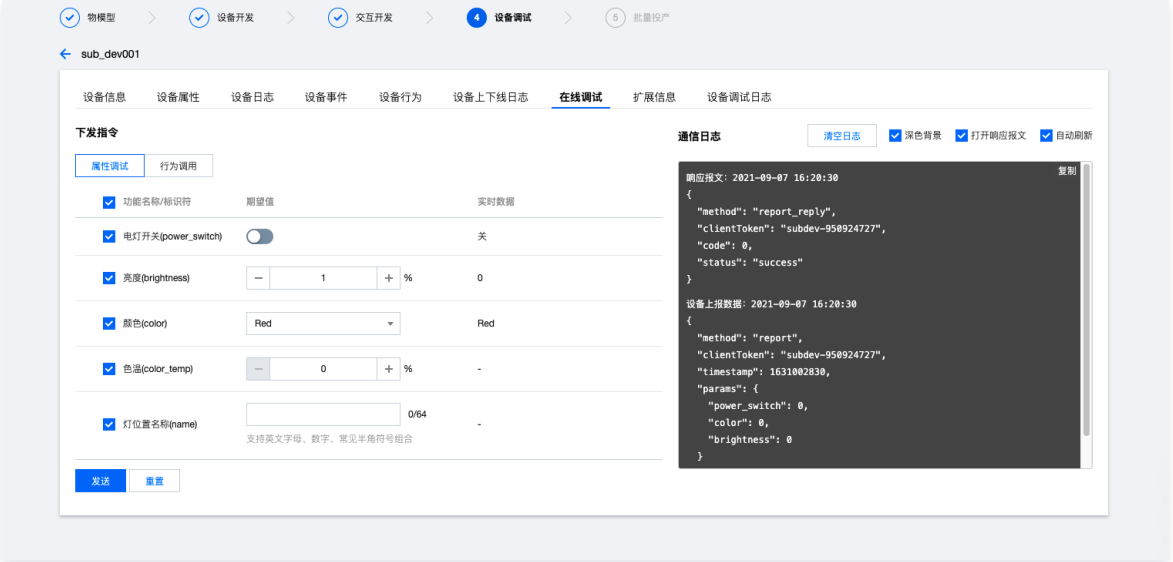
- ProductID：为子设备的产品ID。
- DeviceName：为子设备的设备名称。

子设备可运行多个，打开多个控制台，每个控制台运行1个，需要指定不同的 productId 或 deviceName。

子设备的日志如下：

```
DBG|2021-07-06 15:49:36|subdev_sim_sample.c|main(164): connect gateway OK, subdev is going to be online
// 通知网关设备子设备上线
DBG|2021-07-06 15:49:37|subdev_sim_sample.c|main(205): Report property {"subdev_type":
"report","product_id": "QYEWBRB1PS","device_name": "dev1","msg": "
{"method":"report","clientToken":"subdev-
1871908611","timestamp":1625557777,"power_switch":0,"color":0,"brightness":0}}  // 上报子设备属性到网关设备
DBG|2021-07-06 15:49:53|subdev_sim_sample.c|main(205): Report property {"subdev_type":
"report","product_id": "QYEWBRB1PS","device_name": "dev1","msg": "
{"method":"report","clientToken":"subdev-
1836984528","timestamp":1625557793,"power_switch":0,"color":0,"brightness":0}}
DBG|2021-07-06 15:49:53|subdev_sim_sample.c|main(192): recv
{"method":"report_reply","clientToken":"subdev-1836984528","code":0,"status":"success"} // 接收到平台返回
的上报成功
```

控制台在线调试日志：



网关设备的日志如下：

```
DBG|2021-07-06 15:46:45|gateway_sim_sample.c|_accept_new_subdev(339): Recv conn from 127.0.0.1 29285 //
发现新的子设备
DBG|2021-07-06 15:46:47|gateway_sim_sample.c|_deal_with_subdev_msg(382): Get msg
{"subdev_type":"online","product_id":"QYEWBRB1PS","device_name":"dev1"} from subdev    // 子设备上线
DBG|2021-07-06 15:46:47|gateway_sim_sample.c|_deal_with_subdev_msg(402): itype is 0, pid QYEWBRB1PS
dname dev1
DBG|2021-07-06 15:46:47|gateway_api.c|IOT_Gateway_Subdev_Online(191): there is no session, create a new
session    // 网关新增子设备管理session
DBG|2021-07-06 15:46:47|mqtt_client_publish.c|qcloud_iot_mqtt_publish(347): publish
packetID=0|topicName=$gateway/operation/XKVP9QORAR/test001|payload={"type":"online","payload":
{"devices":[{"product_id":"QYEWBRB1PS","device_name":"dev1"}]}} // 通知平台子设备在线
INF|2021-07-06 15:46:47|gateway_common.c|_gateway_message_handler(387): client_id(QYEWBRB1PS/dev1),
online result 0
DBG|2021-07-06 15:46:47|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(147):
topicName=$thing/down/property/QYEWBRB1PS/dev1|packet_id=4445 // 代理子设备订阅属性下行topic
DBG|2021-07-06 15:46:47|gateway_api.c|_gateway_event_handler(80): gateway sub|unsub(3) success, packet-
id=4445
```

```
INF|2021-07-06 15:46:47|gateway_sim_sample.c|_event_handler(94): subscribe success, packet-id=4445  //
订阅成功
DBG|2021-07-06 15:46:48|gateway_sim_sample.c|_deal_with_subdev_msg(421): sub subdev OK
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_deal_with_subdev_msg(382): Get msg {"subdev_type":
"report","product_id": "QYEWBRB1PS","device_name": "dev1","msg": "
{"method":"report","clientToken":"subdev-
1531564604","timestamp":1625557622,"power_switch":0,"color":0,"brightness":0}} from subdev // 收到子设备
的消息
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_deal_with_subdev_msg(402): itype is 2, pid QYEWBRB1PS
dname dev1
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_deal_with_subdev_msg(425): msg {"subdev_type":
"report","product_id": "QYEWBRB1PS","device_name": "dev1","msg": "
{"method":"report","clientToken":"subdev-
1531564604","timestamp":1625557622,"power_switch":0,"color":0,"brightness":0}}
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_property_topic_publish(246): pid QYEWBRB1PS dname dev1
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_property_topic_publish(259): topic
$thing/up/property/QYEWBRB1PS/dev1
DBG|2021-07-06 15:47:02|gateway_sim_sample.c|_property_topic_publish(260): publish msg
{"method":"report","clientToken":"subdev-1531564604","timestamp":1625557622,"params":
{"power_switch":0,"color":0,"brightness":0}} // 代理子设备上报属性
DBG|2021-07-06 15:47:02|mqtt_client_publish.c|qcloud_iot_mqtt_publish(347): publish
packetID=0|topicName=$thing/up/property/QYEWBRB1PS/dev1|payload=
{"method":"report","clientToken":"subdev-1531564604","timestamp":1625557622,"params":
{"power_switch":0,"color":0,"brightness":0}}
INF|2021-07-06 15:47:03|gateway_sim_sample.c|_message_handler(154): Receive Message With
topicName:$thing/down/property/QYEWBRB1PS/dev1, payload:{"method":"report_reply","clientToken":"subdev-
1531564604","code":0,"status":"success"} // 接收到平台返回的上报返回消息
```

2.4 设备控制

平台下发控制消息时，网关设备的日志如下：

```
INF|2021-07-06 15:51:02|gateway_sim_sample.c|_message_handler(154): Receive Message With
topicName:$thing/down/property/QYEWBRB1PS/dev1, payload:{"method":"control","clientToken":"clientToken-
2fd030d1-7f82-4708-ad5a-8905505ff197","params":{"power_switch":1,"color":0,"brightness":80}} // 收到子设备的
下行消息，透传给子设备
DBG|2021-07-06 15:51:03|gateway_sim_sample.c|_deal_with_subdev_msg(382): Get msg
{"method":"control_reply","clientToken":"clientToken-2fd030d1-7f82-4708-ad5a-8905505ff197","code":0} from
subdev   // 获取子设备的control回复消息
DBG|2021-07-06 15:51:03|gateway_sim_sample.c|_property_topic_publish(246): pid QYEWBRB1PS dname dev1
DBG|2021-07-06 15:51:03|gateway_sim_sample.c|_property_topic_publish(259): topic
$thing/up/property/QYEWBRB1PS/dev1
DBG|2021-07-06 15:51:03|gateway_sim_sample.c|_property_topic_publish(260): publish msg
{"method":"control_reply","clientToken":"clientToken-2fd030d1-7f82-4708-ad5a-8905505ff197","code":0}
DBG|2021-07-06 15:51:03|mqtt_client_publish.c|qcloud_iot_mqtt_publish(347): publish
packetID=0|topicName=$thing/up/property/QYEWBRB1PS/dev1|payload=
{"method":"control_reply","clientToken":"clientToken-2fd030d1-7f82-4708-ad5a-8905505ff197","code":0} // 代
理子设备回复control消息
```

子设备的日志如下：

```
DBG|2021-07-06 15:51:02|subdev_sim_sample.c|main(192): recv
{"method":"control","clientToken":"clientToken-2fd030d1-7f82-4708-ad5a-8905505ff197","params":
{"power_switch":1,"color":0,"brightness":80}}   // 从网关设备透传下来的控制消息
[  lighting  ]|[color: RED ]|[brightness:||||||||||||||||----]   // 执行控制操作
DBG|2021-07-06 15:51:07|subdev_sim_sample.c|main(205): Report property {"subdev_type":
"report","product_id": "QYEWBRB1PS","device_name": "dev1","msg": "
{"method":"report","clientToken":"subdev-
2058913913","timestamp":1625557867,"power_switch":1,"color":0,"brightness":80}} // 上报更新之后的属性
```

子设备在运行一段时间后，会离线，离线后网关设备会代理子设备向平台发送offline消息，网关设备的日志如下：

```
DBG|2021-07-06 16:01:30|gateway_sim_sample.c|_subdev_leave(371): unsubscribe
$thing/down/property/QYEWBRB1PS/dev1 return 10391 // 解订阅对应的子设备下行topic
DBG|2021-07-06 16:01:30|mqtt_client_publish.c|qcloud_iot_mqtt_publish(347): publish
packetID=0|topicName=$gateway/operation/XKVP9QORAR/test001|payload={"type":"offline","payload":{"devices":
[{"product_id":"QYEWBRB1PS","device_name":"dev1"}]}}    // 通知子设备离线
DBG|2021-07-06 16:01:30|gateway_api.c|_gateway_event_handler(80): gateway sub|unsub(6) success, packet-
id=10391 // 解订阅成功
INF|2021-07-06 16:01:30|gateway_sim_sample.c|_event_handler(109): unsubscribe success, packet-id=10391
```

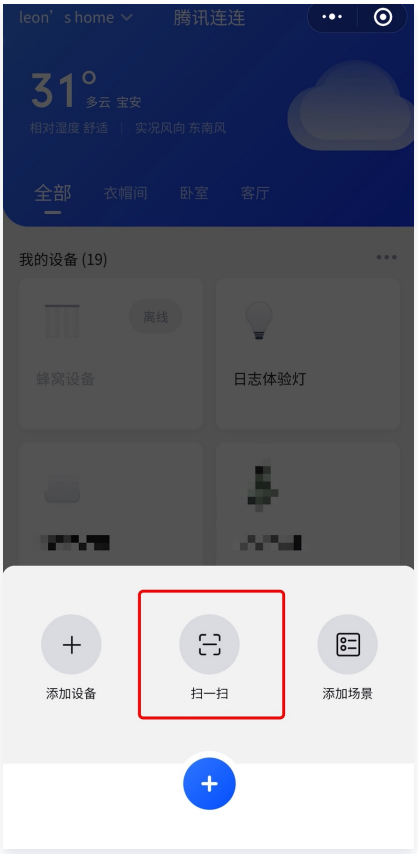
说明：

- 网关和子设备之间使用了本地 TCP socket 通信来模拟，在实际的场景下，可以换成 BLE 通信、ZigBee 通信等等。
- 子设备只演示了 Property 上行和下行操作，如果要使用 Event 和 Action，可以参考 light_data_template_sample.c 中的代码添加到网关设备和子设备的程序中，网关设备需要添加代理订阅对应下行 topic 的代码，子设备则需要添加上下行消息的处理代码。

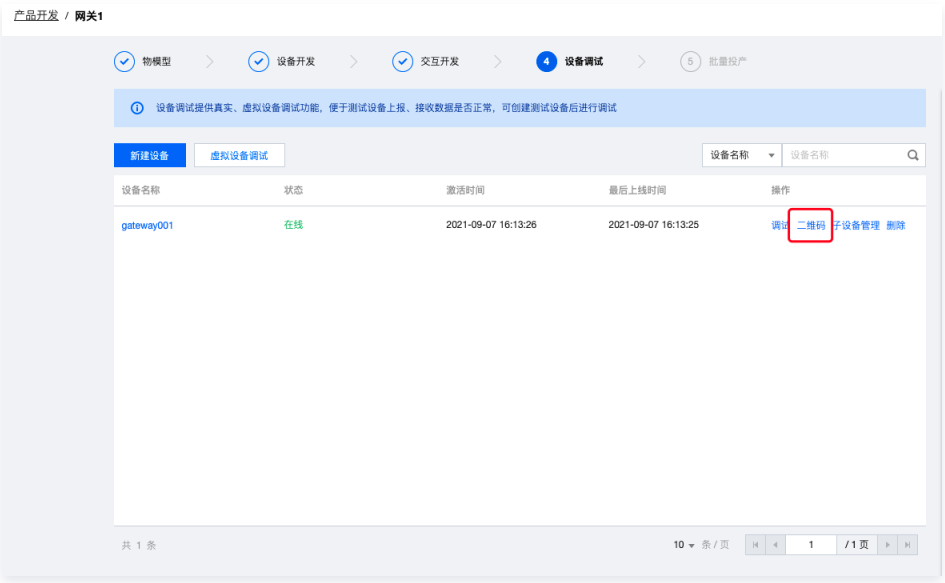
步骤三：小程序绑定、控制子设备

前提条件：前面已经成功运行了一个设备端的网关子设备示例程序，需要使用腾讯连连小程序添加、控制网关子设备。

1. 打开微信，搜索“腾讯连连”小程序，打开“扫一扫”。

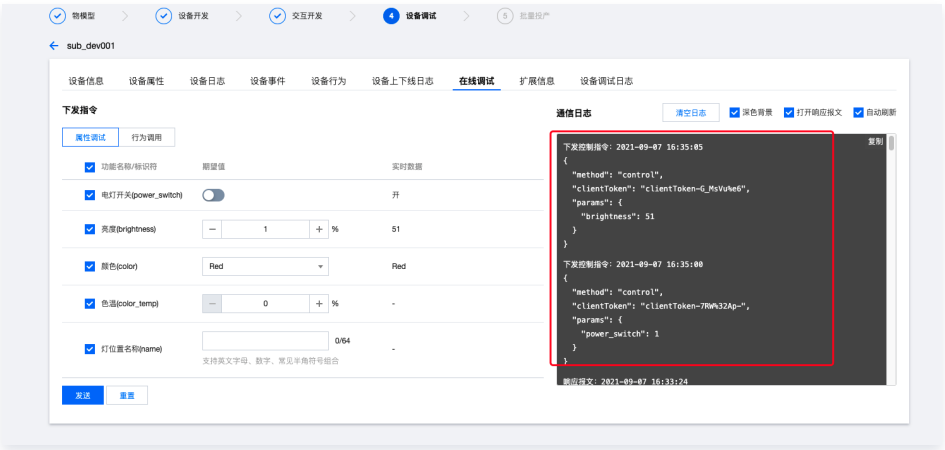


2. 进入网关设备的设备调试页面，单击设备二维码，用手机扫码添加该设备。



3. 可以看到网关和子设备均被添加到小程序家庭下：

4. 此时我们可以通过进入设备控制面板，控制子设备状态，同时可在云端查看设备日志：

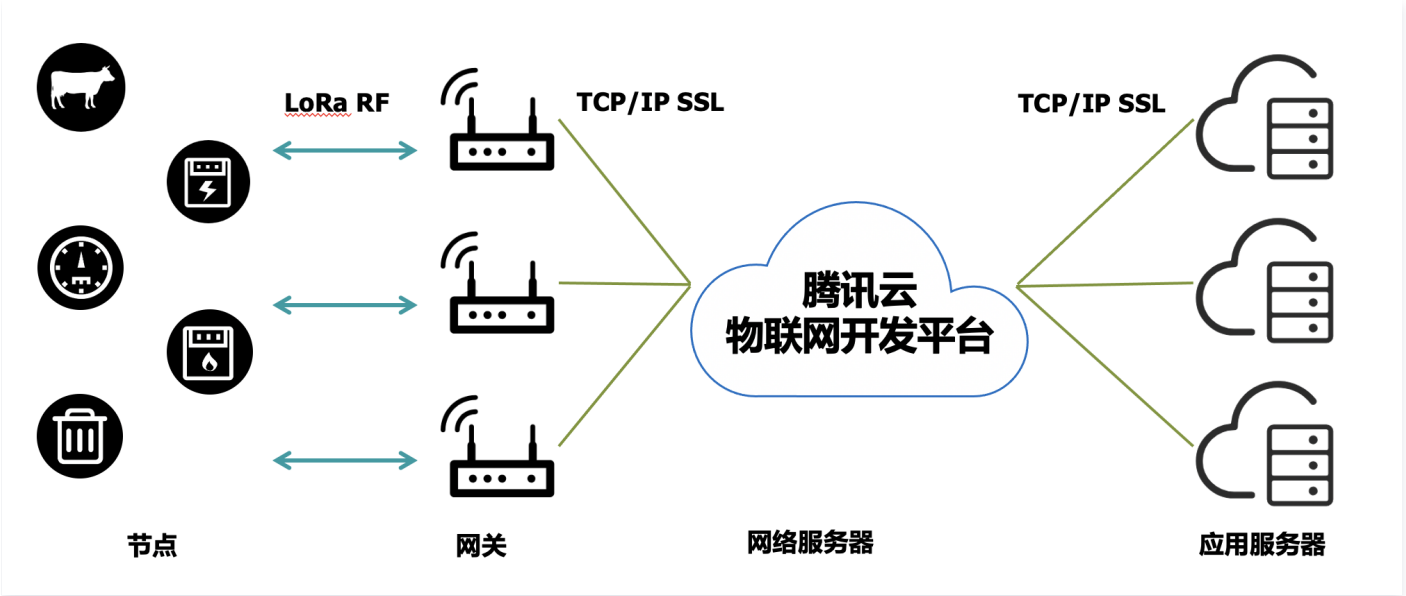


至此，我们就完成了网关子设备设备端 demo 程序的运行、云端调试、小程序控制的网关子设备入门体验。

LoRaWAN 产品开发
LoRaWAN 产品简介

最近更新时间：2022-06-10 14:09:41

网络架构



频谱策略

请参考工信部于2019年11月28日发布的微功率技术要求《微功率短距离无线电发射设备目录和技术要求》公告 第四节民用计量仪表的使用要求，具体要求如下：

（四）民用计量仪表
用于电力、热力、水务、燃气等公用计量业务。在建筑楼宇、住宅小区及村庄等小范围内组网应用，任意时刻限单个信道工作。
民用计量仪表设备应当具备“发射前搜寻”等干扰规避功能，且不能被用户调整或关闭。
若使用频率与当地声音、电视广播电台频率相同时，不得在当地使用；若对当地声音、电视广播接收产生干扰时，应立即停止使用，待消除干扰或调整到无干扰频率后方可重新使用。

1. 使用频率：470-510MHz。
2. 发射功率限值：50mW(e.r.p)。
3. 发射功率谱密度限值：占用带宽小于等于200kHz的,为50mW/200kHz(e.r.p)；占用带宽200-500kHz的，为10mW/100kHz(e.r.p)。
4. 单次发射持续时间：不超过1秒。
5. 占用带宽：不大于500kHz。
6. 频率容限：100 × 10^{-6}。

注意事项

基于上述要求，使用 LoRaWAN 时应注意以下几点：

1. 单终端及网关设备，不能同时使用两个及以上信道发送数据，但接收数据可以。
2. 发送功率加上天线的增益不能超过 17dBm，即 50mw。
3. 单次发送数据的空中时长 ToA 应不超过1s，根据 semtech 官方提供的 计算公式 计算 ToA。

应用特征

LoRaWAN 并不适用每一种通信应用场景，使用者应根据 LoRaWAN 的特点来适配应用场景。

适用场景

- 长距离：根据不同的遮挡环境，可以覆盖几公里到几百公里，一般视距可达 10Km。

- 低功耗：典型场景下使用电池供电可以工作数年的时间。
- 低成本：通信模组成本可控制在 40RMB 以下，随着时间增长，成本会逐步降低。
- 低带宽：通信速率 250bits 到 5Kbits 不同的调制方式下均适用（SF7 到 SF12）。
- 按需部署：不需要依赖运营商的基站，可根据应用需求灵活部署网关。
- 安全：端到端128位 AES 加密。

不适用场景

- 实时数据：按秒为单位实时的上传应用数据的场景。
- 语音通信。
- 实时及高并发的下行控制场景。
- 大带宽：大数据包传输场景，LoRaWAN 物理层单个数据帧仅支持 242 字节传输。

关键功能及参数

工作模式

Class A

双向传输终端（**Class A**）：Class A 的终端在每次上行后都会紧跟两个短暂的下行接收窗口，以此实现双向传输。终端基于自身通信需求来安排传输时隙，在随机时间的基础上具有较小的变化（即 ALOHA 协议）。Class A 操作为应用提供了最低功耗的终端系统，只要求应用在终端上行传输后的很短时间内进行服务器的下行传输，服务器在其他任何时间进行的下行传输都需要等终端的下一次上行。

Class B

划定接收时隙的双向传输终端（**Class B**）：Class B 的终端有更多的接收时隙。除了 Class A 的随机接收窗口，Class B 设备还会在指定时间打开其他的接收窗口。为了让终端可以在指定时间打开接收窗口，终端需要从网关接收时间同步的信标（Beacon），使服务器知晓终端何时处于监听状态。

Class C

最大化接收时隙的双向传输终端（**Class C**）：Class C 的终端基本处于一直打开接收窗口的状态，只在发送时短暂关闭。Class C 的终端会比 Class A 和 Class B 更加耗电，但同时从服务器下发给终端的时延也是最短的。

> ⚠ **注意：**
> 目前腾讯 LoRaServer 支持 Class A、Class C 两种模式，Class B 暂未支持。

标识符

标识符	说明
DevEUI	64位设备终端标识符，设备唯一。
DevAddr	32位设备地址，设备非唯一。
AppEUI	应用标识符，目前腾讯 LoRaServer 未对 AppEUI 标识限制。
GatewayEUI	网关标识符，设备唯一。
NetID	腾讯使用 LoRaWAN 联盟分配地址前缀0x35（十进制53）。

网络接入模式

空中激活 OTAA

空中激活 OTAA 是目前推荐的连接方式，安全性更高，通过网络执行入网的过程，动态地生产会话密钥及 DevAddr。

本地激活 ABP

本地激活 ABP 接入网络的方式更为简单直接，无需入网流程，通过本地预存的会话密钥进行加解密，但存在一些安全性的问题如重放攻击，因此不推荐使用。

> ⚠ **注意：**
> 目前腾讯 LoRaServer 支持 OTAA、ABP 两种网络接入模式，用户需要妥善维护设备的密钥信息。

物理调制方式及速率

物理调制方式

基于 Chirp 扩频技术，LoRaWAN 使用 LoRa 的调制方式，即使在低功率下，也有很好的抗干扰特性，且多径衰弱及多普勒效应对 LoRaWAN 的影响也非常小。LoRaWAN 的速率取决于使用的带宽及扩频因子，目前在中国区可以使用125Khz带宽，SF7到SF12等6种速率，设备端决定使用扩频因子，不同的扩频因子极大的影响传输数据所需的时间和功耗。

> ⚠ **注意：**
> 在信号强度满足传输的条件下，请尽可能使用较高速率的扩频因子，以减少设备使用功耗，增加电池使用寿命及增加网关接入设备数量。

自适应速率 ADR

自适应速率 ADR 是一种用于优化网络速率、传输时间和功耗的机制，设备终端具有足够的稳定的信号强度下，需要启用 ADR。

- 静态终端：当设备长期处于静态状态，例如水表，此时 RF 环境相对稳定，应启用 ADR 功能，确保设备在合适的速率下运行。
- 移动终端：当设备处于移动的状态，例如包裹跟踪定位器，则在移动的状态下应禁用 ADR，使用固定的速率发送数据，确保数据能够被稳定的接收，当设备检测到设备处于静止的状态，应在此期间开启 ADR。

> ⚠ **注意：**
> 终端决定是否使用 ADR 功能，并非由网络或者应用来确定。ADR 开启期间，网络会根据最近多个数据包信号及丢包状态给出合适的速率建议。

LoRaWAN 产品定义

最近更新时间：2023-05-31 10:59:35

操作场景

用户成功注册腾讯云账号后，通过物联网开发平台将 LoRaWAN 设备对接到腾讯云物联网平台时。您需要创建项目，并在项目下创建产品、定义产品的数据模板。本文档主要介绍如何使用开发平台创建项目并进行 LoRaWAN 产品定义。
LoRaWAN 产品定义与其他产品定义的操作步骤基本一致，只有创建产品时有不同，因此本文档中只对创建 LoRaWAN 产品这一步骤进行说明，其他步骤可参考通用的 产品定义 文档。

操作步骤

产品相当于某一类设备的集合，用户通过产品管理其下的所有设备。

1. 登录 物联网开发平台控制台，单击某个已创建的项目。
2. 进入产品开发页，单击**新建产品**，开始定义您的产品。
3. 根据页面提示填写产品基本信息。产品基本信息设置如下：
   - 产品名称：名称支持中文、英文、数字、下划线、空格（非首尾字符）、中英文括号、-、@、\、/的组合，最多不超过40个字符。
   - 产品类型：选择您所创建产品的所属品类，不同类型产品的属性、事件等数据模板会有所不同。详情请参见 物模型 。
   - 设备类型：选择“设备”。
   - 认证方式：选择“密钥认证”。
   - 通信方式：LoRaWAN 产品的通信方式需要选择“LoRaWAN”。
   - 数据协议：产品默认采用自定义透传协议。
   - 描述：字数不能超过80个，您可以根据需要选填。

4. 信息填写完成后，单击**保存**即可。

LoRaWAN 设备开发

最近更新时间：2023-08-17 16:49:11

操作场景

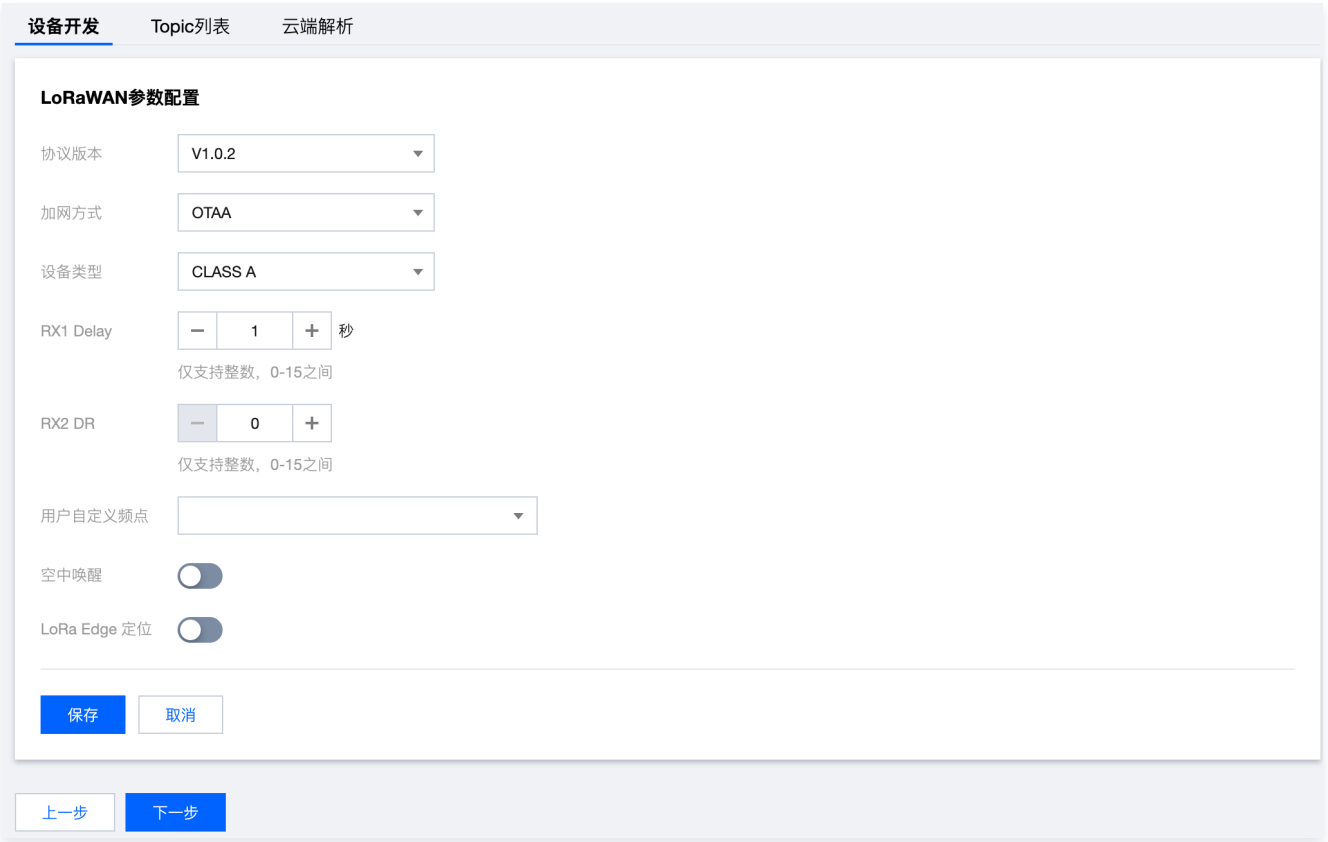
用户定义完产品与数据模板后，需要按接入协议要求将 LoRaWAN 设备接入到平台。本文档主要介绍如何使用开发平台进行 LoRaWAN 设备开发。

前提条件

已创建完产品并定义产品的数据模板。详情请参见 LoRaWAN 产品定义 。

操作步骤

1. 登录 物联网开发平台控制台 ，进入对应项目，单击已创建的产品名称，选择**设备开发**。
2. 选择设备开发方式，并单击右上角**编辑**，按照您的需求调整 LoRaWAN 参数配置。
3. 在弹出的页面中，调整完参数后，单击**保存**，完成 LoRaWAN 参数配置。



参数配置可选功能

功能	说明
协议版本	支持 LoRaWAN V1.0.0、V1.0.1、V1.0.2、V1.0.3。
加网方式	支持 OTAA、ABP 两种方式。
设备类型	支持 Class A、Class B、Class C 三种模式。
RX1 Delay	仅支持整数，0-15之间。
RX2 DR	仅支持整数，0-15之间。
用户自定义频点	用户自定义频点，设置适合用户应用场景的 Region 参数。
空中唤醒	开启后下行的数据会携带前导码，节点可通过前导码进行唤醒，以保证节省功耗同时保证实时性。

	注意：该功能仅支持已对接网关产品使用。

LoRaWAN 设备数据解析

最近更新时间：2022-06-10 14:16:02

操作场景

由于 LoRaWAN 类资源有限设备不适合直接传输 JSON 格式数据，物联网开发平台提供了“设备数据解析”服务。用户通过编写自定义的解析脚本，可以将设备原始数据转化为产品定义的数据模板协议数据。

操作步骤

编写脚本

在设备数据解析页签下，编写脚本。
上行数据解析的脚本主函数为 RawToProtocol，其带有 fPort、bytes 两个入参：

- fPort：设备上报的 LoRaWAN 协议数据的 FPort 字段。
- bytes：设备上报的 LoRaWAN 协议数据的 FRMPayload 字段。

脚本主函数的出参为产品数据模板协议格式的对象。
下行数据解析的脚本主函数为 ProtocolToRaw，其入参为产品数据模板协议格式的对象，其出参为至少3个字节的数组：

- 第1字节：下发给设备的 LoRaWAN 协议数据的 FPort 字段。
- 第2字节：bytes 为下发给设备的 LoRaWAN 协议数据的 MType（0 表示 Unconfirmed Data Down，1 表示 Confirmed Data Down）。
- 第3字节：开始为下发给设备的 LoRaWAN 协议数据的 FRMPayload 字段。

这里以温湿度传感器做示例说明，设备上行数据共4字节：

- 第1字节：温度。
- 第2字节：相对湿度。
- 第3、4字节：表示上报周期（单位秒）。
- 设备下行数据为2字节：上报周期（单位秒）。

在上行数据解析部分，javascript 示例代码如下：

```
function RawToProtocol(fPort, bytes) {
    var data = {
        "method": "report",
        "clientToken" : new Date(),
        "params" : {}
    };
    data.params.temperature = bytes[0];
    data.params.humidity = bytes[1];
    data.params.period = bytes[2] | (bytes[3] << 8);
    return data;
}
```

在下行数据解析部分，javascript 示例代码如下：

```
function ProtocolToRaw(obj) {
    var data = new Array();
    data[0] = 5;// fport=5
    data[1] = 0;// unconfirmed mode
    data[2] = obj.params.period & 0x00FF;
    data[3] = (obj.params.period >> 8) & 0x00FF;
    return data;
}
```

脚本模拟测试

您也可使用数据解析页面下方的模拟调试工具，如需开发更多的功能，请使用以下模拟脚本。
在设备上行数据的编辑框中填入模拟测试的 bytes 数据，为数组类型，编辑框的右上方可以填入模拟测试的 fPort 字段。

- **上行消息**
  假设设备上行原始数据为 0x11451E00，我们将其转化为数组，即上行模拟数据为：[17,69,30,0]，填入设备上行数据的编辑框中。单击**运行**，即可在模拟调试界面右侧查看结果。以温湿度传感器为例：

- **下行消息**
  在设备下行数据的编辑框中填入模拟测试的产品数据模板协议格式数据，为对象类型。
  假设设备下行原始数据如下，将其填入设备下行数据的编辑框中，以温湿度传感器为例：

```
{
"params": {
  "period": 15
}
}
```

  单击**运行**，即可在模拟调试界面右侧查看结果。

提交脚本

确认脚本可以正确解析数据后，单击**提交**，将该脚本提交到物联网开发平台，以供数据上下行时，物联网开发平台调用该脚本解析数据。

LoRaWAN 设备调试

最近更新时间：2024-09-30 17:54:51

操作场景

设备开发完成后，需要进入设备调试阶段，调试设备与云端的通信是否正常。设备调试提供了真实设备在线调试及虚拟设备调试，并可通过控制台查询设备上报的当前数据、历史通信日志、事件及上下线记录等。本文档主要介绍如何进行设备调试。
LoRaWAN 设备调试与其他产品的设备调试的操作步骤基本一致，只有新建设备时有不同，因此本文档中只对新建 LoRaWAN 设备这一步骤进行说明，其他步骤可参考通用的 LoRaWAN 设备调试 文档。

前提条件

已完成设备开发。详情请参见 LoRaWAN 设备开发 。

操作步骤

1. 登录 物联网开发平台控制台，设备开发完成后，单击**设备调试**。
2. 进入设备调试环节，单击**新建设备**，填写设备基本信息，单击**保存**，即可完成创建设备。
   - 设备名称：支持英文、数字、下划线的组合，最多不超过48个字符。
   - DevEUI：仅支持16进制字符，长度16位。
   - AppKey（仅限 OTAA 加网方式）：仅支持16进制字符，长度32位。
   - DevAddr（仅限 ABP 加网方式）：仅支持16进制字符，长度8位。
   - NwkSKey（仅限 ABP 加网方式）：仅支持16进制字符，长度32位。
   - AppSKey（仅限 ABP 加网方式）：仅支持16进制字符，长度32位。

   DevEUI、AppKey、DevAddr、NwkSKey、AppSKey 一般为 LoRaWAN 节点设备厂商提供。如果是自行开发协议栈，可以按需配置，只要平台和节点实际配置的内容一致即可。
   下图示例中为 OTAA 加网方式，如果需要切换到 ABP 加网方式，可以在**设备开发**界面中调整 “LoRaWAN 参数配置” 中的加网方式。

3. 创建成功后，您将会在“设备调试”列表页中，查看到新建成功的设备。

查看设备

1. 创建设备成功后，您将会在“设备调试”列表页中，查看到新建成功的设备。
2. 单击设备名称，可以查看设备相关信息、设备信息、设备日志、透传日志等。

设备信息

单击**设备信息**，即可查看设备基础的信息、名称、密钥、激活时间、最后上线时间等。
此外，在 ABP 模式下，可在设备信息中选择**禁用帧序号校验（临时调试使用）**以及**重置帧序号**功能。

> **说明：**
> 此功能建议用于设备临时调试，以解决设备上传序列号归零导致 MIC 校验错误的问题；不建议用户用于商业应用，存在设备重放攻击的安全隐患。

设备日志

单击**设备日志**，即可查看该设备上行到云端，并从云端接收的信息，可查看7天以内的设备日志内容。

- **上行：**上行表示设备端向云端上报的数据。
- **下行：**下行表示云端向设备端发送的数据。

设备透传日志

单击**设备日志（透传数据）**，即可查看数据透传到用户侧的数据内容，以及 LoRaWAN 网络侧内容数据。

示例

数据内容如下：

```
“eyJtZXRob2QiOiJyZXBvcnQiLCJjbGllbnRUb2tlbiI6IjIwMjEtMDEtMThUMDU6NTE6MDIuMDM4WiIsInBhcmFtcyI6eyJyYXdkYXRhIjoiMDAwMTAyMDMifSwibWV0YUxvUmEiOiJ7XCJmcmFtZVR5cGVcIjoyLFwiZlBvcnRcIjoyLFwiZkNudFwiOjQ2NCxcImZyZXF1ZW5jeVwiOjQ3MDMwMDAwMCxcImRyXCI6NSxcInJzc2lcIjotNTAsXCJzbnJcIjoyNixcInBheWxvYWRTaXplXCI6NH0ifQ==”
```

通过 BASE64 解析成文本的格式如下：

```
{"method":"report","clientToken":"2021-01-18T05:51:02.038Z","params":{"rawdata":"00010203"},"metaLoRa":"{\"frameType\":2,\"fPort\":2,\"fCnt\":464,\"frequency\":470300000,\"dr\":5,\"rssi\":-50,\"snr\":26,\"payloadSize\":4}"}
```

在线调试

当您的真实设备已成功对接到开发平台后，则可使用在线调试对真实设备进行数据收发的测试，具体步骤如下：

1. 单击**在线调试**，即可进入在线调试功能。
2. 在线调试左侧的操控面板是根据设备所属产品的数据模板自动生成，设置需要下发的数据后，单击**发送**，系统会自动触发控制指令到设备端。
3. 设备端接收到指令后，会立刻返回数据到云端并显示在右侧的文本框中。

LoRaWAN 网关管理

最近更新时间：2023-05-31 10:59:35

操作场景

用户创建完 LoRaWAN 设备后，需要通过 LoRaWAN 网关将设备接入到平台。本文档主要介绍如何使用物联网开发平台进行 LoRaWAN 网关管理。

操作步骤

使用腾讯 LoRaWAN 社区网络

1. 登录 物联网开发平台控制台，单击某一个已新建产品的项目。
2. 进入项目列表页，选择左侧菜单**网络管理 > LoRaWAN 网关管理**。导航栏右侧会展示出腾讯 LoRaWAN 社区网络的地图页面。
3. 您可以单击地图，并查看自己的附近是否有正在运行的 LoRaWAN 网关，这些网关为社区的开发者主动共享的，可以直接通过这些网关进行 LoRaWAN 设备的无线接入。

新建用户 LoRaWAN 网关

1. 登录 物联网开发平台控制台，单击某一个已新建产品的项目。
2. 单击项目名称进入项目列表页，选择左侧菜单**网络管理 > LoRa网关管理**，单击**添加网关**。

3. 在新建网关页面，填写网关基本信息。
   - 网关名称：本示例中填写 GW1。
   - GwEUI：为网关唯一 ID。
   - 是否公开：
     - 选择“是”，表示社区开发者可在社区网络中看到该网关，并可通过这个网关进行 LoRaWAN 节点接入。
     - 选择“否”，则只有用户自己才能查看该网关。

4. 网关新建成功后，您可在网关列表页查看到“GW1”。

用户 LoRaWAN 网关操作

> **说明：**
> 用户的 LoRaWAN 网关需支持 Packet Forwarder 协议。

LoRaWAN 网关上的配置需做如下调整：

```
配置接入域名：loragw.things.qcloud.com
接入端口：1700
```

当网关按要求配置并重启运行之后，即可在网关列表页查看到网关的在线状态变为“在线”。

LoRaWAN 用户自定义频点

最近更新时间：2024-08-26 14:55:51

背景介绍

- LoRaWAN 联盟在 lorawan_regional_parameters_v1.x 版本的 CN470**470Mhz，510Mhz**40Mhz频宽中，定义了上行96个信道，下行48个信道。
- LoRaWAN 联盟在 lorawan_regional_parameters_v2.x 版本的 CN470**470Mhz，510Mhz**40Mhz频宽中，定义了26M TypeA 和 TypeB、20M TypeA 和 TypeB 共4种频谱计划。

频谱计划	上行信道数	下行信道数	总频宽	上行频点对应模式
RP1.0	96	48	40M	异频
RP2.0 20M TypeA	64	64	20M	异频
RP2.0 20M TypeB	64	64	20M	同频
RP2.0 26M TypeA	48	24	26M	异频
RP2.0 26M TypeB	48	24	26M	异频

自定义功能说明

- 自定义频点功能兼容 RP1.0、RP2.0 各个版本的频谱计划，同时也可以支持用户自定义频谱计划，以满足不同环境下的要求。
- 自定义频点功能支持 MACCommand 中大部分功能，除了 linkadr 中 MASK 参数，设备回复 linkadrans 应作出相应的策略。

功能	说明
单信道频谱要求	网关工作在单个信道。
多信道频谱要求	网关工作在多个信道，支持2个 – 64个信道，自由选择可用信道。
全双工工作模式	网关上下行发送和接收数据同时工作。
上下同频	上下行共用同一组频点。
上下异频	上下行使用不同组频点。

操作步骤

新建频点计划

1. 登录 物联网开发平台控制台，单击某一个已新建产品的项目。
2. 进入项目列表页，选择左侧菜单**网络管理 > LoRa 网关管理**，在页面上方选择**用户自定义频点**后，单击**添加频点**，进入“添加用户自定义频点”页面。
   - 频点配置名称：支持英文、数字、下划线的组合，最多不超过48个字符。
   - 数据信道配置：分别填入上行信道，下行 RX1 信道，下行 RX2 信道，单位Hz，仅支持数字，范围为470000000-510000000。
   - 入网信道配置：分别填入上行信道，下行 RX1 信道，下行 RX2 信道，单位Hz，仅支持数字，范围为470000000-510000000，若未配置入网信道，则按照数据信道入网。

示例

以双信道频点计划为例，配置两个信道，同频工作模式的频点计划：

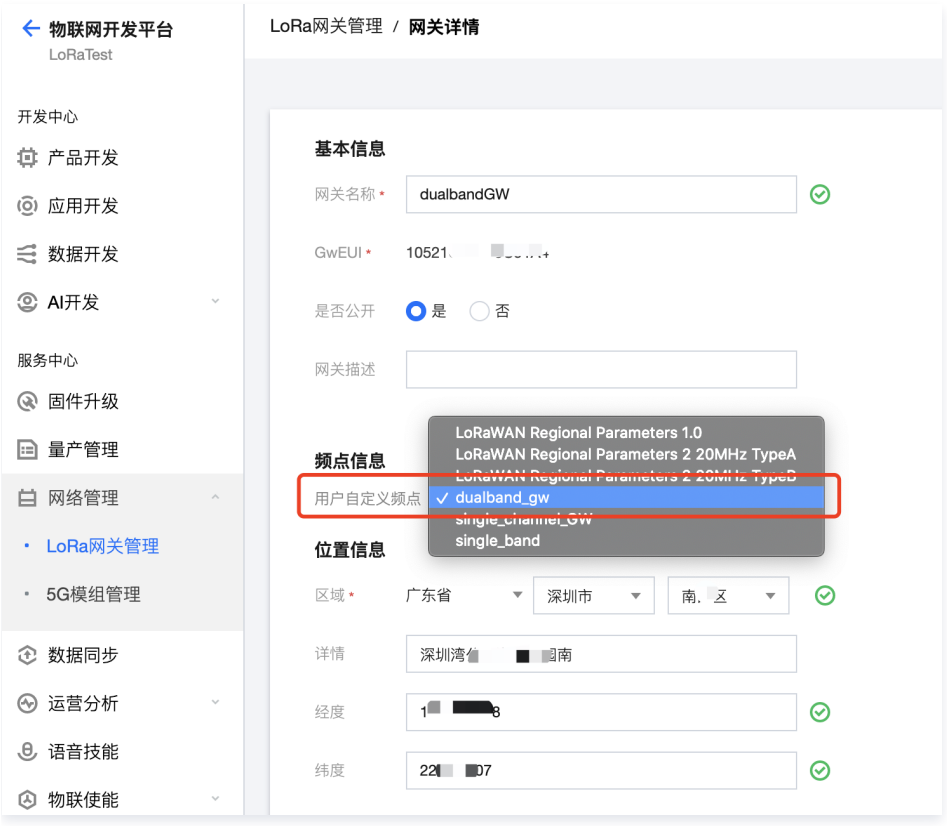
网关关联频点计划

1. 登录 物联网开发平台控制台，单击某一个已新建产品的项目。
2. 进入项目列表页，选择左侧菜单**网络管理 > LoRa 网关管理**，在页面上方选择**用户网关**后，单击**添加网关**，进入新建或者编辑网关界面。
3. 在频点信息处选择用户自定义频点，关联与网关对应的频点计划。

> **说明：**
> 缺省用户频点为 RP1.0 频点计划，可选支持RP2.0 20M TypeA & TypeB。

示例

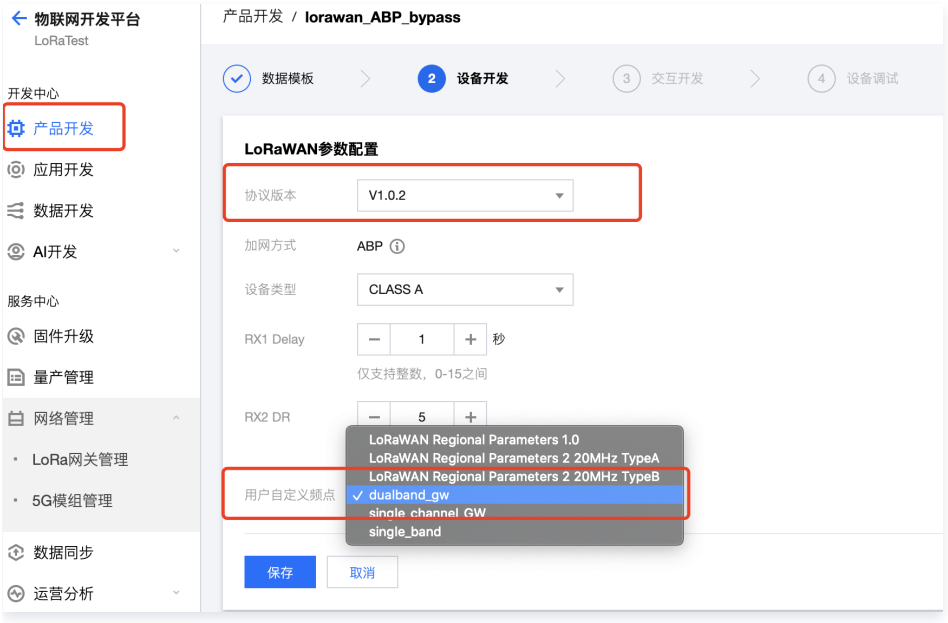
以网关关联双芯片频点计划为例：



设备关联频点计划

1. 登录 物联网开发平台控制台，单击某一个已新建产品的项目。
2. 进入项目列表页，选择左侧菜单**产品开发**，选择需要打开的产品，在“数据模板”页面下方，单击**下一步**进入“设备开发”页面。
3. 在 LoRaWAN 参数配置界面中，选择用户自定义频点，关联与设备对应的频点计划。

> **说明：**
> 同一个网络中，多个网关硬件使用相同的频点配置，若设备或者网关未关联对应的频点计划，则上行数据可以正常接收，下行数据发送只会选择关联频点计划的网关。



应用开发

最近更新时间：2022-06-10 14:16:43

操作场景

应用开发是为了满足设备厂商创建自有品牌小程序的场景，设备厂商可根据开发平台提供的服务为设备厂商的终端用户提供小程序应用，以此提升消费者的体验，提升设备厂商产品附加值。

操作步骤

新建应用

1. 登录 物联网开发平台 。选择对应的项目。
2. 选择左侧菜单**应用开发**，单击**新建应用**。
3. 进入新建应用页面，填写相关信息。
   - **应用名称**：根据实际业务输入便于识别的应用名称。
   - **描述**：输入应用的备注信息。

4. 单击**保存**，即可新建应用。

删除应用

> **注意：**
> 为了防止误操作删除数据影响您的业务，需要确认该应用的数据是否已正在使用，如果正在使用，删除后，您的应用再访问开发平台的 API 时将会出错。

1. 当您无需该应用时，可以在应用列表的操作区域，单击**删除**即可。
2. 确认删除后，系统将删除该应用。

数据开发
数据开发配置

最近更新时间：2023-05-31 18:01:05

操作场景

为了满足开发者根据设备上报的数据及状态，通过在云端定义规则，即可实现将告警、通知消息实时推送至腾讯连连公众号或 App 推送，降低开发者处理设备上报数据的成本。

前提条件

已 创建项目及产品，并定义该产品的 物模型。

新建告警任务

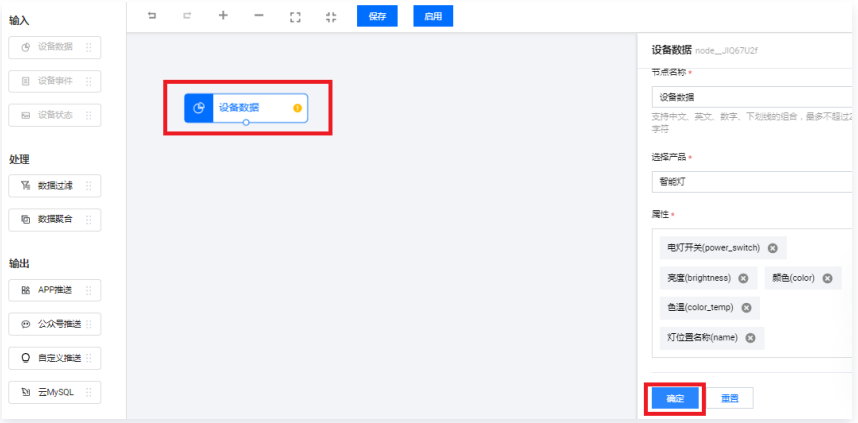
1. 登录 物联网开发平台控制台 ，选择**公共实例**或者您购买的**标准企业实例**进入项目列表页。
2. 单击项目名称进入项目详情页面，单击左侧菜单**数据开发**。
3. 进入数据开发列表页面，单击**新建数据流**。

   - **数据流名称**：输入数据流处理的名称标识。
   - **描述**：输入该数据流处理的备注信息。
4. 单击**保存**，保存成功后则显示数据流列表页。

告警任务配置

设备属性编排

1. 单击数据流列表页中的**数据流名称**，进入编排页面。
2. 鼠标长按左键拖动左侧输入区域中的**设备数据**节点，放置到画布区域。设备数据节点对应的是物模型中设备上报的属性。
3. 单击已拖放的**设备数据**节点，界面右侧显示该节点的配置内容。

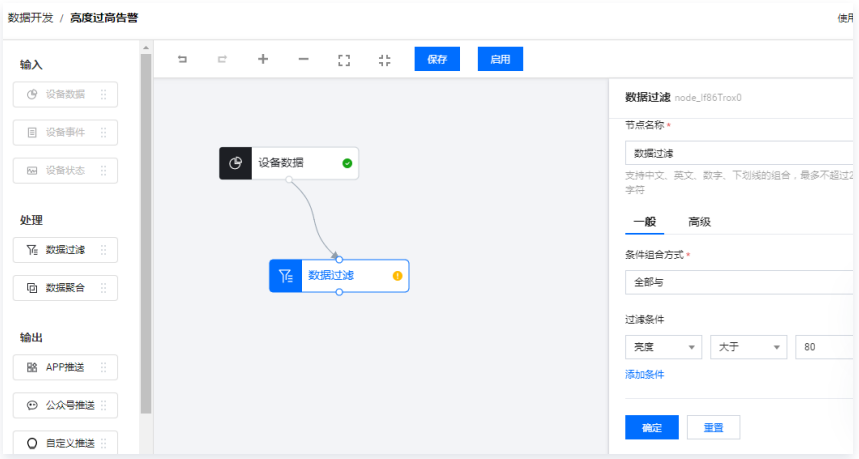


- 节点名称：该**设备数据**的节点名称，默认为“设备数据”可修改。
- 选择产品：下拉选择该项目下的某个产品后，平台则对该产品下所有设备上报的数据进行实时处理。
- 属性：用户可根据需要选择该产品下哪些属性作为输入。

4. 选择完产品、属性后，必须要单击**确定**，系统才会保存该节点的配置数据。
5. 当**设备数据**节点保存成功后，画布中的**设备数据**节点右侧图标将会变成绿色。

数据过滤编排

1. 鼠标长按左键拖动左侧处理区域中的**数据过滤**节点，放置到画布区域。
2. 单击已拖放的**数据过滤**节点，界面右侧显示该节点的配置内容。
3. 在配置“数据过滤”前，必须要指定数据源，即需要将“设备数据”与“数据过滤”两个节点进行连接。

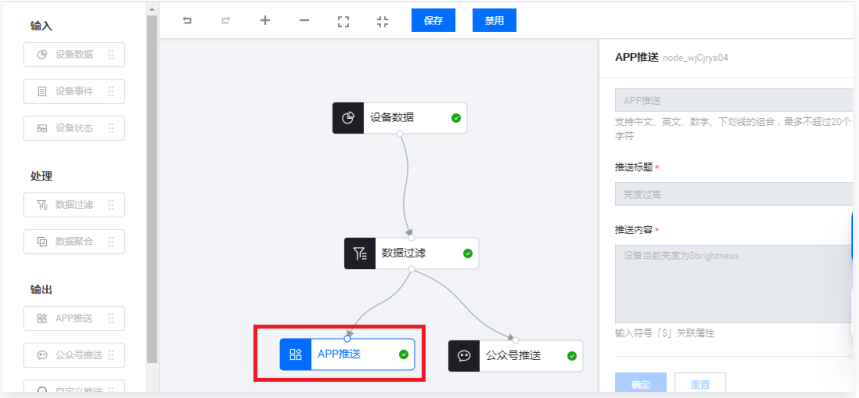


4. 数据过滤条件目前支持全部与、全部或逻辑组合，可根据上一个节点的输出任意组合过滤条件，本文选择条件组合方式为“全部与”。
5. 添加“过滤条件”，选择“亮度”属性，选择条件为“大于”，输入值80；表示只有当设备上报的亮度值为大于80的数据才会输出到下一个处理节点。

腾讯连连 App 推送

若开发者的告警场景是将告警信息推送到腾讯连连 App，则可以将“App 推送”节点拖放到画布。

1. 鼠标长按左键拖动左侧输出区域中的 **App 推送**节点，放置到画布区域。
2. 单击已拖放的 **App 推送**节点，界面右侧显示该节点的配置内容。
3. 在配置“App 推送”前，必须要指定数据源，即需要将“App 推送”与“数据过滤”两个节点进行连接。



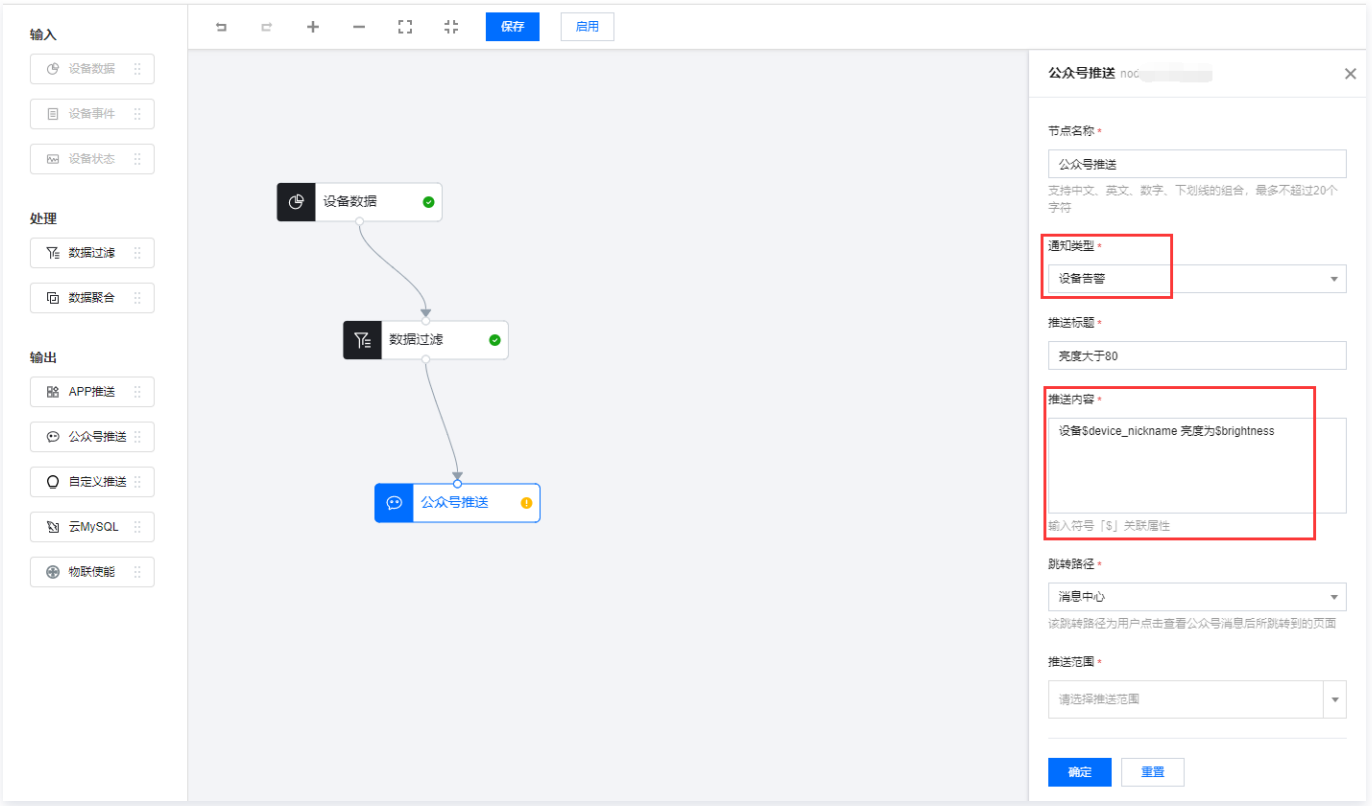
4. 输入推送标题。
5. 推送内容可以由开发者定义，并且可以通过输入$ 获取上一个节点输出的数据。
6. 单击**确定**保存该节点的配置。

腾讯连连公众号推送

若开发者的告警场景是将告警信息推送到腾讯连连公众号，则可以将“公众号推送”节点拖到画布。

1. 鼠标长按左键拖动左侧输出区域中的**公众号推送**节点，放置到画布区域。
2. 单击已拖放的**公众号推送**节点，界面右侧显示该节点的配置内容。

3. 在配置“公众号推送”前，必须要指定数据源，即需要将“公众号推送”与“数据过滤”两个节点进行连接。



4. 选择通知类型：设备告警或通知消息，通知类型的区别在于公众号消息模板标题不同。
5. 输入推送标题。
6. 推送内容可以由开发者定义，并且可以通过输入$ 获取上一个节点输出的数据。包括设备上报的属性数据以及系统级的数据，例如产品 ID、设备 ID、设备别名等。
7. 跳转路径：该跳转路径为用户点击查看公众号消息后所跳转到的页面。可在下拉项中选择。
8. 推送范围：可在下拉项中选择推送给哪些角色。

> ⚠ **注意：**
> 推送内容需遵循微信公众号模板消息规定，单个中间主内容不超过 20 字，且不支持换行。

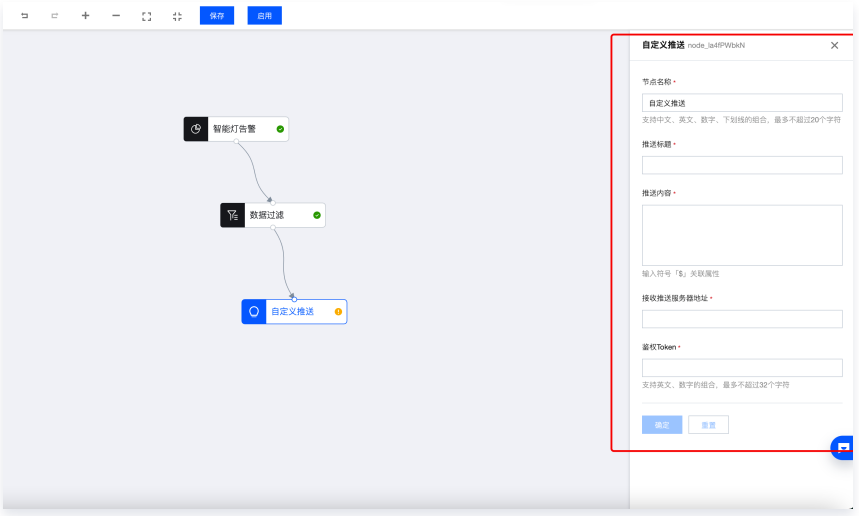
9. 单击**确定**保存该节点的配置。

自定义推送

若开发者需要将过滤、聚合后的数据转发到开发者自己的服务器上，可以使用自定义推送。

1. 鼠标长按左键拖动左侧输出区域中的**自定义推送**节点，放置到画布区域。
2. 单击已拖放的**自定义推送**节点，界面右侧显示该节点的配置内容。

3. 在配置“自定义推送”前，必须要指定数据源，即需要将“自定义推送”与“数据过滤”两个节点连接起来。



4. 输入推送标题以及推送内容。
5. 输入接受告警推送消息的服务器地址和鉴权 Token 并单击**确定**

> **说明：**
> 为了您后台稳定使用，请填写鉴权 Token。您可以任意填写 Token，用作生成签名（该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性）。

推送消息内容格式

1. 平台侧推送到服务端的告警内容消息模板如下：

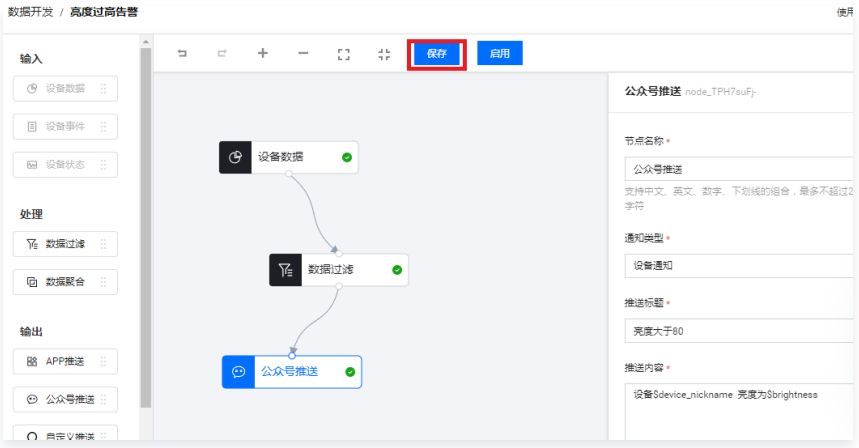
```
{
    "RequestId": "推送请求Id",
    "ProductId": "产品ID",
    "DeviceName": "设备名称",
    "MsgTitle": "推送标题",
    "MsgContent": "推送内容"
}
```

2. 若需要推送到自主品牌公众号，请参见微信公众号官方文档-通过模板消息推送 。
3. 若需要推送到自主品牌小程序或 App 的告警消息，开发者可自行根据业务逻辑实现。

保存与启用数据流

1. 用户设置完输入、处理与输出节点的规则后，单击页面上方**保存** > **启用**。

2. 当启用数据流后，只要该设备上报的数据符合定义的规则，则会触发公众号推送。



体验公众号推送

告警规则如上述步骤配置完成并启用后，可遵循以下两步体验虚拟设备告警推送公众号消息。

“腾讯连连”小程序绑定虚拟设备

1. 进入上述告警规则所对应的产品，单击**设备调试** > **虚拟设备调试**。

2. 打开微信 App，直接使用微信“扫一扫”扫码虚拟设备二维码。

3. 绑定成功后会在设备列表显示如下。

4. 首次绑定需要关注“腾讯连连”公众号，点击下图红色线框关注腾讯连连公众号后，才可以接收云端推送的公众号消息。

虚拟设备模拟上报数据

1. 在虚拟设备操控面板，将亮度值填写为91，其他值也填写对应值，单击**上报**。

2. 进入腾讯连连公众号，将会显示一条推送“消息”

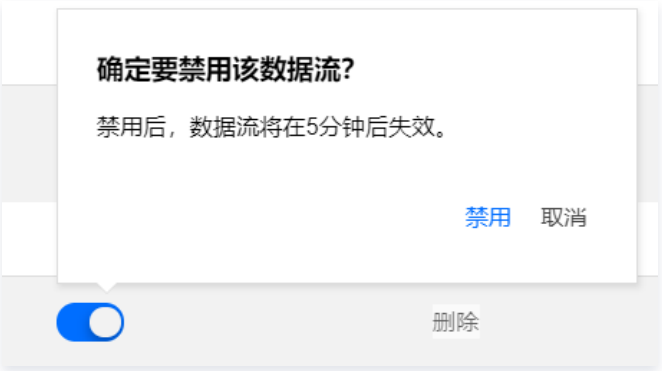
体验 App 推送

1. 参考 通用 App 下载说明，下载对应 App。

2. 体验公众号推送 虚拟设备模拟上报数据后，可查看如下 App 推送消息。

禁用数据流

1. 当用户需要停止某个数据流服务时，进入数据开发页面查看对应的数据流列表。
2. 单击生效状态列下的**关闭**，系统弹出确认提示，单击**禁用**表示停止该数据流服务，即使设备上报数据后，系统将不会进行处理。

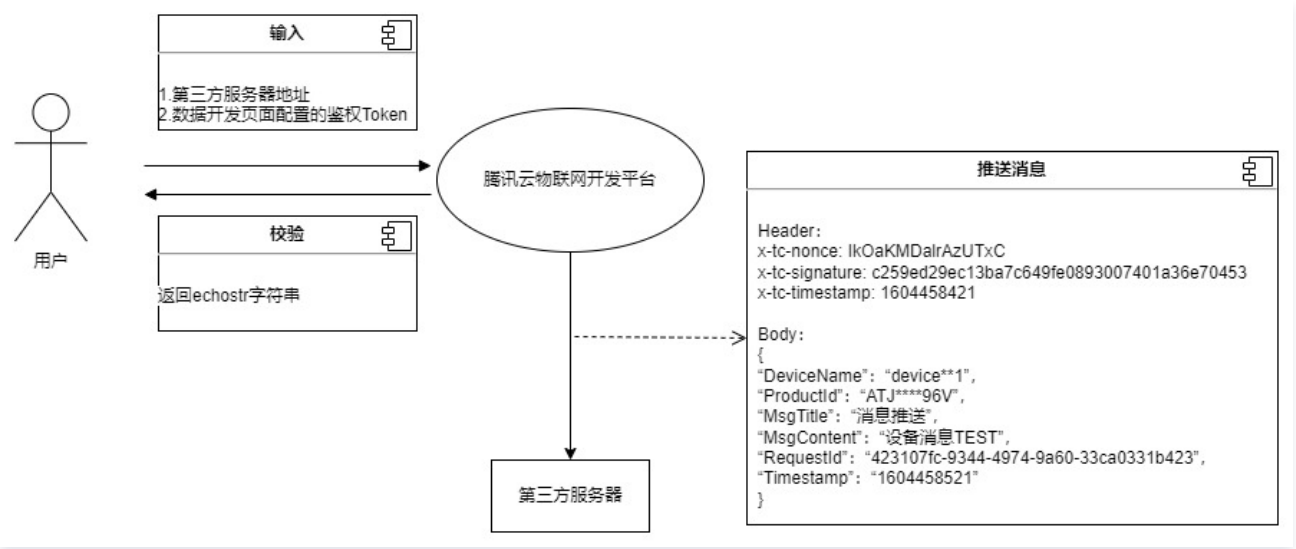


自定义推送示例

最近更新时间：2024-11-05 15:02:52

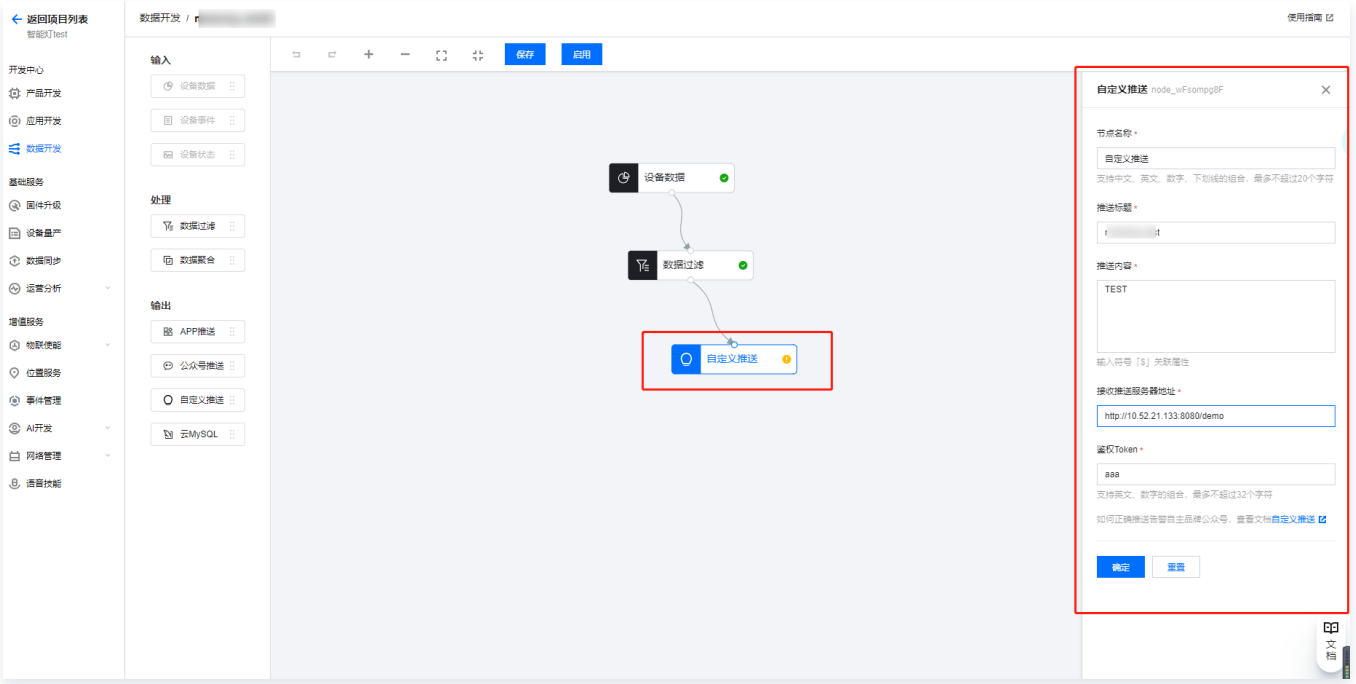
概述

在数据开发模块中，消息推送类型可选择自定义推送，将消息推送给第三方服务器。
下图展示了将消息推送给第三方服务器的整个过程：



填写自定义推送配置

1. 登录 物联网开发平台控制台，选择左侧菜单**数据开发**。
2. 单击**新建数据流**，创建完成之后，单击**数据流名称**，进入数据流配置。
3. 配置好 “输入节点”、“处理节点” 后，“输出节点” 选择**自定义推送**。
4. 单击**自定义推送**，在右侧输入“推送标题”、“推送内容”、“接收推送服务器地址”、“鉴权 Token”。



验证消息来自物联网开发平台

请求标识

自定义推送消息到第三方服务器时，物联网开发平台将在 HTTP 或 HTTPS 请求中头部增加如下字段：

参数	描述
x-tc-nonce	随机数。
x-tc-signature	x-tc-signature 结合了“添加规则”中填写的 Token 参数和请求中的 x-tc-timestamp 参数、x-tc-nonce 参数。
x-tc-timestamp	时间戳。

1. 将 Token、x-tc-timestamp、x-tc-nonce 三个参数进行字典序排序。
2. 将三个参数字符串拼接成一个字符串进行 sha1 加密。
3. 开发者获得加密后的字符串可与 x-tc-signature 对比，标识该请求来源于物联网开发平台。

检验 x-tc-signature 的 PHP 示例代码如下：

```
private function checkSignature()
{
    $signature = $_GET["x-tc-signature"];
    $timestamp = $_GET["x-tc-timestamp"];
    $nonce = $_GET["x-tc-nonce"];

    $token = TOKEN;
    $tmpArr = array($token, $timestamp, $nonce);
    sort($tmpArr, SORT_STRING);
    $tmpStr = implode( $tmpArr );
    $tmpStr = sha1( $tmpStr );

    if( $tmpStr == $signature ){
        return true;
    }else{
        return false;
    }
}
```

例如某次请求，相关参数如下，用户设置 Token 为 aaa。

```
x-tc-nonce: IkOaKMDalrAzUTxC
x-tc-signature: c259ed29ec13ba7c649fe0893007401a36e70453
x-tc-timestamp: 1604458421
```

排序后的字符串是 `1604458421IkOaKMDalrAzUTxCaaa`，最终计算 sha1 结果为 `c259ed29ec13ba7c649fe0893007401a36e70453`。

服务地址校验

1. 当触发一次数据流告警推送时，物联网开发平台将发送一次 GET 请求到填写的服务器地址 URL 上，GET 请求头部增加如下字段：

参数	描述
x-tc-nonce	随机数。
x-tc-signature	x-tc-signature 结合了“添加规则”中填写的 Token 参数和请求中的 x-tc-timestamp 参数、x-tc-nonce 参数。
x-tc-timestamp	时间戳。
echostr	随机字符串。

物联网开发平台向第三方服务发送报文示例：

```
GET / HTTP/1.1
host: **.**.**.**:4443
user-agent: Go-http-client/1.1
```

```
content-type: application/json
echostr: 6a7db17a-90e0-4387-b33e-4dd1578a151b
x-tc-nonce: 624665043113817867
x-tc-signature: abb6c316a8134596d825c5a1295bfa6f7657664d
x-tc-timestamp: 1623149590
accept-encoding: gzip
```

2. 第三方服务若确认此次 GET 请求来自物联网开发平台，请在 body 中原样返回 echostr 参数内容。
第三方服务回复物联网开发平台报文示例：

```
HTTP/1.1 200 OK
Date: Tue, 08 Jun 2021 10:53:10 GMT
Content-Length: 16
Content-Type: text/plain; charset=utf-8
6a7db17a-90e0-4387-b33e-4dd1578a151b
```

GET 请求的 PHP 示例代码如下：

```
if( $_SERVER['REQUEST_METHOD'] === 'GET'){
            $header = get_AllHeaders();
            $signature = $header['signature'];
            $timestamp = $header['timestamp'];
            $nonce = $header['nonce'];
            $echostr = $header['echostr'];
            #验签成功与否标志 $flag
            $flag = checkSignature($signature,$timestamp,$nonce);

            if ($flag === true) {
                header('Content-Type: text/plain; charset=utf-8');
                $len = 'Content-Length: ';
                $len .= strlen($echostr);
                header($len);
                echo $header['echostr'] ;
            }else {
                echo '验签失败',"\r\n";
            }
}
```

3. 物联网开发平台校验返回的 echostr 参数内容，确认服务器地址 URL 是否有效。

推送消息给第三方服务器

满足推送消息条件时，物联网平台会以 POST 方式将消息推送到第三方服务器（注：上述的 GET 请求只是为了校验签名，获取消息需在 POST 请求中获取）。发送的数据为：

Header：

参数	描述
x-tc-nonce	随机数。
x-tc-signature	x-tc-signature 结合了“添加规则”中填写的 Token 参数和请求中的 x-tc-timestamp 参数、x-tc-nonce 参数。
x-tc-timestamp	时间戳。

Body：

参数	描述
DeviceName	设备名称。

ProductId	产品 ID。
MsgTitle	推送标题。
MsgContent	推送内容。
RequestId	唯一请求 ID。
Timestamp	时间戳。

POST 请求的 PHP 示例代码如下：

```
if( $_SERVER['REQUEST_METHOD'] === 'POST') {

    $data = file_get_contents('php://input');

    echo $data;

    // 用户根据自己的业务处理 $data 消息

}
```

基础服务
子账号权限
创建子账号

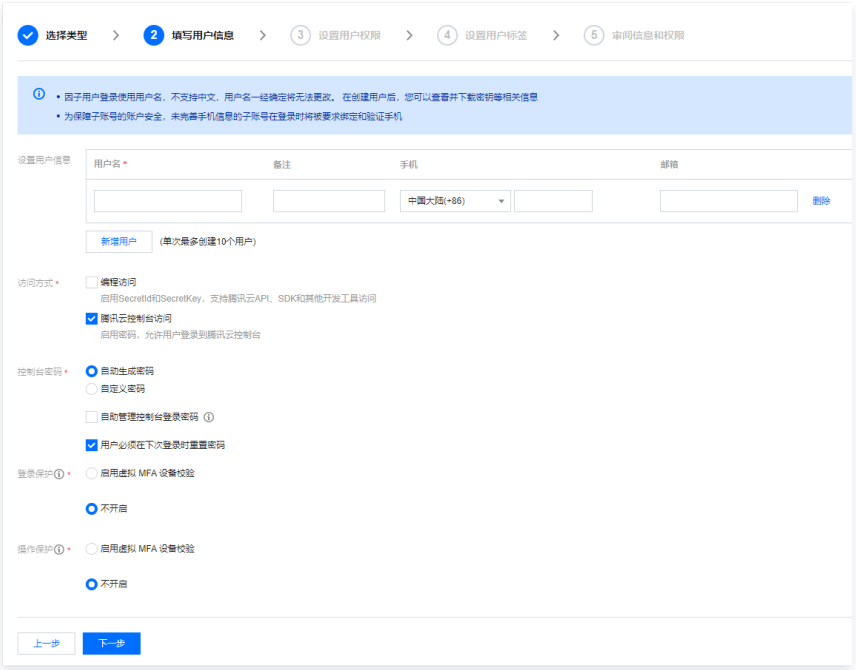
最近更新时间：2023-05-31 18:01:05

操作场景

本文主要介绍如何为主账号添加子账号，赋予子账号一定级别的资源管理权限，以“子用户”为例。

操作步骤

1. 使用腾讯云主账号登录 访问管理控制台，选择左侧菜单栏**用户** > **用户列表**。
2. 进入用户列表页面，单击**新建用户**。
3. 弹出用户类型选择界面，选择**自定义创建**子用户。
4. 选择“可访问资源并接收消息”，单击**下一步**。
5. 填写子用户的必要信息，可以自定义“用户名”，对应手机号码，访问方式选择“控制台访问”。选择控制台访问后，需要设置该子用户的密码。如果您要通过 API 访问，则建议选择“编程访问”，如下图：



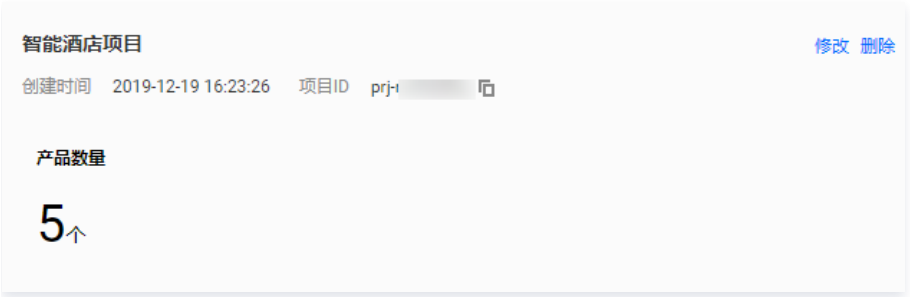
6. 设置完密码后，单击**下一步**，进入使用主账号绑定的手机验证码进行身份验证。
7. 进入授权界面，在文本框中输入预先定义的“策略”，勾选后，单击**下一步**，进入最终确认界面。
8. 单击**完成**，表示主账号分配了某个“策略”给“子用户”。
9. 主账号创建完子用户后，子用户可以通过分配的子账号访问腾讯云控制台，一般在主账户下查看该子用户的用户详情页面，页面右侧会展示控制台登录链接（如下图红线框中内容）。

子账号权限控制

最近更新时间：2022-06-10 14:43:13

操作场景

本文主要介绍如何授予子账号产品级访问控制权限。产品级访问控制权限可以让子账号对自己创建的产品或主账号为其创建的产品拥有访问控制能力。
主账号已创建了一个或多个项目，并在某个项目下建立了若干产品。例如智能酒店项目下有5个产品，分配给5个不同的合作商。如下图所示：



操作步骤

创建策略

1. 使用腾讯云主账号登录 访问管理控制台，单击左侧菜单**策略**。
2. 进入策略页面，单击**新建自定义策略**。
3. 选择**按策略语法创建**。
4. 选择模板类型，勾选**空白模板**，单击**下一步**。
5. 填写自定义策略名称，并按照策略模板编辑策略内容。

示例代码如下：

- 分配子账号所有权限示例：

```
{
  "version": "2.0",
  "statement": [
      {
          "action": [
              "*"
          ],
          "resource": [
              "qcs::iotcloud:gz:uin/your_uid:*",
              "qcs::iotexplorer:gz:uin/your_uid:project/your_project_id",
              "qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/",
```

```
            "qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/product/your_product_id",
            "qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/product/your_product_id/*"
        ],
        "effect": "allow"
    }
  ]
}
```

策略说明如下：

- Resource 对应的就是项目和产品。如果要把主账号某个项目 ID 的某个产品 ID 授权给某个子用户，则需要在 resource 部分增加下面4条，红色标注为需替换部分 :your_uid 为用户账号 ID，your_project_id 为控制台项目 ID，your_product_id 为项目内产品 ID。

```
"qcs::iotexplorer:gz:uin/your_uid:project/your_project_id",
"qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/",
"qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/product/your_product_id",
"qcs::iotexplorer:gz:uin/your_uid:project/your_project_id/product/your_product_id/*"
```

- Action：`*` 号表示所有操作。
- Effect：allow 表示允许， deny 表示不允许。
- 项目策略语法使用说明，请参见 策略语法说明 。

- 禁用子账号部分权限：
  示例代码（此处示例禁用了子账号删除产品和设备）：

```
{
  "version": "2.0",
  "statement": [
      {
          "action": [
              "*"
          ],
          "resource": [
              "qcs::iotcloud:gz:uin/your_uid:*",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/product/your-product-id",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/product/your-product-id/*"
          ],
          "effect": "allow"
      },
      {
          "action": [
              "iotexplorer:DeleteStudioProduct",
              "iotexplorer:DeleteDevice"
          ],
          "resource": [
                "qcs::iotcloud:gz:uin/your_uid:*",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/product/your-product-id",
              "qcs::iotexplorer:gz:uin/your_uid:project/your-project-id/product/your-product-id/*"
          ],
          "effect": "deny"
      }
  ]
}
```

子账号登录控制台，删除项目或产品，会弹出窗口提示无权限：

- Action：输入相关的接口名称，例如：DeleteStudioProduct（删除产品），DeleteDevice（删除设备）等。具体其他接口名称请查阅 API 概览 相关接口。
- Effect：allow 表示允许， deny 表示不允许。

关联策略

1. 自定义策略创建完毕后，进入**用户 > 用户列表**页面，选择想要赋予权限的子账号。
2. 单击用户名称，进入用户详情页，在“权限”栏中，单击**关联策略**。
3. 搜索刚才创建的策略名称，选择后单击**确定**，完成授予策略中定义的权限。

固件升级

最近更新时间：2023-07-13 17:41:23

操作场景

本文档主要介绍固件升级在物联网开发平台的使用方法，帮助您快速使用固件升级服务。

操作步骤

添加固件

1. 登录 物联网开发平台控制台，单击已新建产品的目标项目。
2. 进入项目列表页，选择左侧导航**固件升级**进入固件列表页，可查看当前项目中的全部固件。
3. 单击**添加固件**添加新固件。

- 固件名称：支持中文、英文大小写、数字、部分常用符号（下划线，减号，括弧），必须以中文、英文或数字开头，长度不超过32个字符。
- 所属产品：选择上传固件所属的产品。
- 固件类型：选择上传固件类型，选项为 MCU、模组固件。
- 固件版本号：仅支持英文字母、数字、点、中划线和下划线，长度限制1 - 32字符。
- 选择固件：上传的固件文件必须为 bin 文件或 tar/gz/zip 包，上传的固件文件大小不能超过1024MB。
- 固件描述：对本次上传的固件进行描述和记录，长度限制0 - 100字符。
- 一个账号下最多可上传100个固件 ，若继续上传，则需要删除旧版本固件。

4. 上传完成后，固件将显示在列表中，可对固件进行升级、增删查改、查看详情等操作。

固件升级

固件上传成功后，选择想要升级到的目标固件版本，单击固件列表右侧的**固件升级**发起升级任务。固件升级方式支持按固件版本号升级和按设备名称升级两种批量升级方式。

按固件版本号升级

1. 进入固件升级页，页面展示目标升级固件的信息（例如固件名称、所属产品、固件版本号等）。
2. 选择**批量升级方式**为“按固件版本”升级。

- 待升级版本号：选择下拉框中的固件版本号作为等待被升级的固件，可多选。
- 升级范围：支持两种升级范围，可将待升级版本号下的全部设备或者指定设备作为固件升级目标设备。

  指定设备升级功能常用于灰度验证固件内容。选择升级范围为指定设备时，单击下拉框右侧的**选择设备**按钮，可以从该产品下全部设备中批量勾选目标升级设备。
- 升级确认：支持静默升级、用户确认升级两种固件升级确认方式。如您使用腾讯连连官方应用，静默升级是指无需用户确认，腾讯连连应用端将自动完成升级，再次开启后为升级过的版本；用户确认升级是指用户需主动进入腾讯连连的设备控制界面，在设备详情页检查并确认固件升级。如您使用其他物联网应用，则推荐选择静默升级方式。

- 超时时长配置：当云端超过超时时长没有收到设备固件升级消息时，则会重新下发固件升级任务。静默升级超时时长默认为15分钟。用户确认升级超时时长默认为2分钟。您也可以根据业务实际需求自定义配置。

3. 单击**保存**后，系统将会执行升级任务，下发所选的目标版本固件到升级范围内的目标设备中。

> **说明：**
> 按固件版本升级需要待升级设备上报当前运行的固件版本，如未上传您可选择下文介绍的按设备名称升级。

按设备名称升级

1. 进入固件升级页，页面展示目标升级固件的信息（例如固件名称、所属产品、固件版本号等）。
2. 选择**批量升级方式**为“按设备名称”升级。

- 指定设备：上传需要升级固件的设备清单。单击**下载模板**得到模板文件，并在模板文件中录入准确的 DeviceName 后单击**上传文件**进行上传。一次最多可升级10000个设备，文件仅支持 csv 格式。
- 升级确认：与按固件名称升级相同，支持静默升级、用户确认升级两种固件升级确认方式。

- 超时时长配置：当云端超过超时时长没有收到设备固件升级消息时，则会重新下发固件升级任务。静默升级超时时长默认为15分钟。用户确认升级超时时长默认为2分钟。您也可以根据业务实际需求自定义配置。

3. 单击**保存**后，系统将会执行升级任务，下发固件到目标设备中。

查看固件详情

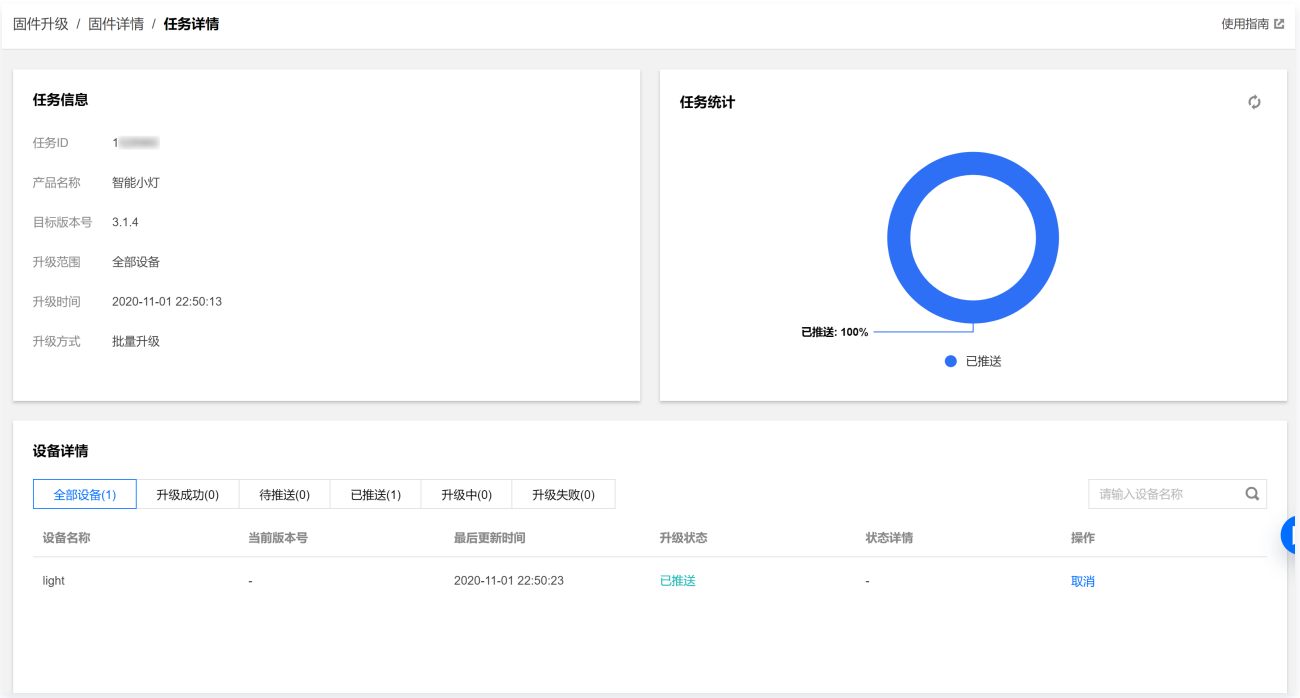
1. 在固件列表单击固件列表右侧的**查看详情**查看固件详情。

2. 进入固件详情页，可查看该固件的详细信息、固件升级设备统计和升级任务管理列表。

- 固件信息：包括固件名称、所属产品、固件版本号、固件签名、签名算法、添加时间与固件描述等。点击右上角的编辑按钮，可修改固件名称与描述。
- 固件升级设备统计：包括对该固件全部批量升级任务中的设备总数，以及不同升级状态的固件升级任务对应的设备数量。
- 任务管理列表：
  - 单击**任务明细**，可查看该固件的全部升级任务。升级任务的任务状态包含4种：未开始、创建中、创建成功、创建失败。

  - 单击**设备明细**，可查看该固件关联的所有升级任务中设备升级的记录明细。设备升级状态包含5种：待推送、已推送、升级中、升级成功和升级失败。

3. 在任务管理的**任务明细**或**设备明细**，单击某次任务右侧的**查看详情**，进入任务详情页，可以查看此次任务升级的设备清单、升级状态以及不同升级状态的设备数量统计。



在设备详情列表，可查看该任务批量升级的所有设备当前的升级状态和状态详情。

- 当升级状态为“待推送”和“已推送”时，不显示状态详情。
- 当升级状态为“升级中”，状态详情则包含：下载中、烧录中，同时显示百分比进度。
- 当升级状态为“升级失败”，状态详情将反馈错误信息。

另外，在设备详情列表右侧，可以根据升级进度进行设备升级的取消、重试操作。取消升级的设备升级状态将标记为升级失败；升级失败的设备可单击**重试**进行重新升级。

资源管理

最近更新时间：2024-12-31 18:00:53

应用场景

资源管理功能主要是用于开发者向设备端下发人脸识别库、图片库、音乐库等标准的设备资源，实现平台与设备间资源内容的上传及下载。
本文档为您介绍资源管理功能的使用方法，帮助您快速使用资源管理功能向设备进行资源下发或者将设备资源上传到物联网开发平台。

操作步骤

1. 登录 物联网开发平台控制台，选择对应实例，单击目标项目名称。默认进入**产品开发**页。
2. 选择左侧导航栏**增值服务** > **资源管理**，单击**添加资源**。

3. 输入资源名称，选择资源所属产品，上传资源及资源缩略图，同时可编辑资源描述。

4. 单击**保存**按钮后，进入资源管理页面，可以单击**资源下发**将资源下载任务下发到设备端。

5. 进入资源下发弹窗页面，下发方式分为单台下发以及批量下发，此处选择单台下发，单击**保存**。

> **说明：**
> - 单台下发：仅向所选设备下发资源下载任务。
> - 批量下发：通过上传 csv 格式文件对文件内的所有 DeviceName 对应设备下发资源下载任务，一次最多可升级10000个设备。

6. 成功创建资源下发任务后，单击**查看任务详情**进入任务详情页面：

7. 任务详情页面包括任务信息、任务统计、设备详情三部分：

- 任务 ID：该资源下发任务的任务 ID。

- 产品名称：该资源文件所属的产品。
- 目标资源：添加资源时，用户定义的资源名称。
- 下发方式：资源任务的下发方式，分为单个下发、批量下发。
- 下发时间：资源任务下发创建成功时间。
- 下发状态：包括下发失败、下发中、已推送、待推送、下发成功。

8. 设备端上线后开始下载该资源任务，下载成功后，状态变为下发成功：

具体设备端与云端的通信协议可查看 资源管理协议 文档。

设备管理

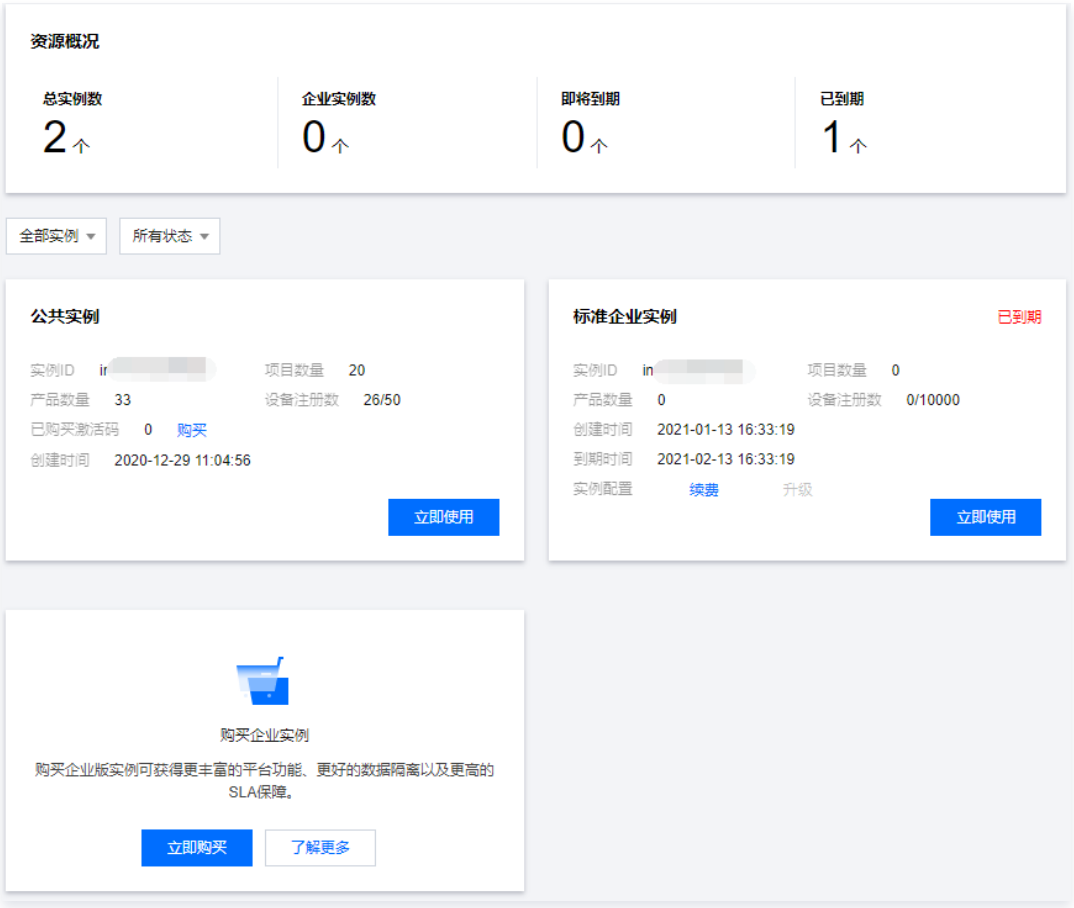
最近更新时间：2024-12-31 18:00:53

操作场景

在物联网开发平台的某个项目中成功创建设备后，您可以在控制台管理和查看具体设备信息。

管理项目下的设备

1. 登录 物联网开发平台 。
2. 在实例概览页面中，找到需要打开的公共实例或标准企业实例，单击对应实例即可跳转到实例详情页面。



3. 在实例详情页面中，找到需要打开的项目，单击项目名称即可跳转到项目详情页面。

4. 在项目详情页面中，单击左侧导航栏的**设备管理**，进入设备管理页，您可以进行以下操作：
   - 查看某个产品下的设备信息：在页面上方下拉菜单中选择某个产品。您可查看设备当前状态：
     - 未激活：设备未接入物联网开发平台。您可下载设备 SDK 进行设备开发，将设备接入物联网平台进行激活。
     - 在线：设备已激活，与物联网开发平台成功连接。

- 离线：设备已激活，与物联网开发平台断开连接。

- 搜索设备：在右侧搜索栏中选择设备名称或设备标签搜索具体设备，可支持模糊搜索。
- 查看设备详情：在列表中找到对应设备，单击**查看**进入设备详情页。
- 删除某个设备：在列表中找到对应设备，单击**删除**即可删除设备。删除设备后，该设备证书信息将会失效，设备在物联网平台上的数据记录也会被删除。

查看具体设备信息

在设备列表中，单击设备对应的**查看**，即可进入设备详情页，您可以进行以下操作：

查看设备信息

在设备详情页中，选择**设备信息**即可查看设备的基本信息，具体包括：

- 设备名称：产品下的唯一设备 ID，一般需要烧录到设备端。
- 产品ID：设备所属产品 ID，一般需要烧录到设备端。
- 所属产品：该设备所属产品名称。
- 设备密钥：平台为每个设备随机生成的密钥，使用密钥认证需要将此信息烧录到设备端。
- 设备创建时间：设备初始创建成功的时间。
- 最后上线时间：设备最后一次连接平台的时间。
- 激活时间：设备第一次成功连接到平台的时间。
- 设备状态：设备当前状态。如设备成功通过 MQTT 连接至平台，则显示“在线”，如果设备离线，则显示“离线”，如果设备从未连接平台，则显示“未激活”。

查看设备云端日志

在设备详情页中，选择**设备云端日志**即可查看该设备上行到云端，并从云端接收的信息。

查看物模型日志

当设备成功连接平台，并向物模型Topic发布消息，则可以在“设备云端日志”下的“物模型日志”查看设备上报的属性、事件、行为等数据。

说明：
当设备使用物模型协议格式上报数据，但在物模型日志无法查看到最新值时，需要确认上报的数据格式是否正确。具体可参考 物模型协议 和 物模型常见问题 。

- 查看设备属性：列表中将该设备的物模型的属性功能项全部列出。
  - 标识符：对应该设备的物模型中的标识符。
  - 功能名称：对应物模型定义中的“功能名称”。
  - 历史数据：单击**查看**，即可查询该功能项的历史上报数据。按时间展示该功能上报到云端的历史数据，验证上报的数据是否正确。
  - 数据类型：对应物模型定义中功能的“数据类型”。
  - 最新值：当设备在向云端上报数据时，只要某个功能的最新上报值发生变化，最新值列都会立刻显示设备上报的最新值。
  - 更新时间：指最新值的变化时间。一般是设备上报该功能的发生时间。
- 查看设备事件：选择物模型日志，单击**事件**即可查看该设备上报到云端的事件信息，具体包括：
  - 事件的定义：在物模型中定义管理。
  - 事件类型：系统将事件类型分为三种，分别是告警、故障、信息。

- 查看设备行为：选择物模型日志，单击**行为**即可查看该设备的行为信息，具体包括：
  - 行为的定义：在物模型中定义管理。
  - 行为描述：用于描述复杂的业务逻辑，可添加多个调用参数和返回参数，可用于让设备执行某项特定的任务。例如：开锁动作需要知道具体用户在何时开锁，以及锁的状态情况等。

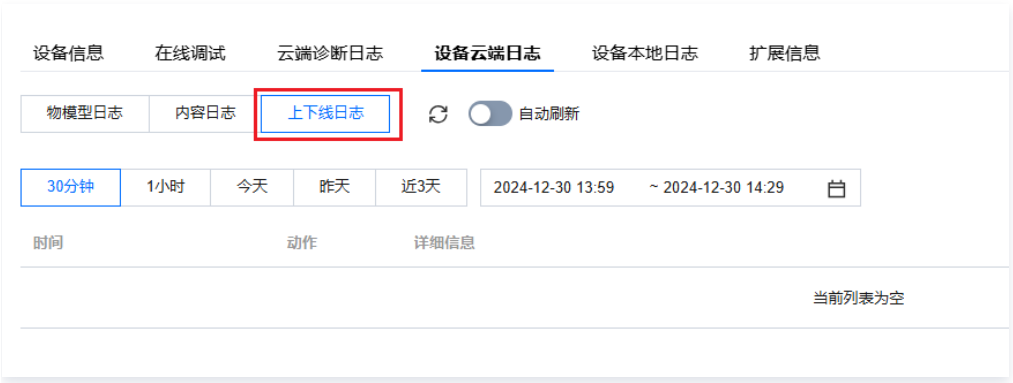
查看内容日志

1. 内容日志为用户提供了按 Topic 查询设备上下行内容日志的功能。用户选择“内容日志”，会展示“日志类型”下拉列表。

2. 日志类型选择“属性”，Topic下拉框自动加载属性对应的Topic，并查询出所选择日期范围设备与平台的所有上下行属性内容日志。用户可以按需选择不同的类型，即可查询不同的Topic所对应的上下行内容数据用于设备调试与问题定位。

查看上下线日志

上下线日志为用户提供查询设备连接到平台（上线）以及设备从平台主动或被动断开连接的日志。



查看云端诊断日志

云端诊断日志功能用于查看设备与云端交互的端到端轨迹日志，帮助用户快速诊断在调试过程中设备出现的异常错误，如订阅 topic 无权限、上行消息发布失败、规则引擎转发第三方服务失败等错误异常事件定位。该文档用于在设备与云端消息通信内容定位原因，并寻求解决方案。

在线调试

1. 当您的真实设备已成功对接到开发平台后，则可以使用在线调试对真实设备进行数据收发的测试。
2. 单击在线调试，即可进入在线调试功能。前提是真实设备已开启并成功连接到开发平台。
3. 在线调试左侧的操控面板是根据设备所属产品的物模型自动生成，设置需要下发的数据后，单击发送后，系统会自动触发控制指令到设备端。
4. 设备端接收到指令后，会立刻返回数据到云端并显示在右侧的文本框中。
5. 用户若想通过 API 下发控制指令与真实设备进行调试，可分别参考 设备远程控制 与 设备透传指令控制 两个 API，一种分别是以物模型协议下发控制指令，另一种是以自定义 Payload 方式下发指令。

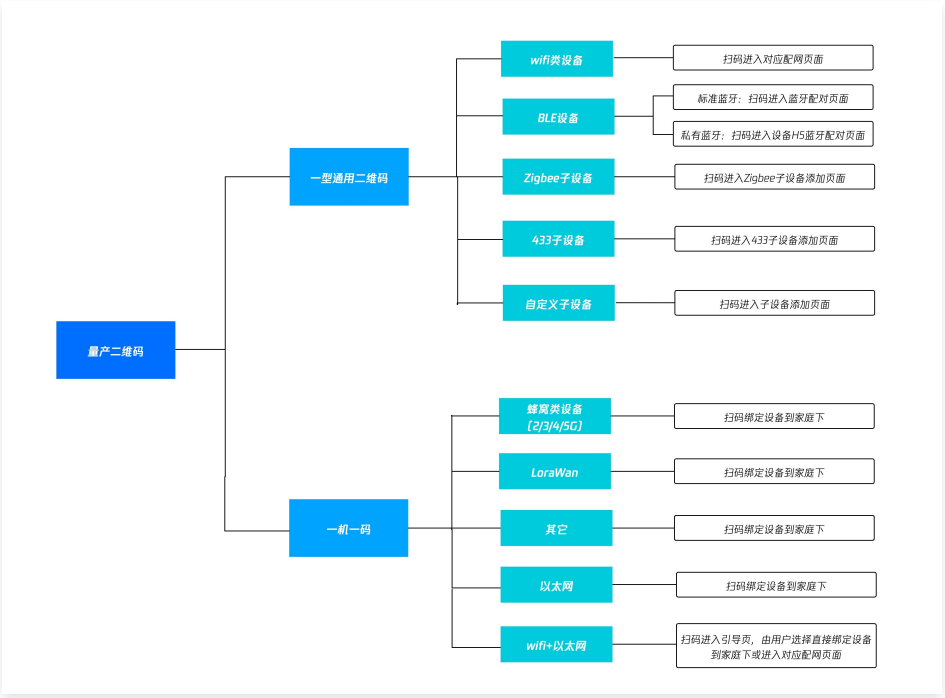
批量投产
量产二维码方案

最近更新时间：2024-08-26 14:55:51

操作场景

当设备基于物联网开发平台完成开发后，设备即将进入批量量产阶段。对于设备厂家而言，无论是使用腾讯连连小程序/通用版 App/自主品牌小程序/非官方 App，都可通过扫描二维码添加设备，从而提升用户使用体验。因此平台提供了各通信方式各品类的量产二维码方案，您可以根据设备类型，获取合适的二维码进行批量生产。

- **一型通用二维码**：表示该产品品类下所有设备均可使用同一个二维码印于包装盒或设备上。例如，生产一款 Wi-Fi 类 SmartConfig 配网的智能灯，则该款智能灯的所有设备均可使用同一个二维码进行扫码配网添加。
- **一机一码**：表示对于该产品下的设备，在量产时需生成与该设备一一对应的二维码，方可通过扫码将设备绑定到家庭下。



前提条件

- 产品已完成前序开发步骤，进入批量投产环节。
- 设备的实际生成环境，已具备量产的资质条件。

一型通用二维码

仅腾讯连连小程序、通用版 App 支持“一型通用二维码”方式，可直接通过微信扫一扫或者腾讯连连小程序内扫一扫进行设备添加，具体说明如下。

设备类型

Wi-Fi 设备、BLE 设备、Zigbee 子设备、433子设备、自定义协议子设备。

获取方式

该类设备的“一型通用二维码”，可以在 物联网开发平台控制台 > **批量投产** > **产品确认**获取。同时，您也可以单击**下载二维码**进行保存。

> **说明：**
> 如何配置扫码后所进入页面，您可单击**交互开发** > **扫一扫产品介绍/配网引导** > **配置**进行操作。

一机一码

腾讯连连小程序、通用版 App、自主品牌小程序、非官方 App 均支持“一机一码”的方式进行设备添加。本文以腾讯连连小程序为示例进行说明，其他设备可参考进行操作，具体说明如下。

设备类型

蜂窝类设备（2G/3G/4G/5G）、LoRaWAN、其它。

获取方式

在产品批量投产阶段，产品投入发布前，腾讯连连小程序和通用版 App 需物联网开发平台审核，通过审核变为可发布后，在您创建设备量产任务时，下载生成的 csv 文件获取一机一码的设备二维码内容。

> **说明：**
> 自主品牌小程序和非官网 App 无需通过物联网开发平台审核，在进行产品发布确认后，即可进行量产管理。

操作步骤

腾讯连连小程序、通用版 App

1. 登录 物联网开发平台控制台，产品在量产前需经物联网开发平台审核通过后才能发布，在产品确认阶段，需要进行单击**确认产品完成开发测试**进入申请发布阶段。

2. 填写申请发布信息，单击**申请发布**提交发布请求。

3. 审核通过后，选择左侧导航菜单**设备量产**，单击**设备管理 > 批量创建**或者单击**量产管理 > 创建量产**。

4. 进入“创建量产”界面后，选择量产产品，在“**一机一密二维码**”栏选择“自动生成”，其他参数填写详情请参见 量产管理，单击**确定**即可完成量产创建。

5. 量产创建成功后，在“量产管理”页面即可出现批次列表，单击该批次右侧**查看**，即可获取量产详情信息。

6. 在量产详情页面单击**下载设备信息**，在生成的文件中，QR-code 后的内容即为每台设备的绑定二维码内容。

自主品牌小程序、非官方 App

1. 登录 物联网开发平台控制台，在产品确认阶段，单击**开发完成并发布**进入量产管理阶段。

2. 选择左侧导航菜单**设备量产**，单击**设备管理** > **批量创建**或者单击**量产管理** > **创建量产**。

3. 进入“创建量产”界面后，选择量产产品，在“**一机一密二维码**”栏选择“自动生成”，其他参数填写详情请参见 量产管理，单击**确定**即可完成量产创建。

4. 量产创建成功后，在**量产管理**页面即可出现批次列表，单击该批次右侧**查看**，即可获取量产详情信息。

5. 在量产详情页面单击**下载设备信息**，在生成的文件中，QR-code 后的内容即为每台设备的绑定二维码内容。

量产管理

最近更新时间：2022-06-10 14:44:09

操作场景

当设备基于物联网开发平台完成开发并测试通过后，设备会进入量产阶段。待产品发布完成后，用户需要生成设备证书，烧录到设备中进行量产。
量产阶段一般至少会经过以下三个环节：

1. 设备所有者在 IoT Explorer 批量生成设备的关键信息（DeviceName、DeviceSecret）。
2. 设备所有者将生成的设备关键信息分发到授权设备厂家进行烧录。
3. 设备厂家按设备所有者的测试要求进行产测，产测通过的设备进行包装和交付，产测不通过的设备则不会包装、交付，需进一步分析产测不通过的原因。

前提条件

- 产品已完成前序开发步骤，产品已发布，进入批量投产环节。
- 设备的实际生成环境，已具备量产的资质条件。

量产步骤

1. 登录 物联网开发平台 ，选择**公共实例**或您购买的**标准企业实例**进入项目列表页面。
2. 选择具体的项目进入，单击**批量投产** > **量产管理**，即可查看当前量产的产品记录。

> **说明：**
> 首次进入量产管理，若无数据，则会在列表区域显示“暂无量产记录，请单击创建量产”。

3. 单击**创建量产**，则需要用户填写具体的产品信息和量产烧录方式。
   - **量产产品**：从下拉框选择已经发布的产品，会自动获取产品 ID。
   - **烧录方式**：可以选择一机一密和一型一密两种烧录方式，详情请参见 选择烧录方式 。
   - **生产方式**：
     - 对于一机一密烧录方式可以选择系统自动随机生成唯一的 DeviceName 和 DeviceSecret， 或者自主上传文件作为 DeviceName 并生成对应的 DeviceSecret。
     - 对于一型一密可以选择自主上传文件作为 DeviceName 并生成对应的 DeviceSecret。
   - **量产数量**：最多一次性量产10000个设备。
   - **一机一密二维码**：根据需求自行选择。

4. 选择好烧录方式后，单击**确定**，后台会对批量任务进行处理。

5. 当后台审核处理完量产任务，则会显示量产信息，提供下载批量设备信息用于厂家烧录。

选择烧录方式

烧录方式分为一机一密（直接烧录）和一型一密（动态注册），一机一密又称为直接烧录，直接烧录的量产流程一般是自己生产、制造、烧录设备的企业使用，DeviceName 与 DeviceSecret 关键信息不会透露给外部合作伙伴，降低了关键信息在分发阶段转手风险。

功能项	直接烧录	动态注册
设备烧录信息	设备证书，即：ProductID、DeviceName、DeviceSecret。	设备证书，即： ProductID、ProductSecret、DeviceName（设备名称，一般为设备本身的 MAC 地址、SN 等）。
生成方式	自动生成和上传文件。	上传文件。
量产数量	单产品下10000个设备。	单产品下10000个设备。
安全性	较高。	较低。
默认开启	是。	控制台人工开启。

一机一密（直接烧录）

直接烧录的流程也分两种：

- 第一种：系统自动生成一批 DeviceName 与 DeviceSecret。
- 第二种：设备所有者上传产品序列号作为 DeviceName，然后系统根据上传的序列号生成与之一一对应的 DeviceSecret。

系统自动生成 DeviceName 与 DeviceSecret

1. 设备所有者选择一机一密直接烧录。
2. 设备所有者选择批量生成的设备数量。
3. 后台根据设备数量自动生成唯一的 DeviceName 与 DeviceSecret。
4. 后台通过下载 CSV 文件的方式输出生成的设备信息。
5. 设备所有者下载文件后可进行具体生成过程中的烧录过程。

系统根据用户导入的 DeviceName，生成配对的 DeviceSecret

1. 设备所有者选择一机一密直接烧录。
2. 设备所有者选择文件上传方式，并导入预先准备好的文件。
3. 后台根据上传文件中第一列的数据作为 DeviceName，并自动生成对应的 DeviceSecret。
4. 后台通过下载 CSV 文件的方式输出生成的设备信息。
5. 设备所有者下载文件后可进行具体生成过程中的烧录过程。

一型一密（设备动态注册）

设备动态注册的目的是在分发阶段不会提供 DeviceSecret，只会提供 ProductID、ProductSecret、DeviceName。设备在产测环节，会根据 ProductID、ProductSecret、DeviceName 去云端动态获取对应的 DeviceSecret，设备端收到后将存储该 DeviceSecret，然后发起设备正常的登录流程。可应用于需要将设备密钥信息分发多次的场景。

控制台操作流程

1. 设备所有者选择**一型一密**动态注册的烧录方式。
2. 设备所有者选择文件上传方式，并导入预先准备好的文件。
3. 后台根据上传文件中第一列的数据作为 DeviceName。
4. 系统为选择“动态注册”的产品输出产品 Secret 参数。
5. 设备所有者将 ProductID、ProductSecret 和 DeviceName 列表文件分发至设备生产厂家。

补充说明

厂家烧录产测环节设备上线动态获取密钥的流程如下：

1. 设备生产厂家将获取到设备所有者分发的数据进行烧录。
2. 烧录完成后进行产测。
3. 设备上电后，固件程序检查本地无 DeviceSecret，则通过设备 SDK 封装好的接口送入产品 ID、产品密钥、 DeviceName，SDK 将向云端获取 DeviceSecret。
   3.1 首先对请求合法性进行签名校验。
   3.2 其次检查 DeviceName 是否已在云端存在。
   - 若不存在，则注册失败，产测也失败。
   - 若设备存在，则云端将为该 DeviceName 返回一个加密后的 DeviceSecret。
4. 设备端收到 DeviceSecret 后，进行解密并存储在本地。
5. 设备端通过动态获取的 DeviceSecret，向云端发起 MQTT 登录请求。
6. 登录成功，则表示产测的第一步通过。

规则引擎
规则引擎概览

最近更新时间：2022-06-10 14:44:20

用途

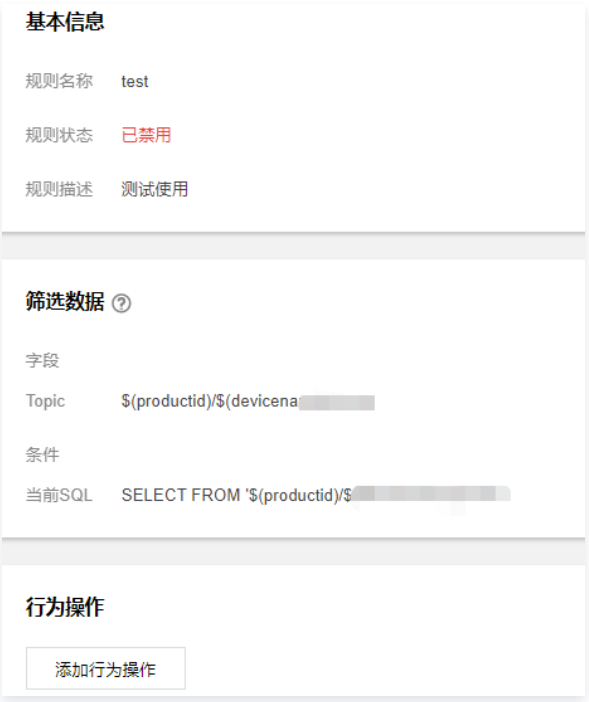
基于 Topic 进行通信时，您可以使用规则引擎对 Topic 中的数据进行处理，然后转发到腾讯云其它服务或用户的业务后台服务。您无需购买服务器部署分布式架构，只需通过规则引擎在控制台上进行配置即可实现采集 + 计算 + 存储等全栈服务。以下是支持转发的类型：

- 数据转发到另一个 Topic。
- 数据转发到第三方服务。
- 数据转发到消息队列 CKafka。
- 数据转发到时序数据库 CTSDB。
- 数据转发到云数据库 MySQL。
- 数据转发到云数据库 MongoDB。

创建规则

1. 登录 物联网开发平台控制台， 选择左侧菜单**规则引擎**。
2. 进入规则引擎页面，单击**创建规则**，填入规则名称后，单击**确定**。
   - 规则名称：支持英文、数字、下划线的组合，最多不超过32个字符。（名称新建后无法修改，请谨慎填写。）
   - 规则描述：0 - 256个字的描述，可修改。

3. 创建成功后，即可自动进入规则详情页面。



至此您可以编写不同的转发规则。

数据处理

最近更新时间：2023-07-13 17:41:23

创建完规则，即可编写 SQL 对某一类 Topic 中数据的处理。物联网开发平台提供了 SQL 填写的方式，简化 SQL 语句的生成。
将 Topic 为智能网关全部设备的物模型属性上报 Topic 中的 JSON 消息的 `action targetDevice count` 三个字段提取出来，再通过 `count <=3` 对数据进行过滤，得到最终处理过的数据，用于下一步数据转发。下图中示例表达的规则：

行为 Action

当从 Topic 提取到了感兴趣的字段后，就要考虑对其进行一些操作，例如，转发或者存储等。目前支持的操作有：

- 数据转发到另一个 Topic。
- 数据转发到第三方服务。
- 数据转发到消息队列 CKafka。
- 数据转发到时序数据库 CTSDB。
- 数据转发到云数据库 MySQL。
- 数据转发到云数据库 MongoDB。

在触发转发行为时，规则引擎会对设备上报的 payload， 进行JSON 封装，格式示例如下：

1. 转发到 Ckafka 的 JSON 示例：

```
{
  "MsgType":    "Publish",
  "Topic":      "AD4GVS5549/device/data",
  "Seq":        13107192,
  "PayloadLen": 17,
  "Payload":    "dGhpcyBpcyBhIGV4YW1wbGU=",
  "ProductId":  "AD4GVS5549",
  "DeviceName": "device",
  "Time":       "2018-08-14 15:12:05"
}
```

各字段含义如下：

- MsgType：取值有 Publish（配置消息队列转发）， Forward（命中规则引擎转发）， StatusChange（状态变化）。
- PayloadLen：设备上报消息 payload 的长度，单位为字节数。
- Payload：原始消息的 payload，默认会对内容使用 Base64 编码。
- Event：仅 StatusChange 类型消息有，目前取值为 Online 和 Offline，代表上线和下线操作。
- Time：转发行为触发的时间戳。

2. 转发到第三方服务（http forward）的 JSON 示例：

```
{
    "devicename": "device",
    "payload": {
        "params": {
            "power_switch": 1,
            "color": 1,
            "brightness": 32
        }
    },
    "productid": "AD4GVS5549",
    "seq": 2,
    "timestamp": 1587109346,
    "topic": "AD4GVS5549/device/data"
}
```

各字段含义如下：

- devicename：设备在物联网通信平台中定义的设备名称。
- Payload：原始消息的 payload，若设备原始上报格式 JSON 将透传转发；若为二进制格式，会对内容使用 Base64 编码。
- seq：内部自增的唯一消息标识符，int 类型。
- timestamp：该转发行为触发时的 Unix 时间戳。

补充说明

字段的定义

- 字段中仅支持' * '、','、'.'、空格、字母和数字，不为空，最多不超过300个字符。
- 字段表示的是 JSON 中的键值 Key，若数据格式为二进制时不可使用字段筛选，可使用' * '将所有二进制数据进行转发。
- 上报的 JSON 数据格式，可以是嵌套的 JSON。例如：{"device_status":{"switch":"on"}},可以通过 device_status.switch 来获取到 switch 的值。
- 暂不支持子 SQL 和 JSON 数组。

Topic 通配符的定义

如果想要监听多个 Topic，可以使用 `#` 和 `+` 通配符来定义多个 Topic。

- `#` 代表0个或多个任意 Topic 段，只能放在 Topic 的最后。
- `+` 代表1个任意 Topic 段，可以放在 Topic 的中间。

例如， `house_monitor/+/get` ：

- 可监听 `house_monitor/thermometer/get` 和 `house_monitor/door/get` 等 Topic。
- 但不可监听 `house_monitor/door/switch/get` ，因为 `+` 只能代表1个 Topic 段。

例如， `house_monitor/#` ：

- 可监听 `house_monitor/thermometer` 和 `house_monitor/door/switch/get` 等 Topic。
- 但 `house/#/get` 是非法的，因为 `#` 只能放在 Topic 结尾。

条件的定义

[条件]表达式用于过滤 Topic 中的消息，只有当消息满足[条件]表达式时，才会被提取并进行后续的处理，支持的表达式见下表：

操作符	描述	举例

=	相等	color = 'red'
<>	不等于	color <> 'red'
AND	逻辑与	color = 'red' AND siren = 'on'
OR	逻辑或	color = 'red' OR siren = 'on'
( )	括号代表一个整体	color = 'red' AND (siren = 'on' OR siren ='isTest')
+	算术加法	age = 4 + 5
-	算术减	age = 5 - 4
/	除	age = 20 / 4
*	乘	age = 5 * 4
%	取余数	age = 0 % 6
<	小于	5 < 6
<=	小于或等于	5 <= 6
>	大于	6 > 5
>=	大于或等于	6 >= 5

规则函数

最近更新时间：2024-08-26 14:55:51

规则引擎提供多种函数，您可以在规则引擎的字段，条件以及数据库字段对应的值中使用这些函数，实现数据的多样化处理。

支持的函数

函数名	用法描述
productId()	返回消息来源的产品 ID。
deviceName()	返回消息来源的设备名字。
timestamp()	返回当前的 Unix 系统时间戳，秒为单位。
topic()	返回消息来源的原始 Topic。
topic(n)	返回消息来源的原始 Topic 以 `/` 分割的第 n 个分段。
payloadLen()	返回 payload 的字节长度。
bin_to_dec()	将二进制数 data 转换为十进制整数。
to_hex ()	将输入的原始消息转换为16进制字符串。
randint(min,max)	返回 min 和 max 之间的随机整数。
upper(string)	返回大写字符串（输入的消息格式需为 JSON 格式，函数对象为对应的 key 值。例如输入消息为 `"tencent":"iot"`，则 `upper(tencent)=IOT`）。
lower(string)	返回小写字符串（输入的消息格式需为 JSON 格式，函数对象为对应的 key 值）。
crypto(field,String)	对 field 的值进行加密，第二个参数 String 为算法字符串。可选：MD5，SHA1，SHA256，SHA384，SHA512。（输入的消息格式需为 JSON 格式，函数对象为对应的 key 值）。
concat(string1, string2)	字符串连接，例如 `concat(deviceid,'a')` 或 `concat(field1,field2)`。
requestId()	返回物联网通信生成的消息 ID。
newuuid()	返回一个随机 UUID 字符串。
replace(source, substring, replacement)	replacement 替换 source 中的 substring。
substring(source, start, end)	字符串截取，返回从 start（包括）到 end（不包括）的字符串。

使用示例

某一家居温湿度设备 dev00 向云端发送的消息内容为：

```
{"room1":{"temperature":31,"humidity":"63%"},
 "room2":{"temperature":26,"humidity":"63%"}}
```

温湿度产品下有 dev00，dev01，dev02 三个设备分别监测了 room1，room2，room3...room6 六个房间的温湿度，只有当 room1 房间的温度高于30摄氏度时需要对数据转入 MySQL 数据库进行处理。依照此案例规则引擎的设置如下：

筛选数据

字段	room1.temperature as temp,room1.humidity as hum
Topic	[redacted]/+/data
条件	topic(2) = 'device00' AND room1.temperature > 30
当前SQL	SELECT room1.temperature as temp,room1.humidity as hum FROM [redacted]'+/data' WHERE topic(2) = 'device00' AND room1.temperature > 30

编辑规则

行为将数据插入到云数据库（MySQL）中，查看文档

行为类型

数据转发到云数据库（MySQL）

地域 *　　实例 *

Mysql数据库 *　　数据表 *

实例登录账户 *　　登录密码 *

数据字段

字段名称	值
table_temperature	${temp}
table_humidity	${hum}
productId	productId()
deviceName	deviceName()

保存　取消

数据转发到另一 Topic

最近更新时间：2023-05-31 10:59:37

概述

通过将业务需求所需要的消息字段转发到另一个 Topic，即可实现不同设备间的 M2M 通信。Topic 的填写支持以下方式：

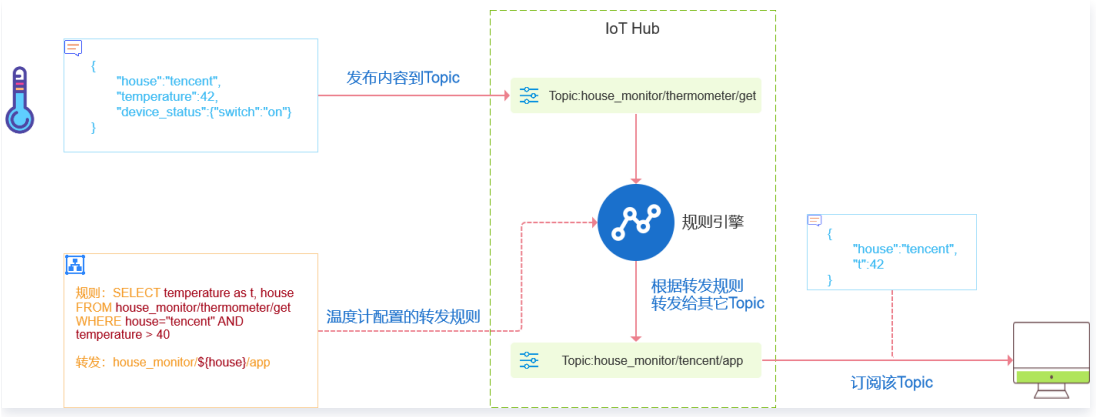
- **填写一个 Topic 名字**
  例如 `${productId}/house_monitor/thermometer`，即可将满足规则的消息转发到这个 Topic。
- **填写带变量的 Topic 名字**
  例如 `${productId}/${house}/device`，其中用 `${}` 括起来的 `house` 就代表一个变量名，这个变量名是 SELECT 语句中选取出来的字段内容。

示例说明

该示例主要说明带变量的转发 Topic 是如何生效的。假设定义了一条规则，示例如下：

```
SELECT temperature as t, house
FROM house_monitor/thermometer/get
WHERE house="tencent" AND temperature > 40
```

此规则从消息中提取了 `t` 和 `house` 这两个字段的值，假定 `house` 字段的内容为 `tencent` 。
此时如果定义了转发给 `house_monitor/${house}/app` 这个 Topic，那么规则引擎则会将这个 Topic 中的 `${house}` 变量替换为 “tencent”， 从而将 `t` 和 `house` 的字段内容发送给 `house_monitor/tencent/app` 这个 Topic。
转发全过程如下图所示：



配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**，单击需要配置的规则。
2. 在规则详情页面，单击**添加行为操作**。
3. 在弹出的“添加规则”窗口，填写相关信息。单击**保存**即可。
   - 选择行为类型为“数据转发到另一个Topic（Republish）”。
   - 选择相应产品及设备。
   - 选择要转发到的另一个 Topic 类型及名称。

物联网开发平台即可将设备上报数据发转至该 Topic。

转发消息服务质量等级

消息从源 Topic 转发到其它 Topic 时消息服务质量等级不会变化。

- 设备端发布的消息服务质量等级为 QOS0 时则规则引擎将按照 QOS0 的消息进行转发，发布的消息服务质量等级为 QOS1 时则按照 QOS1 进行转发。
- 转发的消息服务质量等级为0时，若转发失败则消息会被丢弃；转发的消息服务质量等级为1，若消息转发失败则会进行转发重试。重试按照3s，6s，9s的时间间隔依次进行三次，若三次重试均失败则将消息保存在离线消息队列。

数据转发到第三方服务

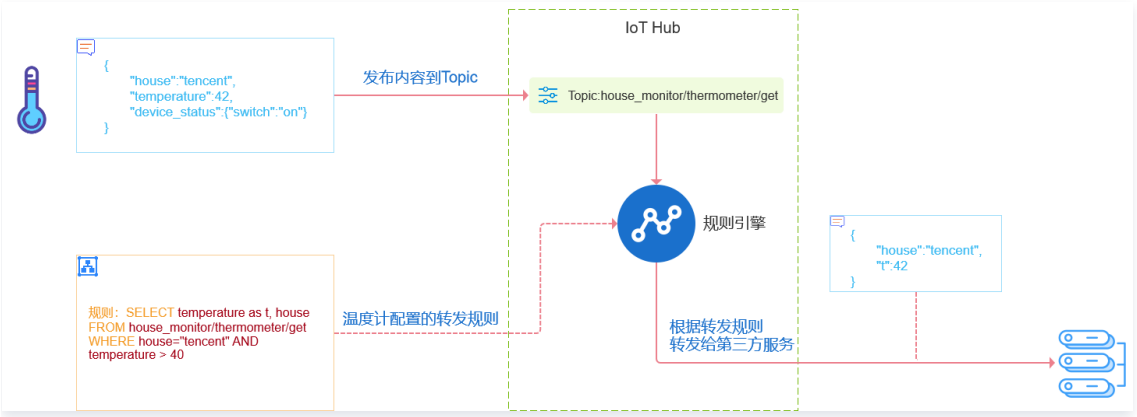
最近更新时间：2023-07-13 17:41:23

概述

将通过规则提取出来的设备数据转发给第三方服务时，您可自定义如何处理这些数据。这种方式是提供给用户灵活性最高的一种消息处理方式。

> **注意：**
> 第三方服务必须以 HTTP 或 HTTPS 的方式提供服务。配置转发第三方服务，需要提供支持 HTTP 或 HTTPS 的网站 URL 和端口。规则引擎转发成功后，第三方服务将收到来自 `42.193.134.62` 的数据包。

下图展示了将数据转发给第三方服务的整个过程：



转发的数据内容和格式，请参见 数据处理 文档。

填写服务器配置

1. 登录 物联网开发平台控制台， 单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 单击所要配置的规则。进入规则详情页面，单击**添加行为操作**。
3. 在弹出的“添加规则”窗口，填写相关信息。单击**保存**即可。
   - 选择行为类型为“数据转发到第三方服务（Forward）”。
   - 选择 API 地址类型，可选“使用已有 HTTP 服务地址”或者“使用物联使能服务地址（推荐）”。
   - 填写您的 HTTP 或 HTTPS 服务地址。物联网开发平台会将设备上报的数据转发至 HTTP 或 HTTPS 服务地址。
   - 请勾选 “增加鉴权 Token”，且填写您的服务对应的 Token；您可以任意填写 Token，用作生成签名（该 Token 会和接口 URL 中包含的 Token 进行比对，从而验证安全性）。

验证消息来自物联网开发平台

> **注意：**
> 为了您后台稳定使用，请选择增加鉴权 Token。

请求标识

用户如果在转发到第三方服务（Forward）即 HTTP 转发，已选择“增加鉴权 Token”，物联网开发平台将在 HTTP 或 HTTPS 请求中头部增加如下字段：

参数	描述
Signature	Signature 结合了“添加规则”中填写的 Token 参数和请求中的 Timestamp 参数、Nonce 参数。
Timestamp	时间戳。
Nonce	随机数。

1. 将 Token、Timestamp、Nonce 三个参数进行字典序排序。
2. 将三个参数字符串拼接成一个字符串进行 sha1 加密。
3. 开发者获得加密后的字符串可与 Signature 对比，标识该请求来源于物联网开发平台。

检验 Signature 的 PHP 示例代码如下：

```
private function checkSignature()
{
    $signature = $_GET["signature"];
    $timestamp = $_GET["timestamp"];
    $nonce = $_GET["nonce"];

    $token = TOKEN;
    $tmpArr = array($token, $timestamp, $nonce);
    sort($tmpArr, SORT_STRING);
    $tmpStr = implode( $tmpArr );
    $tmpStr = sha1( $tmpStr );
```

```
    if( $tmpStr == $signature ){
        return true;
    }else{
        return false;
    }
}
```

例如某次请求，相关参数如下，用户设置 Token 为 aaa。

```
Nonce: IkOaKMDalrAzUTxC
Signature: c259ed29ec13ba7c649fe0893007401a36e70453
Timestamp: 1604458421
```

排序后的字符串是 `1604458421IkOaKMDalrAzUTxCaaa`，最终计算 sha1 结果为 `c259ed29ec13ba7c649fe0893007401a36e70453`。

服务地址校验

1. 当开启规则引擎时，物联网开发平台将发送一次 GET 请求到填写的服务器地址 URL 上，GET 请求头部增加如下字段：

参数	描述
Signature	Signature 结合了“添加规则”中填写的 Token 参数和请求中的 Timestamp 参数、Nonce 参数。
Timestamp	时间戳。
Nonce	随机数。
Echostr	随机字符串。

物联网开发平台向第三方服务发送报文示例：

```
GET / HTTP/1.1
Host: **.**.**.**:4443
User-Agent: Go-http-client/1.1
Content-Type: application/json
Echostr: UPWIAFASvDUFcTEE
Nonce: testrance
Signature: abb6c316a8134596d825c5a1295bfa6f7657664d
Timestamp: 1623149590
Accept-Encoding: gzip
```

2. 第三方服务若确认此次 GET 请求来自物联网开发平台，请在 body 中原样返回 Echostr 参数内容。
第三方服务回复物联网开发平台报文示例：

```
HTTP/1.1 200 OK
Date: Tue, 08 Jun 2021 10:53:10 GMT
Content-Length: 16
Content-Type: text/plain; charset=utf-8

UPWIAFASvDUFcTEE
```

3. 物联网开发平台校验返回的 Echostr 参数内容，确认服务器地址 URL 是否有效。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接受消息的目的，具体说明如下：

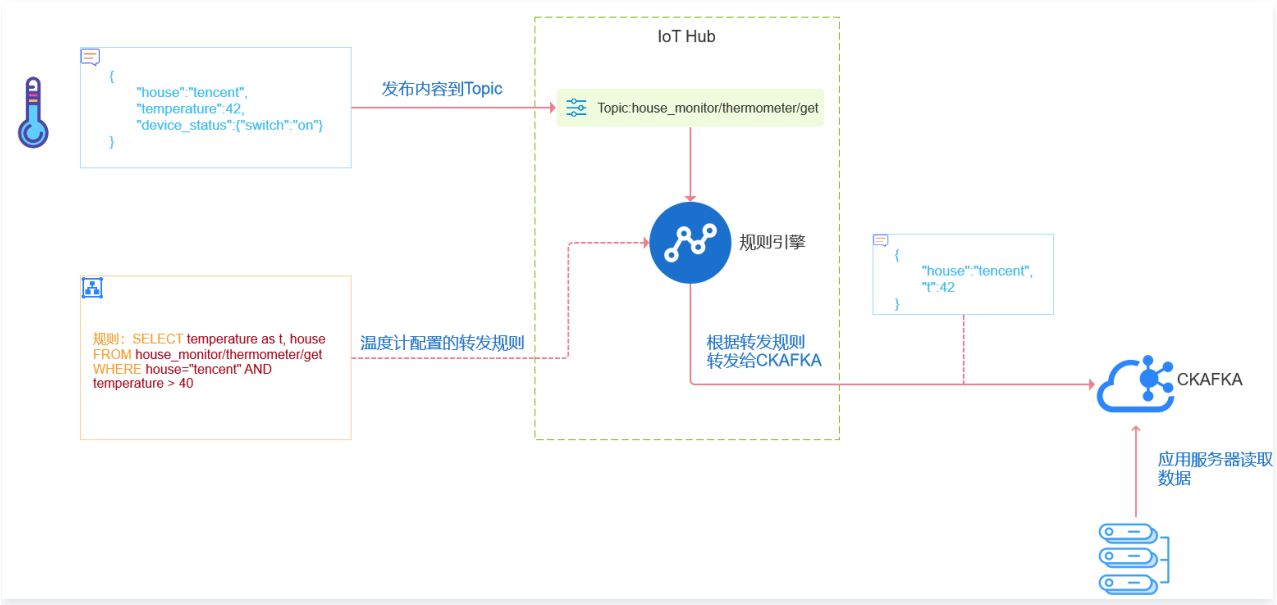
- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

数据转发到消息队列 CKAFKA

最近更新时间：2023-05-24 17:12:56

概述

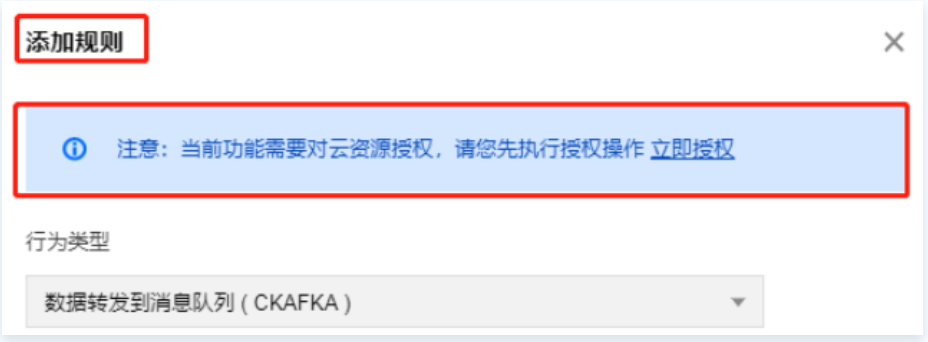
规则引擎支持用户配置规则将符合条件的设备上报数据转发到 消息队列 CKAFKA （以下简称 CKAFKA ），用户的应用服务器再从 CKAFKA 中读取数据内容进行处理。以此利用 CKAFKA 高吞吐量的优势，为用户打造高可用性的消息链路。
下图展示了规则引擎将数据转发给 CKAFKA 的整个过程：



配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则引擎页面，单击需要配置的规则。
3. 在规则详情页面，单击**添加行为操作**。

> **说明：**
> 第一次使用时会提示用户授权访问 CKAFKA，您需单击**授权访问 CKAFKA**才能继续创建。



4. 在弹出的“添加规则”窗口，选择行为“数据转发到消息队列（CKAFKA）”；依次选择 CKAFKA 实例和 Topic，单击**保存**即可。

5. 完成以上配置后，物联网开发平台会将符合规则条件的设备上报数据转发至用户配置的 CKAFKA 。您可以参考 创建实例和 Topic 文档，在应用服务器上读取数据并进行处理。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接受消息的目的，具体说明如下：

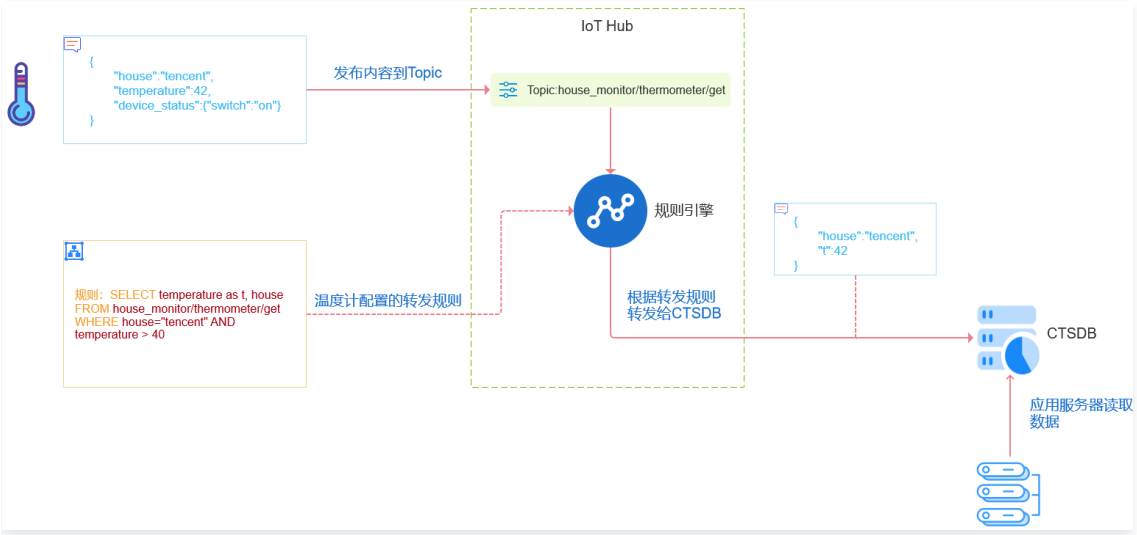
- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

数据转发到时序数据库

最近更新时间：2023-05-31 10:59:37

概述

规则引擎支持用户配置规则将符合条件的设备上报数据转发到 时序数据库 CTSDB （以下简称 CTSDB。当前版本仅支持 CTSDB 1.0，暂不支持 CTSDB 2.0，支持时间另行通知），用户的应用服务器，再从 CTSDB 中读取数据内容进行处理。以此利用 CTSDB 海量数据高存储压缩率、数据聚合展示能力，能有效满足日常设备数据存储、分析、可视化展示的需求。
规则引擎将数据转发给 CTSDB 的整个过程，如下图所示：



配置步骤

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则详情页面，单击**添加行为操作**。

> **说明：**
> 第一次使用时会提示用户授权访问 CTSDB，用户需单击**授权访问 CTSDB**才能继续创建。
>
> 
>

3. 在弹出的“添加规则”窗口，选择行为**数据转发到时序数据库（CTSDB）**，依次选择 CTSDB 地域和实例，并填写基本信息和需要配置的转发字段，单击**保存**即可。

完成以上配置后，物联网开发平台会将符合规则条件的设备上报数据，转发至用户配置的 CTSDB 实例。用户可参考 CTSDB 开发指南 在自己的应用服务器上读取数据进行处理，或者在 CTSDB 控制台 对数据进行聚合检索查询。

配置参数说明

- **实例登录账户**：用户创建 CTSDB 实例时候输入的账户名，需要在配置规则引擎之前创建实例。
- **登录密码**：用户创建 CTSDB 实例时候输入的账户密码，需要在配置规则引擎之前创建实例。
- **metric**：配置数据转发到 CTSDB 的哪个 metric 下，如果配置规则引擎之时没有该 metric ，物联平台会自行创建。
- **timestamp**： 数据写入 CTSDB 时候的时间戳，当前支持4种配置：
  - 通过“${}”引用原始消息的字段值。
  - 系统函数。
  - timestamp()：命中规则引擎的该消息当前时间，插入当前的系统时间。
  - 常量：需要是以秒为单位的 UNIX 时间戳；不填写，则默认为命中规则引擎的该消息当前时间。

> **注意：**
> 如果在规则创建后，用户修改该 CTSDB metric 的 timestamp 为非秒级的单位（如毫秒级），可能导致后续数据写入失败。

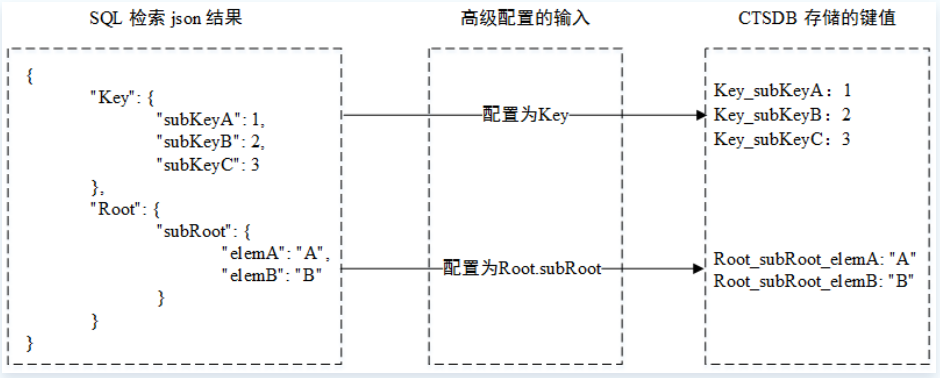
- **数据字段**： 类型可选择 CTSDB 里面的 tag 类型或者 field 类型，字段名称输入限制，请参见 CTSDB 限制，值有3种配置方式：通过“${}”引用原始消息的字段值；常量；固定值。

高级配置说明

高级配置项，适用于设备上报数据字段是动态扩展的，无法预先配置的情况。例如，设备底下有若干传感器需要传输数据，但是不同的设备规格、配置不一样，传感器的数目也不固定，但需要使用规则引擎配置，将设备底下所有传感器的数据都存入 CTSDB，以下为您提供高级配置的方案存储：

说明：

- 默认存储类型：动态扩展存储的字段，在 CTSDB 中的存储类型，默认是 tag 类型。
- key：需要遍历扩展存储的 json 键，物联网开发平台会遍历此 key 下的 json 键值嵌套，以'_'为连接符，最后存储到时序数据库，通过规则引擎配置 SQL SELECT 检索得到的 json 结果与配置（支持配置子 key，支持配置多个），到实际存储进 CTSDB 的数据，如下样例所示：



重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接受消息的目的，具体说明如下：

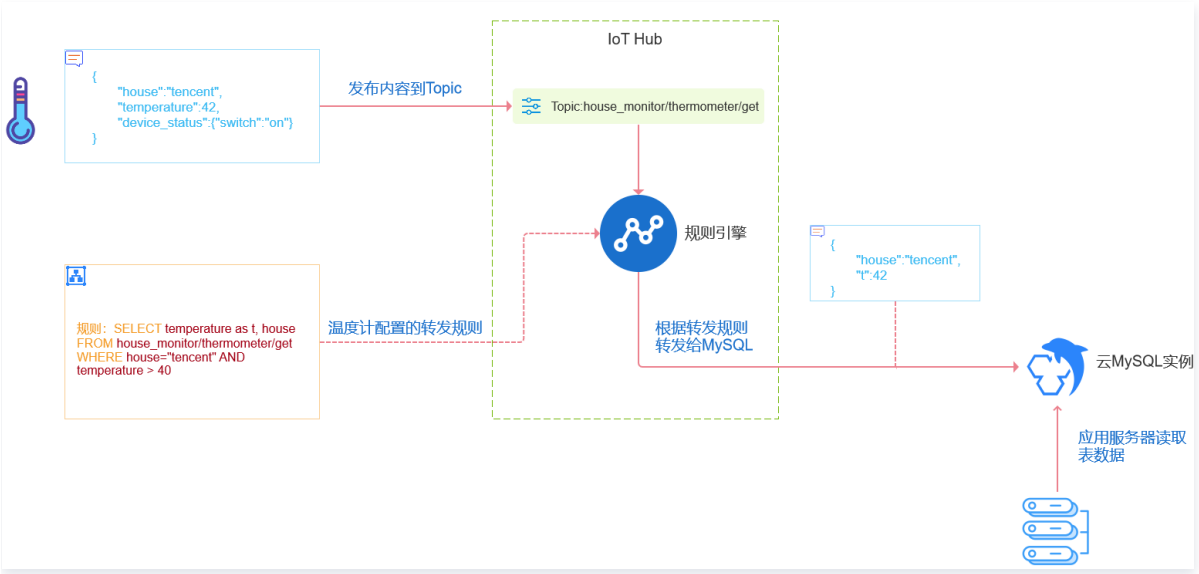
- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

数据转发到云组件 MySQL

最近更新时间：2024-08-26 14:55:51

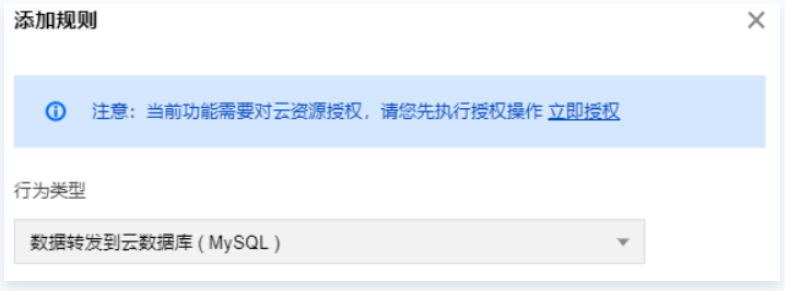
概述

规则引擎支持用户配置转发规则，将符合条件的设备上报数据转发到云组件 MySQL ，您可以在 MySQL 控制台 或者使用云 API 创建 MySQL 实例和表后，即可将设备消息中的指定字段写入到对应的 MySQL 表中。
下图展示了规则引擎将数据转发给 MySQL 的整个过程：



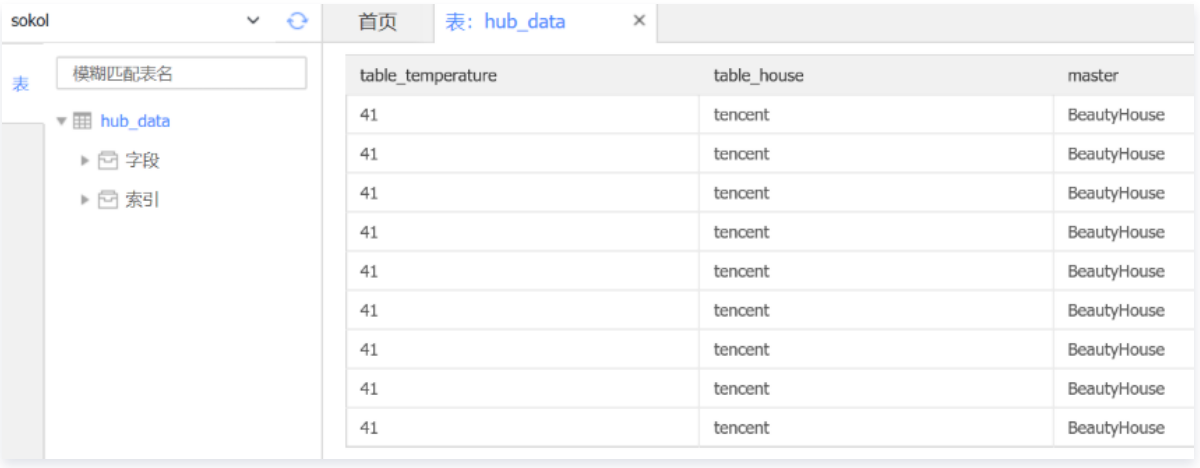
配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则引擎页面，单击需要配置的规则。
3. 在规则详情页面，单击**添加行为操作**。

> **说明：**
> 第一次使用时会提示用户授权访问 MySQL，您需单击**立即授权**才能继续创建。
>
> 
>

4. 在弹出的“添加规则”窗口，选择“数据转发到云数据库（MySQL）选项”，授权成功后，需要配置 MySQL 实例信息和写入的字段信息，如下图所示。配置完成后单击**保存**即可。

转发成功后,MySQL 中显示的信息如下图所示：



配置说明

配置分为如下几个步骤：

1. 选择地区和 MySQL 实例。
2. 输入刚创建的 MySQL 实例的用户名。
3. 输入实例的登录密码。
4. 选择需要写入的数据库名。如果创建的 MySQL 实例下还没有建立数据库，请前往 MySQL 控制台创建一个新的数据库。具体操作请参见 建立数据库和表 。
5. 选择要写入的表。如果创建的数据库下还没有建立表，前往 MySQL 控制台创建一个新的表。

6. 配置要写入的字段。这里有两列："字段名称"和"值"。"字段名称"对应的是数据库表中的字段，表示要写入的字段。"值"表示要写入对应字段的值。值的来源可以是消息体（注意消息体必须是 Json 格式才支持提取值），或者是在这里填入常量。

> ⚠ **注意：**
> - 如果来源是消息体，那么使用“${}”来引用消息体内的字段。如果要指定常量，直接填相应的值就行了，例如5或者 hello 这样的数字或者字符串字面值。
> - 需先在云组件 MySQL 中创建完成数据库，表以及字段名称之后才可成功将数据写入数据库。

更多详情请参见 建立数据库和表 。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接收消息的目的，具体说明如下：

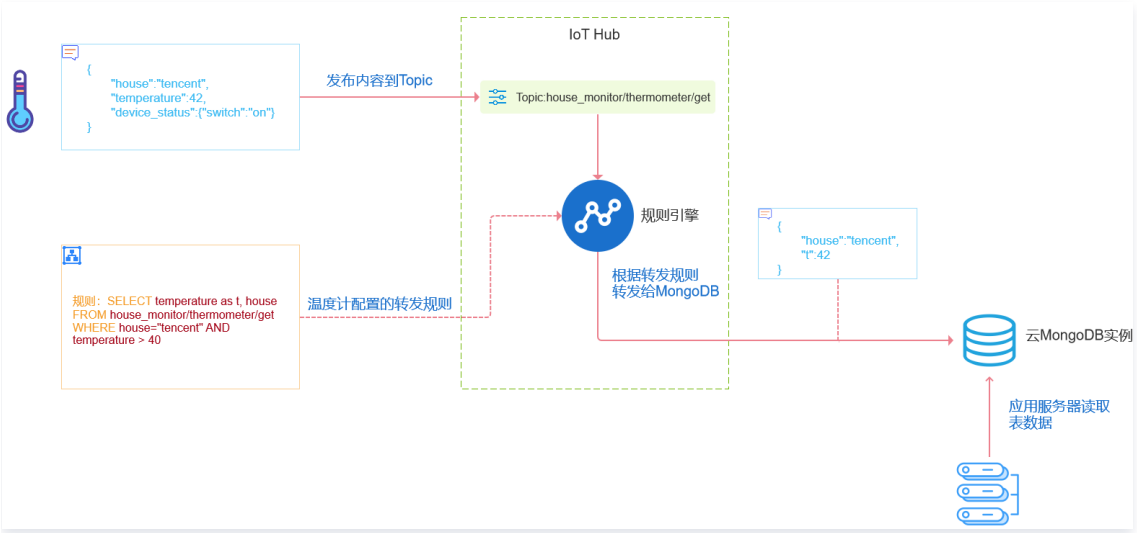
- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

数据转发到云组件 MongoDB

最近更新时间：2024-08-26 14:55:51

概述

规则引擎支持用户配置转发规则，将符合条件的设备上报数据转发到云组件 MongoDB ，您可以在 MongoDB 控制台 或者使用云 API 创建 MongoDB 实例后，即可将设备消息写入到对应的 MongoDB 集合中。
下图展示了规则引擎将数据转发给 MongoDB 的整个过程：



配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则引擎页面，通过单击“规则名称”选择需要配置的规则。
3. 在添加规则页面，单击**添加行为操作**。

说明：
第一次使用时会提示用户授权访问 MongoDB，您需单击**立即授权**才能继续创建。

4. 进入行为类型选择页面，选择“数据转发到云数据库（MongoDB）选项”。

5. 授权成功之后，需要配置 MongoDB 实例信息，如下图所示，配置分为如下几个步骤：

5.1 选择地区和 MongoDB 实例。如果账号下还没有实例，单击**创建实例**跳转到 MongoDB 控制台创建一个。

5.2 输入 MongoDB 实例的用户名，MongoDB 官网默认 mongouser。

5.3 输入 MongoDB 实例的登录密码。

5.4 输入要写入的数据库名。

5.5 输入要写入的集合名。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接收消息的目的，具体说明如下：

- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

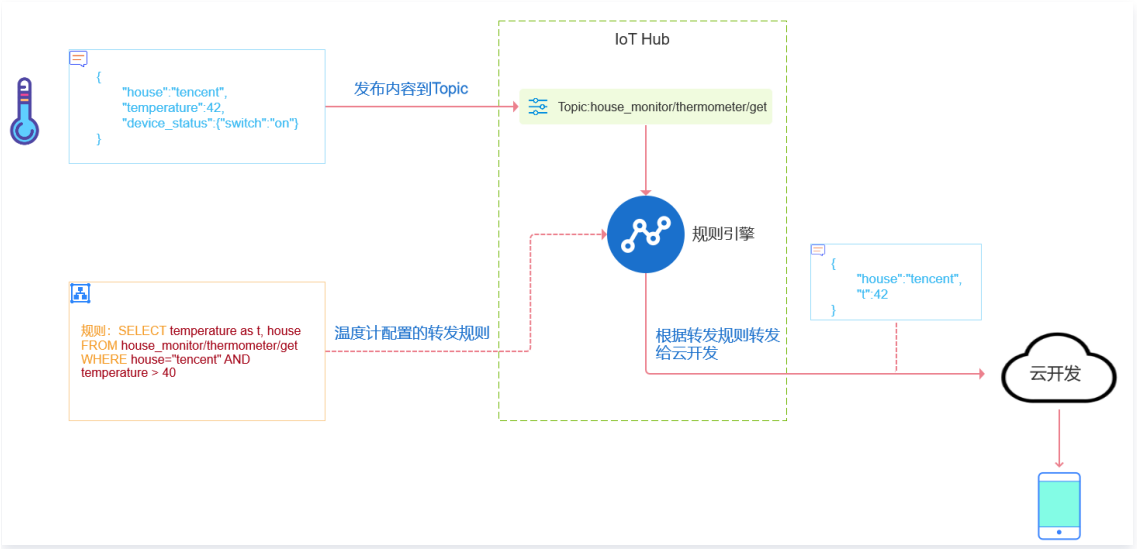
数据转发到云开发

最近更新时间：2024-08-26 14:55:51

概述

规则引擎支持用户配置转发规则，将符合条件的设备上报数据转发到云开发组件，您可以在 云开发 CloudBase 控制台 完成云开发环境的开通，具体操作请参见 开通环境 。

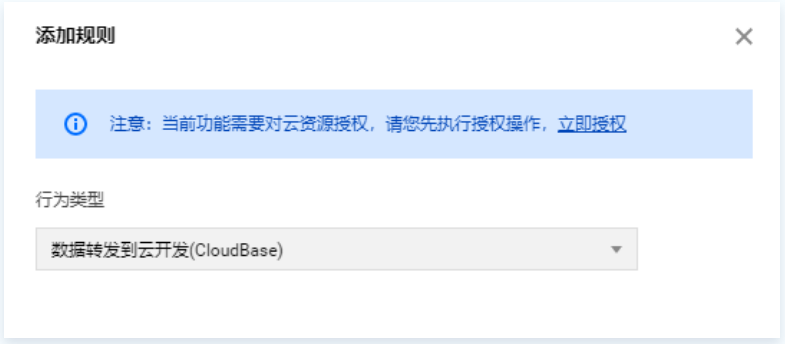
下图展示了规则引擎将数据转发给云开发的整个过程：



配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则引擎页面，单击需要配置的规则。
3. 在规则详情页面，单击**添加行为操作**。

> **说明：**
> 第一次使用时会提示用户授权访问云开发，您需单击**立即授权**才能继续创建。



4. 在弹出的“添加规则”窗口，选择“数据转发到云开发（CloudBase）选项”，选择建立好的环境与函数后，单击**保存**即可。

> **注意：**
> 目前仅支持数据往云开发中云函数的转发，需先在云开发中建立好开发的环境与云函数，才可对环境与云函数进行选择。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接收消息的目的，具体说明如下：

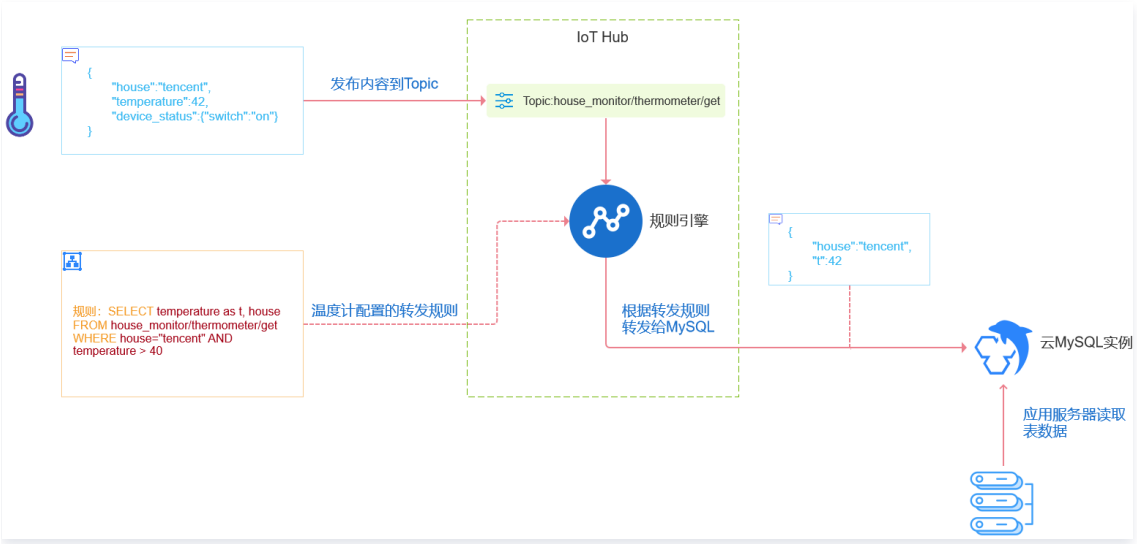
- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

数据转发到云组件 TDSQL-MySQL

最近更新时间：2024-08-26 14:55:52

概述

规则引擎支持用户配置转发规则，将符合条件的设备上报数据转发到云组件 TDSQL-MySQL ，您可以在 TDSQL 控制台 或者使用云 API 创建 TDSQL 实例和表后，即可将设备消息中的指定字段写入到对应的 TDSQL 表中。
下图展示了规则引擎将数据转发给 TDSQL 的整个过程：



配置

1. 登录 物联网开发平台控制台，单击目标实例和项目名称，选择左侧菜单栏**数据流转 > 规则引擎**。
2. 进入规则引擎页面，单击需要配置的**规则名称**进入规则详情页面。
3. 单击**添加行为操作**。若是首次使用时会提示用户授权访问 TDSQL，您需单击**立即授权**才能继续创建。

4. 授权成功后，在弹出的“添加规则”窗口，选择“数据转发到云数据库TDSQL-MySQL”，需要配置 TDSQL-MySQL 实例信息和写入的字段信息，如下图所示。

5. 配置完成后单击**保存**即可。

配置说明

配置 分为以下几个步骤：

1. 选择地域和 TDSQL-MySQL 实例。
2. 输入刚创建的 TDSQL-MySQL 实例的用户名。
3. 输入实例的登录密码。
4. 选择需要写入的数据库名。如果创建的 TDSQL-MySQL 实例下还没有建立数据库，请前往 TDSQL-MySQL 控制台创建一个新的数据库。
5. 选择要写入的表。如果创建的数据库下还没有建立表，前往 TDSQL-MySQL 控制台创建一个新的表。
6. 配置要写入的字段。这里有两列："字段名称"和"值"。"字段名称"对应的是数据库表中的字段，表示要写入的字段。"值"表示要写入对应字段的值。值的来源可以是消息体（注意消息体必须是 Json 格式才支持提取值），或者是在这里填入常量。

> **注意：**
> - 如果来源是消息体，那么使用 “${}”来引用消息体内的字段。如果要指定常量，直接填写相应的值即可，例如5或者 hello 这样的数字或者字符串字面值。
> - 需先在云组件 TDSQL-MySQL 中创建完成数据库，表以及字段名称之后才可成功将数据写入数据库。

重发机制

重发机制用于在消息转发过程中发生失败的情况下，进行再次重发以达到接收消息的目的，具体说明如下：

- 若消息转发失败，系统则会进行转发重试，重试按照1s、3s、10s的时间间隔依次进行，若三次重试均失败，则将消息丢弃掉。
- 若用户配置了“转发错误行为操作”，在三次重试失败后，将按“转发错误行为操作”的配置，再进行一次消息转发，如果仍失败，则将消息丢弃掉。

运营分析

最近更新时间：2024-08-26 14:55:52

操作场景

利用运营分析功能实现对项目内所有产品的激活、活跃、在线的设备数据进行统计分析；以及对产品的活跃设备和激活设备地域分布进行统计分析。

前提条件

存在激活设备、在线设备及连接平台的设备。

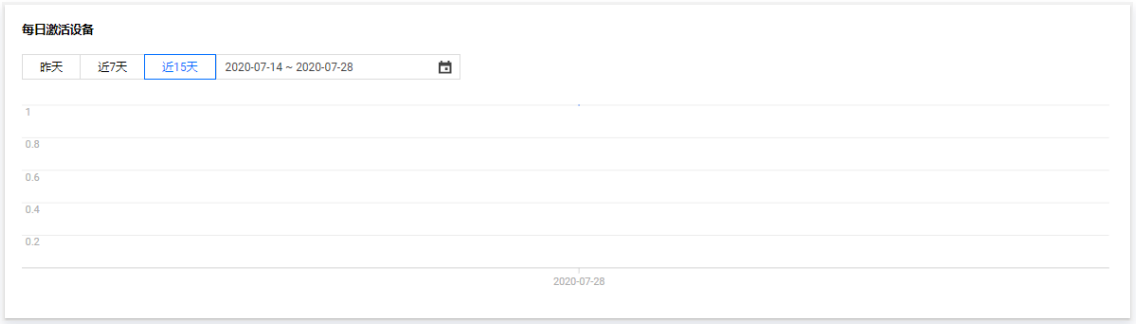
设备概览

设备概览可查看所有产品或通过筛选某个产品来查看该产品的激活、在线、活跃相关数据，以及每日激活设备、每日活跃设备、在线设备数据信息。

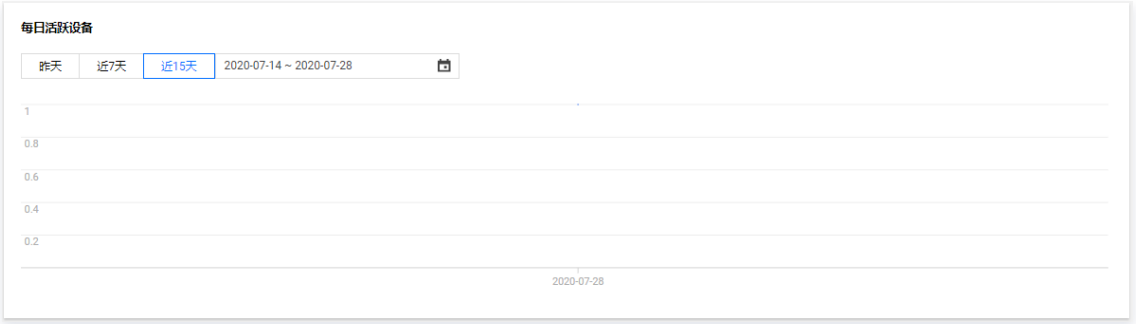
操作步骤

1. 登录 物联网开发平台控制台 ，地域选择“中国区”，单击项目进入项目详情页面，单击**运营分析**，单击**设备概览**进入设备概览界面。
2. 选择需要查看的产品，页面将显示对应的激活设备总数、在线设备数、昨日激活数、昨日活跃数、近7日激活数、近7日活跃数、上7日激活数、上7日活跃数。

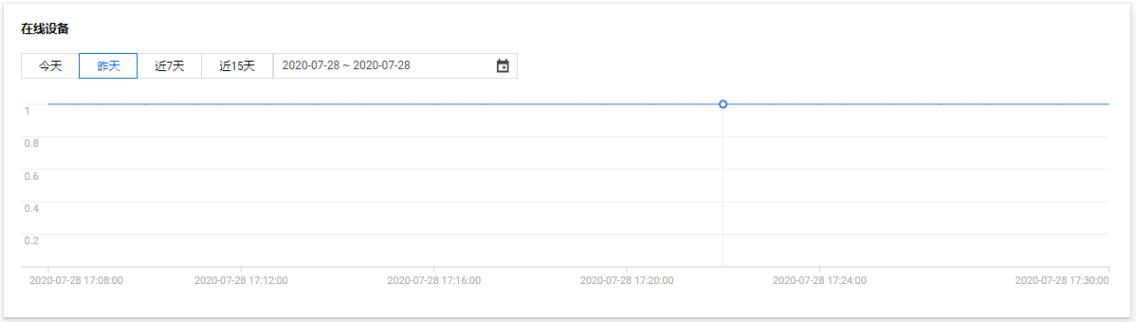
3. 选择需要查看激活设备的时间段，系统将会以图表的形式展示该产品在筛选时间段内每日的激活设备数量。



4. 选择需要查看活跃设备（此设备需为已连接腾讯云物联网平台的设备）的时间段，系统将会以图表的形式展示该产品在筛选时间段内每日的活跃设备数量。



5. 选择需要查看在线设备的时间段，系统将会以图表的形式展示该产品在筛选时间段内每日的在线设备数量。



设备分布

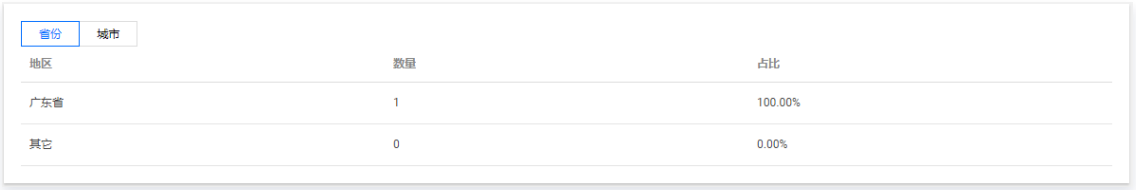
设备分布可查看所有产品，也可筛选某个产品在设定时间内不同地域的活跃设备和激活设备数量，系统提供地图和表格两种展示方式。

操作步骤

1. 登录 物联网开发平台控制台 ，地域选择“中国区”，单击项目进入项目详情页面，单击**运营分析**，单击**设备分布**进入设备分布界面。
2. 选择需要查看的产品、时间段以及设备类型。

3. 系统将根据用户选的产品、时间段以及设备类型等信息，在地图上显示不同地域的设备分布数量，直观地向用户展示各地域的设备分布数量。

4. 系统同时提供表格展示方式，用户可通过表格查看每个省份及城市的详细数量及占比等详细信息。



增值服务
增值服务开通

最近更新时间：2024-12-31 18:00:53

腾讯云物联网开发平台 人脸识别 、实时音视频 、语音识别 、语音助手 功能为付费增值服务，如需使用，请 提交申请 进行业务咨询，工作人员将会与您对接服务购买事宜，详情请参见 增值服务开通申请，以下为您详细介绍申请流程。

操作步骤

1. 登录 物联网开发平台控制台 ，地区选择“**中国区**”并创建项目及产品，详情请参见 产品定义 。
2. 选择已创建的项目进入项目详情页，单击左侧导航菜单**产品开发**进入产品列表页。
3. 选择已创建的产品进入产品详情页，在右侧菜单栏**增值服务**中选择需要开通的增值功能。
4. 当鼠标悬浮在对应功能的开启按钮上时，页面将会弹出提示，您可在弹出的提示框内单击**工单**，进入实时互动-物联版意向客户联络。

5. 在实时互动-物联版意向客户联络页，填写您的相关信息，并单击**提交**。

实时互动-物联版为各行业的设备制造商、方案商及应用开发商提供一站式设备智能化服务。平台提供海量设备连接与管理能力及小程序应用开发能力，并打通腾讯云基础产品及 AI 能力，提升传统行业设备智能化的效率，降低用户的开发运维成本，助力用户业务发展。

您是*

个人 企业

您所在企业名称*

请填入您现在所处的企业名称

企业类型*

芯片模组商 硬件终端商 应用开发商 解决方案商 其他

行业分类*

公共服务 商业场所 生活场所 工业制造 农林牧渔 建筑工地 其他

企业业务场景*

请填入您的企业业务场景描述

业务阶段*

未开发 开发中 测试中 已上线 大规模应用中

联系人*

请填入您的真实姓名

联系电话*

请填入可以联系到您的电话

联系邮箱*

请填入可以联系到您的邮箱

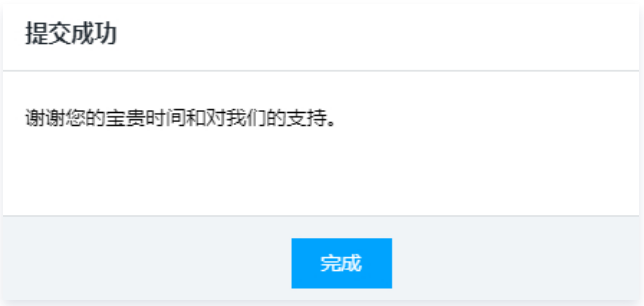
您需要咨询的内容*

请填入您的业务需求、问题描述或使用反馈，我们将尽快联系您

我已阅读和同意《腾讯云隐私保护声明》内容

提交

6. 申请提交成功后，工作人员将会在7个工作日内与您联系服务购买事宜。



版本变更

最近更新时间：2022-06-10 14:46:11

本页主要提供与增值服务相关的 SDK 版本变更记录。

人脸识别

设备端 Android SDK

版本 V3.2.1

- 发布日期：2020/12/10
- 开发语言：Java
- 开发环境：Android Studio
- 内容如下：
  - 底层通信模块复用 hub-device-java。
  - IoT Explorer 下增加人脸识别 SDK 以及 demo。
  - 增加设置自建服务的 brokerUrl 以及 CA 证书的接口。
  - 增加网关子设备升级功能。
  - 设备 SDK 支持 websocket-MQTT 协议。
  - 拓扑关系管理。
  - 修复若干问题。

设备端 C SDK

版本 V1.0.0

- 发布日期：2020/12/8
- 开发语言：C 语言
- 开发环境：Linux/Windows，Cmake/GNU Make
- 内容如下：
  - 支持人脸检索。
  - 支持人脸库在线更新。
  - 支持人脸检索事件上报。
  - 提供示例和文档。

实时音视频

设备端 SDK

版本 V3.3.0

- 发布日期：2021/01/14
- 系统平台：Android
- 开发语言： Java
- 开发环境：Android Studio
- 更新内容：
  - explorer 下增加 RTC 场景通话 SDK 以及 demo。
  - 修复若干问题。

版本 V1.1.1

- 发布日期：2021/01/14
- 开发语言：C 语言
- 开发环境：Linux/Windows，Cmake/GNU Make

- 更新内容：
  - 实现与应用端音视频通话
  - 修复若干问题

应用端 SDK

版本 V1.3.0

- 发布日期：2021/01/14
- 系统平台：iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境：Mac，Xcode/ Android Studio
- 更新内容：优化实时音视频信令逻辑，提升用户体验。

版本 V1.2.1

- 发布日期：2020/12/10
- 系统平台：iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境：Mac，Xcode/ Android Studio
- 更新内容：修复 V1.2.0 版本的 SDK 配置，删除 i386 架构。

版本 V1.2.0

- 发布日期：2020/12/02
- 系统平台：iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境：Mac，Xcode/ Android Studio
- 更新内容：提供接入实时音视频通话场景需求能力。

语音识别

设备端 SDK sample

版本 V3.1.5

- 发布日期：2020/12/2
- 开发语言：C 语言
- 开发环境：Linux/Windows，Cmake/GNU Make
- 内容如下：
  - 新增资源管理功能及示例。
  - 新增 ASR 功能及示例。
  - 新增文件操作 HAL 层适配接口。
  - 优化多线程操作。
  - MQTT 示例实现数据模板协议数据交互。
  - 版本号修改为 V3.1.5

实时音视频

最近更新时间：2024-12-31 18:00:53

本文为您介绍如何开通与使用腾讯云物联网开发平台实时音视频服务。

限制条件

仅支持在以下地区激活设备开通实时音视频服务：
中国大陆（不含港澳台地区）

申请开通实时音视频服务

实时音视频服务为付费增值业务，不提供免费试用，您可以进行 在线咨询 来寻求帮助，工作人员将会与您对接服务购买事宜。

操作步骤

步骤一：开通实时音视频服务

1. 登录 物联网开发平台控制台 ，地区选择“**中国区**”并创建项目及产品，详情请参见 产品定义 。
2. 选择已创建的项目进入项目详情页，默认进入左侧导航菜单**产品开发**页，可查看产品列表。
3. 选择已创建的产品进入产品详情页，在右侧**增值服务**菜单栏中，单击“实时音视频”处按钮即可开通成功。

步骤二：创建设备

单击控制台左侧导航菜单**产品开发**，选择产品进入产品详情页，单击**设备调试** > **新建设备**，填写相关信息单击**保存**即可。

注意：

- 当产品下存在设备时，不可更改增值服务开关状态。
- 若删除已开通增值服务的设备，所购买 License 数量不可恢复。

设备接入指引

腾讯云物联网开发平台实时音视频服务应用端提供小程序插件及 iOS、Android App SDK，设备端现支持 Android 平台、海思 DV300 平台。

小程序插件

- 小程序插件使用指南，详情请参见 小程序插件使用指南 。

- 插件的名词解释和接口使用说明，详情请参见 腾讯连连小程序插件使用说明 。

SDK 获取

- 应用端 SDK 使用 Github 托管，可访问 Github 下载最新版本 Android SDK 、 iOS SDK 。
- 设备端 SDK 使用 Github 托管，可访问 Github 下载最新版本 Android SDK 、 Linux C SDK（请提交意向单咨询）。

开发指南

请参见以下指南：

- 应用端 Android SDK 接入指南
- 应用端 iOS SDK 接入指南
- 设备 Android SDK 接入指南
- 设备端 C SDK 接入指南（请提交意向单咨询）

语音识别

最近更新时间：2024-11-26 15:00:53

本文为您介绍如何开通与使用腾讯云物联网开发平台语音识别服务。

限制条件

仅支持在以下地区激活设备开通语音识别服务：
中国大陆（不含港澳台地区）

申请开通语音识别服务

语音识别服务为付费增值业务，不提供免费试用，您可以进行 在线咨询 来寻求帮助，工作人员将会与您对接服务购买事宜。

语音识别服务接入指南

步骤一：开通语音识别服务

1. 登录 物联网开发平台控制台 ，地区选择“**中国区**”并创建项目及产品，详情请参见 产品定义 。
2. 选择已创建的项目进入项目详情页，单击左侧导航菜单**产品开发**进入产品列表页。
3. 选择已创建的产品进入产品详情页，单击**数据模板**，在右侧**增值服务**菜单栏中，单击“语音识别”处按钮即可开通成功。

步骤二：创建设备

单击控制台左侧导航菜单**产品开发**，选择产品进入产品详情页，单击**设备调试** > **新建设备**，填写相关信息单击**保存**即可。

注意：

- 当产品下存在设备时，不可更改增值服务开关状态。
- 若删除已开通增值服务的设备，所购买 License 数量不可恢复。

设备接入指引

腾讯云物联网开发平台（IoT Explorer）使用语音识别（Automatic Speech Recognition，ASR）产品功能，并结合 C SDK 的 ASR 示例 `asr_data_template_sample` 快速体验 ASR 功能。

配置设备信息

修改设备信息配置 `device_info.json`，将创建的 ASR 的产品和设备信息对应填入，获取设备信息可参考 设备调试 。

将 ASR 结果返回依赖的数据模板属性修改为系统属性，与用户的数据模板解耦示例代码如下：

```
{
        "auth_mode":"KEY",
        "productId":"PRODUCT_ID",                       //ASR产品ID
        "productSecret":"YOUR_PRODUCT_SECRET",
        "deviceName":"YOUR_DEV_NAME",                   //ASR设备名
        "key_deviceinfo":{
            "deviceSecret":"YOUR_IOT_PSK"                 //ASR设备密钥
        },
        "cert_deviceinfo":{
            "devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
            "devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
        },
        "region":"china"
}
```

修改编译选项

修改 CMakeLists.txt（以**密钥认证设备**为例）使能 ASR 和资源管理功能（ASR 功能依赖资源管理功能）有以下两种方式：

- 使用 cmake 编译：

```
set(BUILD_TYPE                    "release")
set(COMPILE_TOOLS                 "gcc")
set(PLATFORM                     "linux")
set(FEATURE_RESOURCE_UPDATE_ENABLED ON)
set(FEATURE_ASR_ENABLED ON)
set(FEATURE_AUTH_MODE "KEY")
```

执行脚本编译：

```
./cmake_build.sh
```

- 使用 makefile 编译：

```
PLATFORM_CC                     = gcc
PLATFORM_AR                     = ar
PLATFORM_OS                     = linux
FEATURE_RESOURCE_UPDATE_ENABLED       = y
```

```
FEATURE_ASR_ENABLED    = y
```

执行 make 编译：

```
make
```

`asr_data_template_sample` 示例输出位于 `output/release/bin` 文件夹中。

示例说明

- 使用 ASR 功能，需要进行以下操作：
  1.1 调用接口 `IOT_Asr_Init` 初始化 `asr_client` 。
  1.2 如果是文件或者一句话识别，则调用 `IOT_Asr_RecordFile_Request` ，传入请求参数、文件名及回调即可；如果是实时语音，则调用 `IOT_Asr_Realtime_Request` ，传入实时音频数据、请求参数及回调，同时，音频数据的编码需要在发起请求前完成。两个接口的调用成功的情况下返回值是 `request_id` ，对应的结果会在回调中返回，返回结果会带上对应的 `request_id` ，即 ASR 的结果返回只支持异步。
- `asr_data_template_sample` 展示了 ASR 在文件、一句话和实时语音三种使用场景下如何使用上述 API，通过修改示例的宏定义 `DEMO_ASR` 选择对应的示例场景，三个场景的数据来源都使用测试文件 `tools/test_file/test.wav` ，编译后，该文件将会被拷贝到 `output/release/bin/test_file/test.wav` ，如果需要使用自己的测试文件，则可以通过替换该测试文件即可。

```
#define DEMO_ASR_FILE                   0
#define DEMO_ASR_REALTIEM               1
#define DEMO_ASR_SENTENCE               2
#define DEMO_ASR                        DEMO_ASR_SENTENCE
```

使用场景说明

ASR 的文件、一句话、实时语音三种使用场景及请求参数，详情可请参见 ASR 官网文档 。

运行说明

- 将 `DEMO_ASR` 配置为 `DEMO_ASR_FILE` ，对应示例 ASR 文件识别的使用场景：

```
./asr_data_template_sample
INF|2020-11-10 16:24:19|qcloud_iot_device.c|iot_device_info_set(55): SDK_Ver: 3.1.4, Product_ID:
W0MHQCSFN5, Device_Name: dev002
INF|2020-11-10 16:24:19|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: q4ZhF success
INF|2020-11-10 16:24:19|asr_data_template_sample.c|event_handler(88): subscribe success, packet-id=2717
INF|2020-11-10 16:24:19|data_template_client.c|IOT_Template_Construct(936): Sync device data
successfully
INF|2020-11-10 16:24:19|asr_data_template_sample.c|main(379): Cloud Device Construct Success
INF|2020-11-10 16:24:19|asr_data_template_sample.c|_register_data_template_property(227): data template
property=asr_response registered.
INF|2020-11-10 16:24:19|asr_data_template_sample.c|main(394): Register data template propertys Success
INF|2020-11-10 16:24:20|asr_data_template_sample.c|event_handler(88): subscribe success, packet-id=2718
INF|2020-11-10 16:24:22|asr_data_template_sample.c|main(492): record file test.wav's request_id 1
INF|2020-11-10 16:24:24|asr_data_template_sample.c|main(492): record file test.wav's request_id 2
INF|2020-11-10 16:24:26|asr_data_template_sample.c|main(492): record file test.wav's request_id 3
INF|2020-11-10 16:24:27|asr_data_template_sample.c|main(492): record file test.wav's request_id 4
INF|2020-11-10 16:24:29|asr_data_template_sample.c|main(492): record file test.wav's request_id 5
INF|2020-11-10 16:24:31|asr_data_template_sample.c|main(492): record file test.wav's request_id 6
INF|2020-11-10 16:24:32|asr_data_template_sample.c|asr_result_cb(340): request_id:3: 1/1 text:
[0:0.000,0:2.800]  北京科技馆。
INF|2020-11-10 16:24:32|asr_data_template_sample.c|asr_result_cb(340): request_id:2: 1/1 text:
[0:1.040,0:3.100]  北京科技馆。
INF|2020-11-10 16:24:33|asr_data_template_sample.c|main(492): record file test.wav's request_id 7
INF|2020-11-10 16:24:35|asr_data_template_sample.c|main(492): record file test.wav's request_id 8
INF|2020-11-10 16:24:36|asr_data_template_sample.c|asr_result_cb(340): request_id:1: 1/1 text:
[0:0.000,0:2.800]  北京科技馆。
```

```
INF|2020-11-10 16:24:36|asr_data_template_sample.c|asr_result_cb(340): request_id:4: 1/1 text:
[0:0.000,0:2.800]  北京科技馆。
```

- 将 `DEMO_ASR` 配置为 `DEMO_ASR_REALTIEM` ，对应示例 ASR 一句话识别的使用场景：

```
./asr_data_template_sample
INF|2020-11-10 16:21:00|qcloud_iot_device.c|iot_device_info_set(55): SDK_Ver: 3.1.4, Product_ID:
W0MHQCSFN5, Device_Name: dev002
INF|2020-11-10 16:21:00|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: oLbdC success
INF|2020-11-10 16:21:00|asr_data_template_sample.c|event_handler(88): subscribe success, packet-
id=53209
INF|2020-11-10 16:21:00|data_template_client.c|IOT_Template_Construct(936): Sync device data
successfully
INF|2020-11-10 16:21:00|asr_data_template_sample.c|main(379): Cloud Device Construct Success
INF|2020-11-10 16:21:00|asr_data_template_sample.c|_register_data_template_property(227): data template
property=asr_response registered.
INF|2020-11-10 16:21:00|asr_data_template_sample.c|main(394): Register data template propertys Success
INF|2020-11-10 16:21:01|asr_data_template_sample.c|event_handler(88): subscribe success, packet-
id=53210
INF|2020-11-10 16:21:03|asr_data_template_sample.c|main(508): record file test.wav's request_id 1
INF|2020-11-10 16:21:04|asr_data_template_sample.c|main(508): record file test.wav's request_id 2
INF|2020-11-10 16:21:05|asr_data_template_sample.c|main(508): record file test.wav's request_id 3
INF|2020-11-10 16:21:06|asr_data_template_sample.c|main(508): record file test.wav's request_id 4
INF|2020-11-10 16:21:06|asr_data_template_sample.c|asr_result_cb(340): request_id:1: 1/1 text:北京科技
馆。
INF|2020-11-10 16:21:07|asr_data_template_sample.c|main(508): record file test.wav's request_id 5
INF|2020-11-10 16:21:07|asr_data_template_sample.c|asr_result_cb(340): request_id:2: 1/1 text:北京科技
馆。
INF|2020-11-10 16:21:07|asr_data_template_sample.c|asr_result_cb(340): request_id:3: 1/1 text:北京科技
馆。
INF|2020-11-10 16:21:08|asr_data_template_sample.c|main(508): record file test.wav's request_id 6
INF|2020-11-10 16:21:08|asr_data_template_sample.c|asr_result_cb(340): request_id:4: 1/1 text:北京科技
馆。
```

- 将 `DEMO_ASR` 配置为 `DEMO_ASR_SENTENCE` ，对应示例 ASR 实时语音识别的使用场景：

```
./asr_data_template_sample
INF|2020-11-10 16:25:39|qcloud_iot_device.c|iot_device_info_set(55): SDK_Ver: 3.1.4, Product_ID:
W0MHQCSFN5, Device_Name: dev002
INF|2020-11-10 16:25:39|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: 1tBx2 success
INF|2020-11-10 16:25:39|asr_data_template_sample.c|event_handler(88): subscribe success, packet-
id=45977
INF|2020-11-10 16:25:39|data_template_client.c|IOT_Template_Construct(936): Sync device data
successfully
INF|2020-11-10 16:25:39|asr_data_template_sample.c|main(379): Cloud Device Construct Success
INF|2020-11-10 16:25:39|asr_data_template_sample.c|_register_data_template_property(227): data template
property=asr_response registered.
INF|2020-11-10 16:25:39|asr_data_template_sample.c|main(394): Register data template propertys Success
INF|2020-11-10 16:25:40|asr_data_template_sample.c|event_handler(88): subscribe success, packet-
id=45978
INF|2020-11-10 16:25:42|asr_data_template_sample.c|main(556): realtime request_id 1
INF|2020-11-10 16:25:43|asr_data_template_sample.c|main(556): realtime request_id 2
INF|2020-11-10 16:25:44|asr_data_template_sample.c|main(556): realtime request_id 3
INF|2020-11-10 16:25:44|asr_data_template_sample.c|asr_result_cb(340): request_id:1: 0/0 text:NULL
INF|2020-11-10 16:25:45|asr_data_template_sample.c|main(556): realtime request_id 4
INF|2020-11-10 16:25:45|asr_data_template_sample.c|asr_result_cb(340): request_id:2: 1/0 text:NULL
INF|2020-11-10 16:25:46|asr_data_template_sample.c|main(556): realtime request_id 5
INF|2020-11-10 16:25:46|asr_data_template_sample.c|asr_result_cb(340): request_id:3: 2/0 text:NULL
INF|2020-11-10 16:25:46|asr_data_template_sample.c|main(556): realtime request_id 6
```

```
INF|2020-11-10 16:25:46|asr_data_template_sample.c|asr_result_cb(340): request_id:4: 3/0 text:NULL
INF|2020-11-10 16:25:47|asr_data_template_sample.c|main(556): realtime request_id 7
INF|2020-11-10 16:25:47|asr_data_template_sample.c|asr_result_cb(340): request_id:5: 4/0 text:NULL
INF|2020-11-10 16:25:48|asr_data_template_sample.c|main(556): realtime request_id 8
INF|2020-11-10 16:25:48|asr_data_template_sample.c|asr_result_cb(340): request_id:6: 5/0 text:NULL
INF|2020-11-10 16:25:49|asr_data_template_sample.c|main(556): realtime request_id 9
INF|2020-11-10 16:25:49|asr_data_template_sample.c|asr_result_cb(340): request_id:7: 6/0 text:NULL
INF|2020-11-10 16:25:50|asr_data_template_sample.c|main(556): realtime request_id 10
INF|2020-11-10 16:25:50|asr_data_template_sample.c|asr_result_cb(340): request_id:8: 7/0 text:NULL
INF|2020-11-10 16:25:51|asr_data_template_sample.c|main(556): realtime request_id 11
INF|2020-11-10 16:25:51|asr_data_template_sample.c|asr_result_cb(340): request_id:9: 8/0 text:NULL
INF|2020-11-10 16:25:52|asr_data_template_sample.c|main(556): realtime request_id 12
INF|2020-11-10 16:25:52|asr_data_template_sample.c|asr_result_cb(340): request_id:10: 9/0 text:NULL
INF|2020-11-10 16:25:53|asr_data_template_sample.c|main(556): realtime request_id 13
INF|2020-11-10 16:25:53|asr_data_template_sample.c|asr_result_cb(340): request_id:11: 10/0 text:NULL
INF|2020-11-10 16:25:53|asr_data_template_sample.c|main(556): realtime request_id 14
INF|2020-11-10 16:25:53|asr_data_template_sample.c|asr_result_cb(340): request_id:12: 11/0 text:NULL
INF|2020-11-10 16:25:54|asr_data_template_sample.c|main(556): realtime request_id 15
INF|2020-11-10 16:25:54|asr_data_template_sample.c|asr_result_cb(340): request_id:13: 12/0 text:北京科
技。
INF|2020-11-10 16:25:55|asr_data_template_sample.c|main(556): realtime request_id 16
INF|2020-11-10 16:25:55|asr_data_template_sample.c|asr_result_cb(340): request_id:14: 13/0 text:北京科
技。
INF|2020-11-10 16:25:56|asr_data_template_sample.c|main(556): realtime request_id 17
INF|2020-11-10 16:25:56|asr_data_template_sample.c|asr_result_cb(340): request_id:15: 14/0 text:北京科技
馆。
INF|2020-11-10 16:25:57|asr_data_template_sample.c|main(556): realtime request_id 18
INF|2020-11-10 16:25:57|asr_data_template_sample.c|asr_result_cb(340): request_id:16: 15/0 text:NULL
INF|2020-11-10 16:25:58|asr_data_template_sample.c|main(556): realtime request_id 19
INF|2020-11-10 16:25:58|asr_data_template_sample.c|asr_result_cb(340): request_id:17: 16/0 text:NULL
INF|2020-11-10 16:25:59|asr_data_template_sample.c|main(556): realtime request_id 20
INF|2020-11-10 16:25:59|asr_data_template_sample.c|asr_result_cb(340): request_id:18: 17/0 text:NULL
INF|2020-11-10 16:26:00|asr_data_template_sample.c|main(556): realtime request_id 21
```

设备端 SDK sample 获取

设备端 SDK sample 使用 Github 托管，可访问 Github 下载最新版本 设备端 SDK sample 。

自定义 H5 面板开发

语音识别的设备支持自定义 H5 控制面板开发，可在腾讯连连小程序进行设备控制，详情请参见 H5 自定义开发 ASR 语音识别 。

语音助手

最近更新时间：2024-11-01 09:45:32

本文为您介绍如何开通与使用腾讯云物联网开发平台语音助手服务。

限制条件

仅支持在以下地区激活设备开通语音助手服务：
中国大陆（不含港澳台地区）

申请开通语音助手服务

语音助手服务为付费增值业务，不提供免费试用，您可以进行 在线咨询 来寻求帮助，工作人员将会与您对接服务购买事宜。

语音助手服务接入指引

步骤一：开通语音助手服务

1. 登录 物联网开发平台控制台 ，地区选择“**中国区**”并创建项目及产品，详情请参见 产品定义 。

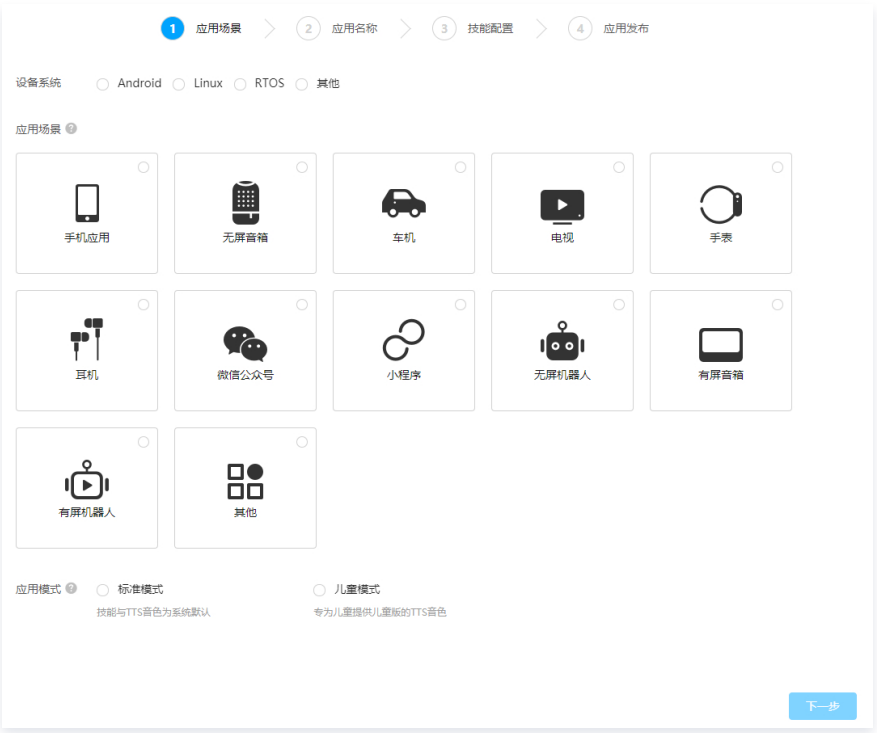
> **说明：**
> 新建产品时，建议将产品品类选择为：智能生活\影音办公\智能音响。开发者也可以选择其他品类，进行自定义开发。

2. 选择已创建的项目进入项目详情页，单击左侧导航菜单**产品开发**进入产品列表页。
3. 单击已创建的产品名称进入产品详情页，在右侧**增值服务**菜单栏中，单击“语音助手”处按钮即可开通成功。

步骤二：绑定云小微开放平台应用信息

> **注意：**
> 请务必保证 App Key、App Secret 不被泄露。

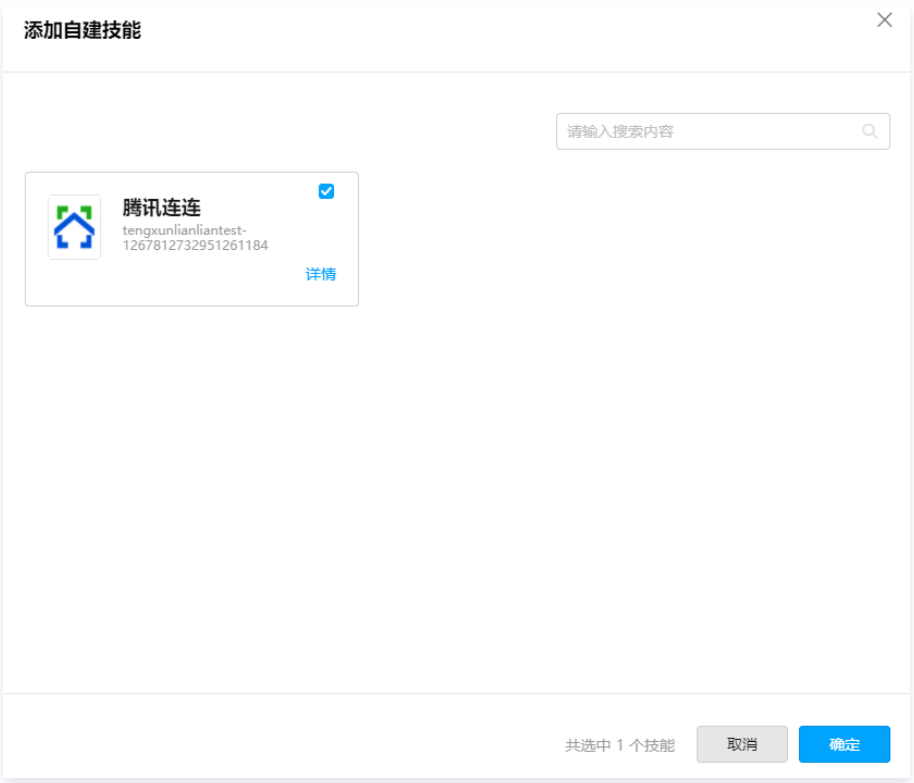
1. 登录 腾讯云小微 开放平台，单击**设备平台 > 新建应用**进入应用新建页面，填写相关信息。



- **设备系统：包含 Android、Linux、RTOS 和其他。**
- **应用场景：包含手机应用、无屏音响、车机等，本次选“无屏音响”。**
- **应用模式：包含“标准模式”和“儿童模式”。**

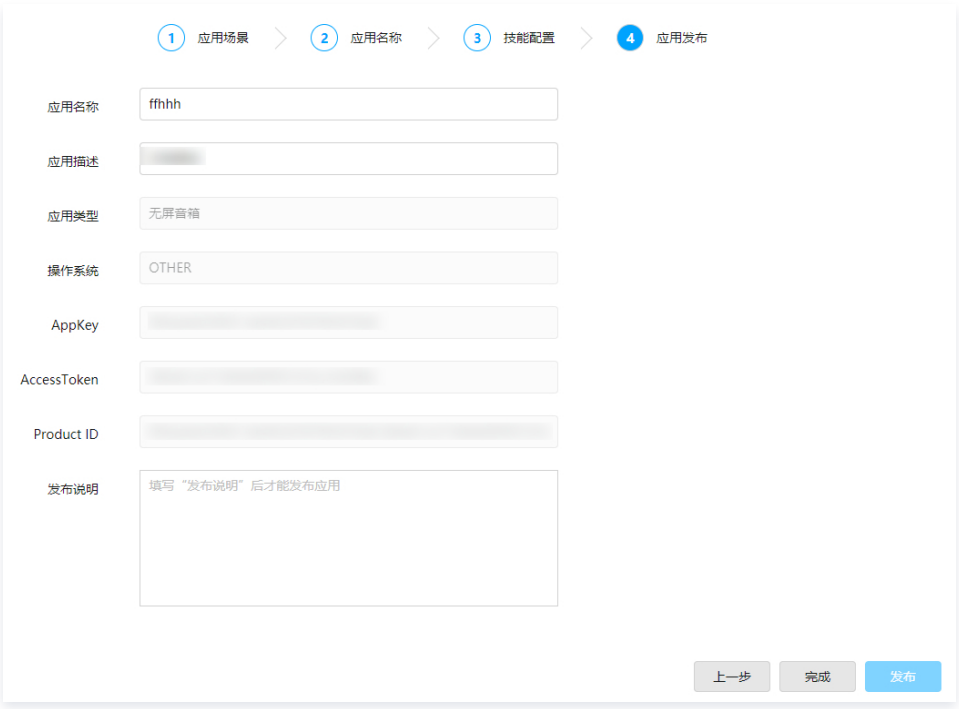
2. **单击下一步，填写应用名称和应用描述。**

3. 单击**下一步**，选择**自定义** > **从自建技能库导入**，勾选**腾讯连连**并单击**确定**。

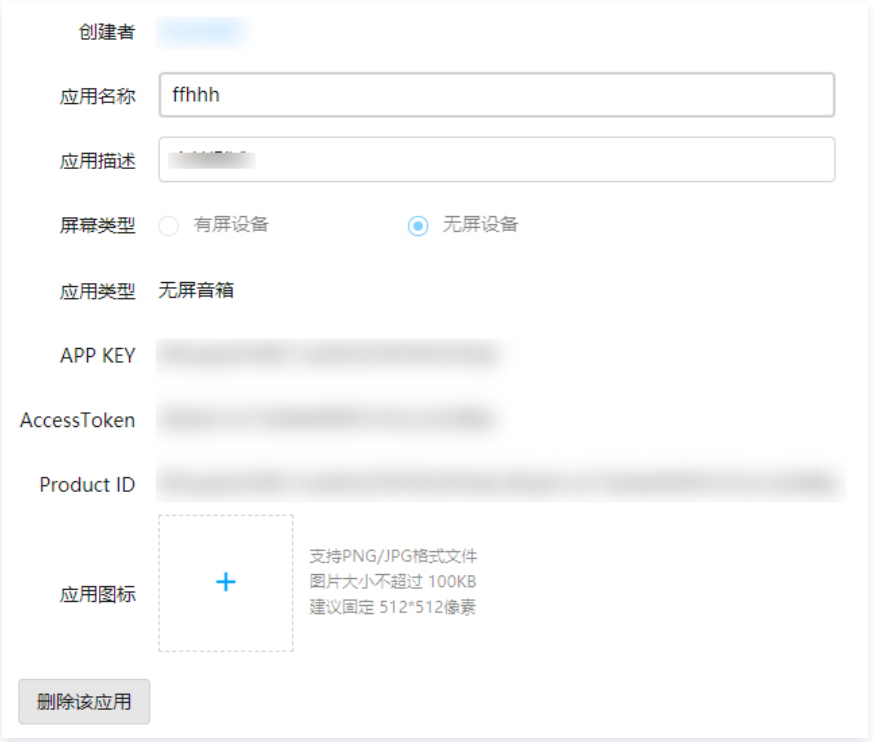


4. 填写版本号，格式为x.x.x.x，例如1.0.0.0。版本号生效以后将无法修改。版本号主要用来配合终端版本的更新，也可以用版本号来区分所创建的应用。单击**下一步**。

5. 填写发布说明，单击**完成**后，再单击**发布**即可创建应用成功。



6. 单击**应用概览**则可看到刚创建的应用，**从而获取应用对应的 App Key、App Secret、Product ID，请务必保证 App Key、App Secret 不被泄露，并与腾讯云物联网开发工程师进行线下对接。**



步骤三：主控设备接入

1. 基于腾讯云 IoT 和云小微融合版本 SDK 进行开发，SDK 请线下联系腾讯云同事或代理商提供。
2. 基于腾讯云三元组信息（设备名称、设备密钥和产品 ID）、云小微 tvs_pid 和 DSN 进行设备端对接开发。

> **说明：**
> - 三元组信息获取，详情可参见 设备信息 。
> - DSN 由“产品ID_设备名称 ”拼接组成。

- 被控设备支持云小微技能，需提前申请开通 云小微语音技能服务，服务开通后需与腾讯云物联网开发工程师进行线下对接。

步骤四：设备量产

登录 物联网开发平台控制台 ，在已开通“语言助手”的产品上，提交量产申请，审核通过后即可量产，详情可参见 批量投产 。

小程序操作指南

1. 主控设备配网绑定。
   使用腾讯连连小程序，扫码绑定主控设备配网二维码进行配网绑定。
2. 激活主控设备。

3. 主控设备关联被控设备。
   主控设备激活后，可关联被控设备，关联成功后，即可进行语音控制。

> **说明：**
> 在添加被控设备前，需要对被控设备进行改名后才能对智能设备进行控制。

4. QQ 音乐授权使用。
   若开发者在 步骤二 接入了音乐技能，在小程序中的主控设备面板单击“QQ 音乐”，进行授权后，则可使用音乐技能。

> **说明：**
> - 小程序音乐点播的标准控制面板正在研发中，后续会进行发布。开发者也可以选择 自定义 H5 面板开发 。
> - 音乐技能接入指引：在云小微技能平台，版本管理处选择 QQ 音乐技能，详情请参见 音乐服务 。

自定义 H5 开发

开发者可以自定义开发主控设备或者被控设备的控制面板，详情请见 自定义 H5 开发 。

酷狗音乐服务

最近更新时间：2024-11-01 09:45:32

本文为您介绍如何开通与使用腾讯云物联网开发平台酷狗音乐增值服务。

限制条件

仅支持在以下地区激活设备开通酷狗音乐服务：
中国大陆（不含港澳台地区）

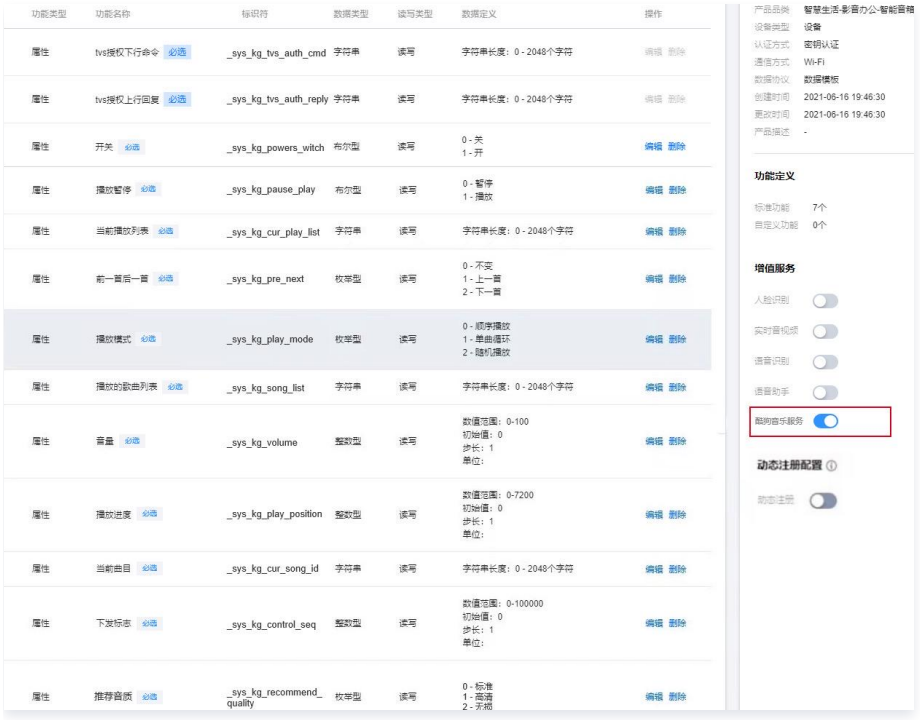
申请开通酷狗音乐服务

酷狗音乐服务为付费增值业务，不提供免费试用，您可以 提交工单 进行业务咨询，工作人员将会与您对接服务购买事宜。

操作步骤

步骤一：开通酷狗音乐服务

1. 登录 物联网开发平台控制台 ，地区选择“**中国区**”并创建项目及产品，详情请参见 产品定义 。
2. 选择已创建的项目进入项目详情页，单击左侧导航菜单**产品开发**进入产品列表页。
3. 选择已创建的产品进入产品详情页，在右侧**增值服务**菜单栏中，单击**酷狗音乐服务**处按钮即可开通成功。
4. 成功开通酷狗音乐服务后，该产品的物模型会自动添加酷狗音乐服务相关的系统属性。



步骤二：创建设备

单击**设备调试** > **新建设备**，填写相关信息单击**保存**即可。

注意：

- 当产品下存在设备时，不可更改增值服务开关状态。
- 若删除已开通增值服务的设备，所购买 License 数量不可恢复。

步骤三：设备端接入指引

设备端接入根据不同物联网终端的系统资源和安全要求不同分为带屏类设备的 Android SDK 接入 和低功耗物联网设备的 C-SDK 接入 。

Android SDK 接入指引

提供面向车机、智能屏幕、TV 等 Android 终端设备的 IoT+ 音乐内容服务的 Android SDK 接入。设备移植 SDK 后，绑定在腾讯连连小程序后，可在腾讯连连小程序授权设备接入酷狗音乐的权限；通过酷狗音乐小程序可控制设备播放酷狗音乐的音频资源。

集成 SDK 方式

依赖本地 sdk 源码构建修改应用模块的 **build.gradle**，使应用模块依赖 explorer-device-tme 源码，示例如下：

```
dependencies {
    implementation project(':explorer:explorer-device-tme')
}
```

说明：

需要集成该 SDK 请线下联系，Demo 示例工程使用的是 依赖本地 explorer-device-tme 的 sdk 源码构建方式。

集成 SDK 时的注意点

- 在 build.gradle 文件中添加如下配置：

```
//目前 sdk 仅提供 armeabi-v7a 和 x86 两种so,加上 abiFilters 以防某些情况下出现加载不了 so 库的问题
android {
    ndk {
        abiFilters 'armeabi-v7a', 'x86'
    }
    if (findProject(':explorer:explorer-device-tme') != null) {
        api project(':explorer:explorer-device-tme')
    }

    implementation 'io.reactivex.rxjava2:rxjava:2.2.10'
    implementation 'io.reactivex.rxjava2:rxandroid:2.1.1'
    implementation 'com.squareup.retrofit2:retrofit:2.6.0'
    implementation 'com.squareup.retrofit2:adapter-rxjava2:2.6.0'
    def room_version = "2.2.5"
    implementation "androidx.room:room-runtime:$room_version"
    annotationProcessor "androidx.room:room-compiler:$room_version"
    implementation "androidx.room:room-rxjava2:$room_version"
```

```
}
```

- 在 Android6.0 以上系统时，以下需求权限需要动态申请，请确保已授权。

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

- 在 Application#onCreate 里调用 UltimateTv#onApplicationCreate()。

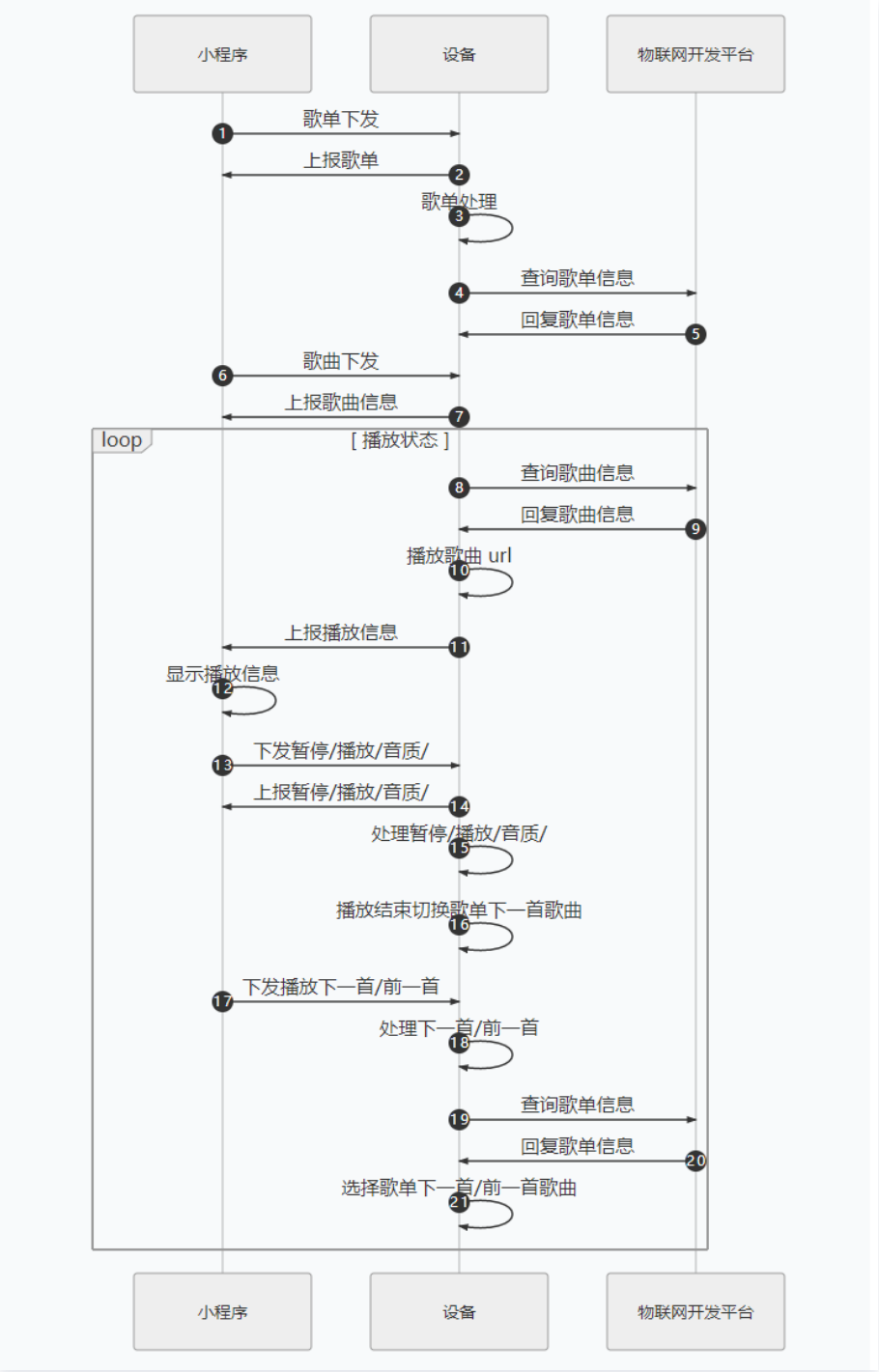
```
public class MyApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        //调用sdk的onAppcationCreate
        UltimateTv.getInstance().onAppcationCreate(this);
        ...
    }
    ...
}
```

C-SDK 接入指引

概述

面向穿戴、小家电、儿童教育类等 Linux 或嵌入式操作系统终端设备的 IoT+ 酷狗音乐内容服务的 C-SDK 接入。设备移植 SDK 后，绑定在腾讯连连小程序后，可在腾讯连连小程序授权设备接入酷狗音乐的权限；通过酷狗音乐小程序可控制设备播放酷狗音乐的音频资源。

交互流程



数据模板

- **小程序通过数据模板下发歌单，待播放歌曲 ID。**
- **控制播放/暂停/下一首/前一首。**
- **设置播放进度/播放音质等。**
- **设备通过数据模板上报播放信息，当前歌曲 ID/播放音质/播放进度等。**

topic

```
$thing/up/property/{product_id}/{device_name}
$thing/down/property/{product_id}/{device_name}
```

歌曲查询与歌单查询

topic

```
$thing/up/service/{product_id}/{device_name}
$thing/down/service/{product_id}/{device_name}
```

数据格式

歌曲信息查询

```
{
    "method": "kugou_query_song",
    "clientToken": "与之前不重复的字符串",
    "params": {
         "song_id": "xxxxx",
    }
}
{
    "method": "kugou_query_song_reply",
    "clientToken": "与上行消息的保持一致",
    "code": 0,
    "status": "",
    "data":{"song_info"}
}
```

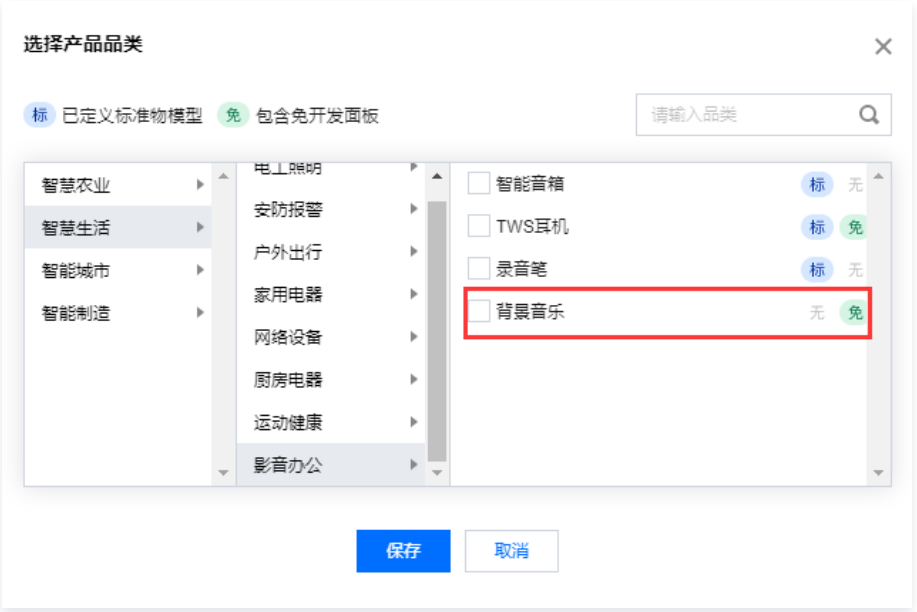
歌单查询

```
{
    "method": "kugou_user_command",
    "clientToken": "与之前不重复的字符串",
    "timestamp": 1594108563000
    "params": {"album_id/playlist_id/top_id":"歌单ID","page":请求歌单的第几页歌曲从1开始， "size":每页几首歌曲,"kugou_command":"歌单查询命令"}
}
{
  "method": "kugou_user_command_reply",
  "clientToken": "20a4ccfd-d308-11e9-86c6-5254008a4f10",
  "code": 0,
  "status": "",
  "data": {
    "歌单信息"
    "total": 歌单总共多少首歌曲,
    "songs": [{"song_id 等信息"},{"song_id 等信息"},{"song_id 等信息"}]
  }
}
```

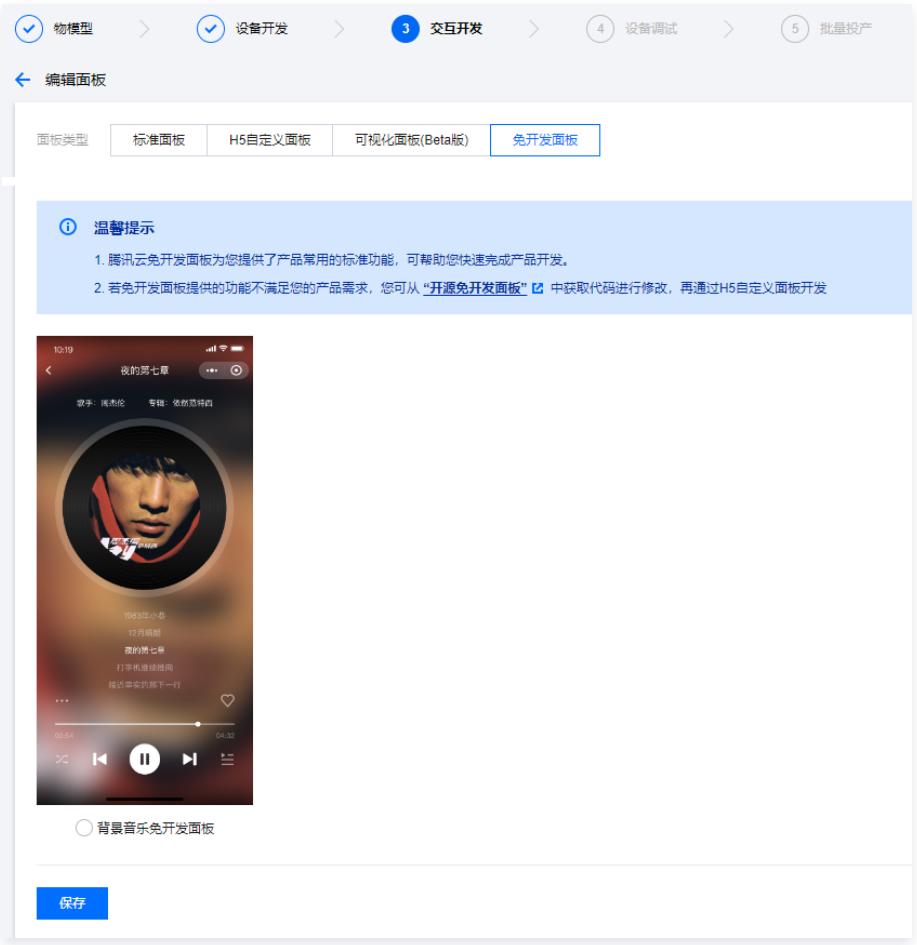
步骤四：应用端开发接入指引

基于腾讯连连酷狗音乐免开发面板接入

- 需在 控制台 创建产品时选择**标准品类**，指定**智慧生活 > 影音办公 > 背景音乐**品类。



- **开通酷狗音乐服务后，在交互开发中选择面板类型为“免开发面板”，即可使用产品常用的标准功能，快速完成产品开发。**


物模型
设备开发
3
交互开发
4
设备调试
5
批量投产
编辑面板
面板类型
标准面板
H5自定义面板
可视化面板(Beta版)
免开发面板
温馨提示
1. 腾讯云免开发面板为您提供了产品常用的标准功能，可帮助您快速完成产品开发。
2. 若免开发面板提供的功能不满足您的产品需求，您可从 “开源免开发面板” 中获取代码进行修改，再通过H5自定义面板开发
夜的第七章
背景音乐免开发面板
保存

- 选择小程序免开发面板后，即可获取海量音乐资源通过设备进行播控。

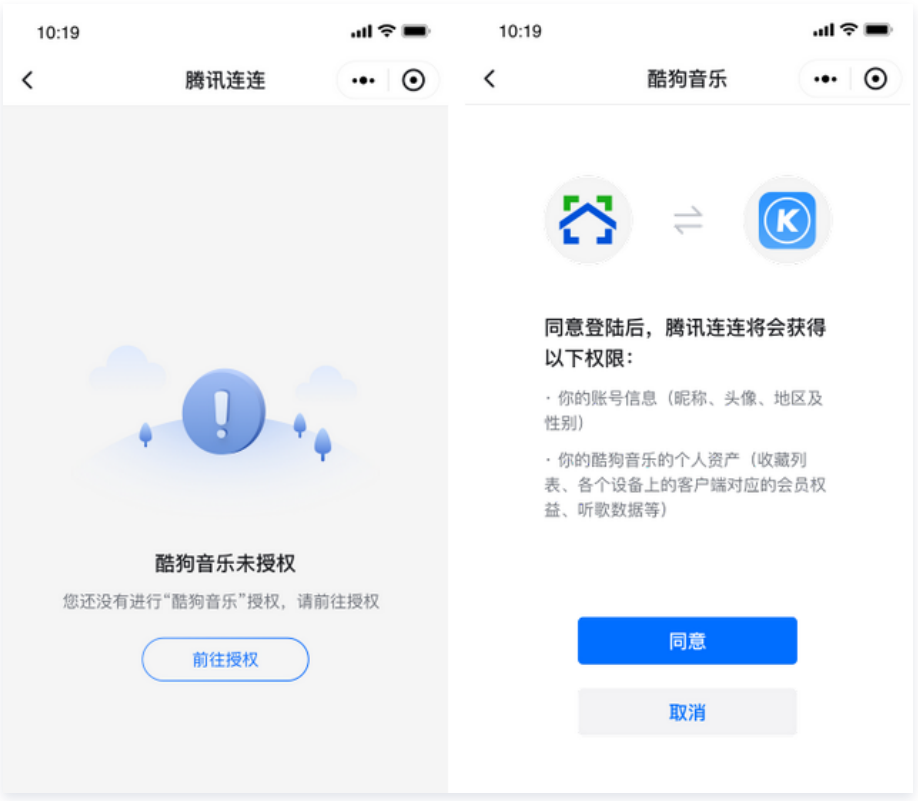
基于音乐服务 SDK 接入

对于该增值服务，提供了音乐服务 SDK 供自定义H5开发与自主品牌小程序开发使用，详见 音乐服务 SDK 。

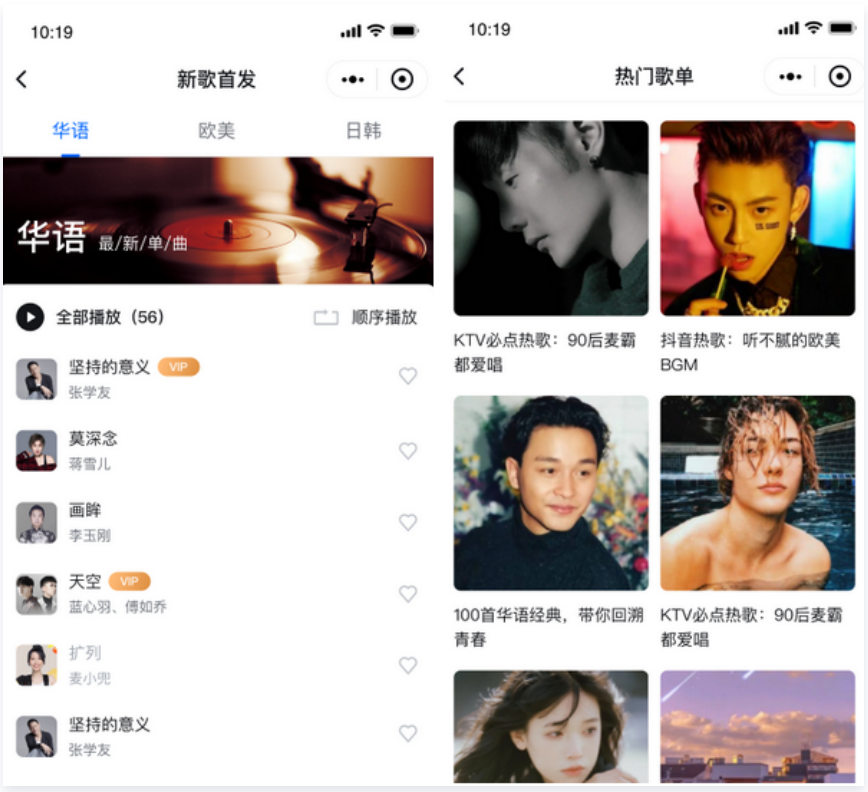
连连小程序用户使用指南

1. 设备绑定：使用腾讯连连小程序，扫描设备二维码进行设备绑定。（根据设备通信类型不同选择不同的绑定方式，Wi-Fi 类设备配网绑定，蜂窝类设备扫码绑定。）

2. 通过连连小程序绑定设备，选择设备后进入主界面入口授权酷狗音乐服务。



3. 授权酷狗音乐服务后，即可获取海量音乐资源播控：



SDK 接入指引

- 应用端 SDK 使用可访问官网参考最新版本 H5 自定义开发 SDK 。
- 设备端 SDK 使用 Github 托管，可访问 Github 下载最新版本 Android SDK 接入指南 、设备端 C-SDK 接入指南 。

语音技能服务
Alexa 语音技能服务

最近更新时间：2023-10-07 16:31:41

本实例中使用第三方平台 Amazon Alexa 实现语音智能服务，关于 Alexa 详细的官方文档，详情请参见 Amazon Alexa 文档 。

限制条件

激活设备对接 Amazon Alexa 仅支持以下地区

美国

语音技能仅支持以下语言

英语

控制台开通第三方语音技能服务

步骤1：确认产品范围和功能

新建产品时，产品品类需选择平台指定支持的品类，若选择其他品类或自定义将无法使用语音技能服务。

您可以通过下表，查看支持的品类和功能是否满足您的产品开发。

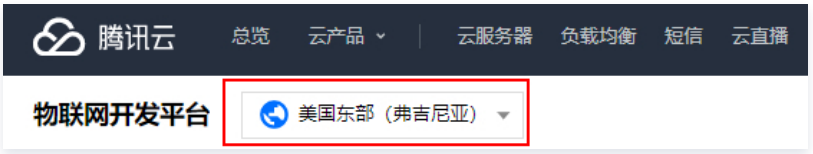
支持的品类	支持的功能
智慧生活—电工照明—灯	开关、亮度调节、颜色调节、色温调节。
智慧生活—电工照明—开关面板	开关。
智慧生活—电工照明—插座	开关。
智慧生活—电工照明—窗帘	开关、百分比调节。
智慧生活—家用电器—香薰机	开关。

智慧生活—家用电器—扫地机器人	开关。
智慧生活—家用电器—空气净化器	开关、风速（调大调小）、模式。

说明：
腾讯云物联网开发平台关于 Amazon Alexa 支持的品类或功能，后续将会支持更多，若您有接入需求，您可以在腾讯云官网通过 在线客服，描述您的产品需求并提交开通申请，我们将安排相关工作人员与您对接。

步骤2：申请开通 Amazon Alexa 服务

1. 登录 物联网开发控制台 ，地区选择美国东部，创建项目及产品，详情请参考 产品定义 。



2. 单击项目进入项目详情界面，单击**语音技能** > **Amazon Alexa** > **申请开通**，进入申请界面。

3. 选择需要开通的产品，填写申请信息后，还需勾选“我了解并同意《开发者须知与授权》”，单击**提交申请**，我们将安排相关工作人员与您进行对接。
   - **选择产品**：该项目下创建的产品。
   - **其他需求描述**：最多不超过250个字符。

4. 申请通过后，您也可以在**选择产品**处新增您该项目下的产品。

5. 选择需要添加的产品，单击**确定**。

6. 添加产品审核通过后，即可生效语音技能。

> **说明：**
> 使用腾讯连连小程序进行设备调试，配网绑定您的设备后，可根据下方 消费者使用 步骤，绑定 Amazon Alexa，即可实现音箱控制设备的功能。

消费者使用

前提条件

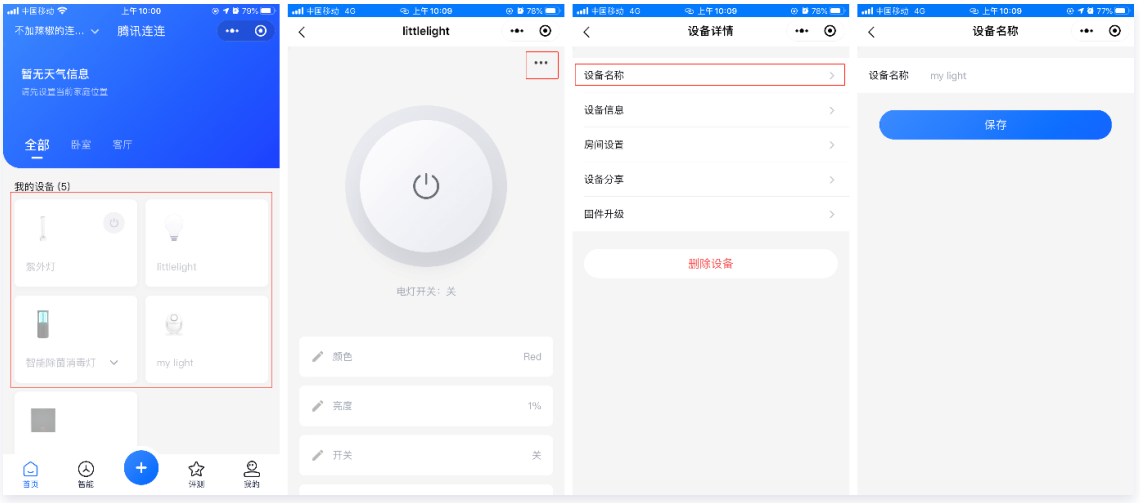
1. 拥有一台 Alexa 设备，以及 Amazon Alexa 可正常使用的账号。
2. 拥有一台及以上物联网开发平台发布的智能设备，且使用微信小程序“腾讯连连”绑定该设备。
3. 拥有可以顺畅访问 Amazon 服务的 Wi-Fi 网络。

操作步骤

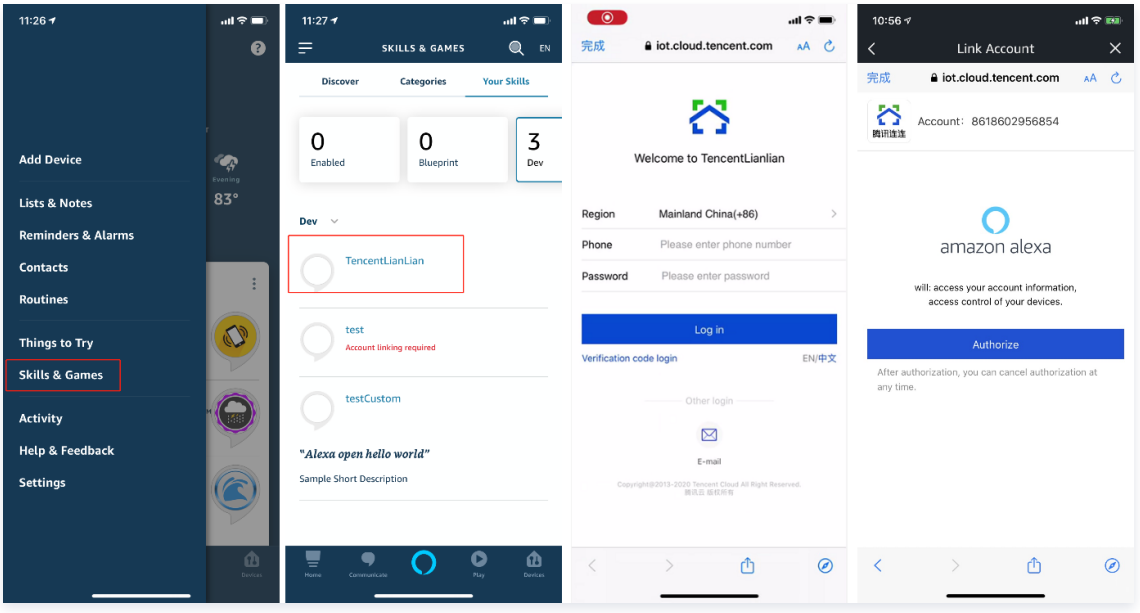
1. 用户使用微信小程序“腾讯连连”绑定物联网开发平台发布的智能设备产品。

> **注意：**
> 使用微信小程序“腾讯连连”登录的用户，**需要前往个人中心绑定手机号或者邮箱号并且设置密码**，路径为选择**我的 >个人信息**，进入账号与安全页面，即可绑定手机号或者邮箱号。

2. **将已绑定的设备改为英文名**，例如：my light，修改后的名称避免使用符号。**设备修改路径为**：小程序首页 > 选择指定产品名称 > 打开设备详情 > 单击**设备名称**，修改后单击**保存**即可。



3. 拥有一台 Amazon Alexa 智能音箱，下载 Amazon Alexa App 并绑定该音箱。
4. Amazon Alexa App 登录腾讯连连的账号，授权设备的控制权。打开菜单，选择 **Skill&Games**，发现腾讯连连，选择后进行账号绑定。



5. 控制设备前，Amazon Alexa 音箱需要先发现设备。您可以对 Amazon Alexa 音箱说：“Alexa, discover devices。”

> **注意：**
> 若是在腾讯连连中修改了产品名称，则在 Amazon Alexa 音箱中需要重新绑定设备。

使用 Amazon Alexa 音箱控制产品，支持的功能例句可参考下表。

品类	功能	语音例句（以实际使用场景为准）
灯	开关	Alexa, turn on the light. Alexa, turn off the light.
开关面板	开关	Alexa, turn on the switch. Alexa, turn off the switch.
插座	开关	Alexa, turn on the socket. Alexa, turn off the socket.
窗帘	开关、百分比调节	Alexa, turn on curtain. Alexa, turn off curtain. Alexa, open|close the curtain 50%.
香薰机	开关	Alexa, turn on the Aroma. Alexa, turn off the Aroma.
扫地机器人	开关	Alexa, turn on the worker. Alexa, turn off the worker.
空气净化器	开关、风速（调大调小）、模式	Alexa, turn on the purifier. Alexa, turn off the purifier. Alexa, set the purifier FanSpeed to 3. Alexa, set the purifier to 3. Alexa, setFanSpeed.

云小微语音技能服务

最近更新时间：2023-10-07 16:31:42

本文为您介绍如何使用腾讯云小微服务实现对智能设备的控制。

限制条件

仅支持在以下地区激活设备对接腾讯云小微

中国大陆（不含港澳台地区）

语音技能仅支持以下语言

中文

控制台开通第三方语音技能服务

步骤1：确认产品范围和功能

新建产品时，产品品类需选择平台指定支持的品类，若选择其他品类或自定义将无法使用语音技能服务。

您可以通过下表，查看支持的品类和功能是否满足您的产品开发。

支持的品类	支持的功能
智慧生活—电工照明—灯	开关、亮度调节、颜色调节、色温调节。
智慧生活—电工照明—开关面板	开关。
智慧生活—电工照明—插座	开关。
智慧生活—电工照明—窗帘	开关。
智慧生活—家用电器—香薰机	开关。
智慧生活—家用电器—空气净化器	开关、风速（调大调小）、模式。

智慧生活—家用电器—空调	开关、模式、风速（调大调小）、温度（调高调低、温度指定）。
智慧生活—家用电器—风扇	开关、风速（调大调小）。
智慧生活—家用电器—扫地机器人	开关。
智慧生活—家用电器—电视	开关、频道调节、音量调节。

说明：
腾讯云物联网平台的关于腾讯云小微支持的品类或功能，将会逐渐的增加，若您有接入需求，您可以在腾讯云官网通过 在线客服，描述您的产品需求并提交开通申请，工作人员将会与您进行对接。

步骤2：申请开通腾讯云小微服务

1. 登录 物联网开发控制台 ，地区选择中国，创建项目及产品，详情请参考 产品定义 。

2. 单击项目进入项目详情界面，单击**语音技能** > **腾讯云小微** > **申请开通**，进入申请界面。

3. 选择需要开通的产品，填写申请信息后，还需勾选“我了解并同意《开发者须知与授权》”，单击**提交申请**，我们将安排相关工作人员与您进行对接。
   - **选择产品**：该项目下创建的产品。
   - **其他需求描述**：最多不超过250个字符。

4. 申请通过后，您也可以在**选择产品**处新增您该项目下的产品。
5. 选择需要添加的产品，单击**确定**。

6. 添加产品审核通过后，即可生效语音技能。

> **说明：**
> 使用腾讯连连小程序进行设备调试，配网绑定您的设备后，可根据下方 消费者使用 步骤，绑定腾讯云小微，即可实现音箱控制设备的功能。

消费者使用

前提条件

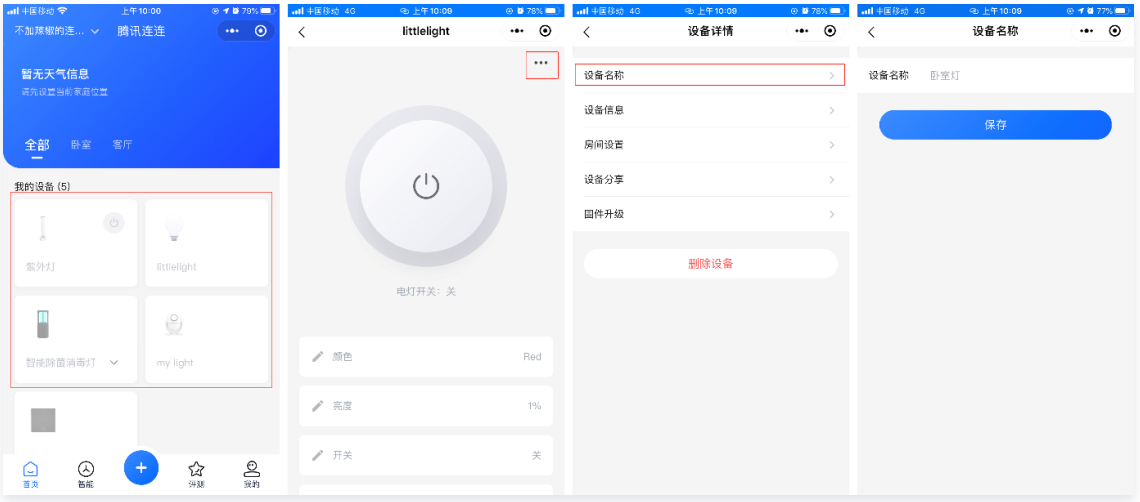
1. 拥有一台搭载云小微语音技能的设备，例如：智能音箱，语音智能灯等。
2. 拥有一台及以上物联网开发平台发布的智能设备，且使用微信小程序“腾讯连连”绑定该设备。
3. 使用第三方的产品绑定腾讯连连账号及授权设备使用，具体的绑定流程根据第三方平台产品提示进行。

> **注意：**
> 具体的消费者使用教程，请以第三方厂商的指引为准。

操作步骤

1. 申请腾讯连连账号后绑定手机号或者邮箱号，并设定密码。

2. 在腾讯连连小程序中添加的语音技能产品需要修改设备名称，例如：卧室灯，修改后的名称避免使用符号。



使用腾讯云小微控制智能设备，支持的功能例句可参考下表。

品类	功能	语音例句（以实际使用场景为准）
灯	开关、亮度调节、色温调节、颜色调节。	开灯、关灯。 打开卧室灯、关闭卧室灯。 灯光调亮一些。 灯光色温调暖一些。 把灯调为蓝色。
开关面板	开关	打开开关。 关闭开关。
插座	开关	打开插座。 关闭插座。
窗帘	开关	打开窗帘。 关闭窗帘。 窗帘开到50%。
香薰机	开关	打开香薰机。 关闭香薰机。
扫地机器人	开关	打开扫地机器人。 关闭扫地机器人。
空气净化器	开关、风速（调大调小）、模式。	打开空气净化器。 关闭空气净化器。 调大风速。 调小风速。 打开睡眠模式。
空调	开关、模式、风速（调大调小）、温度（调高调低、温度指定）。	打开空调、关闭空调。 打开制冷模式。 温度调高、温度调低、温度调到26度。
电视	开关、频道调节、音量调节。	打开电视、关闭电视。 下一个频道、上一个频道、我想看中央一台。 增大音量、调低音量、电视静音。
风扇	开关、风速（调大调小）。	打开风扇、关闭风扇。 风速调大、风速调小。

Google 语音技能服务

最近更新时间：2024-11-14 15:47:32

本文为您介绍如何开通 Google Assistant 语音智能服务，实现通过 Google Home 音箱对智能设备的控制。

限制条件

激活设备对接 Google Assistant 仅支持在以下地区

美国

语音技能支持语言

- 英语
- 中文（部分语义）

控制台开通第三方语音技能服务

步骤1：确认产品范围和功能

新建产品时，产品品类需选择平台指定支持的品类，若选择其他品类或自定义将无法使用语音技能服务。

您可以通过下表，查看支持的品类和功能是否满足您的产品开发。

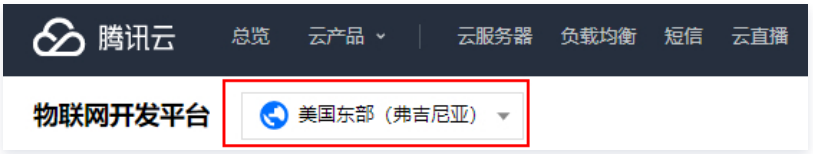
支持的品类	支持的功能（状态获取&指令）
智慧生活—电工照明—灯	开关、颜色调节、亮度调节。
智慧生活—电工照明—开关面板	开关。
智慧生活—电工照明—插座	开关。
智慧生活—电工照明—窗帘	开关、百分比调节。
智慧生活—家用电器—香薰机	开关。

智慧生活—家用电器—扫地机器人	开关。
智慧生活—家用电器—风扇	开关

说明：
腾讯云物联网平台关于 Google Assistant 支持的品类或功能，后续将会支持更多，若您有接入需求，您可以在腾讯云官网通过 在线客服，描述您的产品需求并提交开通申请，我们将安排相关工作人员与您进行对接。

步骤2：申请开通 Google Assistant 服务

1. 登录 物联网开发平台，地区选择美国，创建项目及产品，详情请参考 产品定义 。



2. 单击项目进入项目详情界面，单击**语音技能 > Google Assistant > 申请开通**，进入申请界面。

3. 选择需要开通的产品，填写申请信息后，还需勾选“我了解并同意《开发者须知与授权》”，单击【提交申请】，我们将安排相关工作人员与您进行对接。
   - **选择产品**：该项目下创建的产品。

- **其他需求描述**：最多不超过250个字符。

4. 申请通过后，您也可以在**选择产品**处新增您该项目下的产品。

5. 选择需要添加的产品，单击**确定**。

6. 添加产品审核通过后，即可生效语音技能。

> **说明：**
> 使用腾讯连连小程序进行设备调试，配网绑定您的设备后，可根据下方 消费者使用 步骤，绑定 Google Home，即可实现音箱控制设备的功能。

消费者使用

前提条件

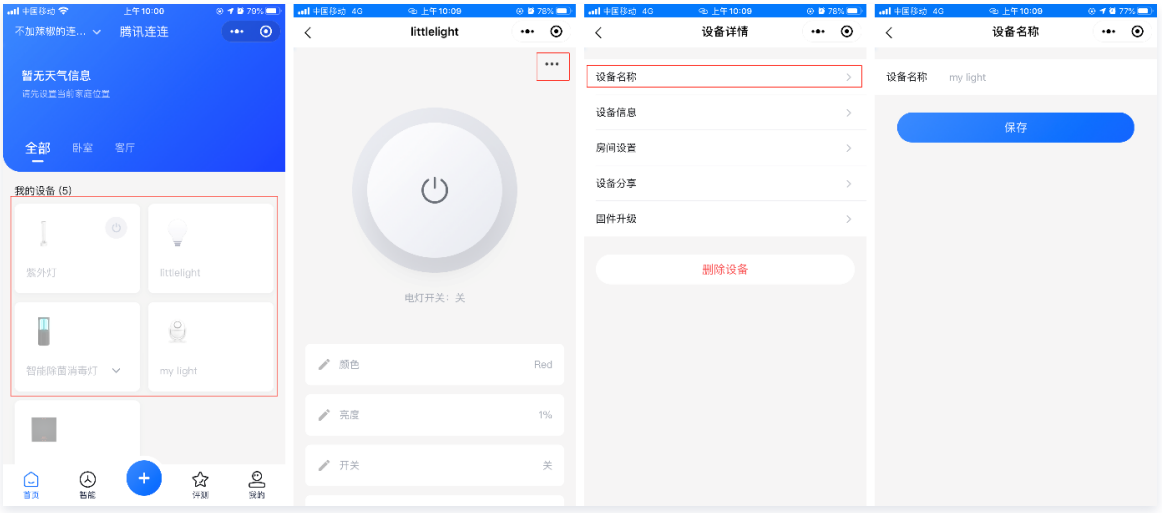
1. 拥有一台 Google Home 设备，以及 Google Home App 的登录账号。
2. 拥有一台及以上物联网开发平台发布的智能设备，且使用微信小程序“腾讯连连”绑定该设备。
3. 可以访问 Google 服务的 Wi-Fi 网络。

使用步骤

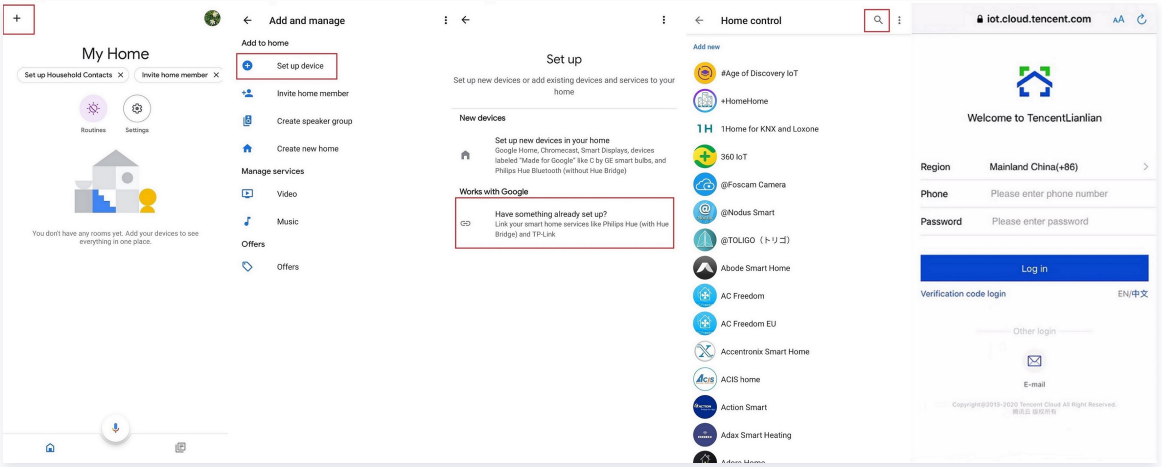
1. 用户使用微信小程序“腾讯连连”绑定物联网开发平台发布的智能设备产品。

> **注意：**
> 使用微信小程序“腾讯连连”登录的用户，**需要前往个人中心绑定手机号或者邮箱号并且设置密码**，路径为选择**我的 > 个人信息**，进入账号与安全页面，即可绑定手机号或者邮箱号。

2. **将已绑定的设备改为英文名**，例如：my light，修改后的名称避免使用符号。**设备修改路径为**：小程序首页 > 选择指定产品名称 > 打开设备详情 > 单击**设备名称**，修改后单击**保存**即可。



3. 下载安装 Google Home 或者 Google Assistant App 并绑定 Google Home 智能音箱。
4. 在 Google Home App 主页上点击“+”按钮添加设备，选择 **Set up devices** 列表下的 **Works with Google**。搜索并选择“tencentlianlian”，登录腾讯连连的账号进行账号绑定，授权设备的控制权。绑定成功后，您的设备会显示在 Home Control 的 Devices 列表中。



使用 Google Home 音箱控制产品，支持的功能例句可参考下表。

品类	功能	语音例句（以实际使用场景为准）
灯	开关、颜色调节、亮度调节。	Hey/Ok Google，turn on bedroom light. Hey/Ok Google，turn off bedroom light. Hey/Ok Google，set bedroom light to blue. Hey/Ok Google，brighten bedroom light

开关面板	开关。	Hey/Ok Google，turn on the switch. Hey/Ok Google，turn off the switch.
插座	开关。	Hey/Ok Google，turn on the socket. Hey/Ok Google，turn off the socket.
窗帘	开关、百分比调节。	Hey/Ok Google，turn on the curtain. Hey/Ok Google，turn off the curtain. Hey/Ok Google，open/close the curtain 50%.
香薰机	开关。	Hey/Ok Google，turn on the Aroma. Hey/Ok Google，turn off the Aroma.
扫地机器人	开关。	Hey/Ok Google，turn on the worker. Hey/Ok Google，turn off the worker.
风扇	开关。	Hey/Ok Google，turn on the fan. Hey/Ok Google，turn off the fan.

小度语音技能服务

最近更新时间：2024-12-31 18:00:53

本文为您介绍如何开通百度小度第三方语音服务，实现通过小度音箱对智能设备的控制。

限制条件

激活设备对接小度仅支持在以下地区

中国大陆（不含港澳台地区）

语音技能支持语言

中文

控制台开通第三方语音技能服务

步骤1：确认产品范围和功能

新建产品时，产品品类需选择平台指定支持的品类，若选择其他品类或自定义将无法使用语音技能服务。

您可以通过下表，查看支持的品类和功能是否满足您的产品开发。

支持的品类	支持的功能（状态获取&指令）
智慧生活—电工照明—灯	开关、颜色调节、亮度调节、色温调节。
智慧生活—电工照明—开关面板	开关。
智慧生活—电工照明—插座	开关。
智慧生活—电工照明—窗帘	开关。
智慧生活—家用电器—香薰机	开关。
智慧生活—家用电器—扫地机器人	开关。
智慧生活—家用电器—空气净化器	开关、风速（调大调小）、模式。

智慧生活—家用电器—空调	开关、模式、风速（调大调小）、温度（调高调低、温度指定）、摆风。
智慧生活—家用电器—电视	开关、暂停、继续、音量调节、频道调节。
智慧生活—家用电器—风扇	开关、风速（调大调小）、模式。

说明：
腾讯云物联网平台的关于百度小度支持的品类或功能，后续将会支持更多，若您有接入需求，您可以在腾讯云官网通过 在线客服，描述您的产品需求并提交开通申请，我们将安排相关工作人员与您进行对接。

步骤2：申请开通百度小度服务

1. 登录 物联网开发平台，地区选择中国，创建项目及产品，详情请参考 产品定义 。

2. 单击项目进入项目详情界面，单击**语音技能** > **小度** > **申请开通**，进入申请界面。

3. 选择需要开通的产品，填写申请信息后，还需勾选“我了解并同意《开发者须知与授权》”，单击**提交申请**，我们将安排相关工作人员与您进行对接。
   - **选择产品**：该项目下创建的产品。
   - **其他需求描述**：最多不超过250个字符。

4. 申请通过后，您也可以在“**选择产品**”处新增您该项目下的产品。

5. 选择需要添加的产品，单击**确定**。

6. 添加产品审核通过后，即可生效语音技能。

> **说明：**
> 使用腾讯连连小程序进行设备调试，配网绑定您的设备后，可根据下方 消费者使用 步骤，绑定小度音箱，即可实现音箱控制设备的功能。

消费者使用

前提条件

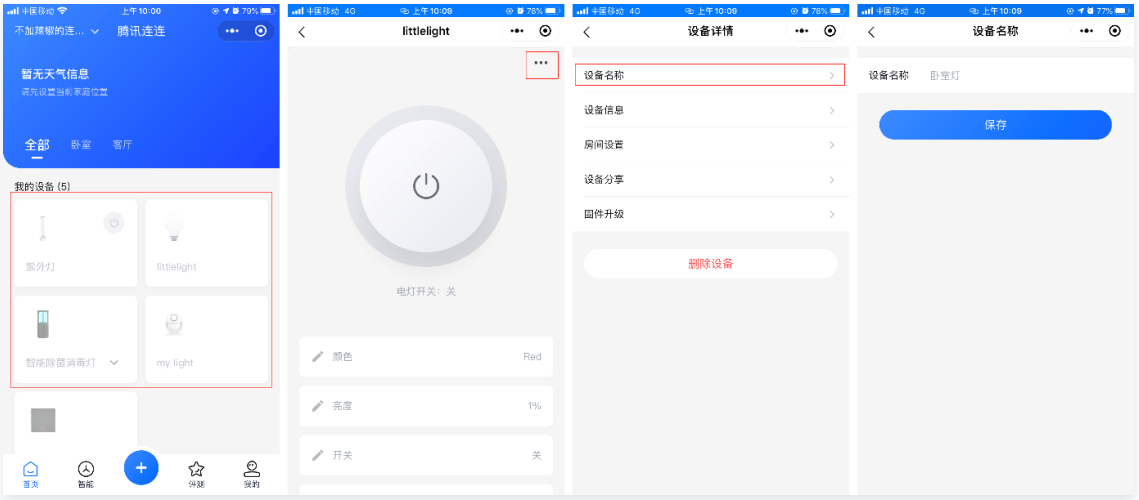
1. 拥有一台搭载小度语音服务的音箱设备，以及小度音箱 App 的登录账号。
2. 拥有一台及以上物联网开发平台发布的智能设备，且使用微信小程序“腾讯连连”绑定该设备；

操作步骤

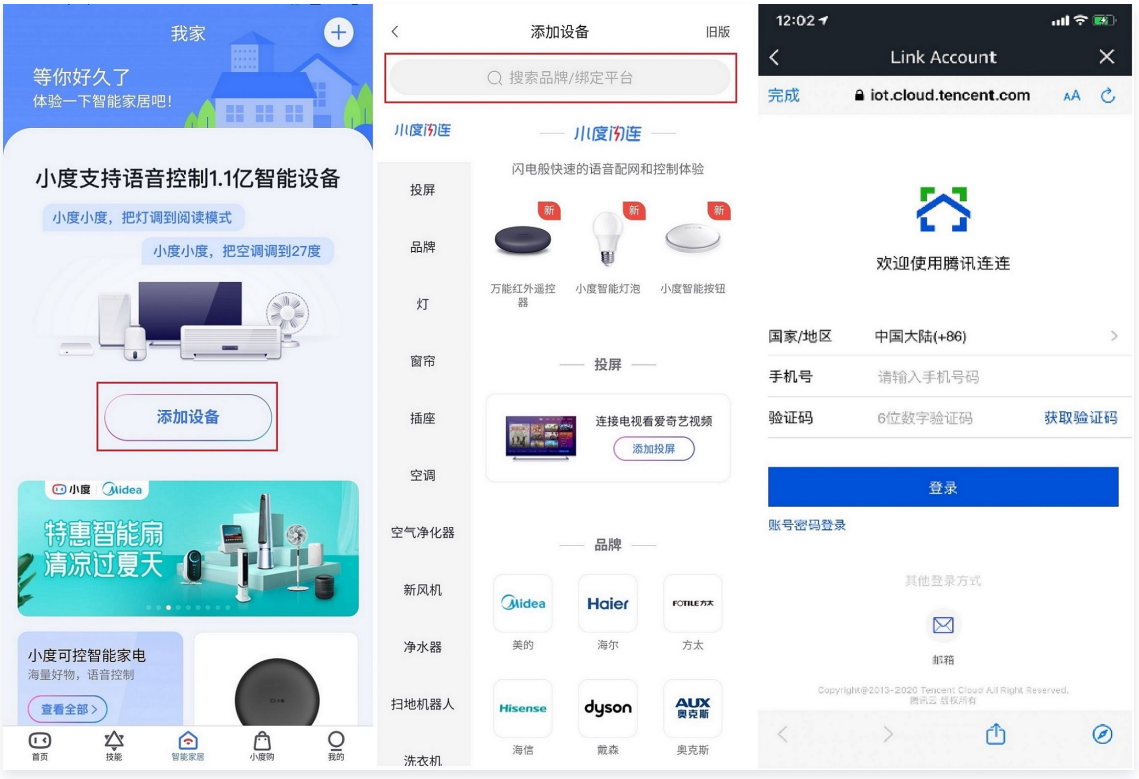
1. 用户使用微信小程序“腾讯连连”绑定物联网开发平台发布的智能设备产品。

> **注意：**
> 使用微信小程序“腾讯连连”登录的用户，**需要前往个人中心绑定手机号或者邮箱号并且设置密码**，路径为选择**我的** > **个人信息**，进入账号与安全页面，即可绑定手机号或者邮箱号。

2. 在腾讯连连小程序中添加的语音技能产品需要修改设备名称，例如卧室灯，修改后的名称避免使用符号。



3. 拥有一台百度小度智能音箱，下载小度音箱 App 并绑定该音箱。
4. 在小度音箱 App 首页进入**智能家居**，单击**添加设备**，搜索选择**“腾讯连连”**并登录腾讯连连账号，授权设备的控制权。



使用小度音箱控制产品，支持的功能例句可参考下表。

品类	功能	语音例句（以实际使用场景为准）
灯	开关、颜色调节、亮度调节、色温调节。	开灯、关灯。 打开卧室灯、关闭卧室灯。 把灯调为蓝色。 灯光调亮一些。 灯光色温调暖一些。
开关面板	开关。	打开开关。 关闭开关。
插座	开关。	打开插座。 关闭插座。

窗帘	开关。	打开窗帘。 关闭窗帘。
香薰机	开关。	打开香薰机。 关闭香薰机。
扫地机器人	开关。	打开扫地机器人。 关闭扫地机器人。
空气净化器	开关、风速（调大调小）、模式。	打开空气净化器。 关闭空气净化器。 调大风速。 调小风速。 打开睡眠模式。
空调	开关、模式、风速（调大调小）、温度（调高调低、温度指定）、摆风。	打开空调、关闭空调。 打开制冷模式。 温度调高、温度调低、温度调到26度。 设置空调左右摆风。
电视	开关、暂停、继续、频道调节、音量调节。	打开电视、关闭电视。 电视暂停播放、电视继续播放、下一个频道、上一个频道、我想看中央一台。 增大音量、调低音量、电视静音。
风扇	开关、风速（调大调小）、模式。	打开风扇、关闭风扇。 风速调大、风速调小。 打开睡眠模式。

位置服务
功能介绍

最近更新时间：2024-12-31 18:00:53

目前物联网开发平台 IoT Explorer 位置服务支持多种设备定位属性，包括 GPS 定位（GPS_Info、GPS_ExtInfo）、蜂窝定位（Cell_Info）、Wi-Fi 定位（Wifi_Info ）以及 LoRa Edge 定位（Wifi_Info、GNSS_NAV），从而确定设备所在具体位置。

- **GPS 定位：** 设备能够直接上报经纬度，可使用位置服务属性（GPS_Info、GPS_ExtInfo）定位设备。
- **蜂窝定位：** 若设备为 2G/4G 类设备，则可通过上报基站信息，使用位置服务属性（Cell_Info）定位设备。
- **Wi-Fi 定位：** 若设备为 Wi-Fi 类设备，可通过上报附近 Wi-Fi 路由器的 MAC 地址，使用位置服务功能属性（Wifi_Info ）定位设备。
- **LoRa Edge 定位：** 若设备为 LoRa Edge 类设备，可通过上报附近 Wi-Fi 路由器的 MAC 地址，使用位置服务功能属性（Wifi_Info）定位。也可通过上报视野内的卫星信息，使用位置服务功能属性（GNSS_NAV）定位。

前提条件

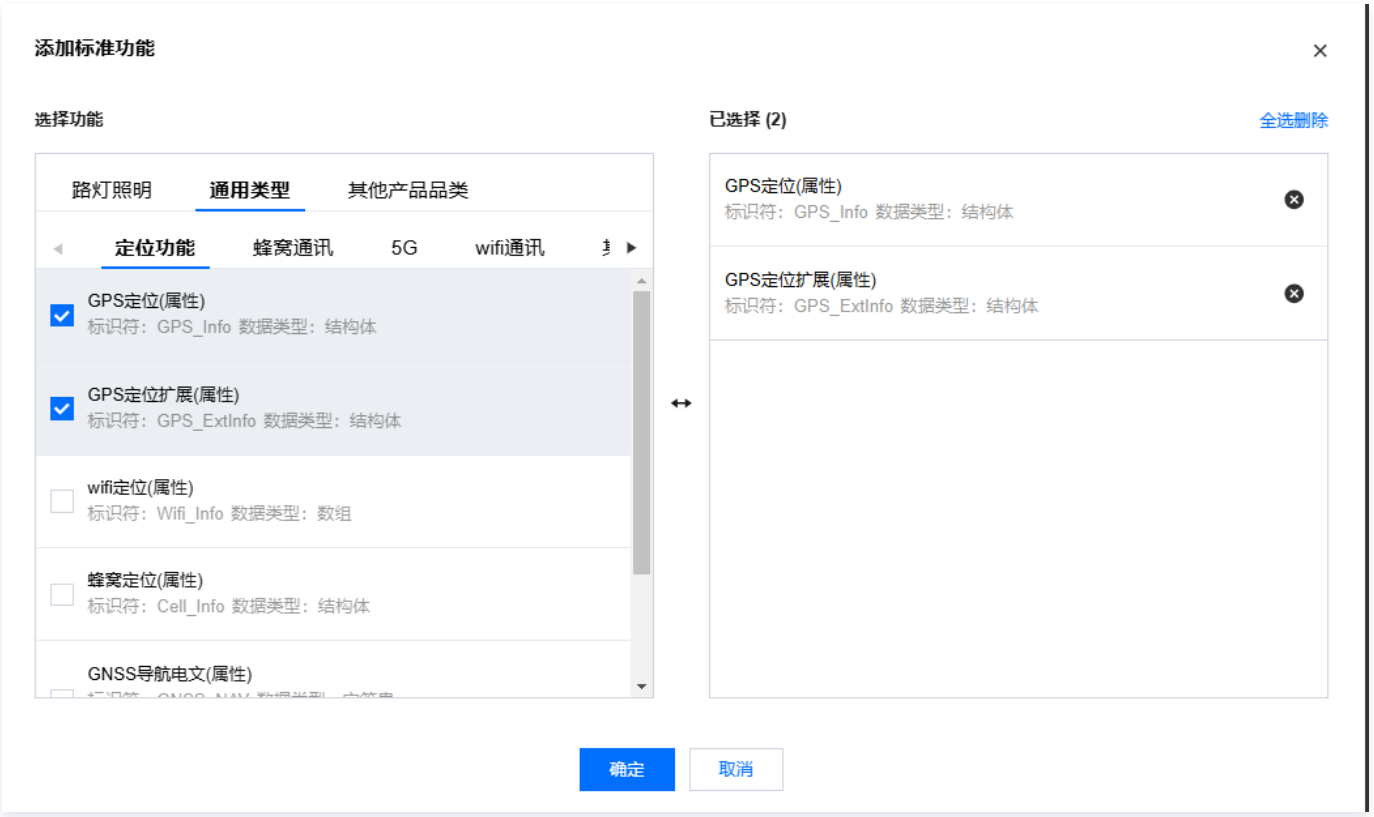
已在控制台 创建项目及产品 。

GPS 定位

1. 登录 物联网开发平台控制台 ，选择已创建的项目进入项目详情页。
2. 默认进入左侧菜单**产品开发**页，选择已创建的产品进入产品**物模型**页面，单击**添加标准功能**。

3. 在弹出“添加标准功能”的弹窗上，单击**通用类型** > **定位功能**并勾选“GPS定位”。

> **说明：**
> 真实设备在使用位置服务进行定位时，需要遵循属性的物模型定义上报数据，详情请参见 相关物模型说明 。



4. 单击**确认**，完成地理位置标准功能的添加。
5. 设备将会通过数据模板协议上报位置到云端，详情请参见 数据模板协议 。
   - 设备上报协议如下所示：
     - 上行请求 Topic：`$thing/up/property/{ProductID}/{DeviceName}` 。
     - 下行响应 Topic：`$thing/down/property/{ProductID}/{DeviceName}` 。
   - 请求示例如下所示：

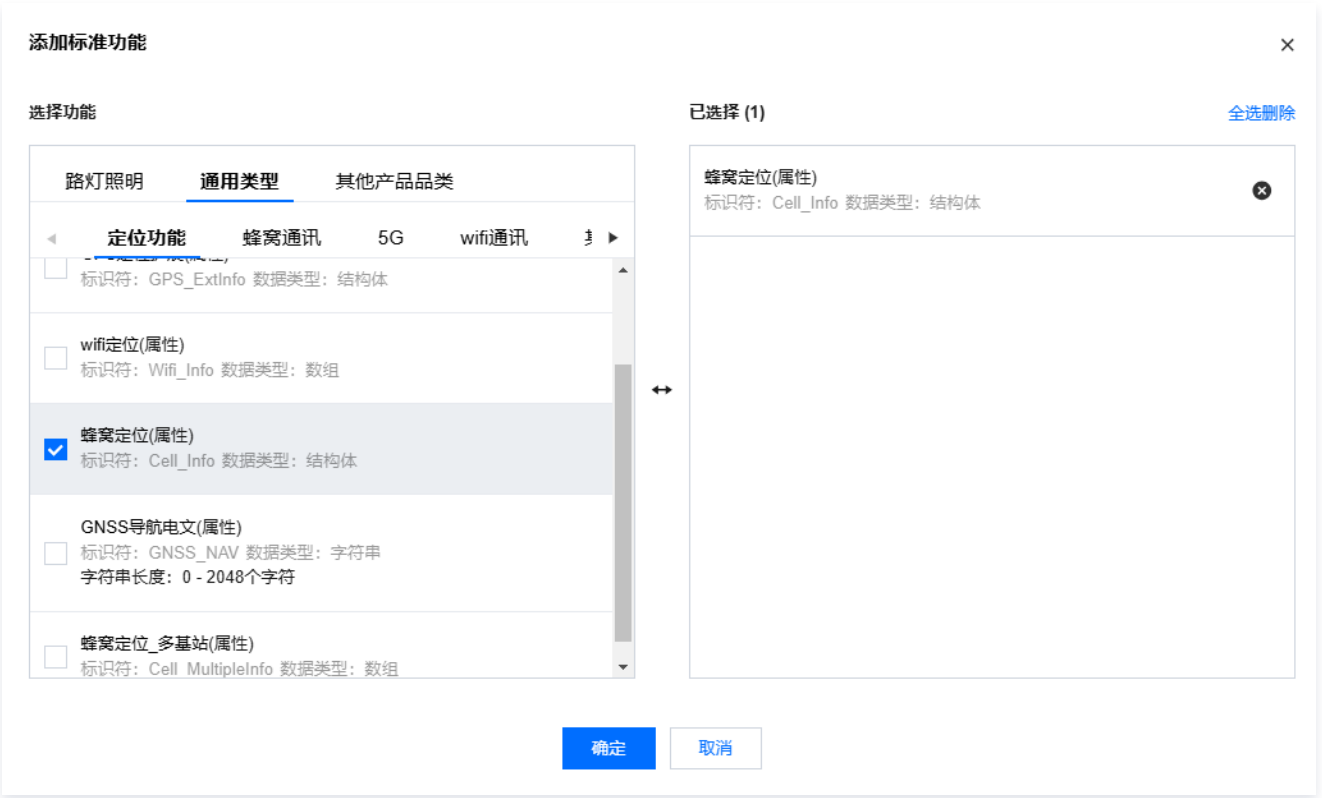
```
{
    "clientToken": "123",
    "method": "report",
    "params": {
    "GPS_Info":{"longtitude":112.59014,"latitude":22.28014}
    }
}
```

蜂窝定位

1. 登录 物联网开发平台控制台 ，选择已创建的项目进入项目详情页。

2. 默认进入左侧菜单**产品开发**页，选择已创建的产品进入产品**物模型**页面，单击**添加标准功能**。

3. 在弹出“添加标准功能”的弹窗上，单击**通用类型** > **定位功能**并勾选“蜂窝定位”。



4. 单击**确认**，完成蜂窝定位标准功能的添加。
5. 设备将会通过数据模板协议上报位置到云端，详情可参见 数据模板协议 。
   - 设备上报协议如下所示：
     - 上行请求 Topic：`$thing/up/property/{ProductID}/{DeviceName}` 。
     - 下行响应 Topic：`$thing/down/property/{ProductID}/{DeviceName}` 。
   - 请求示例如下所示：

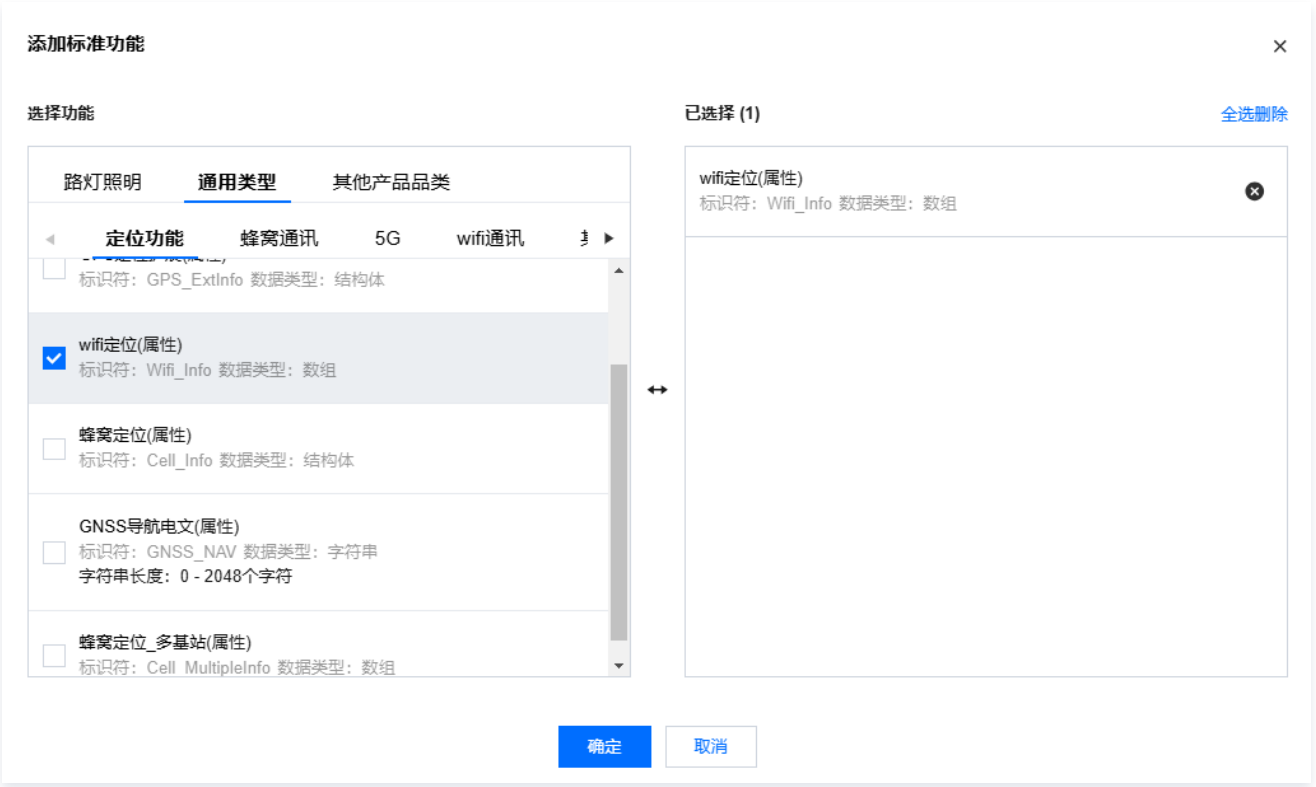
```
{
    "clientToken": "123",
    "method": "report",
    "params": {
    "Cell_Info": {"rss":-85,"networkType":1,"mcc":460,"mnc":13824,"lac":3,"cid":33}
    }
```

```
}
```

Wi-Fi 定位

1. 登录 物联网开发平台控制台，选择已创建的项目进入项目详情页。
2. 默认进入左侧菜单**产品开发**页，选择已创建的产品进入产品**物模型**页面，单击**添加标准功能**。

3. 在弹出“添加标准功能”的弹窗上，单击**通用类型** > **定位功能**并勾选“wifi定位”。



4. 单击**确认**，完成 Wi-Fi 定位标准功能的添加。
5. 设备可通过数据模板协议上报位置到云端，详情可参见 数据模板协议 。
   - 设备上报协议如下所示：
     - 上行请求 Topic：`$thing/up/property/{ProductID}/{DeviceName}` 。
     - 下行响应 Topic：`$thing/down/property/{ProductID}/{DeviceName}` 。
   - 请求示例如下所示：

```
{
    "clientToken": "123",
```

```
    "method": "report",
    "params": {
    "Wifi_Info":[{"Mac":"78a106a5b166","Rssi":-81},{"Mac":"608f5c66b3f7","Rssi":-82},
{"Mac":"282cb293a5d6","Rssi":-83}]
    }
}
```

LoRa Edge 定位

详细介绍请参阅 LoRa Edge 介绍、LoRa LR1110 快速入门。

空间管理

最近更新时间：2024-08-26 14:55:52

本文为您介绍在控制台创建、编辑和删除空间的相关操作。

前提条件

已完成 设备定位 。

新建空间

1. 登录 物联网开发平台控制台，选择已创建的项目进入项目详情页。
2. 单击左侧菜单**位置服务**进入位置服务界面，单击**新建空间**，填写相应信息。

- 空间名称：为空间命名，空间名称不允许与其他空间重复。支持中文、英文、数字、下划线的组合，最多不超过20个字符。
- 关联产品：将所选的产品与该空间项目关联，关联后空间项目方可获得该产品下设备的相关状态信息。
- 授权形式：选择空间项目对关联产品设备的权限，目前空间项目只具备设备的查看显示权。
- 备注：可输入文字，用来描述空间项目。字数限制为25字符。

3. 单击**保存**，新建的空间将更新至位置服务列表页。
4. 单击列表某个位置空间“空间名称”，将进入该空间项目的可视化操作界面。

编辑空间

1. 在位置服务列表页面，单击某个空间右侧菜单的**编辑**进入空间详情页。

序号	名称	备注	创建时间	操作
1	蜂窝设备空间	关于蜂窝类设备的围栏空间管理项目	2020-11-20 16:13:46	编辑 删除
2	位置空间444	for some project	2020-11-19 22:11:14	编辑 删除
3	位置空间333	位置空间管理深圳南山区的相关设备	2020-11-19 18:55:13	编辑 删除

2. 可对空间的相关信息进行修改。

3. 单击**保存**，将更新编辑保存后的空间信息。

删除空间

1. 在位置服务列表页面，单击某个空间右侧菜单的**删除**。

序号	名称	备注	创建时间	操作
1	蜂窝设备空间	关于蜂窝类设备的围栏空间管理项目	2020-11-20 16:13:46	编辑 删除
2	位置空间	位置空间管理	2020-11-19 18:55:13	编辑 删除

2. 在弹窗中单击**确定**。
   - 若空间项目下不存在 地理围栏，则提示“删除成功”，并从位置列表中删除。
   - 若空间项目下仍存在 地理围栏，则无法删除并提示“无法删除该空间项目”，需手动删除空间项目下所有地理围栏，才能够执行删除操作。


无法删除该空间项目
该空间下仍存在地理围栏，需删除所有围栏后才允许删除空间
确定

空间可视化

最近更新时间：2024-09-30 17:54:51

腾讯云物联网开发位置空间可视化服务，让您可在地图上实时展示设备的状态信息，支持查看设备属性数据、设备所在具体位置、热力图（点）、热力图（面）等功能，支持地图缩放、全屏查看，方便您查看及管理设备。

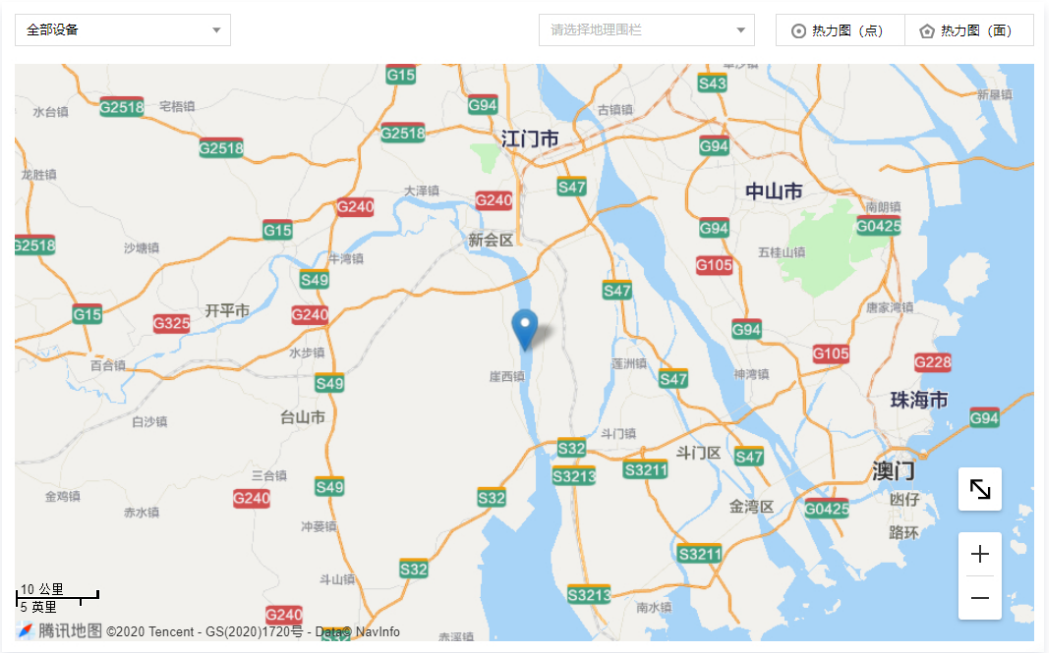
前提条件

已完成 空间创建 。

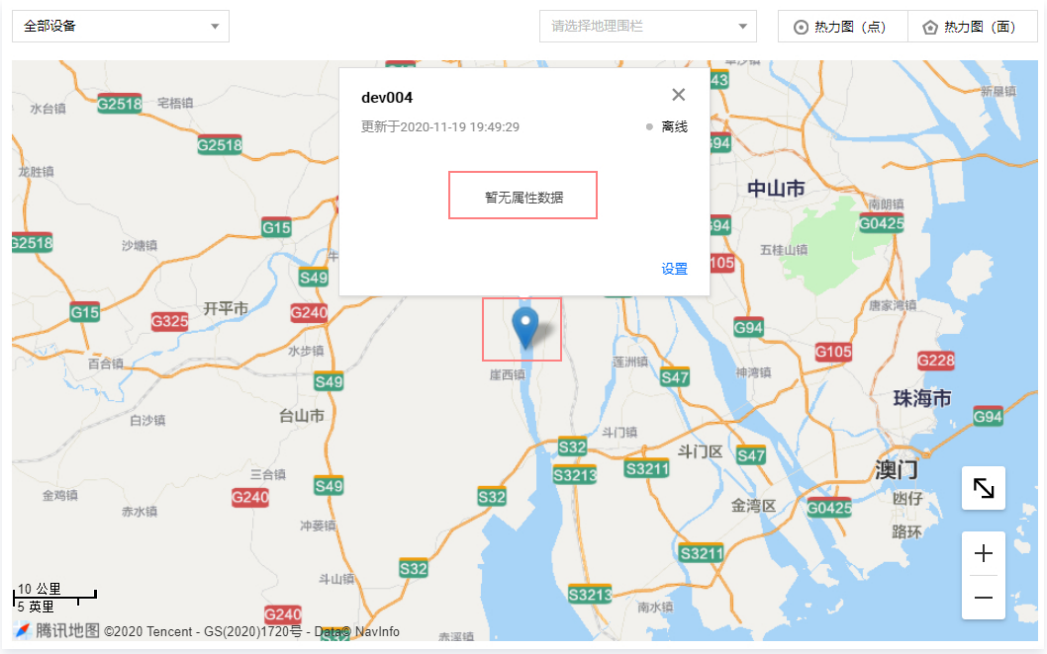
设备位置可视化

1. 登录 物联网开发平台控制台 ，选择已创建的项目进入项目详情页。
2. 单击左侧菜单**位置服务**进入位置服务页面。
3. 在位置服务页面，单击某个已经创建的“空间名称”。

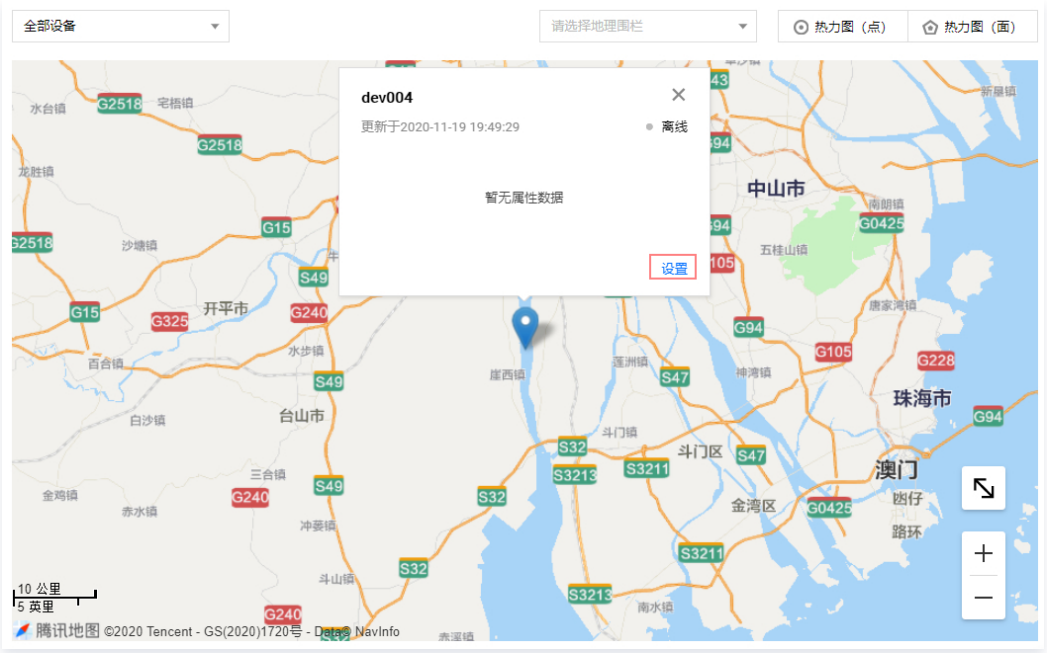
4. 进入空间可视化界面，上报位置信息的设备将以图标的形式显示在地图上。



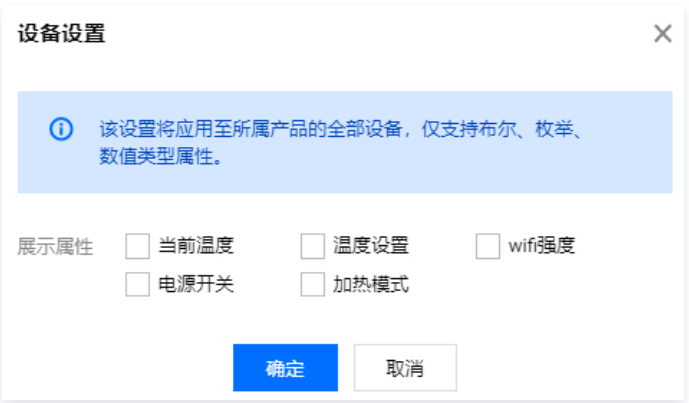
5. 首次进入该位置空间时，单击地图上的“设备图标”，将会在设备弹窗显示“暂无属性数据”。



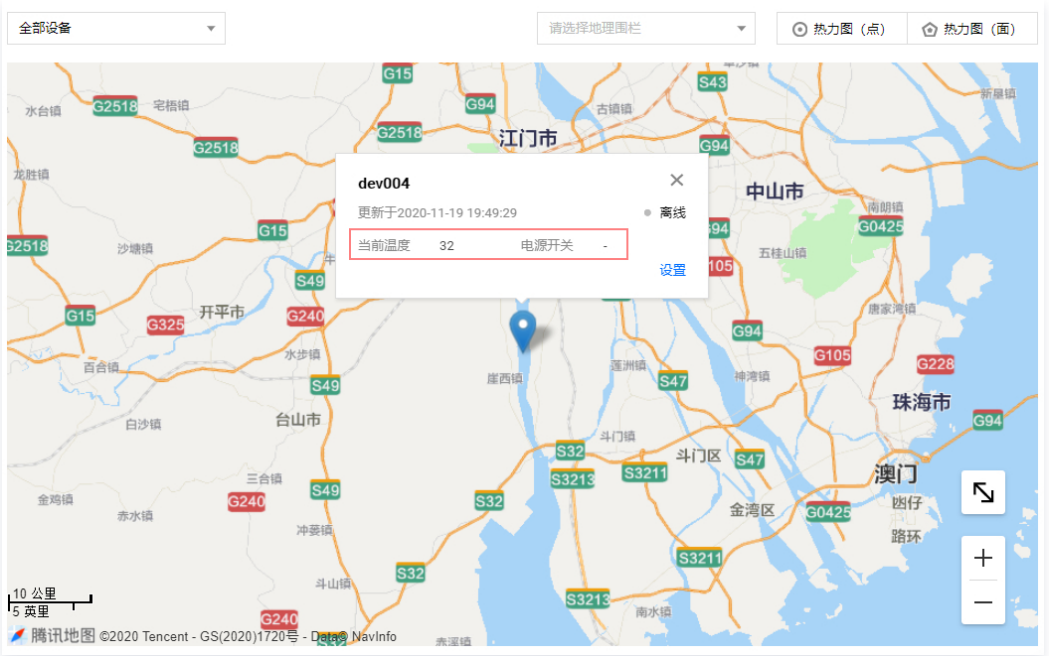
6. 此时需要在设备弹窗右下角单击**设置**。



7. 在“设备设置”弹窗中选择您需要在设备弹窗内显示的属性数据。

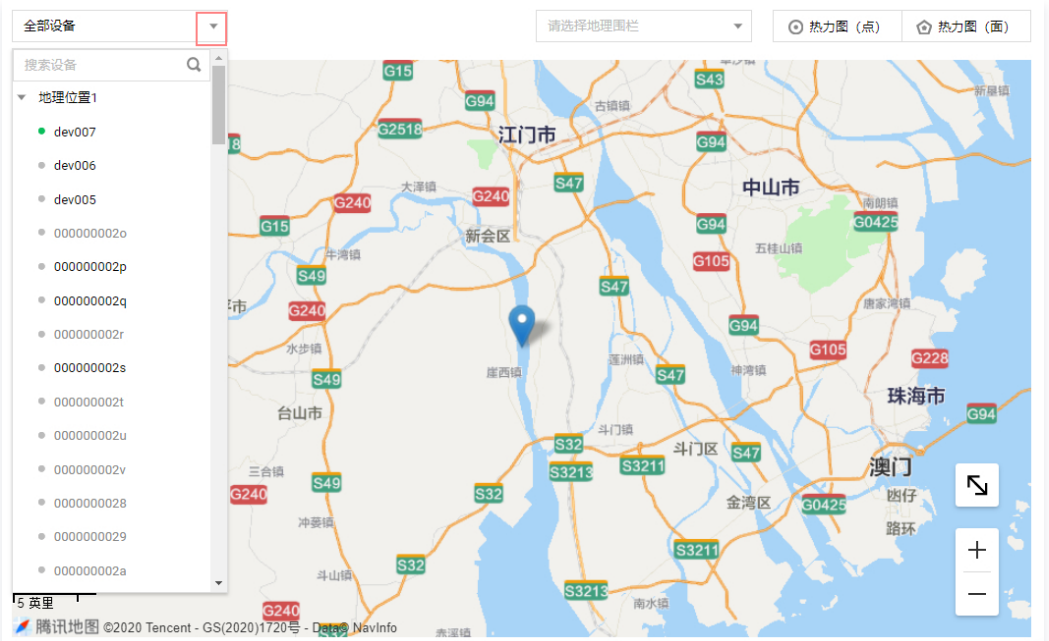


8. 单击**保存**即可设置完毕。再次进入查看该设备图标的数据显示，可以查看刚被设置为需显示的属性数据。

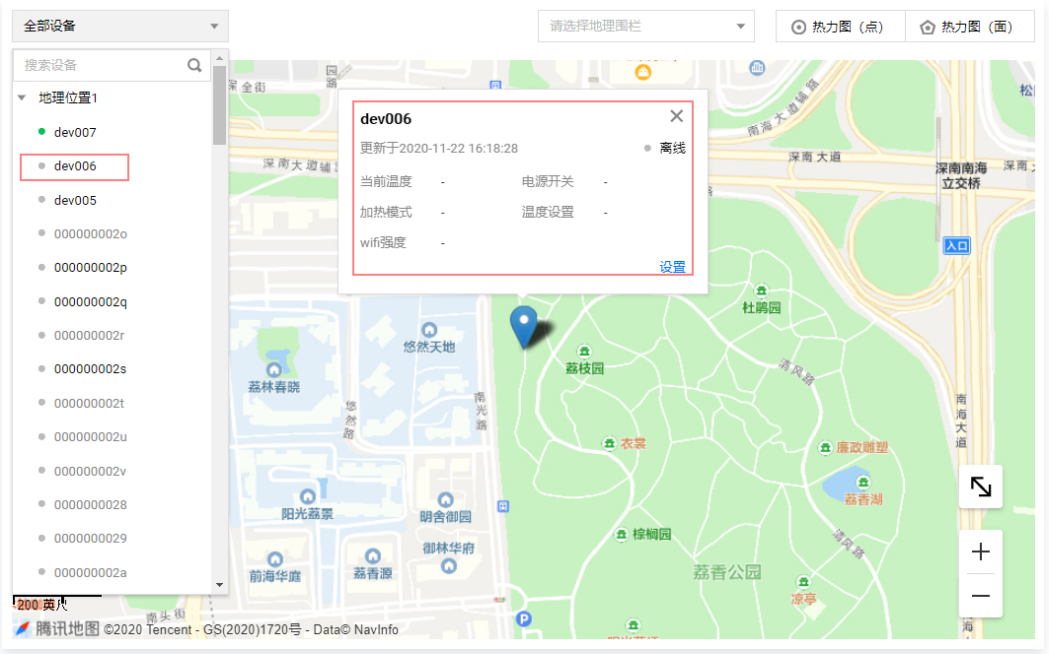


查看指定设备位置

1. 进入设备可视化 步骤4 页面进行继续操作。
2. 展开地图左上角下拉框，可查看空间项目关联产品下的所有设备。

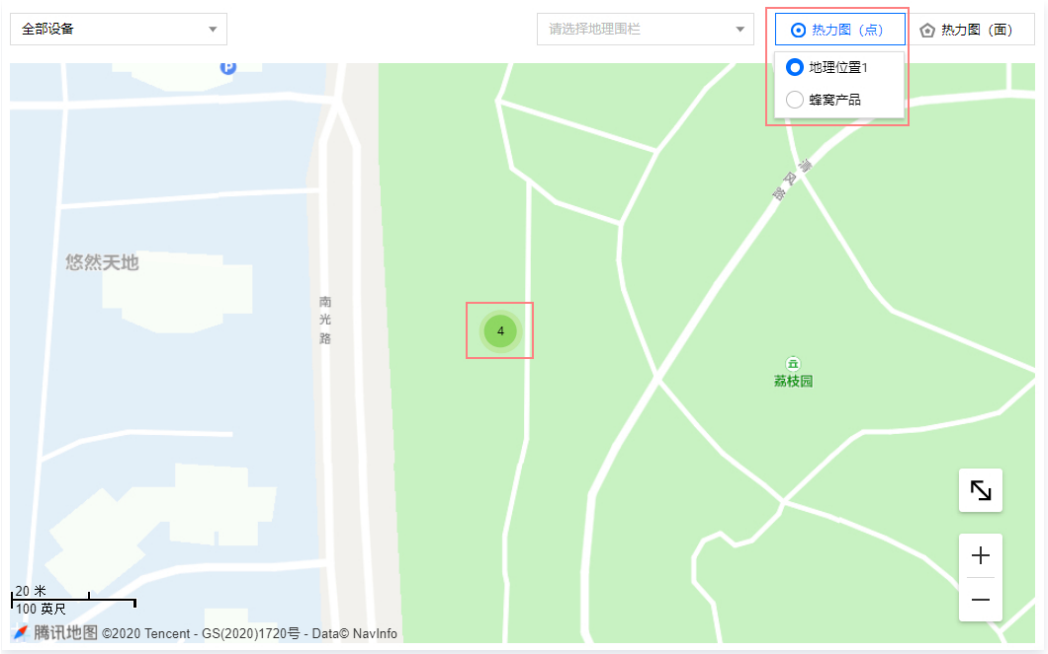


3. 单击选择某个指定设备，将跳转到该设备所在位置，并展示设备属性状态数据。



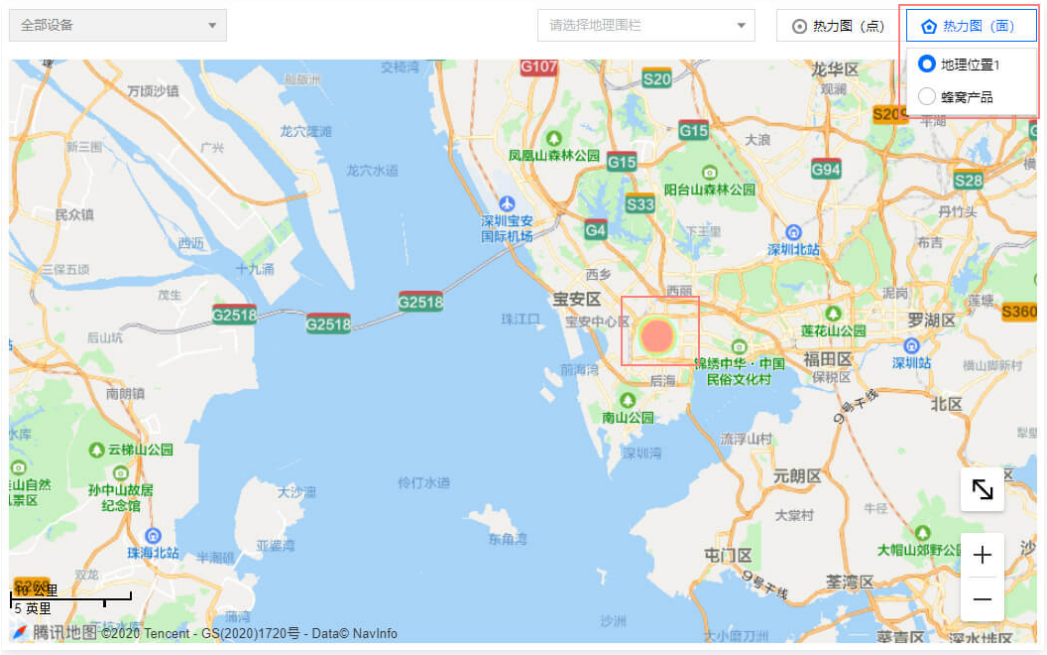
热力图（点）

在地图上单击“热力图（点）”功能，选择需展示的某个产品，进入热力图（点）聚合查看状态。



热力图（面）

在地图上单击“热力图（面）”功能，选择需展示的某个产品，进入热力图（面）聚合查看状态。



地理围栏

最近更新时间：2024-12-31 18:00:53

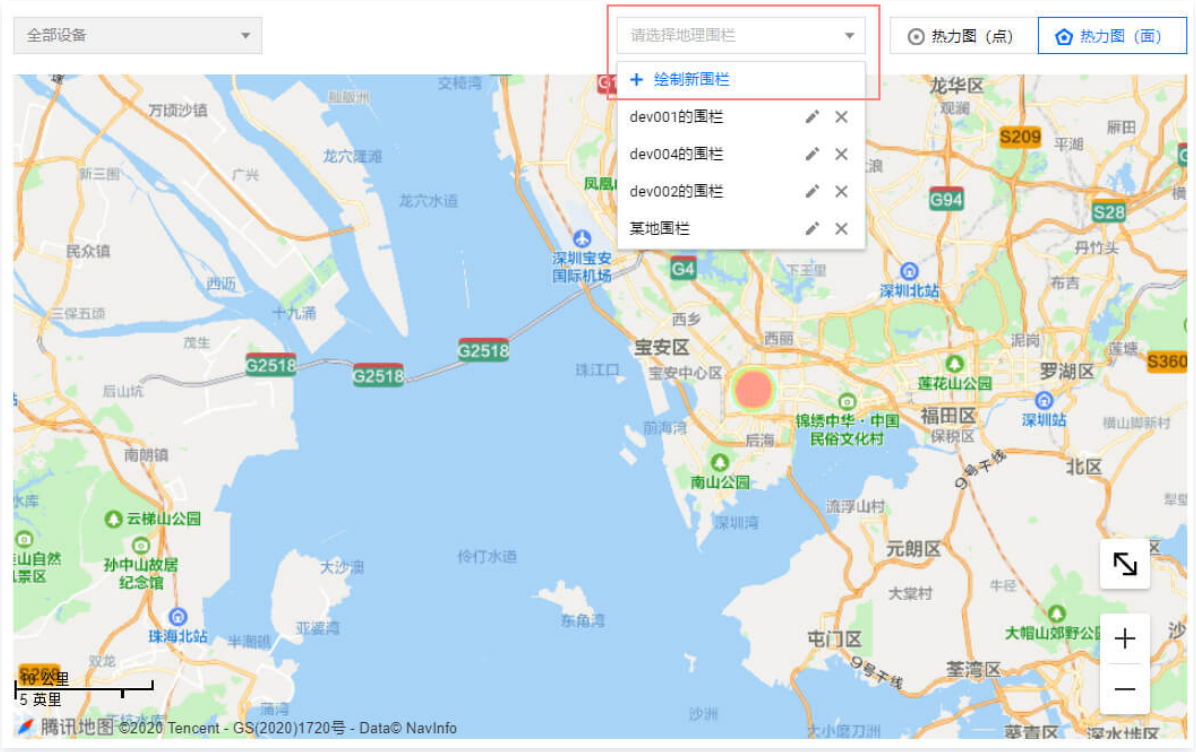
腾讯云物联网开发平台位置服务功能为用户提供地理围栏能力，支持圆形、多边形、行政区多种围栏类型，可广泛应用于定位器、智能手表、智慧牧场等需要地理范围限制告警的场景。本文档主要介绍地理围栏功能的使用方法。

前提条件

已完成 位置空间的创建，且该空间所关联的产品设备必须能够上报相关位置属性信息。具体位置上报请参见 功能介绍 和 物模型说明 。

创建围栏

1. 登录 物联网开发平台控制台，选择已创建的项目进入项目详情页。
2. 单击左侧菜单**位置服务**进入位置服务页面。
3. 在位置服务空间列表中，创建或进入某个空间，进入空间的可视化界面。
4. 单击**地理围栏 > 绘制新围栏**。



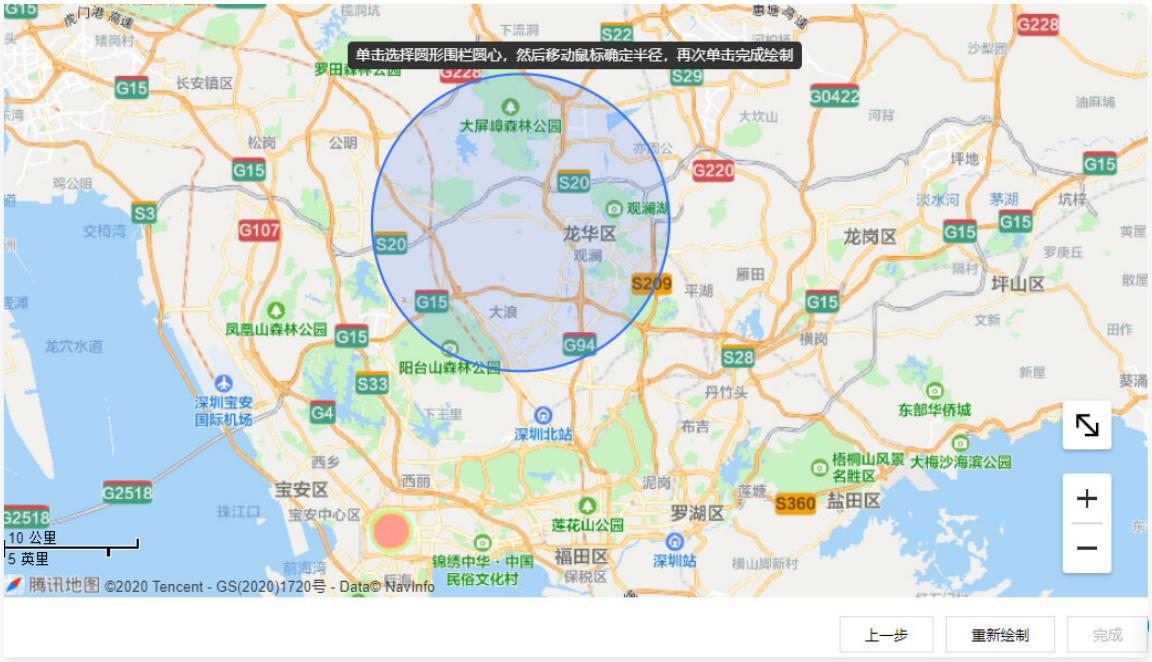
5. 进入围栏信息填写界面，填写相关信息。

- 围栏名称：设置围栏名称，支持中文、英文、数字、下划线的组合，最多不超过10个字符。
- 围栏类型：支持三种类型围栏，圆形、多边形以及行政区围栏，可根据实际业务需求选择。
- 关联设备：选择需要与创建的围栏相关联的设备，只有与围栏相关联的设备，才会有该围栏告警的触发。
- 触发条件：设置触发围栏报警的条件，进入围栏时或离开围栏时。

6. 单击**下一步**，开始绘制围栏，不同的围栏类型有不同的绘制方式。围栏绘制完成后，若所绘制的围栏若需要修改，可单击**重新绘制**进行再次绘制。

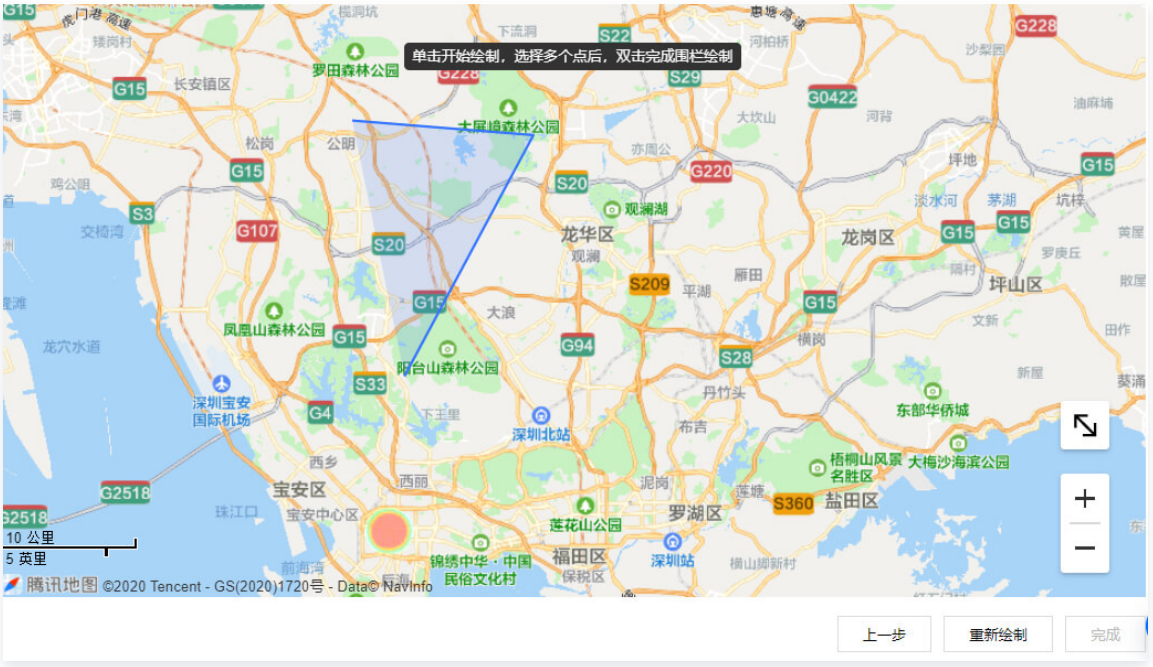
圆形围栏

单击选择圆形围栏圆心，然后移动鼠标确定半径，再次单击完成绘制。



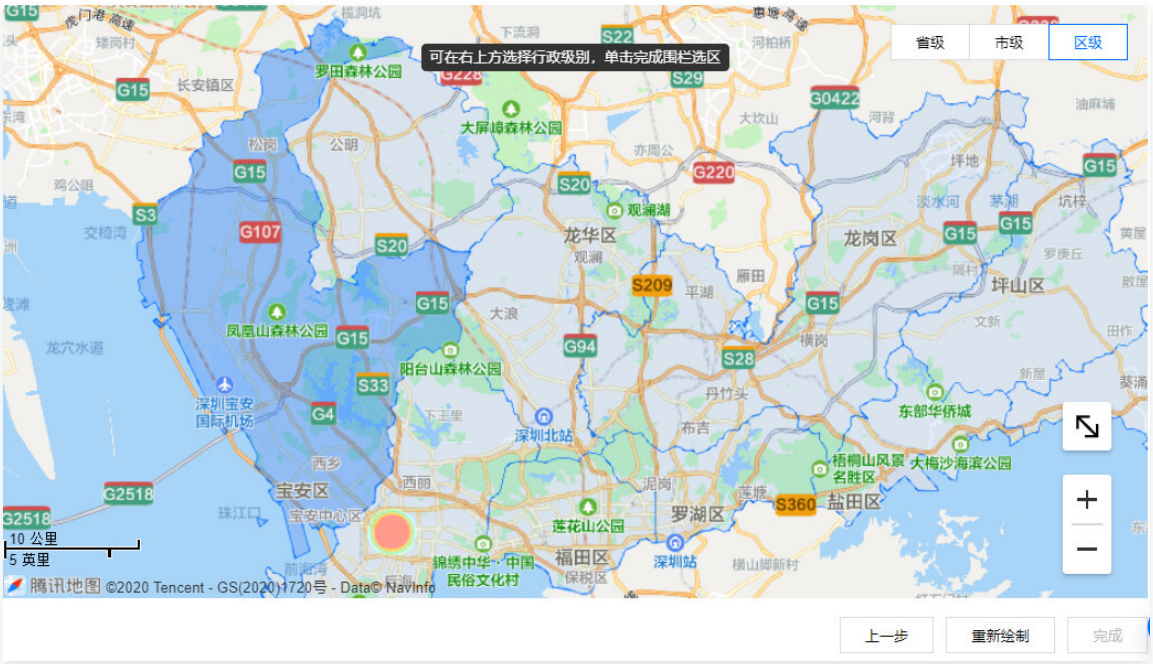
多边形围栏

单击开始绘制，选择多个点后，双击完成围栏绘制。



行政区围栏

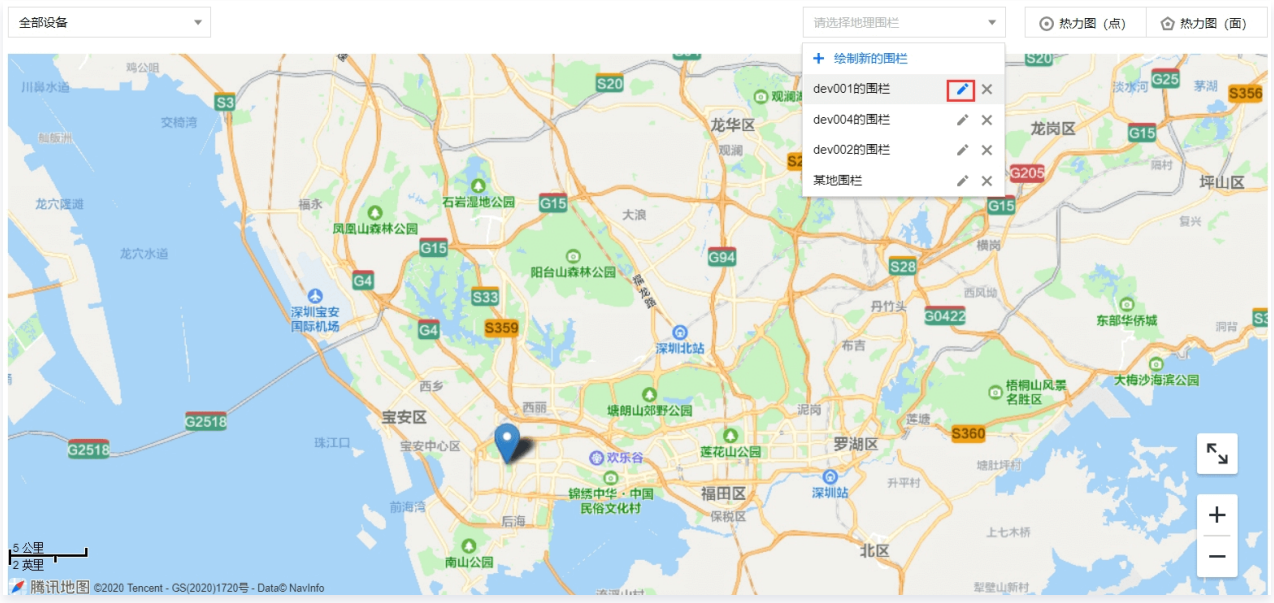
可在右上方选择行政级别（省-市-区），单击完成围栏选区。



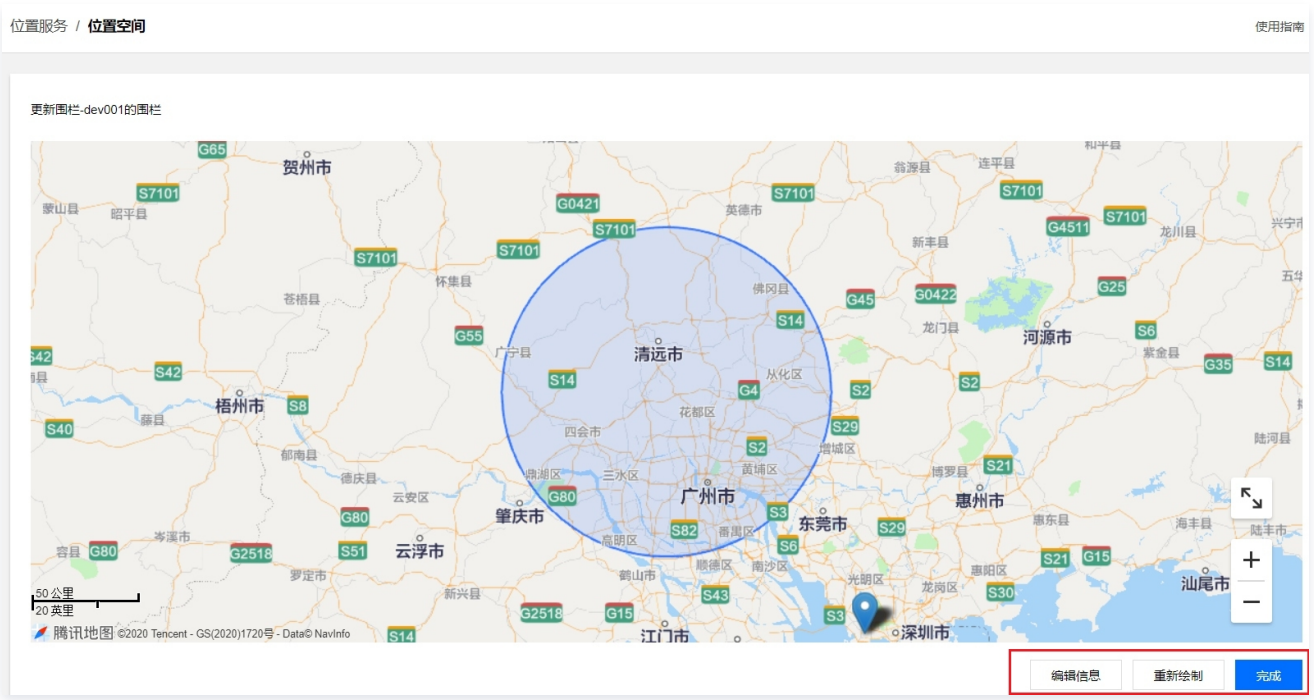
7. 绘制围栏后，单击**完成**，将提示“围栏添加成功”，即可完成围栏创建。

更新围栏

1. 完成 围栏创建 后，在围栏下拉框内，单击某个地理围栏的“编辑”图标。



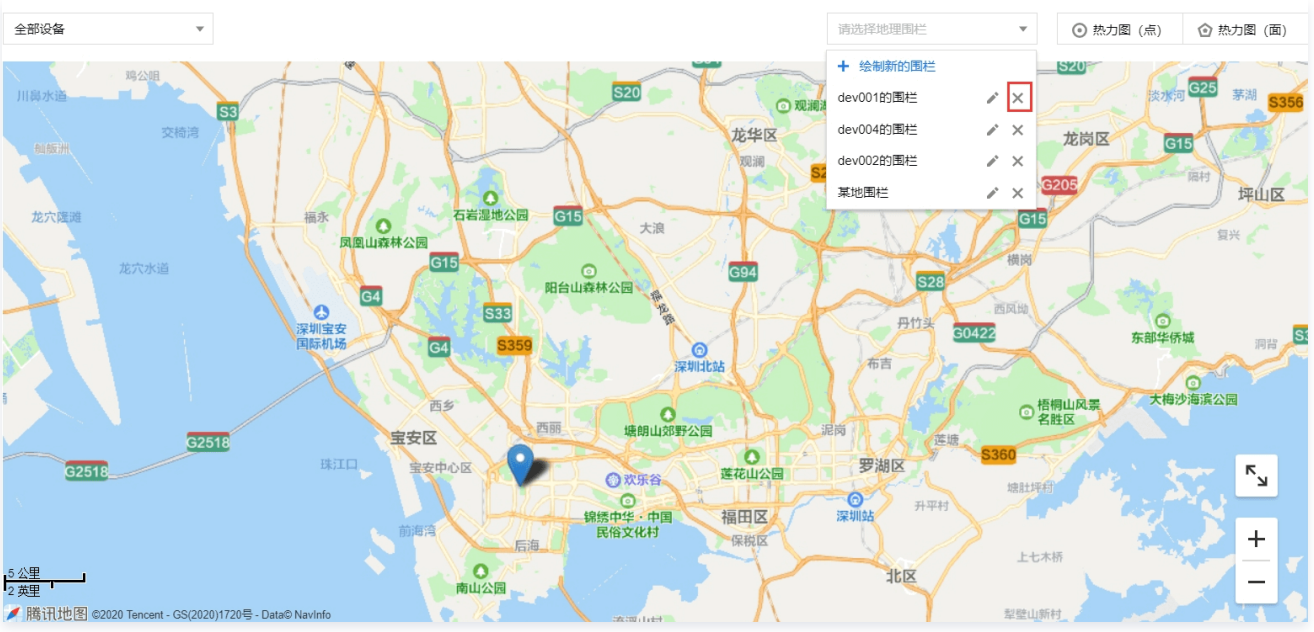
2. 进入围栏更新界面，可单击**编辑信息**修改围栏配置相关信息；单击**重新绘制**后将删除原有围栏，绘制新围栏。



3. 更新操作完成后，单击**完成**，即可完成围栏相关信息的更新。

删除围栏

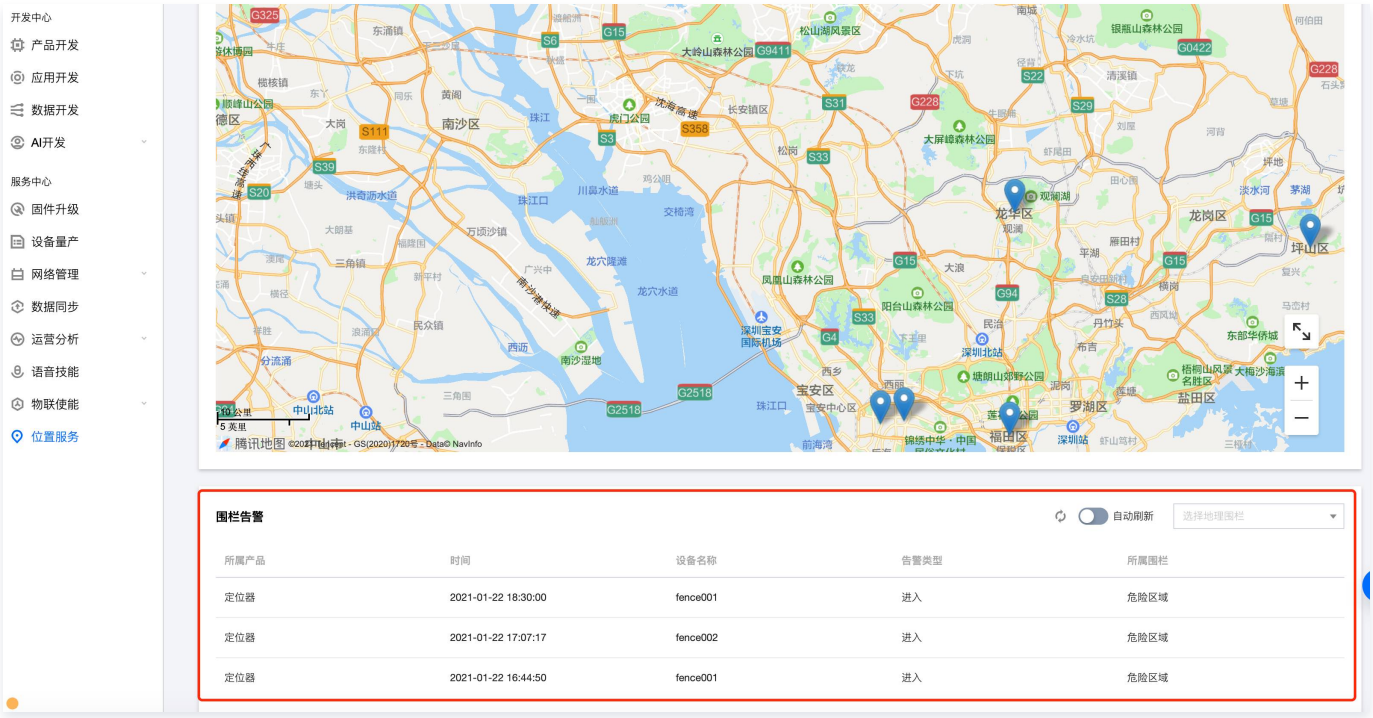
1. 完成 围栏创建 后，在围栏下拉框内，单击某个地理围栏的“删除”图标。



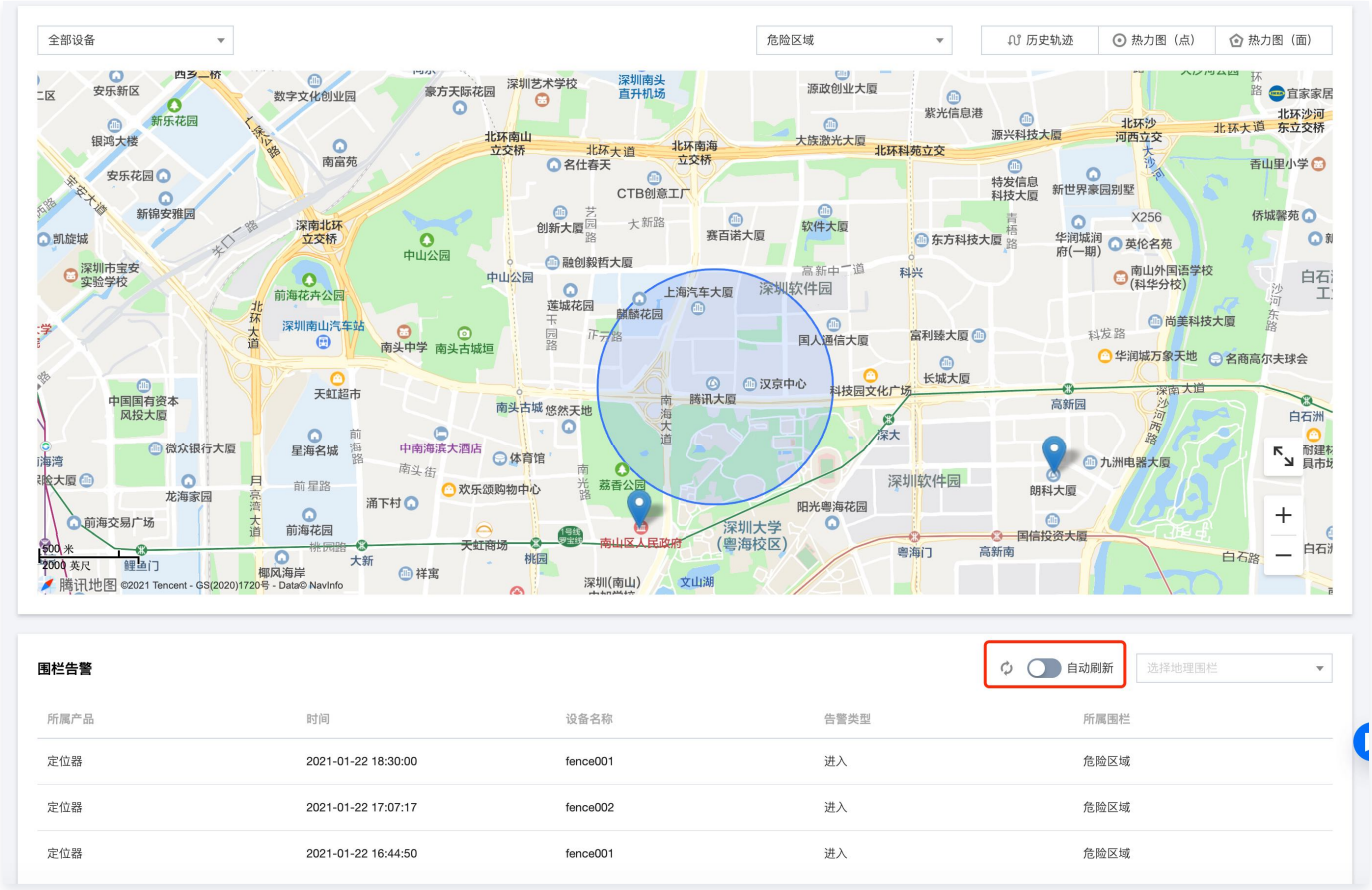
2. 弹出“确认删除围栏”提示框，单击**确认**，将删除该地理围栏。

围栏告警事件查询

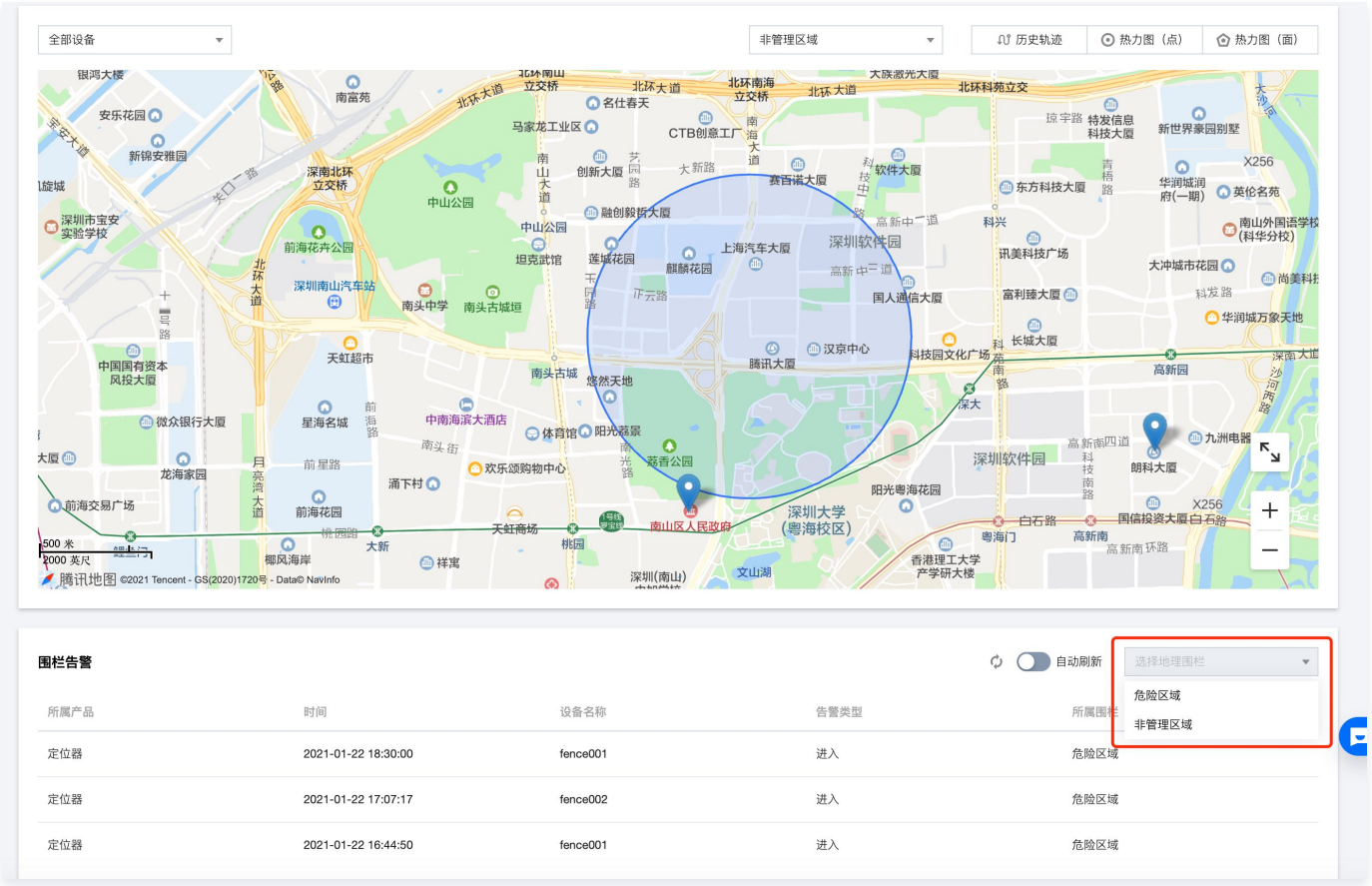
1. 当与创建的地理围栏关联的设备，在设备进入或离开围栏时，平台将提供围栏告警的推送，可在界面下方的围栏告警列表查询设备的围栏告警事件。



2. 在围栏告警列表内，可通过手动刷新或自动刷新收取新的围栏告警事件。单击“手动刷新”的图标，则将刷新一次围栏告警列表，加载最新告警事件。若开启**自动刷新**，则列表将在每5s内刷新一遍告警事件。



3. 同时，为了方便用户管理各围栏下的事件告警，还可以通过右侧选择具体围栏，查看某一特定围栏下的告警事件。



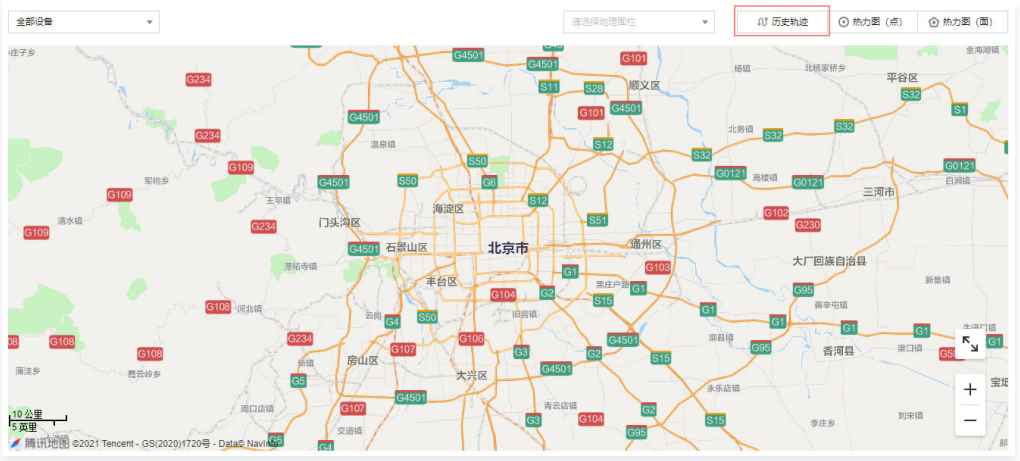
历史轨迹

最近更新时间：2024-12-31 18:00:53

腾讯云物联网开发平台位置服务功能支持保存用户设备的历史轨迹点位置及历史轨迹可视化查询。

选取历史轨迹

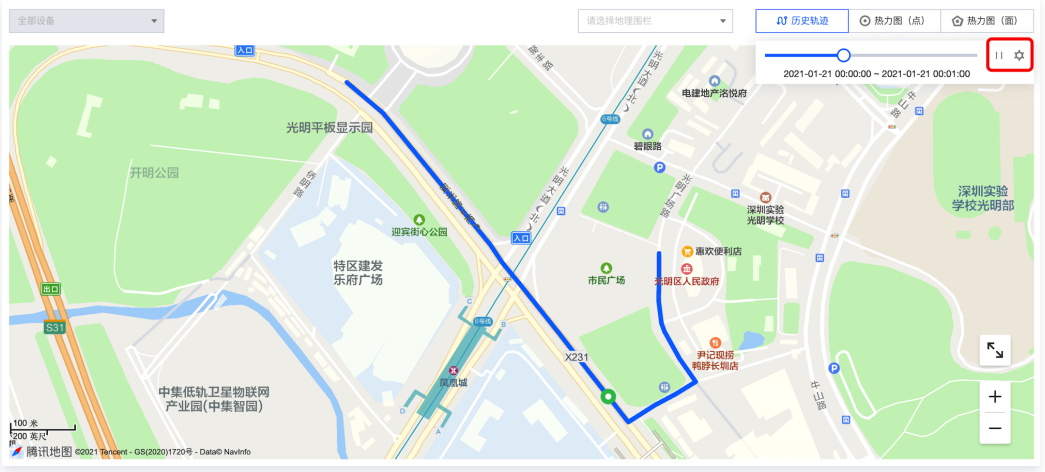
1. 登录 物联网开发平台控制台，选择已创建的项目进入项目详情页。
2. 单击左侧菜单**增值服务 > 位置服务**进入位置服务页面。
3. 在位置服务页面中，单击右上方**历史轨迹**进入历史轨迹选取界面。



4. 在历史轨迹选取界面中，选取所需查询的历史轨迹。

- **轨迹时间段**：您可以选择查看一个月内的具体时间段。
- **选择设备**：目前控制台最多同时支持三个设备的轨迹查询。
- **定位精度**：输入历史轨迹定位精度，默认是60米，用户可修改。如果定位精度误差预估超过该阈值，则在轨迹中不予展示。

5. 选取完毕后，单击**确定**即可生成历史轨迹，同时右上方会弹出历史轨迹的播放进度条。



6. 单击播放进度条右侧的“播放”图标，即可开始播放历史轨迹，设备也将开始根据轨迹点进行移动。

说明：

若您需要查看其他时间段或其他设备的历史轨迹，可单击播放进度条右侧的 重新选取历史轨迹。

相关物模型说明

最近更新时间：2024-12-31 18:00:53

GPS 定位物模型

- **属性标识符**：GPS_Info。
- **类型**：结构体。
- **参数说明**

字段名称	是否必选	描述
latitude	是	GPS 纬度；数值范围：-90 – 90；单位：度；6位小数点。
longtitude	是	GPS 经度；数值范围：-180 – 180；单位：度；6位小数点。

- **示例代码**

```
{
  "clientToken": "***",
  "method": "report",
  "params": {"GPS_Info":{"longtitude":112.59014,"latitude":22.28014}}
}
```

GPS 定位-扩展物模型

- **属性标识符**：GPS_ExtInfo。
- **类型**：结构体。
- **参数说明**

字段名称	是否必选	描述
latitude	是	GPS 纬度；数值范围：-90 – 90；单位：度；6位小数点。
longtitude	是	GPS 经度；数值范围：-180 – 180；单位：度；6位小数点。
altitude	否	• 海拔，数值型。 • 数值范围：-5000 – 99999 。 • 初始值：0 。 • 单位：m。
gps_speed	否	• GPS 速度，整形。 • 数值范围：0 – 1000 。 • 初始值：0 。 • 单位：km/h。
direction	否	• 方向角。 • 数值范围：0 – 360。 • 初始值：0。 • 单位：度。
location_state	否	定位状态，整形。 • 0：无效 。 • 1：有效。
satellites	否	卫星数，整形。
gps_time	否	GPS 时间，时间型；时间戳精度到秒，从卫星上采集的时间。

collect_time	否	采集时间，时间型；时间戳精度到秒，采集到设备数据的时间。

- **示例代码**

```
{
    "clientToken": "***",
    "method": "report",
    "params": {"GPS_ExtInfo":
{"longtitude":112.59014,"latitude":22.28014,"altitude":200,"gps_speed":80,"direction":30}}
}
```

> **说明：**
> 若您需要自定义设置以上参数时，纬度（lat）、经度（lon）必须定义，其他参数可以根据实际情况进行添加或删除。

蜂窝定位物模型

- **目标：**对于2G、4G等设备，可上报单个基站信息到云端定位。
- **属性标识符：**Cell_Info。
- **类型：**结构体。
- **参数说明**

字段名称	是否必选	描述
mcc	是	基站国家码（460）。
mnc	是	基站网络码（00）。
lac	是	基站小区号（5位十进制数）。
cid	是	基站 ID（5位十进制数）。
rss	是	基站信号强度，单位dbm。
networktype	是	• 1：GSM • 2：CDMA • 3：WCDMA • 4：TD_CDMA • 5：LTE
collect_time	否	设备采集到基站信息的时间。

- **示例代码**

```
{
    "clientToken": "***",
    "method": "report",
    "params": {"LBS_BS":"mcc:460;mnc:13824;lac:3;cid:33:rss:-85;networktype:1"}
}
```

Wi-Fi 定位物模型

- **目标：**对于 Wi-Fi 类设备，可上报附近多个 Wi-Fi 路由器设备的 MAC 地址到云端完成定位。
- **属性标识符：**Wifi_Info。
- **类型：**结构体。
- **参数说明**

字段名称	是否必选	描述

Mac	是	String 型，Wi-Fi 路由器的 MAC。
Rssi	是	int 型，Wi-Fi 路由器的信号强度。

- **示例代码**

```
{
    "clientToken": "***",
    "method": "report",
    "params":{"Wifi_Info":[{"Mac":"78a106a5b166","Rssi":-81},{"Mac":"608f5c66b3f7","Rssi":-82},
{"Mac":"282cb293a5d6","Rssi":-83}]}
}
```

LoRa Edge™ 服务介绍

最近更新时间：2024-12-26 18:07:33

简介

LoRa Edge™ 是 Semtech 推出的面向资产管理的产品组合，集成了超低功耗的 LoRa® 收发器、全球导航卫星系统（GNSS）和 Wi-Fi 扫描技术。LoRa Edge 解决方案与基于云的定位服务相结合，创建了一个独特的系统架构，可以同时实现精准定位和低功耗。
LoRa Edge 能够提供覆盖室内和室外的定位能力。传统的基于芯片的地理定位设备需要承载巨大的计算量，导致它们往往只有数周至数月的电池寿命。而 LoRa Edge 解决方案可以在定位设备和云平台之间分配计算量，使设备可拥有长达数月甚至数年的电池寿命，因此更为适用于物联网应用，包括工业、楼宇、家居、农业、交通运输和物流等许多领域。
LoRa Edge 能够为户外和室内应用提供超低功耗地理定位，典型精度为10米 – 50米。在最佳条件下，GNSS 地理定位或在开放环境中的性能可以达到5米精度；当室内 AP 指纹数据库覆盖良好时，Wi-Fi 地理定位的性能甚至可以达到3米精度。基于该系统的定位方案，可根据客户端具体需求进行量身定制，其终端电池寿命可达到数月，乃至数年，其模组成本也可做到小于5美元，公有网络及私有网融合实现室内外连续覆盖。

腾讯云与 Semtech 合作为中国用户提供 LoRa Edge 服务

2022年初，腾讯云与 Semtech 宣布，双方达成最终协议，LoRa Cloud™ 调制解调器以及地理定位服务已可通过腾讯云物联网开发平台为客户提供服务。
LoRa Cloud™ 作为 LoRa Edge™ 地理定位平台的一部分，已集成至腾讯云物联网开发平台，支持中国的用户快速地将基于 LoRa Edge 的物联网设备连接到云端，并结合腾讯地图高可信、高覆盖的 Wi-Fi 定位能力，为中国的企业及开发者提供灵活、低功耗、高性价比的地理定位服务方案。
LoRa Edge 的 GNSS 及 Wi-Fi 扫描数据将在腾讯云物联网开发平台直接进行解算获得位置信息，无需访问海外的 LoRa Cloud 站点，在提升访问速度的同时，也进一步满足相关应用的合规性要求。
LoRa Edge 的 Wi-Fi 扫描数据将使用腾讯地图高可信、高覆盖的 Wi-Fi 定位服务进行解析。腾讯地图在多个C端产品的规模覆盖，以及与手机厂商的深度合作中，积累了丰富的WiFi连接、识别数据，可为终端设备提供更为精准的室内外定位解析结果。
用户可以利用腾讯连连小程序、IoT Enable 等功能快速开发地理定位相关的特色应用。

使用说明

除了 LR1110，LoRa Edge 的其他产品也都将在腾讯云物联网开发平台获得支持。

- 使用 LoRaWAN® 连接的设备，详细操作指南见 LoRa LR1110 快速入门 。
- 不使用 LoRaWAN® 连接的设备，使用与 LoRa Edge 相关的物模型，设备使用物模型协议上报相应的 JSON 字段即可。
  需要登录 控制台 操作：进入目标产品的**物模型定义** > **添加标准功能** > **通用类型** > **勾选“wifi 定位”、“GNSS 导航电文”**。
- 腾讯云物联网开发平台提供 LoRa Edge 定位解析接口，第三方云平台可以直接调用该云 API，请求入参使用GNSS 及 Wi-Fi 扫描数据，可获得位置解析结果。

自主品牌小程序开发
概述

最近更新时间：2024-08-26 14:55:51

当前消费物联网领域更多的是通过 App 去管理、操控设备，主要存在以下两大问题。

- 对消费者来说，存在着 App 下载门槛高，打开率低的问题。
- 对设备厂商来说，通过产品的智能化升级以便更好地连接消费者，为消费者提供更便捷的服务，但实际上 App 在实际应用时打开率并不是特别高，导致厂商无法获取更多的消费者信息反馈。

针对以上两个问题，微信小程序可有效解决 App 带来的问题，消费者无需下载安装 App，扫码使用即可。为了更好的提升用户体验，腾讯云物联网开发平台秉持开放策略，向开发者开放小程序 SDK，以便需要自主品牌小程序的开发者，能快速基于小程序 SDK 构建自主品牌小程序，降低研发成本，并提升消费者用户体验。

平台功能

小程序 SDK 将通用能力进行封装，开发者无需投入资源研发，小程序 SDK 包含以下功能：

- 设备配网
- 设备管理
- 长连接通信
- 应用端 API 封装

快速入门

最近更新时间：2024-10-21 14:47:11

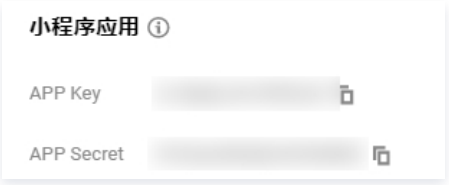
腾讯云物联网开发平台为开发者提供一个 Demo 小程序参考，开发者可以按本文档的指引流程部署并体验一个属于自己的 Demo 小程序。Demo 小程序以开源的方式向开发者开放，便于开发者在开发自主品牌小程序时进行参考。在开始之前需完成以下操作：

- 已注册 腾讯云账号 并完成 实名认证 。
- 已安装 微信开发者工具 。
- 已安装 Node.js 。
- 已在物联网开发平台中 创建产品 。

步骤1：创建应用

1. 登录腾讯云 物联网开发平台，选择公共实例或您购买的标准企业实例。
2. 在项目列表中选择项目，进入项目详情页面。
3. 在左侧菜单中选择**应用开发**，单击**新建应用**，进入新建应用页面。
4. 填写应用信息。

   - 应用名称：必填，根据实际业务填写便于识别的应用名称。
   - 备注：非必填，填写应用的备注信息（选填）。
5. 单击**保存**，页面提示保存成功，回到应用开发页面。
6. 单击应用列表中应用的名称，进入应用详情页面。
7. 记录小程序应用下显示的 APP Key 和 APP Secret。



8. 在页面下方的关联产品列表中，单击**关联**列的开关，使小程序与产品关联（开关为开启状态）。

> **注意：**
> 小程序只能对已关联产品下的设备进行绑定、控制等操作，请确保小程序已关联其需要操作的产品。若小程序尝试绑定、控制未关联的产品下的设备，会出现“APP对操作该产品无权限”的错误提示。

步骤2：注册、配置小程序

1. 前往 微信公众平台 注册小程序。
2. 登录微信公众平台的小程序后台，选择**开发管理 > 开发设置**。
3. 记录开发者 ID 中显示的 **AppID（小程序 ID）**。

4. 在服务器域名中单击**开始配置**（若曾经配置过则单击**修改**），根据页面指引完成身份确认。

5. 填写服务器信息。

   - request 合法域名：`https://iot.cloud.tencent.com`
   - socket 合法域名：`wss://iot.cloud.tencent.com`

6. 单击**保存并提交**。

> ⚠ **注意：**
> 小程序只能与服务器域名列表中指定的域名进行网络通信。若未配置小程序的服务器域名，则小程序不能正常连接到物联网开发平台。在真机预览时若遇到接口出错，请确认此步骤域名是否配置正确。

步骤3：下载、配置 Demo 小程序

1. 前往 qcloud-iotexplorer-appdev-miniprogram-sdk-demo，选择 **Code** > **Download ZIP**，下载 Demo小程序代码到本地并解压。
2. 在 `demo/miniprogram/app.js` 文件中填写 步骤1 获得的 AppKey。

```
const APP_KEY = 'YOUR_APP_KEY_HERE'; // 填写 AppKey
```

3. 在 `demo/cloudfunctions/login/index.js` 文件中填写 步骤1 获得的 AppKey 与 AppSecret。

```
const APP_KEY = 'YOUR_APP_KEY_HERE'; // 填写 AppKey
const APP_SECRET = 'YOUR_APP_SECRET_HERE'; // 填写 AppSecret
```

4. 打开命令行，切换到 Demo 代码中的 `demo/miniprogram` 目录。
5. 执行以下命令安装依赖。

```
npm install
```

6. 运行微信开发者工具，在项目列表中单击“+”，进入创建小程序页面，填写项目信息。

- 项目名称：可自行填写。
- 目录：选择 Demo 代码中的 `demo` 目录。
- AppID：填写 步骤2 获取的 AppID。

7. 单击界面右下角的**新建**，项目创建完成后，进入微信开发者工具的主界面。选择菜单栏的**工具 > 构建 npm**，构建成功后界面提示完成构建。

8. 选择菜单栏的**项目 > 重新打开此项目**，以加载上一步构建的 npm 依赖。

步骤4：开通微信云开发并部署云函数

本步骤指导您使用微信云开发部署登录接口，以实现小程序用户登录物联网开发平台。

> ⚠ **注意：**
> 小程序 Demo 的登录功能依赖于微信云开发服务，微信云开发为用户提供了一定的免费额度，超出免费额度的部分需要收费，详情请参见 微信云开发计费说明 。
> 您也可以通过自建后端服务的方式实现小程序用户登录物联网开发平台，请参见 接入微信登录 。

1. 单击微信开发者工具主界面上方的**云开发**，进入开通小程序云开发页面。
2. 单击**开通**，进入创建云开发环境页面。
3. 填写云开发环境名称，单击**新建**。

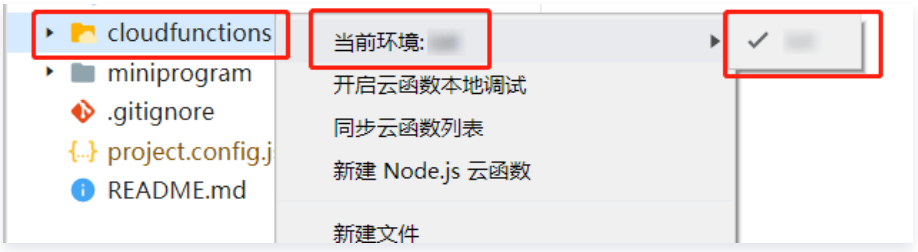
云开发环境创建完成后，自动进入云开发控制台页面。

4. 在云开发控制台界面右上角复制该环境的环境 ID。

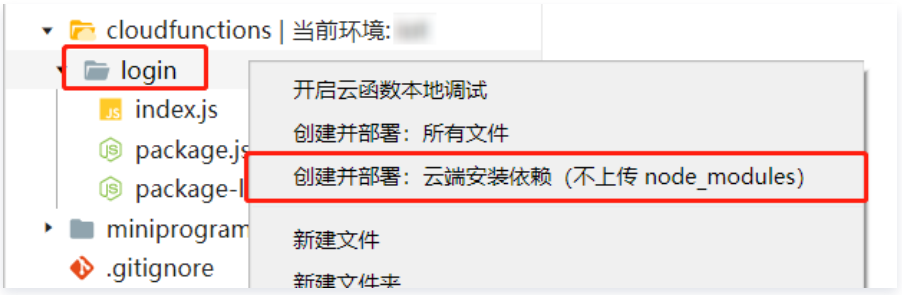
5. 在 `demo/miniprogram/app.js` 文件中填写上述步骤获得的环境 ID。

```
wx.cloud.init({
    env: '此处填写您的云开发环境 ID',
});
```

6. 单击微信开发者工具主界面上方的**编辑器**，打开文件编辑器。
7. 右键单击文件列表中的 **cloudfunctions**，选择**当前环境**，然后选择上述步骤中创建的云开发环境。



8. 展开文件列表中的 **cloudfunctions > login**，右键单击 **login**，选择**创建并部署：云端安装依赖（不上传 node_modules）**，上传完成后界面右上角提示上传云函数 login 成功。



步骤5：编译、运行小程序

1. 在微信开发者工具的主界面，单击界面上方的**编译**，编译完成后小程序在模拟器中运行（或单击界面上方的**预览**，编译完成后小程序在真机中运行）。
2. 小程序启动后会自动登录，并进入设备列表页面。

步骤6：通过小程序绑定设备

配网绑定设备

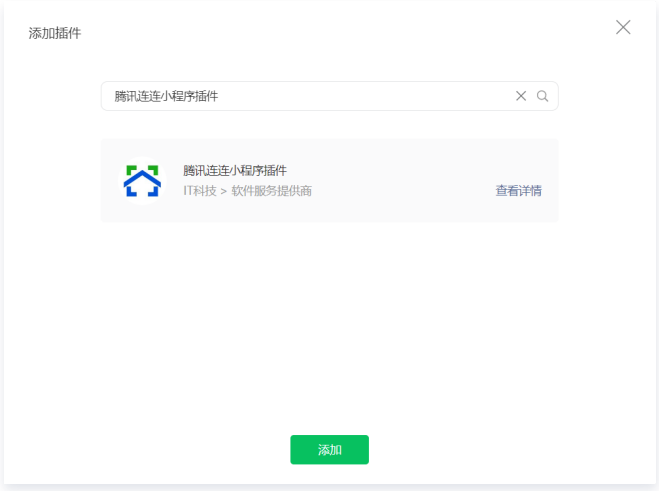
配网绑定设备需要配合真实设备（如 ESP8266 模组或 ESP32 模组）进行操作。关于配网协议与配网流程，详情请参见 配网开发概述 。在 Demo 小程序中进行设备配网的步骤如下：

自定义配网 UI 方式

1. 在小程序的设备列表页面，选择 **添加设备 > 自定义配网 UI 方式**。
2. 在配网方式列表中，选择设备支持的配网方式，然后根据页面指引进行设备配网。

配网插件方式

1. 登录小程序后台，进入“设置-第三方设置-插件管理”，单击**添加插件**。
2. 在输入框中输入**腾讯连连小程序插件**，单击**搜索**。
3. 选择**腾讯连连小程序插件**，单击**添加**。

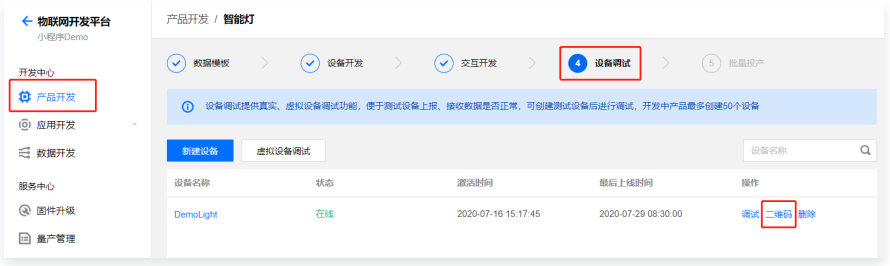


4. 参考 文档 中的步骤，将腾讯连连小程序插件添加到小程序中。
5. 在小程序的设备列表页面，选择**添加设备 > 配网插件方式**。
6. 输入要配网的设备对应的产品 ID。
7. 根据页面指引进行设备配网。

扫描设备调试二维码绑定设备

> **说明：**
> 物联网开发平台提供的设备二维码可以用于快速绑定真实设备，帮助开发者降低开发难度。量产后为了安全性，将会关闭二维码入口。

1. 登录腾讯云 物联网开发平台，选择公共实例或您购买的标准企业实例。
2. 在项目列表中选择产品所属的项目，进入项目详情页面。
3. 在产品列表中选择设备所属的产品，进入产品开发页面。
4. 单击页面上方的**设备调试**，进入设备调试页面。根据设备类型，按照以下步骤获取设备调试二维码。
   - **真实设备**：在设备列表中单击**二维码**，页面展示设备调试二维码。



   - **虚拟设备**：单击**虚拟设备调试**，进入虚拟设备调试页面，页面展示虚拟设备调试二维码。

5. 在小程序的设备列表页面，选择**添加设备 > 扫描设备调试二维码**，进入扫码页面。

6. 小程序扫描控制台页面展示的二维码。扫码成功后小程序提示绑定设备成功。单击**确定**回到设备列表页面，设备列表中显示已绑定的设备。

步骤7：通过小程序控制设备

1. 小程序进入设备列表页面，单击要控制的设备，进入设备操控页面。

2. 单击设备属性列表中列出的属性，可以修改对应的设备属性。

说明：
物联网开发平台提供了真实设备在线调试及虚拟设备调试功能，可通过控制台查询设备上报的当前数据、历史通信日志、事件及上下线记录等。具体操作请参见 设备调试 。

开发指南

最近更新时间：2023-05-24 17:13:04

本文介绍开发者如何基于物联网开发平台小程序 SDK（下称 SDK）构建自主品牌小程序，通过 SDK 使用平台提供的能力。

前提条件

1. 使用 SDK 需要您的运行环境包含 Node.js 以及 npm。
2. 登录 物联网开发平台控制台 创建项目及产品，具体操作请参见 产品定义 。
3. 登录 物联网开发平台控制台 获取 AppKey 和 AppSecret，具体操作请参见 获取应用 AppKey 和 AppSecret 。

接入 SDK

安装 SDK

1. 在微信小程序目录下，通过 npm 安装 SDK 。

```
npm install qcloud-iotexplorer-appdev-sdk
```

2. 在 微信开发者工具 的项目界面中，单击界面右上角的**详情**，选择**本地设置**，勾选“使用 npm 模块”。

3. 在微信开发者工具的项目界面中，选择菜单栏的**工具 > 构建 npm**，构建成功后界面提示完成构建。

配置小程序服务器域名

小程序 SDK 通过以下域名连接到物联网开发平台。

- request 域名：`https://iot.cloud.tencent.com`
- socket 域名：`wss://iot.cloud.tencent.com`

在接入小程序 SDK 时，需要将上述域名添加到小程序的服务器域名列表中，步骤如下。

1. 登录 小程序后台 。
2. 选择**开发管理 > 开发设置**。

3. 在“服务器域名”页面单击**开始配置**（若曾经配置过则单击**修改**），根据页面指引完成身份确认。

4. 填写服务器信息。

   - **request 合法域名：** `https://iot.cloud.tencent.com`
   - **socket 合法域名：** `wss://iot.cloud.tencent.com`

5. 单击**保存并提交**。

接入微信登录

使用物联网开发平台的 微信号注册登录 应用端 API，可以让小程序用户通过微信注册登录到物联网开发平台。出于安全考虑，不建议在小程序端直接调用微信号注册登录 API，请在自建的后端服务调用该 API，以避免密钥的泄露。

小程序接入微信登录需要实现以下流程：

1. 小程序调用 wx.login 获取临时登录凭证 `code` 。
2. 小程序将临时登录凭证 `code` 传递给自建的后端服务。
3. 后端服务调用微信服务端 API auth.code2Session ，通过临时登录凭证 code 获取小程序用户的 OpenID。
4. 后端服务调用物联网开发平台应用端 API 微信号注册登录 ，通过小程序用户的 OpenID 获取 AccessToken。
5. 后端服务将 AccessToken 返回给小程序。
6. 小程序将 AccessToken 传入小程序 SDK 以完成小程序 SDK 的初始化。

示例代码

请按照以下步骤并参考示例代码以实现接入微信登录：

1. 在 `AppDevSdk` 的构造函数参数中，将 `appKey` 的值修改为您实际获取的 AppKey。
2. 补充 `getAccessToken` 函数的实现。在 `getAccessToken` 函数中，您需要调用自建后端服务的登录接口，传入登录所需的信息，并将获取到的 AccessToken 放入 `getAccessToken` 函数的返回值。

```
// app.js
const { AppDevSdk } = require('qcloud-iotexplorer-appdev-sdk');

App({
  onLaunch() {
    this.sdk = new AppDevSdk({
      appKey: '此处填写您的 AppKey',
      getAccessToken: () => this.getAccessToken()
    });
```

```
    sdk.init().then(() => {
      // SDK 初始化完成
    });
  },

  async getAccessToken() {
    // 获取小程序登录凭证 code
    const code = await new Promise((resolve, reject) => {
      wx.login({
        success: (res) => resolve(res.code),
        fail: reject
      });
    });

    // 调用开发者自建后端服务，传入 code，获取物联网开发平台的 AccessToken
    // 请根据实际情况调整实现
    const resp = await new Promise((resolve, reject) => {
      wx.request({
        url: '开发者自建后端服务的 URL',
        method: 'POST',
        data: {
          code: code,
          nickName: '新用户的默认昵称',
          avatarUrl: '新用户的默认头像 URL'
        },
        header: {
          'content-type': 'application/json'
        },
        success(res) {
          resolve(res.data);
        },
        fail(err) {
          reject(err);
        }
      });
    });

    return {
      Token: resp.Token,
      ExpireAt: resp.ExpireAt,
    };
  }
})
```

使用 SDK

调用应用端 API

应用端 API 是物联网开发平台为了满足智能家居场景，为用户开发自有品牌的小程序或 App 而提供的云端服务，用户无需实现用户管理、设备管理、设备定时、家庭管理等基础能力，可通过调用应用端 API 快速完成移动应用端的开发。关于应用端 API 的更多信息，请参见 应用端 API 简介 。
SDK 对应用端 API 的调用过程进行了封装，发送请求时会自动带上公共参数 `AccessToken` 与 `RequestId` 。
以下示例代码以调用 获取用户绑定设备列表 应用端 API 为例。

```
sdk.requestApi(
  'AppGetFamilyDeviceList', // 请求应用端 API 的 Action 名
  { FamilyId: 'default' } // 请求应用端 API 的数据
)
  .then(data => {
    // 请求成功
```

```
    console.log(data);
  })
  .catch(err => {
    // 请求失败
    console.error(err);
  });
```

设备配网

SDK 目前支持 softAP 和 SmartConfig 和 simpleConfig 和 AirKiss方式进行设备配网。

softAP 配网

关于 softAP 方式配网的流程，详情请参见 softAP 配网开发 。具体参数说明和流程说明可参见小程序 SDK 章节中的 设备配网 部分。

```
const WifiConfConstants = require('qcloud-iotexplorer-appdev-plugin-wificonf-core').constants;

const {
   // 错误的中文描述
   WifiConfErrorMsg,
   // 步骤code
   WifiConfStepCode,
   // 步骤code的中文描述
   WifiConfStepDesp
} = WifiConfConstants;

/**
 * softap配网
 */
function SoftApConfigure({
  token,
  wifiInfo = {
    SSID: '';
    password: '';
    BSSID: '';
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = (data) => {
    reporter.info(data.code, data.detail);

    switch (data.code) {
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(1);
        break;

      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.SOFTAP_GET_DEVICE_SIGNATURE_SUCCESS:
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
```

```
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    onStatusChange({
      status: 'success',
      productId,
      deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    reporter.error(code, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfSoftAp'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError
  });
}

module.exports = SoftApConfigure;
```

SmartConfig 配网

关于 SmartConfig 方式配网的流程，详情请参见 SmartConfig 配网开发 。具体参数说明和流程说明可参见小程序 SDK 章节中的 设备配网 部分。

```
const WifiConfConstants = require('qcloud-iotexplorer-appdev-plugin-wificonf-core').constants;

const {
   // 错误的中文描述
   WifiConfErrorMsg,
   // 步骤code
   WifiConfStepCode,
   // 步骤code的中文描述
   WifiConfStepDesp
} = WifiConfConstants;

/**
 * smartconfig一键配网
 */
function SmartConfigConfigure({
  token,
  wifiInfo = {
    SSID: '';
    password: '';
    BSSID: '';
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
```

```
  onStatusChange,
}) {
  const onProgress = (data) => {
    reporter.info(data.code, data.detail);

    switch (data.code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    onStatusChange({
      status: 'success',
      productId,
      deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    reporter.error(code, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfSmartConfig'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError
  });
}

module.exports = SmartConfigConfigure;
```

simpleConfig 配网

关于 simpleConfig 方式配网的流程，详情请参见 simpleConfig 配网开发 。具体参数说明和流程说明可参见小程序 SDK 章节中的 设备配网 部分。

```
const WifiConfConstants = require('qcloud-iotexplorer-appdev-plugin-wificonf-core').constants;

const {
   // 错误的中文描述
   WifiConfErrorMsg,
   // 步骤code
   WifiConfStepCode,
```

```
    // 步骤code的中文描述
    WifiConfStepDesp
} = WifiConfConstants;

/**
 * SimpleConfig一键配网
 */
function SimpleConfigConfigure({
  token,
  wifiInfo = {
    SSID: '';
    password: '';
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = (data) => {
    reporter.info(data.code, data.detail);

    switch (data.code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    onStatusChange({
      status: 'success',
      productId,
      deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    reporter.error(code, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfSimpleConfig'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError
```

```
  });
}

module.exports = SimpleConfigConfigure;
```

AirKiss 配网

关于 AirKiss 方式配网的流程，详情请参见 AirKiss 配网开发 。具体参数说明和流程说明可参见小程序 SDK 章节中的 设备配网 部分。

```
const WifiConfConstants = require('qcloud-iotexplorer-appdev-plugin-wificonf-core').constants;

const {
   // 错误的中文描述
   WifiConfErrorMsg,
   // 步骤code
   WifiConfStepCode,
   // 步骤code的中文描述
   WifiConfStepDesp
} = WifiConfConstants;

/**
 * AirKiss一键配网
 */
function AirKissConfigure({
  token,
  wifiInfo = {
    SSID: '';
    password: '';
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = (data) => {
    reporter.info(data.code, data.detail);

    switch (data.code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    onStatusChange({
      status: 'success',
      productId,
      deviceName,
    });
```

```
  };

  const onError = async ({ code, detail }) => {
    reporter.error(code, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfAirKiss'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError
  });
}

module.exports = AirKissConfigure;
```

控制设备

通过调用 用户控制设备 应用端 API，可以对用户已绑定的设备发起控制操作。

```
// 指定要控制的设备的属性数据
const deviceData = {
  light_switch: 0,
};

sdk.requestApi('AppControlDeviceData', {
  ProductId: '要控制的设备的产品 ID',
  DeviceName: '要控制的设备的设备名称',
  Data: JSON.stringify(deviceData) // 控制报文 JSON
})
  .then(data => {
    // 请求成功
      console.log(data);
  })
  .catch(err => {
    // 请求失败
    console.error(err);
  });
```

长连接通信能力

SDK 支持通过 WebSocket 为用户订阅所绑定设备的上报数据以及状态信息，SDK 初始化时默认自动连接 WebSocket 服务端。

通过设备 ID 列表订阅

```
sdk.subscribeDevices([
  'Product1/Device1',
  'Product1/Device2',
  'Product2/Device3'
]);
```

通过设备列表订阅

`subscribeDevices` 函数支持传入设备列表（如 获取用户绑定设备列表 应用端 API 返回的设备列表）以订阅设备信息，要求参数数组中的设备信息包含 `DeviceId` 字段。

```
sdk.requestApi('AppGetFamilyDeviceList', { FamilyId: 'default' })
  .then(data => {
    sdk.subscribeDevices(data.DeviceList);
  })
  .catch(err => {
    // 请求失败
      console.error(err);
  });
```

监听设备上报数据及状态信息

调用 SDK 的 监听事件 接口，可以监听 WebSocket 订阅的设备上报数据及状态信息，请参照以下示例代码。

```
const { EventTypes } = require('qcloud-iotexplorer-appdev-sdk').AppDevSdk.constants;

// 监听设备上报数据推送
sdk.on(EventTypes.WsReport, ({ deviceId, deviceData }) => {
  console.log('websocket device report', deviceId, deviceData);
});

// 监听设备在线状态变更推送
sdk.on(EventTypes.WsStatusChange, ({ deviceId, deviceStatus }) => {
  console.log('websocket device status change', deviceId, deviceStatus);
});

// 监听设备控制推送
sdk.on(EventTypes.WsControl, ({ deviceId, deviceData }) => {
  console.log('websocket device control', deviceId, deviceData);
});
```

获取用户基本信息

通过读取 SDK 对象上的属性，可以获取用户的基本信息，请参照以下示例代码。

```
// sdk.isLogin: boolean; 用户是否登录
console.log(sdk.isLogin);

// sdk.userId: string; 用户 ID
console.log(sdk.userId);

// sdk.nickName: string; 用户昵称
console.log(sdk.nickName);

// sdk.userInfo: UserInfo; 用户详细信息
console.log(sdk.userInfo);
console.log(sdk.userInfo.Avatar);

// sdk.getDefaultFamilyId: () => Promise< string >; 获取用户默认的家庭 ID
sdk.getDefaultFamilyId().then(familyId => console.log(familyId));
```

UserInfo 数据结构说明

属性名	属性描述	类型
Avatar	头像。	string

CountryCode	国家代码。	string
Email	邮箱。	string
NickName	昵称。	string
PhoneNumber	电话号码。	string
UserID	用户 ID。	string

SDK Demo

- Demo 项目 GitHub 地址：qcloud-iotexplorer-appdev-miniprogram-sdk-demo 。
- Demo 使用指引：请参见 快速入门 。

小程序配网插件

最近更新时间：2024-12-26 18:07:33

基础介绍

> **说明：**
> - 小程序基础库版本默认要求 ⩾ 2.15.0，但如果需要使用"标准蓝牙广播协议"，基础库版本要求 ⩾ 2.22.1。
> - 使用前建议先阅读小程序 插件介绍 的内容。

目前插件功能主要分为两部分。

- 插件 SDK `(pluginSdk)`：全局单例，提供登录态管理、应用端接口调用、蓝牙等能力。
- 设备绑定组件：目前支持 Wi-Fi 类设备配网以及蓝牙设备绑定。

使用指南

1. 申请使用

在小程序管理后台的**设置 > 第三方服务 > 插件管理**中搜索"腾讯连连小程序插件"并添加。待腾讯云物联网开发平台相关工作人员审批通过相关申请后即可使用。
更多使用说明参见 小程序使用插件 介绍。

2. 引入插件

使用插件前，使用者要在 `app.json` 中声明需要使用的插件。如需使用 "设备绑定组件"，请在 `app.json` 上额外声明位置信息权限。

```
{
  "permission": {
    "scope.userLocation": {
      "desc": "您的位置信息将用于小程序位置接口的效果展示"
    }
  },
  "plugins": {
    "iotexplorer-weapp-plugin": {
      "version": "插件最新的版本号",
      "provider": "wxb711dd9e4296e7f6"
    }
  }
}
```

3. 初始化插件

在使用插件任意功能前，请务必调用 `pluginSdk.init` 进行初始化，详见 pluginSdk API 参考 。
使用建议
对于 OEM 小程序，如需使用"设备绑定组件"，建议：

- 先在 `app.js` 上实例化 AppDevSdk，详情参见 SDK 初始化文档。
- 在分包上引用并初始化插件时传入 AppDevSdk 实例（小程序规定每个包大小不能超过2M，插件大小目前为1M左右，所以建议不要占用主包的空间）。
- 需要等待 `pluginSdk.init` 初始化完成后，再渲染配网组件 `<DeviceConfiguraiton>`。

示例代码

```
// device-configuration-plugin.json
{
  "usingComponents": {
    "device-configuration": "plugin://iotexplorer-weapp-plugin/DeviceConfiguration"
  }
}

// device-configuration-plugin.wxml
<device-configuration wx:if="{{pluginInitFinish}}" props="{{deviceConfProps}}" />
```

```
<view wx:else>插件未初始化</view>

// device-configuration-plugin.js
const app = getApp();

Page({
  data: {
    deviceConfProps: { /* ... */ },
    pluginInitFinish: false,
  },

  async onLoad() {
    await pluginSdk.init({
      appDevSdk: app.sdk,
    });

    this.setData({
      pluginInitFinish: true,
    });
  },
});
```

4. 适配插件

因为小程序插件无法调用 `wx.onAppHide` 、`wx.onAppShow` 接口，所以需要依赖引用方主动通知，当对应事件触发时，调用 pluginSdk 对应的方法来通知插件。

示例代码

```
wx.onAppShow(function () {
  pluginSdk.onAppShow();
});

wx.onAppHide(function () {
  pluginSdk.onAppHide();
});
```

pluginSdk API 参考

基本属性

参数	类型	说明
appDevSdk	AppDevSdk 实例	AppDevSdk 参考文档
uin	string	获取当前已登录用户的 userId。
userInfo	object	获取当前已登录用户的信息。
userInfo.Avatar	string	头像
userInfo.Email	string	邮箱（用户已绑定邮箱才能获取）。
userInfo.NickName	string	昵称
userInfo.PhoneNumber	string	昵称（用户已绑定手机才能获取）。
userInfo.UserID	string	用户 ID

初始化

1. 接口定义

```
pluginSdk.init(options: Object);
```

2. **输入参数**

参数名称	类型	必填	参数描述
appDevSdk	Object	是	AppDevSdk 实例。
getLoginAccessToken	() => Promise<{ Token: string; ExpireAt: number }>	是	获取登录 Token。
wifiConfProtocolList	Array<WiFi配网插件>	否	按需注入 WiFi 配网插件，支持的协议类型查看下文。
bluetoothAdapter	Object	-	蓝牙适配器 BluetoothAdapter 实例。
bluetoothDeviceAdapterList	Array<蓝牙设备适配器>	否	按需注入蓝牙协议 DeviceAdapte，支持的协议类型查看下文。

> ⚠ **注意：**
> appDevSdk、getLoginAccessToken 两个参数为互斥关系，初始化时必须传入其中一个。

wifiConfProtocolList 参数说明

按需注入 WiFi 配网插件，目前支持的插件如下：

- AirKiss
- SimpleConfig
- SmartConfig
- SoftAp
- BleCombo
- Qrcode - 二维码配网

bluetoothDeviceAdapterList 参数说明

按需注入蓝牙设备适配器，目前支持的蓝牙 DeviceAdapter 如下：

- 标准蓝牙协议 - LLSync
- 标准蓝牙广播协议 - LLSync Advert
- 标准 BLE 辅助配网 - LLSync Blecombo
- 标准 BLE 辅助配网 - LLSync 双路通信
- 乐鑫 BLE 辅助配网 - Blufi

> ⚠ **注意：**
> 使用蓝牙辅助配网时请注意，除了注入 WiFi 配网插件外，还要注入相应协议的蓝牙 DeviceAdapter。

示例代码

```
const { pluginSdk } = requirePlugin('iotexplorer-weapp-plugin');

/*
* WiFi设备相关
*/
const SimpleConfigPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-simpleconfig').default;
const AirKissPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-airkiss').default;
const SmartConfigPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-smartconfig').default;
const SoftApPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-softap').default;
const BleComboPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-blecombo').default;
const QrcodePlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-qrcode').default;
const {
  // 适用于使用BleCombo乐鑫协议的设备
```

```
  BleComboEspDeviceAdapter,
  // 适用于使用标准蓝牙辅助协议的设备
  BleComboLLSyncDeviceAdapter,
  // 适用于使用双路通讯协议的设备
  BleComboDualModeDeviceAdapter,
} = require('qcloud-iotexplorer-appdev-plugin-wificonf-blecombo');

/**
* 蓝牙设备相关
*/
// 标准蓝牙协议
const { StandardDeviceAdapter4Mp: StandardDeviceAdapter } = require('qcloud-iotexplorer-bluetooth-adapter-
llsync');
// 标准蓝牙广播协议
const StandardAdvertDeviceAdapter = require('qcloud-iotexplorer-bluetooth-adapter-llsync-advert');

Page({
  async onLoad() {
    console.log('设备适配器', [
      StandardDeviceAdapter,
      StandardAdvertDeviceAdapter,
      BleComboEspDeviceAdapter,
      BleComboLLSyncDeviceAdapter,
      BleComboDualModeDeviceAdapter,
    ]);
    try {
      // Todo 这里需要补充appDevSdk初始化的代码
      await pluginSdk.init({
        // 注意：appDevSdk、getLoginAccessToken、getLoginTicket 三个为互斥关系，初始化时只需传入一个即可
        appDevSdk: getApp().appDevSdk, // 对于OEM小程序，建议先在app.js上初始化，然后调用plugin.init时注入

        // 当使用getLoginAccessToken时，需要传入appKey
        // appKey: 'xxx',
        // getLoginAccessToken: async () => {
        //   // ...await some asynchronous 异步请求Token
        //   return {
        //     Token: 'xxx',
        //     ExpireAt: 't',
        //   };
        // },

        // getLoginTicket: async () => {
        //   // ...await some asynchronous 异步请求Ticket
        //   return {
        //     Ticket: 'xxx'
        //   };
        // },

        // 往插件里按需注入WiFi设备配网协议
        wifiConfProtocolList: [
          AirKissPlug,
          SimpleConfigPlug,
          SmartConfigPlug,
          BleComboPlug,
          SoftApPlug,
          QrcodePlug,
        ],
        // 往插件里按需注入蓝牙DeviceAdapter
        // 注意：BleCombo比较特殊，除了需要注入"BleComboPlug"外，因为涉及到蓝牙，也要注入对应协议的"DeviceAdapter"
        bluetoothDeviceAdapterList: [
          StandardDeviceAdapter,
```

```
          StandardAdvertDeviceAdapter,
          BleComboEspDeviceAdapter,
          BleComboLLSyncDeviceAdapter,
          BleComboDualModeDeviceAdapter,
        ],
      });
    } catch (err) {
      /* 调用方自行处理出错后的逻辑 */
      console.error('插件初始化失败', err);
    }

    // 因为小程序插件无法使用onAppShow、onAppHide 所以需要依赖调用方来实现，当对应事件触发时，调用pluginSdk对应的方法
    // 请务必实现
    wx.onAppShow(this.onAppShow);
    wx.onAppHide(this.onAppHide);
  },
  onUnload() {
    wx.offAppShow(this.onAppShow);
    wx.offAppHide(this.onAppHide);
  },
  onAppShow() {
    pluginSdk.onAppShow();
  },
  onAppHide() {
    pluginSdk.onAppHide();
  }
});
```

设备绑定组件

目前插件使用 Taro 开发，基于 Taro 的限制，需要将参数都放入 `props` 属性上 ，如下示例代码：

示例代码

```
<device-configuration
  props="{{deviceConfProps}}"
/>
```

```
Page({
  data: {
    deviceConfProps: {
      productId: 'xxx',
      familyId: 'xxx',
      wifiConfProps: {
        extraInfo: {}
      },
      onSuccess() {},
      onError() {},
      onComplete() {},
      navigateBack: () => wx.navigateBack(),
    }
  },
})
```

组件参数

参数名称	类型	必填	参数描述
productId	string	是	产品 ID

navigateBack	function	是	因为小程序插件无法调用 `wx.navigateBack` 接口，需要依赖引用方主动通知。
familyId	string	否	如不传入，组件会先让用户选择家庭。
onSuccess	function	否	配网成功时回调，回调参数 `{ deviceId: string; familyId: string }` 。
onError	function	否	配网失败时回调。
onComplete	function	否	配网完成时回调；仅在配网成功时，用户单击 `完成` 按钮时才触发 回调参数 `{ deviceId: string; familyId: string }` 。

应用端 SDK
小程序 SDK
SDK 初始化

最近更新时间：2024-12-26 18:07:33

SDK 初始化后，才能通过 SDK 使用物联网开发平台提供的云端能力。初始化 SDK 需要依次完成以下步骤：
1. 调用 `AppDevSdk` 的构造函数并传入配置项，创建 AppDevSdk 实例对象。
2. 调用实例对象的 `init` 方法初始化 SDK。

安装依赖

```
npm install qcloud-iotexplorer-appdev-sdk
```

初始化 SDK

AppDevSdk 构造函数

1. 接口描述

接口功能：调用 `AppDevSdk` 的构造函数并传入配置项，创建 SDK 对象。
接口声明：`new AppDevSdk()`

2. 输入参数

参数名	类型	必填	参数描述
getAccessToken	function	是	获取 accessToken 的回调，返回一个 Promise，其值为 微信号注册登录 应用端 API 的返回结果。
appKey	string	是	在 物联网开发平台控制台 > 应用开发 > 小程序应用中申请的 AppKey。
debug	boolean	否	是否为调试模式，默认为：false。开启调试模式后会开启打印调试日志。
wsConfig	WsOptions	否	WebSocket 的配置。
plugins	AppDevPlugin[]	否	导入到 SDK 的配网插件数组。
apiUrl	string	否	物联网开发平台的接口 URL，默认为：`https://iot.cloud.tencent.com/api/exploreropen/`，一般无需更改。
defaultUin	string	否	未登录状态下的默认 uin，以及调试模式下的固定 uin，默认为：unknown，一般无需更改。
reporter	function	否	SDK 运行日志的回调函数。

- **getAccessToken 回调函数说明**

  SDK 初始化时，将调用 `getAccessToken` 回调函数以取得物联网开发平台的 AccessToken。开发者需要在 `getAccessToken` 函数中实现获取 AccessToken 的流程，请参见 接入微信登录 。
- **WsOptions 数据结构**

属性名	类型	必填	属性描述
autoReconnect	boolean	否	WebSocket 断开后是否自动连接，默认为：true，自动重连每两秒尝试一次。
disconnectWhenAppHide	boolean	否	当 App.onHide 触发时，是否自动断开 WebSocket，默认为：true。
connectWhenAppShow	boolean	否	当 App.onShow 触发时，是否自动连接 WebSocket，默认为：true。

url	string	否	websocket 服务的 URL，默认为：wss://iot.cloud.tencent.com/ws/explorer。
heartbeatInterval	number	否	心跳包的发送间隔，单位毫秒，默认为：60000。

3. 输出参数

参数名	类型	必填	参数描述
-	Object	是	AppDevSdk 实例对象。

4. 示例代码

```
const sdk = new AppDevSdk({
  appKey: '此处填写您的 AppKey',
  getAccessToken: getAccessToken,
});

const getAccessToken = async () => {
  // 获取小程序登录凭证（code）
  const code = await new Promise((resolve, reject) => {
    wx.login({
      success: (res) => resolve(res.code),
      fail: reject
    });
  });

  // 获取小程序用户信息
  const userInfo = await new Promise((resolve, reject) => {
    wx.getUserInfo({
      success: (res) => resolve(res.userInfo),
      fail: reject
    });
  });

  // 调用开发者自建的后台服务端获取 AccessToken
  // 请根据实际情况调整实现
  const resp = await new Promise((resolve, reject) => {
    wx.request({
      url: '开发者自建的后台服务端 URL',
      data: {
        code: code,
        userInfo: userInfo
      },
      header: {
        'content-type': 'application/json'
      },
      success(res) {
        resolve(res.data);
      },
      fail(err) {
        reject(err);
      }
    });
  });

  return {
    Token: resp.Token,
    ExpireAt: resp.ExpireAt,
  };
```

```
};
```

SDK 初始化

```
sdk.init(options) => Promise<void>
```

调用后将依次执行：

1. 登录（调用 `getAccessToken` 回调函数，取得平台的 AccessToken）。
2. 连接 WebSocket。

`init` 函数可同时多次调用（返回同一个缓存的 Promise）。若一次执行未完成或已执行成功，多次调用后拿到的会是同一个 Promise。若执行失败，则该缓存的 Promise 在 reject 之后会被释放，再次调用则将重新执行。

参数说明

参数名	类型	必填	参数描述
reload	boolean	否	是否清理缓存的 Promise 并重新执行，默认为 false。

示例代码

```
const getAccessToken = async () => {
  // 获取小程序登录凭证（code）
  const code = await new Promise((resolve, reject) => {
    wx.login({
      success: (res) => resolve(res.code),
      fail: reject
    });
  });

  // 获取小程序用户信息
  const userInfo = await new Promise((resolve, reject) => {
    wx.getUserInfo({
      success: (res) => resolve(res.userInfo),
      fail: reject
    });
  });

  // 调用开发者自建的后台服务端获取 AccessToken
  // 请根据实际情况调整实现
  const resp = await new Promise((resolve, reject) => {
    wx.request({
      url: '开发者自建的后台服务端 URL',
      data: {
        code: code,
        userInfo: userInfo
      },
      header: {
        'content-type': 'application/json'
      },
      success(res) {
        resolve(res.data);
      },
      fail(err) {
        reject(err);
      }
    });
  });

  return {
```

```
    Token: resp.Token,
    ExpireAt: resp.ExpireAt,
  };
};

const sdk = new AppDevSdk({
  appKey: '此处填写您的 AppKey',
  getAccessToken: getAccessToken
});

// SDK 初始化
sdk.init();
```

调用应用端 API

最近更新时间：2024-12-26 18:07:33

应用端 API 是物联网开发平台为了满足智能家居场景，为用户开发自有品牌的小程序或 App 而提供的云端服务，包括用户管理、设备管理、设备定时、家庭管理等基础能力。关于应用端 API 的更多信息，请参见 应用端 API 简介 。

调用应用端 API

调用应用端 API 并获得响应数据。

- **接口定义**

```
sdk.requestApi(Action: string, payload?: object, options?: object) => Promise< response >
```

- **参数说明**

参数名	类型	必填	参数描述
Action	string	是	请求应用端 API 的 Action 名。
payload	object	否	请求应用端 API 的数据，会自动带上公共参数 `AccessToken` 与 `RequestId` 。
options	object	否	请求的选项，将透传给 wx.request 。

- **返回值**
  - 请求成功（code=0）：返回一个 resolved 的 Promise，其值为应用端 API 响应中的 `Response` 部分数据。
  - 请求失败：返回一个 rejected 的 Promise，其值的数据结构为：`{ code, msg, ...detail }` 。
- **示例代码**

```
sdk.requestApi('AppGetFamilyDeviceList', { FamilyId: 'default' })
  .then(data => {
    // 请求成功
    console.log(data);
  })
  .catch(err => {
    // 请求失败
    console.error(err);
  });
```

> **注意：**
> - 腾讯云物联网开发平台是基于家庭的设备体系，每个家庭有其对应的 `FamilyId` ，每台设备均归属一个家庭。
> - 开发者也可以选择不关注家庭这一概念，对所有需要传 `FamilyId` 的接口（例如 获取用户绑定设备列表 ）传入 `default` 作为 `FamilyId` ，SDK 会自动完成内部的家庭相关的逻辑（SDK 会为用户创建一个默认家庭，若 `FamilyId` 入参的值为 `default` ，SDK 会自动替换为用户默认家庭的 `FamilyId` ）。

设备配网

最近更新时间：2024-11-14 15:47:32

Wi-Fi 设备配网

概述

我们提供了设备配网 SDK，目前支持 SoftAP、SmartConfig、simpleConfig、AirKiss、BLE-Combo 这五种方式进行设备配网，详细的SDK使用及配网流程参考 设备配网 SDK 。

配网示例

腾讯连连中 `SoftAP` 配网过程如下，供开发者参考。

通过四步可以运行配网插件，以 `SoftAp` 配网为例，其余配网方式步骤相同，关于配网步骤和参数说明详见 设备配网 SDK 。

1. 安装依赖

```
npm install qcloud-iotexplorer-appdev-plugin-wificonf-softap
```

2. 注册插件

调用插件上 `install` 方法来注册插件，调用时需要传入实例化的小程序SDK。
插件注册时的名称分别为：wifiConfSoftAp、wifiConfSmartConfig、wifiConfSimpleConfig、wifiConfBleCombo、wifiConfAirKiss，之后可通过插件名称来获取插件实例。

```
const SoftApPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-softap');
SoftApPlug.install(appDevSdk);
```

3. 生成配网 Token

调用应用端 API 生成 Wi-Fi 设备配网 Token 来获取 Wi-Fi 设备配网 Token。

```
sdk.requestApi('AppCreateDeviceBindToken')
  .then((data) => {
    const bindDeviceToken = data.Token;
    console.log('wifi设备配网Token=', bindDeviceToken);
  });
```

4. 使用配网插件

从appDevSdk实例的 `plugins` 中获取已注册的插件实例，调用配网插件实例的 `start` 方法开始配网流程。
示例代码

```
sdk.plugins['wifiConfSoftAp'].start({
  wifiConfToken: bindDeviceToken,
  targetWifiInfo: wifiInfo,
  onProgress,
  onComplete,
  onError,
  autoRetry: true, // 自动处理故障流程
});
```

蓝牙设备配网

通过 qcloud-iotexplorer-bluetooth-adapter-llsync sdk， 小程序可以完成和标准蓝牙设备进行连接，绑定，控制等流程。通过下面的图片可以直观地了解整个流程:

您也可以通过官方的 小程序 SDK demo 的添加标准蓝牙协议（LLSync）的设备部分，来掌握连接标准蓝牙设备的流程。
下面是添加一个LLSync蓝牙设备的示例。

1. 创建一个蓝牙适配器

蓝牙适配器（bluetoothAdapter）可以用来搜索设备、连接设备等。代码如下:

```
import { BlueToothAdapter } from 'qcloud-iotexplorer-bluetooth-adapter';
```

```
import { LLSyncDeviceAdapter } from 'qcloud-iotexplorer-bluetooth-adapter-llsync';

// 关于appDevSdk文档，详见https://www.npmjs.com/package/qcloud-iotexplorer-appdev-sdk
const options = {
  appDevSdk, // 通过qcloud-iotexplorer-appdev-sdk得到的实例
}
LLSyncDeviceAdapter.injectOptions(options);
export const bluetoothAdapter = new BlueToothAdapter({
  deviceAdapters: [
    LLSyncDeviceAdapter,
  ],
});
```

2. 获取蓝牙设备列表

通过 bluetoothAdapter.startSearch方法，我们可以发现设备，获得设备列表。

```
  const serviceIds = [LLSyncDeviceAdapter.serviceId];
  await bluetoothAdapter.startSearch({
    serviceIds,
    onError: (error) => {
      console.log('----error', error);
      // 搜索设备出错
      bluetoothAdapter.stopSearch();
    },
    onSearch: (devices) => {
      console.log('searched devices', devices);
      if (devices.length > 0) {
        console.log('找到设备', devices); // 此时可以在页面上展示
      }
    },
    timeout: 1.4 * 15 * 1000,
  });
```

在上面的 onSearch 回调函数中，我们可以获得搜寻到的设备列表，这时可以将设备列表展示到页面上，供用户选择要连接哪个设备。

> ⚠ **注意：**
> 如果设备无法搜索到，请确认设备没有被绑定。

3. 连接设备

用户从上面获取到的设备中选择一个，并发起连接操作时，可以调用 `bluetoothAdapter.connectDevice` 方法进行连接。连接成功后会返回一个 deviceAdapter，可以用来向连接的设备发送 Wi-Fi，token 等数据。

> ⚠ **注意：**
> 如果在连接时提示没有权限操作该产品，请到控制台/应用开发对应用和产品进行关联。

```
try {
  // device 参数是上一步 onSearch 回调中获取 devices 数组的某一项
  const deviceAdapter = await bluetoothAdapter.connectDevice(device);

  if (!deviceAdapter) {
    throw {
      code: 'CONNECT_ERROR',
    }
  }
} catch (err) {
  console.error('连接到设备出错');
```

```
}
```

在这一步中，可以通过连接设备获得 deviceAdapter实例 ，通过 deviceAdapter，我们可以完成后续设备的绑定和解绑操作。

4. 绑定设备

绑定设备时，可以传入 familyId, roomId, 从而将设备绑定到特定的家庭和房间，绑定完成后，可以在设备列表中看到该设备。

```
try {
  const deviceId = await deviceAdapter.bindDevice({ familyId, roomId });
} catch (err) {
  console.log(err);
}
```

5. 鉴权设备

设备完成绑定后，并不能立即向蓝牙发送控制指令，在发送指令前需要先完成设备鉴权。

```
// ...连接设备操作

// 接下来连接鉴权
if (!deviceAdapter.authorized) {
    await deviceAdapter.authenticateConnection({
        deviceName: deviceName,
    });
}
```

6. 控制设备

完成绑定和鉴权设备之后，我们就可以向设备发送控制数据了，例如向设备下发一个 `property` 或者一个 `action` 。这一切都可以调用应用端API完成，详见 设备控制 。

> **注意：**
> 如果控制设备无效，可检查上一步鉴权设备是否成功。

7. 解绑设备

设备绑定后，也可以通过小程序发起删除设备的操作，这时，如果没有连接和鉴权，首先要连接和鉴权设备，如第4，第5步所示；然后调用 `unbindDevice` ，解绑完成后会断开连接，设备会恢复到未绑定的状态。

```
try{
  await deviceAdapter.unbindDevice({ familyId, deviceName });
} catch (err) {
  console.log(err);
}
```

8. 断开设备

我们可以通过 `deviceAdapter.disconnectDevice()` 断开设备连接：

```
await deviceAdapter.disconnectDevice()
```

9. deviceAdapter 事件

事件	描述	参数
connect	蓝牙设备连接时触发。	DeviceInfo

disconnect	蓝牙设备连接断开时触发。	DeviceInfo
authorized	蓝牙设备授权完成时触发。	{ version, mtu, otaVersion, }

迁移指南

最近更新时间：2022-02-24 15:18:48

从 v0.x 版本迁移到 v1.x 版本

更新项目依赖

更新小程序 SDK 到 v1.x 版本，需要更新项目依赖，请在项目目录下的命令行中执行以下命令：

```
npm install qcloud-iotexplorer-appdev-sdk@1
```

> **说明：**
> 如果您使用了微信开发者工具的 npm 支持，在更新项目依赖后，需要选择微信开发者工具菜单栏的**工具 > 构建 npm**以重新构建 npm 依赖。

调整导入 SDK 的方式

导入 v0.x 版本小程序 SDK 的方式为：

```
// 导入 SDK
const {
  QcloudIotExplorerAppDevSdk
} = require('qcloud-iotexplorer-appdev-sdk/qcloud-iotexplorer-appdev-sdk');

// 导入常量
const {
  EventTypes,
} = require('qcloud-iotexplorer-appdev-sdk/qcloud-iotexplorer-appdev-sdk');

const sdk = new QcloudIotExplorerAppDevSdk({
  // 参数略
});
```

小程序 SDK 升级到 v1.x 版本后，导入的方式需要调整为：

```
// 导入 SDK
const { AppDevSdk } = require('qcloud-iotexplorer-appdev-sdk');

// 导入常量
const { EventTypes } = AppDevSdk.constants;

const sdk = new AppDevSdk({
  // 参数略
});
```

调整配网代码

v0.x 版本小程序 SDK 内置支持 Softap 配网及 SmartConfig 配网。v1.x 版本小程序 SDK 增加了更多配网方式的支持，并将配网模块独立出来，开发者可以按需导入。关于小程序 SDK 具体支持的配网方式，请参见 设备配网 。
选择需要安装的配网插件，并在命令行中执行相应的安装命令。下面以安装 AirKiss 配网插件为例。

```
npm install qcloud-iotexplorer-appdev-plugin-wificonf-airkiss
```

向 SDK 注册配网插件，请参照以下代码（以 AirKiss 配网插件为例）。

```
import AirKissPlug from 'qcloud-iotexplorer-appdev-plugin-wificonf-airkiss';
```

```
AirKissPlug.install(sdk);
```

调用配网插件进行配网，请参照以下代码（以 AirKiss 配网插件为例）。

```
import { constants as WifiConfConstants } from 'qcloud-iotexplorer-appdev-plugin-wificonf-core';

const {
   WifiConfErrorMsg, WifiConfStepCode,
} = WifiConfConstants;

/**
 * airkiss一键配网
 */
export function AirKissConfigure({
  token,
  wifiInfo = {
    SSID: '';
    password: '';
    BSSID: '';
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = (data) => {
    reporter.info(data.code, data.detail);

    switch (data.code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    onStatusChange({
      status: 'success',
      productId,
      deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    reporter.error(code, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfAirKiss'].start({
```

```
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError
  });
}
```

错误处理

最近更新时间：2022-02-15 14:41:24

数据格式

SDK 所有接口的错误都经过标准化处理为 `{ code, msg, ...detail }` 的形式，具体取值根据接口的不同而不同。

全局错误码

> ⚠ **注意：**
> 下文中描述为一个对象的 detail，实际上是解构到错误对象当中的。例如 `INTERNAL_ERROR` 的具体 Error 为 `{ code: 'INTERNAL_ERROR', msg: Error.message, stack: Error.stack, error: Error }`

错误码	描述
ErrorCode.VERIFY_LOGIN_FAIL	未登录或登录态已失效。
ErrorCode.INTERNAL_ERROR	JS Error 和 detail: { stack, error }分别为错误堆栈和原始错误对象。
ErrorCode.GET_USERINFO_NEED_AUTH	调用 wx.getUserInfo 时用户未授权用户信息权限，遇到该错误时需要引导用户授权。 detail: { errMsg }，小程序 API 的原始错误信息。
ErrorCode.WX_API_FAIL	调用小程序 API 报错。 detail: { errMsg }，小程序 API 的原始错误信息。

调用应用端 API 错误码

除以上全局错误码，其余错误码为应用端 API 响应中的错误码。
具体错误码请查看对应的应用端 API 文档。同时接口的错误中会包含标识该次请求的 `detail.reqId`，可用来查询该次请求的详细日志。

长连接通信

最近更新时间：2024-10-08 18:37:11

订阅设备信息

- **接口定义**

  通过 WebSocket 监听服务端实时推送的设备上下线状态及属性数据。

```
sdk.subscribeDevices(deviceList: string[] | deviceInfo[]): Promise<void>;
```

- **参数说明**

参数名	参数描述	类型	必填
deviceList	设备 ID 列表，或设备信息列表（deviceInfo 需包含 DeviceId 字段）。	string[] | deviceInfo[]	是

- **示例代码**

  通过设备 ID 列表订阅。

```
sdk.subscribeDevices([
  'Product1/Device1',
  'Product1/Device2',
  'Product2/Device3'
]);
```

  通过设备列表订阅。

```
sdk.requestApi('AppGetFamilyDeviceList', { FamilyId: 'default' })
  .then(data => {
    sdk.subscribeDevices(data.DeviceList);
  });
```

手动建立长连接

手动连接 WebSocket。一般不需要调用，除非关闭了 `sdkOptions.disconnectWhenAppHide` 选项。

- **接口定义**

```
sdk.connectWebsocket() => Promise<void>;
```

- **示例代码**

```
sdk.connectWebsocket();
```

手动断开长连接

手动断开 WebSocket。一般不需要调用，除非关闭了 `sdkOptions.autoReconnect` 与 `sdkOptions.connectWhenAppShow` 选项。

- **接口定义**

```
sdk.disconnectWebsocket() => Promise<void>;
```

- **示例代码**

```
sdk.disconnectWebsocket();
```

监听事件

监听 WebSocket 事件。

```
sdk.on(type: EventTypes, listener: (...args) => void) => void;
```

- **参数**

参数名	参数描述	类型	必填
type	要监听的事件。	EventTypes	是
listener	事件触发时的回调函数。	(param) => void	是

- **示例代码**

```
const { EventTypes } = require('qcloud-iotexplorer-appdev-sdk').AppDevSdk.constants;

// 监听设备上报数据推送
sdk.on(EventTypes.WsReport, ({ deviceId, deviceData }) => {
  console.log('websocket device report', deviceId, deviceData);
});
```

取消监听事件

取消监听 WebSocket 事件。

- **接口定义**

```
sdk.off(type: EventTypes, listener: (...args) => void) => void;
```

- **参数说明**

参数名	参数描述	类型	必填
type	要取消监听的事件	EventTypes	是
listener	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。	(param) => void | null	否

- **示例代码**

```
const { EventTypes } = require('qcloud-iotexplorer-appdev-sdk').AppDevSdk.constants;

const listener = ({ deviceId, deviceData }) => {
  console.log('websocket device report', deviceId, deviceData);
};

// 监听设备上报数据推送
sdk.on(EventTypes.WsReport, listener);

// 取消监听设备上报数据推送
sdk.off(EventTypes.WsReport, listener);
```

事件列表

事件类型	描述	参数

EventTypes.WsReport	设备上报数据。	{ deviceId, deviceData }
EventTypes.WsControl	设备控制数据。	{ deviceId, deviceData }
EventTypes.WsStatusChange	设备在线状态变更。	{ deviceId, deviceStatus }
EventTypes.WsEventReport	设备事件上报。	{ Payload }
EventTypes.WsActionReport	设备行为上报。	{ Payload }
EventTypes.WsActionPush	设备行为下发。	{ Payload }
EventTypes.WsPush	WebSocket 推送原始数据。	{ push, action, params }
EventTypes.WsError	WebSocket 发生错误。	WebSocket error 事件的原始错误信息
EventTypes.WsClose	WebSocket 连接关闭。	{ code, reason }

蓝牙模块 SDK
蓝牙适配器

最近更新时间：2024-10-08 18:37:11

基本介绍

蓝牙适配器（BlueToothAdapter），提供了搜索设备、连接设备等方法，进而实现和蓝牙设备之间的通信。
名词解释

名词	含义
serviceId	服务 id，蓝牙服务的 uuid，搜索设备时主要通过 `serviceId` 来过滤我们需要的设备。
deviceId	小程序 API 搜索出来的设备的标识，连接设备时主要通过 `deviceId` 来标识需要连接的设备。
explorerDeviceId	物联网开发平台侧定义的设备 ID，查询设备数据和上报设备数据时以设备 ID 作为设备标识。
DeviceAdapter 设备适配器	真正用来连接设备以及跟设备进行通信的模块，每一个设备连接对应一个设备适配器实例，设备适配器会在连接设备后实例化，并在设备断开连接后销毁。根据不同的 `serviceId` 来区别不同类型设备的适配器构造函数。

初始化

安装依赖

```
import { BlueToothAdapter } from 'qcloud-iotexplorer-bluetooth-adapter';
```

获取蓝牙适配器实例

1. 接口定义

```
new BlueToothAdapter(options: Object);
```

2. 输入参数

参数名称	参数描述	类型	必填
deviceAdapter	设备适配器	array	否
ignoreAnonymousDevices	忽略 `name === '未知设备'` 的蓝牙设备，默认值：true	boolean	否
devMode	开发模式	boolean | function	否

DeviceAdapter 参数说明
参考 设备适配器(DeviceAdapter) 说明文档。

3. 返回结果

参数名称	参数描述	类型
–	蓝牙适配器实例。	object

4. 示例代码

```
import { BlueToothAdapter } from 'qcloud-iotexplorer-bluetooth-adapter';
import { LLSyncDeviceAdapter } from 'qcloud-iotexplorer-bluetooth-adapter-llsync';
import appDevSdk from './appDevSdk';

LLSyncDeviceAdapter.injectOptions({
```

```
  // AppDevSdk实例，文档参考[小程序SDK]
  appDevSdk,
});

export const blueToothAdapter = new BlueToothAdapter({
  deviceAdapters: [
    LLSyncDeviceAdapter,
  ],
});
```

API 参考文档

添加设备适配器

调用本接口将其构造函数添加到蓝牙适配器中，可添加一个或多个设备适配器。

1. 接口定义

```
blueToothAdapter.addAdapter(deviceAdapter);
```

2. 输出参数

参数名	参数描述	类型	必填
deviceAdapter	要添加的设备适配器的构造函数	Function | Array	是

3. 示例代码

```
import { LLSyncDeviceAdapter } from 'qcloud-iotexplorer-bluetooth-adapter-llsync';
import {
  BleComboEspDeviceAdapter,
  BleComboLLSyncDeviceAdapter,
} from 'qcloud-iotexplorer-appdev-plugin-wificonf-blecombo';

LLSyncDeviceAdapter.injectOptions({ appDevSdk });

// 接受一个DeviceAdapter
blueToothAdapter.addAdapter(LLSyncDeviceAdapter);

// 接受多个DeviceAdapter
blueToothAdapter.addAdapter([
  BleComboEspDeviceAdapter,
  BleComboLLSyncDeviceAdapter,
]);
```

初始化蓝牙模块

包括初始化蓝牙模块、打通小程序间蓝牙通信、注册全局回调等。
本接口可重复调用，可在每次使用蓝牙模块前调用。

1. 接口定义

```
blueToothAdapter.init();
```

2. 返回结果

返回一个带缓存的 Promise，初始化成功后 resolve。若初始化未完成或已初始化成功，则多次调用后返回同一个 Promise。若初始化失败，则该缓存的 Promise 在 reject 之后会被释放，再次调用则将重新初始化。

3. 示例代码

```
blueToothAdapter.init().then(() => {
  // 调用蓝牙模块能力
});
```

开始搜索蓝牙设备

开始搜寻附近的蓝牙外围设备。将搜索到的所有蓝牙设备依次调用设备适配器的过滤函数。`DeviceAdapter.deviceFilter`，如果过滤函数返回蓝牙设备信息，将触发 `onSearch` 回调函数。
接口底层将会调用 wx.startBluetoothDevicesDiscovery，比较耗费系统资源，务必在不需要搜索（如已搜索到设备、离开搜索页面）之后调用 停止搜索蓝牙设备 。

1. 接口定义

```
blueToothAdapter.startSearch(options: Object);
```

2. 输入参数

参数名称	参数描述	类型	必填
onSearch	当搜索设备结果更新后调用，返回搜索到的设备列表。	function	是
onError	当搜索过程中发生错误后调用，触发后设备搜索将会中止。	function	是
serviceId	使用指定 serviceId 的设备适配器，默认会使用已添加的全部 DeviceAdapter。	string	否
serviceIds	参数描述同 serviceId。	string[]	否
ignoreServiceIds	通过 serviceId 忽略使用某些设备适配器。	string[]	否
ignoreDeviceIds	需要过滤掉的 deviceId 列表（例如刚添加完的设备），搜索结果中将不会出现这些设备。	string[]	否
timeout	超过指定时长没有搜索到设备，将会触发超时错误。默认值：20000；单位：ms。	number	否

3. 返回结果

返回一个 Promise。

4. 示例代码

```
blueToothAdapter.startSearch({
  onSearch: (devices) => {
    console.log('搜索到设备: ', devices);
    if (devices.length > 0) {
      blueToothAdapter.stopBleSearch();
    }
  },
  onError: (err) => {
    console.log('搜索设备出错', err);
  }
});
```

搜索单个蓝牙设备

开始搜索蓝牙设备，在找到第一个满足条件的设备后停止搜索。

1. 接口定义

```
blueToothAdapter.searchDevice(options: Object);
```

2. 输入参数

参数名称	参数描述	类型	必填
deviceId	蓝牙设备 deviceId，参数同 微信蓝牙接口 回调的 deviceId。	string	是
deviceName	腾讯云物联网开发平台的 deviceName。	string	否
serviceId	使用指定 serviceId 的设备适配器，默认会使用已添加的全部 DeviceAdapter。	string	否
serviceIds	参数描述同 serviceId。	string[]	否
ignoreDeviceIds	可选，需要过滤掉的 deviceId 列表（例如刚添加完的设备），搜索结果中将不会出现这些设备。	string[]	否

3. 返回结果

返回一个 Promise，结果为搜索到第一个满足条件的设备。

4. 示例代码

```
blueToothAdapter.searchDevice({ serviceId: 'xxx' }); // 搜索指定serviceId的设备
```

停止搜索蓝牙设备

接口定义

```
blueToothAdapter.stopSearch();
```

连接蓝牙设备

传入设备信息连接蓝牙设备，连接成功后实例化蓝牙设备对应的设备适配器并返回。

1. 接口定义

```
blueToothAdapter.connectDevice(deviceInfo, options);
```

2. 输入参数

参数名称	参数描述	类型	必填
deviceInfo	设备信息，可将 开始搜索蓝牙设备 或 搜索单个蓝牙设备 返回的蓝牙设备信息传入。	object	是
options	其他配置项。	object	否

deviceInfo 参数说明

参数名称	参数描述	类型	必填
deviceId	蓝牙设备 id，同微信蓝牙接口回调的 deviceId。	string	否
name	蓝牙设备名称，同微信蓝牙接口回调的 name。	string	否
productId	腾讯云物联网开发平台的产品 ID。	string	否
deviceName	腾讯云物联网开发平台的设备名称。	string	否
serviceId	蓝牙设备 serviceId。	string	否

options 参数说明

参数名称	参数描述	类型	必填
autoNotify	可选，默认为 true。	boolean	否

	指定为 true 时，在连接设备后，会自动去拉取服务列表，以及主服务下的特征值列表，并会自动订阅第一个 notifyId 或 indicateId 特征值的 notify。 若设备含有多个服务或多个 notify 特征值，请传 false，并自行通过 getBLEDeviceServices、getBLEDeviceCharacteristics、notifyBLECharacteristicValueChange 等方法获取及订阅特征值。		

3. 返回结果

返回一个 `Promise<DeviceAdapter>`，参数如下表：

参数名称	参数描述	类型	必填
-	设备适配器实例。	object	是

4. 示例代码

```
const deviceAdapter = await blueToothAdapter.connectDevice(device);
```

获取设备适配器实例

获取设备适配器实例，调用 `blublueToothAdapter.connectDevice` 之后会实例化设备适配器。

1. 接口定义

```
blueToothAdapter.getDeviceAdapter(options: Object);
```

2. 参数说明

参数名称	参数描述	类型	必填
deviceId	蓝牙设备 id，同微信蓝牙接口回调的 deviceId，优先级高于 explorerDeviceId。	string	否
explorerDeviceId	腾讯云物联网开发平台的 deviceId。	string	否

3. 返回结果

返回一个 Promise，参数如下：

参数名称	参数描述	类型	必填
-	设备适配器实例。	object | undefined	否

获取所有搜索到的蓝牙设备

底层调用 wx.getBluetoothDevices 获取在蓝牙模块生效期间所有搜索到的蓝牙设备，默认会过滤掉 `name === '未知设备'` 的蓝牙设备。

1. 接口定义

```
blueToothAdapter.getBluetoothDevices();
```

2. 返回结果

返回搜索到的蓝牙设备列表，参数同 wx.getBluetoothDevices 。

蓝牙适配器事件

监听事件

监听 蓝牙适配器事件 。

1. 接口定义

```
blueToothAdapter.on(type, listener);
```

2. 输入参数

参数名称	参数描述	类型	必填
type	事件名称。	string	是
listener	事件触发时的回调函数。	function	是

3. 示例代码

```
blueToothAdapter.on('adapterStateChange', ({ available, discovering }) => {
  console.warn('event adapterStateChange emit', { available, discovering });
});
```

取消监听事件

取消监听蓝牙适配器事件。

1. 接口定义

```
blueToothAdapter.off(type, listener);
```

2. 输入参数

参数名称	参数描述	类型	必填
type	要取消监听的事件。	string	是
listener	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。	function	否

3. 示例代码

```
const onAdapterStateChange = blueToothAdapter.on('adapterStateChange', ({ available, discovering }) => {
  console.warn('event adapterStateChange emit', { available, discovering });
});

blueToothAdapter.off('adapterStateChange', onAdapterStateChange);
```

事件列表

adapterStateChange 事件：当蓝牙适配器状态变化时触发回调，返回当前状态。事件回调函数参数如下：

参数名称	参数描述	类型
available	蓝牙适配器是否可用。	boolean
discovering	蓝牙适配器是否处于搜索状态。	boolean

设备适配器

最近更新时间：2024-10-08 18:37:11

基本介绍

设备适配器（DeviceAdapter）提供了操作蓝牙设备的基本方法，是真正用来连接设备以及跟设备进行通信的模块。
每个蓝牙协议的设备，都对应一个设备适配器实例，设备适配器会在连接设备（调用 `bluetoothAdapter.connectDevice()` ）之后实例化，并在设备断开连接后销毁。根据不同的 `serviceId` 来区别不同类型设备的适配器。
开发者可根据各蓝牙设备协议，自行实现设备适配器。

自定义设备适配器

自定义设备适配器类需要继承 `DeviceAdapter` ，并补充以下实现。
实现内容

- **serviceId**：自定义设备适配器类需要设置该静态属性，代表该设备的主服务 ID。
- **deviceFilter**：自定义设备适配器类需要实现该静态方法，在搜索蓝牙设备时会将每个搜索到的设备信息传入该函数。如果判断是本产品的设备，则需在除入参 deviceInfo 之外返回设备唯一标识 deviceName 及 serviceId，否则返回空。
- **handleBLEMessage**：自定义设备适配器类需要实现该方法，用于处理收到 `onBLECharacteristicValueChange` 回调后的协议解析。
  返回值中如果返回 `reportData` ，则会将该部分数据上报到云端（注意需与产品定义物模型匹配），其他字段则会透传到 `message` 事件的 payload 中。

示例代码

```
import { DeviceAdapter } from 'qcloud-iotexplorer-bluetooth-adapter';

class DemoDeviceAdapter extends DeviceAdapter {
  static serviceId = '0000FFF0-0000-1000-8000-00805F9B34CC';

  static deviceFilter(deviceInfo) {
    if (deviceInfo.advertisServiceUUIDs) {
      const matchedServiceId = deviceInfo.advertisServiceUUIDs.find(id => id ===
DemoDeviceAdapter.serviceId);
      if (matchedServiceId && deviceInfo.advertisData) {
        try {
          const macArr = deviceInfo.advertisData.slice(2);
          const mac = macArr.join(':');
          return {
            ...deviceInfo,
            deviceName: mac,
            serviceId: matchedServiceId,
          };
        } catch (err) {
          console.error('parse mac error', err);
        }
      }
    }
  }

  handleBLEMessage(hex) {
    return {
      type: 'unknown',
      data: hex,
    };
  }
}

const blueToothAdapter = new BlueToothAdapter({
  deviceAdapters: [
    DemoDeviceAdapter,
  ],
```

```
});
```

获取设备适配器实例

设备适配器会在连接设备之后实例化，通过蓝牙适配器连接设备后，返回对应的设备适配器。
示例代码

```
const deviceAdapter = blueToothAdapter.connectDevice(deviceInfo, options);
```

API 参考文档

设备适配器属性

属性名	属性描述	类型
explorerDeviceId	只读，设备的 explorerDeviceId。	string
isConnected	只读，当前是否已连接设备。	boolean
deviceId	只读，设备的 deviceId。	string
serviceId	只读，设备的主服务 ID，与构造函数上的静态属性 DeviceAdapter.serviceId 一致。	string
originName	只读，设备的原始名称，即小程序接口搜索出来时的 name 字段。	string

获取设备服务列表

获取蓝牙低功耗设备所有服务 (service)

1. 接口定义

```
deviceAdapter.getBLEDeviceServices();
```

2. 返回结果

返回一个Promise，参数同 wx.getBLEDeviceServices 的 services。

获取服务的特征值列表

获取蓝牙低功耗设备某个服务中所有特征 (characteristic)

1. 接口定义

```
deviceAdapter.getBLEDeviceCharacteristics({ serviceId: string });
```

2. 输入参数

参数名称	参数描述	类型	必填
serviceId	指定要获取特征值列表的serviceId。默认值：主服务 ID。	string	否

3. 返回结果

返回一个Promise，参数同 wx.getBLEDeviceCharacteristics 的 characteristics。

读取指定特征值的二进制数据

读取蓝牙低功耗设备特征值的二进制数据。注意：必须设备的特征支持 read 才可以成功调用。

1. 接口定义

```
deviceAdapter.readBLECharacteristicValue({
  serviceId: string,
  characteristicId: string
});
```

2. 输入参数

参数名称	参数描述	类型	必填
serviceId	指定某个蓝牙服务。默认值：主服务 ID。	string	否
characteristicId	需要读取的特征值 ID。默认值：取主服务下的第一个 read 特征值。	string	否

3. 返回结果

返回一个Promise，参数同 wx.readBLECharacteristicValue 的返回结果。

启用设备特征值变化时的 notify 功能

启用蓝牙低功耗设备特征值变化时的 notify 功能，订阅特征。

1. 接口定义

```
deviceAdapter.notifyBLECharacteristicValueChange({
    characteristicId?: string,
    serviceId?: string,
    state?: boolean
});
```

2. 输入参数

参数名称	参数描述	类型	必填
serviceId	需要订阅的服务 ID，默认会取主服务 ID。	string	否
characteristicId	需要订阅的特征值 ID，默认会取主服务下的第一个 notify 或 indicate 特征值。	string	否
state	是否启用 notify，默认为 true。	boolean	否

获取设备的信号强度

获取蓝牙低功耗设备的信号强度 (Received Signal Strength Indication, RSSI)。

1. 接口定义

```
deviceAdapter.getBLEDeviceRSSI();
```

2. 返回结果

返回一个Promise，参数同 wx.getBLEDeviceRSSI 的 RSSI。

协商设置蓝牙最大传输单元

协商设置蓝牙低功耗的最大传输单元 (Maximum Transmission Unit, MTU)，本接口仅支持在 Android 系统下调用，iOS 因系统限制不支持。

1. 接口定义

```
deviceAdapter.setBLEMTU({ mtu: number });
```

2. 输入参数

参数名称	参数描述	类型	必填
mtu	最大传输单元。设置范围为 (22,512) 区间内，单位为 bytes。	number	是

3. 返回结果

参数名称	参数描述	类型	必填
mtu	最终协商的 MTU 值。如果协商失败则无此参数。	number	否

写入二进制数据到指定特征值中

向蓝牙低功耗设备特征值中写入二进制数据。注意：必须设备的特征支持 write 才可以成功调用。

1. 接口定义

```
deviceAdapter.write(hexString: string, options: Object);
```

2. 输入参数

参数名称	参数描述	类型	必填
hexString	需要写给蓝牙设备的十六进制字符串。	string	是
options.serviceId	需要写入的服务 ID，默认值：取主服务 ID。	string	否
options.writeId	需要写入的特征值 ID。默认值：取主服务下的第一个 writeId。	string	否

3. 返回结果

返回一个Promise，参数同 wx.writeBLECharacteristicValue 的返回结果。

断开设备连接

底层调用 `wx.closeBLEConnection` 断开与蓝牙低功耗设备的连接，触发设备适配器的 `disconnect` 事件。
接口定义

```
deviceAdapter.disconnectDevice();
```

设备适配器事件

监听事件

1. 接口定义

```
deviceAdapter.on(type, listener);
```

2. 输入参数

参数名称	参数描述	类型	必填
type	事件名称。	string	是
listener	事件触发时的回调函数。	function	是

取消监听事件

1. 接口定义

```
deviceAdapter.off(type, listener);
```

2. 输入参数

参数名称	参数描述	类型	必填
type	要取消监听的事件。	string	是
listener	要取消监听的事件的回调函数，不传则清除该事件的所有回调函数。	function	否

事件列表

- **connect 事件**：设备连接后触发。
- **disconnect 事件**：设备断开后触发。

- **message 事件：当收到 `onBLECharacteristicValueChange` 回调，并经过 `handleBLEMessage` 处理后触发。**

参数名	参数描述	类型
timestamp	收到设备消息的时间戳，单位毫秒。	number
dataReported	收到设备的消息是否已上报云端。	boolean
（其他）	handleBLEMessage 函数返回的其他参数将会透传到 message 事件中。	any

- **bLEConnectionStateChange 事件：当 `onBleConnectionStateChange` 触发时触发，若 `connected` 为 true，则接下来会触发 `connect` 事件，否则会触发 `disconnect` 事件。**

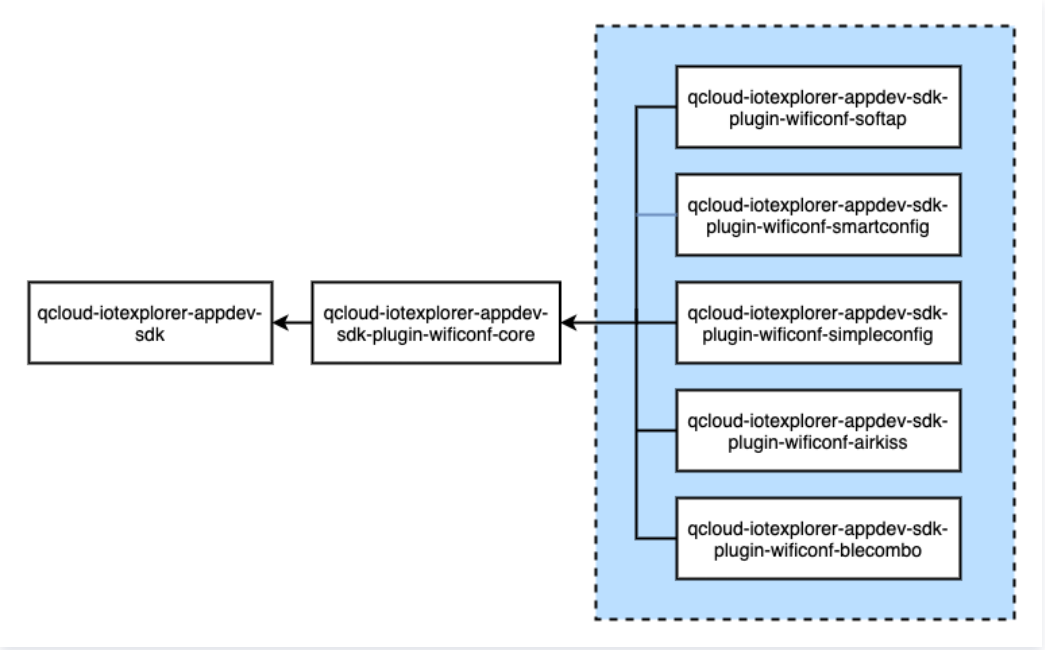
参数名	参数描述	类型
connected	设备是否连接。	boolean

设备配网 SDK

最近更新时间：2024-10-08 18:37:11

配网概述

SDK 目前支持 SoftAP、SmartConfig、simpleConfig、AirKiss、BLE-Combo 这五种方式进行设备配网。
这五种配网方式的 SDK 都是以**插件**的方式按需引入的，为了方便大家理解，下图可以看出这五个 SDK 的依赖关系。



基本使用

通过四步可以运行配网插件，以 `SoftAp` 配网为例，其余配网方式步骤相同，后面关于配网步骤和参数说明会有具体阐述。

1. 安装依赖

```
npm install qcloud-iotexplorer-appdev-plugin-wificonf-softap
```

2. 注册插件

调用插件上 `install` 方法来注册插件，调用时需要传入实例化的小程序 SDK。
插件注册时的名称分别为：**wifiConfSoftAp、wifiConfSmartConfig、wifiConfSimpleConfig、wifiConfBleCombo、wifiConfAirKiss**，之后可通过插件名称来获取插件实例。

```
const SoftApPlug = require('qcloud-iotexplorer-appdev-plugin-wificonf-softap');
SoftApPlug.install(appDevSdk);
```

3. 生成配网 Token

调用应用端 API 生成 Wi-Fi 设备配网 Token 来获取 Wi-Fi 设备配网 Token。

```
sdk.requestApi('AppCreateDeviceBindToken')
  .then((data) => {
    const bindDeviceToken = data.Token;
    console.log('wifi设备配网Token=', bindDeviceToken);
  });
```

4. 使用配网插件

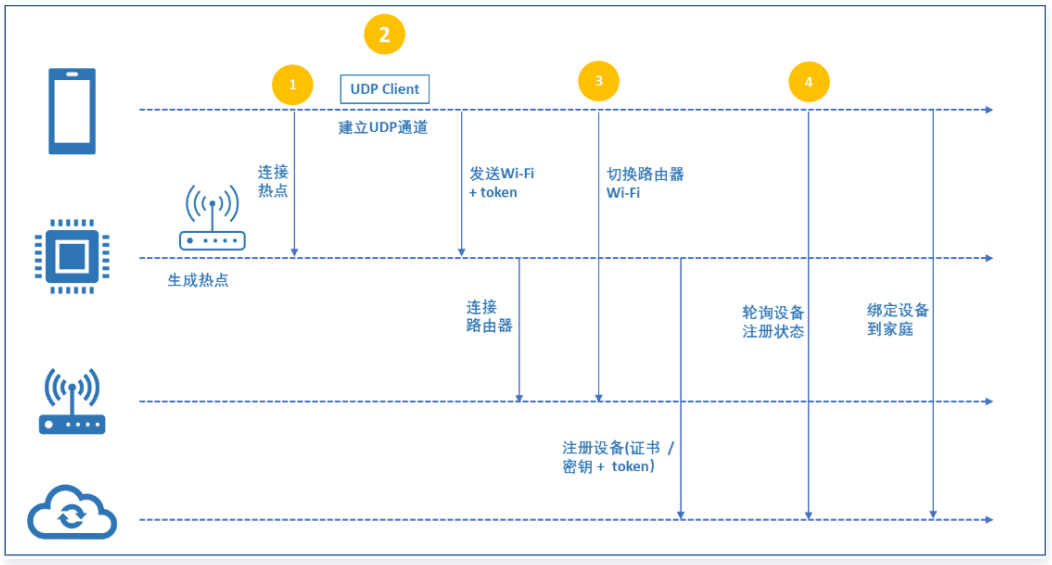
从 appDevSdk 实例的 `plugins` 中获取已注册的插件实例，调用配网插件实例的 `start` 方法开始配网流程。

示例代码

```
sdk.plugins['wifiConfSoftAp'].start({
  wifiConfToken: bindDeviceToken,
  targetWifiInfo: wifiInfo,
  onProgress,
  onComplete,
  onError,
  autoRetry: true, // 自动处理故障流程
});
```

SoftAP 配网

关于 softAP 方式配网的流程，请参见 softAP 配网开发 。各端交互的流程如下：



腾讯连连中 `SoftAP` 配网页面如下，供开发者参考。

SoftAP 配网参数

参数名	参数描述	类型	必填
wifiConfToken	Wi-Fi 设备配网 Token，从后台接口 生成 Wi-Fi 设备配网 Token 获取。	string	是
targetWifiInfo	目标 Wi-Fi 信息，需要设备去连接的 Wi-Fi 的信息。	WifiInfo	是
softAPInfo	设备热点信息，如果传该配置，则首先会调用 wx.connectWifi 去连接设备热点；如果不传，则需要自行引导用户去连接设备热点。	WifiInfo	否
onProgress	配网过程执行到每个步骤时触发的回调，回调函数入参如下： • code：步骤代码，详见 配网步骤 小节 • msg：步骤描述，自行从 `WifiConfStepDesp` 用 code 取描述内容。 • detail：步骤详情，根据每个步骤不同而不同。	function	是
onError	配网失败时触发，回调函数入参如下： • code：错误代码，详见 错误码 小节。 • msg：错误描述，自行从 `WifiConfErrorMsg` 拿 code 取描述内容。 • detail：错误详情。	function	是
onComplete	配网完成后触发，回调函数入参如下： deviceInfo：设备信息。	function	是
familyId	家庭 ID，默认为：'default'，即用户默认家庭 ID。	string	否
roomId	房间 ID，默认为：''，即用户默认房间 ID。	string	否
udpAddress	连接上设备热点后，小程序发起 UDP 通信的地址，默认为：'192.168.4.1'，一般无需更改。	string	否

udpPort	连接上设备热点后，小程序发起 UDP 通信的端口，默认为：8266，一般无需更改。	number	否
stepInterval	配网过程中，每一步中间等待的间隔，单位毫秒，默认为：1000，一般无需更改。	number	否
autoRetry	配网失败之后是否要启动自动错误处理后直接重试，自动错误处理 章节，默认为：false。	boolean	否

WifiInfo 数据结构

属性名	属性描述	类型	必填
SSID	Wi-Fi 的 SSID	string	是
password	Wi-Fi 的 密码	string	是

示例代码

```
import { constants as WifiConfConstants } from 'qcloud-iotexplorer-appdev-plugin-wificonf-core';

const {
  // 步骤code
  WifiConfStepCode,
  // 步骤code的中文描述
  WifiConfStepDesp,
  // 错误的中文描述
  WifiConfErrorMsg,
} = WifiConfConstants;

/**
 * softap配网
 */
function SoftApConfigure({
  token,
  wifiInfo = {
    SSID: '',
    password: '',
  },
  familyId = 'default',
  roomId,
  /* 更新配网步骤 */
  onStepChange,
  /* 更新配网状态(success/error) */
  onStatusChange,
}) {
  const onProgress = ({ code, detail }) => {
    const msg = WifiConfStepDesp[code];
    console.log('配网步骤更新(onProgress)', code, msg, detail);
    switch (code) {
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.SOFTAP_GET_DEVICE_SIGNATURE_SUCCESS:
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
```

```
  };

  const onComplete = (deviceInfo) => {
    console.log('配网完成(onComplete):', deviceInfo);
    onStatusChange({
      status: 'success',
      productId: deviceInfo.productId,
      deviceName: deviceInfo.deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    const msg = WifiConfErrorMsg[code];
    console.log('配网错误(onError)', code, msg, detail);
    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfSoftAp'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError,
  });
}

module.exports = SoftApConfigure;
```

配网步骤v1.0（已废弃，不建议使用）

步骤	描述
WifiConfStepCode.WIFI_CONF_START	开始配网。
WifiConfStepCode.PROTOCOL_START	配网协议开始。
WifiConfStepCode.CONNECT_SOFTAP_START	开始连接设备热点。
WifiConfStepCode.CONNECT_SOFTAP_SUCCESS	连接设备热点成功。
WifiConfStepCode.CREATE_UDP_CONNECTION_START	开始与设备建立 UDP 连接。
WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS	与设备建立 UDP 连接成功。
WifiConfStepCode.SOFTAP_SEND_TARGET_WIFIINFO_START	开始发送目标 Wi-Fi 信息。
WifiConfStepCode.SOFTAP_SEND_TARGET_WIFIINFO_SUCCESS	发送目标 Wi-Fi 信息成功。 detail: { response }，收到设备的具体响应。
WifiConfStepCode.SOFTAP_GET_DEVICE_SIGNATURE_START	开始获取设备签名。
WifiConfStepCode.SOFTAP_GET_DEVICE_SIGNATURE_SUCCESS	获取设备签名成功。 detail: { signature }
WifiConfStepCode.SOFTAP_RECONNECT_TARGET_WIFI_START	开始手机连接目标 Wi-Fi。
WifiConfStepCode.SOFTAP_RECONNECT_TARGET_WIFI_SU	手机连接目标 Wi-Fi 成功。

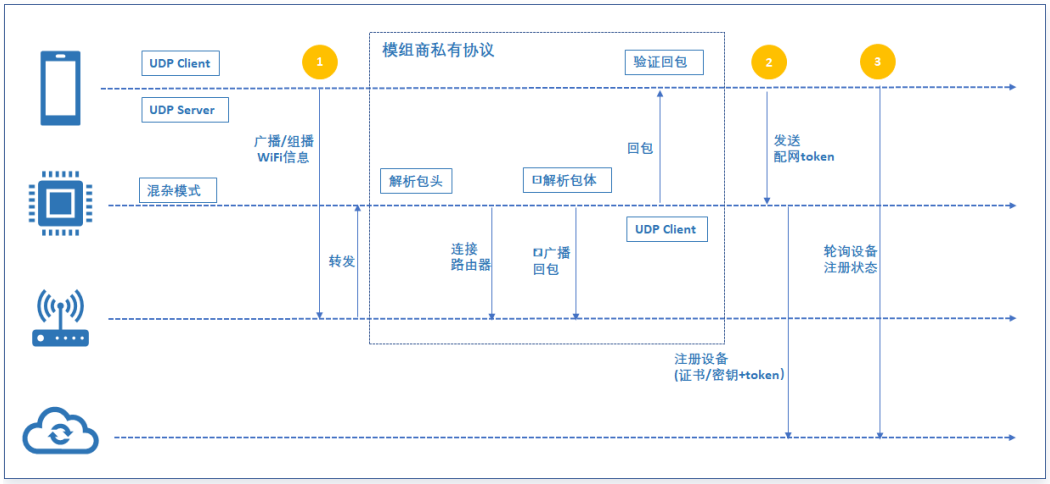
CCESS	
WifiConfStepCode.PROTOCOL_SUCCESS	配网协议成功。
WifiConfStepCode.BUSINESS_START	业务流程开始。
WifiConfStepCode.BUSINESS_ADD_DEVICE_START	开始添加设备。
WifiConfStepCode.BUSINESS_ADD_DEVICE_SUCCESS	添加设备成功。
WifiConfStepCode.BUSINESS_SUCCESS	业务流程成功。 detail: { productId, deviceName}，请求参数。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

配网步骤v2.0

步骤	描述
WifiConfStepCode.WIFI_CONF_START	开始配网。
WifiConfStepCode.PROTOCOL_START	配网协议开始。
WifiConfStepCode.CONNECT_SOFTAP_START	开始连接设备热点。
WifiConfStepCode.CONNECT_SOFTAP_SUCCESS	连接设备热点成功。
WifiConfStepCode.CREATE_UDP_CONNECTION_START	开始与设备建立 UDP 连接。
WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS	与设备建立 UDP 连接成功。
WifiConfStepCode.SOFTAP_SEND_TARGET_WIFIINFO_START	开始发送目标 Wi-Fi 信息。
WifiConfStepCode.SOFTAP_SEND_TARGET_WIFIINFO_SUCCESS	发送目标 Wi-Fi 信息成功。 detail: { response }，收到设备的具体响应。
WifiConfStepCode.SOFTAP_RECONNECT_TARGET_WIFI_START	开始手机连接目标 Wi-Fi。
WifiConfStepCode.SOFTAP_RECONNECT_TARGET_WIFI_SUCCESS	手机连接目标 Wi-Fi 成功。
WifiConfStepCode.PROTOCOL_SUCCESS	配网协议成功。
WifiConfStepCode.BUSINESS_START	业务流程开始。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_START	开始查询配网TOKEN状态。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS	查询配网TOKEN状态成功。
WifiConfStepCode.BUSINESS_ADD_DEVICE_START	开始添加设备。
WifiConfStepCode.BUSINESS_ADD_DEVICE_SUCCESS	添加设备成功。
WifiConfStepCode.BUSINESS_SUCCESS	业务流程成功。 detail: { productId, deviceName}，请求参数。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

SmartConfig 配网

关于 SmartConfig 方式配网的流程，请参见 SmartConfig 配网开发 。一键配网的配网流程图文版本如下：



腾讯连连中 `SmartConfig` 配网页面如下，供开发者参考。

SmartConfig 配网参数

参数名	参数描述	类型	必填
wifiConfToken	Wi-Fi 设备配网 Token，从后台接口 生成 Wi-Fi 设备配网 Token 获取。	string	是
targetWifiInfo	目标 Wi-Fi 信息，需要设备去连接的 Wi-Fi 的信息。	WifiInfo	是
onProgress	配网过程执行到每个步骤时触发的回调，回调函数入参如下： • code：步骤代码，详见 配网步骤 小节。 • msg：步骤描述，自行从 `WifiConfStepDesp` 用code取描述内容。 • detail：步骤详情，根据每个步骤不同而不同。	function	是

onError	配网失败时触发，回调函数入参如下： • code：错误代码，详见 错误码 小节。 • msg：错误描述，自行从 `WifiConfErrorMsg` 拿code取描述内容。 • detail：错误详情。	function	是
onComplete	配网完成后触发，回调函数入参如下： deviceInfo：设备信息。	function	是
familyId	家庭 ID，默认为：'default'，即用户默认家庭 ID。	'default' | string	否
roomId	房间 ID，默认为：''，即用户默认房间 ID。	'' | string	否
udpPort	小程序和设备连上同一个局域网之后，小程序发起 UDP 通信的端口，默认为：8266，一般无需更改。	number	否
stepInterval	配网过程中，每一步中间等待的间隔，单位毫秒，默认为：1000，一般无需更改。	number	否
autoRetry	配网失败之后是否要启动自动错误处理后直接重试，自动错误处理 章节，默认为：false。	boolean	否

WifiInfo 数据结构

属性名	属性描述	类型	必填
SSID	Wi-Fi 的 SSID。	string	是
BSSID	Wi-Fi 的 BSSID。	string	是
password	Wi-Fi 的 密码。	string	是

示例代码

```
import { constants as WifiConfConstants } from 'qcloud-iotexplorer-appdev-plugin-wificonf-core';

const {
  // 步骤code
  WifiConfStepCode,
  // 步骤code的中文描述
  WifiConfStepDesp,
  // 错误的中文描述
  WifiConfErrorMsg,
} = WifiConfConstants;

/**
 * smartconfig一键配网
 */
function SmartConfigConfigure({
  token,
  wifiInfo = {
    SSID: '',
    password: '',
    BSSID: '',
  },
  familyId = 'default',
  roomId,
  /* 更新配网步骤 */
  onStepChange,
  /* 更新配网状态 */
  onStatusChange,
}) {
  const onProgress = ({ code, detail }) => {
    const msg = WifiConfStepDesp[code];
```

```
    console.log('配网步骤更新(onProgress)', code, msg, detail);
    switch (code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = (deviceInfo) => {
    console.log('配网完成(onComplete):', deviceInfo);
    onStatusChange({
      status: 'success',
      productId: deviceInfo.productId,
      deviceName: deviceInfo.deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    const msg = WifiConfErrorMsg[code];
    console.log('配网错误(onError)', code, msg, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins['wifiConfSmartConfig'].start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError,
  });
}

module.exports = SmartConfigConfigure;
```

配网步骤

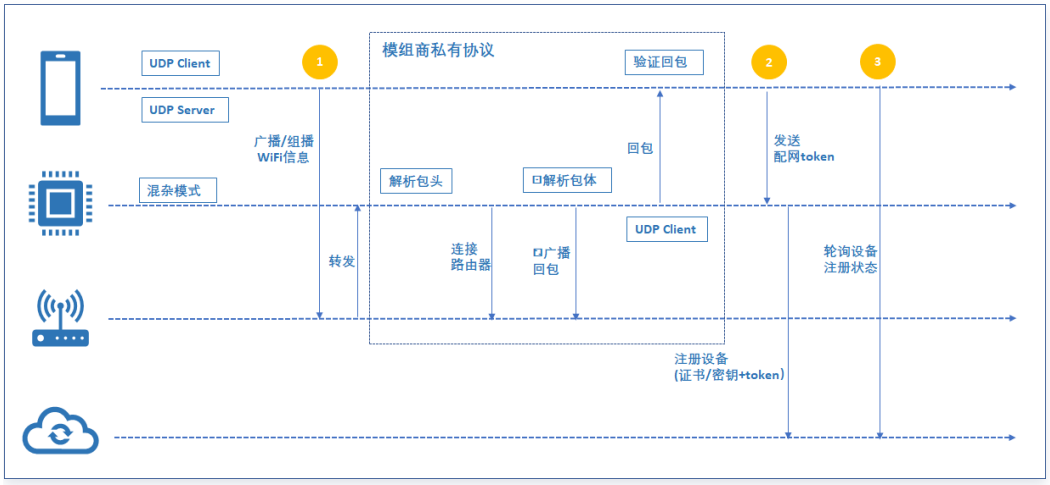
`sdk.plugins['wifiConfSmartConfig'].start()` 配网过程中，每执行完一个步骤就会触发一次 `onProgress` 回调，入参为：`{ code, detail }` 形式。

步骤	描述
WifiConfStepCode.WIFI_CONF_START	开始配网。
WifiConfStepCode.PROTOCOL_START	配网协议开始。
WifiConfStepCode.PROTOCOL_DETAIL	配网协议的细节，详细日志。
WifiConfStepCode.PROTOCOL_SUCCESS	配网协议成功，获取到设备地址 `detail: { data: { address } }`，

	收到设备局域网地址，用于给设备发送信息。
WifiConfStepCode.BUSINESS_START	业务流程开始。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_START	开始查询配网 TOKEN 状态。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS	查询配网 TOKEN 状态成功。
WifiConfStepCode.BUSINESS_ADD_DEVICE_START	开始添加设备。
WifiConfStepCode.BUSINESS_ADD_DEVICE_SUCCESS	添加设备成功。
WifiConfStepCode.BUSINESS_SUCCESS	业务流程成功。 detail: { productId, deviceName}，请求参数。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

SimpleConfig 配网

关于 simpleConfig 方式配网的流程，请参见 simpleConfig 配网开发 。一键配网的配网流程图文版本如下：



腾讯连连中 `SimpleConfig` 配网页面如下，供开发者参考。

SimpleConfig 配网参数

参数名	参数描述	类型	必填
wifiConfToken	Wi-Fi 设备配网 Token，从后台接口 生成 Wi-Fi 设备配网 Token 获取。	string	是
targetWifiInfo	目标 Wi-Fi 信息，需要设备去连接的 Wi-Fi 的信息。	WifiInfo	是
onProgress	配网过程执行到每个步骤时触发的回调，回调函数入参如下： • code：步骤代码，详见 配网步骤 小节。 • msg：步骤描述，自行从 `WifiConfStepDesp` 用code取描述内容。 • detail：步骤详情，根据每个步骤不同而不同。	function	是
onError	配网失败时触发，回调函数入参如下： • code：错误代码，详见 错误码 小节。 • msg：错误描述，自行从 `WifiConfErrorMsg` 拿code取描述内容。 • detail：错误详情。	function	是
onComplete	配网完成后触发，回调函数入参如下： deviceInfo：设备信息。	function	是
familyId	家庭 ID，默认为：'default'，即用户默认家庭 ID。	string	否
roomId	房间 ID，默认为：''，即用户默认房间 ID。	string	否
udpPort	小程序和设备连上同一个局域网之后，小程序发起 UDP 通信的端口，默认为：8266，一般无需更改。	number	否
stepInterval	配网过程中，每一步中间等待的间隔，单位毫秒，默认为：1000，一般无需更改。	number	否
autoRetry	配网失败之后是否要启动自动错误处理后直接重试，自动错误处理 章节，默认为：false。	boolean	否

WifiInfo 数据结构

属性名	属性描述	类型	必填
SSID	Wi-Fi 的 SSID。	string	是
password	Wi-Fi 的 密码。	string	是

示例代码

```
import { constants as WifiConfConstants } from 'qcloud-iotexplorer-appdev-plugin-wificonf-core';

const {
  // 步骤code
  WifiConfStepCode,
  // 步骤code的中文描述
  WifiConfStepDesp,
  // 错误的中文描述
  WifiConfErrorMsg,
} = WifiConfConstants;

/**
 * simpleConfig一键配网
 */
function SimpleConfigConfigure({
  token,
  wifiInfo = {
    SSID: '',
    password: '',
  },
  familyId = 'default',
  roomId,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = ({ code, detail }) => {
    const msg = WifiConfStepDesp[code];
    console.log('配网步骤更新(onProgress)', code, msg, detail);

    switch (code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = (deviceInfo) => {
    console.log('配网完成(onComplete):', deviceInfo);

    onStatusChange({
      status: 'success',
      productId: deviceInfo.productId,
```

```
      deviceName: deviceInfo.deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    const msg = WifiConfErrorMsg[code];
    console.log('配网错误(onError)', code, msg, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins.wifiConfSimpleConfig.start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError,
  });
}

module.exports = SimpleConfigConfigure;
```

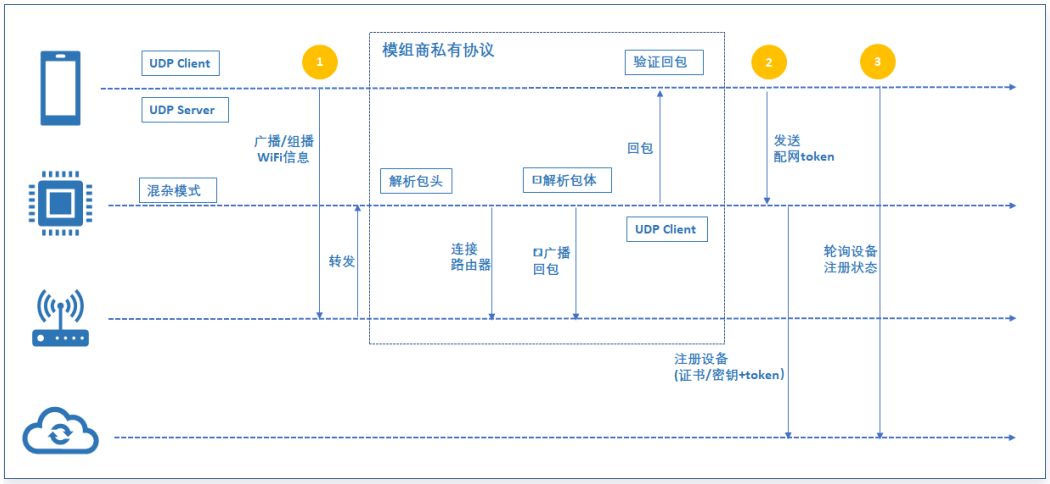
配网步骤

`sdk.plugins['wifiConfSimpleConfig'].start()` 配网过程中，每执行完一个步骤就会触发一次 `onProgress` 回调，入参为：`{ code, detail }` 形式。

步骤	描述
WifiConfStepCode.WIFI_CONF_START	开始配网。
WifiConfStepCode.PROTOCOL_START	配网协议开始。
WifiConfStepCode.PROTOCOL_DETAIL	配网协议的细节，详细日志。
WifiConfStepCode.PROTOCOL_SUCCESS	配网协议成功，获取到设备地址 `detail: { data: { address } }`，收到设备局域网地址，用于给设备发送信息。
WifiConfStepCode.BUSINESS_START	业务流程开始。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_START	开始查询配网 TOKEN 状态。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS	查询配网 TOKEN 状态成功。
WifiConfStepCode.BUSINESS_ADD_DEVICE_START	开始添加设备。
WifiConfStepCode.BUSINESS_ADD_DEVICE_SUCCESS	添加设备成功。
WifiConfStepCode.BUSINESS_SUCCESS	业务流程成功。 detail: { productId, deviceName}，请求参数。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

AirKiss 配网

关于 AirKiss 方式配网的流程，请参见 AirKiss 配网开发 。一键配网的配网流程图文版本如下：



腾讯连连中 `AirKiss` 配网页面如下，供开发者参考。

AirKiss 配网参数

参数名	参数描述	类型	必填
wifiConfToken	Wi-Fi 设备配网 Token，从后台接口 生成 Wi-Fi 设备配网 Token 获取。	string	是
targetWifiInfo	目标 Wi-Fi 信息，需要设备去连接的 Wi-Fi 的信息。	WifiInfo	是
onProgress	配网过程执行到每个步骤时触发的回调，回调函数入参如下： • code：步骤代码，详见 配网步骤 小节。 • msg：步骤描述，自行从 `WifiConfStepDesp` 用code取描述内容。 • detail：步骤详情，根据每个步骤不同而不同。	function	是
onError	配网失败时触发，回调函数入参如下：	function	是

	• code：错误代码，详见 错误码 小节。 • msg：错误描述，自行从 `WifiConfErrorMsg` 拿code取描述内容。 • detail：错误详情。		
onComplete	配网完成后触发，回调函数入参如下： deviceInfo：设备信息。	function	是
familyId	家庭 ID，默认为：'default'，即用户默认家庭 ID。	string	否
roomId	房间 ID，默认为：''，即用户默认房间 ID。	string	否
udpPort	小程序和设备连上同一个局域网之后，小程序发起 UDP 通信的端口，默认为：8266，一般无需更改。	number	否
stepInterval	配网过程中，每一步中间等待的间隔，单位毫秒，默认为：1000，一般无需更改。	number	否
autoRetry	配网失败之后是否要启动自动错误处理后直接重试，自动错误处理 章节，默认为：false。	boolean	否

WifiInfo 数据结构

属性名	属性描述	类型	必填
SSID	Wi-Fi 的 SSID。	string	是
password	Wi-Fi 的 密码。	string	是

示例代码

```
import { constants as WifiConfConstants } from 'qcloud-iotexplorer-appdev-plugin-wificonf-core';

const {
  // 步骤code
  WifiConfStepCode,
  // 步骤code的中文描述
  WifiConfStepDesp,
  // 错误的中文描述
  WifiConfErrorMsg,
} = WifiConfConstants;

/**
 * AirKiss一键配网
 */
function AirKissConfigure({
  token,
  wifiInfo = {
    SSID: '',
    password: '',
  },
  familyId = 'default',
  roomId,
  reporter,
  onStepChange,
  onStatusChange,
}) {
  const onProgress = ({ code, detail }) => {
    const msg = WifiConfStepDesp[code];
    console.log('配网步骤更新(onProgress)', code, msg, detail);

    switch (code) {
      case WifiConfStepCode.PROTOCOL_SUCCESS:
        onStepChange(1);
        break;
```

```
      case WifiConfStepCode.CREATE_UDP_CONNECTION_SUCCESS:
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS:
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS:
        onStepChange(4);
        break;
    }
  };

  const onComplete = (deviceInfo) => {
    console.log('配网完成(onComplete):', deviceInfo);

    onStatusChange({
      status: 'success',
      productId: deviceInfo.productId,
      deviceName: deviceInfo.deviceName,
    });
  };

  const onError = async ({ code, detail }) => {
    const msg = WifiConfErrorMsg[code];
    console.log('配网错误(onError)', code, msg, detail);

    onStatusChange({ status: 'error' });
  };

  sdk.plugins.wifiConfAirKiss.start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    autoRetry: true, // 自动处理故障流程
    familyId,
    roomId,
    onProgress,
    onComplete,
    onError,
  });
}
```

配网步骤

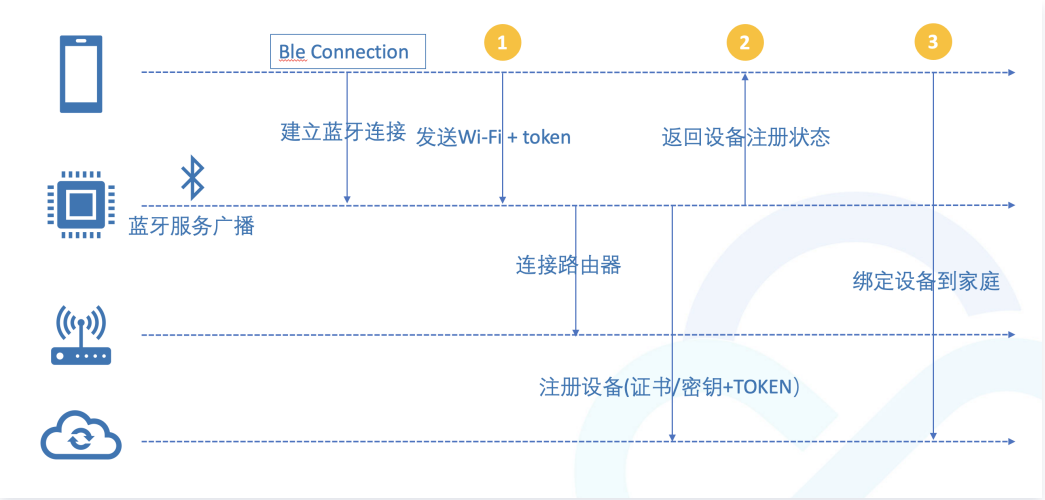
`sdk.plugins['wifiConfAirKiss'].start()` 配网过程中，每执行完一个步骤就会触发一次 `onProgress` 回调，入参为：`{ code, detail }` 形式。

步骤	描述
WifiConfStepCode.WIFI_CONF_START	开始配网。
WifiConfStepCode.PROTOCOL_START	配网协议开始。
WifiConfStepCode.PROTOCOL_DETAIL	配网协议的细节，详细日志。
WifiConfStepCode.PROTOCOL_SUCCESS	配网协议成功，获取到设备地址 `detail: { data: { address } }`，收到设备局域网地址，用于给设备发送信息。
WifiConfStepCode.BUSINESS_START	业务流程开始。
WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_START	开始查询配网 TOKEN 状态。

WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS	查询配网 TOKEN 状态成功。
WifiConfStepCode.BUSINESS_ADD_DEVICE_START	开始添加设备。
WifiConfStepCode.BUSINESS_ADD_DEVICE_SUCCESS	添加设备成功。
WifiConfStepCode.BUSINESS_SUCCESS	业务流程成功。 detail: { productId, deviceName}，请求参数。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

BleCombo 蓝牙辅助配网

关于 蓝牙辅助方式配网的流程，请参见 蓝牙辅助配网开发 。一键配网的配网流程图文版本如下：



腾讯连连中 `AirKiss` 配网页面如下，供开发者参考。

获取设备适配器

在配网之前，我们需要先发现设备，然后与设备建立蓝牙连接，并获得一个 DeviceAdapter 实例，用于蓝牙设备的通信。
这个过程可以通过蓝牙适配器 (BlueToothAdapter) 完成。流程如下：

1. 创建一个蓝牙适配器

蓝牙适配器 (bluetoothAdapter) 主要用于搜索设备、连接设备。示例代码如下：

```
import { BlueToothAdapter } from 'qcloud-iotexplorer-bluetooth-adapter';
import {
  BleComboEspDeviceAdapter,
  BleComboLLSyncDeviceAdapter,
} from 'qcloud-iotexplorer-appdev-plugin-wificonf-blecombo';

export const bluetoothAdapter = new BlueToothAdapter({
  deviceAdapters: [
    BleComboEspDeviceAdapter,
    BleComboLLSyncDeviceAdapter,
  ],
});
```

在实例化 blueToothAdapter时，我们需要传入想要支持设备的 DeviceAdapter。目前插件内置了两种 `DeviceAdapter`：

- BleComboEspDeviceAdapter：支持通过 BluFi 协议 进行蓝牙辅助配网。
- BleComboLLSyncDeviceAdapter：支持通过 LLSync 协议 进行蓝牙辅助配网。

2. 获取蓝牙设备列表

通过 `bluetoothAdapter.startSearch` 方法，我们可以发现设备，获得设备列表。

```
await bluetoothAdapter.startSearch({
  ignoreDeviceIds,
  serviceIds,
  ignoreServiceIds,
  onError: (error) => {
    console.log('搜索设备出错', error);
    bluetoothAdapter.stopSearch();
  },
  onSearch: (devices) => {
    if (devices.length > 0) {
      // 可以在页面上展示搜索到的设备
      console.log('搜索到设备', devices);
    }
  },
  timeout: 1.4 * 15 * 1000,
});
```

在 onSearch 回调函数中，我们可以获得搜寻到的设备列表，这时可以将设备列表展示到页面上，供用户选择要连接的设备。

3. 连接设备

用户点击要连接的蓝牙设备后，可以调用 `bluetoothAdapter.connectDevice()` 方法进行连接，并调用 `bluetoothAdapter.stopSearch()` 结束搜索蓝牙设备。
连接成功后会返回一个 deviceAdapter，可以用来向连接的设备发送 Wi-Fi、token 等数据。

```
try {
  // device参数是上一步获取的devices中的某一项
  const deviceAdapter = await bluetoothAdapter.connectDevice(device);

  if (!deviceAdapter) {
    throw {
      code: 'CONNECT_ERROR',
    };
  }
} catch (err) {
  console.error('连接设备出错');
}
```

在上面三步完成之后，我们已经通过蓝牙连接到了设备，并获得了可以与设备通信的 deviceAdapter，接下来就可以正式进行配网了。

BleCombo 配网参数

参数名	参数描述	类型	必填
wifiConfToken	Wi-Fi 设备配网 Token，从后台接口 生成 Wi-Fi 设备配网 Token 获取。	string	是
targetWifiInfo	目标 Wi-Fi 信息，需要设备去连接的 Wi-Fi 的信息。	WifiInfo	是
onProgress	配网过程执行到每个步骤时触发的回调，回调函数入参如下： • code：步骤代码，详见 配网步骤 小节。 • msg：步骤描述，自行从 `WifiConfStepDesp` 用code取描述内容。 • detail：步骤详情，根据每个步骤不同而不同。	function	是
onError	配网失败时触发，回调函数入参如下： • code：错误代码，详见 错误码 小节。 • msg：错误描述，自行从 `WifiConfErrorMsg` 拿code取描述内容。 • detail：错误详情。	function	是
onComplete	配网完成后触发，回调函数入参如下： deviceInfo：设备信息。	function	是
deviceAdapter	用于和设备进行蓝牙通信的设备适配器实例，连接蓝牙之后获得，详见 DeviceAdapter 。	DeviceAdapter	是
familyId	家庭 ID，默认为：'default'，即用户默认家庭 ID。	string	否
roomId	房间 ID，默认为：''，即用户默认房间 ID。	string	否
bleComboProto	使用的蓝牙配网协议，目前支持 ESP 官方和 LLsync 两种协议。	'BLE_COMBO_ESP' / 'BLE_COMBO_LLSYNC'	否

WifiInfo 数据结构

属性名	属性描述	类型	必填
SSID	Wi-Fi 的 SSID。	string	是
password	Wi-Fi 的 密码。	string	是

示例代码

```
function BleComboConfigure({
  // 用于设备连接云端的token
  token,
  wifiInfo,
  familyId = 'default',
  roomId,
  // 连接设备之后获得
  deviceAdapter,
}) {
  /**
   * 这里可以进行一些UI进度更新操作
   */
  const onStepChange = (progress) => {
    console.log(progress);
  };

  // 这里是配网进行过程中的回调函数
  const onProgress = (data) => {
```

```
    console.info(data.code, data.detail);
    switch (data.code) {
      case WifiConfStepCode.PROTOCOL_START: // 开始配网
        onStepChange(1);
        break;
      case WifiConfStepCode.PROTOCOL_SUCCESS: // 设备联网成功，设备可以访问互联网
        onStepChange(2);
        break;
      case WifiConfStepCode.BUSINESS_QUERY_TOKEN_STATE_SUCCESS: // 发送token到设备成功，设备开始连接云端
        onStepChange(3);
        break;
      case WifiConfStepCode.WIFI_CONF_SUCCESS: // 配网成功
        onStepChange(4);
        break;
    }
  };

  const onComplete = ({ productId, deviceName }) => {
    // 配网成功后，可以拿到设备的 productId 和 设备名称
    console.log('配网成功', productId, deviceName);
  };

  const onError = async ({ code, detail }) => {
    console.error('配网出错', code, detail);
  };

  // 开始执行配网逻辑
  sdk.plugins.wifiConfBleCombo.start({
    wifiConfToken: token,
    targetWifiInfo: wifiInfo,
    deviceAdapter,
    familyId,
    roomId,
    onProgress,
    onError,
    onComplete,
  });
}
```

配网步骤

`sdk.plugins['wifiConfBleCombo'].start(bleComboOpts)` 配网过程中，每执行完一个步骤就会触发一次 `onProgress` 回调，入参为：`{ code, detail }` 形式。

步骤	描述
WifiConfStepCode.PROTOCOL_START	开始配网。
WifiConfStepCode.PROTOCOL_SUCCESS	设备联网成功，设备可以访问互联网。
WifiConfStepCode.BLE_SEND_TOKEN_START	开始发送 token 到设备，用于连接云端。
WifiConfStepCode.BLE_SEND_TOKEN_SUCCESS	发送 token 到设备成功，设备开始连接云端。
WifiConfStepCode.WIFI_CONF_SUCCESS	配网成功。

蓝牙辅助配网错误码

在 `onError` 回调函数中，我们可以拿到配网失败的错误码。

CODE	描述

PROTOCOL_FAIL	设备连接失败。
BLE_SEND_TOKEN_ERROR	发送 token 到设备失败。
WIFI_CONF_FAIL	配网失败。

配网错误码

错误码	描述
UDP_NOT_RESPONSED	超时未收到设备响应。
UDP_CLOSED	设备连接中断。
UDP_ERROR	配网过程中触发 udp.onError 事件。 `detail: { errMsg }`，错误信息。
UDP_SEND_MSG_FAIL	与设备 UDP 通信时，发送消息失败。
CONNECT_SOFTAP_FAIL	手机连接设备热点失败。 `detail: { errMsg }`，错误信息。
BUSINESS_WIFI_RECONNECT_FAIL	手机连接 Wi-Fi 路由器失败。 `detail: { errMsg }`，错误信息。
BUSINESS_DEVICE_ERROR	收到设备响应的错误。 `detail: { errMsg }`，错误信息。
BUSINESS_INVALID_RESPONSE	收到非法的设备响应。 `detail: { response }`，具体的设备端响应。
BUSINESS_DEVICE_CONNECT_MQTT_FAIL	设备连接 MQTT 服务失败。
BUSINESS_DEVICE_CONNECT_WIFI_FAIL	设备连接目标 Wi-Fi 失败。
BUSINESS_QUERY_BIND_TOKEN_TIMEOUT	设备连接云端超时。
WIFI_CONF_FAIL	配网流程失败。 `detail: { errMsg }`，错误信息。
PROTOCOL_FAIL	配网协议失败。 `detail: { errMsg }`，错误信息。
PROTOCOL_TIMEOUT	配网协议超时。
PROTOCOL_INVALID_RESPONSE	配网协议收到非法响应。 `detail: { errMsg }`，错误信息。

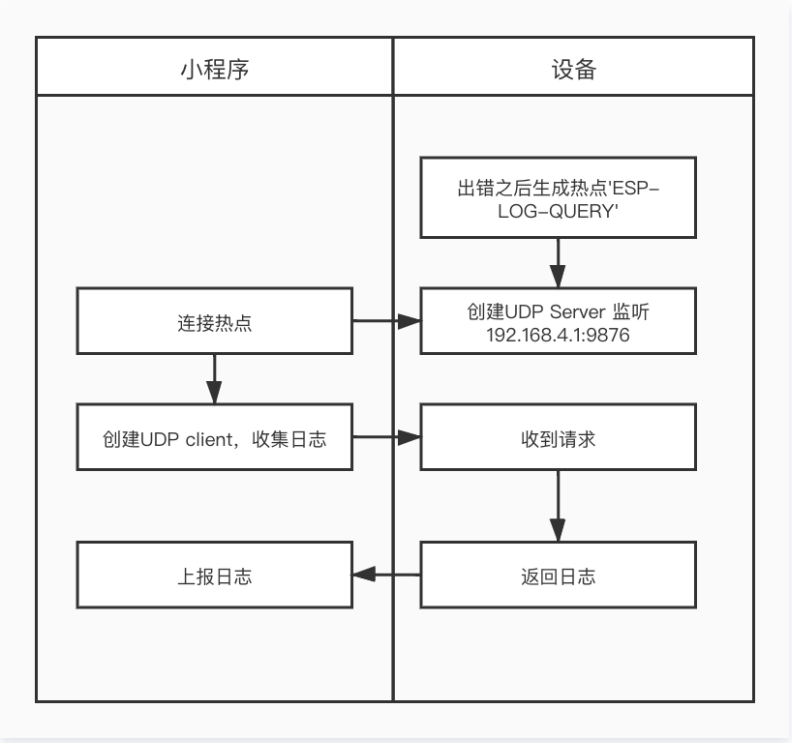
自动错误处理

配网流程中会出现一些错误，在成功率不断优化的实践当中，总结一些可以自动处理的错误类型，处理成功后自动重试，全程用户无感知。

错误码	错误处理方式
PROTOCOL_TIMEOUT	超时未收到设备响应，其原因可能是中途网络被切走，导致设备和手机无法通信造成超时；自动处理方式：检查当前网络是不是目标网络，否则切到目标网络，重新配网。
UDP_ERROR	UDP 通道发生错误，可能的原因其一如上（中途网络被切走，导致设备和手机无法通信造成超时），其二是 Wi-Fi 切换之后，底层 UDP 还未切换，会发包失败；自动处理方式：延迟2s之后重新配网。
UDP_SEND_MSG_FAIL	同上两种处理方式。

模组日志收集

当发生错误的时候，只看 `onProgress` 或者 `onError` 里面打印出来的信息并不能准确定位到问题，需要结合设备端日志查看，我们制定了跟设备端的日志交互协议，原理如下：



示例代码

```
const { collectModuleLog } =  require('qcloud-iotexplorer-appdev-plugin-wificonf-core').utils;

collectModuleLog({
   // 用于上报的对象
   reporter: console,
   sdk: sdk,
})
```

日志上报打印详情

日志收集的过程以及结果会通过 `reporter.info(code, detail)` 的回调打印出来，可以通过这个方法来收集信息以及日志的上报。

步骤	描述
WifiConfStepCode.MODULE_REPORT_START	开始配网日志收集
WifiConfStepCode.MODULE_REPORT_CONNECT_WIFI_START	日志收集开始连接设备热点
WifiConfStepCode.MODULE_REPORT_CONNECT_WIFI_SUCCESS	日志收集连接设备热点成功
WifiConfStepCode.MODULE_REPORT_COMMUNICATE_AP_START	开始收集设备端日志
WifiConfStepCode.MODULE_REPORT_COMMUNICATE_AP_SUCCESS	收集成功。`detail: { moudleDetail}`，日志详情。

错误码

错误码	描述
MODULE_REPORT_COMMUNICATE_AP_ERROR	和设备端通信失败。
MODULE_REPORT_TIMEOUT	收集日志超时

音乐服务 SDK

最近更新时间：2024-10-08 18:37:11

初始化SDK

获取实例

1. 自主品牌小程序中使用

```
npm install qcloud-iotexplorer-tme-sdk

import { TMESdkForMiniProgram } from 'qcloud-iotexplorer-tme-sdk';
// 注：appDevSdk为初始化后的实例
const tmeSdk = new TMESdkForMiniProgram(appDevSdk);
```

2. 自定义 H5 中使用

```
const tmeSdk = await window.h5PanelSdk.getTMESdk();
```

接口统一返回值

接口调用的返回值统一为 `Promise<TMEResponse>` 类型。

```
interface TMEResponse {
  error_code: number;
  error_msg: string;
  data?: any;
}
```

- 调用成功：返回一个 resolved 的 Promise，其值为 TMEResponse 类型，error_code=0，data 为返回结果。
- 调用失败：返回一个 rejected 的 Promise，包含错误码（error_code）及提示信息（error_msg）。

属性名	描述	类型
error_code	错误码。	number
error_msg	错误信息。	string
data	响应数据。	object

错误码列表

错误码	说明
200001	参数错误。
200002	系统繁忙，如幂等接口并发调用等，通常由于用户并发操作造成。
200003	认证信息过期或错误，请重新登录。
200004	设备未激活。
200005	当前 sp 暂未支持此接口。
200006	系统错误，如内部调用超时等，由于服务内部异常导致。
200200	可直充剩余次数为0。
400000	登录授权失败。
400001	设备端超时无响应。
400002	调用 SDK 参数错误。

登录授权

用户设备登录授权

跳转酷狗音乐小程序授权，当再次返回 H5 或小程序时，Promise 状态改变。

- **接口定义**

```
tmeSdk.login(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 Id。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

用户设备登出

原 token 将登出。

- **接口定义**

```
tmeSdk.logout(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

校验设备授权

- **接口定义**

```
tmeSdk.checkDeviceAuth(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

获取用户信息

- **接口定义**

```
tmeSdk.getUserInfo(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>`，其中 data 如下：

属性名	描述	类型
userid	用户 ID。	string
nick_name	用户昵称。	string
img	用户头像。	string
is_vip	是否 vip 。 • 0：否。 • 1：是。	enum: `0` `1`
vip_end_time	vip 有效期终止时间。	string
car_vip_end_time	车机会员有效期终止时间。	string
svip_end_time	豪 V 有效期终止时间。	string

播控部分

接口描述

调用播控 SDK，会下发物模型属性 +control_seq，需要设备上报相同的 control_seq。

- 若在超时范围内收到上报，视为下发播控成功，返回 resolved 状态的 `Promise<TMEResponse>` 。
- 若超时未收到上报，返回 rejected 状态的 `Promise<TMEResponse>` 。

超时设置可以通过 `tmeSdk.config.timeout` 来配置，默认值为10000，单位：毫秒（ms）。

通用播控接口

- 接口定义

```
tmeSdk.controlKugouDeviceData(deviceData, deviceId: string) => Promise<TMEResponse>
```

- 参数说明

参数名	参数描述	类型	必填
deviceData	设备物模型数据。	object	是
deviceId	设备 ID。	string	是

- 返回值

返回一个 `Promise<TMEResponse>` 。

播放

- 接口定义

```
tmeSdk.play(deviceId: string) => Promise<TMEResponse>
```

- 参数说明

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- 返回值

返回一个 `Promise<TMEResponse>` 。

暂停

- **接口定义**

```
tmeSdk.pause(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

上一首

- **接口定义**

```
tmeSdk.preSong(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

下一首

- **接口定义**

```
tmeSdk.nextSong(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

设置播放模式

- **接口定义**

```
tmeSdk.setPlayMode(playMode: number, deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
playMode	播放模式： • 0：顺序播放 。 • 1：单曲循环 。 • 2：随机播放。	enum: `0` `1` `2`	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 `Promise<TMEResponse>` 。

设置音量

- **接口定义**

```
tmeSdk.setVolume(volume: number, deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
volume	音量：0-100之间。	number	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 `Promise<TMEResponse>` 。

设置播放进度

- **接口定义**

```
tmeSdk.setPlayPosition(playPosition: number, deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
playPosition	播放进度：单位:秒(s)。	number	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 `Promise<TMEResponse>` 。

设置播放质量

- **接口定义**

```
tmeSdk.setPlayQuality(recommendQuality: number, deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
recommendQuality	播放质量： • 0：标准 。 • 1：高清 。 • 2：无损。	enum: `0` `1` `2`	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 `Promise<TMEResponse>` 。

设置当前播放歌曲

- **接口定义**

```
tmeSdk.playSong(songId: string, songIndex: string, newQueueType: string, newQueueId: string | number,
deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	参数描述	类型	必填
songId	歌曲 ID。	string	是
songIndex	歌曲所在播放列表的位置，从0开始。	string	是
newQueueType	播放列表的类型：`playlist`、`newSongs`、`recommendDailty`。	string	是
newQueueId	播放列表 ID（当类型为“每日推荐”时，不存在 id，传undefined）。	string | number	是
deviceId	设备 ID。	string	是

播放列表目前支持三种类型：歌单（playlist）、新歌首发（newSongs）、每日推荐（recommendDaily）。

- **返回值**

返回一个 `Promise<TMEResponse>` 。

内容部分

拉取内容通用接口

请求酷狗 API 拉取内容通用接口。

- **接口定义**

```
tmeSdk.requestTMEApi(action: string, params, deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	描述	类型	必填
action	接口 action。	string	是
params	请求参数，无请求参数时，传{}。	object	是
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。

> **说明：**
> action、params 及 返回值 data 参考 音乐服务 。

获取设备当前播放歌曲

- **接口定义**

```
tmeSdk.getCurrentPlaySong(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` ，data 为歌曲信息。

获取设备当前播放列表

根据目前支持的播放类型（playType），拉取对应的歌单列表，并查出歌曲的详细信息。

- **接口定义**

```
tmeSdk.getCurrentPlayQueue(deviceId: string) => Promise<TMEResponse>
```

- **参数说明**

参数名	描述	类型	必填
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` 。
TMEResponse 中 data 如下：

属性名	描述	类型
playType	播放列表类型。	enum： `playlist` 、 `newSongs` 、 `recommendDaily` 。
queueId	当前播放列表 ID，根据 playType 对应 playlist_id、album_id、top_id。	string | number
total	列表中歌曲总数。	number
songs	歌曲数组，具体歌曲属性参考 TME 文档中 Song 属性。	Array[]

获取歌曲详细信息

- **接口定义**

```
tmeSdk.getSongDetail(songId: string, deviceId: string) => Promise<TMEResponse>
```

通过调用 requestTMEApi，请求歌曲播放链接与歌曲信息，返回歌曲的详细信息。

- **参数说明**

参数名	描述	类型	必填
songId	歌曲 ID。	string	是
deviceId	设备 ID。	string	是

- **返回值**

返回一个 `Promise<TMEResponse>` ，data 为歌曲信息。

获取歌单详细信息

- **接口定义**

```
tmeSdk.getPlaylistDetail(action: string, params, deviceId: string) => Promise<TMEResponse>
```

通过调用 requestTMEApi，请求歌单列表与歌曲信息，丰富列表中的歌曲信息，返回歌单列表。

- **参数说明**

参数名	描述	类型	必填
action	新歌首发(awesome_newsong)、每日推荐（awesome_everyday）、歌单歌曲（playlist_song）。	string	是

params	参考应用端 API > 音乐服务 中对应 API 的 KugouParams。	object	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 `Promise<TMEResponse>`，data 为歌单列表。

文件资源管理 SDK

最近更新时间：2022-03-04 18:12:54

使用指南

基本参数

属性名	类型	描述
sdk	object	依赖的 sdk，h5 依赖 qcloud-iotexplorer-h5-panel-sdk，小程序依赖 qcloud-iotexplorer-appdev-sdk。
request	Function	调用应用端 api。

获取实例

1. **自主品牌小程序使用**

```
const { FileSdkForMiniProgram } = require('qcloud-iotexplorer-fileresource-sdk');
const fileSdk = new FileSdkForMiniProgram(appDevSdk);
```

2. **自定义H5使用**

```
import { FileSdkForH5 } from 'qcloud-iotexplorer-fileresource-sdk';
let fileSdk = new FileSdkForH5(window.h5PanelSdk);
```

上传文件资源

- **接口定义**

```
const ResourceName = fileSdk.handleUpload(file: File, productId: string) => Promise<string>
```

- **参数说明**

参数名	参数描述	类型	必填
file	需要上传的文件资源。	File	是
productId	产品 ID。	string	是

- **返回值**

返回一个 Promise，输出参数如下。

参数名	参数描述	类型
ResourceName	资源名称。	string

下发资源到设备

- **接口定义**

```
fileSdk.controlDeviceResource(ResourceName: string, deviceId: string) => Promise<void>
```

- **参数说明**

参数名	参数描述	类型	必填
ResourceName	资源名称。	string	是

deviceId	设备 ID。	string	是

- **返回值**

  返回一个 Promise。

获取指定资源信息

- **接口定义**

```
fileSdk.getDeviceResource(ResourceName: string, deviceId: string)=> Promise
```

- **参数说明**

参数名	参数描述	类型	必填
ResourceName	资源名称。	string	是
deviceId	设备 ID。	string	是

- **返回值**

  返回一个 Promise，资源信息可通过 `promise.then()` 获取。

音视频服务 Video-sdk

最近更新时间：2024-04-25 18:16:51

介绍

通过腾讯云 IoT Video 小程序 P2P 服务，引入 IoT Video X-P2P 插件和 P2P-Player 插件，可实现摄像头和小程序直接打洞传输视频流；配合云端的 Server SDK，可实现小程序和小程序，小程序和 App 之间的数据共享。

准备工作

- 申请腾讯云 IoT Video P2P 服务，获取访问密钥（联调阶段可直接使用 demo 里的密钥，正式发布时请使用我们邮件提供给您的正式密钥）。
- 向腾讯云 IoT Video 团队申请使用 IoT Video X-P2P 插件 和 IoT Video P2P-Player 插件。
- 有使用 live-player 的权限，详见 官方文档。
- 如果使用 1v多 模式，需要将 flv 流的域名加到小程序的 `request 合法域名` 和 `tcp 合法域名` 配置中，详请参见 服务器域名配置官方文档。

微信版本限制

- 微信 8.0.10 以上版本
- 基础库 2.19.3 以上版本。

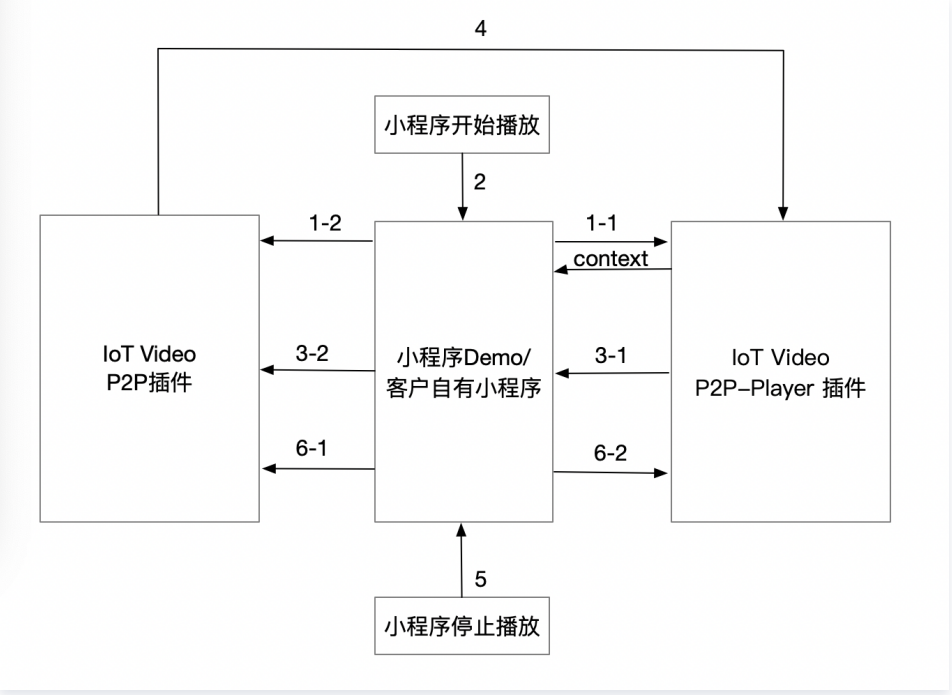
开发指引

开发文档

- IoT Video X-P2P 插件开发文档
- IoT Video P2P-Player 插件开发文档
- Demo 源码

> **说明：**
> 若**未购买**音视频激活码，请先联系商务进行购买或 提交工单 咨询。

原理介绍



- 0：初始化阶段。
  小程序在启动时就可以调用 IoT Video P2P 插件的 init 接口，初始化 p2p 模块，入参填写腾讯云 IoT Video 分配的 appKey 和 appKeySecret 等信息。

- 1：小程序用户选择某个摄像头。
  - 小程序引用 IoT Video P2P-Player，返回 `livePlayerContext` ， 可以直接对 `live-player` 的 context 进行操作。
  - 如果是1v1，小程序调用 IoT Video P2P 插件的 startP2PService 接口，开始建立 p2p 连接。
- 2：小程序利用 `livePlayerContext` 触发播放。
- 3：插件启动播放。
  - P2P-Player 插件抛出拉流事件 playerStartPull 给小程序应用。
  - 小程序应用调用 XP2P 插件的 startStream 接口，传入需要播放的摄像头 ID 和播放 URL，并设置消息接收回调和数据接收回调。
- 4：数据流会通过第3步设置的数据接收回调，传递给 P2P-Player 插件播放。
- 5：小程序用户停止播放。
- 6：插件终止播放。
  - 小程序调用 P2P 插件的 stopStream、stopServiceById 终止传输数据。
  - 小程序 Demo 或自有小程序操作 live-player 的 context，停止播放。
- 7：消耗。
  小程序退出时，调用 IoT Video P2P 插件的 destroy 接口，销毁 p2p 模块。

小程序 Demo

Demo 地址

- 源码

> **说明：**
> - 若**未购买**音视频激活码，请先联系商务进行购买或 提交工单 咨询。
> - 该体验版二维码对应 3.x.x 版本的 xp2p 插件。

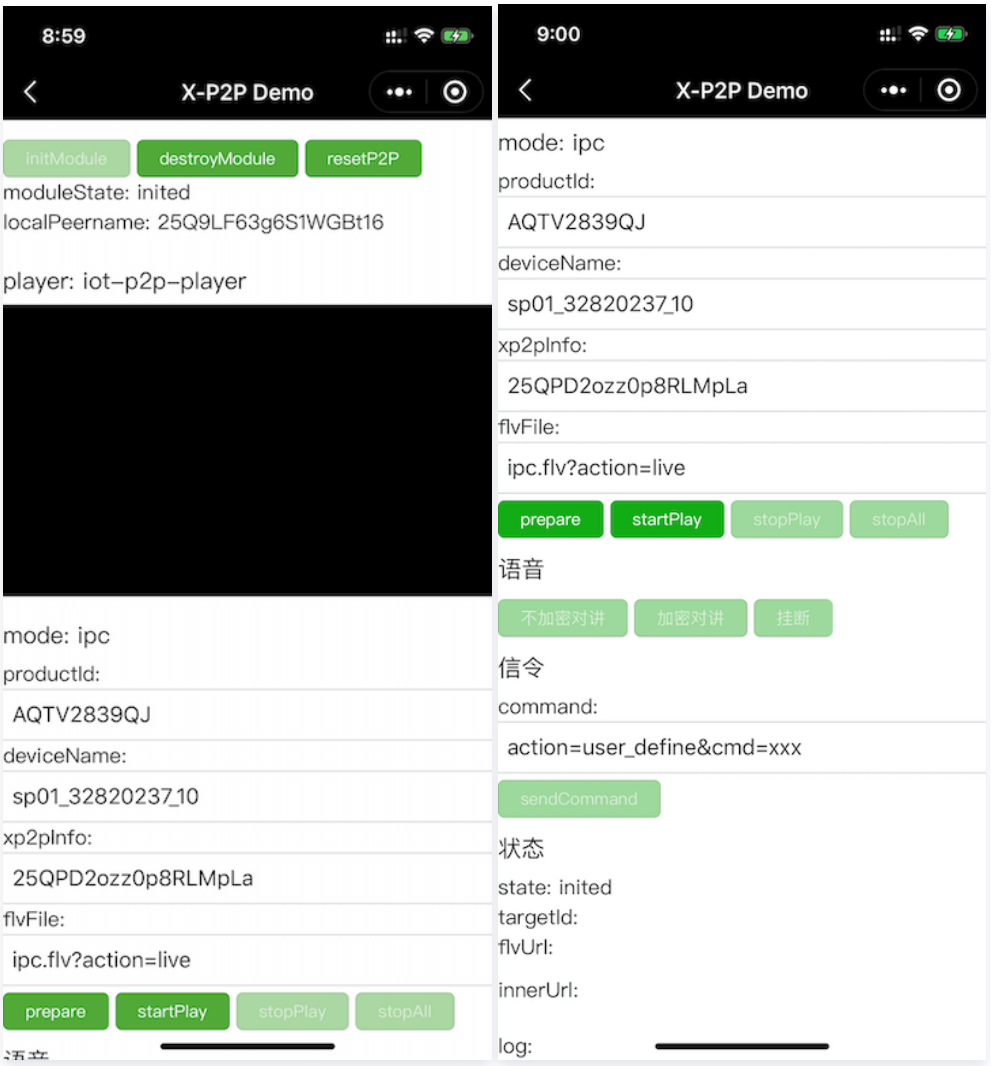
Demo 使用

> **注意：**
> Demo UI 交互可能更新，但主要流程不变。

1. 进入主页面。

   - "X-P2P Demo IPC" 演示1V1 P2P 直连摄像头场景。
   - "X-P2P Demo 1vN-xntp" 和 "X-P2P Demo 1vN-tcp" 演示 1V 多 P2P 场景。
   - “多播放器” 演示多播放器的调用。

2. 1V1 P2P 直连摄像头场景：



- 如果 x-p2p 插件还未初始化，先单击 **initModule** 初始化。
- 填写 "productId"、"deviceName"、"xp2pInfo"，xp2pInfo 即设备属性中的 "sys_p2p_info" 字段，可通过 获取设备属性数据的 API 获取。
- “flvFile”字段可设置播放清晰度，取值参考 信令交互文档 。
- 单击 **prepare** 和 **startPlay** 即可开始播放。
- 单击**不加密对讲**和**加密对讲**可演示小程序语音对讲，单击**挂断**停止对讲。
- 修改信令 command 的 "cmd=xxx" 可演示自定义信令。

自主品牌 App 开发
概述

最近更新时间：2024-12-26 18:07:33

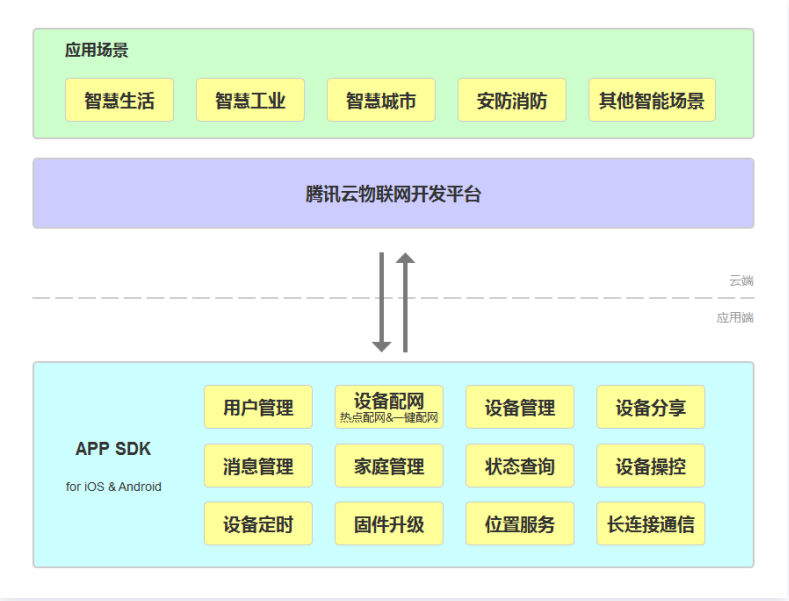
为进一步构建物联网开放生态，由腾讯云物联网平台打造的腾讯连连 App SDK，集成通用版 App 的多功能模块。设备厂商可通过 SDK 将设备接入腾讯云物联网平台进行设备管理，涵盖家用电器、运动健康、网络设备等众多设备。
目前，腾讯连连 App SDK 的核心模块包含设备配网的两种模式，分别是 SmartConfig 与 Soft AP；其他模块包括设备消息操作、账户系统、设备管理等。

SDK 下载

SDK 名称	**腾讯连连 App SDK**
版本号	v1.5.6
SDK 介绍	为开发者提供腾讯连连通用版 App 的开放应用能力，如设备配网、设备消息、设备管理等功能。赋能设备厂商快速接入腾讯云物联网平台，构建面向消费者的自主品牌应用 App。
服务提供方	深圳市腾讯计算机系统有限公司
合规使用说明	《腾讯连连 App SDK 合规使用指南》
个人信息处理规则	《腾讯连连 App SDK 个人信息保护规则》
更新日志	1、隐私合规优化：增加可选功能配置能力； 2、修复已知问题。
下载 SDK	• Android端 SDK： 前往下载 • iOS端 SDK： 前往下载 • 接入指引： SDK 接入指南

SDK 的依赖关系

在腾讯云物联网平台中，App SDK 扮演的角色如图所示。App 通过接入 App SDK 来实现与智能设备的配网，并通过物联网平台对智能设备进行管理。目前 App SDK 中与设备配网方式提供 SmartConfig 配网 和 SoftAP 配网 模式。



更新日志

V1.5.6

版本	开发语言	开发环境	系统平台	更新时间	更新日志

v1.5.6	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2023-12-26	1. 隐私合规优化：增加可选功能配置能力； 2. 修复已知问题。

v1.5.5

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.5.5	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2023-06-23	1. 修改账号登录页面。 2. 获取验证码和微信登录需同意隐私协议文档。 3. 补充相机等权限触发逻辑。

V1.5.4

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.5.4	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2022-01-18	1. 隐私政策支持查看第三方信息共享清单。 2. 增加部分个人敏感信息、系统授权弹窗。 3. 账号中心增加个人权限管理功能。 4. 可复制个人信息用户 id。

V1.3.0

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.3.0	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2021-01-15	1. 新增扫一扫产品落地详情页，支持控制台详情页配置。 2. 优化实时音视频信令逻辑，提升用户体验。 3. 修复 App H5 自定义控制面板的已知问题。 4. App 主要页面、按钮风格优化。 5. 设备绑定优化，为用户自动填写默认设备名称。

V1.2.1

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.2.1	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2020-12-10	修复 V1.2.0 版本的 SDK 配置，删除 i386 架构。

V1.2.0

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.2.0	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2020-12-02	提供接入实时音视频通话场景需求能力。

V1.1.0

版本	开发语言	开发环境	系统平台	更新时间	更新日志
v1.1.0	OC 语言/Java	Mac、Xcode/Android studio	iOS/Android	2020-10	1. 提供 SoftAP、SmartConfig 配网管理。 2. 实现设备管理、设备分享、设备定时。 3. 支持家庭管理、消息管理。

					4. 实现长链接通信获取设备状态。

接入指南

最近更新时间：2024-10-21 14:47:11

App SDK 模块说明

iOS

子模块	实现相关功能
QCDeviceCenter	配网模块。
QCAPISets	设备控制、消息相关、家庭管理、账户管理等 API。
QCFoundation	工具类。

Android

子模块	实现相关功能
link	配网模块。
auth	设备控制、消息相关、家庭管理、账户管理等 API。
utils	工具类。
log	日志模块。

SDK 接入详情

接入前 API 参数对照表

基础参数对照表：

参数名称	参数说明
phoneNumber	手机号。
countryCode	国际区号，如中国大陆区号为86。
email	邮箱地址。
familyId	家庭 ID。
familyName	家庭名称。
familyAddress	家庭地址。
Role	1是所有者， 0是普通成员。
roomId	房间 ID。
roomName	房间名。
ProductId	设备产品 ID。
Avatar	用户信息中头像链接。
signature	使用绑定设备 API 时传入，设备签名。
DeviceId	设备 ID。

设备控制面板列表参数对照表：

参数名称	参数说明
ID	设备可控属性。
name	设备可控属性名，例如：“电源开关”、“颜色”。
big	设备可控属性，面板按钮是否是大按钮。

type	设备可控属性，面板按钮类型，例如：btn-big、btn-col-1。
value	属性值。
familyAddress	家庭地址。
LastUpdate	最后一次更新时间戳。

SDK 使用举例

```
#iOS举例：设备配网接入
#1.创建配网对象QCSmartConfig或者QCSoftAP（视配网方式决定），注：SDK 内不持有配网对象，需使用者自己持有
self.sc = [[QCSmartConfig alloc] initWithSSID:name PWD:password BSSID:bssid];
self.sc.delegate = self;

#2.遵循TIoTCoreAddDeviceDelegate协议，设置代理，并接入代理方法：
- (void)onResult:(QCResult *)result{
        if (result.code == 0) {// 配网成功
        }
        else  {// 配网失败
        }
}

#3.开始配网流程
[self.sc startAddDevice];
```

更多功能

账户系统、家庭相关操作、设备控制等详细操作，请参考 SDK Demo 工程 iOS 版本 LinkSDKDemo 或 Android 版本 sdkdemo 。

详细接口对照表

物联网应用开发文档	对应文档地址
应用端 API	应用端 API 文档
用户管理	用户管理文档
配网管理	配网管理文档
设备管理	设备管理文档
设备分享	设备分享文档
家庭管理	家庭管理文档
设备定时	设备定时文档
消息管理	消息管理文档
长连接通信	长连接通信文档
数据结构	数据结构文档

若接入过程中有其他问题，请参考 基于 SDK 常见问题 。

创建引导

最近更新时间：2024-11-29 11:07:42

准备工作

- 注册 腾讯云账号 并完成 实名验证 。
- 进入 物联网开发平台控制台 > 应用开发获取目标应用的 App Key 和 App Secret。

> ⚠ 注意：
> 签名算法务必在服务端实现，腾讯连连 App 开源版的使用方式仅为演示，登录鉴权请参考 自建服务 。

iOS 安装环境

- 安装 Xcode 开发工具 。
- 集成 SDK 方式。
  执行如下命令采用 Cocoapods 方式集成。

```
pod TIoTLinkKit
```

- 手动集成。
  将 LinkCore 目录拖入到工程中。

Android 安装环境

- 安装 Android Studio 开发工具
- 集成 SDK 方式
- 依赖 maven 远程构建

```
implementation 'com.tencent.iot.explorer:explorer-link-android:1.0.0'
```

- 依赖本地 SDK 源码 构建
  修改应用模块的 build.gradle，使应用模块依赖 SDK 源码，示例如下：

```
implementation project(path: ':sdk')
```

使用说明

1. 导入配置，初始化 SDK。

iOS

参考 `Source/LinkSDKDemo/Supporting Files/AppDelegate.m` 文件，并配置物联网开发平台申请的 App Key 和 App Secret。

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary
*)launchOptions {
    /*
     * 此处仅供参考，需自建服务接入物联网平台服务，以免 App Secret 泄露
     * 自建服务可参考此处 https://cloud.tencent.com/document/product/1081/67504#e4359606-62fe-4f83-bb7e-
576eadce8aef
    */
    TIoTCoreAppEnvironment *environment = [TIoTCoreAppEnvironment shareEnvironment];
    [environment setEnvironment];
    environment.appKey = @"物联网开发平台申请的 App Key";
    environment.appSecret = @"物联网开发平台申请的 App Secret";
```

```
    self.window.rootViewController = [[UINavigationController alloc] initWithRootViewController:
[UIViewController new]];
    return YES;
}
```

Android

选择 `sdkdemo/src/main/java/com/tencent/iot/explorer/link/core/demo/App.java`，配置 App key。

```
class App : Application() {
    companion object {
        val data = AppData.instance
    }
    private val APP_KEY = "物联网开发平台申请的 App Key"
    private val APP_SECRET = "物联网开发平台申请的 App Secret"
    override fun onCreate() {
        super.onCreate()
        /*
         * 此处仅供参考，需自建服务接入物联网平台服务，以免 App Secret 泄露
         * 自建服务可参考此处 https://cloud.tencent.com/document/product/1081/67504#e4359606-62fe-4f83-
bb7e-576eadce8aef
         */
        IoTAuth.openLog(true)
        IoTAuth.init(APP_KEY, APP_SECRET)
    }
    override fun onTerminate() {
        super.onTerminate()
    }
}
```

2. App SDK 功能划分说明。
   - iOS

iOS 对应模块	实现相关功能
QCDeviceCenter	配网模块。
QCAPISets	设备控制、消息相关、家庭管理、账户管理等 API。
QCFoundation	工具类。

   - Android

Android 子模块	实现相关功能
link	配网模块。
auth	设备控制、消息相关、家庭管理、账户管理等 API。
utils	工具类。
log	日志模块。

3. 账户相关接口，包含手机号、邮箱注册，登入登出，密码操作，用户信息操作。账户详细接口请参考 官方文档，或者 App SDK 文件中 iOS 文件（TIoTCoreAccountSet.h）或 Android 文件（IoTAuth.kt）。

> ⚠ 注意：

> 此处仅为 Demo 演示功能，**请遵从官方建议自建账户后台服务后，由自建服务接入物联网平台服务，以免 App Secret 泄露。**

4. 详细功能请参考 App SDK 接入指南 。

自建服务

搭建后台服务，将 App API 调用由设备端发起切换为由自建后台服务发起。
关于应用端 API，请参见 应用端 API 简介 。

> ⚠ **注意:**
> 登录前所使用的 API URL 为 `https://iot.cloud.tencent.com/api/exploreropen/appapi` ，不建议在设备端调用，需要替换为自建的后台服务，以避免密钥的泄漏。

- api/studioapp/* 为公版 App 专用，OEM 的 App 使用的是应用端 API（api/exploreropen/），当在 App 参数写入配置文件中配置 TencentIotLinkAppkey 后, api/studioapp 调用将自动切换为应用端 API 调用。
- App API（api/exploreropen/appapi）请在自建后台进行调用，Token API（api/exploreropen/tokenapi）可安全在设备端调用。
- iOS 版本 可通过 TIoTAppEnvironment.m 的 selectEnvironmentType 方法中设置此 API 。
  登录前所使用的 API URL 在 environment.oemAppApi 配置，请务必替换成自建的后台服务地址。

```
- (void)selectEnvironmentType {
    TIoTAppConfigModel *model = [TIoTAppConfig loadLocalConfigList];
    TIoTCoreAppEnvironment *environment = [TIoTCoreAppEnvironment shareEnvironment];
    [environment setEnvironment];
    environment.appKey = model.TencentIotLinkAppkey;
    environment.appSecret = model.TencentIotLinkAppSecret;
    // 请在 [environment setEnvironment]; 之后设置 oemAppApi 以免被覆盖。
    environment.oemAppApi = @"需要替换为自建后台服务地址";
}
```

TIoTCoreAppEnvironment.m 的 setEnvironment方法中默认 API 配置说明。

```
- (void)setEnvironment {

    //公版&开源体验版使用  当在 app-config.json 中配置 TencentIotLinkAppkey TencentIotLinkAppSecret 后，将自动切换为 OEM 版本。
    self.studioBaseUrl = @"https://iot.cloud.tencent.com/api/studioapp";
    self.studioBaseUrlForLogined = @"https://iot.cloud.tencent.com/api/studioapp/tokenapi";
    //OEM App 使用
    self.oemAppApi = @"https://iot.cloud.tencent.com/api/exploreropen/appapi"; // 需要在 TIoTAppEnvironment.m 的 -selectEnvironmentType: 中替换为自建后台服务地址。
    self.oemTokenApi = @"https://iot.cloud.tencent.com/api/exploreropen/tokenapi";  // 可安全在设备端调用。
    self.wsUrl = @"wss://iot.cloud.tencent.com/ws/explorer";
    self.h5Url = @"https://iot.cloud.tencent.com/explorer-h5";
}
```

- Android 版本 可通过 HttpRequest 中设置 API 。
  登录前所使用的 API URL 在 OEM_APP_API 配置，请务必替换成自建的后台服务地址。

```
/*
  接口请求文件
 */
class HttpRequest private constructor() {
companion object {
// 公版&开源体验版使用  当在 app-config.json 中配置 TencentIotLinkAppkey TencentIotLinkAppSecret 后，将自动切换为 OEM 版本。
const val STUDIO_BASE_URL = "https://iot.cloud.tencent.com/api/studioapp"
const val STUDIO_BASE_URL_FOR_LOGINED = "https://iot.cloud.tencent.com/api/studioapp/tokenapi"
// OEM App 使用
```

```
const val OEM_APP_API = "https://iot.cloud.tencent.com/api/exploreropen/appapi" // 需要替换为自建后台服务地
址
const val OEM_TOKEN_API = "https://iot.cloud.tencent.com/api/exploreropen/tokenapi"  // 可安全在设备端调
用。
const val APP_COS_AUTH = "https://iot.cloud.tencent.com/api/studioapp/AppCosAuth"
const val BUSI_APP = "studioapp"
const val BUSI_OPENSOURCE = "studioappOpensource"
}
}
```

开发常见问题

最近更新时间：2024-10-08 18:37:11

Android SDK 开发问题

怎么分辨我用的网络是 2.4G 的还是 5G 的?

PC 端查看步骤如下：

1. 单击 Windows 系统桌面右下角的网络标志。
2. 在弹出的框中，单击**网络和 Internet 设置**。
3. 在打开的**设置**窗口中，单击 **WLAN** 下的**硬件属性**，即可看到网络频带。

接入 SDK，调用 SDK 初始化方法后应用程序报错：java.lang.ClassNotFoundException: Didn't find class "org.java_websocket.client.WebSocketClient"。

您需要在 App 的 application 模块的 `build.gradle` 文件中 dependecies 加入：

```
dependencies {
  implementation "org.java-websocket:java-WebSocket:1.4.0"
}
```

接入SDK，调用 SDK 方法后应用程序报错：`java.lang.NoClassDefFoundError: Failed resolution of: Lkotlinx/coroutines/Dispatchers` 或者 `java.lang.ClassNotFoundException: Didn't find class "kotlinx.coroutines.Dispatchers"`

您需要在 App 的 application 模块的 `build.gradle` 文件中 dependecies 加入：

```
dependencies {
  implementation "org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.4"
}
```

iOS SDK 开发问题

怎么分辨我用的网络是 2.4G 的还是 5G 的?

Mac 查看步骤：按住 option 键，点击桌面右上角 Wi-Fi 图标，即可看到频段信息。

App SDK 开发其他问题

有多少用户可以同时使用一个账户登录?

您好，没有限制。

为什么用邮箱注册账号的时候收不到验证码?

您好，一般情况下都是可以收到验证码的。如果有超时现象，需要首先跟用邮箱注册的用户确认是否该验证码邮件被收在垃圾邮箱。我们的验证码发送邮箱地址是 cloud_smart@tencent.com，请确认是否收到该邮箱发送的邮件。有些邮箱可能会拦截我们的验证码邮件，可以设置邮箱的白名单，不拦截此账号发送的邮件。如仍有问题，您可以提供下未收到验证码的 App 信息和用户 App 账号，提交给专业的工程师处理。

设备连接的 Wi-Fi 名称和密码有什么规范么?

在 App 添加设备联网时，Wi-Fi 的名称没有限制，Wi-Fi 的密码长度最多58位。

无线路由器的设备接入上限是多少?

连接设备的数量是由路由器决定的，一般普通的家用路由器可以连接10个左右，根据您所选的路由器参数不同上限数量也会有不同。

Smartconfig（智能）配网模式与 Soft AP（自助）配网模式有什么区别?

- Smartconfig（智能）配网模式：
  Smartconfig 是手机 App 端发送包含 Wi-Fi 用户名和 Wi-Fi 密码的 UDP 广播包或者组播包，智能终端的 Wi-Fi 芯片可以接收到该 UDP 包，只要知道 UDP 的组织形式，就可以通过接收到的 UDP 包解密出 Wi-Fi 用户名和密码，然后智能硬件配置将接收到的 WIFI 用户名密码发送到指定的 Wi-Fi AP 上。
- Soft AP（自助）配网模式：
  App 配置手机连接到智能硬件（ Wi-Fi 芯片的 AP），手机与 Wi-Fi 芯片直接建立通讯，将要配置的 Wi-Fi 用户名和 Wi-Fi 密码发送给智能硬件，此时智能硬件便可以连接到配置的路由器上。

当我使用新的路由器，如何进行变更设置?

当变更了路由器和家庭网络之后，原先添加的设备会离线，请将原先的设备从 App 移除后，使用新的网络（5G暂时不支持，需使用2.4G）重新添加一次即可。

设备添加成功后显示离线，怎么检查?

出现设备离线的情况，请按照以下列举的方法进行排查：

1. 请检查设备是否正常通电。
2. 设备是否有断过电或者断过网的情况，如断开过链接，上线有一个过程，请2分钟后确认是否显示在线。
3. 请排查下设备所在网络是否稳定，排查办法：将手机或者 iPad 置于同一个网络，并放到设备边上，尝试打开网页。
4. 请确认家庭 Wi-Fi 网络是否正常，或者是否修改过 Wi-Fi 名称、密码等，如果有，也需要重置设备并重新添加。
5. 如果网络正常，但是设备还是离线，请确认 Wi-Fi 连接数量是否过多。可以尝试重启路由器，给设备断电后重新上电，然后静待2分钟 – 3分钟后查看设备是否可以恢复连接。
6. 检查固件是否是最新版本，App 端检查路径：我 > 设置 > 关于 > 检查更新。

如果以上都已排除但还是有问题，建议您移除设备重新添加。移除后重新添加如果还是存在问题，请在 App 用户反馈中选择该设备，填写登录账号、设备 ID 后提交反馈给到我们，我们将会提交给专业技术工程师查询原因。

Wi-Fi 设备联网失败可能是什么原因?

请通过以下步骤进行排查：

1. 确保设备通电并开机。
2. 确保设备处于待配网（快闪/慢闪）状态，且指示灯状态与 App 配网状态一致。
3. 确保设备、手机、路由器三者靠近。
4. 确保设备所在网络流畅稳定，排查办法：将手机或者 iPad 置于同一个网络，并放到设备边上，尝试打开网页。
5. 确保输入的路由器密码正确，注意密码前后是否有空格。
6. 确保使用2.4G的 Wi-Fi 频段添加设备，Wi-Fi 需要开启广播，不可设置为隐藏。检查2.4G和5G是否共用为一个 SSID，建议修改为不同的 SSID。
7. 确保路由器无线设置中加密方式为 WPA2-PSK 类型、认证类型为 AES，或两者皆设置为自动。 无线模式不能为 11n only。
8. 若路由器接入设备量达到上限，可尝试关闭某个设备的 Wi-Fi 功能空出通道重新配置。
9. 若路由器开启无线 MAC 地址过滤，可尝试将设备移出路由器的 MAC 过滤列表，保证路由器没有禁止设备联网。
10. 确保路由器开启了DHCP服务，没有开启的话会导致地址被占用。
11. 如果以上未能解决，则可能是路由器跟设备的兼容性不好，建议您更换路由器再次尝试。

最多可以拥有多少个“家庭”？

最多可拥有20个家庭。

一个家庭内最多可以创建多少房间?

最多可拥有20个房间。

一个家庭里可以有多少个成员?

最多可以有20个成员。

一个家庭内，最多可以绑定多少设备?

最多不可超过1000个设备。

音视频服务 Video-sdk
概述

最近更新时间：2024-05-09 11:34:41

SDK介绍

IoT Video X-P2P SDK集成了音视频通讯模块，具体包含 iOS 版本 或 Android 版本 ，用户可参考 SDK Demo 快速接入 App SDK。

App SDK版本更新

V2.4

- 发布日期：2022-08-30
- 系统平台： iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境： Mac、Xcode / Android studio
- 更新内容：
  - 支持MJPEG类型设备出图。
  - 底层XP2P通讯协议升级，支持中转和打洞并行，快速出图。

V2.3

- 发布日期：2021-09-30
- 系统平台： iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境： Mac、Xcode / Android studio
- 更新内容：更新底层 XP2P 通讯协议，从而支持小程序插件

V2.2

- 发布日期：2021-07-30
- 系统平台： iOS / Android
- 开发语言：OC 语言 / Java
- 开发环境： Mac、Xcode / Android studio
- 更新内容：
  - 增加支持多通道设备预览。
  - Demo App 支持视频清晰度切换。
  - Demo App 支持边录边播、截图、云台控制。
  - Demo App 支持多路同屏。

服务提供方：腾讯云计算（北京）有限责任公司
参考文档：

- 开发者合规指南
- IoT Video X-P2P SDK个人信息保护规则

Android 应用端

最近更新时间：2024-11-26 18:00:11

本文为您介绍 Android 应用端通过 so 库和 aar 库方法集成 SDK。

使用动态库 so

下载路径

- **稳定版**
  进入 so 下载地址，选择对应 Latest Version（例如2.0.3），单击右侧 **Downloads > aar**。
- **SNAPSHOT 版**
  进入 so下载地址，单击 **Repositories > Snapshots > Browse Storage** 的 Path Lookup 输入框中输入 `com/tencent/iot/thirdparty/android` > xp2p-sdk >选择对应版本（例如1.0.2-SNAPSHOT），选择最新的 `.aar` 单击右键 **Downloads**。

> **说明：**
> 建议使用稳定版本，SNAPSHOT 版仅供开发自测使用。

引用工程

1. 解压上一步骤下载下来的 aar，目录结构如下：

```
├── assets
│   └── appWrapper.h （头文件）
├── jni
│   ├── arm64-v8a
│   │   └── libxnet-android.so
│   └── armeabi-v7a
│       └── libxnet-android.so
```

2. 将头文件和 so 动态库放在您工程目录下，确保 CMakeList.txt 文件可以找到对应的路径即可。使用示例如下：

```
├── cpp
│   ├── CMakeLists.txt
│   ├── include
│   │   └── appWrapper.h
│   ├── libs
│   │   ├── arm64-v8a
│   │   │   └── libxnet-android.so
│   │   └── armeabi-v7a
│   │       └── libxnet-android.so
│   └── native-lib.cpp
```

在 CMakeLists.txt 中加上以下代码即可。

```
add_library(test-lib SHARED IMPORTED)
set_target_properties(test-lib PROPERTIES IMPORTED_LOCATION
${PROJECT_SOURCE_DIR}/../jniLibs/${ANDROID_ABI}/libxnet-android.so)
include_directories(${PROJECT_SOURCE_DIR}/include)
target_link_libraries( native-lib test-lib ${log-lib})
```

使用 Android aar 库

引用稳定版

在应用模块的 build.gradle 中配置版本号，具体版本号请参见 版本号列表 。

```
dependencies {
    implementation 'com.tencent.iot.video:video-link-android:*.*.*'
}
```

引用 SNAPSHOT 版

1. 在工程的 build.gradle 中配置仓库 url。

```
allprojects {
    repositories {
        google()
        jcenter()
        maven {
            url "https://oss.sonatype.org/content/repositories/snapshots"
        }
    }
}
```

2. 在应用模块的 build.gradle 中配置版本号。

```
dependencies {
    implementation 'com.tencent.iot.video:video-link-android:*.*.*-SNAPSHOT'
}
```

> **说明：**
> 建议使用稳定版本，SNAPSHOT 版仅供开发自测使用。

SDK 接入指南

具体请参见 SDK 接入指引 。

App 接入 SDK

第三方 App 在接入 Video SDK 时，**建议将 secretId 和 secretKey 保存到自建后台，并请务必保证不被泄露**；而 xp2p info 需要 App 侧从本地的业务后台获取，获取到 xp2p info 后，可以通过上述的 startServiceWithXp2pInfo 接口将该 info 传给 SDK，示例代码如下：

```
...
String xp2p_info = getXP2PInfo(...) // 从自建后台获取xp2p info
XP2P.setCallback(this)
XP2P.startServiceWithXp2pInfo(id, product_id, device_name, xp2p_info)
```

iOS 应用端

最近更新时间：2024-11-26 18:00:11

本文为您介绍 iOS 应用端通过 C++ .a 库和 iOS 库方法集成 SDK。

使用 C++ .a 库方法

1. 库 git 地址 。
2. 工程引用步骤如下：将所有 .a 与 AppWrapper.h 文件加入工程中，同时加入 libc++，libsqlite3 和 libz 系统库。

使用 iOS 库方法

1. 库 git 地址 。
2. 工程引用步骤如下：
   2.1 安装 Xcode 开发工具 。
   2.2 集成 SDK 方式。
   执行如下命令采用 Cocoapods 方式集成。

```
pod 'TIoTLinkVideo'
```

SDK 接入指南

具体请参见 SDK 接入指引 。

错误码

最近更新时间：2024-11-26 18:00:11

错误码

错误码	十进制值	含义
XP2P_ERR_NONE	0	成功。
XP2P_ERR_INIT_PRM	-1000	入参为空。
XP2P_ERR_GET_XP2PINFO	-1001	SDK 内部请求 XP2P info 失败。
XP2P_ERR_PROXY_INIT	-1002	本地 P2P 代理初始化失败。
XP2P_ERR_UNINIT	-1003	数据接收或发送服务未初始化。
XP2P_ERR_ENCRYPT	-1004	数据加密失败。
XP2P_ERR_TIMEOUT	-1005	请求超时。
XP2P_ERR_REQUEST_FAIL	-1006	请求错误。
XP2P_ERR_VERSION	-1007	设备版本过低。
XP2P_ERR_APPLICATION	-1008	服务 application 初始化失败。
XP2P_ERR_REQUEST	-1009	服务 request 初始化失败。
XP2P_ERR_DETECT_NOTREADY	-1010	P2P 探测未完成。
XP2P_ERR_P2P_INITED	-1011	当前 Id 对应的 P2P 已完成初始化。
XP2P_ERR_P2P_UNININ	-1012	当前 Id 对应的 P2P 未初始化。
XP2P_ERR_NEW_MEMERY	-1013	内存申请失败。
XP2P_ERR_XP2PINFO_RULE	-1014	获取到的 XP2P info 格式错误。
XP2P_ERR_XP2PINFO_DECRYPT	-1015	获取到的 XP2P info 解码失败。
XP2P_ERR_PROXY_LISTEN	-1016	本地代理监听端口失败。
XP2P_ERR_CLOUD_EMPTY	-1017	云端返回空数据。
XP2P_ERR_JSON_PARSE	-1018	JSON 解析失败。
XP2P_ERR_SERVICE_NOTRUN	-1019	当前 Id 对应的服务没有在运行。
XP2P_ERR_CLIENT_NULL	-1020	从 map 中取出的 client 为空。

解决方案

P2P 初始化接口失败

- 当前传入的 Id 已经完成了 P2P 初始化，返回错误码：`XP2P_ERR_P2P_INITED` ，需要先销毁 P2P 资源或使用新 Id，日志如下：

```
p2p service is running with id:cam01, please stop it first
```

- 向云端请求 XP2P info 时云端回复数据为空，返回错误码：`XP2P_ERR_GET_XP2PINFO` 。需排查设置的设备三元组和云 API 账号信息是否正确，日志如下：

```
request xp2p_info failed, errmsg:empty reply from cloud
```

- 向云端请求 XP2P info 时云端回复数据中没有指定 JSON 字段，返回错误码：`XP2P_ERR_GET_XP2PINFO`。需排查设备是否上报了 XP2P info 到云端，日志如下：

```
request xp2p_info failed, errmsg:parse reply error
```

- 请求到的设备 XP2P info 格式错误，返回错误码：`XP2P_ERR_XP2PINFO_RULE`。一般为设备 SDK 版本过低所致，日志如下：

```
remote xp2p_info rule wrong:$xp2p_info
```

- 设备 SDK 版本与 App SDK 版本不匹配，返回错误码：`XP2P_ERR_VERSION`。需升级设备 SDK，日志如下：

```
The xp2p_device_sdk is low, Please upgrade the device version to at least $version
```

- 解码获取到的设备 XP2P info 失败，返回错误码：`XP2P_ERR_XP2PINFO_DECRYPT`。需排查传入的设备三元组信息是否正确，日志如下：

```
decrypt xp2p_info error
```

- 获取到的设备 XP2P info 信息无效，返回错误码：`XP2P_ERR_PROXY_INIT`。需检查设备端网络，确保网络正常，日志如下：

```
remote xp2pinfo is invalid
```

- 本地代理无法监听 TCP 端口，返回错误码：`XP2P_ERR_PROXY_INIT`。需检查 App 端网络，确保网络正常，日志如下：

```
proxy listen failed!
```或
```shell
cannot listen a port
```

启动数据传输服务失败

- P2P 未成功初始化便启动数据传输服务，返回错误码：`XP2P_ERR_P2P_UNININ`。需确保 P2P 初始化成功后再进行后续操作，日志如下：

```
p2p service is not running with id:cam01, please run it first
```

- P2P 探测未完成便启动数据传输服务，返回错误码：`XP2P_ERR_DETECT_NOTREADY`。需等待 ready 回调触发后再进行后续操作，日志如下：

```
p2p detect is not ready, state:0
```

- 创建 Application 失败。需检查 App 网络环境，确保网络正常，日志如下：

```
create AudioStream application failed
```

- 创建 Request 失败。需检查 App 网络环境，确保网络正常，日志如下：

```
create AudioStream request failed
```

语音数据发送失败

- 语音发送服务未启动，返回错误码：`XP2P_ERR_UNINIT`。需先启动语音发送接口，日志如下：

```
connot found request with service:AudioStream
```

- 语音服务已关闭，返回错误码：`XP2P_ERR_UNINIT`。需重新启动语音发送服务或停止语音发送接口调用，日志如下：

```
application is invalid
```

无法收到回调消息

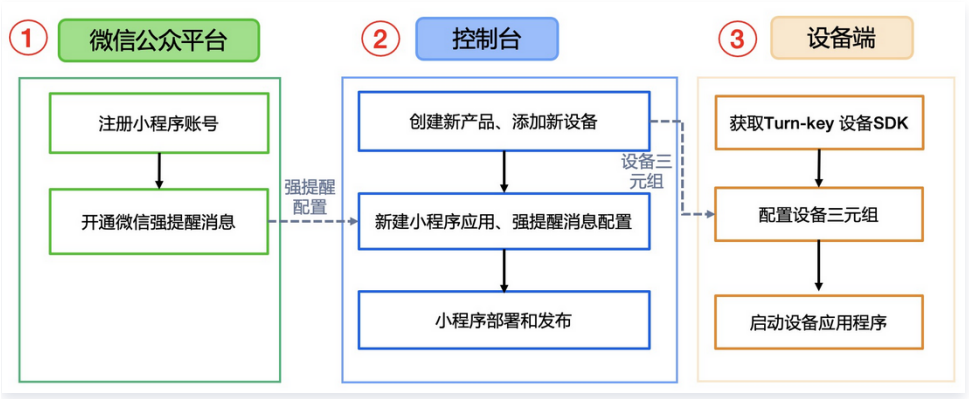
需要注册回调函数到 SDK。

低代码小程序托管
概述

最近更新时间：2024-11-27 11:00:11

搭建低代码的自主品牌小程序工作主要分为3个部分：准备工作、授权发布小程序和设备端功能开发。

流程图



流程简介

准备工作

作为由 IoT 平台代开发的小程序，其所有权归属于注册小程序的企业。因此需要登录 微信公众平台 注册小程序账号，并完成相关的配置后，开通订阅消息以实现微信消息强提醒推送。

授权发布小程序

在完成准备工作后，用户可在 物联网开发平台控制台 上创建产品、添加设备，并在应用开发模块将注册小程序授权给 IoT 平台，然后配置强提醒消息等信息后，即可体验生成的小程序，如满足要求则可提交微信审核并正式发布小程序。

设备端接入

在获得小程序后，则可根据在控制台添加设备得到的设备三元组信息对提供的相关版本设备端 SDK 进行相关功能的适配和调试。目前支持摄像机、图片流和视频流门锁设备。

准备工作

最近更新时间：2024-07-03 16:49:01

1. 注册微信小程序

访问 小程序，单击**立即注册 > 小程序**按指引完成微信小程序账号注册。

2. 设备接入小程序

需要确保小程序主体公司的营业执照经营范围含有**智能硬件销售及服务**或**物联网设备销售及服务**等描述，并且在国家工商官网可查询到，否则无法通过微信审核。

2.1 添加服务类目

登录 小程序管理后台，选择左侧最下方**设置 > 基本设置 > 基本信息 > 服务类目**后，点击右侧**详情**，进入下方页面。添加“工具 > 设备管理”类目（该类目无需资质，如已有该类目可忽略此流程。）

2.2 微信认证

> **注意：**
> 完成微信认证是开通设备管理的必要操作，请务必先完成微信认证后，再申请添加硬件设备。

1. 签署《微信公众平台认证服务协议》，勾选同意，单击**下一步**。

2. 填写资料：选择认证主体类型，提交相应的认证材料。
3. 发票填写：目前只支持电子发票和纸质增值税专用发票。其中增值税专用发票还需提交《税务登记证》（办理三证合一的企业直接上传新的营业执照）、《银行开户证明》，审核公司会对资质进行审核。资质审核通过后由腾讯公司开具并寄送发票。

> **注意：**
> - 发票资料提交后不能修改，请填写正确的发票类型和寄送地址，若填写错误造成发票开具错误、寄送错误或选择不开具发票，后续将无法重新开具并寄送发票。
> - 增值税专用发票的抬头为认证申请机构的全称，电子发票内暂不支持开具地址、电话、开户行及账号信息，腾讯云目前开具的发票也是可以正常报销的，如您需要开具以上信息，建议后续选择纸质增值税专用发票。
> - 订单完成后（包括认证成功和失败），腾讯会在认证审核完成后8个工作日左右开具电子发票，30个工作日开具并寄出增值税专用发票。

4. 支付费用：支付方式目前仅支持微信支付。

2.3 开通设备管理功能

进入小程序管理后台的**功能 > 硬件设备**页面，阅读设备使用条件和接入流程等后，单击**开通**并由小程序管理员扫码确认后，显示开通成功，即可进入设备管理页面。

2.4 添加硬件设备

进入**硬件设备**，单击**添加设备**，按照每个字段对应的说明填写设备相关信息，如门锁品类，则可添加一个门锁类的硬件设备。此步骤需要微信审核，审核时间为1天 – 3天。

2.5 获取设备 model_id

设备添加成功后，可在**设备管理**的列表中，获得平台分配的 model_id ，model_id 是调用小程序设备能力相关接口的重要凭证。获取 model_id 后，小程序须为设备申请**消息能力**等设备能力。此步骤需要微信审核，审核时间为1天 – 3天。

3. 获取消息通知模板 ID

3.1 登录小程序管理后台，单击**功能 > 订阅消息 > 公共模板库 > 长期订阅**，查看可选用的设备消息模板。

摄像头品类订阅：

- **摄像头监测异常报警**
- **门铃呼叫提醒**

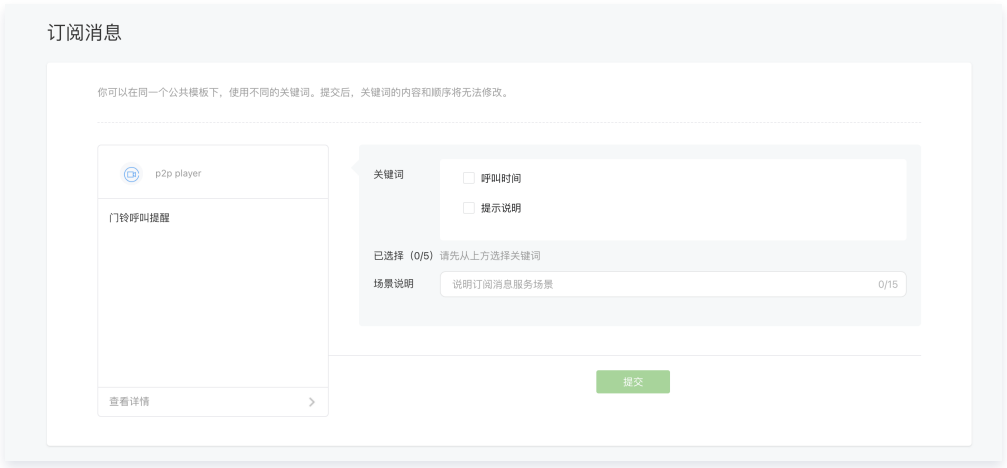
门锁品类订阅：

- **门锁异常提醒**
- **门铃呼叫提醒**

- 门锁安全告警
- 开关门提醒
- 门锁电量不足提醒
- 门锁撬动告警

3.2 在模板右侧单击**选用**，则可以进入具体页面，选择设备消息模板中需要的关键词，并单击**提交**。

> **注意：**
> 设备消息模板的关键词内容由平台生成，为枚举值，开发者不能够自定义内容。



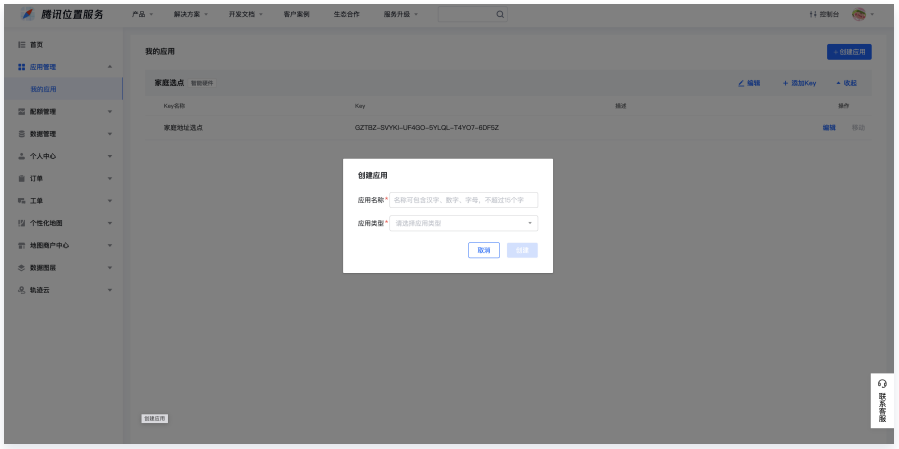
3.3 在**我的模板**中找到对应模板的模板 ID ，每个模板以 **template_id** 为标记。

> **注意：**
> - 摄像头品类的设备消息在订阅消息时，暂不支持勾选所在位置、剩余电量，否则会导致消息推送失败。

- 门锁品类的设备消息在订阅门锁电量不足提醒时，暂不支持勾选剩余电量，否则会导致消息推送失败。

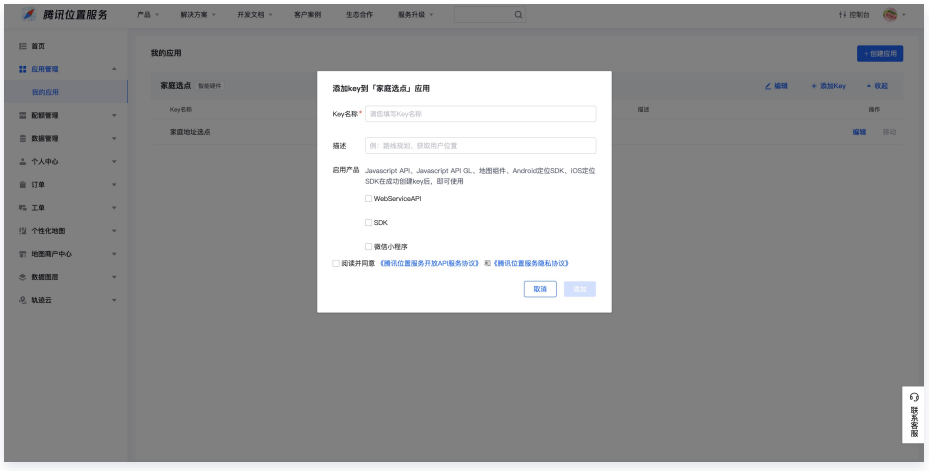
4. 接入小程序插件

- 申请 X-P2P插件
- 申请 p2p-player插件
- 申请 腾讯连连小程序插件
- 申请 腾讯位置服务地图选点插件
- 创建地图选点应用
  - 微信登录 腾讯位置服务 后，在应用管理中创建一个地图选点的应用。



  - 对创建的应用单击添加 key。

    注：请勾选WebService API

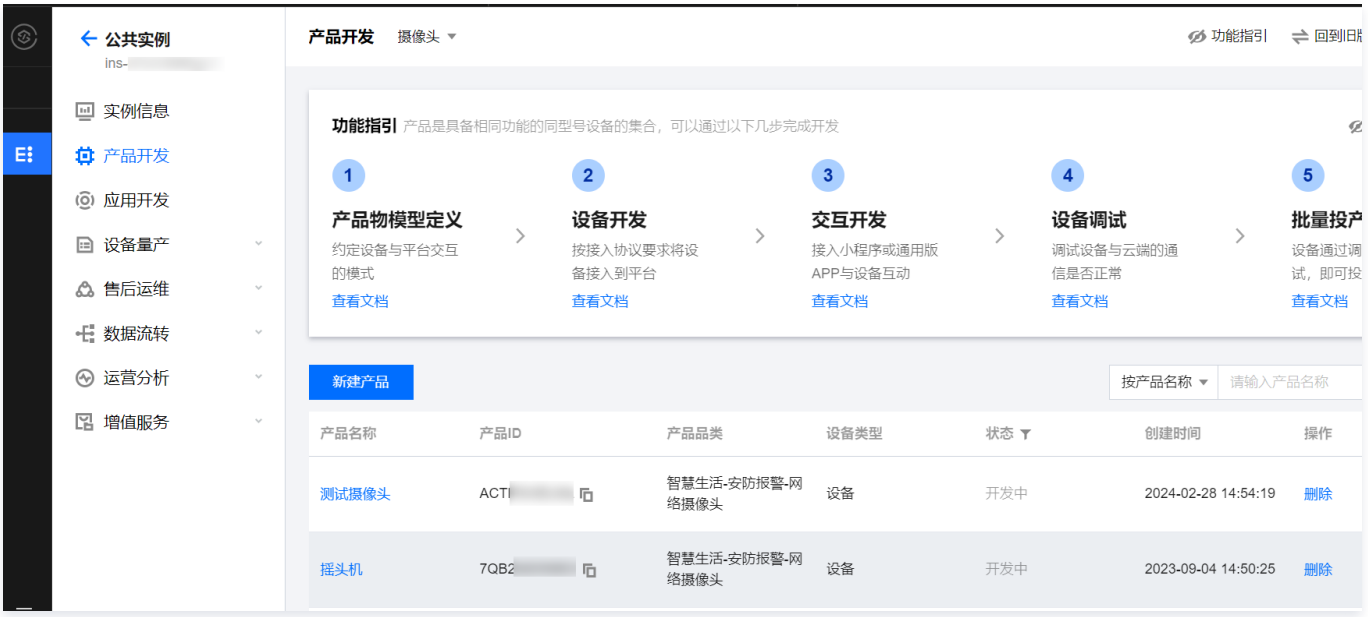


- 将注册小程序的 App ID 填入后，单击添加后，可在应用列表获得对应的 Key 字段。

5. 创建产品及设备

5.1 创建视频产品

1. 进入 控制台 > 产品开发，单击新建产品。



2. 创建产品品类，请选择视频服务中，对应的产品品类。

3. 填写产品信息时，选择购买的 license 对应的码率。如果提示激活码数量不足，请先购买才能创建产品。

5.2 创建设备

单击目标产品名称，进入设备调试，完成设备创建。

6. 微信支付设置信息

操作平台：微信支付 – 中国领先的第三方支付平台 | 微信支付提供安全快捷的支付方式

获取如下信息：

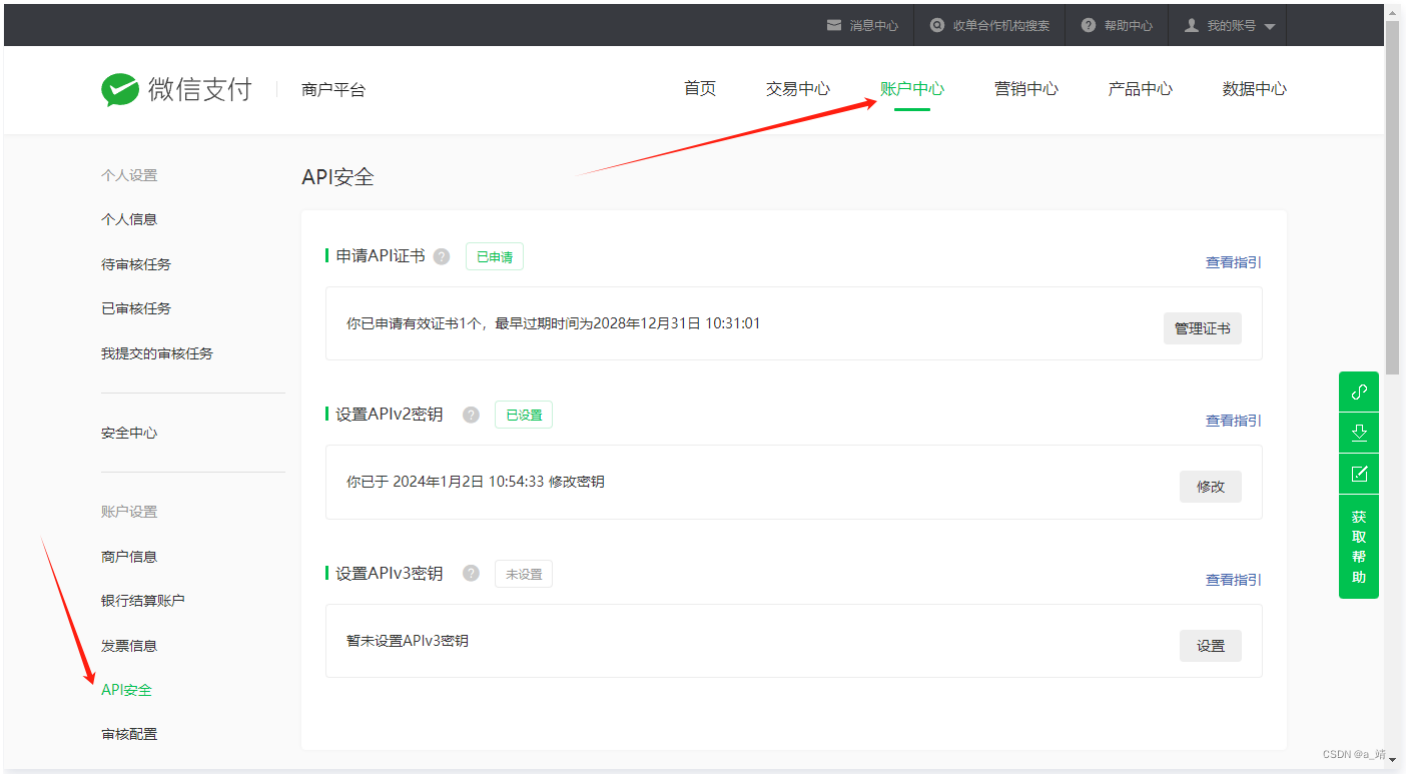
①微信支付商户号

②微信支付商户API密钥

③微信支付商户证书密钥

④微信支付商户证书

> **注意：**
> **请设置 APIv3 密钥。**



授权部署发布小程序

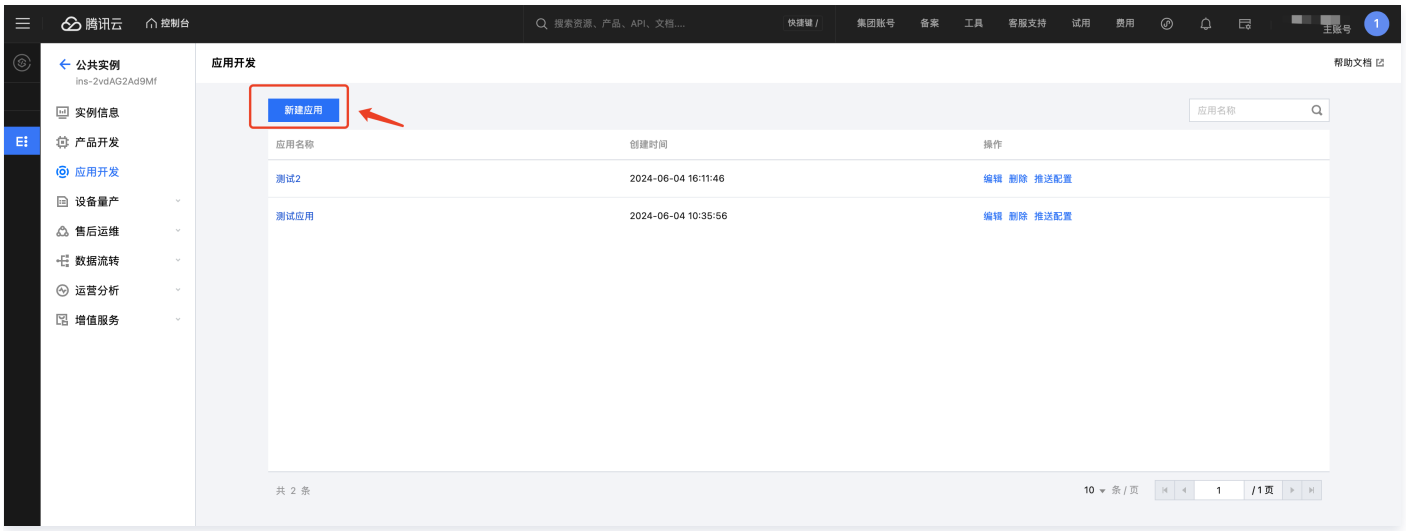
最近更新时间：2024-12-20 15:27:32

1.注册登录

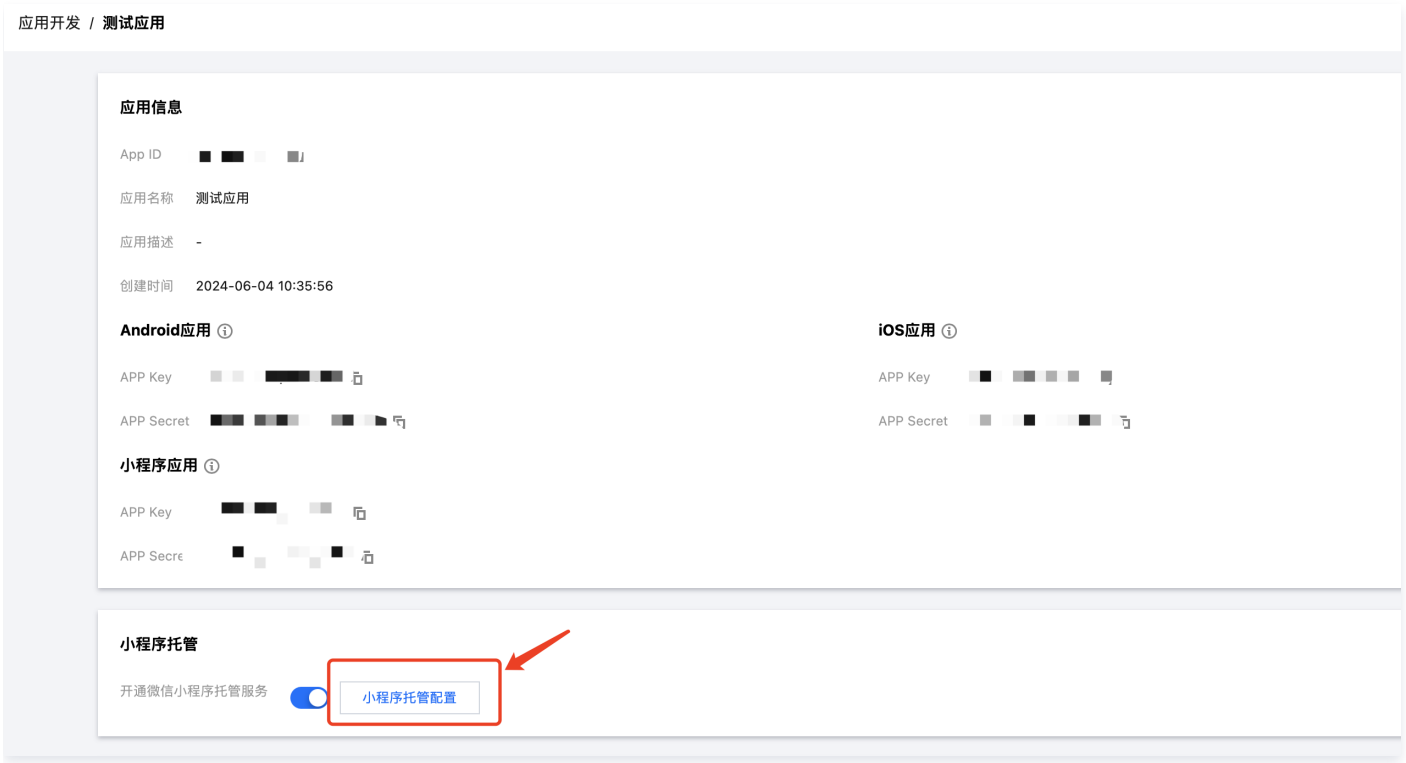
登录 腾讯云官网 。如果您之前未注册腾讯云账号，请参见 账号注册教程 完成注册。

2.创建应用

登录账号后，进入 物联网开发平台控制台 > 公共实例 > 应用开发，单击**新建应用**。



输入应用名称后，单击**保存**即创建完成。单击目标应用名称进入应用信息页面，然后开启小程序托管服务，并单击**小程序托管配置**进行下一步配置。

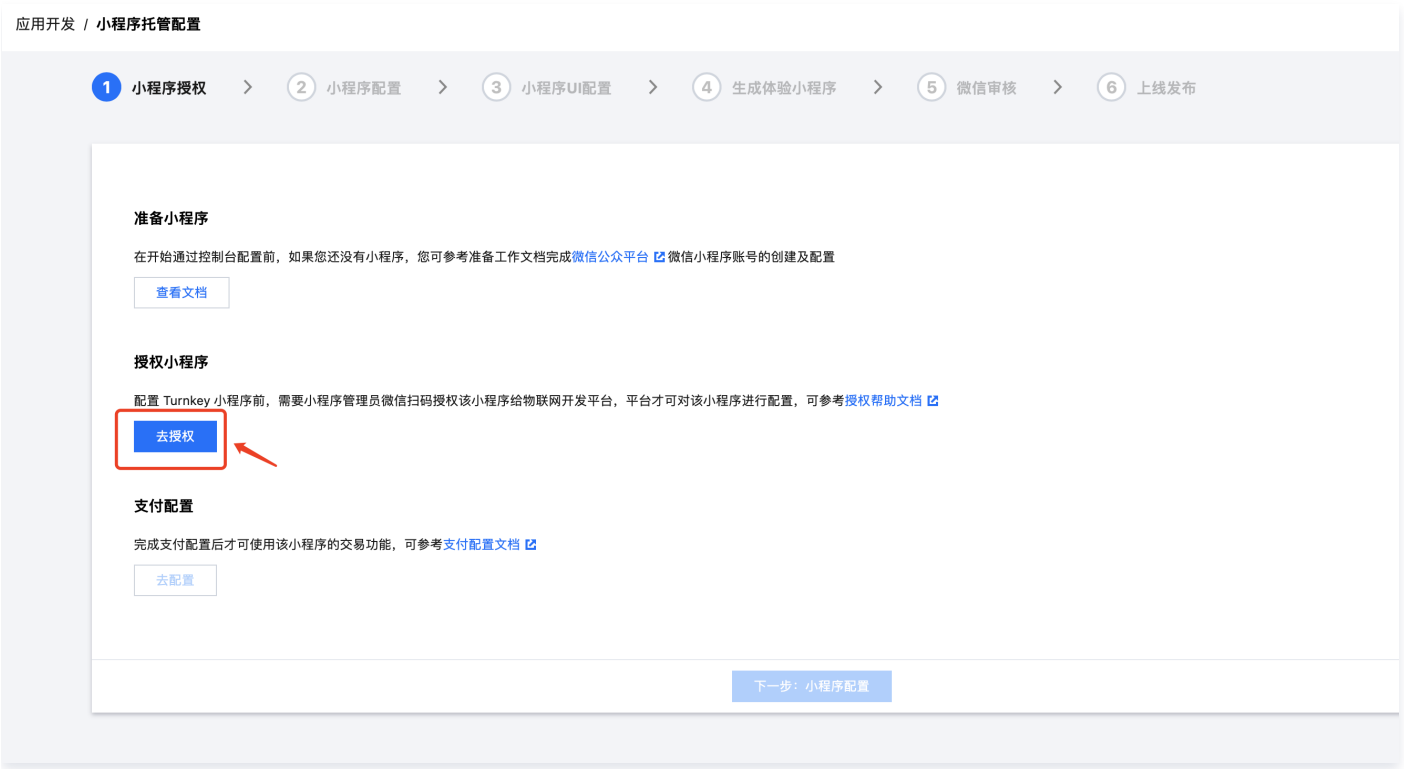


3.授权小程序

在小程序配置页面，单击**去授权**按钮，并完成后续授权动作。

> **说明：**

配置 Turnkey 小程序前，需要小程序管理员微信扫码授权该小程序给物联网开发平台，平台才可对该小程序进行配置。



4.小程序配置

家庭位置应用 key：填写准备工作 创建地图选点应用 时，获取得到的 Key 值。

强提醒消息配置：

- 产品品类：目前支持摄像头、门锁，共两种品类，支持多选。
- model_id：填写在小程序管理后台添加的指定品类的 model_id。
- 消息模板：填写小程序管理后台订阅的消息模板 id，注意检查是否填写正确，若填写错误，则小程序接收不到微信推送的强提醒消息。

售后与反馈：

- 售后电话（选填）：配置在小程序售后反馈的电话，不填写则不会在小程序上显示相关内容。
- 问题反馈指引（选填）：配置问题反馈的文案内容。
- App下载二维码（选填）：本地上传二维码图片，建议宽度不超过150px。

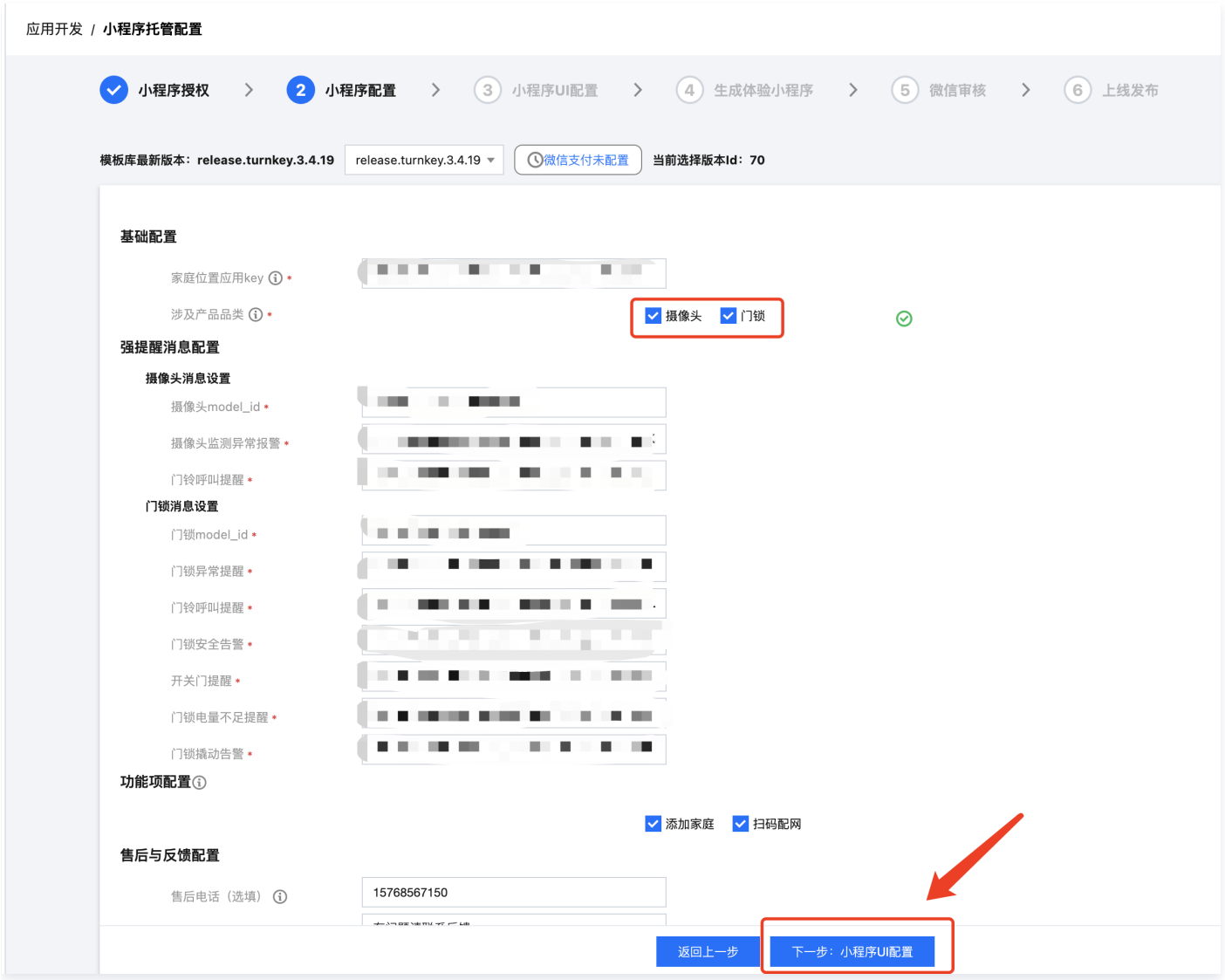
小程序使用说明配置：

小程序使用说明书（选填）：本地上传说明书的图片，建议宽度不超过370px。

产品使用说明配置：

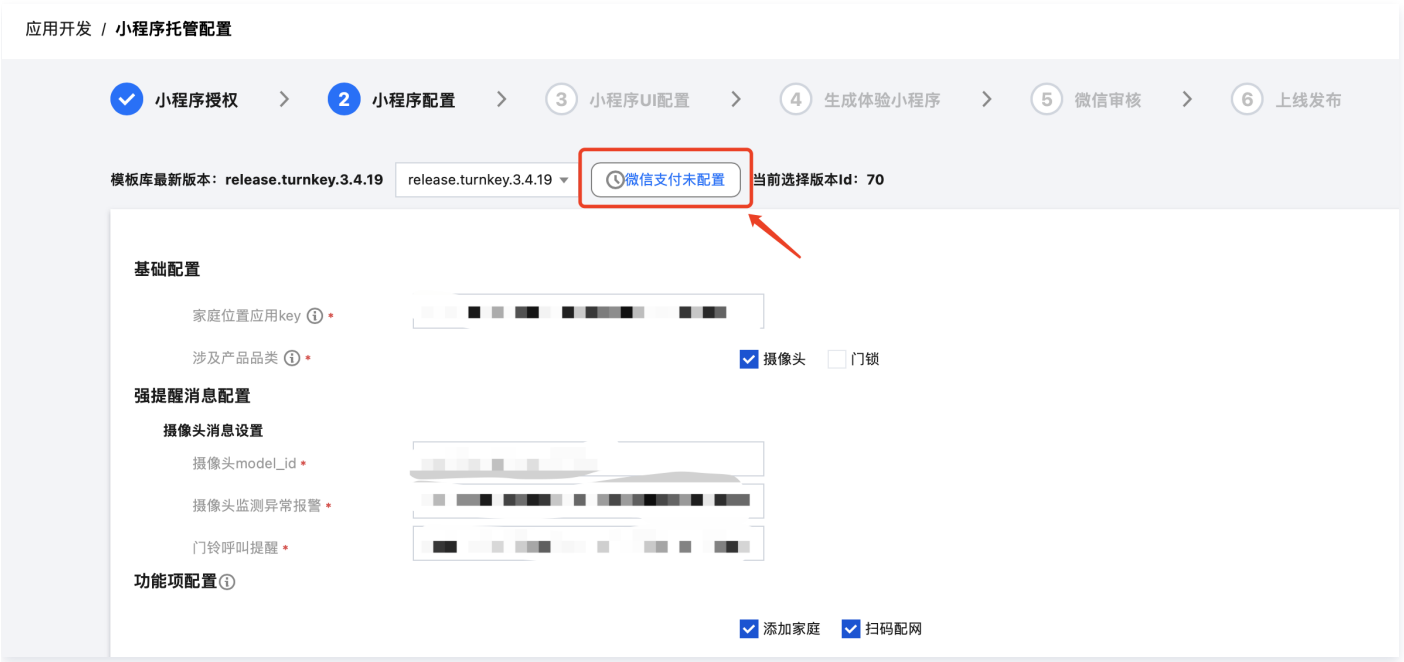
产品使用说明配置（选填）：单击下方添加产品使用说明后，输入硬件产品的名称，上传对应的产品使用说明书。

> **说明：**
> 支持添加多个产品使用说明书。



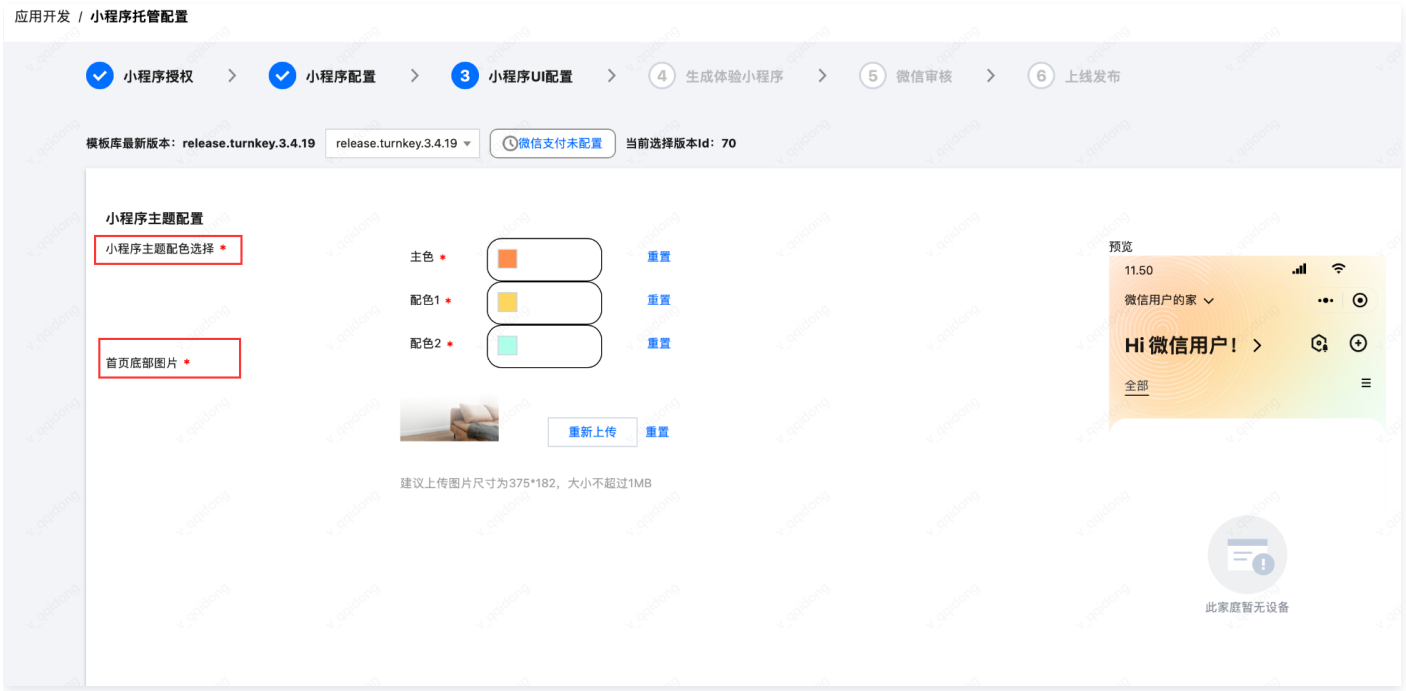
5.微信支付配置

单击微信支付配置按钮，可进入配置微信支付，配置信息填写 准备工作 中微信支付设置获取的信息。



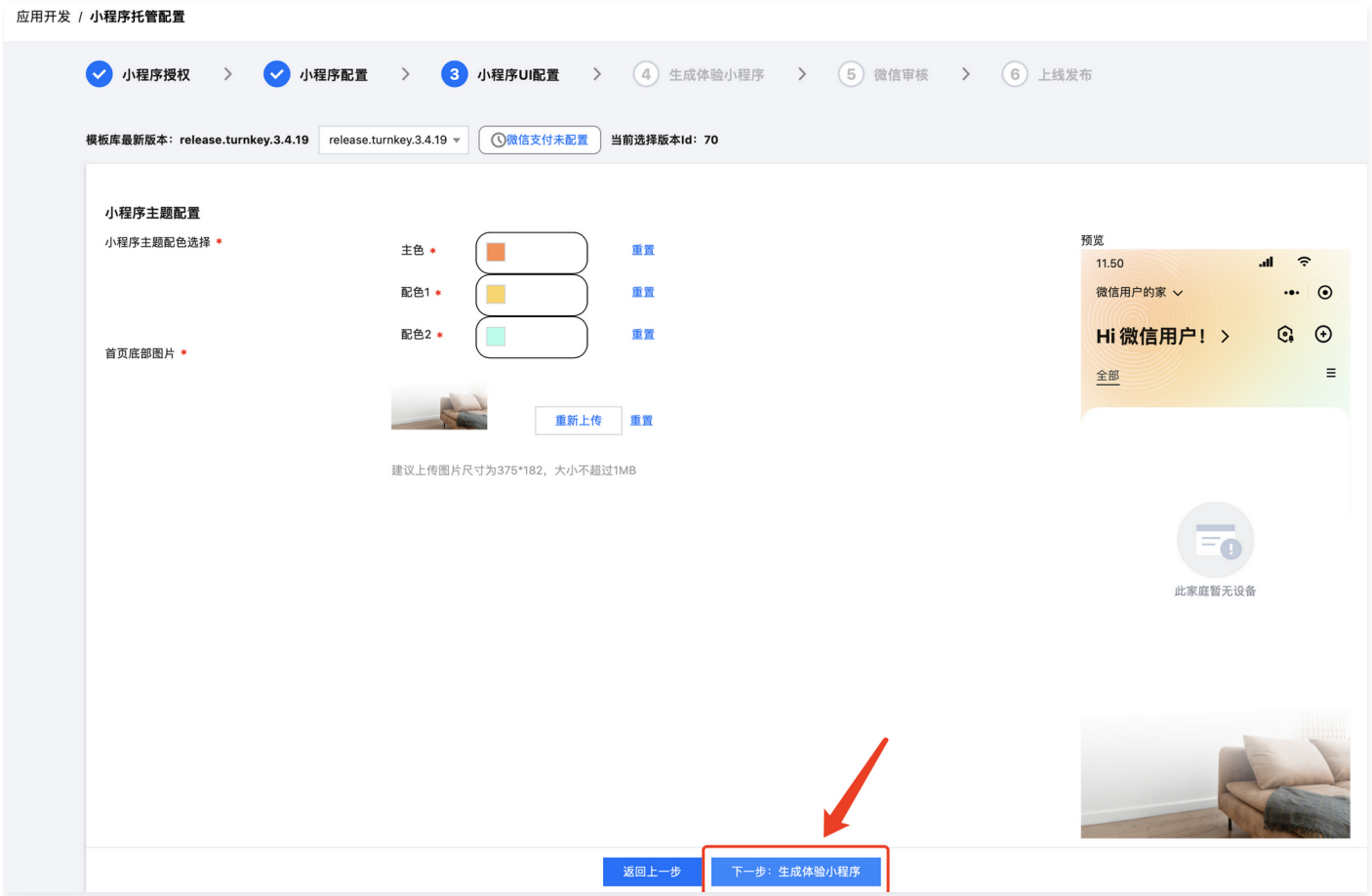
6.小程序 UI 配置

小程序主题配色选择：显示小程序默认的配色和预览效果，支持对主色和配色进行修改和重置。
首页底部图片：支持替换首页底部的图片，可按照建议本地上传。



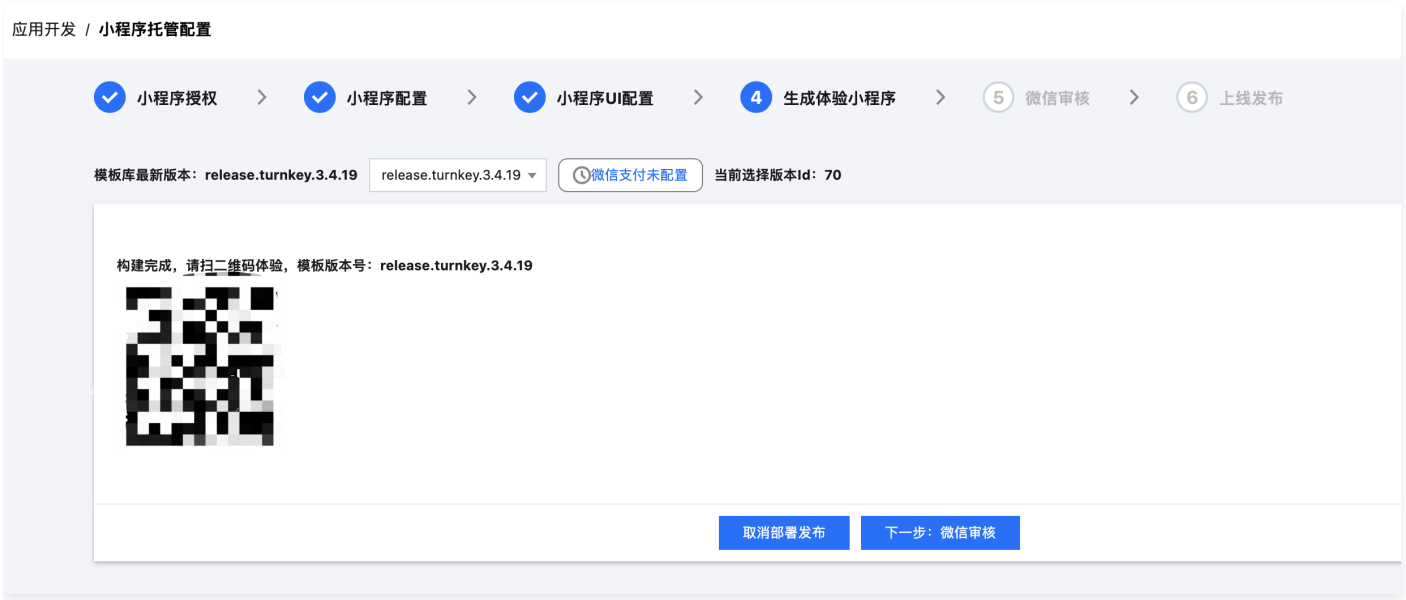
7.生成体验小程序

单击小程序配置页下方的**下一步：生成体验小程序**可跳到对应页面。



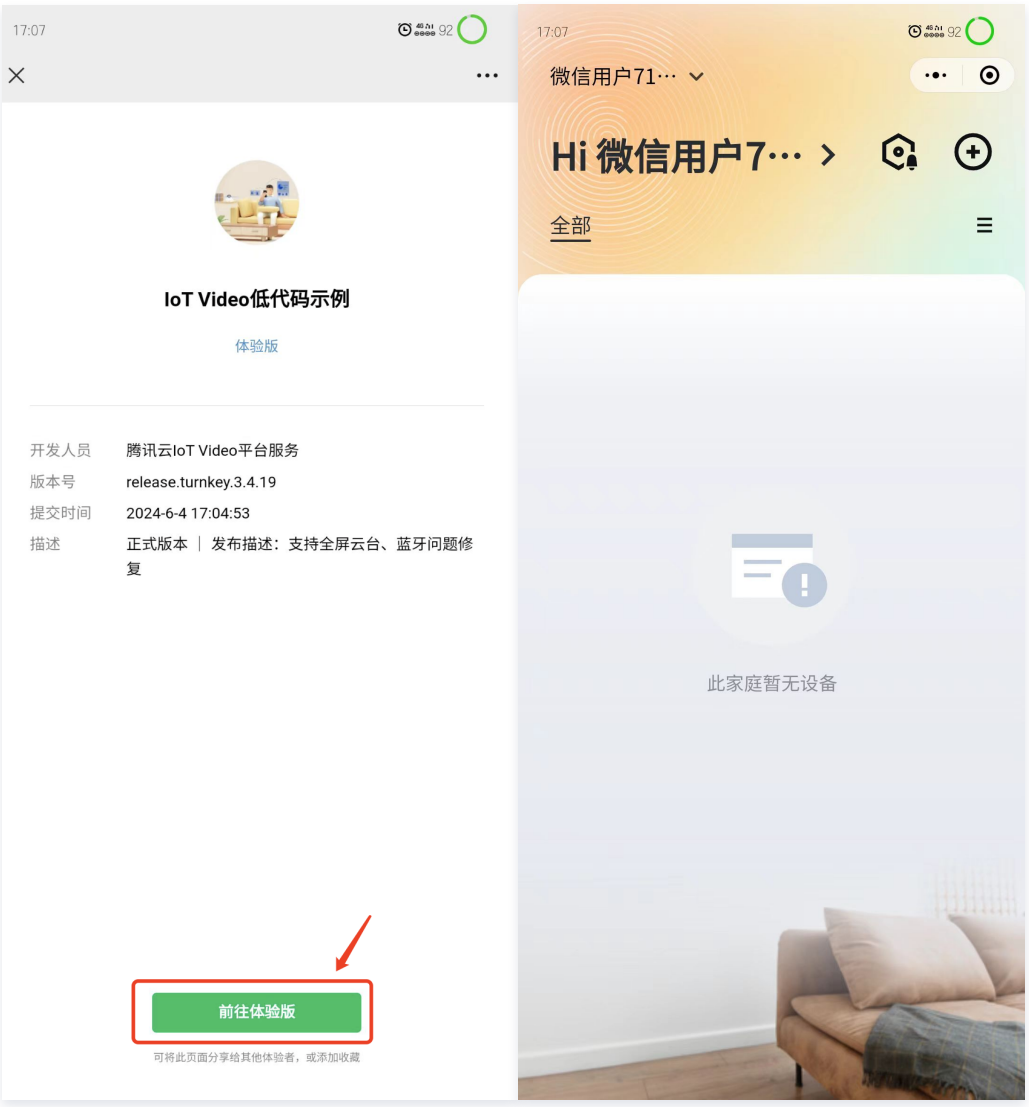
单击**立即构建**后，3分钟内生成体验二维码，通过手机扫描二维码进入体验版小程序进行功能体验并与设备端开启功能调试。

若单击**取消部署发布**，则可回到小程序配置步骤，重新进行相关配置。



8.手机预览小程序并添加设备

8.1. 手机扫描体验码进入小程序。

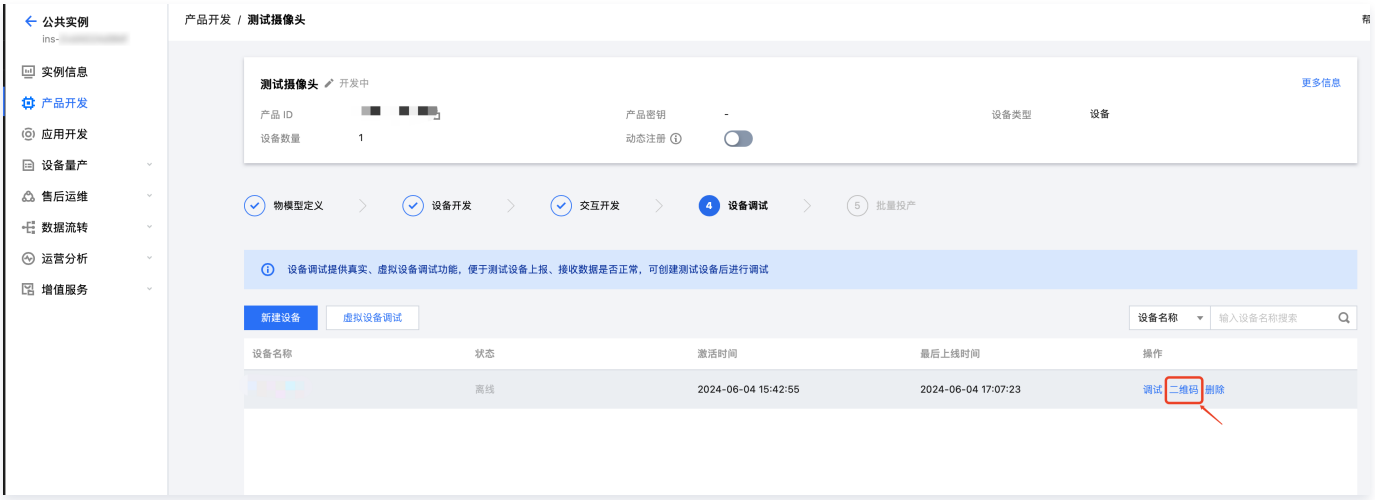


8.2. 操作应用关联产品

进入 **物联网开发平台控制台 > 公共实例 > 应用开发**，单击目标应用名称，进入应用关联产品设置，开启关联 **准备工作** 中创建的产品。

8.3. 获取设备二维码

进入 **物联网开发平台控制台 > 公共实例 > 产品开发**，单击目标产品名称后，进入设备调试页面，单击**二维码**即可成功获取。



8.4. 小程序端扫码添加设备

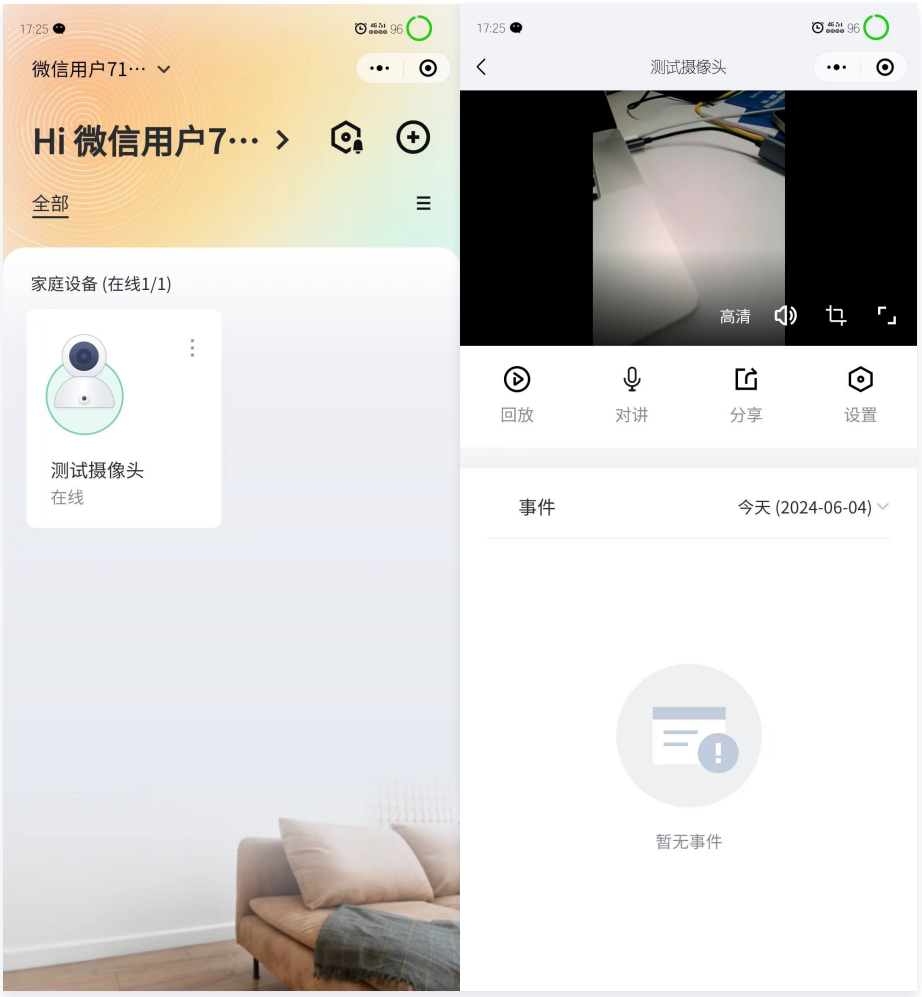
8.5. 用手机安装 DEMO 模拟设备

下载 Android 设备端 demo（app-debug.apk）安装在一个 Android 手机上，模拟设备端。

demo下载地址：https://drive.weixin.qq.com/s?k=AJEAIQdfAAodDxMR55

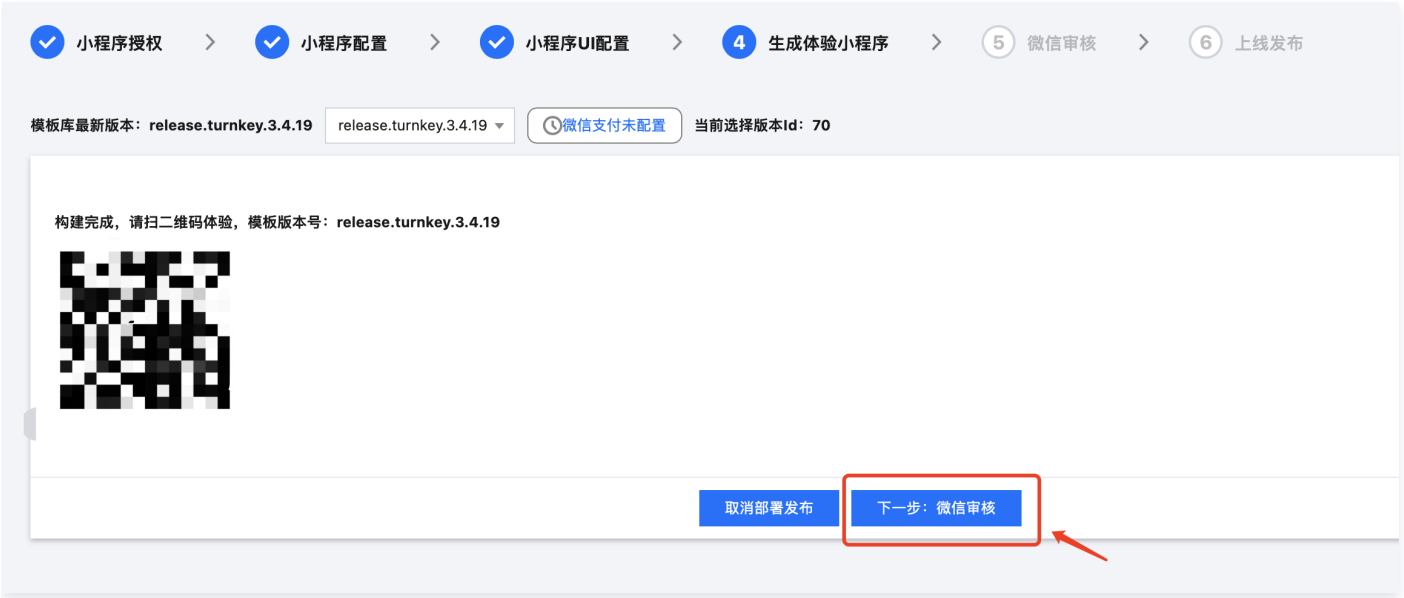
设备安装好 DEMO 后，输入设备三元组信息，设备即可上线，体验操作。

此时小程序端，设备也是上线状态，可以查看设备并操作。



9.微信审核

单击**下一步：微信审核**，则可进入微信审核页面，在单击**立即审核**后，则可等待微信审核，时间为1天 – 7天。期间可关闭弹窗，并可再次单击**部署发布**按钮进入弹窗查看审核结果，同时审核结果也会在审核完毕后第一时间以微信服务通知发送给小程序管理员。



10.上线发布

待微信审核通过后，可单击**下一步：上线发布**进入上线发布页面。单击**发布**后，即可正式发布小程序，并支持微信中搜索发布的正式版小程序并开始使用。

设备端接入（摄像机）

最近更新时间：2024-11-27 17:34:33

1. 摄像头扫码配网

设备侧需自行完成门锁摄像头扫码解析小程序二维码的功能，具体如下：

1.1 启动小程序，选择**手动添加设备**，确认设备重置后，选择家庭 Wi-Fi 并输入密码，则可使用摄像头扫描小程序生成的配网二维码，其中二维码解析后的格式如下：

- 内容格式：len str len str len str。其中 len：2个字符，代表紧随内容 utf-8 解码的长度，str 顺序：ssid、password、token。
- 示例：

  以 Tencent-wifi，密码为 l，token 为 v3_00d……为例，设备侧需要解析的字符串为：

  ```
  12Tencent-WiFi01l32v3_00d02b0248546d5a5fbf95a5812e4
  ```

  其对应的3个字段如下：

  ```
  {   "SSID": "Tencent-wifi",
      "password": "l",
      "token": "v3_00d02b0248546d5a5fbf95a5812e4"  }
  ```

1.2 设备侧解析完 Wi-Fi ssid 和密码后，自行完成联网操作。

1.3 连接完成后调用 系统模块 iv_sys_init() 和 物模型模块 iv_dm_init() 初始化必要的功能模块。

1.4 在确认初始化接口返回成功且确保设备上线后，调用 iv_ad_send_bind_info() 配网接口，上报上述配网 token。

1.5 上报成功后，刷新小程序主页面即可看到刚添加的设备。

2. 云台控制

设备侧需完成小程序云台信令的响应，参考 交互信令 。

3. 语音对讲

- 设备对接协议：设备到小程序上行仅支持 AAC，小程序到设备下行支持 AAC 16K 采样率或者设置任意采样率的 PCM，但下行 PCM 不支持小程序回音消除，推荐使用 AAC 16K 。
- 采样率：为了有更好的延时效果，设备到小程序的 AAC 格式请设置为 8K 采样率以上。
- 语音对讲设备侧 响应信令 。

4. 本地录像回放

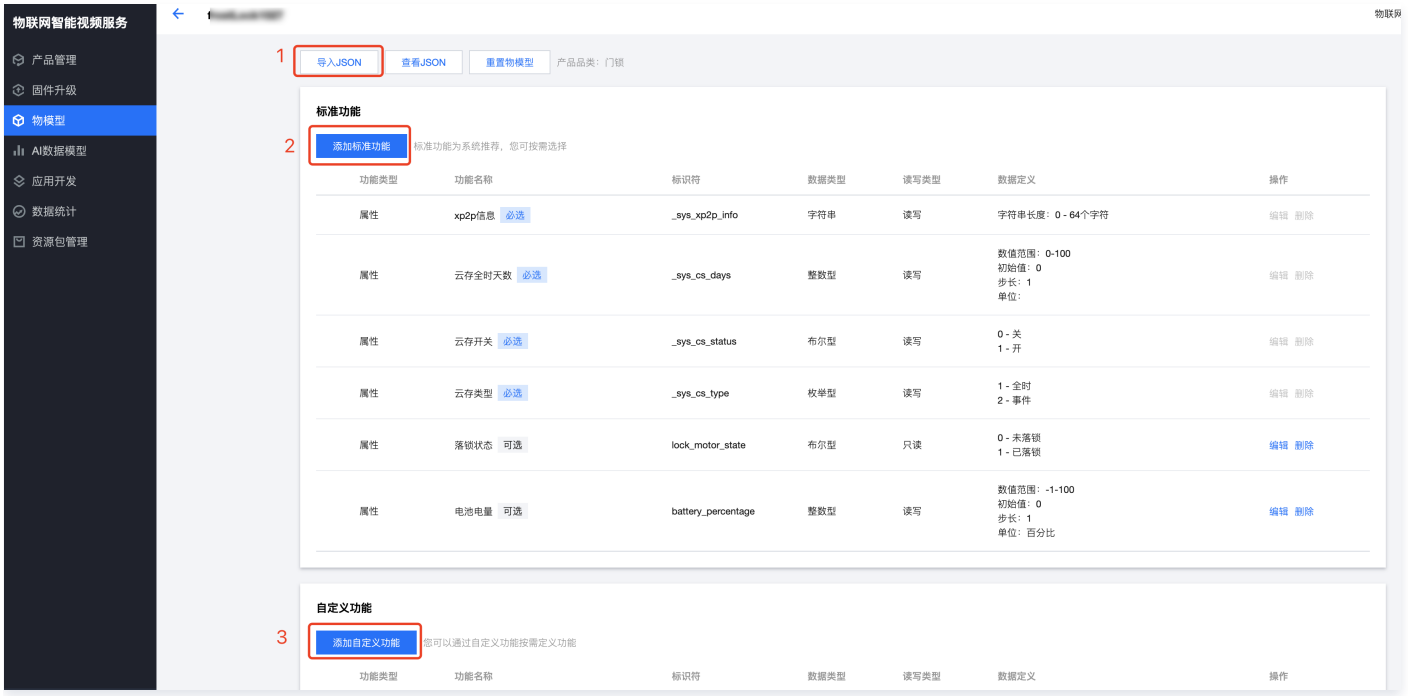
- 设备侧需响应 本地录像回放指令 。
- 获取某月有本地录像文件的日期：设备侧按需返回日期。
- 获取某一天所有本地录像文件的列表：设备侧需返回 json 列表。
- 点播某个录像文件：设备侧需 推送音视频流 。
- 暂停播放某个录像文件：设备侧需停止推送音视频流。
- 恢复播放某个录像文件：设备侧需继续推送音视频流。
- 实时显示某个录像文件的播放进度：设备侧需实时响应当前播放进度请求，并返回相对时间戳。

5. 云存回放

- 设备端对接设备端 SDK，某一个事件发生时，传入事件 ID、图片，并根据回调进行音视频流推送。
- 云存储模块说明 。
- 事件类型：
  - 有人活动：event id = 1。
  - 画面发生变化：event id = 2。

6. 相关物模型

所需的物模型需要在控制台的物模型页面进行操作，主要包括需要导入的相关物模型和添加的必选标准功能的物模型，同时也支持添加自定义功能的物模型。



导入相关物模型 json 内容：

```
{
  "version": "1.0",
  "profile": {
    "ProductId": "FT92OO0JFG",
    "CategoryId": "113"
  },
  "properties": [
    {
      "id": "_sys_xp2p_info",
      "name": "xp2p信息",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "string",
        "min": "0",
        "max": "64"
      },
      "required": false
    },
    {
      "id": "_sys_cs_days",
      "name": "云存全时天数",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "int",
        "min": "0",
        "max": "100",
        "start": "0",
        "step": "1",
        "unit": ""
      },
      "required": false
```

```
    },
    {
      "id": "_sys_cs_status",
      "name": "云存开关",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    },
    {
      "id": "_sys_cs_type",
      "name": "云存类型",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "1": "全时",
          "2": "事件"
        }
      },
      "required": false
    },
    {
      "id": "_sys_ai_status",
      "name": "AI开关",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    },
    {
      "id": "record_enable",
      "name": "本地录像使能",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    }
  ],
  "events": [],
  "actions": []
```

```
}
```

设备端接入（图片流门锁）

最近更新时间：2024-11-27 17:34:33

1. 门锁摄像头扫码配网

设备侧需自行完成门锁摄像头扫码解析小程序二维码的功能，具体如下：

1.1 启动小程序，选择**手动添加设备**，确认设备重置后，选择家庭 Wi-Fi 并输入密码，则可使用摄像头扫描小程序生成的配网二维码，其中二维码解析后的格式如下：

- 内容格式：len str len str len str。其中 len：2个字符，代表紧随内容 utf-8 解码的长度，str 顺序：ssid、password、token。
- 示例：

  以 Tencent-wifi，密码为 l，token 为 v3_00d……为例，设备侧需要解析的字符串为：

  ```
  12Tencent-WiFi01l32v3_00d02b0248546d5a5fbf95a5812e4
  ```

  其对应的3个字段如下：

  ```
  {   "SSID": "Tencent-wifi",
      "password": "l",
      "token": "v3_00d02b0248546d5a5fbf95a5812e4"  }
  ```

1.2 设备侧解析完 Wi-Fi ssid 和密码后，自行完成联网操作。

1.3 连接完成后调用 系统模块 iv_sys_init() 和 物模型模块 iv_dm_init() 初始化必要的功能模块。

1.4 在确认初始化接口返回成功且确保设备上线后，调用 iv_ad_send_bind_info() 配网接口，上报上述配网 token。

1.5 上报成功后,刷新小程序主页面即可看到刚添加的设备。

2. 语音对讲

- 设备对接协议：设备到小程序上行仅支持 AAC，小程序到设备下行支持 AAC 16K 采样率或者设置任意采样率的 PCM，但下行 PCM 不支持小程序回音消除，推荐使用 AAC 16K。
- 采样率：为了有更好的延时效果，设备到小程序的 AAC 格式请设置为 8K 采样率以上。
- 相关信令 。

3. 云存回放

- 购买云存储，需要通过 云 API 后台操作，为设备配置云存储套餐。
- 设备端对接设备端 SDK，某一个事件发生时，通过事件上报接口 iv_cs_event_start（ ）或者 iv_cs_event_directly_report（ ）向控制台后台传入事件 ID、图片，并根据回调进行音视频流推送。
- 云存储模块说明 。
- 事件类型：目前支持事件类型见下表，注意 ID 和事件名称的对应关系，后续将持续拓展。

事件类型	事件 ID
有人按门铃	event ID = 3
撬锁告警	event ID = 4
密码/指纹错误冻结告警	event ID = 5
胁迫告警	event ID = 6
徘徊告警	event ID = 7
开门通知	event ID = 8
关门通知	event ID = 9
指纹禁试告警	event ID = 10
密码禁试告警	event ID = 11

卡片禁试告警	event ID = 12
人脸禁试告警	event ID = 13
逗留侦测	event ID = 14
系统禁试告警	event ID = 15
假锁告警	event ID = 16
防遮挡告警	event ID = 17

4. 消息上报

- 设备端对接设备端 SDK：当事件发生时，设备端通过 iv_msg_send_notice（ ）传入通知 ID 到控制台后台，进行消息模板匹配，将通知参数传入小程序 {{time1.DATA}} {{enum_string2.DATA}} 后，即可进行消息通知推送。
- 通知类型：目前支持通知类型见下表，注意 ID 和通知的对应关系，后续将持续拓展。通知将显示在小程序设备详情页中，保存有效期30天。

通知类型	通知 ID
开门通知	notice ID = 1
关门通知	notice ID = 2
忘拔钥匙	notice ID = 3
反锁	notice ID = 4
低电量告警	notice ID = 5
有人按门铃	notice ID = 6
钥匙开门	notice ID = 7
临时密码开门	notice ID = 8
门未关提醒	notice ID = 9
未上锁提醒	notice ID = 10
门虚掩提醒	notice ID = 11
未知情况	notice ID = 12

说明：
- 设备端通过 iv_msg_send_notice（notice_id：1，ext：{unlock_type: xx}）上报开门通知。不传入 unlock_type，消息显示为“开门通知”；传入unlock_type，消息显示对应的 unlock_type。
- 设备端通过 iv_msg_send_notice（notice_id：12，ext：{unknown_info_id: xx}）上报消息。不传入 unknown_info_id，消息显示为“未知情况”；传入unknown_info_id，消息显示 id 对应的消息通知。

unlock_type 有下表几种：

unlock_type	通知
1	密码开门
2	人脸开门
3	指纹开门
4	卡片开门
5	门内开锁

6	APP蓝牙开门
7	指纹+密码开锁
8	人脸+指纹开锁
9	人脸+密码开锁

unknown_info_id 有下表几种：

unknown_info_id	通知
1	锁舌无法伸出告警
2	锁舌无法收回告警
3	防遮挡告警
4	布防告警
5	假期模式关闭告警
6	假锁告警

5. 落锁状态

- 设备端对接设备端 SDK，通过 mqtt 消息上报“落锁状态”属性物模型，同步门锁开关信息。

功能类型	功能名称	标识符	数据类型	读写类型	数据定义
属性	电池电量	battery_percentage	整数型	读写	数值范围：0-100 初始值：0 步长：1 单位：百分比
属性	落锁状态	lock_motor_state	布尔型	只读	0 - 未落锁 1 - 已落锁

- 物模型模块介绍 。

6. 远程解锁

设备端对接设备端 SDK，通过 SDK 接收到服务器下发的远程解锁行为物模型，接收解锁行为通知后，开锁指令下发成功，待小程序获取到落锁状态物模型由“已落锁”转为“未落锁”，则说明锁已开，提示用户“开锁成功”。

行为	远程解锁	unlock_remote	-	-	-

调用参数

参数名称	参数标识符	数据类型	数据定义
		暂无调用参数	

返回参数

参数名称	参数标识符	数据类型	数据定义
开锁成功状态	result	布尔型	0 - 开锁失败 1 - 开锁成功

7. 电池电量信息上报

设备端对接设备端 SDK，通过上报“电池电量”属性物模型，同步门锁电量信息。

功能类型	功能名称	标识符	数据类型	读写类型	数据定义
属性	电池电量	battery_percentage	整数型	读写	数值范围：-1-100 初始值：0 步长：1 单位：百分比

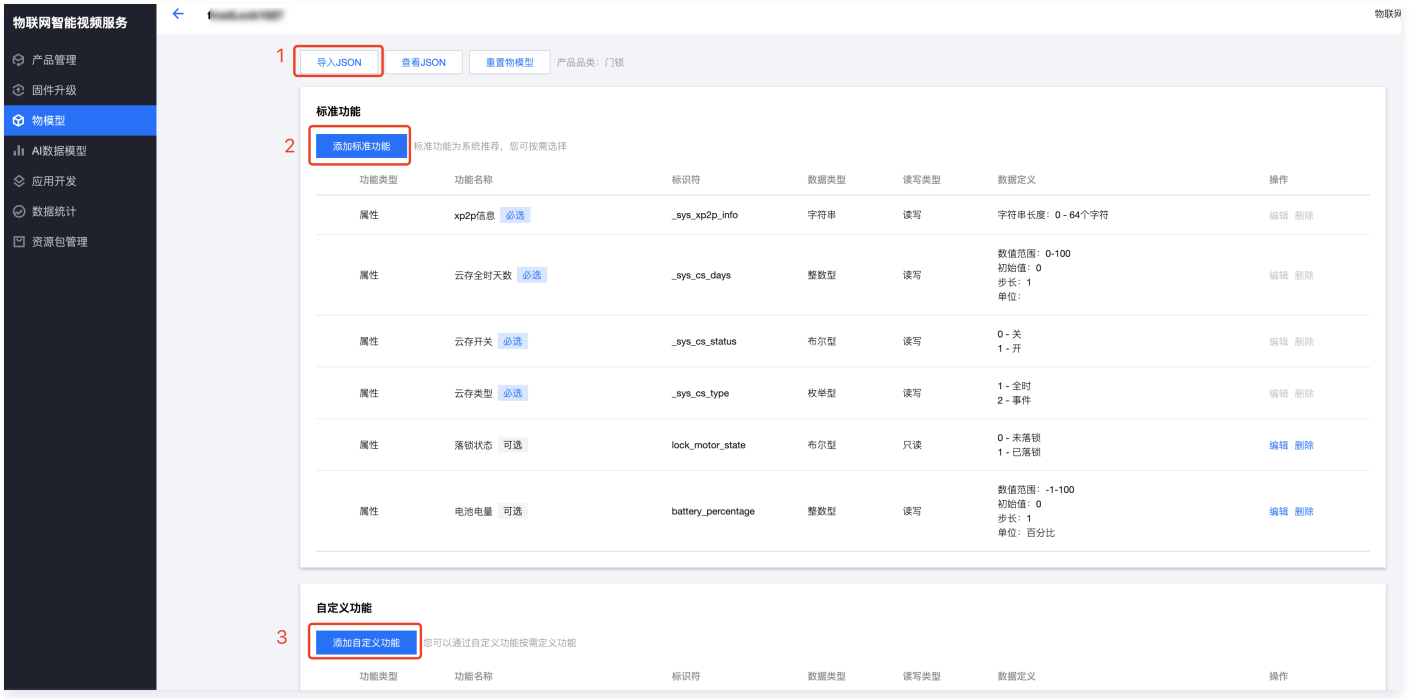
8. 临时密码解锁

- 原理：RFC4226
  - perio：密码有效期间隔，30 秒一个。
  - 将设备 PSK 密钥按照 base64 解码后得到原始二进制，得到 secret。
  - 将时间戳除以 period，向下取整，得到 counter。
  - 对 counter 计算 sha1。
  - 将 sha1 使用 dynamic truncation 计算为6位数字。
- 代码参考
- 设备端需自行实现 RFC4226 算法，设备端在用户输入临时密码时，获取输入密码对应的时间戳后，通过 RFC4226 算法生成密码，并与输入密码进行逐一比对，至对比成功则可正常开锁。

> **注意：**
> 当门锁状态显示已开锁时，临时密码不可使用。

9. 相关物模型

所需的物模型需要在控制台的物模型页面进行操作，主要包括需要导入的相关物模型和添加的必选标准功能的物模型，同时也支持添加自定义功能的物模型。



导入相关物模型：

```
{
  "version": "1.0",
  "profile": {
    "ProductId": "OSO00VC9U8",
    "CategoryId": "595"
  },
  "properties": [
```

```
    {
      "id": "battery_percentage",
      "name": "电池电量",
      "desc": "-1表示未获取到电量",
      "mode": "rw",
      "define": {
        "type": "int",
        "min": "-1",
        "max": "100",
        "start": "0",
        "step": "1",
        "unit": "百分比"
      },
      "required": false
    },
    {
      "id": "lock_state",
      "name": "门锁状态",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "0": "未落锁",
          "1": "已落锁",
          "3": "门锁异常"
        }
      },
      "required": false
    },
    {
      "id": "_sys_xp2p_info",
      "name": "xp2p信息",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "string",
        "min": "0",
        "max": "64"
      },
      "required": false
    },
    {
      "id": "_sys_cs_days",
      "name": "云存全时天数",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "int",
        "min": "0",
        "max": "100",
        "start": "0",
        "step": "1",
        "unit": ""
      },
      "required": false
    },
    {
      "id": "_sys_cs_status",
      "name": "云存开关",
      "desc": "",
```

```
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    },
    {
      "id": "_sys_cs_type",
      "name": "云存类型",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "1": "全时",
          "2": "事件"
        }
      },
      "required": false
    },
    {
      "id": "_sys_ai_status",
      "name": "AI开关",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    },
    {
      "id": "mjpg_info",
      "name": "图片流标识",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "struct",
        "specs": []
      },
      "required": false
    }
  ],
  "events": [],
  "actions": [
    {
      "id": "unlock_remote",
      "name": "远程解锁",
      "desc": "",
      "input": [],
      "output": [
        {
          "id": "result",
          "name": "开锁成功状态",
```

```
          "define": {
            "type": "bool",
            "mapping": {
              "0": "开锁失败",
              "1": "开锁成功"
            }
          }
        }
      ],
      "required": false
    }
  ]
}
```

设备端接入（视频流门锁）

最近更新时间：2024-11-27 17:34:33

1. 门锁摄像头扫码配网

设备侧需自行完成门锁摄像头扫码解析小程序二维码的功能，具体如下：

1.1 启动小程序，选择**手动添加设备**，确认设备重置后，选择家庭 Wi-Fi 并输入密码，则可使用摄像头扫描小程序生成的配网二维码，其中二维码解析后的格式如下：

- 内容格式：len str len str len str。其中 len：2个字符，代表紧随内容 utf-8 解码的长度，str 顺序：ssid、password、token。
- 示例：

  以 Tencent-wifi，密码为l，token 为 v3_00d……为例，设备侧需要解析的字符串为：

  ```
  12Tencent-WiFi01132v3_00d02b0248546d5a5fbf95a5812e4
  ```

  其对应的3个字段如下：

  ```
  {   "SSID": "Tencent-wifi",
      "password": "1",
      "token": "v3_00d02b0248546d5a5fbf95a5812e4"  }
  ```

1.2 设备侧解析完 Wi-Fi ssid 和密码后，自行完成联网操作。

1.3 连接完成后调用 系统模块 iv_sys_init() 和 物模型模块 iv_dm_init() 初始化必要的功能模块。

1.4 在确认初始化接口返回成功且确保设备上线后，调用 iv_ad_send_bind_info() 配网接口，上报上述配网 token。

1.5 上报成功后，刷新小程序主页面即可看到刚添加的设备。

2. 语音对讲

- 设备对接协议：设备到小程序上行仅支持 AAC，小程序到设备下行支持 AAC 16K 采样率或者设置任意采样率的 PCM，但下行 PCM 不支持小程序回音消除，推荐使用 AAC 16K。
- 采样率：为了有更好的延时效果，设备到小程序的 AAC 格式请设置为 8K 采样率以上。
- 相关信令参考 信令交互说明 。

3. 云存回放

- 购买云存储，需要通过 云 API 后台操作，为设备配置云存储套餐。
- 设备端对接设备端 SDK，某一个事件发生时，通过事件上报接口 iv_cs_event_start（）或者 iv_cs_event_directly_report（）向控制台后台传入事件 ID、图片，并根据回调进行音视频流推送。
- 云存储模块说明
- 事件类型：目前支持事件类型如下表，注意 ID 和事件名称的对应关系，后续将持续拓展。

事件类型	事件 ID
有人按门铃	event ID = 3
撬锁告警	event ID = 4
密码/指纹错误冻结告警	event ID = 5
胁迫告警	event ID = 6
徘徊告警	event ID = 7
开门通知	event ID = 8
关门通知	event ID = 9

指纹禁试告警	event ID = 10
密码禁试告警	event ID = 11
卡片禁试告警	event ID = 12
人脸禁试告警	event ID = 13
逗留侦测	event ID = 14
系统禁试告警	event ID = 15
假锁告警	event ID = 16
防遮挡告警	event ID = 17

4. 消息上报

- 设备端对接设备端 SDK：当事件发生时，设备端通过 iv_msg_send_notice（）传入通知 ID 到控制台后台，进行消息模板匹配，将通知参数传入小程序 {{time1.DATA}}
{{enum_string2.DATA}} 后，即可进行消息通知推送。
- 通知类型：目前支持通知类型如下表，注意 ID 和通知的对应关系，后续将持续拓展。通知将显示在小程序设备详情页中，保存有效期30天。

通知类型	通知 ID
开门通知	notice ID = 1
关门通知	notice ID = 2
忘拔钥匙	notice ID = 3
反锁	notice ID = 4
低电量告警	notice ID = 5
有人按门铃	notice ID = 6
钥匙开门	notice ID = 7
临时密码开门	notice ID = 8
门未关提醒	notice ID = 9
未上锁提醒	notice ID = 10
门虚掩提醒	notice ID = 11
未知情况	notice ID = 12

说明：
- 设备端通过 iv_msg_send_notice（notice_id：1，ext：{unlock_type: xx}）上报开门通知。不传入 unlock_type，消息显示为“开门通知”；传入unlock_type，消息显示对应的 unlock_type。
- 设备端通过 iv_msg_send_notice（notice_id：12，ext：{unknown_info_id: xx}）上报消息。不传入 unknown_info_id，消息显示为“未知情况”；传入 unknown_info_id，消息显示 id 对应的消息通知。

unlock_type 有以下几种：

unlock_type	通知
1	密码开门
2	人脸开门
3	指纹开门

4	卡片开门
5	门内开锁
6	APP蓝牙开门
7	指纹+密码开锁
8	人脸+指纹开锁
9	人脸+密码开锁

unknown_info_id 有以下几种：

unknown_info_id	通知
1	锁舌无法伸出告警
2	锁舌无法收回告警
3	防遮挡告警
4	布防告警
5	假期模式关闭告警
6	假锁告警

5. 落锁状态

- 设备端对接设备端 SDK，通过 mqtt 消息上报“落锁状态”属性物模型，同步门锁开关信息。

功能类型	功能名称	标识符	数据类型	读写类型	数据定义
属性	电池电量	battery_percentage	整数型	读写	数值范围：0-100 初始值：0 步长：1 单位：百分比
属性	落锁状态	lock_motor_state	布尔型	只读	0 - 未落锁 1 - 已落锁

- 物模型模块介绍 。

6. 远程解锁

设备端对接设备端 SDK，通过 SDK 接收到服务器下发的远程解锁行为物模型，接收解锁行为通知后，开锁指令下发成功，待小程序获取到落锁状态物模型由“已落锁”转为“未落锁”，则说明锁已开，提示用户“开锁成功”。

行为	远程解锁	unlock_remote	-	-	-

调用参数

参数名称	参数标识符	数据类型	数据定义
		暂无调用参数	

返回参数

参数名称	参数标识符	数据类型	数据定义
开锁成功状态	result	布尔型	0 - 开锁失败 1 - 开锁成功

7. 电池电量信息上报

设备端对接设备端 SDK，通过上报“电池电量”属性物模型，同步门锁电量信息。

功能类型	功能名称	标识符	数据类型	读写类型	数据定义
属性	电池电量	battery_percentage	整数型	读写	数值范围：-1-100 初始值：0 步长：1 单位：百分比

8. 临时密码解锁

- 原理：RFC4226
  - perio：密码有效期间隔，30 秒一个。
  - 将设备 PSK 密钥按照 base64 解码后得到原始二进制，得到 secret。
  - 将时间戳除以 period，向下取整，得到 counter。
  - 对 counter 计算 sha1。
  - 将 sha1 使用 dynamic truncation 计算为6位数字。
- 代码参考
- 设备端需自行实现 RFC4226 算法，设备端在用户输入临时密码时，获取输入密码对应的时间戳后，通过 RFC4226 算法生成密码，并与输入密码进行逐一比对，至对比成功则可正常开锁。

> ⚠ **注意：**
> 当门锁状态显示已开锁时，临时密码不可使用。

9. 主动唤醒

实现流程如下：

9.1. 小程序获取**唤醒状态**物模型的值为0，即休眠状态，则会下发**唤醒门锁**行为。

	属性	唤醒状态 可选	wakeup_state	布尔型	只读	0 - 休眠状态 1 - 唤醒状态

▾	行为	唤醒门锁 可选	wake_up	-	-	-

调用参数

参数名称	参数标识符	数据类型	数据定义
		暂无调用参数	

返回参数

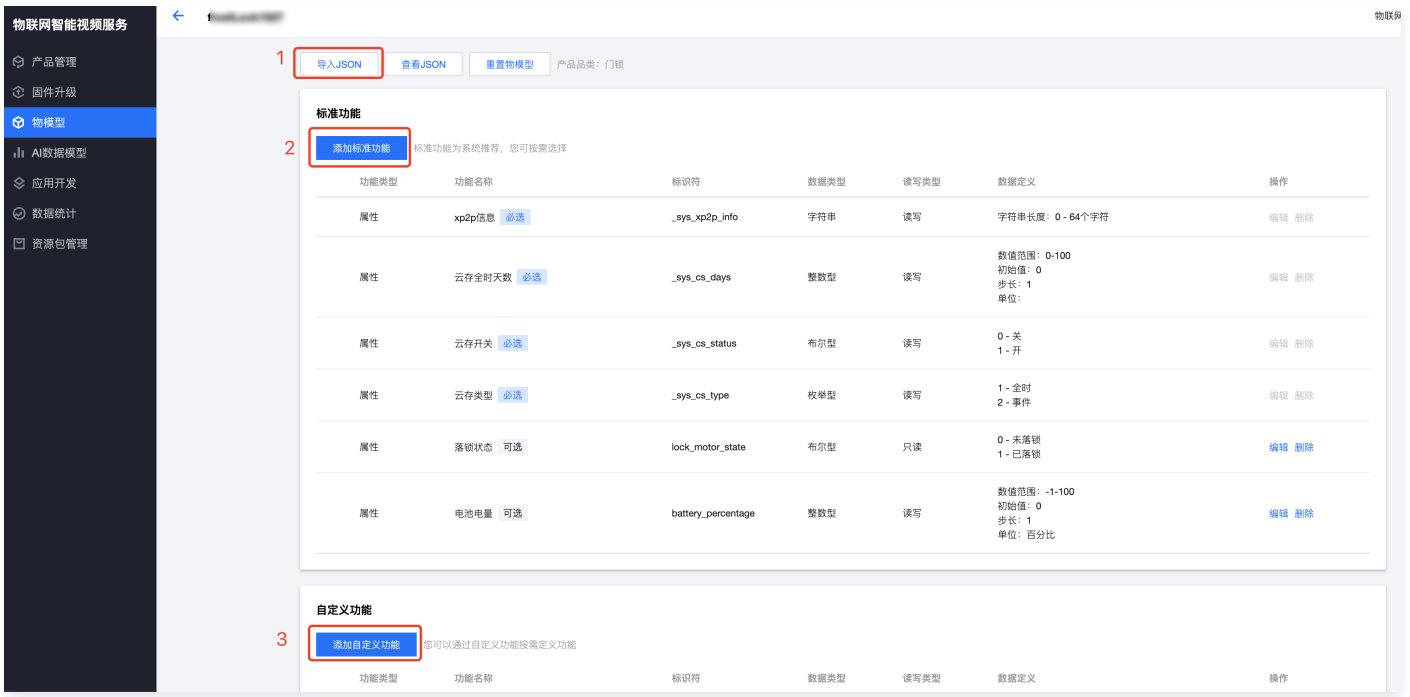
参数名称	参数标识符	数据类型	数据定义
唤醒状态	wakeup_state	布尔型	0 - 未唤醒 1 - 已成功唤醒

9.2. 设备 Wi-Fi 收到唤醒命令，唤醒主控。待主控启动，调用 IoT Video SDK，待初始化成功后，设备 Wi-Fi 上报**唤醒状态**为“唤醒状态”。

9.3. 小程序获取**唤醒状态**为“唤醒状态”时，出图显示门锁实况。

10. 相关物模型

所需的物模型需要在**控制台 > 物模型** 页面进行操作，主要包括需要导入的相关物模型和添加的必选标准功能的物模型，同时也支持添加自定义功能的物模型。



导入相关物模型：

```
{
  "version": "1.0",
  "profile": {
    "ProductId": "OSO00VC9U8",
    "CategoryId": "595"
  },
  "properties": [
    {
      "id": "sensitivity",
      "name": "灵敏度",
      "desc": "人体移动监测灵敏度",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "0": "低",
          "1": "中",
          "2": "高"
        }
      },
      "required": false
    },
    {
      "id": "battery_percentage",
      "name": "电池电量",
      "desc": "-1表示未获取到电量",
      "mode": "rw",
      "define": {
        "type": "int",
        "min": "-1",
        "max": "100",
        "start": "0",
        "step": "1",
        "unit": "百分比"
      },
      "required": false
```

```
    },
    {
      "id": "volume",
      "name": "音量",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "0": "静音",
          "1": "低",
          "2": "中",
          "3": "高"
        }
      },
      "required": false
    },
    {
      "id": "wakeup_state",
      "name": "唤醒状态",
      "desc": "设备主动上报是否处于唤醒状态",
      "mode": "r",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "休眠状态",
          "1": "唤醒状态"
        }
      },
      "required": false
    },
    {
      "id": "faces",
      "name": "面容",
      "desc": "",
      "mode": "rw",
      "define": {
        "arrayInfo": {
          "type": "struct",
          "specs": [
            {
              "id": "id",
              "name": "id",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "100"
              }
            },
            {
              "id": "face_name",
              "name": "face_name",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "20"
              }
            }
          ]
        },
        "type": "array"
```

```
      },
      "required": false
    },
    {
      "id": "cards",
      "name": "卡片",
      "desc": "",
      "mode": "rw",
      "define": {
        "arrayInfo": {
          "type": "struct",
          "specs": [
            {
              "id": "id",
              "name": "id",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "100"
              }
            },
            {
              "id": "card_name",
              "name": "card_name",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "20"
              }
            }
          ]
        },
        "type": "array"
      },
      "required": false
    },
    {
      "id": "passwords",
      "name": "密码",
      "desc": "",
      "mode": "rw",
      "define": {
        "arrayInfo": {
          "type": "struct",
          "specs": [
            {
              "id": "id",
              "name": "id",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "100"
              }
            },
            {
              "id": "password_name",
              "name": "password_name",
              "dataType": {
                "type": "string",
                "min": "0",
                "max": "20"
```

```
            }
          }
        ]
      },
      "type": "array"
    },
    "required": false
  },
  {
    "id": "fingerprints",
    "name": "指纹",
    "desc": "",
    "mode": "rw",
    "define": {
      "arrayInfo": {
        "type": "struct",
        "specs": [
          {
            "id": "id",
            "name": "id",
            "dataType": {
              "type": "string",
              "min": "0",
              "max": "100"
            }
          },
          {
            "id": "fingerprints_name",
            "name": "fingerprints_name",
            "dataType": {
              "type": "string",
              "min": "0",
              "max": "20"
            }
          }
        ]
      },
      "type": "array"
    },
    "required": false
  },
  {
    "id": "lock_motor_state",
    "name": "落锁状态",
    "desc": "",
    "mode": "r",
    "define": {
      "type": "bool",
      "mapping": {
        "0": "未落锁",
        "1": "已落锁"
      }
    },
    "required": false
  },
  {
    "id": "child_lock_state",
    "name": "童锁状态",
    "desc": "",
    "mode": "rw",
    "define": {
```

```
        "type": "bool",
        "mapping": {
          "0": "关闭",
          "1": "开启"
        }
      },
      "required": false
    },
    {
      "id": "disarming",
      "name": "布防模式",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关闭",
          "1": "开启"
        }
      },
      "required": false
    },
    {
      "id": "inner_lock_state",
      "name": "反锁状态",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关闭",
          "1": "开启"
        }
      },
      "required": false
    },
    {
      "id": "user_list",
      "name": "绑定终端用户列表",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "struct",
        "specs": [
          {
            "id": "owner",
            "name": "主人",
            "dataType": {
              "type": "string",
              "min": "0",
              "max": "2048"
            }
          },
          {
            "id": "shared_person",
            "name": "被分享者",
            "dataType": {
              "type": "array",
              "arrayInfo": {
                "type": "string",
                "min": "0",
```

```
              "max": "2048"
            }
          }
        }
      ]
    },
    "required": false
  },
  {
    "id": "lock_state",
    "name": "门锁状态",
    "desc": "",
    "mode": "rw",
    "define": {
      "type": "enum",
      "mapping": {
        "0": "未落锁",
        "1": "已落锁",
        "3": "门锁异常"
      }
    },
    "required": false
  },
  {
    "id": "_sys_xp2p_info",
    "name": "xp2p信息",
    "desc": "",
    "mode": "rw",
    "define": {
      "type": "string",
      "min": "0",
      "max": "64"
    },
    "required": false
  },
  {
    "id": "_sys_cs_days",
    "name": "云存全时天数",
    "desc": "",
    "mode": "rw",
    "define": {
      "type": "int",
      "min": "0",
      "max": "100",
      "start": "0",
      "step": "1",
      "unit": ""
    },
    "required": false
  },
  {
    "id": "_sys_cs_status",
    "name": "云存开关",
    "desc": "",
    "mode": "rw",
    "define": {
      "type": "bool",
      "mapping": {
        "0": "关",
        "1": "开"
      }
```

```
      },
      "required": false
    },
    {
      "id": "_sys_cs_type",
      "name": "云存类型",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "enum",
        "mapping": {
          "1": "全时",
          "2": "事件"
        }
      },
      "required": false
    },
    {
      "id": "_sys_ai_status",
      "name": "AI开关",
      "desc": "",
      "mode": "rw",
      "define": {
        "type": "bool",
        "mapping": {
          "0": "关",
          "1": "开"
        }
      },
      "required": false
    }
  ],
  "events": [],
  "actions": [
    {
      "id": "wake_up",
      "name": "唤醒门锁",
      "desc": "",
      "input": [],
      "output": [
        {
          "id": "wakeup_state",
          "name": "唤醒状态",
          "define": {
            "type": "bool",
            "mapping": {
              "0": "未唤醒",
              "1": "已成功唤醒"
            }
          }
        }
      ],
      "required": false
    },
    {
      "id": "unlock_remote",
      "name": "远程解锁",
      "desc": "",
      "input": [],
      "output": [
        {
```

```
          "id": "result",
          "name": "开锁成功状态",
          "define": {
            "type": "bool",
            "mapping": {
              "0": "开锁失败",
              "1": "开锁成功"
            }
          }
        }
      ],
      "required": false
    }
  ]
}
```

应用端 API
应用端 API 简介

最近更新时间：2024-12-25 18:07:22

简介

应用端 API 是开发平台为了满足智能家居场景，为用户开发自有品牌的小程序或 App 而提供的云端服务，用户无需实现用户管理、设备管理、设备定时、家庭管理等基础能力，可通过调用应用端 API 快速完成移动应用端的开发。

调用方式

1. **登录前所使用的 API URL 为：**

> **说明：**
> - 国内平台：https://iot.cloud.tencent.com/api/exploreropen/appapi
> - 海外平台：https://oversea.iotcloud.tencentiotcloud.com/api/exploreropen/appapi

其中公共参数有：Action，RequestId，AppKey，Signature，Timestamp，Nonce。
- Action 用于标识请求的方法名称。
- RequestId 用于标识一个唯一请求，推荐使用 uuid 作为参数值，定位问题时建议提供该参数值。
- AppKey 为应用的密钥。
- Signature 为本次请求的签名，具体计算方法见本文下方示例。
- Timestamp 为本次请求的 UNIX 秒级时间戳。
- Nonce 为随机正整数，用于和时间戳一起，防范 API 重放攻击。

2. **登录后所使用的 API URL 为：**

> **说明：**
> - 国内平台：https://iot.cloud.tencent.com/api/exploreropen/tokenapi
> - 海外平台：https://oversea.iotcloud.tencentiotcloud.com/api/exploreropen/tokenapi

其中公共参数有：Action，RequestId，AccessToken。
- Action 用于标识请求的方法名称。
- RequestId 用于标识一个唯一请求，推荐使用 uuid 作为参数值，定位问题时建议使用该参数值。
- AccessToken 用于标识一个已经登录的用户。

> **说明：**
> 调用应用端 API 接口前，需要先调用登录相关接口（应用端 API/用户管理），登录成功后得到 `AccessToken`。
> `AccessToken` 用于标识一个用户。当用户登录成功后，再携带 `AccessToken` 调用其他应用端 API。
> **登录前调用 URL 后缀为 `.../appapi`、登录后调用 URL 后缀为 `.../tokenapi`**

签名算法

获取应用 AppKey 和 AppSecret

如果用户不使用腾讯官方的“腾讯连连”小程序，用户也可通过平台开放能力开发自有品牌小程序。在创建应用的时候，平台会为用户生成小程序对应的安全凭证。安全凭证包括 AppKey 和 AppSecret。AppKey 是用于标识 API 调用者身份，AppSecret 是用于加密签名字符串和服务器端验证签名字符串的密钥。

> **注意：**
> **用户应严格保管其 AppSecret，避免泄露。**

具体获取步骤如下：

1.1 登录 物联网开发平台控制台，进入开发中心。

1.2 选择左侧菜单**应用开发 > 小程序开发**，新建小程序，具体新建步骤参见 应用开发。

1.3 创建小程序成功后，即可获取系统自动生成的 AppKey 与 AppSecret。

生成签名串

有了安全凭证 AppKey 和 AppSecret 后，就可以生成签名串了。下面给出了一个生成签名串的详细过程。
假设用户的 AppKey 和 AppSecret 分别是：

- AppKey： `ahPxdK****TGrejd`
- AppSecret： `NcbHqk****TCGbKnQH`

> **说明：**
> 本文仅为示例，请您根据自己实际的 `AppKey` 和 `AppSecret` 进行后续操作。

以通过手机号注册账号 `AppCreateCellphoneUser` 请求为例，当用户调用这一接口时，其请求参数可能如下：

参数名称	类型	描述	参考数值
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。	8b8d499bbba1ac28b6da21b4
Action	String	公共参数，调用的接口方法名称。	AppCreateCellphoneUser
AppKey	String	公共参数，应用 AppKey ，用于标识对应的 App。	ahPxdK****TGrejd
Signature	String	公共参数，请求的签名。	Szxai9Qs7****OXahbFbseZ+uE=
Timestamp	Int64	公共参数，当前的 UNIX 时间戳（秒级）。	1546315200
Nonce	Int	公共参数，随机正整数，与时间戳一起，用于 API 防重放。	71087795
CountryCode	String	国家区码。	86
PhoneNumber	String	手机号码。	13900000000
Password	String	密码。	password
VerificationCode	String	短信验证码。	123456

> **说明：**
> - 请求参数中的公共请求参数有：RequestId、Action、AppKey、Timestamp、Nonce、Signature。
> - AppCreateCellphoneUser 接口特有参数：CountryCode、PhoneNumber、Password、VerificationCode。

而参数 Signature（签名串）正是由上述参数共同生成的，具体步骤如下：

1. 对参数排序。
   对所有请求参数按参数名做字典序升序排列，所谓字典序升序排列，直观上就如同在字典中排列单词一样排序，按照字母表或数字表里递增顺序的排列次序，即先考虑第一个“字母”，在相同的情况下考虑第二个 “字母”，依此类推。您可以借助编程语言中的相关排序函数来实现这一功能，例如 PHP 中的 ksort 函数。上述示例参数的排序结果如下：

```
{
    Action=AppCreateCellphoneUser,
    AppKey=ahPxdK****TGrejd,
    CountryCode=86,
    Nonce=71087795,
    Password=My!P@ssword,
    PhoneNumber=13900000000,
    RequestId=8b8d499bbba1ac28b6da21b4,
    Timestamp=1546315200,
    VerificationCode=123456
}
```

使用其它程序设计语言开发时，可对上面示例中的参数进行排序，得到的结果一致即可。

2. 拼接请求字符串。
   将把上一步排序好的请求参数格式化成“参数名称” = “参数值”的形式，例如对 Action 参数，其参数名称为 "Action"，参数值为 "AppCreateCellphoneUser"，因此格式化后就为 Action=AppCreateCellphoneUser。
   - “参数值”为原始值而非 URL 编码后的值。
   - 若输入参数中的“键”包含下划线，则需要将其转换为 “.” 。对于“值”不需要额外操作。

   将格式化后的各个参数用"&"拼接在一起，最终生成的请求字符串为：

```
Action=AppCreateCellphoneUser&AppKey=ahPxdK****TGrejd&CountryCode=86&Nonce=71087795&Password=My!P@sswor
d
&PhoneNumber=13900000000&RequestId=8b8d499bbba1ac28b6da21b4&Timestamp=1546315200&VerificationCode=12345
6
```

3. 生成签名串。
   使用 HMAC-SHA1 算法对上一步中获得的签名原文字符串进行签名，然后将生成的签名串使用 Base64 进行编码，即可获得最终的签名串。
   具体代码如下，以 PHP 语言为例：

```
$secretKey = 'NcbHqk****TCGbKnQH';
$srcStr =
'Action=AppCreateCellphoneUser&AppKey=ahPxdK****TGrejd&CountryCode=86&Nonce=71087795&Password=My!P@sswo
rd&PhoneNumber=13900000000&RequestId=8b8d499bbba1ac28b6da21b4&Timestamp=1546315200&VerificationCode=123
456';
$signStr = base64_encode(hash_hmac('sha1', $srcStr, $secretKey, true));
echo $signStr
```

   最终得到的签名串为：

```
Szxai9Qs7O3lBoOXahbFbseZ+uE=
```

   使用其它程序设计语言开发时，可用上面示例中的原文进行签名验证，得到的签名串与例子中的一致即可。

常见问题

对于参数为字符串且为空的，应将其从签名串中省略，不需要将其参与签名。

调试工具

使用应用端 API 调试工具，快速调试应用端 API。

用户管理
微信号注册登录

最近更新时间：2023-07-13 17:41:24

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppGetTokenByWeiXin）用于微信用户注册登录，获取开发平台的用户访问 Token，首次调用时，自动为该微信号注册对应账号。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppGetTokenByWeiXin。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的小程序或 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 UNIX 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
WxOpenID	String	是	微信用户的 OpenID 或 UnionID。
NickName	String	是	昵称。
Avatar	String	是	头像。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
ExpireAt	Int64	截止时间，UINX 秒级时间戳。
Token	String	开发平台返回的 AccessToken，通过该 Token 进行登录后的接口请求。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
"WxOpenID": "0123456789",
"NickName": "tests",
"Avatar": "http://example.com/path/to/avatar.jpg",
"Signature": "IY0yHX7zWCZKZ97xDonQoy16W4U=",
"Timestamp": 1552621825,
"Nonce": 2,
"Action": "AppGetTokenByWeiXin",
"AppKey": "ahPxdK*****TGrejd",
"RequestId": "rest-client"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "ExpireAt": 1556076201,
      "Token": "d2******************8514"
    },
    "RequestId": "rest-client"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.ErrorUserNotExists",
      "Message": "账号不存在"
    },
    "RequestId": "rest-client"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
ErrorRequiredParamNotFound	必选参数缺失。
InvalidAction	Action 非法。
InvalidParameterValue	参数异常。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.NickNameLengthInvalid	昵称长度非法

手机号注册用户

最近更新时间：2023-07-13 17:41:24

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppCreateCellphoneUser）提供手机号码方式的用户注册。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppCreateCellphoneUser。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 UNIX 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
CountryCode	String	是	国家号。
PhoneNumber	String	是	手机号。
Password	String	是	登录密码。
VerificationCode	String	是	短信验证码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
UserID	String	用户 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
  "CountryCode": "86",
  "PhoneNumber": "13900000000",
  "Password": "MyPassword",
  "VerificationCode": "123456",
  "Signature": "CKu55Y3ZD6RuxpjPySM6U99imbs=",
  "Timestamp": 1546315200,
  "Nonce": 71087795,
  "Action": "AppCreateCellphoneUser",
  "AppKey": "ahPxdK*****TGrejd",
  "RequestId": "8b8d499bbba1ac28b6da21b4"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": { "UserID": "1234567890" },
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.PhoneNumberInvalid",
      "Message": "号码错误"
    },
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
ErrorRequiredParamNotFound	必选参数缺失。
InvalidAction	Action 非法。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.CheckVerifyCodeFailed	验证码错误。
InvalidParameterValue.PhoneNumberUsed	电话号码已注册。
InvalidParameterValue.PhoneNumberInvalid	号码错误。

邮箱账号注册用户

最近更新时间：2023-07-13 17:41:24

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppCreateEmailUser）提供邮箱账号方式注册用户。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppCreateEmailUser。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时需要提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 UNIX 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
Email	String	是	用户的邮箱地址。
Password	String	是	登录密码。
VerificationCode	String	是	发送到邮箱的验证码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
UserID	String	用户 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
  "Email":"test@example.com",
  "Password":"My!P@ssword",
  "VerificationCode": "123456",
  "Signature":"CKu55Y3ZD6RuxpjPySM6U99imbs=",
  "Timestamp": 1546315200,
  "Nonce": 71087795,
  "Action":"AppCreateEmailUser",
  "AppKey":"ahPxdK*****TGrejd",
  "RequestId":"8b8d499bbba1ac28b6da21b4"
}
```

输出示例：成功

```
{
    "Response": {
```

```
        "Data": {
            "UserID": "1"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.EmailUsed",
            "Message": "邮箱已注册"
        },
        "RequestId": "rest-client"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.EmailUsed	邮箱已注册。
InvalidParameterValue.CheckVerifyCodeFailed	验证码错误。

随机发送邮箱验证码

最近更新时间：2024-10-11 17:41:24

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppSendEmailVerificationCode）用于邮箱注册、绑定、重置密码和登录时，发送验证码。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppSendEmailVerificationCode
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性
Timestamp	Int64	是	公共参数，请求的 Unix 时间戳（秒级）
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击
Type	String	是	验证码类型。 • register：注册。 • resetpass：重置密码。 • login：登录。
Email	String	是	邮箱。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
UserId	String	用户 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
            "Type": "register",
            "Email": "someone@example.com",
            "Signature": "8tEbOa9wk0X3wTBn0BBoFWdgMjo=",
            "Timestamp": 1552621825,
            "Nonce": 2,
            "Action": "AppSendEmailVerificationCode",
            "AppKey": "ahPxd********rejd",
            "RequestId": "f92406b3-5a9a-****-bc43-45e3d794bb68"
}
```

输出示例：成功

```
  {
    "Response": {
```

```
    "Data": "OK",
    "RequestId": "f92406b3-5a9a-****-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.EmailInvalid",
      "Message": "邮箱错误"
    },
    "RequestId": "f92406b3-5a9a-****-bc43-45e3d794bb68"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.EmailInvalid	邮箱错误。
InvalidParameterValue.SendVerifyCodeTooQuick	验证码发送太频繁。

随机发送手机短信

最近更新时间：2023-07-13 17:41:24

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppSendVerificationCode）用于手机号注册、绑定、重置密码时，发送验证码。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppSendVerificationCode。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时需要提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 UNIX 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
Type	String	是	验证码类型：register 注册，resetpass 重置密码。
CountryCode	String	是	国家代码。
PhoneNumber	String	是	电话号码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
UserID	String	用户 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
 "Type": "register",
 "CountryCode": "86",
 "PhoneNumber": "1390000000",
 "Signature": "8tEbOa9wk0X3wTBn0BBoFWdgMjo=",
 "Timestamp": 1552621825,
 "Nonce": 2,
 "Action": "AppSendVerificationCode",
 "AppKey": "ahPxdK*****TGrejd",
 "RequestId": "rest-client"
}
```

输出示例：成功

```
  {
    "Response": {
```

```
    "Data": "OK",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.PhoneNumberInvalid",
      "Message": "号码错误"
    },
    "RequestId": "rest-client"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.PhoneNumberInvalid	号码错误。
InvalidParameterValue.SendVerifyCodeTooQuick	验证码发送太频繁。

手机号或邮箱账号登录

最近更新时间：2023-07-13 17:41:24

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppGetToken）用于手机号码、邮箱账号登录，获取用户访问 Token。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppGetToken。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 Unix 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
Type	String	否	账号类型。 • phone：手机号。 • email：邮箱。
CountryCode	String	否	国家号。
PhoneNumber	String	否	手机号。
Email	String	否	邮箱。
Password	String	是	登录密码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	响应结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
  "Email": "test@example",
  "Password": "123456",
  "Type": "email",
  "Signature": "w2kTSOU7IKc2aJ+UplV34VHp1RI=",
  "Timestamp": 1552621825,
  "Nonce": 2,
  "Action": "AppGetToken",
  "AppKey": "ahPxdK*****TGrejd",
  "RequestId": "rest-client"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "ExpireAt": 1556076201,
            "Token": "d2******************8514"
        },
        "RequestId": "rest-client"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InternalError",
            "Message": "InvalidParameterValue.UserLoginFailed|用户登录失败"
        },
        "RequestId": "rest-client"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
ErrorRequiredParamNotFound	必选参数缺失。
InvalidAction	Action 非法。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.UserLoginFailed	用户登录失败。
InvalidParameterValue.UserLoginForbidden	账号被锁定。

使用手机号重置密码

最近更新时间：2024-10-11 17:41:24

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppResetPasswordByCellphone）用于使用手机号重置密码。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppResetPasswordByCellphone。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 Unix 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
CountryCode	String	是	手机号国家码。
PhoneNumber	String	是	手机号码。
VerificationCode	String	是	验证码。
Password	String	是	新设置的密码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
    "CountryCode": "86",
    "PhoneNumber": "139000000000",
    "VerificationCode": "123456",
    "Password": "password",
    "Signature": "8tEbOa9wk0X3wTBn0BBoFWdgMjo=",
    "Timestamp": 1552621825,
    "Nonce": 2,
    "Action": "AppResetPasswordByCellphone",
    "AppKey": "ahPxd********rejd",
    "RequestId": "f92406b3-5a9a-****-****-45e3d794bb68"
}
```

输出示例: 成功

```
{
    "Response": {
        "Data": "OK",
        "RequestId": "f92406b3-5a9a-****-****-45e3d794bb68"
    }
}
```

输出示例: 失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.PhoneNumberInvalid",
            "Message": "号码错误"
        },
        "RequestId": "rest-client"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.PhoneNumberInvalid	号码错误。
InvalidParameterValue.SendVerifyCodeTooQuick	验证码发送太频繁

使用邮箱重置密码

最近更新时间：2024-10-11 17:41:25

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppResetPasswordByEmail）用于使用邮箱重置密码。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppResetPasswordByEmail。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 Unix 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。
Email	String	是	邮箱。
VerificationCode	String	是	验证码。
Password	String	是	新设置的密码。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/appapi HTTP/1.1
content-type: application/json
{
            "Email": "someone@example.com",
            "Password": "password",
            "Signature": "8tEbOa9wk0X3wTBn0BBoFWdgMjo=",
            "Timestamp": 1552621825,
            "Nonce": 2,
            "Action": "AppResetPasswordByEmail",
            "AppKey": "ahPxd********rejd",
            "RequestId": "f92406b3-5a9a-****-bc43-45e3d794bb68"
}
```

输出示例：成功

```
 {
    "Response": {
        "Data": "OK",
```

```
        "RequestId": "f92406b3-5a9a-****-bc43-45e3d794bb68"
    }
}
```

输出示例：失败

```
{
  "Response": {
      "Error": {
          "Code": "InvalidParameterValue.ErrorUserNotExists",
          "Message": "账号未创建或是已删除"
      },
      "RequestId": "rest-client"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.ErrorUserNotExists	账号未创建或是已删除。
InvalidParameterValue.SendVerifyCodeTooQuick	验证码发送太频繁。

用户注销

最近更新时间：2023-07-13 17:41:25

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppLogoutUser）用于用户退出登录态。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppLogoutUser。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	响应结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
"AccessToken": "bf*****************d098",
"Action": "AppLogoutUser",
"RequestId": "rest-client"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": "OK",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidAccessToken",
      "Message": "Token无效"
    },
    "RequestId": ""
  }
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
ErrorRequiredParamNotFound	必选参数缺失。
InvalidAction	Action 非法。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.InvalidAccessToken	Token 无效

修改用户信息

最近更新时间：2023-07-17 18:02:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppUpdateUser）用于修改用户信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppUpdateUser。
NickName	String	否	昵称。
Avatar	String	否	图标。
CountryCode	String	否	手机号国家码。
PhoneNumber	String	否	手机号码。
Email	String	否	邮箱。
VerificationCode	String	否	手机或邮箱验证码（绑定手机或邮箱时，该参数必填）。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "NickName": "tests",
  "Avatar": "http://example.com/path/to/avatar.jpg",
  "AccessToken": "bf******************d098",
  "Action": "AppUpdateUser",
  "RequestId": "rest-client"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": "OK",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidAccessToken",
      "Message": "Token无效"
    },
    "RequestId": ""
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NickNameLengthInvalid	昵称长度非法。
InvalidParameterValue.PhoneNumberInvalid	号码错误。
InvalidParameterValue.PhoneNumberUsed	号码已注册。
InvalidParameterValue.EmailInvalid	邮箱错误。
InvalidParameterValue.EmailUsed	邮箱已注册。
InvalidParameterValue.CheckVerifyCodeFailed	验证码验证不通过

获取用户信息设置

最近更新时间：2023-07-17 18:02:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetUserSetting）用于获取用户设置的配置信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetUserSetting。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
UserSetting	UserSetting	用户设置信息。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
"AccessToken": "bf*****************d098",
"Action": "AppGetUserSetting",
"RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "UserSetting": {
      "EnableWechatPush": 1,
      "EnableDeviceMessagePush": 0,
      "EnableFamilyMessagePush": 0,
      "EnableNotifyMessagePush": 0
    }
  }
}
```

输出示例：失败

```
  {
    "Response": {
      "Error": {
```

```
        "Code": "InvalidParameterValue.InvalidAccessToken",
        "Message": "Token无效"
      },
      "RequestId": "1555507435215"
    }
  }
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改用户信息设置

最近更新时间：2023-07-17 18:02:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppUpdateUserSetting）用于修改用户配置信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppUpdateUserSetting。
EnableWechatPush	Int	是	• 0：不允许推送微信模板消息。 • 1：允许推送微信模板消息。
EnableDeviceMessagePush	Int	是	• 0：不允许推送告警信息。 • 1：允许推送告警信息。
EnableFamilyMessagePush	Int	是	• 0：不允许推送家庭信息。 • 1：允许推送家庭信息。
EnableNotifyMessagePush	Int	是	• 0：不允许推送通知信息。 • 1：允许推送通知信息。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
 "AccessToken": "bf******************d098",
 "Action": "AppUpdateUserSetting",
 "EnableWechatPush": 1,
 "EnableDeviceMessagePush": 1,
 "EnableFamilyMessagePush": 1,
 "EnableNotifyMessagePush": 1,
 "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
```

```
    }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidAccessToken",
      "Message": "Token无效"
    },
    "RequestId": "5cc7ac-f4c7-4bc5-91-dde"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数非法。
InvalidParameterValue.InvalidJSON	请求格式不是 JSON。
InvalidParameterValue.InvalidAccessToken	Token 无效。

配网管理
生成 Wi-Fi 设备配网 Token

最近更新时间：2024-10-11 18:02:11

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（AppCreateDeviceBindToken）用于生成 Wi-Fi 配网任务的随机 Token。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppCreateDeviceBindToken。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
Token	String	生成的配网 Token。

4. 示例

示例1 生成 Wi-Fi 设备配网 Token

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppCreateDeviceBindToken",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "Token": "d7*************1b3b",
            "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
        }
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
```

```
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

查询配网 Token 状态

最近更新时间：2024-09-30 17:54:51

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppGetDeviceBindTokenState）用于查询配网 Token 的当前状态。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppGetDeviceBindTokenState。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
Token	String	是	由 AppCreateDeviceBindToken 接口生成的配网 Token。

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
State	Uint	Token 状态。 • 1：初始生产。 • 2：可使用状态。

4. 示例

示例1 查询配网 Token 状态

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppCreateDeviceBindToken",
  "AccessToken":"bf*****************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "Token": "d7a1**********a861b3b"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "State": 1,
            "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
        }
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.TokenNotExist",
            "Message": "Token不存在"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenNotExist	Token 不存在。
InvalidParameterValue.TokenIsExpire	Token 已过期。
InvalidParameterValue.ReadTokenInfoError	Token 读取错误。

用户绑定 Wi-Fi 设备

最近更新时间：2024-10-11 18:02:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppTokenBindDeviceFamily）用于小程序或 App 用户绑定 Wi-Fi 类设备。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppTokenBindDeviceFamily。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
Token	String	是	配网 Token。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
FamilyId	String	是	家庭 ID。
RoomId	String	是	房间 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。

4. 示例

用户绑定 Wi-Fi 设备

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppTokenBindDeviceFamily",
  "AccessToken":"tesetoken",
  "Token": "bf*****************d098",
  "ProductId": "*******",
  "DeviceName": "*******",
  "FamilyId": "f************cc1",
  "RoomId": "0"
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
}
```

输出示例 成功

```
{
    "Response": {
        "Data": {
            "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
```

```
        }
    }
}
```

输出示例 失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenNotExist	Token 不存在。
InvalidParameterValue.TokenIsExpire	Token 已过期。
InvalidParameterValue.ReadTokenInfoError	Token 读错误。
InvalidParameterValue.TokenNotBind	Token 未绑定，即在上一步接口 AppGetDeviceBindTokenState 查询返回状态为初始状态，非可使用状态。
InvalidParameterValue.FamilyDeviceCountReadError	读设备数量错误。
UnauthorizedOperation	无操作权限。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.NoPermissionToFamily	操作该家庭无权限。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioFamilyNotExist	家庭未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
LimitExceeded	数量限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
InternalError	内部错误。
InternalError.InternalServerException	内部错误。
InternalError.InternalServerExceptionDB	内部 DB 错误。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定，请勿重复绑定

设备管理
获取产品信息

最近更新时间：2024-10-11 18:02:12

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetProducts）获取产品信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetProducts。
ProductIds	Array of String	是	产品 ID 数组。
DeviceIds	Array of String	否	设备 ID 数组。

3. 输出参数

名称	类型	描述
Products	Array	返回产品信息。
RequestId	String	公共参数，唯一请求 ID，与入参相同。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec",
    "RequestId": "27HPvi1ylv",
    "Action": "AppGetProducts",
    "ProductIds":["FVYYYEL4ON"],
    "DeviceIds":["QDA1PZLBNB/aaa", "QDA1PZLBNB/bbb"]
}
```

输出示例：成功

```
{
  "Response":{
      "Products":[
          {
              "ProductId":"FVYYYEL4ON",
              "Name":"c1",
              "Description":"yes",
              "State":"",
              "DataTemplate":"{"version":"1.0","profile":
{"ProductId":"FVYYYEL4ON","CategoryId":"1"},"properties":[{"id":"mac","name":"mac地址","desc":"wifi路由器mac
```

```
地址","required":true,"mode":"r","define":{"type":"string","min":"0","max":"30"}},
{"id":"signal","name":"wifi强度","desc":"wifi信号强度值","mode":"r","define":
{"type":"int","unit":"dbm","step":"1","min":"-180","max":"180","start":"1"}},{"id":"ssid","name":"wifi热点
名","desc":"设备连接的wifi热点名","mode":"r","define":{"type":"string","min":"0","max":"64"}},
{"id":"severip","name":"wifi路由器接入的ip","desc":"wifi路由器接入的ip","mode":"r","define":
{"type":"string","min":"0","max":"64"}},{"id":"nmacs","name":"其他热点mac地址","desc":"可接受到的其他热点mac地
址","mode":"r","define":{"type":"string","min":"0","max":"64"}}],"events":[],"actions":[]}",
                "AppTemplate":"",
                "NetType":"wifi",
                "CategoryId":1,
                "ProductType":0,
                "UpdateTime":1600159203
            },
            {
                "ProductId":"QDA1PZLBNB",
                "Name":"light5勿删",
                "Description":"",
                "State":"",
                "DataTemplate":"{"version":"1.0","profile":
{"ProductId":"QDA1PZLBNB","CategoryId":"141"},"properties":[{"id":"power_switch","name":"电灯开
关","desc":"控制电灯开灭","required":true,"mode":"rw","define":{"type":"bool","mapping":
{"0":"关","1":"开"}}},{"id":"color","name":"颜色","desc":"灯光颜色","mode":"rw","define":
{"type":"enum","mapping":{"0":"Red","1":"Green","2":"Blue"}}},{"id":"brightness","name":"亮度","desc":"灯光
亮度","mode":"rw","define":{"type":"int","unit":"%","step":"1","min":"0","max":"100","start":"1"}},
{"id":"name","name":"灯位置名称","desc":"灯位置名称：书房、客厅等","mode":"rw","required":false,"define":
{"type":"string","min":"0","max":"64"}},{"id":"Temperature","name":"温度","desc":"","mode":"rw","define":
{"type":"float","min":"0","max":"100","start":"0","step":"1","unit":""},"required":false},
{"id":"temp_max","name":"最高温度","desc":"","mode":"rw","define":
{"type":"int","min":"0","max":"100","start":"0","step":"1","unit":""},"required":false},
{"id":"LBS_BS","name":"LBS_BS","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false},
{"id":"FOO","name":"FOO","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false},
{"id":"GPS_Ext","name":"GPS_Ext","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false}],"events":
[{"id":"status_report","name":"DeviceStatus","desc":"Report the device
status","type":"info","required":false,"params":[{"id":"status","name":"running_state","desc":"Report
current device running state","define":{"type":"bool","mapping":{"0":"normal","1":"fault"}}},
{"id":"message","name":"Message","desc":"Some extra message","define":
{"type":"string","min":"0","max":"64"}}]},{"id":"low_voltage","name":"LowVoltage","desc":"Alert for device
voltage is low","type":"alert","required":false,"params":[{"id":"voltage","name":"Voltage","desc":"Current
voltage","define":{"type":"float","unit":"V","step":"1","min":"0.0","max":"24.0","start":"1"}}]},
{"id":"hardware_fault","name":"Hardware_fault","desc":"Report hardware
fault","type":"fault","required":false,"params":[{"id":"name","name":"Name","desc":"Name like: memory,tf
card, censors ...","define":{"type":"string","min":"0","max":"64"}},
{"id":"error_code","name":"Error_Code","desc":"Error code for fault","define":
{"type":"int","unit":"","step":"1","min":"0","max":"2000","start":"1"}}]}],"actions":[]}",
                "AppTemplate":"",
                "NetType":"else",
                "CategoryId":141,
                "ProductType":0,
                "UpdateTime":1600265188
            },
            {
                "ProductId":"QDA1PZLBNB",
                "Name":"light5勿删",
                "Description":"",
                "State":"",
                "DataTemplate":"{"version":"1.0","profile":
{"ProductId":"QDA1PZLBNB","CategoryId":"141"},"properties":[{"id":"power_switch","name":"电灯开
关","desc":"控制电灯开灭","required":true,"mode":"rw","define":{"type":"bool","mapping":
```

{"0":"关","1":"开"}}},{"id":"color","name":"颜色","desc":"灯光颜色","mode":"rw","define":
{"type":"enum","mapping":{"0":"Red","1":"Green","2":"Blue"}}},{"id":"brightness","name":"亮度","desc":"灯光
亮度","mode":"rw","define":{"type":"int","unit":"%","step":"1","min":"0","max":"100","start":"1"}},
{"id":"name","name":"灯位置名称","desc":"灯位置名称：书房、客厅等","mode":"rw","required":false,"define":
{"type":"string","min":"0","max":"64"}},{"id":"Temperature","name":"温度","desc":"","mode":"rw","define":
{"type":"float","min":"0","max":"100","start":"0","step":"1","unit":""},"required":false},
{"id":"temp_max","name":"最高温度","desc":"","mode":"rw","define":
{"type":"int","min":"0","max":"100","start":"0","step":"1","unit":""},"required":false},
{"id":"LBS_BS","name":"LBS_BS","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false},
{"id":"FOO","name":"FOO","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false},
{"id":"GPS_Ext","name":"GPS_Ext","desc":"","mode":"rw","define":
{"type":"string","min":"0","max":"2048"},"required":false}],"events":
[{"id":"status_report","name":"DeviceStatus","desc":"Report the device
status","type":"info","required":false,"params":[{"id":"status","name":"running_state","desc":"Report
current device running state","define":{"type":"bool","mapping":{"0":"normal","1":"fault"}}},
{"id":"message","name":"Message","desc":"Some extra message","define":
{"type":"string","min":"0","max":"64"}}]},{"id":"low_voltage","name":"LowVoltage","desc":"Alert for device
voltage is low","type":"alert","required":false,"params":[{"id":"voltage","name":"Voltage","desc":"Current
voltage","define":{"type":"float","unit":"V","step":"1","min":"0.0","max":"24.0","start":"1"}}]},
{"id":"hardware_fault","name":"Hardware_fault","desc":"Report hardware
fault","type":"fault","required":false,"params":[{"id":"name","name":"Name","desc":"Name like: memory,tf
card, censors ...","define":{"type":"string","min":"0","max":"64"}},
{"id":"error_code","name":"Error_Code","desc":"Error code for fault","define":
{"type":"int","unit":"","step":"1","min":"0","max":"2000","start":"1"}}]}],"actions":[]}",
            "AppTemplate":"",
            "NetType":"else",
            "CategoryId":141,
            "ProductType":0,
            "UpdateTime":1600265188
        }
    ],
    "RequestId":"27HPvi1ylv"
  }
}

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

用户删除设备

最近更新时间：2023-07-17 18:02:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteDeviceInFamily）为用户提供删除设备的功能，删除后用户需要重新配网进行绑定。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteDeviceInFamily。
FamilyId	String	是	家庭 ID，成功创建家庭后，返回的 FamilyId。
RoomId	String	是	房间 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "42a96dbf-bdca-48f0-8f44-e36146d44e17",
    "Action": "AppDeleteDeviceInFamily",
    "AccessToken":"bf******************d098",
    "FamilyId": "1",
    "ProductId": "R32ONVL0EU",
    "DeviceName": "df2eSJyY"
}
```

输出示例 成功

```
{
    "Response": {
        "RequestId": "42a96dbf-bdca-48f0-8f44-e36146d44e17"
    }
}
```

5. 错误码

错误码	描述

InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取用户绑定设备列表

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFamilyDeviceList）用于获取用户已绑定设备列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFamilyDeviceList。
FamilyId	String	是	家庭 ID。
RoomId	String	是	房间 ID。
Offset	Int	否	消息偏移量。
Limit	Int	否	最大返回消息条数，最大值为50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Total	Int	设备总数。
DeviceList	Array	设备列表，请参见 DeviceList 。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetFamilyDeviceList",
    "FamilyId": "e0e23a2b33a24652b6**********a1cf",
    "RoomId": "5",  //如果要获取家庭所有设备，则该字段为"", "0" 则为默认房间
    "Offset": 0,
    "Limit": 2,
    "AccessToken": "bf*****************d098"
}
```

输出实例

```
{
    "Response": {
        "DeviceList": [{
            "ProductId": "R32*****EU",
            "DeviceName": "df2eSJyY",
```

```
            "DeviceId": "R32****0EU/df****yY",
            "AliasName": "df2eSJyY",
            "UserID": "1",
            "RoomId": "r_4b27c753ef774d458e**********86ad",
            "IconUrl": "",
            "CreateTime": 1574664969,
            "UpdateTime": 1574668779
        }],
        "RequestId": "req_1",
        "Total": 1
    }
}
```


5. 错误码

错误码	描述
InternalError	内部错误
InvalidParameterValue	参数取值错误
InvalidParameterValue.InvalidAccessToken	Token 无效

获取设备当前状态

最近更新时间：2023-07-17 18:02:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceStatuses）用于查询设备状态。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceStatuses。
DeviceIds	Array of String	是	设备 ID 数组。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Devices	Array of Devices	返回设备。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "892497d8-a733-480b-91d5-5559c6b49551",
    "Action": "AppGetDeviceStatuses",
    "DeviceIds": ["HY4DHFM5P6/81386276"],
    "AccessToken": "6bf4003d7f********400f3186f"
}
```

输出示例：成功

```
{
    "Response": {
        "DeviceStatuses": [{
            "DeviceId": "22F9Y6II7O/light1",
            "ProductId": "22F9Y6II7O",
            "DeviceName": "light1",
            "Online": 0 //0 在线；1：离线
        }],
        "RequestId": "892497d8-a733-480b-91d5-5559c6b49551"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改设备名称

最近更新时间：2023-07-17 18:02:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppUpdateDeviceInFamily）用于修改设备名称。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppUpdateDeviceInFamily。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
AliasName	String	是	设备别名。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
DeviceId	String	设备 ID。
ProductId	String	产品 ID。
DeviceName	String	设备名称。
AliasName	String	设备别名。
IconUrl	String	图标 URL。
FamilyId	String	家庭 ID。
RoomId	String	房间 ID。
CreateTime	Int64	创建时间。
UpdateTime	Int64	更新时间。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "641150a3-4238-430d-888b-c01b11298351",
    "Action": "AppUpdateDeviceInFamily",
    "AccessToken":"xxxv2",
    "ProductId": "VZS14OZSN5",
    "DeviceName": "~virtualDev",
    "AliasName": "12345"
```

```
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "DeviceId": "R32ONVL0EU/df2eSJyY",
            "ProductId": "R32ONVL0EU",
            "DeviceName": "df2eSJyY",
            "AliasName": "12345",
            "IconUrl": "",
            "FamilyId": "",
            "RoomId": 0,
            "CreateTime": 1574859101,
            "UpdateTime": 1574859144
        },
        "RequestId": "641150a3-4238-430d-888b-c01b11298351"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取设备详情

最近更新时间：2023-07-17 18:26:01

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceInFamily）用于查询设备详情。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceInFamily。
ProductId	String	是	产品 ID。
FamilyId	String	是	家庭 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
DeviceId	String	设备 ID，是产品 ID/设备名称组合。
ProductId	String	产品 ID。
DeviceName	String	设备名称。
AliasName	String	设备别名。
IconUrl	String	图标 URL。
FamilyId	String	家庭 ID。
RoomId	String	房间 ID。
CreateTime	Int64	创建时间，UNIX 秒级时间戳。
UpdateTime	Int64	最后一次更新时间，UNIX 秒级时间戳。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
        "RequestId": "082b70e-3d05-4179-9c94-6c70082e4a7",
        "Action": "AppGetDeviceInFamily",
        "ProductId":"R32ONVL0EU",
        "FamilyId":"xxx",
        "DeviceName":"df2eSJyY",
        "AccessToken": "xxxv2"
```

```
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "DeviceId": "R32ONVL0EU/df2eSJyY",
      "ProductId": "R32ONVL0EU",
      "DeviceName": "df2eSJyY",
      "AliasName": "12345",
      "IconUrl": "",
      "FamilyId": "",
      "RoomId": "",
      "DeviceType": 0, // 0 普通  1 网关设备  2 子设备
      "CreateTime": 1574931773,
      "UpdateTime": 1574931945
    },
    "RequestId": "082b70e-3d05-4179-9c94-6c70082e4a7"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioMemberNotExist	家庭成员未创建或是已删除。

获取设备扩展信息

最近更新时间：2024-09-29 18:26:01

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceExtInfo）用于获取设备的扩展信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceExtInfo。
DeviceId	String	是	设备 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
ManufacturerName	String	厂家名称。
ProductModel	String	产品型号。
Mac	String	设备上报基础信息 中的 MAC 地址。
FwVer	String	设备上报基础信息 中的固件版本。
IP	String	设备的 IP 地址。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "Action": "AppGetDeviceExtInfo",
  "DeviceId": "PR******ID/de***01",
  "AccessToken": "871d2d42****************692afae0"
}
```

输出示例：成功

```
{
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "ManufacturerName": "厂家名称",
  "ProductModel": "产品型号",
  "Mac": "AABBCCDDEEFF",
```

```
  "FwVer": "1.2.0",
  "IP": ""
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

设备更换房间

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyFamilyDeviceRoom）设备更换房间。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyFamilyDeviceRoom。
FamilyId	String	是	家庭 ID。
RoomId	String	是	房间 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json

{
    "RequestId": "8c012671-3934-4dc2-b07c-82c5ba3766f2",
    "Action": "AppModifyFamilyDeviceRoom",
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec",
    "FamilyId": "1",
    "RoomId": "r_4b27c753ef774d458e**********86ad",
    "ProductId": "R32****0EU",
    "DeviceName": "df2eSJyY"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "8c012671-3934-4dc2-b07c-82*****766f2"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

添加子设备到用户绑定列表

最近更新时间：2024-10-11 16:01:44

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppBindSubDeviceInFamily）用于将已经绑定到网关设备的子设备绑定到家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppBindSubDeviceInFamily。
GatewayProductId	String	是	网关产品 ID。
GatewayDeviceNam e	String	是	网关设备名称。
ProductId	String	是	子产品 ID。
DeviceName	String	是	子设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
AppDeviceInfo	Object of AppDeviceInfo	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "keyi20191119-012345",
  "Action": "AppBindSubDeviceInFamily",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "ProductId":"LAEG4YJE1A",
  "DeviceName":"subdev2",
  "GatewayProductId":"NJ27OVLZT4",
  "GatewayDeviceName":"gwdev"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
```

```
            "AppDeviceInfo": {
                "DeviceId": "R32ONVL0EU/df2eSJyY",
                "ProductId": "R32ONVL0EU",
                "DeviceName": "df2eSJyY",
                "AliasName": "",
                "IconUrl": "",
                "FamilyId": "1",
                "RoomId": "0",
                "CreateTime": 1574152773,
                "UpdateTime": 1574152773
            }
        },
        "RequestId": "keyi20191119-012345"
    }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取指定网关设备的子设备列表

最近更新时间：2024-10-11 18:26:01

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetGatewayBindDeviceList）用于获取指定网关设备的子设备列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetGatewayBindDeviceList。
ProductId	String	是	子设备的 ProductId。
GatewayProductId	String	是	网关设备的 ProductId。
GatewayDeviceName	String	是	网关设备的 DeviceName。
Offset	Int	是	消息偏移量。
Limit	Int	是	最大返回消息条数，最大值为50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Total	String	总数量。
DeviceList	Array of DeviceList	设备列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetGatewayBindDeviceList",
    "ProductId":"LAEG4YJE1A",  //子设备的productId
    "GatewayProductId":"NJ27OVLZT4",  //网关设备的productId
    "GatewayDeviceName":"gwdev",   //网关设备的deviceName
    "Offset":0,
    "Limit":2,
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例

```
{
    "Response": {
```

```
            "DeviceList": [{
                "ProductId": "LAEG4YJE1A",
                "DeviceName": "subdev2",     //子设备的deviceName
                "DeviceId": "LAEG4YJE1A/subdev2", //子设备的deviceId
                "AliasName": "",
                "IconUrl": "",
                "BindStatus": 0    //未绑定到家庭
            }, {
                "ProductId": "LAEG4YJE1A",
                "DeviceName": "subdev1",
                "DeviceId": "LAEG4YJE1A/subdev1",
                "AliasName": "",
                "IconUrl": "",
                "BindStatus": 1  //已经绑定到家庭
            }],
            "RequestId": "req_1",
            "Total": 2
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalRPCError	调用超时。
InternalError.InternalRPCError	调用返回失败。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.NotSupportVirtualDevice	不支持虚拟设备。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioMemberNotExist	家庭成员未创建或是已删除。

获取已绑定到家庭下的指定网关的子设备列表

最近更新时间：2024-10-11 18:26:01

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFamilySubDeviceList）用于获取家庭下网关的子设备列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFamilySubDeviceList。
GatewayProductId	String	是	网关设备产品 ID。
GatewayDeviceName	String	是	网关设备 Name。
Offset	Int	是	消息偏移量。
Limit	Int	是	最大返回消息条数，最大值为50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Total	String	总数量。
DeviceList	Array of DeviceList	设备列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetFamilySubDeviceList",
    "GatewayProductId":"NJ27OVLZT4",
    "GatewayDeviceName":"gwdev",
    "Offset":0,
    "Limit":2,
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例

```
{
    "Response": {
        "DeviceList": [{
            "FamilyId": "f_9b309d84c962****0a11b4c3d9588fcc1",
            "ProductId": "LAEG4YJE1A",
```

```
            "DeviceName": "subdev1",
            "DeviceId": "LAEG4YJE1A/subdev1",
            "AliasName": "",
            "RoomId": "0",
            "IconUrl": "",
            "DeviceType": 2,
            "CreateTime": 1583492992,
            "UpdateTime": 1583492992
        }],
        "RequestId": "req_1",
        "Total": 1
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalRPCError	调用超时。
InternalError.InternalRPCError	调用返回失败。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.ERR_APP_INVALID_IOTAPPID	App 无效 IotAppID。
ResourceNotFound	资源不存在
ResourceNotFound.DeviceNotExist	设备未创建或是已删除

蓝牙上报数据

最近更新时间：2024-10-11 18:26:01

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppReportDataAsDevice）用于小程序或 App 进行数据上报。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppReportDataAsDevice。
DeviceId	String	是	设备 ID。
Data	String	是	修改的物模型数据。
TimeStamp	Int64	是	Unix 毫秒级时间戳。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Data	String	返回的数据信息。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "123e4567-e89b-12d3-a456-426614174000",
    "Action": "AppReportDataAsDevice",
    "DeviceId": "22*****O/light1",
    "Data": "{\"light_switch\":0}",
    "TimeStamp": 1581422682000,
    "AccessToken": "8b4a70dd16**********2826868b18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": "",
        "RequestId": "123e4567-e89b-12d3-a456-426614174000"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。

蓝牙设备发送设备行为的回复消息

最近更新时间：2024-10-11 18:26:02

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppPublishMsgAsDevice）提供小程序或 App 模拟设备行为的回复消息功能。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppPublishMsgAsDevice。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Topic	String	是	上报的 Topic。
Payload	String	是	消息内容。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "123e4567-e89b-12d3-a456-426614174000",
    "Action": "AppPublishMsgAsDevice",
    "ProductId": "22*****O",
    "DeviceName": "light1",
    "Topic": "$thing/up/action/22*****O/light1"
    "Payload": "e1wibGlnaHRfc3dpdGNoXCI6MH0="
    "AccessToken": "8b4a70dd16**********2826868b18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "123e4567-e89b-12d3-a456-426614174000"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。

蓝牙设备上报事件

最近更新时间：2024-10-11 18:26:02

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppReportDeviceEvent）用于蓝牙设备通过 App 或小程序上报事件。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppReportDeviceEvent。
DeviceId	String	是	设备 ID，由 ProductId/DeviceName 拼接组成。
EventId	String	是	事件 ID。
Params	String	是	JSON 格式的 Event 参数。
Method	String	是	上报方法，可以是 ReportEventAsDevice、reported（默认）。
EventTimeStamp	Int64	是	Unix 毫秒级时间戳。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "123e4567-e89b-12d3-a456-426614174000",
    "Action": "AppReportDeviceEvent",
    "DeviceId": "QD****BNB/dev01",
    "EventId": "low_voltage",
    "Params": "{\"voltage\":3.1}",
    "Method": "ReportEventAsDevice",
    "AccessToken": "ningtoken******"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": "",
        "RequestId": "123e4567-e89b-12d3-a456-426614174000"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。

配网绑定设备

最近更新时间：2024-10-30 11:43:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppSigBindDeviceInFamily）用于小程序或 App 进行 Wi-Fi 设备配网绑定操作。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppSigBindDeviceInFamily。
FamilyId	String	是	家庭 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
RoomId	String	否	房间 ID。
DeviceTimestamp	Int64	是	设备时间戳，Unix 秒级时间戳。
ConnId	String	否	随机字符串，建议5个字节长度。
Signature	String	是	动态签名，由设备根据配网协议传输到小程序、App 端的签名。计算示例见下文。
BindType	String	是	绑定类型。 • wifi_sign：WIFI 绑定（默认）。 • bluetooth_sign：蓝牙绑定。 • other_sign：其他。
SignMethod	String	否	签名算法。 • hmacsha1：HMACSHA1 加密算法（默认）。 • hmacsha256：HMACSHA256 加密算法。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
AppDeviceInfo	Object of AppDeviceInfo	设备信息。

4. 动态签名计算

签名计算步骤：
1. 确认是WIFI配网还是蓝牙或者其他配网，需要加密的明文串拼接略有不同。
2. 获取设备的 psk，可以在控制台查看或者通过 API 获取。
3. 将 psk 通过 base64 解码后作为 hash 的 key 对明文编码。
4. hash 后的结果转十六进制字符串拼接即可。

WIFI 绑定和蓝牙或其他绑定签名计算步骤一致，只是加密明文拼接略有不同，详见示例代码。

WIFI 配网绑定签名

明文拼接格式： `"DeviceName=%s&DeviceTimestamp=%d&ProductId=%s&ConnId=%s"`

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AppBind {
    private static final String HMAC_SHA1_ALGORITHM = "HmacSHA1";
    private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
    public static final String DEVICE_PSK = "4B***==";
    public static final String PRODUCT_ID = "productId";
    public static final String DEVICE_NAME = "d1";
    public static final String CONN_ID = "12345";
    public static final int TIMESTAMP = 1694141664;
    public static final String ContentFormat = "DeviceName=%s&DeviceTimestamp=%d&ProductId=%s&ConnId=%s";

    public static void main(String[] args) throws Exception {
      String content = String.format(ContentFormat, DEVICE_NAME, TIMESTAMP, PRODUCT_ID, CONN_ID);
      // sha1
      System.out.println(calculateHMACSHA1(content, Base64.getDecoder().decode(DEVICE_PSK)));
      // sha256
      System.out.println(calculateHMACSHA256(content, Base64.getDecoder().decode(DEVICE_PSK)));
    }

    // 计算sha1签名
    public static String calculateHMACSHA1(String data, byte[] keyBytes) throws Exception {
        SecretKeySpec signingKey = new SecretKeySpec(keyBytes, HMAC_SHA1_ALGORITHM);
        Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
        mac.init(signingKey);
        byte[] dataBytes = data.getBytes();
        byte[] signatureBytes = mac.doFinal(dataBytes);

        return bytesToHex(signatureBytes);
    }

    // 计算sha256签名
    public static String calculateHMACSHA256(String data, byte[] keyBytes) throws Exception {
      SecretKeySpec signingKey = new SecretKeySpec(keyBytes, HMAC_SHA1_ALGORITHM);
      Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
      mac.init(signingKey);
      byte[] dataBytes = data.getBytes();
      byte[] signatureBytes = mac.doFinal(dataBytes);

      return bytesToHex(signatureBytes);
    }

    private static String bytesToHex(byte[] bytes) {
      StringBuilder builder = new StringBuilder();
      for (byte b : bytes) {
        builder.append(String.format("%02x", b));
      }

      return builder.toString();
    }
}
```

Go

```
import (
"crypto/hmac"
"crypto/sha1"
"encoding/base64"
"encoding/hex"
"fmt"
"hash"
)

var (
devicePsk  = "Re****Q=="
contentFmt = "DeviceName=%s&DeviceTimestamp=%d&ProductId=%s&ConnId=%s"
)

func main() {
    fmt.Println(getSign("deviceName", "productId", "connId", 1693898532))
}

// 获取sha1签名
func getSha1Sign() string {
    psk, _ := base64.StdEncoding.DecodeString(devicePsk)
    var mac hash.Hash
    mac = hmac.New(sha1.New, psk)
    mac.Write([]byte(contentFmt))
    result := mac.Sum(nil)

    return hex.EncodeToString(result)
}

// 获取sha256签名
func getSha256Sign() string {
    psk, _ := base64.StdEncoding.DecodeString(devicePsk)
    var mac hash.Hash
    mac = hmac.New(sha256.New, psk)
    mac.Write([]byte(contentFmt))
    result := mac.Sum(nil)

    return hex.EncodeToString(result)
}
```

C

```
#include <stdio.h>
#include <string.h>
#include <openssl/hmac.h>

const char *devicePsk = "Re5***==";
const char *contentFmt = "DeviceName=%s&DeviceTimestamp=%ld&ProductId=%s&ConnId=%s";

char *getSign(char *deviceName, char *productId, char *connId, long deviceTimestamp);

int main() {
    printf("%s\n", getSign("d1", "BRS**F", "9102344", 1693898532));
    return 0;
}

char *getSign(char *deviceName, char *productId, char *connId, long deviceTimestamp) {
    unsigned char psk[16];
    int psk_len = EVP_DecodeBlock(psk, devicePsk, strlen(devicePsk));
```

```
    printf("%s\n", psk);

    char content[256];
    sprintf(content, contentFmt, deviceName, deviceTimestamp, productId, connId);
    printf("%s\n", content);

    unsigned char result[20];
    unsigned int result_len;
    HMAC(EVP_sha1(), psk, psk_len, (unsigned char *) content, strlen(content), result, &result_len);

    char *hexResult = malloc(result_len * 2 + 1);
    for (int i = 0; i < result_len; i++) {
      sprintf(hexResult + i * 2, "%02X", result[i]);
    }
    hexResult[result_len * 2] = '\0';
    return hexResult;
}
```

蓝牙和其他配网绑定签名

明文拼接格式： `"%s%s;%s;%d"`，**字段依次是** `ProductId、DeviceName、ConnId、DeviceTimestamp。`

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class AppBind {
    private static final String HMAC_SHA1_ALGORITHM = "HmacSHA1";
    private static final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
    public static final String DEVICE_PSK = "4B****==";
    public static final String PRODUCT_ID = "productId";
    public static final String DEVICE_NAME = "d1";
    public static final String CONN_ID = "12345";
    public static final int TIMESTAMP = 1694141664;
    public static final String ContentFormat = "%s%s;%s;%d"; //
ProductId;DeviceName;ConnId;DeviceTimestamp

    public static void main(String[] args) throws Exception {
      String content = String.format(ContentFormat,PRODUCT_ID, DEVICE_NAME, CONN_ID, TIMESTAMP);
      // sha1
      System.out.println(calculateHMACSHA1(content, Base64.getDecoder().decode(DEVICE_PSK)));
      // sha256
      System.out.println(calculateHMACSHA256(content, Base64.getDecoder().decode(DEVICE_PSK)));
    }

    // 计算sha1签名
    public static String calculateHMACSHA1(String data, byte[] keyBytes) throws Exception {
      SecretKeySpec signingKey = new SecretKeySpec(keyBytes, HMAC_SHA1_ALGORITHM);
      Mac mac = Mac.getInstance(HMAC_SHA1_ALGORITHM);
      mac.init(signingKey);
      byte[] dataBytes = data.getBytes();
      byte[] signatureBytes = mac.doFinal(dataBytes);

      return bytesToHex(signatureBytes);
    }

    // 计算sha256签名
    public static String calculateHMACSHA256(String data, byte[] keyBytes) throws Exception {
```

```
        SecretKeySpec signingKey = new SecretKeySpec(keyBytes, HMAC_SHA1_ALGORITHM);
        Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
        mac.init(signingKey);
        byte[] dataBytes = data.getBytes();
        byte[] signatureBytes = mac.doFinal(dataBytes);

        return bytesToHex(signatureBytes);
    }

    private static String bytesToHex(byte[] bytes) {
        StringBuilder builder = new StringBuilder();
        for (byte b : bytes) {
            builder.append(String.format("%02x", b));
        }

        return builder.toString();
    }
}
```

Go

```
import (
"crypto/hmac"
"crypto/sha1"
"encoding/base64"
"encoding/hex"
"fmt"
"hash"
)

var (
    devicePsk             = "Re****=="
    ProductId             = "productId"
    DeviceName            = "d1"
    ConnId                = "12345"
    DeviceTimestamp int64 = 1694141664
    contentFmt            = fmt.Sprintf("%s%s;%s;%d", ProductId, DeviceName, ConnId, DeviceTimestamp) //
ProductId;DeviceName;ConnId;DeviceTimestamp)
func main() {
    fmt.Println(getSign())
}
func getSign() string {
    psk, _ := base64.StdEncoding.DecodeString(devicePsk)
    var mac hash.Hash
    mac = hmac.New(sha1.New, psk)
    mac.Write([]byte(contentFmt))
    result := mac.Sum(nil)

    return hex.EncodeToString(result)
}
```

C

```
#include <stdio.h>
#include <string.h>
#include <openssl/hmac.h>

const char *devicePsk = "Re5***==";
const char *contentFmt = "%s%s;%s;%d"; // ProductId;DeviceName;ConnId;DeviceTimestamp
```

```
char *getSign(char *deviceName, char *productId, char *connId, long deviceTimestamp);

int main() {
    printf("%s\n", getSign("d1", "BRS**F", "9102344", 1693898532));
    return 0;
}

char *getSign(char *deviceName, char *productId, char *connId, long deviceTimestamp) {
    unsigned char psk[16];
    int psk_len = EVP_DecodeBlock(psk, devicePsk, strlen(devicePsk));
    printf("%s\n", psk);

    char content[256];
    sprintf(content, contentFmt, productId, deviceName, connId, deviceTimestamp);
    printf("%s\n", content);

    unsigned char result[20];
    unsigned int result_len;
    HMAC(EVP_sha1(), psk, psk_len, (unsigned char *) content, strlen(content), result, &result_len);

    char *hexResult = malloc(result_len * 2 + 1);
    for (int i = 0; i < result_len; i++) {
        sprintf(hexResult + i * 2, "%02X", result[i]);
    }
    hexResult[result_len * 2] = '\0';

    return hexResult;
}5. 示例
```

默认 WIFI 配网绑定请求输入示例1：

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "DeviceTimestamp": 1694141664,
    "Action": "AppSigBindDeviceInFamily",
    "ConnId": "1938",
    "RequestId": "dd7b-1014bc4-rey76",
    "AccessToken": "51dbxxx4d4fdb",
    "FamilyId": "f_8xxxxfb1a",
    "Signature": "7CE3518***69EA8C",
    "ProductId": "productId",
    "RoomId": ""
    "BindType": "wifi_sign",
    "DeviceName": "d1"
}
```

默认 WIFI 配网绑定请求输入示例2：

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "DeviceTimestamp": 1694141664,
    "Action": "AppSigBindDeviceInFamily",
    "ConnId": "1938",
    "RequestId": "dd7b-1014bc4-rey76",
    "AccessToken": "51dbxxx4d4fdb",
    "FamilyId": "f_8xxxxfb1a",
```

```
    "Signature": "7CE3518***69EA8C",
    "ProductId": "productId",
    "RoomId": ""
    "BindType": "wifi_sign",
    "SignMethod":"hmacsha256",
    "DeviceName": "d1"
}
```

默认蓝牙配网绑定请求输入示例3：

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "DeviceTimestamp": 1694141664,
    "Action": "AppSigBindDeviceInFamily",
    "ConnId": "1938",
    "RequestId": "dd7b-1014bc4-rey76",
    "AccessToken": "51dbxxx4d4fdb",
    "FamilyId": "f_8xxxxfb1a",
    "Signature": "7CE3518***69EA8C",
    "ProductId": "productId",
    "RoomId": ""
    "BindType": "bluetooth_sign",
    "DeviceName": "d1"
}
```

默认蓝牙配网绑定请求输入示例4：

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "DeviceTimestamp": 1694141664,
    "Action": "AppSigBindDeviceInFamily",
    "ConnId": "1938",
    "RequestId": "dd7b-1014bc4-rey76",
    "AccessToken": "51dbxxx4d4fdb",
    "FamilyId": "f_8xxxxfb1a",
    "Signature": "7CE3518***69EA8C",
    "ProductId": "productId",
    "RoomId": ""
    "BindType": "bluetooth_sign",
    "SignMethod":"hmacsha256",
    "DeviceName": "d1"
}
```

输出示例：成功。

```
{
    "Response": {
        "RequestId": "req_1",
        "data": {
            "AppDeviceInfo": {
                "DeviceId" : "45454/d2",
                "ProductId": "45454",
                "DeviceName": "d2",
                "AliasName": "",
                "CreateTime": 1552910676,
                "UpdateTIme": 1552910676,
                "FamilyId": "1",
```

```
                "RoomId": "0",
                "IconUrl": ""
            }
        }
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。

绑定子设备到网关

最近更新时间：2024-10-11 18:26:02

1. 接口描述

接口请求域名： 接口请求域名替换

本接口(AppSigGatewayBindDevice)绑定子设备到网关设备

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppSigGatewayBindDevice。
ProductId,	String	是	产品 ID。
DeviceName	String	是	设备名称。
GatewayProductId	String	是	网关产品 ID。
GatewayDeviceNam e	String	是	网关设备 Name。
DeviceTimestamp	Int64	是	设备时间戳，Unix 秒级时间戳。
Nonce	String	是	随机字符串，建议 5 个字节长度。
Signature	String	是	动态签名。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。

4. 示例

签名生成步骤：

1. 对参数（deviceName、nonce、productId、timestamp）按字典序升序排序。
2. 将以上参数，按参数名称 = 参数值 & 参数名称 = 参数值拼接成字符串。
3. 使用 HMAC-sha1 算法对上一步中获得的字符串进行计算，密钥为 ProductSecret。
4. 将生成的结果使用 Base64 进行编码，即可获得最终的签名串放入 signature。

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
 "RequestId": "zzY1pZb3sz1N",
 "Action": "AppSigGatewayBindDevice",
 "AccessToken":"8b4a70dd16105f*****************18edd4e78a3bb8ec",
 "GatewayProductId": "NJ27OVLZT4",
 "GatewayDeviceName":"gwdev",
 "ProductId": "LAEG4YJE1A",
```

```
  "DeviceName": "subdev3",
  "DeviceTimestamp": 1583810805,
  "Nonce": "123456",
  "Signature": "YZGf0E***WGUEJGn+SxUf23HYA="
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "zzY1pZb3sz1N"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerException	发生错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InvalidAppParameter	App 请求的参数错误。
InvalidParameterValue.BindDeviceSigMismatch	绑定设备动态签名错误。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceType	设备类型错误。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.NoPermissionToFamily	操作该家庭无权限。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。

解绑子设备网关

最近更新时间：2024-11-05 15:02:52

1. 接口描述

接口请求域名： 接口请求域名替换

本接口(AppGatewayUnbindDevice) 从网关中解绑子设备

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGatewayUnbindDevice。
ProductId,	String	是	产品 ID。
DeviceName	String	是	设备名称。
GatewayProductId	String	是	网关产品 ID。
GatewayDeviceName	String	是	网关设备 Name。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
 POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
 content-type: application/json
{
 "RequestId": "zzY1pZb3sz1N",
 "Action": "AppGatewayUnbindDevice",
 "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
 "GatewayProductId": "NJ****T4",
 "GatewayDeviceName":"gwdev",
 "ProductId": "LA******1A",
 "DeviceName": "dev"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "zzY1pZb3sz1N"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerException	发生错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.BindDeviceNotConnected	设备近期没有连接到云。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InvalidAppParameter	App 请求的参数错误。
InvalidParameterValue.BindDeviceSigMismatch	绑定设备动态签名错误。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceType	设备类型错误。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.NoPermissionToFamily	操作该家庭无权限。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制

设备控制
用户控制设备

最近更新时间：2023-07-19 10:57:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppControlDeviceData）用于用户对绑定的设备发起控制操作。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppControlDeviceData。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名。
Data	String	是	控制设备报文。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "AppId": 1111,
    "Action": "AppControlDeviceData",
    "ProductId":"22F9Y6II7O",
    "DeviceName":"light1",
    "Data":"{\"light_switch\":0}",
    "AccessToken": "8b******************b8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": "",
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

同步调用设备行为

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCallDeviceActionSync）用于用户向已成功绑定的设备发起同步行为调用。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppCallDeviceActionSync
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
ActionId	String	是	行为功能的标识符，ActionId 在产品物模型的行为中定义。
InputParams	String	否	输入参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
ClientToken	String	调用 ID。
Status	String	返回状态。
OutputParqams	String	输出参数。 注意：此字段可能返回 null，表示取不到有效值。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "accesstoken**********acc",
  "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
  "ProductId": "AB*****2345",
  "DeviceName": "DEV",
  "ActionId": "actionid",
  "InputParams": "{\"color\":6}",
  "Action": "AppCallDeviceActionSync"
}
```

输出示例：成功

```
{
  "Response": {
    "ClientToken": "1ee12**********7df703e0d",
    "OutputParams": "",
    "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
    "Status": "succ"
  }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

异步调用设备行为

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCallDeviceActionAsync）用于向用户所绑定的设备发起异步行为调用请求，异步方式调用设备行为返回结果可通过长连接通信实时获取。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppCallDeviceActionAsync
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
ActionId	String	是	行为功能的标识符。
InputParams	String	否	输入参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
ClientToken	String	调用 ID。
Status	String	返回状态。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "accesstoken**********acc",
  "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
  "ProductId": "ABCDE12345",
  "DeviceName": "DEV",
  "ActionId": "actionid",
  "InputParams": "{\"color\":6}",
  "Action": "AppCallDeviceActionAsync"
}
```

输出示例：成功

```
{
  "Response": {
```

```
      "ClientToken": "1ee12**********7df703e0d",
      "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
      "Status": "Sent"
    }
  }
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

设备分享
App 端发送设备分享

最近更新时间：2024-10-11 16:01:45

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppSendShareDeviceInvite）用于发送设备分享邀请。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppSendShareDeviceInvite。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
ToUserID	String	是	被分享用户 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Status	String	请求结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppSendShareDeviceInvite",
  "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec",
  "RequestId": "1555507435215",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1",
  "ToUserID":"123456"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "1555507435215",
        "Status": "OK"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "1555507435215"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

获取设备分享 Token

最近更新时间：2024-10-11 16:01:45

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCreateShareDeviceToken）用于获取设备分享 Token。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateShareDeviceToken。
FamilyId	String	是	家庭 ID，成功创建家庭后，返回的 FamilyId
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
TokenContext	String	否	分享的上下文。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ShareDeviceToken	String	设备分享 Token。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppCreateShareDeviceToken",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "155550****15",
  "ProductId": "22F9****7O",
  "DeviceName": "light1"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "1555507****15",
        "ShareDeviceToken": "49d2b71dc5814****7c68a4c756d3ce9"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "1555507****15"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

获取设备分享 Token 信息

最近更新时间：2024-10-11 16:01:45

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDescribeShareDeviceToken）用于获取设备分享 Token 信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDescribeShareDeviceToken。
ShareDeviceToken	String	是	设备分享 Token。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ShareDeviceTokenInfo	Object of ShareDeviceTokenInfo	响应的结果。
AliasName	String	设备别名
IconUrl	String	图标 URL

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeShareDeviceToken",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "1555507****15",
  "ShareDeviceToken": "49d2b71dc5814354****8a4c756d3ce9"
}
```

输出示例：成功

```
// UserId、UserNick 在被分享者接受前为空
// 被分享者接受分享后，UserId 保存的是被分享者的信息
// 前端可以通过UserId是否为空，来判断绑定情况。
{
  "Response": {
    "RequestId": "1555507****15",
    "ShareDeviceTokenInfo": {
      "FromUserId": "1",
      "FromUserNick": "tests",
      "UserId": "2",
      "UserNick": "test2",
```

```
      "ProductId": "22F9Y6II7O",
      "DeviceName": "light1",
      "AliasName": "my_light",
      "IconUrl": "",
      "BindTime": 0,
      "ExpireTime": 1574757262,
      "CreateTime": 1574152462
    }
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidShareDeviceToken",
      "Message": "设备分享Token无效"
    },
    "RequestId": "1555507****15"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidShareDeviceToken	设备分享 Token 无效。

绑定用户分享的设备

最近更新时间：2024-10-11 16:01:46

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppBindUserShareDevice）用于绑定用户分享的设备。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppBindUserShareDevice。
ShareDeviceToken	String	是	设备分享的 Token。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Status	String	响应结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppBindUserShareDevice",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "1555507****15",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1",
  "ShareDeviceToken": "0820728466424d8f****422ec06bd398"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "1555507****15",
    "Status": "OK"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.DeviceBinded",
      "Message": "设备已经和当前用户绑定"
    },
    "RequestId": "1555507****15"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.DeviecBinded	设备已经和当前用户绑定。
InvalidParameterValue.UserBindExceedLimit	用户绑定分享设备数已达限制。
InvalidParameterValue.DeviceShareExceedLimit	设备已达最大分享用户数限制。
InvalidParameterValue.CannotBindOwnDevice	不能绑定自己分享的设备。
InvalidParameterValue.ShareDeviceTokenUsed	设备分享 Token 已使用。
InvalidParameterValue.InvalidShareDeviceToken	设备分享 Token 无效。

查询用户分享设备列表

最近更新时间：2024-10-11 16:01:46

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppListUserShareDevices）用于查询用户分享的设备列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppListUserShareDevices。
Offset	Int	是	分页偏移，0起始，最大不超过500。
Limit	Int	是	单次拉取数量，默认为10，最大不超过100。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Total	Int	返回的消息总条数。
ShareDevices	Array of ShareDevices	响应设备信息。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppListUserShareDevices",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "1555507****15",
  "Offset": 0,
  "Limit": 10
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "1555507****15",
    "ShareDevices": [
      {
        "ProductId": "WTXXXX0MF2",
        "DeviceName": "device_XXXX001",
        "DeviceId": "WTXXXX0MF2/device_XXXX001",
        "AliasName": "test_device",
        "IconUrl": "https://yyy-xxxx-xxx.cos.ap-
guangzhou.myqcloud.com/113xxxx7a0b273bxxxxc0b9040feeb/1xxxxxxxxx0.png",
        "CreateTime": "2019-12-17T19:39:21+08:00"
```

```
        }
      ],
      "Total": 1
    }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidAccessToken",
      "Message": "Token无效"
    },
    "RequestId": "1555507****15"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除用户分享的设备

最近更新时间：2024-10-11 16:01:46

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppRemoveUserShareDevice）用于删除用户分享的设备。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppRemoveUserShareDevice。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Status	String	响应结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppRemoveUserShareDevice",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "1555507****15",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "1555507****15",
    "Status": "OK"
  }
}
```

输出示例：失败

```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.AppDeviceNotExist",
```

```
      "Message": "没有找到绑定设备"
    },
    "RequestId": "1555507****15"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.AppDeviceNotExist	没有找到绑定设备。

查询设备的用户列表

最近更新时间：2024-10-11 16:01:46

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppListShareDeviceUsers）用于查询设备用户列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppListShareDeviceUsers。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Offset	Int	是	分页偏移，0起始，最大不超过500。
Limit	Int	是	单次拉取数量，默认为10，最大不超过100。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Total	Int	返回的数据总数。
Users	Array of ShareUserInfo	用户列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppListShareDeviceUsers",
  "AccessToken": "user2",
  "RequestId": "1555507****15",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1",
  "Offset": 0,
  "Limit": 10
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "1555507****15",
    "Users": [
      {
        "UserId": "1",
```

```
        "CountryCode": "86",
        "PhoneNumber": "139****5678",
        "NickName": "tests",
        "Avatar": "",
        "BindTime": 1574153536
      }
    ],
    "Total": 1
  }
}
```


输出示例：失败


```
{
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.InvalidAccessToken",
      "Message": "Token无效"
    },
    "RequestId": "1555507****15"
  }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除设备分享的用户

最近更新时间：2023-07-14 16:01:46

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppRemoveShareDeviceUser）用于删除设备分享的用户。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppRemoveShareDeviceUser。
RemoveUserId	String	是	待删除用户 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Status	String	响应结果。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppRemoveShareDeviceUser",
  "AccessToken": "user2",
  "RequestId": "15555******15",
  "RemoveUserId": "1234555",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "155550*****15",
    "Status": "OK"
  }
}
```

输出示例：失败

```
{
```

```
  "Response": {
    "Error": {
      "Code": "InvalidParameterValue.AppDeviceNotExist",
      "Message": "没有找到绑定设备"
    },
    "RequestId": "15555******15"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.AppDeviceNotExist	没有找到绑定设备。
InvalidParameterValue.NoPermission	用户对该设备无权限。

家庭管理
创建家庭

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppCreateFamily）用于创建家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateFamily。
Name	String	是	家庭名称。
Address	String	是	家庭地址。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
FamilyId	String	成功创建的家庭 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppCreateFamily",
    "Name": "family_name",
    "Address": "family_address",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "FamilyId": "e0e23a2b33a24652b6**********a1cf"
        },
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除家庭

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteFamily）用于删除家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteFamily。
Name	String	是	需要删除的家庭名称。
FamilyId	String	是	需要删除的家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDeleteFamily",
    "Name":"XXX的家庭",
    "FamilyId":"e0e23a2b33a24652b6**********a1cf",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改家庭

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppModifyFamily）用于修改家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyFamily。
Name	String	是	修改后的家庭名称。
FamilyId	String	是	需要修改的家庭 ID。
Address	String	是	修改后的家庭地址。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppModifyFamily",
    "Name": "family_name",
    "Address": "family_address",
    "FamilyId":"e0e23a2b33a24652b6**********a1cf",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取家庭列表

最近更新时间：2024-11-01 17:48:52

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFamilyList）用于获取家庭列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFamilyList。
Offset	Int	否	所需要查询数据的偏移量。
Limit	Int	否	返回条数的限制量，最多为50条。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
Total	Int	返回的家庭数量总数。
FamilyList	Array	返回的家庭列表信息数组，请参见 FamilyList 。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetFamilyList",
    "Offset":0,
    "Limit":2,
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1",
        "FamilyList": [{
            "FamilyId": "a1c6939b39d345b897**********a12c",
            "Name": "family_name",
            "Role": 0,    // 1:管理员  0：普通成员
            "CreateTime": 1570786578,
            "UpdateTime": 1570790807
        }],
        "Total":1
    }
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取家庭详情

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDescribeFamily）用于获取家庭详情。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDescribeFamily。
FamilyId	String	是	所需查询详情的家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
FamilyId	String	家庭 ID。
FamilyName	String	家庭名称。
Address	String	家庭地址。
CreateTime	Int64	创建时间，UNIX 时间戳（秒级）。
UpdateTime	Int64	最后一次更新时间，UNIX 时间戳（秒级）。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDescribeFamily",
    "FamilyId": "a1c6939b39d345b897**********a12c",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1",
        "Data": {
            "FamilyId": "a1c6939b39d345b897**********a12c",
            "FamilyName": "family_name",
            "Address": "family_address",
            "CreateTime": 1570786578,
            "UpdateTime": 1570790807
```

```
            }
        }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

新建房间

最近更新时间：2023-07-19 10:57:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCreateRoom）用于新建房间。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateRoom。
FamilyId	String	是	新建房间的所属家庭 ID。
Name	String	是	新建的房间名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
RoomId	String	成功创建的房间 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppCreateRoom",
    "Name": "name",
    "FamilyId": "family_address",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "RoomId": 123456
        },
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述

InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改房间

最近更新时间：2024-10-11 10:57:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyRoom）用于修改房间。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyRoom。
FamilyId	String	是	需要修改的房间所属家庭 ID。
Name	String	否	房间名称。
RoomId	String	是	需要修改的房间 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
        "RequestId": "req_1",
        "Action": "AppModifyRoom",
        "FamilyId":"fa070f59a2fe4****43815310c794fdf",
        "RoomId":"5",
        "Name":"TestName",
        "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。

InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除房间

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteRoom）用于删除房间。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteRoom。
FamilyId	String	是	需要删除的房间所属家庭 ID。
RoomId	String	是	需要删除的房间 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDeleteRoom",
    "FamilyId":"e0e23a2b33a24652b6**********a1cf",
    "RoomId":"123456",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取房间列表

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppGetRoomList）用于获取房间列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetRoomList。
Offset	Int	否	所需要查询的数据的偏移量。
Limit	Int	否	所需要查询的总限制量，最大返回50条。
FamilyId	String	否	需要获取的家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Total	Int	返回的设备总数
RoomList	Array	返回的家庭列表信息数组，请参见 Roomlist。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetRoomList",
    "FamilyId":"fa070f59a2fe4f1da4**********4fdf",
    "Offset":0,
    "Limit":2,
    "AccessToken": "xxx"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1",
        "RoomList": [{
            "RoomId": "123456",
            "RoomName": "name",
            "DeviceNum": 2,
            "CreateTime": 1570786578,
            "UpdateTime": 1570790807
        }],
        "Total": 1
    }
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效

邀请家庭成员

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppInviteMember）用于向微信好友发送邀请加入家庭请求。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppInviteMember。
FamilyId	String	是	家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ShareToken	String	分享 Token，被邀请者可凭此分享 Token 调用 成员加入家庭 接口以加入家庭。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppInviteMember",
    "FamilyId":"e0e23a2b33a24652b6**********a1cf",
    "AccessToken": "e1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "ShareToken": "e0******************a1cf",
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

成员加入家庭

最近更新时间：2024-10-11 10:57:12

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppJoinFamily）用于成员加入家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppJoinFamily。
ShareToken	String	是	分享 Token，由家庭管理员调用 邀请家庭成员 生成，被邀请者可凭此分享 Token 加入相应的家庭。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "550e8400-e29b-41d4-a716-446655440000",
  "Action": "AppJoinFamily",
  "ShareToken":"e0******************a1cf",
  "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "550e8400-e29b-41d4-a716-446655440000"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除家庭成员

最近更新时间：2024-10-11 10:57:13

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteFamilyMember）用于管理员删除家庭成员。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteFamilyMember。
FamilyId	String	否	要移除成员的家庭 ID。
MemberId	String	是	需要移除的成员 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "550e8400-e29b-41d4-a716-446655440000",
    "Action": "AppDeleteFamilyMember",
    "FamilyId":"f_abcd****abcd",
    "MemberId":"12345678",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "550e8400-e29b-41d4-a716-446655440000"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

成员退出家庭

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppExitFamily）用于成员主动退出某个家庭。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppExitFamily。
FamilyId	String	是	需要退出的家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "550e8400-e29b-41d4-a716-44665*****00",
    "Action": "AppExitFamily",
    "FamilyId":"e0e23a2b33a24652b6**********a1cf",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "550e8400-e29b-41d4-a716-44665*****00"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取家庭成员列表

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFamilyMemberList）用于获取家庭成员列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFamilyMemberList。
Offset	Int	否	所需要查询的数据的偏移量。
Limit	Int	否	所需要查询的总限制量，最大返回50条。
FamilyId	String	是	需要获取的家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Total	Int	返回的家庭成员总数。
MemberList	Array	返回的家庭成员数组，请参见 MemberList 。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "FamilyId": "f_b3ac7e6b1cd24392b41e**********5a",
    "Offset": 0,
    "Limit": 100,
    "Action": "AppGetFamilyMemberList",
    "RequestId": "hojkE0Hx88",
    "AccessToken": "3c******************40c5"
}
```

输出示例：成功

```
{
    "MemberList": [{
        "UserID": "7267",
        "NickName": "deny",
        "Avatar":
"https://wx.qlogo.cn/mmopen/vi_32/V2kIBBE35phMrFKu5sra16bzpUuPtokyWTbMRPt7bRGBjx4fEqQrZxoDYyxNVbtkXMosuxQe
IqiafIZJ8ADr8cw/132",
        "Role": 1
    }],
```

```
    "Total": 1,
    "RequestId": "hojkE0Hx88"
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

App 端邀请家庭成员

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppSendShareFamilyInvite）用于发送家庭分享邀请。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppSendShareFamilyInvite
FamilyId	String	是	家庭 ID
ToUserID	String	是	被分享用户 ID

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppSendShareFamilyInvite",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "RequestId": "1555*****5215",
  "FamilyId":"e0e23a2b33a24652b6**********a1cf",
  "ToUserID":"123456"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "1555*****5215"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
```

```
        },
        "RequestId": "1555*****5215"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该家庭无权限。

设备定时
新建定时任务

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppCreateTimer）用于用户创建设备的定时任务。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值 AppCreateTimer。
ProductId	String	是	新建定时任务所属产品 ID。
DeviceName	String	是	新建定时任务控制的设备名称。
TimerName	String	是	定时器的名称。
Days	String	是	定时器开启时间，每一位，0：关闭，1：开启，从左至右依次表示：周日、周一、周二、周三、周四、周五、周六。
TimePoint	String	是	定时器开启时间点。
Repeat	int	是	是否循环。 • 0：不需要。 • 1：需要。
Data	String	是	定时器启动时下发的控制报文。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
TimerId	String	定时器 ID。
TimerName	String	定时器名称。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppCreateTimer",
    "ProductId": "US4*****IK",
    "DeviceName": "411_3",
    "AccessToken": "c1******************847e",
    "TimerName": "timer_test",
    "Days": "1000000", // 每一位 0:关闭,1:开启, 从左至右依次表示: 周日 周一 周二 周三 周四 周五 周六
```

```
    "TimePoint": "06:30",
    "Repeat": 1, //是否需要循环  0:不需要 1 需要
    "Data": "{\"brightness\": 25}"
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "TimerId": "e0e23a2b33a24652b6**********a1cf",
            "TimerName": "timer_test"
        },
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改定时任务

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyTimer）用于修改设备的定时任务。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值 AppModifyTimer。
ProductId	String	是	修改定时器对应的产品 ID。
DeviceName	String	是	定时器对应的设备名称。
TimerName	String	是	定时器的名称。
TimerId	String	是	定时器 ID。
TimePoint	String	是	定时器开启时间点。
Days	String	是	定时器开启时间，每一位，0：关闭，1：开启，从左至右依次表示：周日、周一、周二、周三、周四、周五、周六。
Repeat	Int	是	是否循环。 • 0：不需要。 • 1：需要。
Data	String	是	定时器启动时，下发的控制报文。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppModifyTimer",
    "ProductId": "US*****DIK",
    "DeviceName": "411_3",
    "AccessToken": "c1******************847e",
    "TimerId":"a1c6939b39d345b897**********a12c",
    "TimerName": "timer_test_modify",
    "Days": "1100000",
    "TimePoint": "9:30",
    "Repeat": 0,
    "Data": "{\"brightness\": 28}"
```

```
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除定时任务

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteTimer）用于删除设备定时任务。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值 AppDeleteTimer。
ProductId	String	是	修改定时器对应的产品 ID。
DeviceName	String	是	定时器对应的设备名称。
TimerId	String	是	定时器 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDeleteTimer",
    "ProductId": "US4*****IK",
    "DeviceName": "411_3",
    "TimerId":"e0e23a2b33a24652b6**********a1cf",
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述

InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

修改定时任务状态

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyTimerStatus）用于修改定时任务的启停状态。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值 AppModifyTimerStatus。
ProductId	String	是	需要修改定时任务列表的产品 ID。
DeviceName	String	是	需修改定时器状态设备名称。
TimerId	String	是	定时器 ID。
Status	Int	否	需要修改的定时器状态。 • 0：关闭。 • 1：开启。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppModifyTimerStatus",
    "ProductId": "US4*****IK",
    "DeviceName": "411_3",
    "TimerId":"a1c6939b39d345b897**********a12c",
    "Status":0,
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取定时任务列表

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetTimerList）用于获取定时器列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值 AppGetTimerList。
ProductId	String	是	需要获取定时任务列表的产品 ID。
DeviceName	String	是	需要获取定时任务列表的设备名称。
Offset	Int	否	获取的列表偏移量。
Limit	Int	否	获取到的总消息条数，最大为50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
TimerList	Array	获取到的产品列表，请参见 TimerList。
Total	Int	总共拥有的条数。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetTimerList",
    "ProductId": "US4*****IK",
    "DeviceName": "411_3",
    "Offset":0,
    "Limit":2,
    "AccessToken": "c1******************847e"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1",
        "Total":2,
        "TimerList": [{
            "TimerId": "a1c6939b39d345b897**********a12c",
```

```
            "TimerName": "timer_test_modify",
            "ProductId": "US4*****IK",
            "DeviceName": "411_3",
            "Days": "1100000",
            "TimePoint": "09:30",
            "Repeat": 0,
            "Data": "{\"brightness\": 28}",
            "Status": 0,
            "CreateTime": 1570786578,
            "UpdateTime": 1570790807
        }, {
            "TimerId": "f888dd20c70744939a*********1c3a",
            "TimerName": "timer_test",
            "ProductId": "US4*****IK",
            "DeviceName": "411_3",
            "Days": "1000000",
            "TimePoint": "07:00",
            "Repeat": 1,
            "Data": "{\"brightness\": 25}",
            "Status": 1,
            "CreateTime": 1570786483,
            "UpdateTime": 1570786483
        }]
    }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

固件升级
上报设备固件版本

最近更新时间：2024-10-03 11:18:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppReportFirmwareVersion）用于上报设备固件版本。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppReportFirmwareVersion。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Version	String	是	设备上报的固件版本。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "Action": "AppReportFirmwareVersion",
  "Version": "1.0",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
  "AccessToken": "871d2d42****************692afae0"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

查询设备可升级固件版本

最近更新时间：2024-10-11 10:57:14

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCheckFirmwareUpdate）用于查询设备可升级固件版本。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCheckFirmwareUpdate。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
CurrentVersion	String	设备当前固件版本。
DstVersion	String	固件可升级版本。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "Action": "AppCheckFirmwareUpdate",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
  "AccessToken": "871d2d42****************692afae0"
}
```

输出示例：成功

```
{
  "Response": {
    "DstVersion": "1.1",
    "CurrentVersion": "1.0",
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
FailedOperation	操作失败。
FailedOperation.DeviceInfoOutdated	设备固件版本错误（设备上报的固件版本与当前固件升级任务不匹配）。

查询设备固件下载地址

最近更新时间：2024-10-03 11:18:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceOTAInfo）用于查询设备固件下载地址。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceOTAInfo。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
FirmwareURL	String	固件下载地址。
TargetVersion	String	目标固件版本。
UploadVersion	String	设备上报的固件版本。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "871d2d42****************692afae0",
  "Action": "AppGetDeviceOTAInfo",
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
}
```

输出示例：成功

```
{
  "Response": {
    "FirmwareURL": "https://examplebucket-12500000000.cos.ap-guangzhou.myqcloud.com/path/to/firmware?
sign=***",
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
    "TargetVersion": "1.1",
    "UploadVersion": "1.0"
```

```
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceFirmwareTaskNotExist	未查询到固件升级任务。

确认固件升级任务

最近更新时间：2024-10-11 10:57:14

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppPublishFirmwareUpdateMessage）用于用户确认升级后，云端向设备发起固件升级请求。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppPublishFirmwareUpdateMessage。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "Action": "AppPublishFirmwareUpdateMessage",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
  "AccessToken": "871d2d42****************692afae0"
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。

InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceHasNoFirmware	设备无固件版本。
FailedOperation	操作失败。
FailedOperation.DeviceOffline	设备处于离线状态。

上报固件升级任务状态

最近更新时间：2024-10-03 11:18:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppReportOTAStatus）用于上报设备固件升级状态及进度。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppReportOTAStatus。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Version	String	是	固件升级的目标版本。
State	String	是	固件升级状态： • downloading：下载中。 • fail：升级失败。 • done：升级成功。 • burning：烧录中。 • updating：更新中。
ResultCode	Int	是	错误码： • 0：成功。 • -1：下载超时。 • -2：文件不存在。 • -3：签名过期。 • -4：MD5 不匹配。 • -5：更新固件失败。
ResultMsg	String	否	错误附加信息，当 State 取值为 fail 时必填。
Persent	Int	否	升级进度百分比，取值范围 0 ~ 100。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
```

```
  "AccessToken": "871d2d42****************692afae0",
  "Action": "AppReportOTAStatus",
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
  "State": "done",
  "ResultCode": 0,
  "ResultMsg": "",
  "Version": "1.1",
  "Persent": 100
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b"
  }
}
```

示例2

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "871d2d42****************692afae0",
  "Action": "AppReportOTAStatus",
  "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b",
  "ProductId": "PR******ID",
  "DeviceName": "de***01",
  "State": "downloading",
  "ResultCode": 0,
  "ResultMsg": "",
  "Version": "2.0",
  "Persent": 20
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "584406e9-bf59-46a3-8d45-39f1891bed7b"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

查询设备固件升级状态

最近更新时间：2024-10-11 16:01:49

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDescribeFirmwareUpdateStatus）用于查询设备固件升级状态及进度。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDescribeFirmwareUpdateStatus。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
OriVersion	String	升级任务源版本。
DstVersion	String	升级任务目标版本。
ErrMsg	String	错误信息。
Percent	Int64	升级进展百分比。
Status	Int64	升级状态： • 0：设备离线。 • 1：待处理。 • 2：消息下发成功。 • 3：下载中。 • 4：烧录中。 • 5：失败。 • 6：升级完成。 • 7：正在处理中。 • 8：等待用户确认。 • 20：下载完成。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "Action": "AppDescribeFirmwareUpdateStatus",
    "ProductId":"22*******O",
    "DeviceName":"li****1",
    "AccessToken": "8b4d16105f******************18edd4e78a3bb8ec"
```

```
}
```

输出示例：成功

```
{
    "Response": {
        "DstVersion": "2.0",
        "ErrMsg": "",
        "OriVersion": "1.0",
        "Percent": 0,
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

查看固件信息

最近更新时间：2023-10-24 17:09:51

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDescribeFirmware）用于查询设备固件信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDescribeFirmware。
ProductID	String	是	产品 ID
FirmwareVersion	String	是	固件版本号

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ProductId	String	产品 ID
Name	String	固件名称。 注意：此字段可能返回 null，表示取不到有效值。
Description	String	固件描述。 注意：此字段可能返回 null，表示取不到有效值。
Md5sum	String	固件 Md5 值。 注意：此字段可能返回 null，表示取不到有效值。
Createtime	Integer	固件上传的秒级时间戳。 注意：此字段可能返回 null，表示取不到有效值。
ProductName	String	产品名称
FwType	String	固件升级模块。 注意：此字段可能返回 null，表示取不到有效值。
Version	String	固件版本号

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "Action": "AppDescribeFirmware",
    "ProductId":"22*******O",
    "FirmwareVersion":"t.2",
    "AccessToken": "8b4d16105f******************18edd4e78a3bb8ec"
```

```
}
```

输出示例：成功

```
{
  "Response": {
    "Createtime": 1599626765,
    "Description": "ttttt",
    "Md5sum": "1a5c386576074d22a604b795e8917e1a",
    "Name": "test",
    "ProductId": "I6KTCZ170U",
    "ProductName": "8bvsn6go_测试OTA升级",
    "RequestId": "01365e8c-f025-40b9-97a4-fa583d6c569e",
    "Version": "t.2",
    "FwType": "mcu"
  }
}
```

5. 错误码

错误码	描述
InternalError.InternalRPCError	内部 RPC 错误。
ResourceNotFound.FirmwareNotExist	固件不存在。
ResourceNotFound.StudioProductNotExist	产品不存在。

消息管理
获取消息列表

最近更新时间：2023-07-19 10:57:14

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetMessages）用于查询消息列表，消息包括设备告警消息、操作通知消息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时需要提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetMessages。
Category	Int64	是	主类型。 • 1：设备。 • 2：家庭。 • 3：通知。
MsgID	String	否	消息 ID，首次可不传。
MsgTimestamp	Int64	否	消息的时间戳，首次可不传或传0。
Limit	Int64	是	最大返回条数，最大不超过100。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Listover	String	用于翻页。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "Action": "AppGetMessages",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "AccessToken": "8b******************b8ec",
    "Category": 2,
    "MsgID": "",
    "MsgTimestamp": 0,
    "Limit": 5
}
```

输出示例：成功

```
{
    "Response": {
```

```
        "Data": {
            "Listover": true,
            "Msgs": [{
                "UserID": "1",
                "FromUserID": "",
                "MsgID": "65167410520395776",
                "Category": 1,
                "MsgType": 100,
                "MsgTitle": "Device Warning",
                "MsgContent": "Some message Wed Nov 27 2019 11:52:01 GMT+0000.",
                "MsgTimestamp": 1574855521420,
                "ProductId": "",
                "DeviceName": "",
                "DeviceAlias": "",
                "FamilyId": "",
                "FamilyName": "",
                "RelateUserID": "",
                "CreateAt": "2019-11-27T11:52:01.42Z"
            }, {
                "UserID": "1",
                "FromUserID": "1",
                "MsgID": "65167395785805824",
                "Category": 2,
                "MsgType": 200,
                "MsgTitle": "成员添加结果通知",
                "MsgContent": ""jordan"将"dylan"添加为"深圳的家"的成员，成员可以查看和控制家里的智能设备。 Wed Nov
27 2019 11:51:57 GMT+0000",
                "MsgTimestamp": 1574855517907,
                "ProductId": "",
                "DeviceName": "",
                "DeviceAlias": "",
                "FamilyId": "1",
                "FamilyName": "test",
                "RelateUserID": "2",
                "CreateAt": "2019-11-27T11:51:57.907Z"
            }, {
                "UserID": "1",
                "FromUserID": "1",
                "MsgID": "65167386239569920",
                "Category": 2,
                "MsgType": 201,
                "MsgTitle": "成员删除结果通知",
                "MsgContent": ""jordan"将"dylan"从"深圳的家"中删除，删除后将不能查看和控制家中的智能设备。 Wed Nov
27 2019 11:51:54 GMT+0000",
                "MsgTimestamp": 1574855515631,
                "ProductId": "",
                "DeviceName": "",
                "DeviceAlias": "",
                "FamilyId": "1",
                "FamilyName": "test",
                "RelateUserID": "2",
                "CreateAt": "2019-11-27T11:51:55.631Z"
            }, {
                "UserID": "1",
                "FromUserID": "1",
                "MsgID": "65167339519217664",
                "Category": 3,
                "MsgType": 300,
                "MsgTitle": "设备分享结果通知",
                "MsgContent": ""dylan"已成功接收"jordan"分享的设备"智能灯"，现在可以查看和控制设备了。 Wed Nov 27
2019 11:51:44 GMT+0000",
```

```
                "MsgTimestamp": 1574855504492,
                "ProductId": "22F9Y6II7O",
                "DeviceName": "light1",
                "DeviceAlias": "light1",
                "FamilyId": "",
                "FamilyName": "",
                "RelateUserID": "2",
                "CreateAt": "2019-11-27T11:51:44.492Z"
            }]
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除消息

最近更新时间：2023-07-14 16:01:49

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteMessage）用于删除消息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteMessage。
MsgID	String	是	消息 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "Action": "AppDeleteMessage",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "AccessToken": "8b******************b8ec",
    "MsgID": 1633328834809856
}
```

输出示例：成功

```
{
    "Response": {
        "Data": "OK",
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
```

```
        },
        "RequestId": ""
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.MsgIDInvalid	消息 ID 非法。

手动智能
创建手动智能联动

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppCreateScene）用于创建手动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateScene。
FamilyId	String	是	手动智能联动所属家庭 ID。
SceneName	String	是	手动智能联动的名称。
SceneIcon	String	是	手动智能联动的背景图片地址。
Actions	Array of Action	是	手动智能联动动作列表。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
SceneId	String	手动智能联动 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Actions": [
    {
      "ActionType": 0,
      "ProductId": "R32O***0EU",
      "DeviceName": "df****yY",
      "Data": "{\"brightness\": 25}"
    }
  ],
  "AccessToken": "x***v2",
  "RequestId": "re***_1",
  "Action": "AppCreateScene",
  "FamilyId": "f_9b30*******c3d9588fcc1",
  "SceneName": "name",
  "SceneIcon": "icon"
}
```

输出示例：成功

```
{
  "Response": {
    "SceneId": "a_cd61ddbe*****47d21d6aeffea",
    "RequestId": "req_1"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

获取手动智能联动列表

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppGetSceneList）用于获取手动智能联动列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetSceneList。
FamilyId	String	是	家庭 ID。
Offset	Int	是	消息偏移量。
Limit	Int	是	最大返回消息条数，最大值为 50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
SceneList	Array of Scene	手动智能联动列表。
Total	Int	手动智能联动总数。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "AccessToken": "***v2",
    "RequestId": "req_1",
    "Action": "AppGetSceneList",
    "FamilyId": "f_9b309********4c3d9588fcc1",
    "Offset": 0,
    "Limit": 10
}
```

输出示例：成功

```
{
  "Response": {
    "RequestId": "req_1",
    "SceneList": [
      {
        "SceneId": "s_f7feb440d******28f3a598d00c",
        "FamilyId": "f_9b309d8******a11b4c3d9588fcc1",
        "SceneName": "f****na",
        "SceneIcon": "",
```

```
        "Actions": [         // Actions数组中的每项的字段，为空则不出现
          {
            "ActionType": 0,
            "ProductId": "R32****0EU",
            "DeviceName": "df****yY",
            "Data": "{\"brightness\": 25}",
            "AliasName": "1***45"      // 说明：AliasName为设备别名 。
          },
          {
            "ActionType": 1,
            "Data": "100"
          }
        ],
        "UserId": "1",
        "CreateTime": 1578557536,
        "UpdateTime": 1578560598,
        "Flag": 0,  // Flag ：0 为正常的场景，1：场景中的设备异常，不在家庭中
        "Status": 0 // 场景运行状态 0未运行，1运行中
      }
    ],
    "Total": 1
  }
}
 // IconUrl为具体设备的url
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnsupportedOperation.SceneRunInProgress	场景执行中。

删除手动智能联动

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppDeleteScene）用于删除手动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteScene。
SceneId	String	是	手动智能联动 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "***v2",
  "RequestId": "req_1",
  "SceneId": "s_527cb52******b453cb6a46f653",
  "Action": "AppDeleteScene"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnsupportedOperation.SceneRunInProgress	场景执行中。

修改手动智能联动

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppModifyScene）用于修改手动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyScene。
SceneId	String	是	手动智能联动 ID。
SceneName	String	是	手动智能联动的名称。
SceneIcon	String	是	手动智能联动的背景图片地址。
Actions	Array of Action	是	手动智能联动动作列表。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Actions": [
    {
      "ActionType": 0,
      "ProductId": "R32****0EU",
      "DeviceName": "df****yY",
      "Data": "{\"brightness\": 25}"
    }
  ],
  "AccessToken": "***v2",
  "RequestId": "req_1",
  "Action": "AppModifyScene",
  "SceneId": "s_527cb5*****3cb6a46f653",
  "SceneName": "name",
  "SceneIcon": "****icon"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnsupportedOperation.SceneRunInProgress	场景执行中。

执行手动智能联动

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppRunScene）用于执行手动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppRunScene。
SceneId	String	是	手动智能联动 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "***v2",
  "RequestId": "req_1",
  "SceneId": "s_527cb5*********53cb6a46f653",
  "Action": "AppRunScene"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限
UnsupportedOperation.SceneRunInProgress	场景执行中。

自动智能
创建自动智能联动

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppCreateAutomation）用于创建自动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateAutomation。
FamilyId	String	是	自动智能联动所属家庭 ID。
Icon	String	是	自动智能联动的背景图片地址。
Name	String	是	自动智能联动的名称。
Status	int	是	自动智能联动开关状态。 • 0：关闭。 • 1：启用。
MatchType	int	是	条件匹配类型。 • 0：全部条件满足。 • 1：其中一个条件满足。
Conditions	Array of Condition	是	自动智能联动触发条件。
Actions	Array of Action	是	自动智能联动动作列表。
EffectiveBeginTime	String	是	生效开始时间。
EffectiveEndTime	String	是	生效结束时间。
EffectiveDays	String	是	生效日期由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
AutomationId	String	自动智能联动 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
```

```
    "Action": "AppCreateAutomation",
    "FamilyId": "f_9****c1",
    "Icon": "https://www.icon.com/",
    "Name": "autoName",
    "Status": 0, //联动的开关 0：关闭 1：启用
    "MatchType": 1, //条件匹配类型 0：条件全部与  1：条件全部或
    "EffectiveBeginTime": "00:00",  // 【两个新增参数，用来表示开始和结束时间】
    "EffectiveEndTime": "23:59",
    "EffectiveDays": "1111111", // 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六
    "Conditions": [
        {
            "CondId": "abc", //条件ID，保证在一个联动下唯一即可
            "CondType": 0, //条件类型 0：设备属性值条件  1：定时条件
            "Property": {
                "ProductId": "N****4",
                "DeviceName": "g***ev",
                "AliasName": "devName",
                "IconUrl": "productIcon",
                "PropertyId": "switch", //设备的属性Id
                "Op": "eq", //条件操作符  eq 等于  ne 不等于  gt 大于  lt 小于  ge 大等于  le 小等于
                "Value": 1 //比较的值
            }
        },
        {
            "CondId": "efg",
            "CondType": 1,
            "Timer": {
                "Days": "1000000", // 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六
                "TimePoint": "09:30" // 触发时间，24小时制，例如"14:00"
            }
        }
    ],
    "Actions": [
        {
            "ActionType": 0, // 0:  则为设备动作，具体参数设置为Data   0 ：设备动作  1：延时 2:场景 3：通知
            "ProductId": "R32***0EU",
            "DeviceName": "df***yY",
            "Data": "{\"brightness\": 25}", // ActionType为 0 ：设备动作参数  为1：延时时间单位秒 2：执行场景Id
3:通知内容
            "AliasName": "12345",
            "IconUrl": "url"
        },
        {
            "ActionType": 1, //1:  则为延时动作，具体延时时间为Data，值为秒
            "Data": "100"
        }
    ],
    "AccessToken": "***v2"
}
```

输出示例：成功

```
{
  "Response": {
    "AutomationId": "a_cd61dd*****d21d6aeffea",
    "RequestId": "req_1"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
LimitExceeded	数量超过限制。

获取自动智能联动列表

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppGetAutomationList）用于获取自动智能联动列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetAutomationList。
FamilyId	String	是	家庭 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
List	Array of AutomationListItem	自动智能联动列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetAutomationList",
    "FamilyId": "f_9****c1",
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
  "Response": {
    "List": [
      {
        "AutomationId": "a_cd61********d21d6aeffea",
        "Icon": "https://www.****.com/",
        "Name": "autoName",
        "Status": 0
      }
    ],
    "RequestId": "req_1"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限

获取自动智能联动详情

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppDescribeAutomation）用于获取自动智能联动详情。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDescribeAutomation。
AutomationId	String	是	自动智能联动 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId
Data	Object of Automation	自动智能联动实体

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "req_1",
  "Action": "AppDescribeAutomation",
  "AutomationId": "a_cd61dd*******7d21d6aeffea",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "AutomationId": "a_cd61d*********d6aeffea",
      "Icon": "https://www.****.com/",
      "Name": "autoName",
      "Status": 0,
      "MatchType": 1,
      "EffectiveBeginTime": "00:00",  // 【两个新增参数，用来表示开始和结束时间】
      "EffectiveEndTime": "23:59",
      "EffectiveDays": "1111111", // 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六
      "Conditions": [
        {
          "CondId": "abc",
          "CondType": 0,
          "Property": {
```

```
            "ProductId": "NJ****LZT4",
            "DeviceName": "gw***v",
            "AliasName": "d****me",
            "IconUrl": "productIcon",
            "PropertyId": "switch",
            "Op": "eq",
            "Value": 1
          }
        },
        {
          "CondId": "efg",
          "CondType": 1,
          "Timer": {
            "Days": "1000000",
            "TimePoint": "09:30"
          }
        }
      ],
      "Actions": [
        {
          "ActionType": 0,
          "ProductId": "R3****0EU",
          "DeviceName": "df****yY",
          "Data": "{\"brightness\": 25}",
          "AliasName": "12345",
          "IconUrl": "url"
        },
        {
          "ActionType": 1,
          "Data": "100"
        }
      ]
    },
    "RequestId": "req_1"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

删除自动智能联动

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppDeleteAutomation）用于获取自动智能联动详情。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppDeleteAutomation。
AutomationId	String	是	自动智能联动 ID。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDeleteAutomation",
    "AutomationId": "a_cd61d**********1d6aeffea",
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。

修改自动智能联动

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppModifyAutomation）用于修改自动智能联动。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyAutomation。
AutomationId	String	是	自动智能联动 ID。
Icon	String	是	新建自动智能联动的背景图片地址。
Name	String	是	自动智能联动的名称。
Status	int	是	自动智能联动开关状态。 • 0：关闭。 • 1：启用。
MatchType	int	是	条件匹配类型。 • 0：全部条件满足。 • 1：其中一个条件满足。
Conditions	Array of Condition	是	自动智能联动触发条件。
Actions	Array of Action	是	自动智能联动动作列表。
EffectiveBeginTime	String	是	生效开始时间。
EffectiveEndTime	String	是	生效结束时间。
EffectiveDays	String	是	生效日期 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppCreateAutomation",
    "AutomationId": "a_cd61ddbe********47d21d6aeffea",
    "Icon": "https://www.****.com/",
```

```
    "Name": "autoName",
    "Status": 0, //联动的开关 0：关闭 1：启用
    "MatchType": 1, //条件匹配类型 0：条件全部与  1：条件全部或
    "EffectiveBeginTime": "00:00",  // 【两个新增参数，用来表示开始和结束时间】
    "EffectiveEndTime": "23:59",
    "EffectiveDays": "1111111", // 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六
    "Conditions": [
        {
            "CondId": "abc", //条件ID，保证在一个联动下唯一即可
            "CondType": 0, //条件类型 0：设备属性值条件  1：定时条件
            "Property": {
                "ProductId": "N****4",
                "DeviceName": "gw**v",
                "AliasName": "devName",
                "IconUrl": "productIcon",
                "PropertyId": "switch", //设备的属性Id
                "Op": "eq", //条件操作符  eq 等于  ne 不等于  gt 大于  lt 小于  ge 大等于  le 小等于
                "Value": 1 //比较的值
            }
        },
        {
            "CondId": "efg",
            "CondType": 1,
            "Timer": {
                "Days": "1000000", // 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六
                "TimePoint": "09:30" // 触发时间，24小时制，比如"14:00"
            }
        }
    ],
    "Actions": [
        {
            "ActionType": 0, // 0：  则为设备动作，具体参数设置为Data   0 ：设备动作  1：延时 2:场景 3：通知
            "ProductId": "R3****0EU",
            "DeviceName": "df****yY",
            "Data": "{\"brightness\": 25}", // ActionType为 0 ：设备动作参数  为1：延时时间单位秒 2：执行场景Id
3:通知内容
            "AliasName": "12345",
            "IconUrl": "url"
        },
        {
            "ActionType": 1, //1：  则为延时动作，具体延时时间为Data，值为秒
            "Data": "100"
        }
    ],
    "AccessToken": "***v2"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。

InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
LimitExceeded	数量超过限制。

修改自动智能联动状态

最近更新时间：2024-10-11 16:01:50

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/appapi` 。
本接口（AppModifyAutomationStatus）用于修改自动智能联动状态。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyAutomationStatus。
AutomationId	String	是	自动智能联动 ID。
Status	int	是	自动智能联动开关状态。 • 0：关闭。 • 1：启用。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetAutomationList",
    "AutomationId": "a_cd61dd******1d6aeffea",
    "Status": 1, //联动的开关 0：关闭 1：启用
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{"Response":{"RequestId":"req_1"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

InvalidParameterValue.NoPermission	用户对该设备无权限

数据查询

获取设备物模型数据

最近更新时间：2024-10-11 16:01:51

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceData）用于获取设备物模型数据。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceData。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Data	String	返回数据。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetDeviceData",
    "ProductId":"22F9Y****O",
    "DeviceName":"light1",
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例: 成功

```
{
    "Response": {
        "Data": "{\"brightness\":{\"Value\":8,\"LastUpdate\":1552904784552},\"color\":
{\"Value\":1,\"LastUpdate\":1552904784552},\"light_switch\":
{\"Value\":0,\"LastUpdate\":1552904784552},\"name\":{\"Value\":\"light1\",\"LastUpdate\":1552904784552}}",
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述

InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

6. 错误信息

名称	类型	描述
Code	String	错误码。
Message	String	错误信息

获取设备物模型历史数据

最近更新时间：2024-10-11 16:01:51

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceDataHistory）用于获取设备物模型历史数据。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceDataHistory。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名。
FieldName	String	是	查询的属性名称。
Context	String	否	翻页游标，首次查询时，可不带。
MinTime	Int	是	开始时间，毫秒时间戳。
MaxTime	Int	是	结束时间，毫秒时间戳。
Limit	Int	是	单页数据量。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
Context	String	翻页游标。
FieldName	String	查询的属性名称。
Listover	Bool	数据是否查询完。 • True 表示未读取完可继续读取。 • False 表示已全部读取完。
Results	Array of DataHistoryItem	返回数据。

4. 示例

示例1

输入示例

```
 POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "Action": "AppGetDeviceDataHistory",
  "ProductId": "22F9Y6II7O",
  "DeviceName": "light1",
```

```
    "MinTime": 1560949017000,
    "MaxTime": 1561035417000,
    "Limit": 20,
    "FieldName": "light_switch",
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
  }
```

输出示例：成功

```
{
    "Response": {
      "Context": "abc",
      "FieldName": "brightness",
      "Listover": false,
      "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
      "Results": [
        {
          "Time": "1560949025494",
          "Value": "77"
        }
      ]
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取设备的历史事件

最近更新时间：2024-10-11 16:01:51

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口(AppListEventHistory)用于获取设备的历史事件。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后,获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppListEventHistory。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Type	String	否	搜索的事件类型：alert 表示告警，fault 表示故障，info 表示信息，为空则表示查询上述所有类型事件。
StartTime	Int64	否	起始时间（Unix 时间戳，秒级），为0表示当前时间。
EndTime	Int64	否	结束时间（Unix 时间戳，秒级），为0表示当前时间。
Context	String	否	搜索上下文，用作查询游标。
Size	Int64	否	单次获取的历史数据项目的最大数量。
EventId	String	否	事件标识符，可以用来指定查询特定的事件，如果不指定，则查询所有事件。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
Context	String	搜索上下文，用作查询游标。
Total	Int64	搜索结果数量。
Listover	Bool	搜索结果是否已经结束。
EventHistory	Array	搜集结果集。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "AccessToken": "accesstoken**********acc",
  "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
  "ProductID": "ABCDE12345",
  "DeviceName": "DEV",
  "StartTime": 0,
```

```
    "EndTime": 0,
    "Type": "info",
    "Size": 5,
    "Context": "",
    "Action": "AppListEventHistory"
}
```

输出示例：成功

```
{
  "Response": {
    "ClientToken": "1ee12***********7df703e0d",
    "OutputParams": "",
    "RequestId": "026fe48b-0c31-4a7c-aaf6-83ba3e9s4bcf",
    "Context": "DnF1ZXJ5VGhlbkZld**************SUQ0UWVPZUlMLXI1QnBkbGc=",
    "EventHistory": [{
          "TimeStamp": 1553484903,
          "ProductId": "product1",
          "DeviceName": "light3",
          "EventId": "status_report",
          "Type": "info",
          "Data": "{\"status\": 0,\"message\": \"test message:11:35:03.288277\"}"
      }],
    "Total": 1
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误
InvalidParameterValue.InvalidAccessToken	Token 无效。

位置服务
获取设备当前位置

最近更新时间：2024-10-11 16:46:21

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`。
本接口（AppGetDeviceLocation）用于获取蜂窝网络设备的当前位置。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceLocation。
DeviceId	String	是	设备 ID，由 ProductId/DeviceName 拼接组成。
CoordType	int	否	坐标系： • 1：代表百度地图坐标系。 • 2：代表高德地图或腾讯地图坐标系。 • 3：代表 wgs84 坐标系。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
Data	Object of Position	位置详情。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetDeviceLocation",
    "DeviceId": "QDA******B/de******01",
    "CoordType": 3
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "Ret": 0,
      "ErrMsg": "",
      "CreateTime": 15990136840000****0,
      "Longitude": 102.83191,
      "Latitude": 24.325504
```

```
    },
    "RequestId": "req_1"
  }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取设备历史位置

最近更新时间：2024-10-11 16:46:21

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetDeviceLocationHistory）用于获取蜂窝网络设备的历史位置。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetDeviceLocationHistory。
DeviceId	String	是	设备 ID，由 ProductId/DeviceName 拼接组成。
MinTime	int	是	查询开始时间（毫秒级 UNIX 时间戳），支持最近1个月范围。
MaxTime	int	是	查询结束时间（毫秒级 UNIX 时间戳），支持最近1个月范围。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
Data	Array of Position	位置列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetDeviceLocationHistory",
    "DeviceId": "QDA******B/de******01",
    "MaxTime": 0,
    "MinTime": 0
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "Ret": 0,
      "ErrMsg": "",
      "Pos": [{
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013554000000000
```

```
      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013569000000000
      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013574000000000
      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013579000000000
      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013604000000000
      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013609000000000

      }, {
        "Location": {
          "Longitude": 102.83191,
          "Latitude": 24.325504
        },
        "CreateTime": 1599013684000000000
      }]
    },
    "RequestId": "aaabbbb2"

  }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

创建设备地理围栏

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppCreateFence）用于创建设备地理围栏。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppCreateFence。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
FenceName	String	是	设备地理围栏名称。
FenceArea	String	是	设备地理围栏区域范围。
FenceDesc	String	是	设备地理围栏描述。
AlertCondition	String	是	触发条件。
FenceEnable	Boolean	是	启用状态。
Method	String	是	通知方式。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
FenceId	string	设备地理围栏 ID。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppCreateFence",
    "ProductId":"NJ****T4",
    "DeviceName":"g****v",
    "FenceName":"appFence",
    "FenceArea": "{\"type\":\"FeatureCollection\",\"features\":[{\"type\":\"Feature\",\"geometry\":
{\"type\":\"Polygon\",\"coordinates\":[[[113.9498,22.5319],[113.9709,22.5319],[113.9709,22.5723],
[113.9498,22.5723],[113.9498,22.5319]]]},\"properties\":null}]}",
    "FenceDesc": "test",
    "AlertCondition":"leave",
    "FenceEnable": true,
    "Method":"app push",
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
```

```
}
```

输出示例：成功

```
{
    "Response": {
        "FenceId":7,
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.FenceDupKeyExist	存在重复围栏。
UnauthorizedOperation.SpaceDupKeyExist	存在重复位置空间。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除

获取设备围栏告警事件列表

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFenceEventList）用于获取设备地理围栏告警事件列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFenceEventList。
FenceId	Int64	是	设备地理围栏 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
FenceEnable	Boolean	是	启用状态。
StartTime	Int	是	起始时间（Unix 时间戳，秒级），为0表示当前时间-24h。
EndTime	Int	是	结束时间（Unix 时间戳，秒级），为0表示当前时间。
Offset	Int	否	默认值为0，消息偏移量。
Limit	Int	否	默认值为0，最大返回消息条数，最大值为 50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
List	Array of DeviceFenceEvent	设备地理围栏告警事件列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetFenceEventList",
    "ProductId": "NJ2****ZT4",
    "FenceId": 8,
    "DeviceName":"g****",
    "StartTime": 1604307550000,
    "EndTime": 1604309654000,
    "FenceEnable": true,
    "Offset": 0,
    "Limit": 10,
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
```

```
}
```

输出示例：成功

```
{
    "Response": {
        "List": [
            {
                "EventType": "enter"
                "CreateTime": 1604314065
            }
        ],
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.FenceNotExist	围栏未创建或是已删除。

获取设备地理围栏列表

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppGetFenceList）用于获取设备地理围栏列表。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppGetFenceList。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
Offset	Int	否	消息偏移量。
Limit	Int	否	最大返回消息条数，最大值为 50。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
List	Array of Fence	设备地理围栏列表。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppGetFenceList",
    "ProductId": "NJ****T4",
    "DeviceName": "gw****ev",
    "Offset": 0,
    "Limit": 10,
    "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "List": [
            {
                "FenceId": 8,
                "FenceName": "appFence",
                "FenceDesc": "test",
```

```
                "FenceArea": "{\"type\":\"FeatureCollection\",\"features\":
[{\"type\":\"Feature\",\"geometry\":{\"type\":\"Polygon\",\"coordinates\":[[[113.9498,22.5319],
[113.9709,22.5319],[113.9709,22.5723],[113.9498,22.5723],[113.9498,22.5319]]]},\"properties\":null}]}",
                "AlertCondition": "leave",
                "FenceEnable": true,
                "Method": "app push",
                "CreateTime": 1604314065,
                "UpdateTime": 1604314065
            }
        ],
        "RequestId": "req_1"
    }
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

删除设备地理围栏

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppDeleteFence）用于删除设备地理围栏。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppDeleteFence。
FenceId	Int64	是	设备地理围栏 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
AlertCondition	String	是	触发条件。
FenceEnable	Boolean	是	启用状态。
Method	String	是	通知方式。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppDeleteFence",
    "ProductId": "NJ****T4",
    "FenceId": 7,
    "DeviceName": "gw****ev",
    "AlertCondition": "enter",
    "FenceEnable": true,
    "Method": "app push",
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.FenceDupKeyExist	存在重复围栏。
UnauthorizedOperation.SpaceDupKeyExist	存在重复位置空间。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.FenceNotExist	围栏未创建或是已删除。

修改设备地理围栏

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyFence）用于修改设备地理围栏。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyFence。
FenceId	Int64	是	设备地理围栏 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
FenceName	String	是	设备地理围栏名称。
FenceArea	String	是	设备地理围栏区域范围。
FenceDesc	String	是	设备地理围栏描述。
AlertCondition	String	是	触发条件。
FenceEnable	Boolean	是	启用状态。
Method	String	是	通知方式。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppModifyFence",
    "ProductId": "NJ****T4",
    "FenceId": 7,
    "DeviceName": "gw****ev",
    "FenceName": "appFence",
    "FenceArea": "{\"type\":\"FeatureCollection\",\"features\":[{\"type\":\"Feature\",\"geometry\":
{\"type\":\"Polygon\",\"coordinates\":[[[113.9498,22.5319],[113.9709,22.5319],[113.9709,22.5723],
[113.9498,22.5723],[113.9498,22.5319]]]},\"properties\":null}]}",
    "FenceDesc": "test",
    "AlertCondition": "enter",
    "FenceEnable": true,
    "Method": "app push",
```

```
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.FenceDupKeyExist	存在重复围栏。
UnauthorizedOperation.SpaceDupKeyExist	存在重复位置空间。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.FenceNotExist	围栏未创建或是已删除。

修改设备地理围栏启用状态

最近更新时间：2024-10-11 16:46:22

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi` 。
本接口（AppModifyFenceStatus）用于修改设备地理围栏启用状态。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppModifyFenceStatus。
FenceId	Int64	是	设备地理围栏 ID。
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。
AlertCondition	String	是	触发条件。
FenceEnable	Boolean	是	启用状态。
Method	String	是	通知方式。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "req_1",
    "Action": "AppModifyFenceStatus",
    "ProductId": "NJ****T4",
    "FenceId": 7,
    "DeviceName": "gw****ev",
    "AlertCondition": "enter",
    "FenceEnable": true,
    "Method": "app push",
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "req_1"
    }
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InternalError.InternalServerExceptionDB	DB 错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.NoPermission	用户对该设备无权限。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。
UnauthorizedOperation.FenceDupKeyExist	存在重复围栏。
UnauthorizedOperation.SpaceDupKeyExist	存在重复位置空间。
ResourceNotFound	资源不存在。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.FenceNotExist	围栏未创建或是已删除。

文件管理
查询设备指定文件信息

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`
本接口（AppDescribeDeviceResource）用于查询设备的指定文件信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后,获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppDescribeDeviceResource
DeviceId	String	是	设备 ID。
ResourceName	String	是	厂商侧的文件名称。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
Resource	OBJECT	文件的详细信息。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "1pZb3sz1N",
    "Action": "AppDescribeDeviceResource",
    "DeviceId": "TOIDHQ3AOQ/light100",
    "ResourceName": "test"
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
  {"Response":{"Resource":
{"ProductId":"TOIDHQ3AOQ","DeviceName":"light100","ResourceName":"test","ResourceType":"jpg","ResourceVer"
:"0.0.1","Status":0,"CreateTime":1576747101,"UpdateTime":1576747523},"RequestId":"1pZb3sz1N"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。

InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取下载文件

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppGetResourceDownloadUrl）用于获取文件下载地址。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppGetResourceDownloadUrl
ProductId	String	是	产品 ID
UserResourceName	String	是	用户侧的文件名称。
EffectiveTime	Int	是	有效时间。
ResourceVer	String	是	文件的版本。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
DownloadUrl	String	下载的 URL。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "1pZb113sz1N",
    "Action": "AppGetResourceDownloadUrl",
    "ProductId": "BKTTNGIQOG",
    "UserResourceName": "restest",
    "ResourceVer": "0.0.1",
    "EffectiveTime": 1626329512,
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
  {"Response":
    {"DownloadUrl":""https://gz-g-resource-1256872341.cos.ap-
guangzhou.myqcloud.com/res%2F96*************************7f1135e7"",
     "RequestId":"1pZb113sz1N"}
```

```
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

获取文件上传地址

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppGetResourceUploadURL）用于 获取文件上传地址。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后,获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppGetResourceUploadURL
ProductId	String	是	产品 ID
UserResourceName	String	是	用户侧的文件名称。
FileSize	Int	是	文件大小
ResourceVer	String	是	文件的版本。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
UploadUrl	String	上传的 URL。
ResourceName	String	厂商侧的文件名称。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
 "RequestId": "1pZb113sz1N",
 "Action": "AppGetResourceUploadURL",
 "ProductId": "BKTTNGIQOG",
 "UserResourceName": "restest",
 "ResourceVer": "0.0.1",
 "FileSize": 5179,
 "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{"Response":
  {"UploadUrl":"https://gz-g-resource-1256872341.cos.ap-
guangzhou.myqcloud.com/res%2F96*************************7f1135e7",
```

```
    "ResourceName":"USER_72375312314273792_RES_restest","RequestId":"1pZb113sz1N"}
}
```


5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

上传后创建文件

最近更新时间：2024-11-01 16:30:32

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppCreateProductResource）用于上传后创建(登记)文件。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppCreateProductResource
ProductId	String	是	产品 ID
UserResourceName	String	是	用户侧的文件名称。
FileSize	Int	是	文件的文件字节大小。
ResourceVer	String	是	文件的版本。
FileHash	String	是	文件的 hash 值。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。
ResourceName	String	厂商侧的文件名称。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
    "RequestId": "1pZb3sz1N",
    "Action": "AppCreateProductResource",
    "ProductId": "BKTTNGIQOG",
    "UserResourceName": "restest",
    "ResourceVer": "0.0.1",
    "FileHash": "2a8116bfda285cecd7**********cc0f",
    "FileSize": 5179,
    "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{"Response":
  {"ResourceName": "USER_7237531231*****92_RES_restest",
```

```
    "RequestId": "1pZb3sz1N"}}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

下发文件至设备

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名：`iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppControlDeviceResource）用于下发文件到设备。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，用户通过微信号、手机或邮箱账号登录成功后，获取的访问 Token。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	AppControlDeviceResource
DeviceId	String	是	设备 ID
ResourceName	String	是	厂商侧的文件名称。
ResourceVer	String	是	文件的版本。
Method	String	是	控制命令，只能为 update 或者 delete。下发或者升级填 update；删除则填 delete。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json

{
  "RequestId": "1pZb3sz1N",
  "Action": "AppControlDeviceResource",
  "DeviceId":"BKTTNGIQOG/dev01",
  "ResourceName":"test",
  "ResourceVer":"0.0.1",
  "Method":"update",
  "AccessToken": "8b4a70dd16105f*****************18edd4e78a3bb8ec"
}
```

输出示例：成功

```
{"Response":{"RequestId":"1pZb3sz1N"}}
```

5. 错误码

错误码	描述

InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。

长连接通信
注册监听

最近更新时间：2024-10-05 15:23:11

1. 接口描述

接口请求域名： `wss://iot.cloud.tencent.com/iotstudio_v2_weapp_1`
本接口（DeviceRegistrationMonitor）用于向 WS 服务端进行订阅，订阅成功后可以通过 `ws.onmessage` **的监听获取到 WS 服务端实时推送设备上下线状态及属性数据。**

2. 输入参数

名称	类型	必选	描述
action	String	是	公共参数，本接口取值：ActivePush。
reqId	String	是	公共参数，标示请求序列,唯一请求 ID，可自行生成。
params	Object of ActionPushParam	是	注册设备的时候传入的参数。

3. 输出参数

名称	类型	描述
reqId	String	公共参数，标示请求序列，与入参相同。
error	String	状态码，为空表示正确。
error_message	String	错误详细说明，为空表示正确。
data	Object of ActionPushData	监听之后的服务器回包。

4. 示例

示例1

输入示例（发送前需将数据转换为 json 字符串）

```
const ws = new WebSocket("wss://iot.cloud.tencent.com/iotstudio_v2_weapp_1")

ws.send(JSON.stringify({
  "action": "ActivePush",
  "reqId": "req0001",
  "params":{
    "DeviceIds": ["产品ID/DeviceName","产品ID/DeviceName"],
    "AccessToken": "f6******************3aad"
  }
}));
```

输出示例：成功

```
{
  "error":"",
  "error_message":"",
  "data":{
    "deviceIds":["产品ID/DeviceName","产品ID/DeviceName"]
  },
  "reqId":"req0001"
}
```

输出示例：失败

```
{
  "error":"",
  "error_message":"",
  "data":{
    "deviceIds":[]
  },
  "reqId":"req0001"
}
```

心跳

最近更新时间：2024-09-30 17:54:51

1. 接口描述

接口请求域名： `wss://iot.cloud.tencent.com/iotstudio_v2_weapp_1`

设备保活，通过该接口更新设备时间戳，保持设备监听状态，同时保持 WebSocket 连接。可以多次调用，每60s调用一次。

2. 输入参数

名称	类型	必选	描述
action	String	是	公共参数，本接口取值：YunApi。
reqId	String	是	公共参数，标示请求序列，唯一请求 ID，可自行生成。
params	Object	是	调用接口传入的详细参数，请参见 YunApiParam 。

3. 输出参数

名称	类型	描述
reqId	String	公共参数，表示请求序列，与入参相同。
error	String	状态码,为空表示正确。
error_message	String	错误详细说明，为空表示正确。
data	Object	心跳接口的服务器回包，请参见 HeartBeatData 。

4. 示例

输入示例（发送前需将数据转换为 json 字符串）

```
const ws = new WebSocket("wss://iot.cloud.tencent.com/iotstudio_v2_weapp_1")

ws.send(JSON.stringify({
    "action":"YunApi",
    "reqId":"req0001",
    "params":{
        "Action": "AppDeviceTraceHeartBeat",
        "ActionParams": {
            "DeviceIds": ["产品ID/DeviceName","产品ID/DeviceName"],
            "AccessToken": "f6******************3aad",
            "RequestId":"rest-client"
        }
    }
}))
```

输出示例：成功

```
{
    "error":"",
    "error_message":"",
    "data":{
        "Response": {
            "RequestId":"rest-client"
        }
    },
    "reqId":"req0001"
```

```
}
```

输出示例：失败（服务器错误）

```
{
    "error":"-1",
    "error_message":"后台连接出错，请联系管理员",
    "data":{},
    "reqId":"req0001"
}
```

输出示例：失败（逻辑错误）

```
{
    "error":"",
    "error_message":"",
    "data":{
        "Response": {
            "Error":{
                "Code": "UnauthorizedOperation.APPNoPermissionToStudioProduct",
                "Message": "App对操作该产品无权限"
            },
            "RequestId":"rest-client"
        }
    },
    "reqId":"req0001"
}
```

设备状态推送

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名：`wss://iot.cloud.tencent.com/iotstudio_v2_weapp_1`

设备状态推送用于在小程序或 App 实时获取用户绑定设备的上下线状态、设备上报的属性与事件以及设备行为执行结果。需要成功调用注册设备监听接口，通过监听 WebSocket 的 OnMessage 获取设备状态与属性值的实时推送，若获取到的 `event.data.push` 为 `true`，则代表该条消息为设备状态变更的主动推送。

2. 输入参数

名称	类型	描述
push	Boolean	该条消息为设备状态变更的主动推送。
action	String	固定为 `DeviceChange`，代表设备状态变更。
params	String	返回设备信息，详细说明见 消息类型说明 。

3. 输出参数

名称	类型	描述
RequestId	String	请求 ID。

4. 示例

应用端 ws 订阅设备状态变化

请求示例

```
ws.onmessage = (event) => {
  let data;
  try {
    data = JSON.parse(event.data);
  } catch (e) {
    console.log(`onMessage parse event.data error: ${event.data}`);
    return;
  }

  if (data.push) {
    this.emit('push', data);
  } else if (typeof data.reqId !== 'undefined' && this.requestHandlerMap.has(data.reqId)) {
    this.requestHandlerMap.get(data.reqId)(null, data);
  }
};
```

输入参数示例：

```
{
    "action": "DeviceChange",
    "params": {
        "Time": "2023-07-13T14:19:48+08:00",
        "Type": "Property",
        "SubType": "Report",
        "Topic": "$thing/up/property/productId/deviceName",
        "Payload": "ey******iOjQwfX0=",
        "Seq": 1689229188406,
```

```
        "DeviceId": "productId/deviceName"
    },
    "push": true
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "up03"
    }
}
```

5. params 参数说明

参数名称	参数类型	说明
Time	String	时间，UTC 格式，如：2023-07-13T14:19:48+08:00。
Type	String	消息类型，详见 Type 类型说明 。
SubType	String	消息子类型，详见 SubType 类型说明 。
Topic	String	消息 Topic，可登录 物联网开发平台控制台 进入目标产品开发页，选择**设备开发 > Topic 列表**来查看。
Payload	String	消息体 base64 编码内容。详见 Payload 说明 。
Seq	Int64	序列号。
DeviceId	String	设备 ID，产品 ID/deviceName 格式。

Type 类型说明

Type 类型	描述
StatusChange	设备状态变化消息，包括设备上线与下线消息通知。
Shadow	影子消息。
Property	属性消息。
Template	物模型消息。
Event	事件消息。
Action	行为消息。
UserDefined	用户自定义。

SubType 说明

SubType 类型	描述
Online	设备上线。
Offline	设备下线。
Report	设备上报。
Push	服务端推送到设备。

Payload 说明

Payload 是消息体 base64 编码后字符串。示例如下：

消息体：

```
{
    "method": "control",
    "clientToken": "12345678",
    "params": {
        "switch": 0
    }
}
```

base64编码：

```
"Payload": "eyJtZXRob2QiOiJjb250cm9sIiwiY2xpZW50VG9rZW4iOiIxMjM0NTY3OCIsInBhcmFtcyI6eyJzd2l0Y2giOjB9fQ=="
```

Template 消息示例

物模型消息 Topic 为空，Payload 是消息 base64 编码后内容。

```
{
    "action": "DeviceChange",
    "params": {
        "Time": "2023-07-13T15:00:11+08:00",
        "Type": "Template",
        "SubType": "Push",
        "Topic": "",
        "Payload":
"eyJtZXRob2QiOiJjb250cm9sIiwiY2xpZW50VG9rZW4iOiIxMjM0NTY3OCIsInBhcmFtcyI6eyJzd2l0Y2giOjB9fQ==",
        "Seq": 1689231611907,
        "DeviceId": "productId/deviceName"
    },
    "push": true
}
```

Payload base64 解码消息体

详细说明见 物模型 。

```
{
    "method": "control",
    "clientToken": "12345678",
    "params": {
        "switch": 0
    }
}
```

Event 消息示例

```
{
    "Time": "2019-04-11T20:42:36+08:00",
    "Type": "Event",
    "SubType": "Report",
    "Topic": "$thing/up/event/US4CJ11DIK/LIAO",
    "Payload":
"ewogICAgICAgICJtZXRob2QiOiAiZXZlbnRfcG9zdCIsCiAgICAgICAgInZlcnNpb24iOiAiMS4wIiwKICAgICAgICAiZXZlbnRJZCI6I
CJsb3dfdm9sdGFnZSIsCiAgICAgICAgInBhcmFtcyI6IHsKICAgICAgICAgICAgInZvbHRhZ2UiOiAyLjI3ODA4MQogICAgICAgIH0sCiA
gICAgICAgImNsaWVudFRva2VuIjogImNsaWVudFRva2VuLTU1OSJ9",
    "Seq": 300,
```

```
    "DeviceId": "US4CJ11DIK/LIAO"
  }
```

Payload base64 解码消息体

```
  {
    "method": "event_post",
    "version": "1.0",
    "eventId": "low_voltage",
    "params": {
      "voltage": 2.278081
    },
    "clientToken": "clientToken-559"
  }
```

属性消息示例

```
  {
    "Time": "2019-10-30T10:41:00+08:00",
    "Type": "Property",
    "SubType": "Report",
    "Topic": "$thing/up/property/WTJTZX0MF2/device_property001",
    "Payload":
"eyJ0aW1lc3RhbXAiOiAwLCAicGFyYW1zIjogeyJicmlnaHRuZXNzIjogNTl9LCAibWV0aG9kIjogInJlcG9ydCIsICJjbGllbnRUb2tlbiI6ICJ0bTIwMTkxMDMwMTA0MDUzXzEifQ==",
    "Seq": 6,
    "DeviceId": "WTJTZX0MF2/device_property001"
  }
```

Payload base64 解码消息体

```
  {
    "method": "report",
    "params": {
      "timestamp": 0,
      "params": {
        "brightness": 59
      },
      "method": "report",
      "clientToken": "tm20191030104053_1"
    }
  }
```

StatusChange 消息示例

设备上线消息。

```
  {
    "Time": "2019-03-28T15:17:33+08:00",
    "Type": "StatusChange",
    "SubType": "Online",
    "Topic": "",
    "Payload": "",
    "Seq": 6
  }
```

设备下线。

```
{
  "Time": "2019-03-28T15:17:33+08:00",
  "Type": "StatusChange",
  "SubType": "Offline",
  "Topic": "",
  "Payload": "",
  "Seq": 7
}
```

Action 行为示例

Action 消息推送示例

```
{
  "Time": "2021-05-13T21:34:05+08:00",
  "Type": "Action",
  "SubType": "Push",
  "Topic": "",
  "Payload":
"eyJtZXRob2QiOiJhY3Rpb24iLCJjbGllbnRUb2tlbiI6IjEzNjk0NzQ5Mzo6cmVxXzEiLCJhY3Rpb25JZCI6ImxpZ2h0X2JsaW5rIiwid
GltZXN0YW1wIjoxNjIwOTEyODQ1LCJwYXJhbXMiOnsiY29sb3IiOjEsInRpbWUiOjYsInRvdGFsX3RpbWUiOjYwfX0K",
  "Seq": 1620912845,
  "DeviceId": "0CPSFIRP28/testble"
}
```

Payload base64 解码消息体

```
{
  "method": "action",
  "clientToken": "136947493::req_1",
  "actionId": "light_blink",
  "timestamp": 1620912845,
  "params": {
    "color": 1,
    "time": 6,
    "total_time": 60
  }
}
```

Action 消息 report 示例

```
{
  "Time": "2021-05-13T21:34:05+08:00",
  "Type": "Action",
  "SubType": "Report",
  "Topic": "",
  "Payload":
"eyJtZXRob2QiOiJhY3Rpb25fcmVwbHkiLCJjbGllbnRUb2tlbiI6IjEzNjk0NzQ5Mzo6cmVxXzEiLCJjb2RlIjowLCJzdGF0dXMiOiJzd
WNjIiwicmVzcG9uc2UiOnsicmVzdWx0IjogMX19",
  "Seq": 1620912845,
  "DeviceId": "0CPSFIRP28/testble"
}
```

Payload base64 解码消息体

```
{
  "method": "action_reply",
  "clientToken": "136947493::req_1",
```

```
    "code": 0,
    "status": "succ",
    "response": {
      "result": 1
    }
  }
```

云存服务
拉取云存事件列表

最近更新时间：2024-05-11 14:15:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`
本接口（IotVideoDescribeCloudStorageEvents）用于拉取云存事件列表。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoDescribeCloudStorageEvents。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
StartTime	Integer	否	起始时间（Unix 时间戳，秒级），为0 表示 当前时间 - 24h
EndTime	Integer	否	结束时间（Unix 时间戳，秒级），为0 表示当前时间
Context	String	否	请求上下文, 用作查询游标
Size	Integer	否	查询数据项目的最大数量, 默认为10。假设传Size=10，返回的实际事件数量为N，则 5 <= N <= 10
EventId	String	否	事件标识符，可以用来指定查询特定的事件，如果不指定，则查询所有事件
UserId	String	否	用户ID
ChannelId	Integer	否	通道ID 非NVR设备则不填 NVR设备则必填 默认为无

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
Events	Array of CloudStorageEvent	云存事件列表
Context	String	请求上下文, 用作查询游标
Listover	Boolean	拉取结果是否已经结束
Total	Integer	内部结果数量，并不等同于事件总数
VideoURL	String	视频播放URL

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
```

```
    "Action": "AppDescribeTemplateBinding",
    "AccessToken":"bf*****************d098",
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
    "ProductId": "PRODUCT_ID",
    "DeviceName": "DEVICENAME"
  }
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "bd3ad0f6-5b9f-47cd-83db-223356926772",
        "Total": 13,
        "Events": [
            {
                "StartTime": 842084401,
                "EndTime": 842084401,
                "Thumbnail": "",
                "EventId": "id1_data"
            },
            {
                "StartTime": 842084401,
                "EndTime": 842084401,
                "Thumbnail": "",
                "EventId": "id1_data"
            },
            {
                "StartTime": 842084401,
                "EndTime": 842084401,
                "Thumbnail": "",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
```

```
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            },
            {
                "StartTime": 1614847697,
                "EndTime": 167000000,
                "Thumbnail": "face_dection.jpg",
                "EventId": "id1_data"
            }
        ],
        "Context":
"DnF1ZXJ5VGhlbkZldGNoKAAAAAABxjxlFmlDUHBXME5XVE55eUdHMDUwZTV4WWcAAAAAAcaq9BZFeGQwTXM5X1RvNnFRMFNhRm1MNTlRA
AAAAAHGqvYWRXhkME1zOV9UbzZxUTBTYUZtTDU5UQAAAAABxk0BFnN5Q1hWNUp0UjcyZkZma2licmF3SmcAAAAAAcaq9RZFeGQwTXM5X1R
vNnFRMFNhRm1MNTlRAAAAAAHGhvUWTzl2c3FBNVdUWnFSWXh4d012MGlzdwAAAAABxjxmFmlDUHBXME5XVE55eUdHMDUwZTV4WWcAAAAAA
caG9xZPOXZzcUE1V1RacVJZeHh3TXYwaXN3AAAAAAHGVbkWTVBJSndzX0VReVNoU3JOb2pWUm40UQAAAAABxk0AFnN5Q1hWNUp0UjcyZkZ
ma2licmF3SmcAAAAAAcaG-
BZPOXZzcUE1V1RacVJZeHh3TXYwaXN3AAAAAAHGqvgWRXhkME1zOV9UbzZxUTBTYUZtTDU5UQAAAAABxkfDFjdNdGYtaTR5UTBPNmFFcHk
yUmlUVHcAAAAAAcY88hZOVnkwbjNfNFRqV2YwSWk1LWczc2V3AAAAAAHGhvYWTzl2c3FBNVdUWnFSWXh4d012MGlzdwAAAAABxqr3FkV4Z
DBNczlfVG82cVEwU2FGbUw1OVEAAAAAAcaG-
RZPOXZzcUE1V1RacVJZeHh3TXYwaXN3AAAAAAHGR8IWN010Zi1pNHlRME82YUVweTJSaVRUdwAAAAABxquSFnJKTjhzbk5mUmZxMVVCcVh
pVUNTZFEAAAAAAcY8ZxZpQ1BwVzBOV1ROeXlHRzA1MGU1eFlnAAAAAAHGehEWTTRRa1JkRDhUek9qem1CSmNFbHhUZwAAAAABxqr5FkV4Z
DBNczlfVG82cVEwU2FGbUw1OVEAAAAAAcZNBBZzeUNYVjVKdFI3MmZGZmtpYnJhd0pnAAAAAAHGVboWTVBJSndzX0VReVNoU3JOb2pWUm4
0UQAAAAABxnoQFk00UWtSZEQ4VHpPanptQkpjRWx4VGcAAAAAAcarkxZySk44c25OZlJmcTFVQnFYaVVDU2RRAAAAAAHGTQIWc3lDWFY1S
nRSNzJmRmZraWJyYXdKZwAAAAABxpZ4Fno4eVNTTDlLUVUyblVfMFFFcnhramcAAAAAAcY88xZOVnkwbjNfNFRqV2YwSWk1LWczc2V3AAA
AAAHGTQMWc3lDWFY1SnRSNzJmRmZraWJyYXdKZwAAAAABxkfEFjdNdGYtaTR5UTBPNmFFcHkyUmlUVHcAAAAAAcarlBZySk44c25OZlJmc
TFVQnFYaVVDU2RRAAAAAAHGqvoWRXhkME1zOV9UbzZxUTBTYUZtTDU5UQAAAAABxpZ6Fno4eVNTTDlLUVUyblVfMFFFcnhramcAAAAAAcZ
HxRY3TXRmLWk0eVEwTzZhRXB5MlJpVFR3AAAAAAHGVbsWTVBJSndzX0VReVNoU3JOb2pWUm40UQAAAAABxpZ5Fno4eVNTTDlLUVUyblVfM
FFFcnhramcAAAAAcY8aBZpQ1BwVzBOV1ROeXlHRzA1MGU1eFlnAAAAAAHGqvsWRXhkME1zOV9UbzZxUTBTYUZtTDU5UQAAAAABxpZ7Fno
4eVNTTDlLUVUyblVfMFFFcnhramc=",
        "Listover": false,
        "VideoURL": "http://test.iotvideo.tencentcs.com/timeshift/live/test.m3u8"
    }
}
```


输出示例：失败


```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```


5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

拉取云存事件缩略图

最近更新时间：2024-11-01 09:43:32

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（IotVideoDescribeCloudStorageThumbnail）用于拉取云存事件缩略图。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoDescribeCloudStorageThumbnail。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
Thumbnail	String	是	缩略图文件名

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
ThumbnailURL	String	缩略图访问地址

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME",
  "Thumbnail": "/100008401725/AQTV2839QJ/sp01_32820237_7/events/1615200613.jpg"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "c299d2c9-805f-41a5-853d-db32ba1a16b5",
        "ThumbnailURL": "https://video-cv-1258344699.cos.ap-
guangzhou.myqcloud.com/%2F100008401725%2FAQTV2839QJ%2Fsp01_32820237_5%2Fevents%2F1614908216.jpg?q-sign-
algorithm=sha1&q-ak=**********************************&q-sign-time=1614909378%3B1614912978&q-key-
time=1614909378%3B1614912978&q-header-list=host&q-url-param-list=&q-
signature=59cfa8d69bb77980ba757cb111de62fc6c4880f5"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

获取具有云存的日期

最近更新时间：2024-05-11 14:15:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（IotVideoDescribeCloudStorageDate）用于获取具有云存的日期。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoDescribeCloudStorageDate。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
UserId	String	否	用户ID

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
Data	Array of String	云存日期数组，示例值：["2021-01-05","2021-01-06"]

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "77d90d07-e48c-4cf0-bec5-afe5b137f80e",
        "Data": [
            "2021-01-14"
        ]
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

获取某一天云存时间轴

最近更新时间：2024-05-11 14:15:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（IotVideoDescribeCloudStorageTime）用于获取某一天云存时间轴。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoDescribeCloudStorageTime。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
Date	String	是	云存日期，例如"2020-01-05"
StartTime	Integer	否	开始时间，unix时间
EndTime	Integer	否	结束时间，unix时间
UserId	String	否	用户ID

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
Data	CloudStorageTimeData	返回数据

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME",
  "Date": "2021-01-14"
}
```

输出示例：成功

```
{
    "Response": {
        "RequestId": "ac74ecca-b866-4ab1-87e0-18b92816fb20",
        "Data": {
            "VideoURL": "47b9d579-9088-4cc2-a16e-2b8cf245319b",
```

```
            "TimeList": [
                {
                    "StartTime": 1610607600,
                    "EndTime": 1610608200
                },
                {
                    "StartTime": 1610607600,
                    "EndTime": 1610608200
                },
                {
                    "StartTime": 1610607600,
                    "EndTime": 1610608200
                },
                {
                    "StartTime": 1610607600,
                    "EndTime": 1610608200
                }
            ]
        }
    }
}
```


输出示例：失败


```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```


5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

获取视频防盗链播放 URL

最近更新时间：2024-05-11 14:15:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（IotVideoGenerateSignedVideoURL）用于获取某一天云存时间轴。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoGenerateSignedVideoURL。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
VideoURL	String	是	视频播放原始URL地址
ExpireTime	Integer	是	播放链接过期时间

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
SignedVideoURL	String	视频防盗链播放URL

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME",
  "ExpireTime": 1619331648,
  "VideoURL": "http://zylcb.iotvideo.tencentcs.com/timeshift/live/1.m3u8"
}
```

输出示例：成功

```
{
    "Response": {
        "SignedVideoURL": "http://zylcb.iotvideo.tencentcs.com/timeshift/live/1.m3u8?sign=x",
        "RequestId": "xx"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

拉取图片流数据

最近更新时间：2024-05-11 14:15:22

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（IotVideoDescribeCloudStorageStreamData）用于获取某一天云存时间轴。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：IotVideoDescribeCloudStorageStreamData。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。
StartTime	Integer	是	起始时间

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
VideoStream	String	图片流视频地址
AudioStream	String	图片流音频地址

4. 示例

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME",
  "StartTime": 1619331648
}
```

输出示例：成功

```
{
    "Response": {
        "VideoStream": "http://xxxxxxxx",
        "AudioStream": "http://xxxxxxxx",
        "RequestId": "xx"
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

音乐服务
歌单歌曲

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（playlist_song）用于获取歌单歌曲。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：playlist_song。
DeviceId	String	是	设备 Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_playlist_song	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_playlist_song	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"playlist_song",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

获取歌曲播放链接

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（song_url）用于获取歌曲播放链接。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：song_url。
DeviceId	String	是	设备 Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_song_url	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Interger	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_song_url	返回结果

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"song_url",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {...
        },
```

```
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

每日推荐

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（awesome_everyday）用于获取每日推荐歌单。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：awesome_everyday。
DeviceId	String	是	设备Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_awesome_everyday	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_awesome_everyday	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken": "8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId": "LAEG4YJE1A/dev01",
  "KGCommand": "awesome_everyday",
  "KugouParams": {...}
}
```

输出示例：成功

```
{
```

```
    "Response": {
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

批量查询歌曲信息

最近更新时间：2024-10-08 18:37:11

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（song_infos）用于批量查询歌曲信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：song_infos。
DeviceId	String	是	设备Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_song_infos	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_song_infos	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"song_infos",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
    "Response": {
```

```
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

推荐歌单

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口（playlist_awesome）用于获取推荐歌单。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：playlist_awesome。
DeviceId	String	是	设备 Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_playlist_awesome	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of data_playlist_awesome	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"playlist_awesome",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
```

```
    "Response": {
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

新歌首发

最近更新时间：2024-09-30 17:54:51

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口(awesome_newsong)用于获取首发新歌。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：awesome_newsong。
DeviceId	String	是	设备Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_awesome_newsong	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_awesome_newsong	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"awesome_newsong",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
```

```
    "Response": {
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定，请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

用户登出

最近更新时间：2024-11-14 15:47:32

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口(user_logout)用于用户登出。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：user_logout。
DeviceId	String	是	设备 Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_user_logout	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of data_user_logout	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"user_logout",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
    "Response": {
```

```
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定，请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

用户信息

最近更新时间：2024-10-11 16:46:24

1. 接口描述

接口请求域名： iot.cloud.tencent.com/api/exploreropen/tokenapi
本接口(user_info)用于获取用户信息。

2. 输入参数

名称	类型	必选	描述
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
Action	String	是	公共参数，本接口取值：AppKugouUserCommand。
KGCommand	String	是	子命令，本接口取值：user_info。
DeviceId	String	是	设备 Id 由产品 ID 和设备 Name 组成。
KugouParams	Object of KugouParams_user_info	是	子命令的参数。

3. 输出参数

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。
ErrorCode	Integer	请求返回的错误码，0为没有错误。
ErrorMsg	String	请求返回的错误信息。
Data	Object of Data_user_info	返回结果。

4. 示例

示例1

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "RequestId": "test0191119-012345",
  "Action": "AppKugouUserCommand",
  "AccessToken":"8b4a70dd16105f******************18edd4e78a3bb8ec",
  "DeviceId":"LAEG4YJE1A/dev01",
  "KGCommand":"user_info",
  "KugouParams":{...}
}
```

输出示例：成功

```
{
    "Response": {
```

```
        "Data": {...
        },
        "ErrorCode":0,"ErrorMsg":"",
        "RequestId": "keyi20191119-012345"
    }
}
```

5. 错误码

错误码	描述
InternalError	内部错误。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.InvalidAccessToken	Token 无效。
InvalidParameterValue.InternalRPCError	调用超时。
ResourceNotFound	资源不存在。
ResourceNotFound.DeviceNotExist	设备未创建或是已删除。
ResourceNotFound.StudioProductNotExist	产品尚未创建或已被删除。
ResourceNotFound.AppNotExists	应用未创建或是已删除。
UnsupportedOperation	不支持的操作。
UnsupportedOperation.DeviceNotGateway	不是网关设备。
UnsupportedOperation.NotBindVirtualDevice	无法绑定虚拟设备。
UnsupportedOperation.DeviceNotBind	子设备未绑定到网关设备。
UnsupportedOperation.CannotBindSameFamily	该设备已经绑定请勿重复绑定。
LimitExceeded	超出限制。
LimitExceeded.FamilyDeviceExceedLimit	设备数量超出限制。
UnauthorizedOperation	无权操作。
UnauthorizedOperation.APPNoPermissionToStudioProduct	App 对操作该产品无权限。

微信强提醒
查询已经绑定的第三方模板列表

最近更新时间：2024-10-05 14:54:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`
本接口（AppDescribeTemplateBinding）用于查询已经绑定的第三方模板列表。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppDescribeTemplateBinding。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID。
DeviceName	String	是	设备名称。

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
IsSubscribeable	Bool	是否有可以订阅的模板。
Data	Array of WechatTemplate	第三方模板列表。

4. 示例

示例1 查询已经绑定的第三方模板列表

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppDescribeTemplateBinding",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICENAME"
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "IsSubscribeable": true,
      "Templates": [
        {
          "ModelId": "XXX",
```

```
            "TemplateId": "XXX",
            "Title": "XXX",
            "Status": 0
          }
        ]
    },
    "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
  }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

查询微信授权票据

最近更新时间：2024-06-14 10:35:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（APPGetWechatDeviceTicket）用于获取微信设备票据。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：APPGetWechatDeviceTicket。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品 ID。
Devicename	String	是	设备名称。
IsThirdApp	Int	是	是否是第三方小程序： 1=是。
ModelId	String	否	模板 ID
MiniProgramAppId	String	否	小程序 APPID

3. 输出参数

名称	类型	描述
RequestId	String	RequestId。
Data	WXDeviceInfo	生成的微信设备票据信息。

4. 示例

示例1 获取微信设备票据

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "APPGetWechatDeviceTicket",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "Devicename": "DEVICE_NAME",
  "IsThirdApp": 1
}
```

输出示例：成功

```
{
  "Response": {
    "Data": {
      "DeviceId": "xxx",
```

```
      "WXIoTDeviceInfo": {
        "SN": "xxx",
        "SNTicket": "xxx",
        "ModelId": "xxx"
      },
      "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
  }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
InvalidParameterValue.TokenInfoSaveError	Token 保存失败。

订阅第三方模板

最近更新时间：2024-10-05 14:54:11

1. 接口描述

接口请求域名： `iot.cloud.tencent.com/api/exploreropen/tokenapi`

本接口（AppSubscribeTemplate）订阅第三方模板。

2. 输入参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppSubscribeTemplate。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuid。定位问题时，需要提供该次请求的 RequestId。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
ProductId	String	是	产品ID
DeviceName	String	是	设备名称
ModelId	String	是	产品型号ID
TemplateIds	[]String	是	第三方模板ID列表
Status	Int	是	订阅状态； 0=未订阅；1=已订阅；2=不接收

3. 输出参数

名称	类型	描述
RequestId	String	唯一请求 ID。

4. 示例

示例1 订阅第三方模板

输入示例

```
POST https://iot.cloud.tencent.com/api/exploreropen/tokenapi HTTP/1.1
content-type: application/json
{
  "Action": "AppSubscribeTemplate",
  "AccessToken":"bf******************d098",
  "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68",
  "ProductId": "PRODUCT_ID",
  "DeviceName": "DEVICE_NAME",
  "ModelId": "MODEL_ID",
  "TemplateIds": ["template1","template2"],
  "Status": 1
}
```

输出示例：成功

```
{
    "Response": {
        "Data": {
            "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
```

```
        }
    }
}
```

输出示例：失败

```
{
    "Response": {
        "Error": {
            "Code": "InvalidParameterValue.InvalidAccessToken",
            "Message": "Token无效"
        },
        "RequestId": "f92406b3-5a9a-4fe8-bc43-45e3d794bb68"
    }
}
```

5. 错误码

错误码	描述
InvalidParameter	无效参数。
InvalidParameterValue	参数取值错误。
ResourceNotFound	资源未创建或是已删除。

数据结构

最近更新时间：2024-03-05 14:52:31

Action

自动智能联动动作

被如下接口引用：AppCreateAutomation、AppCreateScene、AppDescribeAutomation、AppGetSceneList、AppModifyAutomation、AppModifyScene。

名称	类型	必选	描述
ActionType	Int	是	动作类型。 • 0：设备动作。 • 1：延时。 • 2：场景。 • 3：通知。
ProductId	String	否	产品 ID。
DeviceName	String	否	设备名称。
Data	String	是	• ActionType=0：设备动作参数。 • ActionType=1：延时时间单位秒。 • ActionType=2：执行场景 Id。 • ActionType=3：通知内容。
AliasName	String	否	设备别名或者场景名称。
IconUrl	String	否	产品或者场景的图标 URL。

ActionHistories

设备动作历史

被如下接口引用：AppGetDeviceActionHistories。

名称	类型	描述
DeviceName	String	设备名称。
ActionId	String	动作 Id。
ActionName	String	动作名称。
ReqTime	int64	请求时间。
RspTime	int64	响应时间。
InputParams	string	输入参数。
OutputParams	string	输出参数。
Calling	string	调用方式。
ClientToken	string	调用 Id。
Status	string	调用状态。

ActionPushData

监听之后的服务器回包

被如下接口引用：DeviceRegistrationMonitor。

名称	类型	描述
deviceIds	[String]	返回成功注册监听的设备列表。 如果入参的设备 ID 无返回，表示该用户未绑定要注册监听的设备。

ActionPushParam

注册设备的时候传入的参数

被如下接口引用：DeviceRegistrationMonitor。

名称	类型	必选	描述
AccessToken	String	是	AccessToken 为调用登录接口之后获取的 Token。
DeviceIds	[String]	是	需要注册监听的设备列表，必须是已经绑定到该用户名下的设备。

AlarmDetailItem

预警详细信息

被如下接口引用：AppGetInsuranceAlarmDetailList。

名称	类型	必选	描述
Name	String	是	预警名称。
Reason	String	是	隐患分析。
Detail	String	是	预警细节。
Suggestion	String	是	处置建议。
AlarmTimestamp	Int64	是	预警时间。

AlarmSummaryItem

预警摘要信息

被如下接口引用：AppGetInsuranceAlarmSummaryList。

名称	类型	必选	描述
Name	String	是	预警名称。
Reason	String	是	隐患分析。
Suggestion	String	是	处置建议。
AlarmTimestamp	Int64	是	预警时间。
AlarmCount	Int64	是	总预警数。

AppDeviceInfo

App 设备信息

被如下接口引用：AppBindSubDeviceInFamily、AppSecureAddDeviceInFamily、AppSigBindDeviceInFamily。

名称	类型	描述
DeviceId	String	设备 ID。
ProductId	String	产品 ID。
DeviceName	String	设备名。
AliasName	String	设备别名。

CreateTime	String	创建 unix 时间戳（秒级）。
UpdateTime	String	最后更新时间，unix 时间戳（秒级）。
FamilyId	String	家庭 ID。
RoomId	String	房间 ID。
IconUrl	String	图标 Url。

AppInsuranceDeviceInfo

设备信息

被如下接口引用：AppGetUnBindDeviceList。

名称	类型	必选	描述
DeviceId	String	是	设备 ID。
DeviceNo	String	是	设备 SN 或车架号。

AppServiceInsuranceInfo

保单信息

被如下接口引用：AppGetInsuranceList。

名称	类型	必选	描述
ItemId	String	是	保单类别。
ItemType	Int64	是	保单类型。 • 0：测试保单。 • 1：正式保单。
InsuranceNo	String	是	保单号。
Address	String	是	地址。
BeginTime	Int64	是	保单生效时间。
EndTime	Int64	是	保单结束时间。
Score	Int64	是	评分。
AlarmNum	Int64	是	预警条数。
Status	Int64	是	保单状态。 • 1：失效。 • 2：保障中。 • 3：待生效。
ServiceName	String	是	服务名称。
BindDeviceId	String	是	绑定设备 ID。

Automation

自动智能联动实体

被如下接口引用：AppDescribeAutomation。

名称	类型	必选	描述
AutomationId	String	是	自动智能联动 ID。
Icon	String	是	自动智能联动的背景图片地址。

Name	String	是	自动智能联动的名称。
Status	int	是	自动智能联动开关状态。 • 0：关闭。 • 1：启用。
MatchType	int	是	条件匹配类型。 • 0：全部条件满足。 • 1：其中一个条件满足。
EffectiveBeginTime	String	是	生效开始时间。
EffectiveEndTime	String	是	生效结束时间。
EffectiveDays	String	是	生效日期 由0和1组成的7位数字，0表示不执行，1表示执行，第1位为周日，依次表示周一至周六。
Conditions	Array of Condition	是	自动智能联动触发条件。
Actions	Array of Action	是	自动智能联动动作列表。

AutomationListItem

自动智能联动信息

被如下接口引用：AppGetAutomationList。

名称	类型	必选	描述
AutomationId	String	是	自动智能联动 ID。
Icon	String	是	自动智能联动的背景图片地址。
Name	String	是	自动智能联动的名称。
Status	int	是	自动智能联动开关状态。 • 0：关闭。 • 1：启用。

Condition

App 设备信息

被如下接口引用：AppCreateAutomation、AppDescribeAutomation、AppModifyAutomation。

名称	类型	必选	描述
CondId	String	是	条件 ID。
CondType	Int32	是	条件类型。 • 0：设备属性条件。 • 1：定时条件。
Property	Property	否	条件属性。
Timer	Timer	否	条件触发时间。

Data

返回数据

被如下接口引用：device_activation。

名称	类型	描述

activation_date	string	设备激活日期。

DataHistoryItem

历史数据记录

被如下接口引用：AppGetDeviceDataHistory。

名称	类型	描述
Time	String	毫秒时间戳。
Value	String	数据取值。

Data_accompany_awesome_personal

返回数据

被如下接口引用：accompany_awesome_personal。

名称	类型	必选	描述
accompany	Array Of Object of accompany_awesome_personal	否	伴奏个性推荐。

Data_accompany_awesome_top

返回数据

被如下接口引用：accompany_awesome_top。

名称	类型	必选	描述
accompany	Array Of Object of accompany_awesome_top	否	伴奏推荐列表。

Data_accompany_category_info

返回数据

被如下接口引用：accompany_category_info。

名称	类型	必选	描述
category_id	String	是	分类 id。
category_name	String	是	分类名。
category_img	String	是	分类图片。

Data_accompany_category_list

返回数据

被如下接口引用：accompany_category_list。

名称	类型	必选	描述
groups	Array Of groups_accompany_category_list of groups_accompany_category_list	否	伴奏分组。
recommends	Array Of recommends_accompany_category_list of recommends_accompany_category_list	否	推荐分类。

Data_accompany_category_song

返回数据

被如下接口引用：accompany_category_song。

名称	类型	必选	描述
accompany	Array Of Object of accompany_category_song	否	伴奏列表。

Data_accompany_favorite_list

被如下接口引用：accompany_favorite_list。

名称	类型	必选	描述
playlist_id	String	否	歌单 id。
total	Integer	否	总数。
accompany	Array Of Object of accompany_favorite_list	否	伴奏列表。

Data_accompany_free_info

返回数据

被如下接口引用：accompany_free_info。

名称	类型	必选	描述
accompany	Array Of accompany_accompany_free_info of accompany_accompany_free_info	是	体验信息。

Data_accompany_free_song_url

返回数据

被如下接口引用：accompany_free_song_url。

名称	类型	必选	描述
accompany_size	String	是	伴奏大小单位字节。
has_original	Integer	否	音频是否有原唱音轨 。 • 0:无。 • 1:有。
singer_name	String	是	歌手名。
playable_code	Integer	是	是否可以播放 枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：音乐方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况） • 9：未知原因，无权播放。
accompany_id	String	是	伴奏 ID。
accompany_url	String	是	伴奏下载链接。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 ID。
song_name	String	是	歌曲名。

Data_accompany_free_top_list

返回数据

被如下接口引用：accompany_free_top_list。

名称	类型	必选	描述
groups	Array Of groups_accompany_free_top_list of groups_accompany_free_top_list	否	分组列表。

Data_accompany_free_top_song

返回数据

被如下接口引用：accompany_free_top_song。

名称	类型	必选	描述
accompany	Array Of Object of accompany_free_top_song	否	伴奏列表。

Data_accompany_free_url

返回数据

被如下接口引用：accompany_free_url.

名称	类型	必选	描述
playable_code	Integer	是	是否可以播放 枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
accompany_id	String	是	伴奏 ID。
accompany_size	String	是	伴奏大小单位字节。
accompany_url	String	是	伴奏下载链接。
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。 • 1：有。
duration	Integer	是	伴奏时长单位毫秒。
singer_name	String	是	歌手名。
song_name	String	是	歌曲名。

Data_accompany_hq_url

返回数据

被如下接口引用：accompany_hq_url。

名称	类型	必选	描述
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。

			• 1：有。
duration	Integer	是	伴奏时长单位毫秒。
song_name	String	是	歌曲名。
playable_code	Integer	是	是否可以播放 枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
accompany_id	String	是	伴奏 ID。
accompany_size	String	是	伴奏大小单位字节。
singer_name	String	是	歌手名。
accompany_url	String	是	伴奏下载链接。
singer_id	String	是	歌手 ID。

Data_accompany_info

返回数据

被如下接口引用：accompany_info。

名称	类型	必选	描述
accompany	Array Of Object of accompany_accompany_info	否	伴奏列表。

Data_accompany_lyric

返回数据

被如下接口引用：accompany_lyric。

名称	类型	必选	描述
lyric	String	是	base64编码的歌词内容。
offset	Integer	是	歌词偏移量。
format	String	是	歌词格式。

Data_accompany_mv

返回数据

被如下接口引用：accompany_mv。

名称	类型	必选	描述
mv_id	String	是	MV id。
mv_name	String	是	MV 名。
duration	Integer	否	MV 时长，单位毫秒。
offset	Integer	否	MV 与伴奏对齐时间偏移量。

Data_accompany_newsearch_song

返回数据

被如下接口引用：accompany_newsearch_song。

名称	类型	必选	描述
accompany	Array Of Object of accompany_newsearch_song	否	伴奏列表。
author	Array Of Object of author_newsearch_song	否	歌手列表。

Data_accompany_opus_del

返回数据

被如下接口引用：accompany_opus_del。

名称	类型	描述
data	object	objects

Data_accompany_opus_list

返回数据

被如下接口引用：accompany_opus_list。

名称	类型	必选	描述
total	Integer	否	总数
opuses	Array Of opuses_accompany_opus_list of opuses_accompany_opus_list	否	opuses

Data_accompany_opus_upload

返回数据

被如下接口引用：accompany_opus_upload。

名称	类型	必选	描述
opus_id	String	否	作品 id。

Data_accompany_opus_url

返回数据

被如下接口引用：accompany_opus_url。

名称	类型	必选	描述
url	String	是	作品下载链接。
duration	Integer	是	时长单位秒。
size	Integer	是	伴奏大小单位字节。

Data_accompany_search_song

返回数据

被如下接口引用：accompany_search_song。

名称	类型	必选	描述
accompany	Array Of Object of	否	伴奏搜索。

	accompany_search_song		
total	Integer	否	总数。

Data_accompany_search_tips

被如下接口引用：accompany_search_tips。

名称	类型	必选	描述
tips	Array Of String	否	提示。

Data_accompany_singer_list

歌手列表

被如下接口引用：accompany_singer_list。

名称	类型	必选	描述
singers	Array Of singers_accompany_singer_list of singers_accompany_singer_list	否	热门歌手。
letters	Array Of letters_accompany_singer_list of letters_accompany_singer_list	否	letters

Data_accompany_singer_song

返回数据

被如下接口引用：accompany_singer_song。

名称	类型	必选	描述
total	Integer	否	总数。
accompany	Array Of Object of accompany_singer_song	否	伴奏列表。

Data_accompany_songpitch

返回数据

被如下接口引用：accompany_songpitch。

名称	类型	必选	描述
pitch	String	是	音高。
offset	Integer	是	音高偏移量。
type	Integer	是	来源。 • 0为默认值。 • 1为算法制作。 • 2为人工导入。
md5sum	String	是	md5 签名，用于完整性校验。

Data_accompany_theme_list

返回数据

被如下接口引用：accompany_theme_list。

名称	类型	必选	描述
themes	Array Of themes_accompany_theme_list of	否	主题列表。

	themes_accompany_theme_list		

Data_accompany_theme_song

被如下接口引用：accompany_theme_song。

名称	类型	必选	描述
accompany	Array Of Object of accompany_theme_song	否	伴奏列表。

Data_accompany_top_list

被如下接口引用：accompany_top_list

名称	类型	必选	描述
groups	Array Of groups_accompany_top_list of groups_accompany_top_list	否	分组列表。

Data_accompany_top_song

返回数据

被如下接口引用：accompany_top_song。

名称	类型	必选	描述
accompany	Array Of Object of accompany_top_song	否	伴奏列表。

Data_accompany_url

返回数据

被如下接口引用：accompany_url。

名称	类型	必选	描述
accompany_id	String	是	伴奏 ID
duration	Integer	是	伴奏时长单位毫秒
singer_id	String	是	歌手 ID
song_name	String	是	歌曲名
playable_code	Integer	是	是否可以播放 枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
status	Integer	是	• 0：成功获取到伴奏下载信息。 • 1：当日免费体验次数已经用完。
accompany_size	String	是	伴奏大小单位字节。
accompany_url	String	是	伴奏下载链接。
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。

			• 1：有。
singer_name	String	是	歌手名。

Data_album_info

返回数据

被如下接口引用：album_info。

名称	类型	描述
album_id	Integer	专辑 id。
album_name	String	专辑名。
album_translate_name	String	专辑译名。
album_img	String	专辑封面。
intro	String	专辑简介。
company	String	唱片公司。
singer_id	String	歌手 id。
singer_name	String	歌手名。
total	Integer	歌曲总数。
publish_time	String	发布时间。
songs	Array of songs_album_info	歌曲列表。

Data_asset_purchased_albums

返回数据

被如下接口引用：asset_purchased_albums。

名称	类型	描述
total	Integer	总数。
albums	Array of albums_asset_purchased_albums	专辑信息。

Data_asset_purchased_songs

返回数据

被如下接口引用：asset_purchased_songs。

名称	类型	描述
total	Integer	总数。
songs	Array of songs_asset_purchased_songs	歌曲列表。

Data_awesome_everyday

返回数据

被如下接口引用：awesome_everyday。

名称	类型	描述
songs	array of songs_awesome_everyday	歌曲列表。

Data_awesome_newsong

返回数据

被如下接口引用：awesome_newsong。

名称	类型	描述
total	Integer	总数。
songs	Array of songs_awesome_newsong	歌曲列表。

Data_awesome_recommend

返回数据

被如下接口引用：awesome_recommend。

名称	类型	描述
songs	Array of songs_awesome_recommend	歌曲列表。

Data_child_home

返回数据

被如下接口引用：child_home。

名称	类型	必选	描述
groups	Array Of groups_child_home of groups_child_home	否	资源分组。

Data_child_resource

返回数据

被如下接口引用：child_resource。

名称	类型	必选	描述
type	Integer	否	大分类。 • 3-歌单。 • 4-专辑。 • 7-小分类。
total	Integer	否	总数。
lists	Array Of lists_child_resource of lists_child_resource	否	资源分组。

Data_favorite_oper_playlist

返回数据

被如下接口引用：favorite_oper_playlist。

名称	类型	描述
playlist_extra_id	String	歌单附加 id，移除收藏时使用。
playlist_id	String	操作的歌单 id。

Data_favorite_oper_song

返回数据

被如下接口引用：favorite_oper_song。

名称	类型	描述
song_extra_id	String	歌曲附加 id，移除收藏时使用。
song_id	String	操作的歌曲 id。

Data_favorite_other_list

返回数据

被如下接口引用：favorite_other_list。

名称	类型	描述
playlists	Array of playlists_favorite_other_list	歌单。

Data_favorite_self_list

返回数据

被如下接口引用：favorite_self_list。

名称	类型	描述
playlists	Array of playlists_favorite_self_list	歌单。

Data_favorite_song

返回数据

被如下接口引用：favorite_song。

名称	类型	描述
total	Integer	总数。
songs	Array of songs_favorite_song	歌曲列表。

Data_give_common_record

返回数据

被如下接口引用：give_common_record。

名称	类型	描述
status	Integer	-
times	Integer	剩余领取次数。
collection_time	String	上一次领取时间。

Data_give_common_scancodelogin

返回数据

被如下接口引用：give_common_scancodelogin。

名称	类型	描述
status	Integer	-
order_no	String	订单号(同 sp 内唯一)。
partner_no	String	厂商订单号(同 sp 内唯一)。
create_time	String	订单创建时间。

vip_end_time	String	vip 有效期终止时间。
collection_time	String	上一次领取时间。

Data_live_program_list

返回数据

被如下接口引用：live_program_list。

名称	类型	必选	描述
programs	Array Of programs_live_program_list of programs_live_program_list	否	节目分组。

Data_live_room_status

返回数据

被如下接口引用：live_room_status。

名称	类型	必选	描述
status	Integer	是	房间内节目状态。 • 0：停播。 • 1：开播。
horizontal	Integer	是	横屏流状态。 • 0：停播。 • 1：开播。
vertical	Integer	是	竖屏流状态。 • 0：停播。 • 1：开播。

Data_live_room_url

返回数据

被如下接口引用：live_room_url。

名称	类型	必选	描述
vertical	Array Of vertical_live_room_url of vertical_live_room_url	否	竖屏流信息。
status	String	是	房间内节目状态。 • 0：停播。 • 1：开播。
room_id	Integer	否	房间 id。
expire	Integer	否	链接过期时间。
horizontal	Array Of horizontal_live_room_url of horizontal_live_room_url	否	横屏流信息。

Data_longaudio_free_home

返回数据

被如下接口引用：longaudio_free_home。

名称	类型	必选	描述
groups	Array Of groups_longaudio_free_home of	否	资源分组。

	groups_longaudio_free_home		

Data_longaudio_resource

返回数据
被如下接口引用：longaudio_resource。

名称	类型	必选	描述
is_end	String	否	是否存在下一页。 **特别提醒**：返回结果个数可能少于请求的 size，建议以瀑布流的方式设计 UI 交互。
lists	Array Of lists_longaudio_resource of lists_longaudio_resource	否	资源列表。

Data_minipauth

返回数据
被如下接口引用：wechat_minipauth。

名称	类型	描述
token	string	用户令牌。
expire	integer	令牌有效期 unix 时间戳，单位秒。
userid	string	用户 id。

Data_monitor_delay

返回值为空。
被如下接口引用：monitor_delay。

Data_monitor_listen

返回值为空。
被如下接口引用：monitor_listen。

Data_mv_category_list

返回数据
被如下接口引用：mv_category_list。

名称	类型	必选	描述
categories	Array Of categories_mv_category_list of categories_mv_category_list	否	分类列表。

Data_mv_category_mv

返回数据
被如下接口引用：mv_category_mv

名称	类型	必选	描述
mvs	Array Of mvs_mv_category_mv of mvs_mv_category_mv	否	MV 列表。

Data_mv_info

返回数据
被如下接口引用：mv_info。

名称	类型	必选	描述
song_id	String	是	歌曲 ID。
singer_id	String	是	歌手 ID。
singer_name	String	是	歌手名称。
author_name	String	否	MV 制作人名称.
intro	String	否	简介。
tags	tags_mv_info	否	MV 信息。
mv_id	String	是	MV ID。
mv_img	String	是	MV 图片。
mv_name	String	是	MV 名称。
accompany_id	String	否	伴奏 ID。

Data_mv_infos

返回数据

被如下接口引用：mv_infos。

名称	类型	必选	描述
mvInfos	Array Of mvInfos_mv_infos of mvInfos_mv_infos	否	MV 信息。

Data_mv_live_tme

返回数据

被如下接口引用：mv_live_tme。

名称	类型	必选	描述
mvs	Array Of mvs_mv_live_tme of mvs_mv_live_tme	否	MV 列表。

Data_mv_search

返回数据

被如下接口引用：mv_search。

名称	类型	必选	描述
mvs	Array Of mvs_mv_search of mvs_mv_search	否	MV 列表。
total	Integer	否	总数。

Data_mv_trial

返回数据

被如下接口引用：mv_trial。

名称	类型	必选	描述
mv_size_ld	Integer	否	流畅版 MV 大小。
mv_url_ld	String	否	流畅版 MV 下载链接。
mv_size_sd	Integer	否	标清版 MV 大小。
mv_url_sd	String	否	标清版 MV 下载链接。

mv_size_qhd	Integer	否	高清版 MV 大小。
mv_url_qhd	String	否	高清版 MV 下载链接。
mv_size_hd	Integer	否	超清版 MV 大小。
mv_url_hd	String	否	超清版 MV 下载链接。
mv_size_fhd	Integer	否	蓝光版 MV大小。
mv_url_fhd	String	否	蓝光版MV 下载链接。
duration	Integer	是	MV 时长单位毫秒。
mv_id	String	是	MV ID。
mv_name	String	是	MV 名。
singer_name	String	是	歌手名。

Data_mv_url

返回数据

被如下接口引用：mv_url

名称	类型	必选	描述
mv_size_hd	Integer	否	超清版 MV 大小。
mv_url_hd	String	否	超清版 MV 下载链接。
mv_size_fhd	Integer	否	蓝光版 MV 大小。
mv_url_fhd	String	否	蓝光版 MV 下载链接。
mv_size_qhd	Integer	否	高清版 MV 大小。
mv_url_qhd	String	否	高清版 MV 下载链接。
mv_size_sd	Integer	否	标清版 MV 大小。
mv_url_sd	String	否	标清版 MV 下载链接。
mv_size_ld	Integer	否	流畅版 MV 大小。
mv_url_ld	String	否	流畅版 MV 下载链接。
duration	Integer	是	MV 时长单位毫秒。
mv_id	String	是	MV ID。
mv_name	String	是	MV 名。
singer_name	String	是	歌手名。

Data_playlist_awesome

返回数据

被如下接口引用：playlist_awesome

名称	类型	描述
playlists	Array of playlists_playlist_awesome	歌单。

Data_playlist_song

返回数据

被如下接口引用：playlist_song。

名称	类型	描述
total	Integer	总数。
songs	Array of songs_playlist_song	歌曲列表。

Data_playlist_top

返回数据

被如下接口引用：playlist_top

名称	类型	描述
groups	Array of groups_playlist_top	分组列表。

Data_radio_list

返回数据

被如下接口引用：radio_list。

名称	类型	描述
groups	Array of groups_radio_list	分组列表。

Data_radio_song

返回数据

被如下接口引用：radio_song。

名称	类型	描述
total	Integer	总数。
songs	Array of songs_radio_song	歌曲列表。

Data_rigion_home

返回数据

被如下接口引用：rigion_home。

名称	类型	必选	描述
groups	Array Of groups_rigion_home of groups_rigion_home	否	资源分组。

Data_rigion_oldman_index

返回数据

被如下接口引用：rigion_oldman_index。

名称	类型	必选	描述
groups	Array Of groups_rigion_oldman_index of groups_rigion_oldman_index	否	资源分组。

Data_search_song

返回数据

被如下接口引用：search_song。

名称	类型	必选	描述
total	Integer	是	总数。
keyword	String	是	关键词。
songs	Array Of songs_search_song of songs_search_song	是	歌曲列表。

Data_search_tips

返回数据
被如下接口引用：search_tips。

名称	类型	必选	描述
tips	Array Of String	是	tips

Data_search_voice

返回数据
被如下接口引用：search_voice。

名称	类型	必选	描述
tts_text	String	是	用于语音提示的文本。
songs	songs_search_voice	是	语音搜索歌曲列表。

Data_singer_album

返回数据
被如下接口引用：singer_album。

名称	类型	描述
total	Integer	歌曲总数。
albums	Array of albums_singer_album	专辑列表。

Data_singer_info

返回数据
被如下接口引用：singer_info。

名称	类型	描述
singer_id	String	歌手 id。
singer_name	String	歌手名。
area	String	歌手地区。
translate_name	String	歌手译名。
song_count	Integer	歌手歌曲数。
singer_img	String	歌手相片。

Data_singer_list

返回数据
被如下接口引用：singer_list。

名称	类型	描述
singers	Array of singers_singer_list	歌手

Data_singer_song

返回数据
被如下接口引用：singer_song。

名称	类型	描述
total	Integer	歌曲总数。
songs	Array of songs_singer_song	歌曲列表。

Data_song_accompany

返回数据
被如下接口引用：song_accompany。

名称	类型	描述
accompany	Array of accompany_song_accompany	伴奏。

Data_song_infos

返回数据
被如下接口引用：song_infos。

名称	类型	描述
songs	Array of songs_song_infos	歌曲列表。

Data_song_lyric

返回数据
被如下接口引用：song_lyric。

名称	类型	描述
lyric	string	歌词。

Data_song_tolisten

返回数据
被如下接口引用：song_tolisten。

名称	类型	描述
songs	Array of songs_song_tolisten	歌曲列表。

Data_song_url

返回数据
被如下接口引用：song_url。

名称	类型	描述
song_id	String	歌曲 id。
song_name	String	歌曲名。

singer_name	String	歌手名。
song_url	String	歌曲下载链接。
song_size	String	歌曲大小。
song_url_hq	String	歌曲(hq)下载链接。
song_size_hq	String	歌曲(hq)大小。
song_url_sq	String	歌曲(sq)下载链接。
song_size_sq	String	歌曲(sq)大小。
duration	Integer	歌曲时长，单位毫秒。
playable_code	Integer	是否可以播放 枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_url	String	歌曲试听链接。
try_size	Integer	歌曲试听大小，单位字节。
try_begin	Integer	歌曲试听开始，单位毫秒。
try_end	Integer	歌曲试听结束，单位毫秒。

Data_top_list

返回数据

被如下接口引用：top_list。

名称	类型	描述
groups	Array of groups_top_list	分组。

Data_top_song

返回数据

被如下接口引用：top_song。

名称	类型	描述
total	Integer	总数。
songs	songs_top_song	歌曲列表。

Data_user_info

返回数据

被如下接口引用：user_info。

名称	类型	描述
userid	String	用户 id。
nick_name	String	用户昵称。

img	String	用户头像。
is_vip	Integer	是否vip。 • 0：否。 • 1：是。
vip_end_time	String	vip 有效期终止时间。
car_vip_end_time	String	车机会员有效期终止时间。
svip_end_time	String	豪 V 有效期终止时间。

Data_user_logout

返回数据为空。
被如下接口引用：user_logout。

Data_user_qrcode_auth

返回数据
被如下接口引用：user_qrcode_auth。

名称	类型	描述
userid	String	用户 ID。
token	String	用户令牌。
expire	Integer	令牌有效期 unix 时间戳，单位秒。

Data_user_qrcode_get

返回数据
被如下接口引用：user_qrcode_get。

名称	类型	描述
qrcode	String	登录二维码 url，自行渲染。
ticket	String	获取二维码授权信息的凭证。

Data_vip_activity_partner_give

返回数据
被如下接口引用：vip_activity_partner_give

名称	类型	必选	描述
create_time	String	否	TME 订单创建时间。
vip_end_time	String	否	vip 有效期终止时间。
status	Integer	是	-
order_no	String	否	TME 订单号(同 sp 内唯一)。
partner_no	String	否	厂商订单号(同 sp 内唯一)。

Data_vip_ksing_info

返回数据
被如下接口引用：vip_ksing_info。

名称	类型	必选	描述

userid	String	是	用户 ID。
nick_name	String	是	用户昵称。
img	String	是	用户头像。
is_vip	Integer	是	是否vip。 • 0：否。 • 1：是。
vip_end_time	String	是	vip 有效期终止时间。

Data_vip_ksing_partner_give

返回数据

被如下接口引用：vip_ksing_partner_give

名称	类型	必选	描述
order_no	String	是	TME 订单号(同 sp 内唯一)。
partner_no	String	是	厂商订单号(同 sp 内唯一)。
create_time	String	是	TME 订单创建时间。
vip_end_time	String	是	vip 有效期终止时间。

Data_vip_svip_common_give

返回数据

被如下接口引用：vip_svip_common_give

名称	类型	必选	描述
order_no	String	是	TME 订单号(同 sp 内唯一)。
partner_no	String	是	厂商订单号(同 sp 内唯一)。
create_time	String	是	TME 订单创建时间。

Data_vip_tme_partner_give

返回数据

被如下接口引用：vip_tme_partner_give

名称	类型	必选	描述
order_no	String	是	TME 订单号(同 sp 内唯一)。
partner_no	String	是	厂商订单号(同 sp 内唯一)。
create_time	String	是	TME 订单创建时间。
vip_end_time	String	是	vip 有效期终止时间。

Data_wechat_appinfocode

返回数据

被如下接口引用：wechat_appinfocode

名称	类型	描述
ticket	String	获取第三方 App 信息凭证。
req_name	String	拉起的小程序的请求名。

req_path	String	拉起的小程序的请求路径。

Data_wechat_openlink

返回数据
被如下接口引用：wechat_openlink

名称	类型	描述
ticket	String	获取扫码授权结果的凭证。
openlink	String	微信小程序 openlink。
expire	Integer	令牌有效期 unix 时间戳，单位秒。

Data_wechat_qrcodeauth

返回数据
被如下接口引用：wechat_qrcodeauth

名称	类型	描述
token	String	用户令牌。
userid	String	用户 id。
expire	Integer	令牌有效期 unix 时间戳，单位秒。

Data_wechat_qrcodeget

返回数据
被如下接口引用：wechat_qrcodeget

名称	类型	描述
ticket	String	获取扫码授权结果的凭证。
qrcode	String	微信小程序二维码图像二进制数据 base64 编码。

Device

设备信息
被如下接口引用：AppGetDeviceStatuses

名称	类型	描述
DeviceName	String	设备名。
Online	Int	在线状态。
LoginTime	Int64	登录时间。
LastDataTime	Int64	最近登录时间。

DeviceFenceEvent

设备地理围栏告警事件
被如下接口引用：AppGetFenceEventList

名称	类型	必选	描述
EventType	String	是	设备地理围栏名称。
CreateTime	String	否	创建 unix 时间戳（秒级）。

DeviceItem

设备状态

被如下接口引用：AppGetInsuranceDeviceStatus

名称	类型	必选	描述
ProductName	String	是	设备展示名称。
OnlineStatus	Int	是	设备状态： • 0：离线。 • 1：在线。 • 2：未知。

DeviceList

设备列表

被如下接口引用：AppGetFamilyDeviceList、AppGetFamilySubDeviceList、AppGetGatewayBindDeviceList

名称	类型	描述
ProductId	String	设备 ID。
DeviceName	String	设备名称。
DeviceId	String	设备 Id。
AliasName	String	设备别名。
UserId	String	用户 Id。
RoomId	String	房间 ID。
FamilyId	String	家庭 ID。
IconUrl	String	图标 Url。
DeviceType	Int32	设备类型： • 0：普通。 • 1：网关设备。 • 2：子设备。
CreateTime	Int64	创建 unix 时间戳（秒级）。
UpdateTime	Int64	最后更新的 unix 时间戳（秒级）。

ErrorMessage

错误信息

名称	类型	描述
Code	String	错误码。
Message	String	错误信息。

EventHistory

搜集结果集

名称	类型	描述
TimeStamp	Int64	事件的时间戳。

ProductId	String	事件的产品 ID。
DeviceName	String	事件的设备名称。
EventId	String	事件的标识符 ID。
Type	String	事件的类型。
Data	string	事件的数据。

FamilyList

被如下接口引用：AppGetFamilyList

名称	类型	描述
FamilyId	String	家庭 ID。
Name	String	名称。
Role	Int	角色。
CreateTime	Int	创建时间。
UpdateTime	Int	最后更新时间。

Fence

设备地理围栏信息
被如下接口引用：AppGetFenceList

名称	类型	必选	描述
FenceId	Int64	是	设备地理围栏 ID。
FenceName	String	是	设备地理围栏名称。
FenceArea	String	是	设备地理围栏区域范围。
FenceDesc	String	是	设备地理围栏描述。
AlertCondition	String	是	触发条件。
FenceEnable	Boolean	是	启用状态。
Method	String	是	通知方式。
CreateTime	String	否	创建 unix 时间戳（秒级）。
UpdateTime	String	否	最后更新时间，unix 时间戳（秒级）。

HeartBeatData

心跳接口的服务器回包
被如下接口引用：HeartBeat

名称	类型	描述
RequestId	String	公共参数，唯一请求 ID，与入参相同，定位问题时，需提供该次请求的 RequestId。

HeartBeatParam

设备心跳接口入参
被如下接口引用：HeartBeat

名称	类型	描述
DeviceIds	[String]	设备列表。

IotUser

用户信息
被如下接口引用：AppFindUser

名称	类型	描述
UserId	String	用户 Id。
NickName	String	昵称。
Avatar	String	头像。
CountryCode	String	手机号国家码。
PhoneNumber	String	手机号码。
Email	String	邮箱。

IotUserDetail

用户信息
被如下接口引用：AppGetUser

名称	类型	描述
UserId	String	用户 Id。
NickName	String	昵称。
Avatar	String	头像。
CountryCode	String	手机号国家码。
PhoneNumber	String	手机号码。
Email	String	邮箱。
HasPassword	Int	是否已设置密码。 • 0：未设置。 • 1：已设置。
HasWxOpenID	Int	是否已绑定微信。 • 0：未绑定。 • 1：已绑定。

KugouParams

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：device_activation。

KugouParams_accompany_awesome_personal

请求参数
被如下接口引用：accompany_awesome_personal

名称	类型	必选	描述
size	Integer	是	每页?项。
page	Integer	是	第?页。

KugouParams_accompany_awesome_top

请求参数

被如下接口引用：accompany_awesome_top。

名称	类型	必选	描述
size	Integer	是	每页?项。
page	Integer	是	第?页。
type	Integer	是	排行榜类型。 • 1：热门榜。 • 2：飙升榜。

KugouParams_accompany_category_info

请求参数

被如下接口引用：accompany_category_info。

名称	类型	必选	描述
category_id	String	是	伴奏分类 id。

KugouParams_accompany_category_list

[object Object]

被如下接口引用：accompany_category_list。

KugouParams_accompany_category_song

请求参数

被如下接口引用：accompany_category_song。

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
category_id	String	是	伴奏分类 id。

KugouParams_accompany_favorite_list

请求参数

被如下接口引用：accompany_favorite_list。

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。

KugouParams_accompany_free_info

请求参数这里为空，传递空对象 {} 即可。

被如下接口引用：accompany_free_info。

KugouParams_accompany_free_song_url

请求参数

被如下接口引用：accompany_free_song_url

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_accompany_free_top_list

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：accompany_free_top_list。

KugouParams_accompany_free_top_song

请求参数
被如下接口引用：accompany_free_top_song

名称	类型	必选	描述
group_id	String	是	分组 id。
top_id	String	是	榜单、电台 id。

KugouParams_accompany_free_url

请求参数
被如下接口引用：accompany_free_url。

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。
free_token	String	否	免费令牌。

KugouParams_accompany_hq_url

请求参数
被如下接口引用：accompany_hq_url。

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_accompany_info

请求参数
被如下接口引用：accompany_info。

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_accompany_lyric

请求参数
被如下接口引用：accompany_lyric。

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_accompany_mv

请求参数
被如下接口引用：accompany_mv。

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_accompany_newsearch_song

请求参数
被如下接口引用：accompany_newsearch_song

名称	类型	必选	描述
keyword	String	是	关键词。

KugouParams_accompany_opus_del

请求参数
被如下接口引用：accompany_opus_del

名称	类型	必选	描述
opus_id	String	是	作品 id。

KugouParams_accompany_opus_list

请求参数
被如下接口引用：accompany_opus_list

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。

KugouParams_accompany_opus_upload

请求参数
被如下接口引用：accompany_opus_upload

名称	类型	必选	描述
opus_hash	String	是	作品 hash。
opus_desc	String	是	作品描述。
krc_id	String	是	歌词 id。
grade	String	是	等级。
opus_name	String	是	作品名。
accompany_id	String	是	伴奏 id。
duration	Integer	是	作品播放时长，单位秒。
score	String	是	总分。
average_score	String	是	平均分。

KugouParams_accompany_opus_url

请求参数
被如下接口引用：accompany_opus_url

名称	类型	必选	描述

opus_id	String	是	作品 id。

KugouParams_accompany_search_song

请求参数

被如下接口引用：accompany_search_song

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
keyword	String	是	关键词。

KugouParams_accompany_search_tips

请求参数

被如下接口引用：accompany_search_tips

名称	类型	必选	描述
keyword	String	是	关键词。

KugouParams_accompany_singer_list

请求参数

被如下接口引用：accompany_singer_list

名称	类型	必选	描述
type	Integer	是	歌手类型。 • 1：全部。 • 2：男歌手。 • 3：女歌手。 • 4：组合。
area	Integer	是	歌手区域。 • 1：华语。 • 2：欧美。 • 3：日韩。

KugouParams_accompany_singer_song

请求参数

被如下接口引用：accompany_singer_song

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
singer_id	String	是	歌手 id。

KugouParams_accompany_songpitch

请求参数

被如下接口引用：accompany_songpitch

名称	类型	必选	描述

accompany_id	String	是	伴奏 id。

KugouParams_accompany_theme_list

请求参数

被如下接口引用：accompany_theme_list

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。

KugouParams_accompany_theme_song

请求参数

被如下接口引用：accompany_theme_song

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
theme_id	String	是	主题 id。

KugouParams_accompany_top_list

请求参数这里为空，传递空对象 {} 即可。

被如下接口引用：accompany_top_list。

KugouParams_accompany_top_song

请求参数

被如下接口引用：accompany_top_song

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
group_id	String	是	分组 id。
top_id	String	否	榜单 id。

KugouParams_accompany_url

请求参数

被如下接口引用：accompany_url

名称	类型	必选	描述
accompany_id	String	是	伴奏 id。

KugouParams_album_info

请求参数

被如下接口引用：album_info

名称	类型	描述
page	Integer	第?页。

size	Integer	每页?项。
album_id	String	专辑 id。

KugouParams_asset_purchased_albums

请求参数
被如下接口引用：asset_purchased_albums

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。

KugouParams_asset_purchased_songs

请求参数
被如下接口引用：asset_purchased_songs

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。

KugouParams_awesome_everyday

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：awesome_everyday。

KugouParams_awesome_newsong

请求参数
被如下接口引用：awesome_newsong

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。
top_id	Int	榜单 id。 • 1：华语。 • 2：欧美。 • 3：韩语。 • 4：日语。

KugouParams_awesome_recommend

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：awesome_recommend。

KugouParams_child_home

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：child_home。

KugouParams_child_resource

请求参数
被如下接口引用：child_resource

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
resource_type	Integer	是	大分类 7-小分类
resource_id	String	是	资源 id。

KugouParams_favorite_oper_playlist

请求参数

被如下接口引用：favorite_oper_playlist

名称	类型	描述
cmd	Integer	• 1：收藏歌单。 • 2：取消收藏。
playlist_id	String	歌单 id。
playlist_extra_id	String	歌单附加 id。 • 收藏时传入歌单 id。 • 取消收藏时传入获取歌单列表(或收藏歌单)时返回的同名字段。

KugouParams_favorite_oper_song

请求参数

被如下接口引用：favorite_oper_song

名称	类型	描述
cmd	Integer	• 1：收藏歌单。 • 2：取消收藏。
playlist_id	String	歌单 id。
song_id	String	歌曲 id。
song_extra_id	String	歌曲附加 id。 • 收藏时传入歌曲 id 。 • 取消收藏时传入获取歌单歌曲列表时返回的同名字段。

KugouParams_favorite_other_list

请求参数这里为空，传递空对象 {} 即可。

被如下接口引用：favorite_other_list。

KugouParams_favorite_self_list

请求参数这里为空，传递空对象 {} 即可。

被如下接口引用：favorite_self_list。

KugouParams_favorite_song

请求参数

被如下接口引用：favorite_song

名称	类型	描述
playlist_id	String	歌单 id。

page	Integer	第?页。
size	Integer	每页?项。
type	String	• 默认：other。 • self：自建歌单。 • other：收藏的歌单。

KugouParams_give_common_record

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：give_common_record。

KugouParams_give_common_scancodelogin

请求参数
被如下接口引用：give_common_scancodelogin

名称	类型	描述
partner_no	String	厂商订单号(同 sp 内唯一)。

KugouParams_live_program_list

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：live_program_list。

KugouParams_live_room_status

请求参数
被如下接口引用：live_room_status

名称	类型	必选	描述
room_id	String	是	房间 id。

KugouParams_live_room_url

请求参数
被如下接口引用：live_room_url

名称	类型	必选	描述
room_id	String	是	房间 id。

KugouParams_longaudio_free_home

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：longaudio_free_home。

KugouParams_longaudio_resource

请求参数
被如下接口引用：longaudio_resource

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
tag_id	String	是	分类 tag_id。

KugouParams_monitor_delay

请求参数

被如下接口引用：monitor_delay

名称	类型	必选	描述
pid	String	是	牛方案 2.0 pid，产品接入时申请获得。
device_id	String	是	厂商设备唯一标示，由 ascii 可见字符组成。
client_ver	String	是	客户端版本号，厂商自行填写自家版本号，用于辅助监控统计。
platform	String	是	客户端平台版本信息，用于辅助监控统计。
timestamp	Integer	是	unix 时间戳，单位秒。
nonce	String	是	随机字串，由 ascii 可见字符组成。
data	Array Of data_monitor_delay of data_monitor_delay	是	数据。

KugouParams_monitor_listen

请求参数

被如下接口引用：monitor_listen

名称	类型	必选	描述
pid	String	是	牛方案 2.0 pid，产品接入时申请获得。
device_id	String	是	厂商设备唯一标示，由 ascii 可见字符组成。
timestamp	Integer	是	unix 时间戳，单位秒。
nonce	String	是	随机字串，由 ascii 可见字符组成。
data	Array Of data_monitor_listen of data_monitor_listen	是	数据。

KugouParams_mv_category_list

请求参数这里为空，传递空对象 {} 即可。

被如下接口引用：mv_category_list。

KugouParams_mv_category_mv

请求参数

被如下接口引用：mv_category_mv

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
category_id	String	是	分类 id。
sort	Integer	是	排序。 • 1：发布时间。 • 2：标签创建时间。 • 3：播放量（倒序）。 • 4：算法。

			• 5：mv 飙升值。

KugouParams_mv_info

请求参数

被如下接口引用：mv_info

名称	类型	必选	描述
mv_id	String	是	mv_id

KugouParams_mv_infos

请求参数

被如下接口引用：mv_infos

名称	类型	必选	描述
mvs_id	Array Of String	是	mvid 列表。

KugouParams_mv_live_tme

请求参数

被如下接口引用：mv_live_tme

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
live_type	Integer	否	1-所有视频，2-完整演唱会视频。

KugouParams_mv_search

请求参数

被如下接口引用：mv_search

名称	类型	必选	描述
userid	String	是	TME 用户 id。
page	Integer	是	第?页。
size	Integer	是	每页?项。
keyword	String	是	关键词。

KugouParams_mv_trial

请求参数

被如下接口引用：mv_trial

名称	类型	必选	描述
mv_id	String	是	mv_id

KugouParams_mv_url

请求参数

被如下接口引用：mv_url

名称	类型	必选	描述
mv_id	String	是	mv_id

KugouParams_playlist_awesome

请求参数
被如下接口引用：playlist_awesome

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。

KugouParams_playlist_song

请求参数
被如下接口引用：playlist_song

名称	类型	描述
playlist_id	String	歌单分类 id。
page	Integer	第?页。
size	Integer	每页?项。

KugouParams_playlist_top

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：playlist_top。

KugouParams_radio_list

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：radio_list。

KugouParams_radio_song

请求参数
被如下接口引用：radio_song

名称	类型	描述
radio_id	String	电台 id。

KugouParams_rigion_home

请求参数
被如下接口引用：rigion_home

名称	类型	必选	描述
id	String	是	专区 id。

KugouParams_rigion_oldman_index

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：rigion_oldman_index。

KugouParams_search_song

请求参数
被如下接口引用：search_song

名称	类型	必选	描述
page	Integer	是	第?页。
size	Integer	是	每页?项。
keyword	String	是	关键词。

KugouParams_search_tips

请求参数
被如下接口引用：search_tips

名称	类型	必选	描述
keyword	String	是	关键词。

KugouParams_search_voice

请求参数
被如下接口引用：search_voice

名称	类型	必选	描述
query	String	是	用户语音在处理后的原始文本。
slots	Array Of slots_search_voice of slots_search_voice	是	语音搜索列表。

KugouParams_singer_album

请求参数
被如下接口引用：singer_album

名称	类型	描述
singer_id	String	歌手 id。
page	Integer	第?页。
size	Integer	每页?项。
sort	Integer	排序。 • 1：最热 。 • 2：最新。

KugouParams_singer_info

请求参数
被如下接口引用：singer_info

名称	类型	描述
singer_id	String	歌手 id。

KugouParams_singer_list

请求参数
被如下接口引用：singer_list

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。
area	Integer	歌手地区。 • 0：全部。 • 1：内地。 • 2：欧美。 • 3：日本。 • 4：韩国。 • 5：港台。 • 6：其它。
type	Integer	歌手类型。 • 0：全部。 • 1：男。 • 2：女。 • 3：组合。

KugouParams_singer_song

请求参数

被如下接口引用：singer_song

名称	类型	描述
singer_id	String	歌手 id。
page	Integer	第?页。
size	Integer	每页?项。

KugouParams_song_accompany

请求参数

被如下接口引用：song_accompany

名称	类型	描述
song_id	String	歌曲 id。

KugouParams_song_infos

请求参数

被如下接口引用：song_infos

名称	类型	描述
songs_id	Array of songs_id_song_infos	歌曲 id 列表。

KugouParams_song_lyric

请求参数

被如下接口引用：song_lyric

名称	类型	描述
song_id	String	歌曲 id。

KugouParams_song_tolisten

请求参数
被如下接口引用：song_tolisten

名称	类型	描述
song_id	String	歌曲 id。

KugouParams_song_url

请求参数
被如下接口引用：song_url

名称	类型	描述
song_id	String	歌曲 id。

KugouParams_top_list

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：top_list。

KugouParams_top_song

请求参数
被如下接口引用：top_song

名称	类型	描述
page	Integer	第?页。
size	Integer	每页?项。
top_id	String	榜单 id。

KugouParams_user_info

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：user_info。

KugouParams_user_logout

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：user_logout。

KugouParams_user_qrcode_auth

请求参数
被如下接口引用：user_qrcode_auth

名称	类型	描述
ticket	String	获取二维码授权信息的凭证。

KugouParams_user_qrcode_get

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：user_qrcode_get。

KugouParams_vip_activity_partner_give

请求参数
被如下接口引用：vip_activity_partner_give

名称	类型	必选	描述
partner_no	String	是	厂商订单号(同 sp 内唯一)。
month	Integer	是	购买月份数。
day	Integer	是	购买天数，天数和月数二选一，不可同时为0。

KugouParams_vip_ksing_info

请求参数这里为空，传递空对象 {} 即可。
被如下接口引用：vip_ksing_info。

KugouParams_vip_ksing_partner_give

请求参数
被如下接口引用：vip_ksing_partner_give

名称	类型	必选	描述
day	Integer	是	1. 购买天数，天数和月数二选一，不可同时为0。 2. 天数为1为酷狗线上测试充值渠道(请提前联系 TME 导入配额)，酷我Q音目前暂时不支持。
month	Integer	是	购买月份数。
partner_no	String	是	厂商订单号(同 sp 内唯一)。

KugouParams_vip_svip_common_give

请求参数
被如下接口引用：vip_svip_common_give

名称	类型	必选	描述
partner_no	String	是	厂商订单号(同 sp 内唯一)。
pay_no	String	是	用户支付订单号。
day	Integer	是	1. 购买天数，天数和月数二选一，不可同时为0。 2. 天数为1为酷狗线上测试充值渠道(请提前联系 TME 导入配额),酷我Q音目前暂时不支持。
month	Integer	是	购买月份数。

KugouParams_vip_tme_partner_give

请求参数
被如下接口引用：vip_tme_partner_give

名称	类型	必选	描述
userid	String	是	TME 用户 id。
token	String	否	TME 用户 token，非首次充值支持免 token（KW 和 QM 传过期或有效 token，KG 传有效 token 或空）。
partner_no	String	是	厂商订单号(同 sp 内唯一)。
day	Integer	是	1. 购买天数，天数和月数二选一，不可同时为0。 2. 天数为1为酷狗线上测试充值渠道(请提前联系 TME 导入配额)，酷我Q音目前暂时不支持。
month	Integer	是	购买月份数。

KugouParams_wechat_appinfocode

请求参数
被如下接口引用：wechat_appinfocode

名称	类型	描述
package	String	包名需事先向 TME 注册。

KugouParams_wechat_minipauth

请求参数
被如下接口引用：wechat_minipauth

名称	类型	描述
auth_code	string	获取用户信息凭证。

KugouParams_wechat_openlink

请求参数
被如下接口引用：wechat_openlink

名称	类型	描述
package	String	包名需事先向TME注册对应手机 APP 的包名（如没有对应手机 APP 可暂时预留一个）。

KugouParams_wechat_qrcodeauth

请求参数
被如下接口引用：wechat_qrcodeauth

名称	类型	描述
ticket	String	获取扫码授权结果的凭证。

KugouParams_wechat_qrcodeget

请求参数
被如下接口引用：wechat_qrcodeget

名称	类型	描述
package	String	包名需事先向 TME 注册。

Location

经纬度
被如下接口引用：AppGetDeviceLocation、AppGetDeviceLocationHistory

名称	类型	描述
Longitude	float32	经度
Latitude	float32	纬度

MemberList

被如下接口引用：AppGetFamilyMemberList

名称	类型	描述
UserId	String	用户 Id。

NickName	String	用户昵称。
Avatar	String	头像。
Role	Int	角色类型。 • 0是普通成员。 • 1是管理员。

MessageAttachment

消息加信息

名称	类型	描述
ShareToken	String	分享设备或邀请成员加入家庭时，用于绑定的凭证。

Msgs

消息体

名称	类型	描述
UserId	String	接受者用户 Id。
FromUserId	String	发送者用户 Id。
MsgId	String	消息 Id。
Category	Int	主类型（1：设备，2：家庭，3：通知）。
MsgType	Int64	类型。 • 101: 设备告警提醒。 • 200：成员添加结果通知。 • 201: 成员删除结果通知。 • 202：社区成员添加结果通知。 • 203: 社区成员删除结果通知。 • 204：邀请成员加入。 • 300: 设备分享结果通知。 • 301: 设备分享邀请。
MsgTitle	String	消息标题。
MsgContent	String	消息内容。
MsgTimestamp	Int64	消息时间戳。
ProductId	String	产品 ID。
DeviceName	String	设备名。
DeviceAlias	String	设备别名。
FamilyId	String	家庭 ID。
FamilyName	String	家庭名称。
RelateUserId	String	关联用户，在家庭消息中，一般指被添加或删除的用户。
CreateAt	String	创建时间，RFC3339 格式。
Attachments	Object of MessageAttachment	消息加信息。

Position

位置
被如下接口引用：AppGetDeviceLocation、AppGetDeviceLocationHistory

名称	类型	描述
Location	Location	经纬度。
CreateTime	int64	创建时间。

ProductList

关联产品列表

名称	类型	描述
ProductId	String	产品 ID。
Name	String	产品名称。
Description	String	产品描述。
UIConfig	String	交互配置信息 json 字符串，包括产品图片、配网引导等。

Products

产品列表

名称	类型	描述
ProductId	String	产品 ID。
Name	String	产品名称。
Description	String	产品描述。
DataTemplate	String	产品数据模板。
NetType	String	产品网络连接类型。
CategoryId	Int32	产品分类 ID。
ProductType	Int32	产品类型。
UpdateTime	Int64	最后更新的 unix 时间戳（秒级）。

ProductsConfigRsp

产品界面配置返回值
被如下接口引用：AppGetProductsConfig

名称	类型	描述
ProductId	String	产品ID
Config	String	产品配置

Property

条件属性被如下接口引用：AppCreateAutomation、AppDescribeAutomation、AppModifyAutomation

名称	类型	必选	描述
ProductId	String	是	产品 ID。
DeviceName	String	是	设备名称。

AliasName	String	否	设备别名。
IconUrl	String	否	产品图标 URL。
PropertyId	String	否	产品属性 ID。
Op	String	是	条件操作符。 • eq：等于。 • ne：不等于。 • gt：大于。 • lt：小于。 • ge：大于等于。 • le：小于等于。
Value	Any	是	比较的值。

RiskItem

扣分项结构体被如下接口引用：AppGetInsuranceEvaluation

名称	类型	必选	描述
Name	String	是	扣分项名称。
Reason	String	是	隐患分析。
Suggestion	String	是	处置建议。

Roomlist

房间列表信息被如下接口引用：AppGetRoomList

名称	类型	描述
RoomId	String	房间的所属 Id。
RoomName	String	房间名称。
DeviceNum	Int	房间拥有的设备数。
CreateTime	Int	创建时间，unix 时间戳(秒级)。
UpdateTime	Int	最后更新时间，unix 时间戳(秒级)。

Scene

场景实体被如下接口引用：AppGetSceneList

名称	类型	必选	描述
SceneId	String	是	手动智能联动 ID。
FamilyId	String	是	手动智能联动所属家庭 ID。
SceneName	String	是	手动智能联动的名称。
SceneIcon	String	否	手动智能联动的背景图片地址。
Actions	Array of Action	是	手动智能联动动作列表。
UserId	String	是	用户 ID。
CreateTime	int	否	创建时间戳。
UpdateTime	int	否	更新时间戳。

Flag	int	否	设备状态。 • 0：正常。 • 1：设备异常。
Status	int	否	运行状态。 • 0：未运行。 • 1：运行中。

ShareDeviceTokenInfo

响应结果被如下接口引用：AppDescribeShareDeviceToken

名称	类型	描述
FromUserId	String	分享用户 Id。
FromUserNick	String	分享用户昵称。
UserId	String	被分享用户 Id。
UserNick	String	被分享用户昵称。
ProductId	String	产品 ID。
DeviceName	String	设备名。
AliasName	String	设备别名。
IconUrl	String	图标 URL。
BindTime	Int64	绑定的 unix 时间戳（秒级）。
ExpireTime	Int64	有效期（秒级）。
CreateTime	Int64	创建的 unix 时间戳(秒级)。

ShareDevices

分享设备信息被如下接口引用：AppDescribeShareDevice、AppListUserShareDevices

名称	类型	描述
ProductId	String	产品 ID。
DeviceName	String	设备名称。
DeviceId	String	设备 ID，是产品 ID/设备名称拼接而来。
IconUrl	String	图标的 url。
AliasName	String	设备别名、备注。
CreateTime	String	设备创建时间，这里是 RFC3339 的格式。

ShareUserInfo

分享用户信息被如下接口引用：AppListShareDeviceUsers

名称	类型	描述
UserId	String	被分享用户 Id。
NickName	String	昵称。
CountryCode	String	国家码。

PhoneNumber	String	手机号。
BindTime	Int64	绑定时间，秒级 UNIX 时间戳。

Timer

条件触发时间被如下接口引用：AppCreateAutomation、AppDescribeAutomation、AppModifyAutomation

名称	类型	必选	描述
Days	String	是	定时器开启时间，每一位——0:关闭，1:开启，从左至右依次表示：周日、周一、周二、周三、周四、周五、周六。
TimePoint	String	是	定时器开启时间点。

TimerList

定时器列表信息被如下接口引用：AppGetTimerList

名称	类型	描述
TimerId	String	定时器 ID。
TimerName	String	定时器名称。
ProductId	String	定时器所属的产品 ID。
DeviceName	String	定时器对应的设备名称。
Days	String	定时器开启时间，每一位——0是关闭，1是开启，从左至右依次表示：周日、周一、周二、周三、周四、周五、周六。
TimePoint	String	定时器开启时间点。
Repeat	Int	是否循环。 • 0：不需要。 • 1：需要。
Data	String	定时器启动时下发的状态。
Status	String	开启状态。0表示关闭，1表示开启。
CreateTime	Int64	创建时间，unix 时间戳(秒级)。
UpdateTime	Int64	最后一次更新时间，unix 时间戳(秒级)。

TokenResultInfo

用户信息被如下接口引用：AppGetToken、AppGetTokenByWeiXin

名称	类型	描述
ExpireAt	Int64	截止时间，Unix 秒级时间戳。
Token	String	开发平台返回的 AccessToken，通过该 Token 进行登录后的接口请求。
CancelAccountTime	Int64	仅当用户提交注销申请（AppUserCancelAccount）后再次登录时返回（仅申请首次登录返回），此字段用来返回注销申请提交时间，提醒用户注销流程已取消，Unix 秒级时间戳。

UserSetting

用户设置被如下接口引用：AppGetUserSetting

名称	类型	必选	描述

EnableWechatPush	Int	否	微信推送。
EnableDeviceMessagePush	Int	否	设备信息推送。
EnableFamilyMessagePush	Int	否	家庭信息推送。
EnableNotifyMessagePush	Int	否	通知信息推送。
AllowEditions	Int	否	小程序白名单版本。0表示仅限家庭版，1表示家庭版和社区版都允许。
UsingEdition	Int	否	小程序当前使用版本。0表示家庭版，1表示社区版。
TemperatureUnit	String	否	温度单位：C 表示摄氏度， F 表示华氏度。
Region	String	否	时区设置，有效时区参见：AppGetGlobalConfig 中的 RegionListEN 或 RegionListCN。
NoDisturbTime	String	否	消息免打扰时段，为空则表示未开启免打扰，开启时则用：HH:MM-HH:MM 表示免打扰时段，例如：21:00-08:00。
NoDisturbDeviceList	Array of String	否	免打扰设备 ID 列表。
NoDisturbFamilyList	Array of String	否	免打扰家庭列表。

YunApiParam

设备心跳接口被如下接口引用：HeartBeat

名称	类型	必选	描述
AccessToken	String	是	AccessToken 为调用登录接口之后获取 Token。
ActionParams	Object of HeartBeatParam	是	设备心跳接口入参。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。

accompany_accompany_free_info

体验信息被如下接口引用：accompany_free_info

名称	类型	必选	描述
accompany_id	String	否	已经体验的伴奏 id。

accompany_accompany_info

伴奏列表被如下接口引用：accompany_info

名称	类型	必选	描述
has_original	Integer	否	音频是否有原唱音轨。 • 0：无 。 • 1：有。
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
mv_id	String	是	MVID
hash_key	String	是	hashkey

song_id	String	是	歌曲 id。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。
has_original	Integer	是	音频是否有原唱音轨。 • 0：无。 • 1：有。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。
krc_id	String	是	歌词 ID。
original_mv_id	String	是	原版本 MV ID。

accompany_awesome_personal

伴奏个性推荐
被如下接口引用：accompany_awesome_personal

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。
has_original	Integer	是	音频是否有原唱音轨。 • 0：没有。 • 1：有。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。

has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_awesome_top

伴奏推荐列表被如下接口引用：accompany_awesome_top

名称	类型	必选	描述
weight	Integer	是	热度值/飙升值。
is_hq	Integer	是	是否显示HQ标识。 • 0：不显示。 • 1：显示。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。
has_original	Integer	是	音频是否有原唱音轨。 • 0：没有。 • 1：有。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_category_song

伴奏列表被如下接口引用：accompany_category_song

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。

has_original	Integer	是	音频是否有原唱音轨。 • 0：没有。 • 1：有。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_favorite_list

伴奏列表被如下接口引用：accompany_favorite_list

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
has_original	Integer	否	音频是否有原唱音轨。 • 0：没有。 • 1：有。
singer_name	String	是	歌手名。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名字。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 id。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_free_top_song

伴奏列表被如下接口引用：accompany_free_top_song

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
has_original	Integer	否	音频是否有原唱音轨。 • 0：没有。

			• 1：有。
singer_name	String	是	歌手名。
free_token	String	是	免费令牌。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名字。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 id。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_newsearch_song

伴奏列表被如下接口引用：accompany_newsearch_song

名称	类型	必选	描述
song_hot	Integer	是	热度。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_search_song

伴奏搜索被如下接口引用：accompany_search_song

名称	类型	必选	描述
song_hot	Integer	是	热度。
accompany_id	String	是	伴奏 id。

song_name	String	是	伴奏名。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
duration	Integer	是	伴奏时长单位毫秒。
album_img	String	是	专辑图片。
has_original	Integer	是	音频是否有原唱音轨。 • 0：无。 • 1：有。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_singer_song

伴奏列表被如下接口引用：accompany_singer_song

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。 • 1：有。
singer_name	String	是	歌手名。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名字。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 id。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_song_accompany

伴奏

被如下接口引用：song_accompany

名称	类型	描述
accompany_id	String	伴奏 id。
song_name	String	伴奏名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
duration	Integer	伴奏时长单位毫秒。
album_img	String	专辑图片。
has_original	Integer	音频是否有原唱音轨。 • 0：无。 • 1：有。
is_ktv	Integer	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	比特率。
has_pitch	Integer	是否有音高。 • 0：没有。 • 1：有。

accompany_theme_song

伴奏列表
被如下接口引用：accompany_theme_song

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。 • 1：有。
singer_name	String	是	歌手名。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名字。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 id。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

accompany_top_song

伴奏列表

被如下接口引用：accompany_top_song

名称	类型	必选	描述
is_hq	Integer	是	是否显示 HQ 标识。 • 0：不显示。 • 1：显示。
has_original	Integer	否	音频是否有原唱音轨。 • 0：无。 • 1：有。
singer_name	String	是	歌手名。
accompany_id	String	是	伴奏 id。
song_name	String	是	伴奏名字。
duration	Integer	是	伴奏时长单位毫秒。
singer_id	String	是	歌手 id。
album_img	String	是	专辑图片。
is_ktv	Integer	是	是否有 KTV。 • 0：没有。 • 1：有。
bit_rate	Integer	是	比特率。
has_pitch	Integer	是	是否有音高。 • 0：没有。 • 1：有。

albums_asset_purchased_albums

专辑信息

被如下接口引用：asset_purchased_albums

名称	类型	描述
album_id	String	专辑 id。
album_name	String	专辑名。
album_translate_name	String	专辑译名。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
intro	String	专辑简介。
company	String	唱片公司。
singer_id	String	歌手 id。
singer_name	String	歌手名。

publish_time	String	发布时间.

albums_singer_album

专辑信息

被如下接口引用：singer_album

名称	类型	描述
album_id	String	专辑 id。
album_name	String	专辑名。
album_translate_name	String	专辑译名。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
intro	String	专辑简介。
company	String	唱片公司。
singer_id	String	歌手 id。
singer_name	String	歌手名。
publish_time	String	发布时间。

appapiCommon

AppApi 公共参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppCreateEmailUser。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。
AppKey	String	是	公共参数，应用 AppKey ，用于标识对应的 App。
Signature	String	是	公共参数，请求签名，需用户自行生成，用于校验请求的合法性。
Timestamp	Int64	是	公共参数，请求的 Unix 时间戳（秒级）。
Nonce	Int	是	公共参数，随机正整数，与 Timestamp 联合起来，防止重放攻击。

author_newsearch_song

歌手列表

被如下接口引用：accompany_newsearch_song

名称	类型	必选	描述
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
singer_img	String	是	歌手图片。

categories_accompany_category_list

分类
被如下接口引用：accompany_category_list

名称	类型	必选	描述
category_name	String	是	分类名。
category_id	String	是	分类 id。

categories_mv_category_list

分类列表
被如下接口引用：mv_category_list

名称	类型	必选	描述
category_id	String	是	分类 id。
category_name	String	是	分类名。

data_monitor_delay

数据
被如下接口引用：monitor_delay

名称	类型	必选	描述
api	String	是	api path，必须与 api 实际 path 保持一致。
userid	String	是	TME 用户 id。
client_ip	String	是	接口使用者 ip，如设备直接调用 TME api 则上报设备 ip，反之则厂商服务器 ip。
server_ip	String	是	接口实际调用时的服务器 ip。
ecode	Integer	是	业务错误码（error_code）。
reqtime	Integer	是	请求时间时间戳单位秒。
rspsize	Integer	是	响应包大小，单位字节。
sp	String	是	服务提供商对应枚举值分别为酷狗，酷我，Q音。
apn	String	是	网络类型。
etype	Integer	是	错误类型。0：调用成功。1：网络错误。2：http 错误码。3：业务错误码。
reqsize	Integer	是	请求包大小，单位字节。
retry	Integer	是	重试次数（本次是第几次重试）。
delay	Integer	是	接口调用时延，单位 ms。

data_monitor_listen

数据
被如下接口引用：monitor_listen

名称	类型	必选	描述
play_quality	String	否	歌曲/长音频播放品质 d，取值：SQ、HQ、standard；视频播放品质：FHD、HD、LD、QHD、SD。
duration	Integer	是	歌曲时长（单位 ms）。
play_time	Integer	是	歌曲实际播放时长（单位 ms）。

play_type	Integer	是	• 1：在线。 • 2：缓存。 • 3：下载。
client_ip	String	是	接口实际使用者 ip，即硬件终端 ip。
media_type	Integer	否	• 1：听歌。 • 2：视频MV。 • 3：伴奏K歌业务。 • 4：有声音频。 • 5：直播。
lvt	String	是	用户播放时间 format: 2019-09-19 00:00:00。
try_play	Integer	否	0：可完整播放。1：vip 试听类型。2：免登录试听。 说明：可完整播放（该用户有完播权限），vip 试听类型（非 vip 用户试听 vip 歌曲片段），免登录试听（用户未登录，免费试听权限）。
sp	String	是	服务提供商对应枚举值分别为酷狗，酷我，Q音。
userid	String	是	TME 用户 id。
song_id	String	是	根据多媒体类型，赋值相应的 ID：歌曲 ID、伴奏 ID、MV 的 ID。
api	String	是	api path，必须与 api 实际 path 保持一致。
source_id	String	是	歌曲 id 来源, 歌曲来源为歌单上传歌单 id，歌曲来源为榜单，上传榜单 id。如果都不是，传空。

groups_accompany_category_list

伴奏分组

被如下接口引用：accompany_category_list

名称	类型	必选	描述
group_name	String	否	分组名。
categories	Array Of categories_accompany_category_list of categories_accompany_category_list	否	分类。

groups_accompany_free_top_list

分组列表

被如下接口引用：accompany_free_top_list

名称	类型	必选	描述
group_id	String	是	分组 id。
group_name	String	是	分组名。
tops	Array Of tops_accompany_free_top_list of tops_accompany_free_top_list	否	榜单列表。

groups_accompany_top_list

分组列表

被如下接口引用：accompany_top_list

名称	类型	必选	描述
group_id	String	是	分组 id。

group_name	String	是	分组名。
tops	Array Of tops_accompany_top_list of tops_accompany_top_list	否	tops

groups_child_home

资源分组
被如下接口引用：child_home

名称	类型	必选	描述
name	String	否	资源名称。
infos	Array Of infos_child_home of infos_child_home	否	资源列表。

groups_longaudio_free_home

资源分组
被如下接口引用：longaudio_free_home

名称	类型	必选	描述
name	String	否	资源名称。
tag_id	String	否	资源 tag_id，值为-1即没有更多的资源。
infos	Array Of infos_longaudio_free_home of infos_longaudio_free_home	否	资源列表。

groups_playlist_top

分组列表
被如下接口引用：playlist_top

名称	类型	描述
category_id	String	分类 id 推荐歌单传入同名字段获取相应歌单。
category_name	String	分类名。
category_icon	String	分类封面。

groups_radio_list

分组列表
被如下接口引用：radio_list

名称	类型	描述
group_id	String	分组 id。
group_name	String	分组名。
radios	Array of radios_radio_list	电台列表。

groups_rigion_home

资源分组
被如下接口引用：rigion_home

名称	类型	必选	描述

name	String	否	资源名称。
infos	Array Of infos_rigion_home of infos_rigion_home	否	资源列表。

groups_rigion_oldman_index

资源分组
被如下接口引用：rigion_oldman_index

名称	类型	必选	描述
id	Integer	否	专区 id。
name	String	否	专区名称。
pic	String	否	专区封面。
summary	String	否	描述。

groups_top_list

分组
被如下接口引用：top_list

名称	类型	描述
group_id	String	分组 id。
group_name	String	分组名。
tops	Array of tops_top_list	榜单。

horizontal_live_room_url

横屏流信息
被如下接口引用：live_room_url

名称	类型	必选	描述
rate	String	否	清晰度标识 (1：流畅。2：标清。3：高清。4：超清。5：1080P。)
hls	Array Of String	否	hls 流信息。
httpflv	Array Of String	否	httpflv 流信息。
httpsflv	Array Of String	否	httpsflv 流信息。
rtmp	Array Of String	否	rtmp 流信息。
httpshls	Array Of String	否	httpshls 流信息。
streamName	String	否	流名称。

infos_child_home

资源列表
被如下接口引用：child_home

名称	类型	必选	描述
resource_type	String	是	大分类。 3：歌单。4：专辑。7：小分类。
resource_id	String	是	资源 id。

name	String	是	资源名称。
pic	String	是	资源图。

infos_longaudio_free_home

资源列表
被如下接口引用：longaudio_free_home

名称	类型	必选	描述
resource_id	Integer	是	资源 id。
name	String	是	资源名称。
pic	String	是	资源封面。

infos_rigion_home

资源列表
被如下接口引用：rigion_home

名称	类型	必选	描述
resource_type	String	是	大分类。 2：单曲。3：歌单。103：歌单合集。4：专辑。104：专辑合集。5：歌手。6：排行榜。7：分类。8：MV。108：MV 合集。9：电台。109：电台合集。
resource_id	String	是	资源 id。
name	String	是	资源名称。
pic	String	是	资源封面。

letters_accompany_singer_list

letters
被如下接口引用：accompany_singer_list

名称	类型	必选	描述
letter	String	是	歌手首字母。
singers	Array Of singers_accompany_singer_list of singers_accompany_singer_list	是	歌手列表。

lists_child_resource

资源分组
被如下接口引用：child_resource

名称	类型	必选	描述
id	String	是	资源 id。
name	String	是	资源名称。
pic	String	是	资源图片。

lists_longaudio_resource

资源列表
被如下接口引用：longaudio_resource

名称	类型	必选	描述
resource_id	Integer	是	资源 id。
name	String	是	资源名称。
pic	String	是	资源封面。

mvInfos_mv_infos

mv 信息
被如下接口引用：mv_infos

名称	类型	必选	描述
author_name	String	否	MV 制作人名称。
tags	tags_mv_infos	否	mv 信息。
mv_id	String	是	MV id
mv_name	String	是	MV 名称。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名称。
accompany_id	String	否	伴奏 id。
intro	String	否	简介。
song_id	String	是	歌曲 id。
mv_img	String	是	MV 图片。

mvs_mv_category_mv

MV 列表
被如下接口引用：mv_category_mv

名称	类型	必选	描述
mv_img	String	是	MV 图片。
mv_name	String	是	MV 名称。
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名称。
song_id	String	是	歌曲 id。
mv_id	String	是	MV id

mvs_mv_live_tme

MV 列表
被如下接口引用：mv_live_tme

名称	类型	必选	描述
mv_id	String	是	MV id
mv_img	String	是	MV 图片。
mv_name	String	是	MV 名称。

singer_id	String	是	歌手 id。
singer_name	String	是	歌手名称。
song_id	String	是	歌曲 id。

mvs_mv_search

mv 列表
被如下接口引用：mv_search

名称	类型	必选	描述
singer_name	String	是	歌手名称。
has_acc	Integer	是	是否有伴奏。 • 0：没有。 • 1：有。
mv_id	String	是	MV id
mv_img	String	是	MV 图片。
mv_name	String	是	MV 名称。
song_id	String	是	歌曲 id。

opuses_accompany_opus_list

opuses
被如下接口引用：accompany_opus_list

名称	类型	必选	描述
opus_name	String	是	作品名称。
score	Integer	否	总分数。
grade	String	否	等级。
listen_num	Integer	是	试听人数。
album_img	String	是	专辑封面。
accompany_id	String	是	伴奏 id。
opus_id	String	是	作品 id。
average_score	Integer	否	平均分。
is_private	Integer	是	是否公开。 • 0：公开作品。 • 1：私密作品。
duration	Integer	是	作品播放时间，单位秒。

playlists_favorite_other_list

歌单
被如下接口引用：favorite_other_list

名称	类型	描述
playlist_id	String	歌单 id。

playlist_name	String	歌单名称。
pic	String	歌单封面。
playlist_extra_id	String	歌单附加 id。

playlists_favorite_self_list

歌单

被如下接口引用：favorite_self_list

名称	类型	描述
playlist_id	String	歌单 id。
playlist_name	String	歌单名称。
pic	String	歌单封面。
update_time	String	歌单更新时间。
create_time	String	歌单创建时间。
total	Integer	歌单歌曲数量。

playlists_playlist_awesome

歌单

被如下接口引用：playlist_awesome

名称	类型	描述
playlist_id	String	歌单 id。
playlist_name	String	歌单名称。
pic	String	歌单封面。
update_time	String	歌单更新时间。
create_time	String	歌单创建时间。
intro	string	歌单简介。

programs_live_program_list

节目分组

被如下接口引用：live_program_list

名称	类型	必选	描述
room_id	Integer	是	房间 id。
status	String	是	节目状态。0是未开播，1是开播。
price	String	否	价格。
name	String	是	节目名称。
startTime	String	是	开始时间。
endTime	String	是	结束时间。
intro	String	是	节目描述。
pic	String	是	封面图片。

pay_type	Integer	否	是否付费演出会。0：不是。1：是。

radios_radio_list

电台列表

被如下接口引用：radio_list

名称	类型	描述
radio_id	String	电台 id。
radio_name	String	电台名。
radio_img	String	电台封面。

recommends_accompany_category_list

推荐分类

被如下接口引用：accompany_category_list

名称	类型	必选	描述
category_id	String	是	分类 id。
category_name	String	是	分类名。
category_img	String	是	分类图片。

singers_accompany_singer_list

热门歌手

被如下接口引用：accompany_singer_list

名称	类型	必选	描述
singer_id	String	是	歌手 id。
singer_name	String	是	歌手名。
singer_img	String	是	歌手图片。

singers_accompany_singer_list_1

歌手列表

名称	类型	必选	描述
singer_name	String	是	歌手名。
singer_img	String	是	歌手图片。
singer_id	String	是	歌手 id。

singers_singer_list

歌手

被如下接口引用：singer_list

名称	类型	描述
singer_id	String	歌手 id。
singer_name	String	歌手名。

area	String	歌手地区。
translate_name	String	歌手译名。
singer_img	String	歌手相片。

slots_search_voice

语音搜索列表
被如下接口引用：search_voice

名称	类型	必选	描述
name	String	是	-
value	String	是	词槽参数值。

songs_album_info

歌曲列表
被如下接口引用：album_info

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否vip歌曲。0：否。1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。0：否。1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否。

		• 1：有。
language	String	语言。

songs_asset_purchased_songs

歌曲列表

被如下接口引用：asset_purchased_songs

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：是。
language	String	语言。

songs_awesome_everyday

歌曲列表

被如下接口引用：awesome_everyday

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
language	String	语言。

songs_awesome_newsong

歌曲列表

被如下接口引用：awesome_newsong

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。

album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
language	String	语言。

songs_awesome_recommend

歌曲列表

被如下接口引用：awesome_recommend

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。

is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
language	String	语言。

songs_favorite_song

歌曲列表

被如下接口引用：favorite_song

名称	类型	描述
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
song_extra_id	String	歌曲附加 id，仅获取自建歌单歌曲列表时返回（取消收藏时使用）。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。

		• 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。

songs_id_song_infos

歌曲id列表

被如下接口引用：song_infos

名称	类型	必选	描述
Values	Array of String	是	一个或者多个过滤值。

songs_playlist_song

歌曲列表

被如下接口引用：playlist_song

名称	类型	描述
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。

songs_radio_song

歌曲列表

被如下接口引用：radio_song

名称	类型	描述
singer_img	String	歌手头像。

album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
language	String	语言。

songs_search_song

歌曲列表

被如下接口引用：search_song

名称	类型	必选	描述
singer_name	String	是	歌手名。
album_id	String	是	歌曲所属专辑 id。
album_name	String	是	歌曲所属专辑名。
song_id	String	是	歌曲 id。
song_name	String	是	歌曲名。
singer_id	String	是	歌手 id。
playable_code	Integer	是	是否可以播放。枚举值依次对应： • 0：可以。

			• 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
has_accompany	Integer	是	是否有伴奏。 • 0：否 。 • 1：是。
singer_img	String	是	歌手头像。
album_img_mini	String	是	专辑封面100px左右。
language	String	否	语言。
is_vip_song	Integer	是	是否 vip 歌曲。 • 0：否 。 • 1：是。
album_img	String	是	专辑封面。
album_img_small	String	是	专辑封面300px左右。
album_img_medium	String	是	专辑封面500px左右。
mv_id	String	是	mv id

songs_search_voice

语音搜索歌曲列表
被如下接口引用：search_voice

名称	类型	必选	描述
singer_name	String	否	歌手名。
duration	Integer	是	歌曲时长，单位 ms。
song_id	String	是	歌曲 id。
song_name	String	是	歌曲名。

songs_singer_song

歌曲列表
被如下接口引用：singer_song

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。

is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
language	String	语言。

songs_song_infos

歌曲列表

被如下接口引用：song_infos

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。

		• 0：否 。 • 1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。
album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
topic_url	String	歌曲专题页面(购买链接)。
language	String	语言。

songs_song_tolisten

歌曲列表

被如下接口引用：song_tolisten

名称	类型	描述
singer_img	String	歌手头像。
album_img	String	专辑封面。
album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
album_img_medium	String	专辑封面500px左右。
is_vip_song	Integer	是否 vip 歌曲。0：否 。1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。0：否 。1：是。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
singer_name	String	歌手名。

album_id	String	专辑 id。
album_name	String	专辑名。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 9：未知原因，无权播放。
try_playable	Integer	是否有试听片段。 • 0：否 。 • 1：有。
topic_url	String	歌曲专题页面(购买链接)。
language	String	语言。

songs_top_song

歌曲列表

被如下接口引用：top_song

名称	类型	描述
singer_name	String	歌手名。
album_id	String	歌曲所属专辑 id。
album_name	String	歌曲所属专辑名。
song_id	String	歌曲 id。
song_name	String	歌曲名。
singer_id	String	歌手 id。
playable_code	Integer	是否可以播放。枚举值依次对应： • 0：可以。 • 1：海外地区不能播放。 • 2：歌曲无版权不能播放。 • 3：会员歌曲，非会员不能播放。 • 4：付费内容，须购买才可播放。 • 5：牛方案策略，非会员不能播放。 • 6：因定向版权下架不能播放（针对 APP 有权但设备端无权的情况）。 • 9：未知原因，无权播放。
is_vip_song	Integer	是否 vip 歌曲。 • 0：否 。 • 1：是。
mv_id	String	mv id
has_accompany	Integer	是否有伴奏。 • 0：否 。 • 1：是。
singer_img	String	歌手头像。

album_img_mini	String	专辑封面100px左右。
album_img_small	String	专辑封面300px左右。
language	String	语言。
album_img	String	专辑封面。
album_img_medium	String	专辑封面500px左右。

tags_mv_info

mv 信息
被如下接口引用：mv_info

名称	类型	必选	描述
tag_id	String	否	标签 id。
parent_id	String	否	标签 parentId。
tag_name	String	否	标签的名称。

tags_mv_infos

mv 信息
被如下接口引用：mv_infos

名称	类型	必选	描述
tag_id	String	否	标签 id。
parent_id	String	否	标签 parentId。
tag_name	String	否	标签的名称。

themes_accompany_theme_list

主题列表
被如下接口引用：accompany_theme_list

名称	类型	必选	描述
theme_thumbnail	String	是	主题缩略图。
theme_id	String	是	主题 id。
theme_name	String	是	主题名。
theme_img	String	是	主题图片。

tokenapiCommon

tokenApi的公共参数

名称	类型	必选	描述
Action	String	是	公共参数，本接口取值：AppLogoutUser。
AccessToken	String	是	公共参数，AccessToken 用于对一个已经登录的用户鉴权。
RequestId	String	是	公共参数，唯一请求 ID，可自行生成，推荐使用 uuId。定位问题时，需提供该次请求的 RequestId。

tops_accompany_free_top_list

榜单列表

被如下接口引用： accompany_free_top_list

名称	类型	必选	描述
top_id	String	是	榜单 id。
top_name	String	是	榜单名。
header_url	String	是	榜单图片。

tops_accompany_top_list

tops

被如下接口引用： accompany_top_list

名称	类型	必选	描述
top_id	String	是	榜单 id。
top_name	String	是	榜单名。

tops_top_list

榜单

被如下接口引用： top_list

名称	类型	描述
top_id	String	榜单 id。
top_name	String	榜单名。
header_url	String	榜单图片。

vertical_live_room_url

竖屏流信息

被如下接口引用： live_room_url

名称	类型	必选	描述
rtmp	Array Of String	否	rtmp 流信息。
httpshls	Array Of String	否	httpshls 流信息。
streamName	String	否	流名称。
rate	String	否	清晰度标识 (1：流畅。2：标清。3：高清。4：超清。5：1080P。)
hls	Array Of String	否	hls 流信息。
httpflv	Array Of String	否	httpflv 流信息。
httpsflv	Array Of String	否	httpsflv 流信息。

WechatTemplate

第三方微信模板

被如下接口引用： AppDescribeTemplateBinding

名称	类型	必选	描述

ModelId	String	否	产品型号ID。
TemplateId	String	否	微信模板ID
Title	String	否	微信模板标题
Status	Int	否	订阅状态：0未订阅；1已订阅；2不接收

WechatDeviceInfo

微信设备票据

被如下接口引用：APPGetWechatDeviceTicket

名称	类型	必选	描述
DeviceId	String	否	设备ID。
WXIoTDeviceInfo.SN	String	否	设备SN。
WXIoTDeviceInfo.SNTicket	String	否	设备票据。
WXIoTDeviceInfo.ModelId	String	否	产品型号ID。

设备配网开发
配网开发概述

最近更新时间：2022-06-10 15:24:53

简介

Wi-Fi 配网，指由外部向 Wi-Fi 设备提供 SSID 和密码（PSW），让 Wi-Fi 设备可以连接指定的热点或路由器，并加入后者所建立的 Wi-Fi 网络。

- 对于具备丰富人机界面包括屏幕/键盘的设备，例如电脑或手机，可以直接输入 SSID/PSW 进行连接。
- 对于不具备丰富人机交互界面的物联网 Wi-Fi 设备，例如智能灯、扫地机器人等，则可以借助手机等智能设备，以某种配网方式将 SSID/PSW 传递该设备。

特点

配网有多种方式，包括 SmartConfig、softAP、Airkiss 等，各自的特点比较如下：

配网方式	特点
WPS	存在安全性问题。
SmartConfig	较便捷，但一般为各厂商采用私有协议，兼容性和互操作性较差。
softAP	适配性兼容性较好，但手机端需要做两次 Wi-Fi 连接设置的切换，步骤较复杂。
Airkiss	操作便捷无需热点配置，但为微信客户端私有协议，需设备端适配兼容。
ble combo	操作便捷，但需要设备端支持 Wi-Fi+BLE 的 Combo 芯片方案。

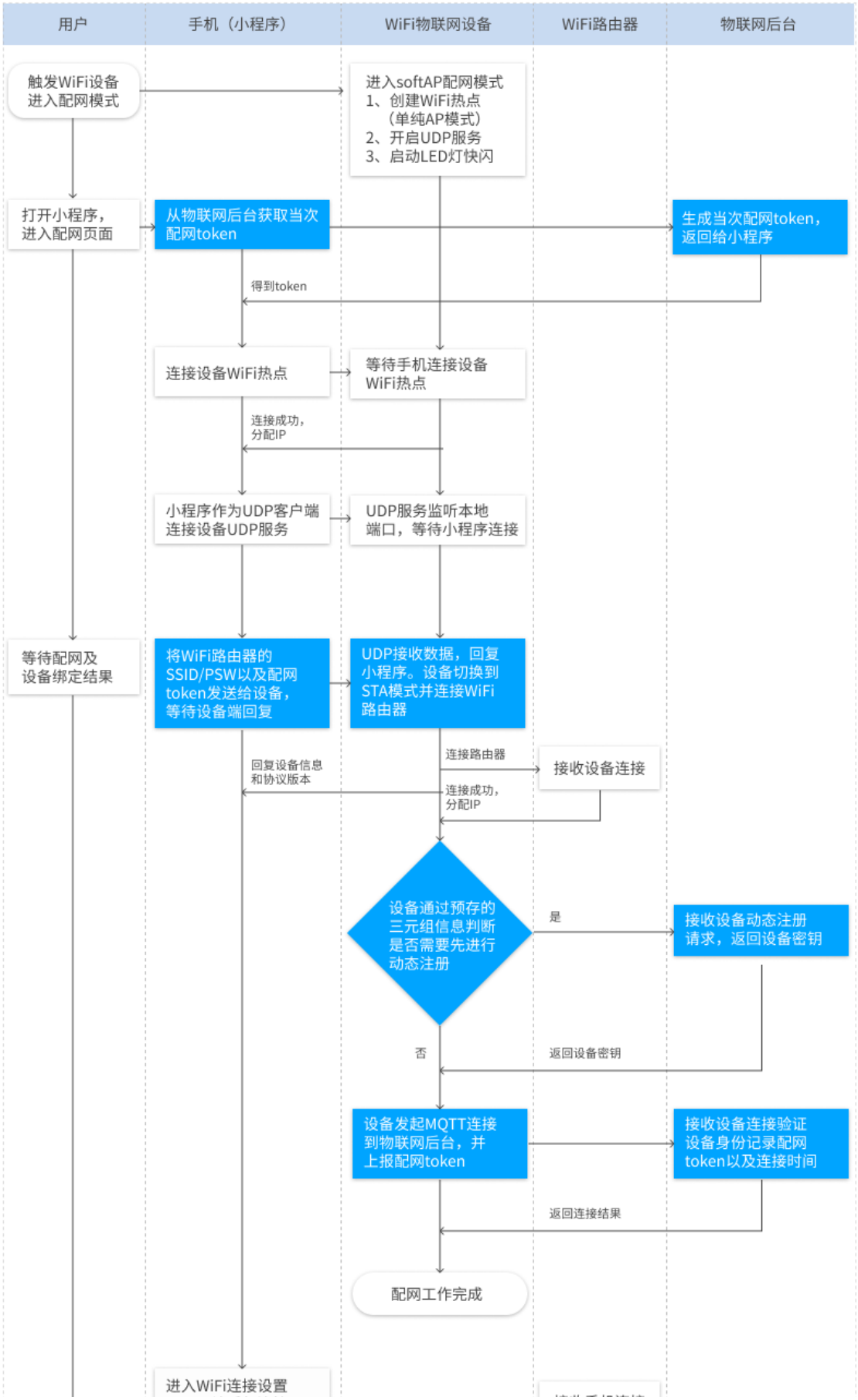
softAP 配网开发

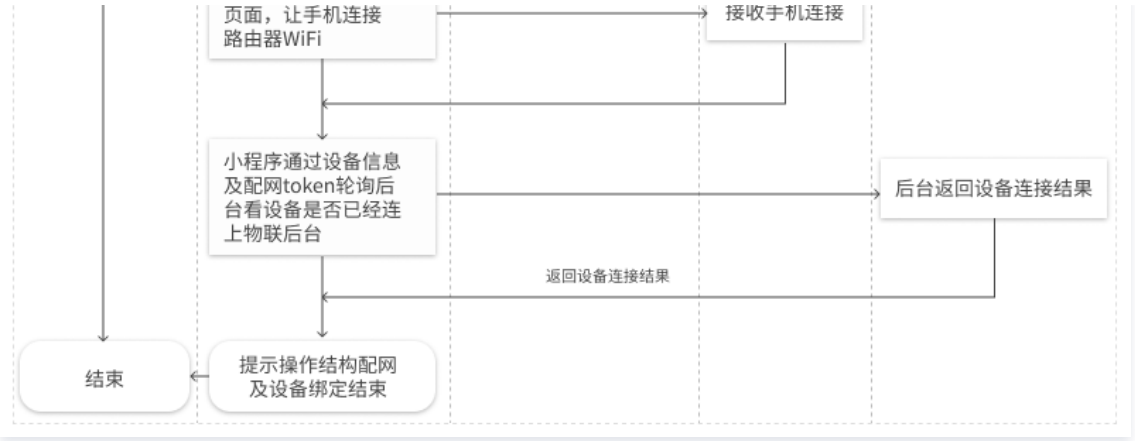
最近更新时间：2023-06-05 12:11:41

操作场景

设备通过 softAP 方式创建一个 Wi-Fi 热点，手机连接该热点，再通过数据通道例如 TCP/UDP 通讯，将目标 Wi-Fi 路由器的 SSID/PSW 传递该设备，设备获取后，即可连接 Wi-Fi 路由器从而连接互联网。同时，为了对设备进行绑定，手机 App 可以利用该 TCP/UDP 数据通道，将后台提供的配网 Token 发送给设备，并由设备转发至物联网后台，依据 Token 可以进行设备绑定。本文档主要指导您如何使用softAP 方式配网开发。
腾讯连连小程序已经支持 softAP 配网，并提供了相应的 小程序 SDK 。
基于 Token 的 softAP 方式配网及设备绑定的示例流程图，如下图所示：

softAP方式配网及设备绑定流程v2.0





操作步骤

softAP 配网协议示例

本示例基于 ESP8266 腾讯云定制模组配合腾讯连连小程序。

1. 腾讯连连小程序进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关操作可以参考 生成 Wi-Fi 设备配网 Token 。
2. 使 Wi-Fi 设备进入 softAP 配网模式，若设备有指示灯在快闪，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，再选择设备 softAP 热点的 SSID/PSW。
4. 手机连接设备 softAP 热点成功后，小程序作为 UDP 客户端会连接 Wi-Fi 设备上面的 UDP 服务（默认 IP 为**192.168.4.1**，端口为**8266**）。
5. 小程序给设备 UDP 服务，发送目标 Wi-Fi 路由器的 SSID/PSW 以及配网 Token，JSON 格式为：

```
{"cmdType":1,"ssid":"Home-WiFi","password":"abcd1234","token":"6aa11111****23****546****11****d"}
```

发送完成后，等待设备 UDP 回复设备信息及配网协议版本号：

```
{"cmdType":2,"productId":"OSPB5ASRWT","deviceName":"dev_01","protoVersion":"2.0"}
```

6. 如果2秒之内，未收到设备回复，则重复步骤5，UDP 客户端重复发送目标 Wi-Fi 路由器的 SSID/PSW 及配网 Token。（如果重复发送5次，都没有收到回复，则认为配网失败，Wi-Fi 设备有异常）
7. 如果步骤5收到设备回复，则说明设备端已收到 Wi-Fi 路由器的 SSID/PSW 及 Token，正在连接 Wi-Fi 路由器，并上报 Token。此时小程序会提示手机也将连接 Wi-Fi 路由器，并通过 Token 轮询物联网后台，来确认配网及设备绑定是否成功。小程序相关操作可以参考 查询配网Token状态 。
8. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将小程序发送的配网 Token，通过下面 MQTT 报文上报给后台服务：

```
    topic: $thing/up/service/ProductID/DeviceName
    payload: {"method":"app_bind_token","clientToken":"client-1234","params":
{"token":"6****345****234ee7****6e528a0fd"}}
```

设备端也可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 Token 上报的结果。

> ⚠ **注意：**
> 如果设备需要通过动态注册来创建设备并获取设备密钥，则会先进行动态注册再连接 MQTT。

9. 在以上步骤5 – 步骤7中，需观察以下情况：
   - 如果小程序收到设备 UDP 服务发送过来的错误日志，且 deviceReply 字段的值为“Current_Error”，则表示当前配网绑定过程中出错，需要退出配网操作。
   - 如果 deviceReply 字段是“Previous_Error”，则为上一次配网的出错日志，只需要上报，不影响当前操作。

   错误日志 JSON 格式，示例如下：

```
{"cmdType":2,"deviceReply":"Current_Error","log":"ESP WIFI connect error! (10, 2)"}
```

10. 如果设备成功上报了 Token，物联网后台服务已确认 Token 有效性，小程序会提示配网完成，设备添加成功。

11. 设备端会记录配网的详细日志，如果配网或者添加设备失败，可以让设备端创建一个特殊的 softAP 和 UDP 服务，通过小程序可以从设备端获取更多日志用于错误分析。

> ⚠ **注意：**
> UDP 相比 TCP 是不可靠的通讯，存在丢包的可能，特别在比较嘈杂的无线 Wi-Fi 环境中，丢包率会比较大。为了保证小程序和设备之间的数据交互是可靠的，需要在应用层设计一些应答以及超时重发的机制。

ESP8266 使用 softAP 配网接口

配网协议在 ESP8266 设备端的参考代码和 AT 固件，请参见 GitHub 工程 qcloud-iot-esp-wifi 。

腾讯云 IoT AT 指令 ESP8266 定制固件

如果 ESP8266 烧写了腾讯云 IoT AT 指令 ESP8266 定制固件，则只要通过指令 AT+TCDEVINFOSET 配置好设备信息，再通过下面的指令启动 softAP 配网即可。

```
AT+TCSAP="ESP8266-SAP","12345678"
```

关于 AT 指令的详细说明，请参见 qcloud-iot-at-esp8266 目录文档。

配网代码示例

在 qcloud-iot-esp8266-demo/main/wifi_config 目录下，提供了 softAP 配网 v2.0 在 ESP8266 上面的参考实现，您可以使用 qcloud-iot-esp8266-demo 工程进行体验。

使用示例

配网接口说明请查看 wifi_config/qcloud_wifi_config.h，您可以按照以下方式使用：

```
/* 在微信小程序中使用WiFi配置和设备绑定 */
int wifi_config_state;
int ret = start_softAP("ESP8266-SAP", "12345678", 0);
if (ret) {
        Log_e("start wifi config failed: %d", ret);
} else {
        /* 最大等待时间: 150 * 2000ms */
        int wait_cnt = 150;
        do {
                Log_d("waiting for wifi config result...");
                HAL_SleepMs(2000);
                wifi_config_state = query_wifi_config_state();
        } while (wifi_config_state == WIFI_CONFIG_GOING_ON && wait_cnt--);
}

wifi_connected = is_wifi_config_successful();
if (!wifi_connected) {
        Log_e("wifi config failed!");
        // 设置softAP向小程序上传log
        start_log_softAP();
}
```

代码设计说明

配网代码将核心逻辑与平台相关底层操作分离，便于移植到不同的硬件设备上。

代码	设计说明
qcloud_wifi_config.c	配网相关接口实现，包括 UDP 服务及 MQTT 连接及 Token 上报，主要依赖腾讯云物联网 C-SDK 及 FreeRTOS/lwIP 运行环境。

`wifi_config_esp.c`	设备硬件 Wi-Fi 操作相关接口实现，依赖于 ESP8266 RTOS，当使用其他硬件平台时，需要进行移植适配。
`wifi_config_error_handle.c`	设备错误日志处理，主要依赖于 FreeRTOS。
`wifi_config_log_handle.c`	设备配网日志收集和上报，主要依赖于 FreeRTOS。

配网代码示例及移植指引

配网协议的设备端代码和 AT 固件，详情请参见 配网代码说明和移植指引 。

SmartConfig 配网开发

最近更新时间：2023-10-27 10:00:51

操作场景

基本原理

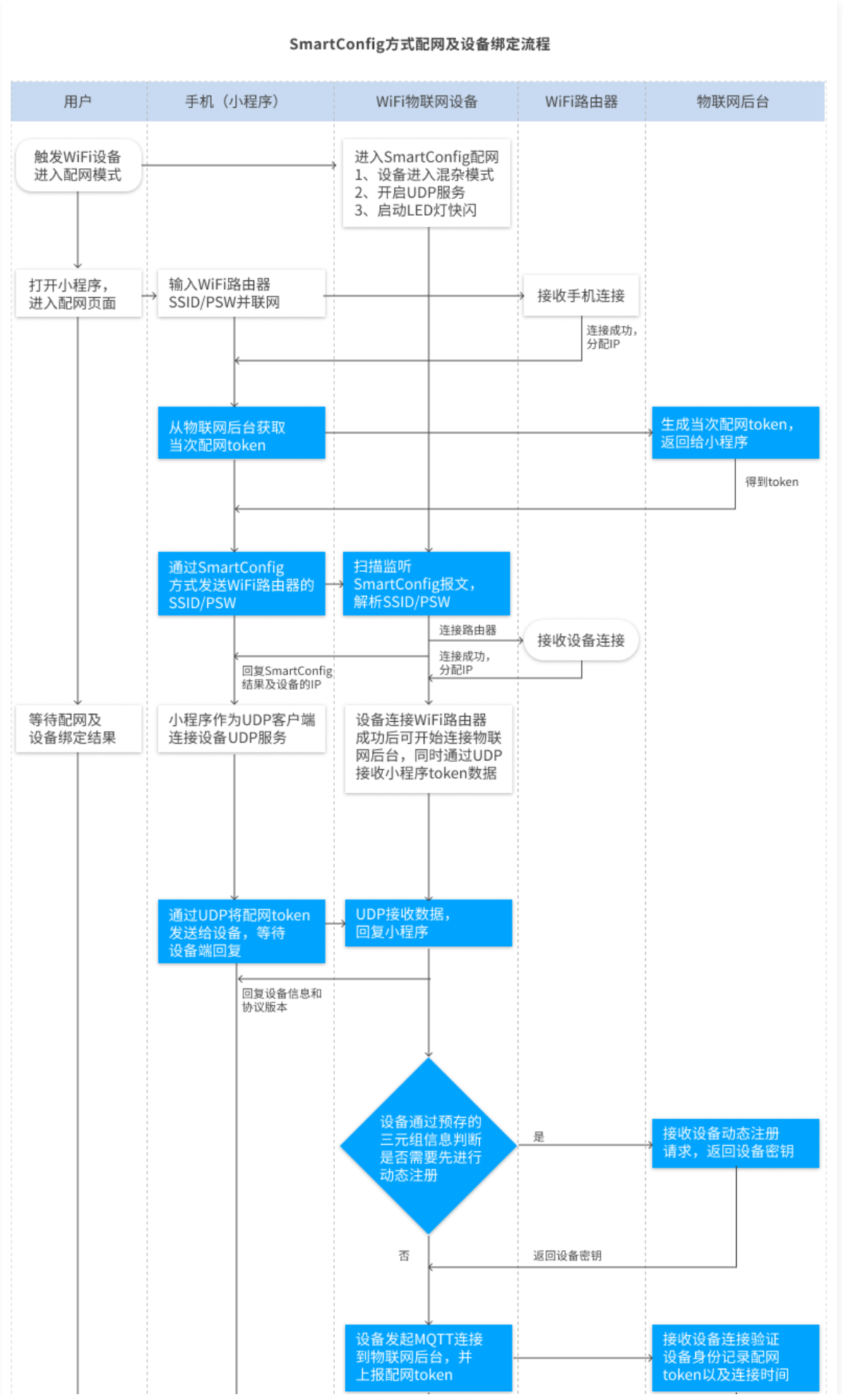
1. 设备进入 Wi-Fi 混杂模式（promiscuous mode）以监听捕获周围的 Wi-Fi 报文。由于设备暂未联网，且 Wi-Fi 网络的数据帧已通过加密，设备无法获取 payload 的内容，但可以获取报文的某些特征数据，例如每个报文的长度。同时对于某些数据帧，例如 UDP 的广播包或多播包，其报文的帧头结构比较固定，较容易识别。
2. 此时在手机 App 或者小程序端，即可通过发送 UDP 的广播包或多播包，并利用报文的特征，例如长度变化进行编码。
3. 将目标 Wi-Fi 路由器的 SSID/PSW 字符以约定的编码方式发送出去，设备端在捕获到 UDP 报文后，按约定的方式进行解码，即可得到目标 Wi-Fi 路由器的相关信息并进行联网。

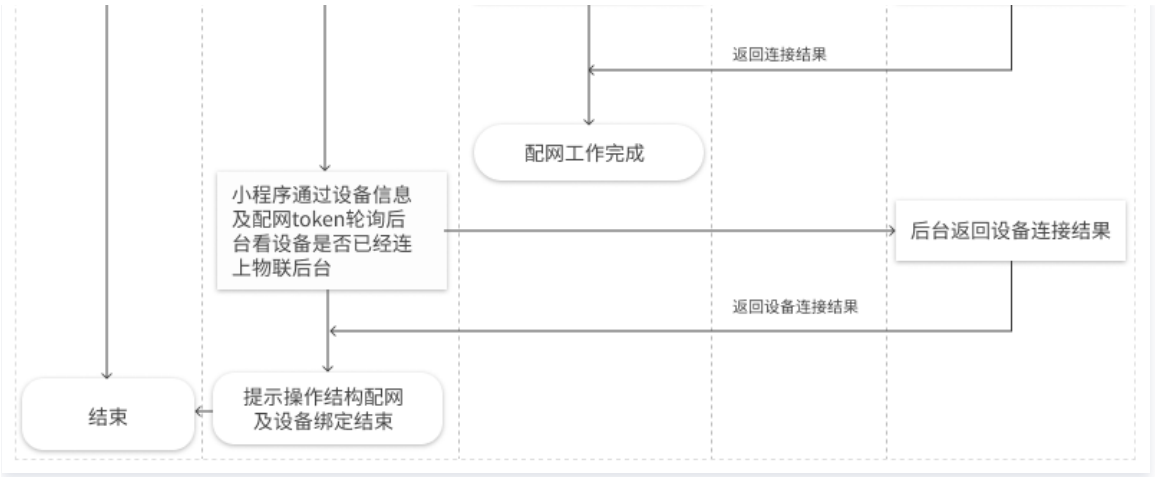
设备绑定流程

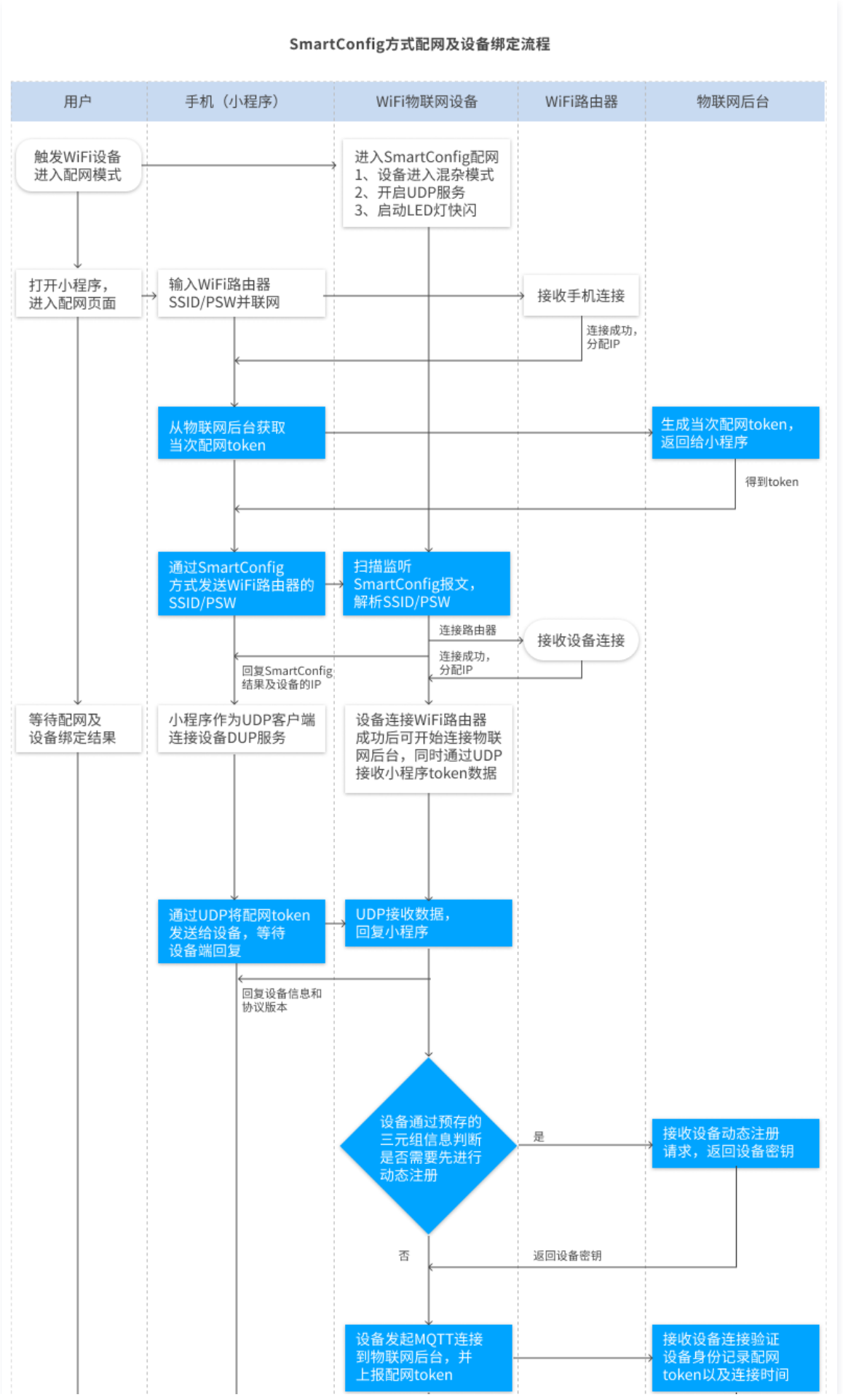
SmartConfig 方式配网，每个厂商的编码方式和报文选择上有自己的协议，对于 ESP8266，采用的协议是乐鑫 ESP-TOUCH协议 。

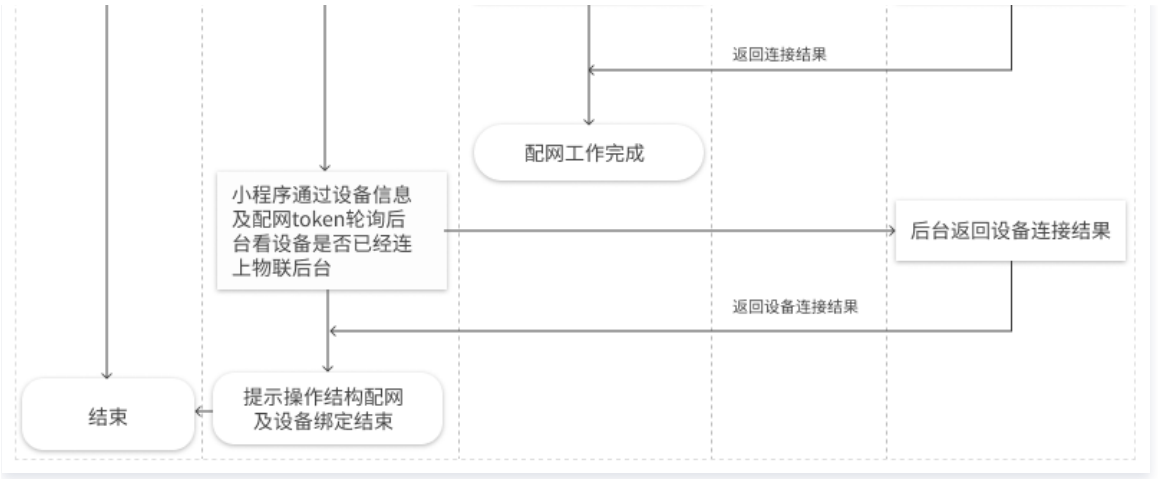
- 基于该协议，设备端在连接 Wi-Fi 路由器成功后，将会告知手机端自己的 IP 地址。
- 此时手机端可以通过数据通道，例如 TCP/UDP 通讯将后台提供的配网 Token 发送给设备，并由设备转发至物联网后台，依据 Token 进行设备绑定。

目前腾讯连连小程序已支持采用 ESP-TOUCH 协议进行 SmartConfig 配网，并提供相应的 小程序 SDK 。
SmartConfig 方式配网及设备绑定的示例流程图如下：


SmartConfig方式配网及设备绑定流程
用户
手机（小程序）
WiFi物联网设备
WiFi路由器
物联网后台
触发WiFi设备
进入配网模式
进入SmartConfig配网
1、设备进入混杂模式
2、开启UDP服务
3、启动LED灯快闪
打开小程序，
进入配网页面
输入WiFi路由器
SSID/PSW并联网
接收手机连接
连接成功，
分配IP
从物联网后台获取
当次配网token
生成当次配网token，
返回给小程序
得到token
通过SmartConfig
方式发送WiFi路由器的
SSID/PSW
扫描监听
SmartConfig报文，
解析SSID/PSW
连接路由器
接收设备连接
连接成功，
分配IP
回复SmartConfig
结果及设备的IP
等待配网及
设备绑定结果
小程序作为UDP客户端
连接设备UDP服务
设备连接WiFi路由器
成功后可开始连接物联
网后台，同时通过UDP
接收小程序token数据
通过UDP将配网token
发送给设备，等待
设备端回复
UDP接收数据，
回复小程序
回复设备信息和
协议版本
设备通过预存的
三元组信息判断
是否需要先进行
动态注册
是
接收设备动态注册
请求，返回设备密钥
否
返回设备密钥
设备发起MQTT连接
到物联网后台，并
上报配网token
接收设备连接验证
设备身份记录配网
token以及连接时间


返回连接结果
配网工作完成
小程序通过设备信息及配网token轮询后台看设备是否已经连上物联后台
后台返回设备连接结果
返回设备连接结果
结束
提示操作结构配网及设备绑定结束


SmartConfig方式配网及设备绑定流程
用户
手机（小程序）
WiFi物联网设备
WiFi路由器
物联网后台
触发WiFi设备进入配网模式
进入SmartConfig配网
1、设备进入混杂模式
2、开启UDP服务
3、启动LED灯快闪
打开小程序，进入配网页面
输入WiFi路由器SSID/PSW并联网
接收手机连接
连接成功，分配IP
从物联网后台获取当次配网token
生成当次配网token，返回给小程序
得到token
通过SmartConfig方式发送WiFi路由器的SSID/PSW
扫描监听SmartConfig报文，解析SSID/PSW
连接路由器
接收设备连接
连接成功，分配IP
回复SmartConfig结果及设备的IP
等待配网及设备绑定结果
小程序作为UDP客户端连接设备DUP服务
设备连接WiFi路由器成功后可开始连接物联网后台，同时通过UDP接收小程序token数据
通过UDP将配网token发送给设备，等待设备端回复
UDP接收数据，回复小程序
回复设备信息和协议版本
设备通过预存的三元组信息判断是否需要先进行动态注册
是
接收设备动态注册请求，返回设备密钥
否
返回设备密钥
设备发起MQTT连接到物联网后台，并上报配网token
接收设备连接验证设备身份记录配网token以及连接时间



操作步骤

SmartConfig 配网协议示例

SmartConfig 配网设备端与腾讯连连小程序及后台交互的数据协议操作如下：

1. 腾讯连连小程序进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关操作可以参考 生成 Wi-Fi 设备配网 Token 。
2. 使 Wi-Fi 设备进入 SmartConfig 配网模式，若设备有指示灯在快闪，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，按下一步后，将通过 SmartConfig 方式发送报文。
4. 设备端通过监听捕获 SmartConfig 报文，解析出目标路由器的 SSID/PSW 并进行联网，联网成功后，设备会告知小程序自己的 IP 地址，同时开始连接物联网后台。
5. 小程序作为 UDP 客户端会连接 Wi-Fi 设备上面的 UDP 服务（默认端口为8266）。给设备发送配网 Token，JSON 格式为：

```
{"cmdType":0,"token":"6xx82618a9d529a2ee777****528a0fd"}
```

发送完成后，等待设备 UDP 回复设备信息及配网协议版本号：

```
{"cmdType":2,"productId":"OSPB5ASRWT","deviceName":"dev_01","protoVersion":"2.0"}
```

6. 如果2秒之内未收到设备回复，则重复步骤5，UDP 客户端重复发送配网 Token。（如果重复发送5次都没有收到回复，则认为配网失败，Wi-Fi 设备有异常）
7. 如果步骤5收到设备回复，则说明设备端已经收到 Token，并准备上报 Token。此时小程序会开始通过 Token 轮询物联网后台来确认配网及设备绑定是否成功。小程序相关操作可以参考 查询配网 Token 状态 。
8. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将小程序发送来的配网 Token 通过下面 MQTT 报文上报给后台服务：

```
    topic: $thing/up/service/ProductID/DeviceName
    payload: {"method":"app_bind_token","clientToken":"client-1234","params":
{"token":"6xx82618a9d529a2ee777****528a0fd"}}
```

设备端也可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 Token 上报的结果。

> ⚠ **注意：**
> 注意如果设备需要通过动态注册来创建设备并获取设备密钥，则会先进行动态注册再连接 MQTT。

9. 在以上步骤5 – 步骤7中，需观察以下情况：
   - 如果小程序收到设备 UDP 服务发送过来的错误日志，且 deviceReply 字段的值为"Current_Error"，则表示当前配网绑定过程中出错，需要退出配网操作。
   - 如果 deviceReply 字段是"Previous_Error"，则为上一次配网的出错日志，只需要上报，不影响当前操作。

错误日志 JSON 格式例子：

```
{"cmdType":2,"deviceReply":"Current_Error","log":"ESP WIFI connect error! (10, 2)"}
```

10. 如果设备成功上报了 Token，物联网后台服务确认了 Token 有效性，小程序会提示配网完成，设备添加成功。
11. 设备端会记录配网的详细日志，如果配网或者添加设备失败，可以让设备端创建一个特殊的 softAP 和 UDP 服务，通过小程序可以从设备端获取更多日志用于错误分析。

ESP8266 使用 SmartConfig 配网接口

配网协议在 ESP8266 设备端的参考代码和 AT 固件，请参见 GitHub 工程 qcloud-iot-esp-wifi 。

腾讯云 IoT AT 指令 ESP8266 定制固件

如果 ESP8266 烧写了腾讯云 IoT AT 指令 ESP8266 定制固件，则只要通过指令 AT+TCDEVINFOSET 配置好设备信息，再通过下面的指令启动 SmartConfig 配网即可。

```
AT+TCSTARTSMART
```

关于 AT 指令的详细说明，请参考 qcloud-iot-at-esp8266 目录文档。

配网代码示例

在 qcloud-iot-esp8266-demo/main/wifi_config 目录下，提供了 SmartConfig 配网在 ESP8266 上面的参考实现，用户可以使用 qcloud-iot-esp8266-demo 工程进行体验。

使用示例

配网接口说明请查看 wifi_config/qcloud_wifi_config.h，可以按照下面方式使用：

```
/* 在微信小程序中使用WiFi配置和设备绑定 */
int wifi_config_state;
int ret = start_smartconfig();
if (ret) {
        Log_e("start wifi config failed: %d", ret);
} else {
        /* 最大等待时间: 150 * 2000ms */
        int wait_cnt = 150;
        do {
                Log_d("waiting for wifi config result...");
                HAL_SleepMs(2000);
                wifi_config_state = query_wifi_config_state();
        } while (wifi_config_state == WIFI_CONFIG_GOING_ON && wait_cnt--);
}

wifi_connected = is_wifi_config_successful();
if (!wifi_connected) {
        Log_e("wifi config failed!");
        // 设置softAP向小程序上传log
        start_log_softAP();
}
```

代码设计说明

配网代码将核心逻辑与平台相关底层操作分离，便于移植到不同的硬件设备上。

代码	设计说明
`qcloud_wifi_config.c`	配网相关接口实现，包括 UDP 服务及 MQTT 连接及 Token 上报，主要依赖腾讯云物联网 C-SDK 及 FreeRTOS/lwIP 运行环境。

`wifi_config_esp.c`	设备硬件 Wi-Fi 操作相关接口实现，依赖于 ESP8266 RTOS，当使用其他硬件平台时，需要进行移植适配。
`wifi_config_error_handle.c`	设备错误日志处理，主要依赖于 FreeRTOS。
`wifi_config_log_handle.c`	设备运行上报日志处理，主要依赖于 FreeRTOS。

注意：
如果将 SmartConfig 移植到不同的芯片平台，需要确保平台支持 ESP-TOUCH 配网协议。同时由于小程序框架限制，小程序通过 UDP 广播/多播发送 ESP-TOUCH 协议报文时，会往报文 body 填入一个固定的 IP 地址，设备端在回复结果时不应该依赖于该地址，而应当以 UDP 报文 header 的源 IP 地址为准。

配网代码示例及移植指引

配网协议的设备端代码和 AT 固件，详情请参见 配网代码说明和移植指引 。

AirKiss配网开发

最近更新时间：2023-06-05 12:11:41

操作场景

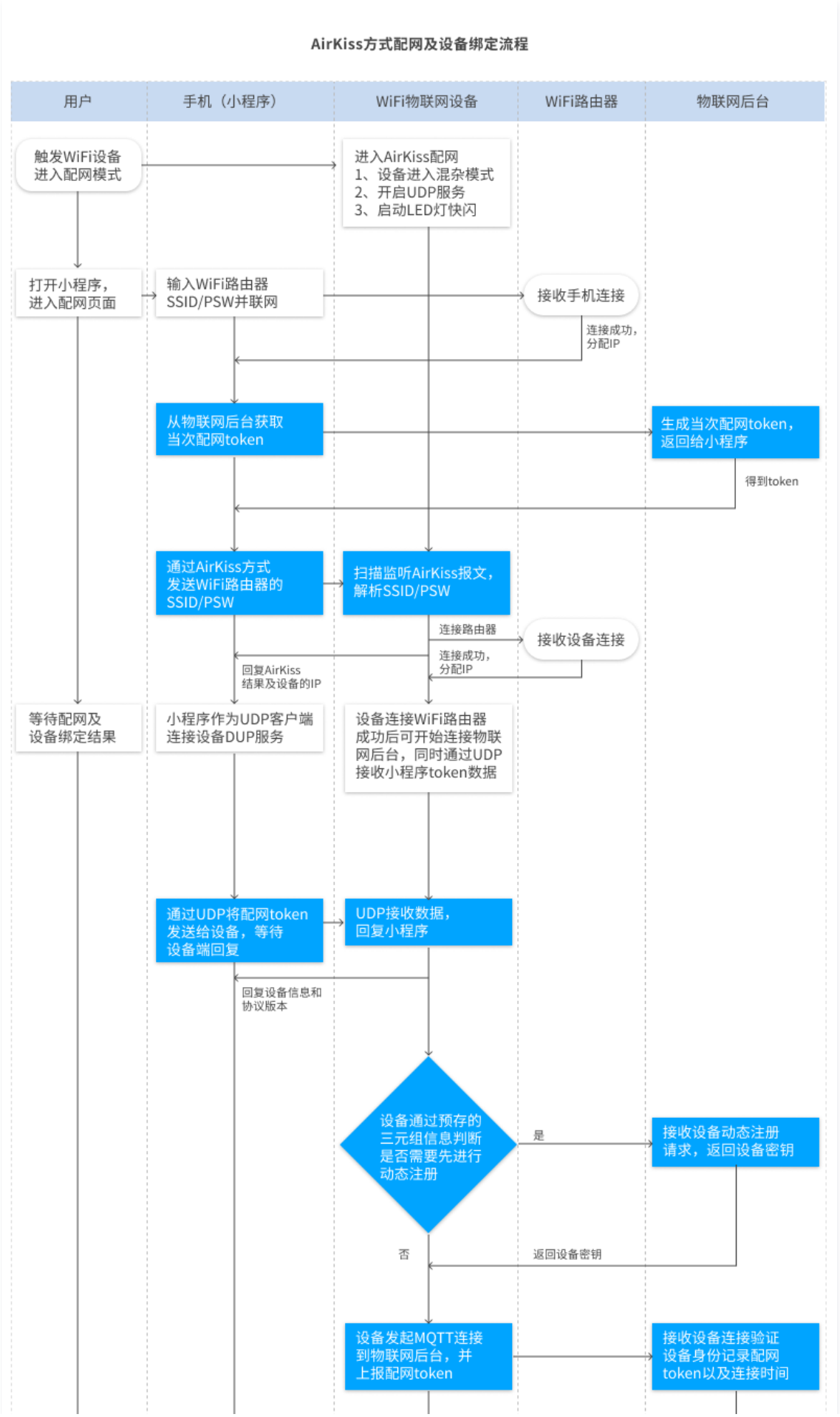
AirKiss 是微信为 Wi-Fi 设备提供的配网技术，详情请参见 AirKiss 概述及应用场景
其配网原理跟 SmartConfig 一样，如下所述：

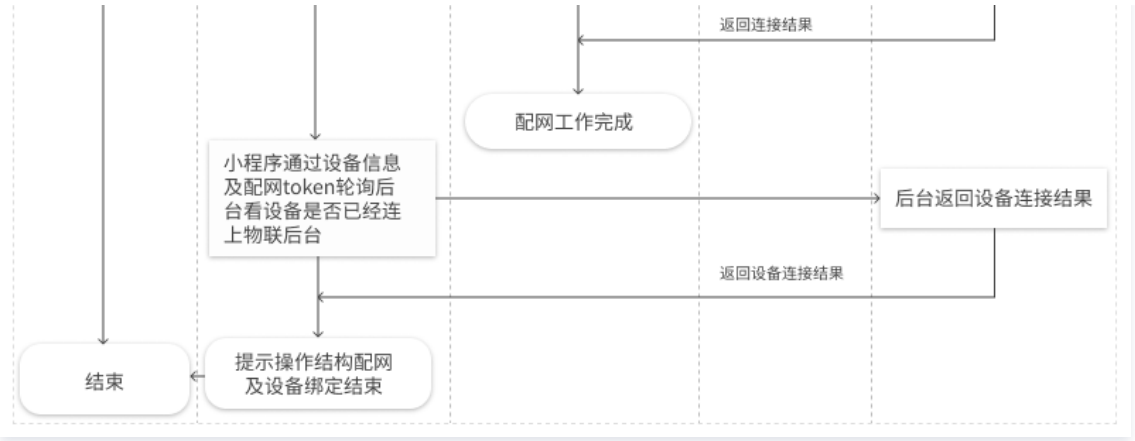
基本原理

1. 设备进入 Wi-Fi 混杂模式（promiscuous mode）以监听捕获周围的 Wi-Fi 报文。由于设备暂未联网，且 Wi-Fi 网络的数据帧已通过加密，设备无法获取 payload 的内容，但可以获取报文的某些特征数据，例如每个报文的长度，同时对于某些数据帧；例如 UDP 的广播包或多播包，其报文的帧头结构比较固定，较容易识别。
2. 此时在手机 App 或者小程序侧，即可通过发送 UDP 的广播包或多播包，并利用报文的特征，例如长度变化进行编码。
3. 将目标 Wi-Fi 路由器的 SSID/PSW 字符以约定的编码方式发送出去，设备端在捕获到 UDP 报文后，按约定的方式进行解码，即可得到目标 Wi-Fi 路由器的相关信息并进行联网。

设备绑定流程

基于 AirKiss 协议，设备端在连接 Wi-Fi 路由器成功后，会往手机端回复 UDP 报文，手机端获取设备端的 IP 地址之后，同样可以通过 UDP，将后台提供的配网 Token 发送给设备，并由设备转发至物联网后台，依据 Token 可以进行设备绑定。
腾讯连连小程序已支持采用 AirKiss 协议进行配网，并提供相应的 小程序 SDK 。
AirKiss 方式配网及设备绑定的示例流程图如下：


AirKiss方式配网及设备绑定流程
用户
手机（小程序）
WiFi物联网设备
WiFi路由器
物联网后台
触发WiFi设备
进入配网模式
进入AirKiss配网
1、设备进入混杂模式
2、开启UDP服务
3、启动LED灯快闪
打开小程序，
进入配网页面
输入WiFi路由器
SSID/PSW并联网
接收手机连接
连接成功，
分配IP
从物联网后台获取
当次配网token
生成当次配网token，
返回给小程序
得到token
通过AirKiss方式
发送WiFi路由器的
SSID/PSW
扫描监听AirKiss报文，
解析SSID/PSW
连接路由器
接收设备连接
连接成功，
分配IP
回复AirKiss
结果及设备的IP
等待配网及
设备绑定结果
小程序作为UDP客户端
连接设备DUP服务
设备连接WiFi路由器
成功后可开始连接物联
网后台，同时通过UDP
接收小程序token数据
通过UDP将配网token
发送给设备，等待
设备端回复
UDP接收数据，
回复小程序
回复设备信息和
协议版本
设备通过预存的
三元组信息判断
是否需要先进行
动态注册
是
接收设备动态注册
请求，返回设备密钥
否
返回设备密钥
设备发起MQTT连接
到物联网后台，并
上报配网token
接收设备连接验证
设备身份记录配网
token以及连接时间



操作步骤

AirKiss 配网协议示例

AirKiss 配网设备端与腾讯连连小程序及后台交互的数据协议操作如下：

1. 腾讯连连小程序进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关详情操作请参见 生成 Wi-Fi 设备配网 Token 。
2. 使 Wi-Fi 设备进入 AirKiss 配网模式，若设备有指示灯在快闪，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，按下一步后，将通过 AirKiss 方式发送报文。
4. 设备端通过监听捕获 AirKiss 报文，解析出目标路由器的 SSID/PSW 并进行联网，联网成功后，设备会告知小程序自己的 IP 地址，同时开始连接物联网后台。
5. 小程序作为 UDP 客户端会连接 Wi-Fi 设备上面的 UDP 服务（默认端口为**8266**）。给设备发送配网 Token，JSON 格式为：

```
{"cmdType":0,"token":"6aa12345a9d529a2****7aa6e528a0fd"}
```

发送完成后，等待设备 UDP 回复设备信息及配网协议版本号：

```
{"cmdType":2,"productId":"AAAA5AAAAA","deviceName":"dev_01","protoVersion":"2.0"}
```

6. 如果2秒之内没有收到设备回复，则重复步骤5，UDP 客户端重复发送配网 Token。（如果重复发送5次都没有收到回复，则认为配网失败，Wi-Fi 设备有异常。）
7. 如果步骤5收到设备回复，则说明设备端已经收到 Token，并准备上报 Token。此时小程序会开始通过 Token 轮询物联网后台来确认配网及设备绑定是否成功。小程序相关操作可以参考 查询配网 Token 状态 。
8. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将小程序发送来的配网 Token 通过下面 MQTT 报文上报给后台服务：

```
    topic :  $thing/up/service/ProductID/DeviceName
    payload :  { "method" : "app_bind_token" , "clientToken" : "client-1234" , "params" :
{ "token" : "6aa12345a9d529****777aa6e528a0fd" } }
```

设备端可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 token 上报的结果。

> ⚠ **注意：**
> 如果设备需要通过动态注册来创建设备并获取设备密钥，则会先进行动态注册再连接 MQTT。

9. 在以上步骤5 – 步骤7中，需观察以下情况：
   - 如果小程序收到设备 UDP 服务发送过来的错误日志，且 deviceReply 字段的值为"Current_Error"，则表示当前配网绑定过程中出错，需要退出配网操作。
   - 如果 deviceReply 字段是"Previous_Error"，则为上一次配网的出错日志，只需要上报，不影响当前操作。

错误日志 JSON 格式例子：

```
{"cmdType":2,"deviceReply":"Current_Error","log":"ESP WIFI connect error! (10, 2)"}
```

10. 如果设备成功上报了 Token，物联网后台服务确认了 Token 有效性，小程序会提示配网完成，设备添加成功。
11. 设备端会记录配网的详细日志，如果配网或者添加设备失败，可以让设备端创建一个特殊的softAP和UDP服务，通过小程序可以从设备端获取更多日志用于错误分析。

ESP8266 使用 AirKiss 配网接口

配网协议在 ESP8266 设备端的参考代码和 AT 固件，请参见 GitHub 工程 qcloud-iot-esp-wifi 。
在 ESP8266 上面，AirKiss协议属于 SmartConfig 配网方式的一种，只需要按照下面启动 SmartConfig 模式就可以同时支持ESP-TOUCH 和 AirKiss。

```
esp_smartconfig_set_type(SC_TYPE_ESPTOUCH_AIRKISS);
```

配网代码示例

在 qcloud-iot-esp8266-demo/main/wifi_config 目录下，提供了 AirKiss 配网在 ESP8266 上面的参考实现，用户可以使用 qcloud-iot-esp8266-demo 工程进行体验。
配网接口说明请查看 wifi_config/qcloud_wifi_config.h，可以按照下面方式使用：

```
/* 在微信小程序中使用WiFi配置和设备绑定 */
int wifi_config_state;
int ret = start_smartconfig();
if (ret) {
        Log_e("start wifi config failed: %d", ret);
} else {
        /* 最大等待时间: 150 * 2000ms */
        int wait_cnt = 150;
        do {
                Log_d("waiting for wifi config result...");
                HAL_SleepMs(2000);
                wifi_config_state = query_wifi_config_state();
        } while (wifi_config_state == WIFI_CONFIG_GOING_ON && wait_cnt--);
}

wifi_connected = is_wifi_config_successful();
if (!wifi_connected) {
        Log_e("wifi config failed!");
        //设置softAP向小程序上传log
        start_log_softAP();
}
```

代码设计说明

配网代码将核心逻辑与平台相关底层操作分离，便于移植到不同的硬件设备上。

代码	设计说明
qcloud_wifi_config.c	配网相关接口实现，包括 UDP 服务及 MQTT 连接及 Token 上报，主要依赖腾讯云物联网 C SDK 及 FreeRTOS/lwIP运行环境。
wifi_config_esp.c	设备硬件 Wi-Fi 操作相关接口实现，依赖于 ESP8266 RTOS，当使用其他硬件平台时，需要进行移植适配。
wifi_config_error_handle.c	设备错误日志处理，主要依赖于 FreeRTOS。
wifi_config_log_handle.c	设备运行上报日志处理，主要依赖于 FreeRTOS。

simpleConfig 配网开发

最近更新时间：2024-08-26 14:55:51

操作场景

基本原理

1. 设备进入 Wi-Fi 混杂模式（promiscuous mode）以监听捕获周围的 Wi-Fi 报文。由于设备暂未联网，且 Wi-Fi 网络的数据帧已通过加密，设备无法获取payload 的内容，但可以获取报文的某些特征数据，例如，每个报文的长度，同时对于某些数据帧，例如，UDP 的广播包或多播包，其报文的帧头结构比较固定，较容易识别。
2. 此时在手机 App 或者小程序侧，即可通过发送 UDP 的广播包或多播包，并利用报文的特征，例如，长度变化进行编码。
3. 将目标 Wi-Fi 路由器的 SSID/PSW 字符以约定的编码方式发送出去，设备端在捕获到 UDP 报文后，按约定的方式进行解码，即可得到目标 Wi-Fi 路由器的相关信息并进行联网。

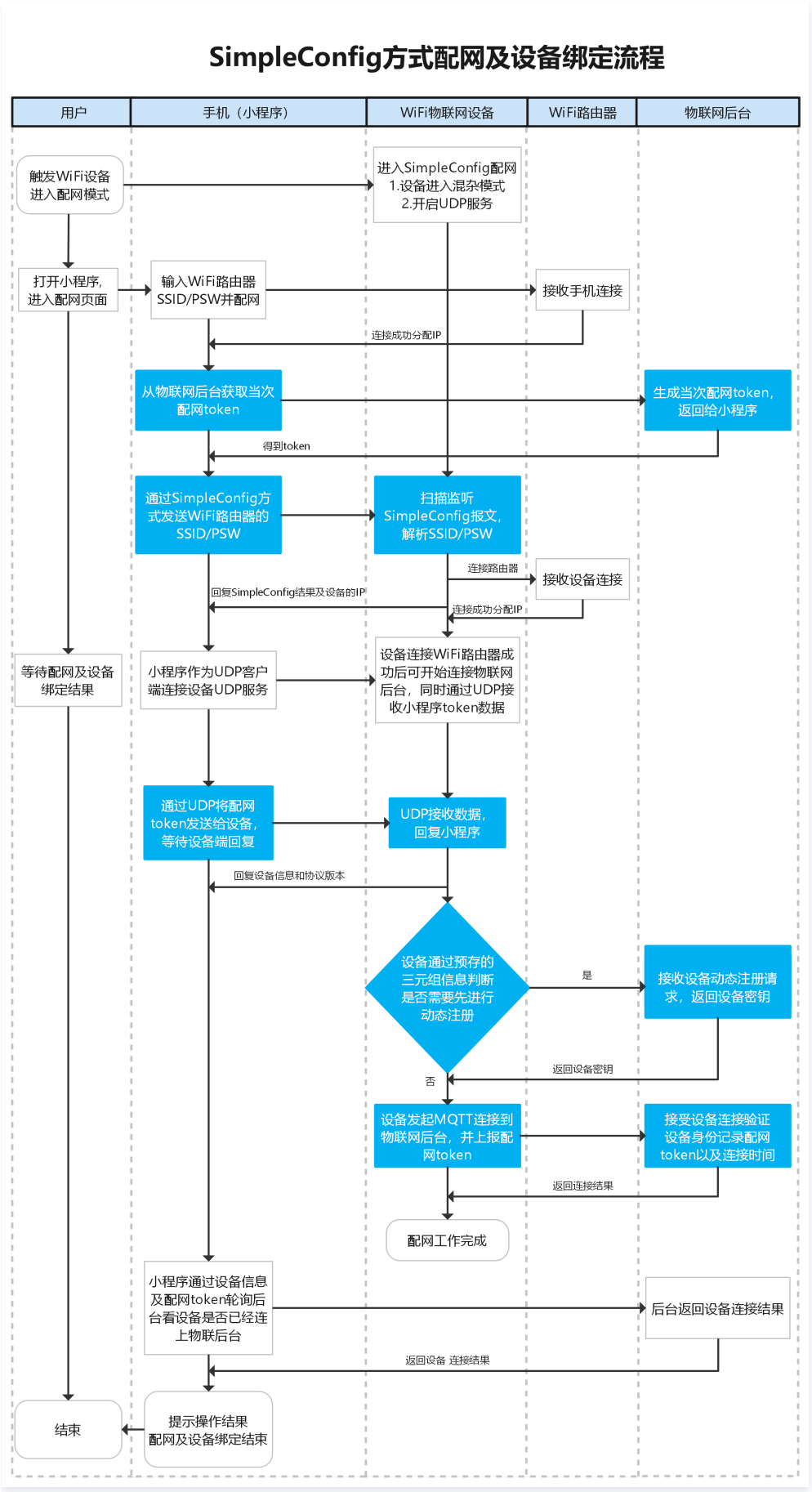
设备绑定流程

一键配网方式配网，每个厂商编码方式和报文选择上有自己的协议，对于 RTK8720CF，采用的协议是 Realtek simpleConfig 协议，参考文档 Realtek 提供的文档 `AN0011 Realtek wlan simple configuration.pdf` 。

- 基于该协议，设备端在连接 Wi-Fi 路由器成功后，会告知手机端自己的 IP 地址。
- 此时手机端可以通过数据通道，例如，TCP/UDP 通讯将后台提供的配网 Token 发送给设备，并由设备转发至物联网后台，依据 Token 可以进行设备绑定。

目前腾讯连连小程序已支持 simpleConfig 配网，并提供相应的 小程序 SDK 。

simpleConfig 方式配网及设备绑定的示例流程图如下：



操作步骤

simpleConfig 配网步骤

simpleConfig 配网设备端与腾讯连连小程序及后台交互的数据协议操作如下：

1. 腾讯连连小程序进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关操作可以参考 生成 Wi-Fi 设备配网 Token 。
2. 使 Wi-Fi 设备进入 SmartConfig 配网模式，若设备有指示灯在快闪，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，按下一步后，将通过 SmartConfig 方式发送报文。
4. 设备端通过监听捕获 SmartConfig 报文，解析出目标路由器的 SSID/PSW 并进行联网，联网成功后，设备会告知小程序自己的 IP 地址，同时开始连接物联网后台。
5. 小程序作为 UDP 客户端会连接 Wi-Fi 设备上面的 UDP 服务（默认端口为**8266**）。给设备发送配网 Token，JSON 格式为：

```
{"cmdType":0,"token":"6xx82618a9d529a2ee777****528a0fd"}
```

发送完成后，等待设备 UDP 回复设备信息及配网协议版本号：

```
{"cmdType":2,"productId":"OSPB5ASRWT","deviceName":"dev_01","protoVersion":"2.0"}
```

6. 如果2秒之内未收到设备回复，则重复步骤5，UDP 客户端重复发送配网 Token。（如果重复发送5次都没有收到回复，则认为配网失败，Wi-Fi 设备有异常）
7. 如果步骤5收到设备回复，则说明设备端已经收到 Token，并准备上报 Token。此时小程序会开始通过 Token 轮询物联网后台来确认配网及设备绑定是否成功。小程序相关操作可以参考 查询配网 Token 状态 。
8. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将小程序发送来的配网 Token 通过下面 MQTT 报文上报给后台服务：

```
   topic: $thing/up/service/ProductID/DeviceName
   payload: {"method":"app_bind_token","clientToken":"client-1234","params":
{"token":"6xx82618a9d529a2ee777****528a0fd"}}
```

设备端也可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 Token 上报的结果。

9. 在以上步骤5 – 步骤7中，需观察以下情况：
   - 如果小程序收到设备 UDP 服务发送过来的错误日志，且 deviceReply 字段的值为"Current_Error"，则表示当前配网绑定过程中出错，需要退出配网操作。
   - 如果 deviceReply 字段是"Previous_Error"，则为上一次配网的出错日志，只需要上报，不影响当前操作。
     错误日志 JSON 格式例子：

```
{"cmdType":2,"deviceReply":"Current_Error","log":"ESP WIFI connect error! (10, 2)"}
```

10. 如果设备成功上报了 Token，物联网后台服务确认了 Token 有效性，小程序会提示配网完成，设备添加成功。
11. 设备端会记录配网的详细日志，如果配网或者添加设备失败，可以让设备端创建一个特殊的 softAP 和 UDP 服务，通过小程序可以从设备端获取更多日志用于错误分析。

基于Ambz2 SDK使用 simpleConfig 与小程序配网

simpleConfig配网协议配合腾讯连连基于Ambz2 SDK的实现参见 GitHub 工程 qcloud-iot-rtk-wifi-based-ambz2 。

配网代码示例

在 qcloud-iot-rtk-wifi-based-ambz2\component\common\example\qcloud_iot_c_sdk\wifi_config 目录下，提供 simpleConfig 配网在 Ambz2 SDK 上面的参考实现，配网接口说明请查看 wifi_config/qcloud_wifi_config.h。
配网框架对simpleConfig配网做了封装，开发者只要调用API启动simpleConfig配网，然后在配网结果回调中判断配网结果即可。`qcloud_demo_task` 中介绍 simpleConfig 配网的使用：

```
static void qcloud_demo_task(void *arg)
{
    int ret;
```

```
    set_wifi_config_result(false);
    while (!wifi_is_up(RTW_STA_INTERFACE)) {
        HAL_SleepMs(1000);
    }
#if WIFI_PROV_SOFT_AP_ENABLE
    Log_d("start softAP wifi provision");
    eSoftApConfigParams apConf = {"RTK8720-SAP", "12345678", 6};
    ret = qiot_wifi_config_start(WIFI_CONFIG_TYPE_SOFT_AP, &apConf, _wifi_config_result_cb);
#elif WIFI_PROV_SIMPLE_CONFIG_ENABLE
    Log_d("start simple config wifi provision");
    ret = qiot_wifi_config_start(WIFI_CONFIG_TYPE_SIMPLE_CONFIG, NULL, _wifi_config_result_cb);
#else
    Log_e("not supported wifi provision method");
    ret = -1;
#endif
    if (ret) {
        Log_e("start wifi config failed: %d", ret);
        goto exit;
    }
}
```

使能宏定义 `WIFI_PROV_SIMPLE_CONFIG_ENABLE` ， 调用配网接口 `qiot_wifi_config_start` ，传入simpleConfig配网模式、配网结果回调，则配网结果回调函数中会返回配网的结果。

> **注意：**
> demo 需要关闭 softAP 的宏定义 `WIFI_PROV_SOFT_AP_ENABLE` 才会跑 simpleConfig 的配网方式。

代码设计说明

配网代码将核心逻辑与平台相关底层操作分离，便于移植到不同的硬件设备上。

代码	设计说明
`qcloud_wifi_config.c`	配网框架，统一各种配网方式，实现配网启动、配网停止、配网结果回调，用户只需要将特定配网方法注册到 `sg_wifi_config_methods` 即可，不依赖任何软硬件平台。
`qiot_comm_service.c`	配网相关接口实现，实现 UDP 服务和与小程序的数据交互，主要依赖腾讯云物联网 C-SDK 及 FreeRTOS/lwIP 运行环境。
`qiot_device_bind.c`	配网相关接口实现，实现配网过程的token交互与设备绑定，主要依赖腾讯云物联网 C-SDK 及 FreeRTOS/lwIP 运行环境
`rtk_soft_ap.c`	softAP Wi-Fi 操作相关接口实现，依赖于 Ambz2 SDK提供的WiFi相关操作接口，当使用其他硬件平台时，可以参考移植适配。
`rtk_simple_config.c`	simpleConfig Wi-Fi 操作相关接口实现，依赖于 Ambz2 SDK提供的WiFi相关操作接口，当使用其他硬件平台时，可以参考移植适配。
`wifi_config_error_handle.c`	设备错误日志处理，主要依赖于 FreeRTOS。
`wifi_config_log_handle.c`	设备配网日志收集和上报，主要依赖于 FreeRTOS。

LLSync 辅助配网开发

最近更新时间：2024-12-31 18:00:53

LLSync 蓝牙辅助配网功能是腾讯云IoT推出的针对 `Wi-Fi + BLE` 的官方 `Combo` 芯片方案，通过 `BLE` 创建指定的 `GATT` 服务，手机连接该 `GATT SERVER`，利用 `BLE` 的无线通信能力，将物联网设备连接所需的 `SSID`、`PASSWORD` 等信息传输给 `Wi-Fi + BLE` 的 `Combo` 芯片或模组，使设备顺利接入物联网平台，继而完设备绑定等功能。
蓝牙辅助配网移植的主要流程如下：

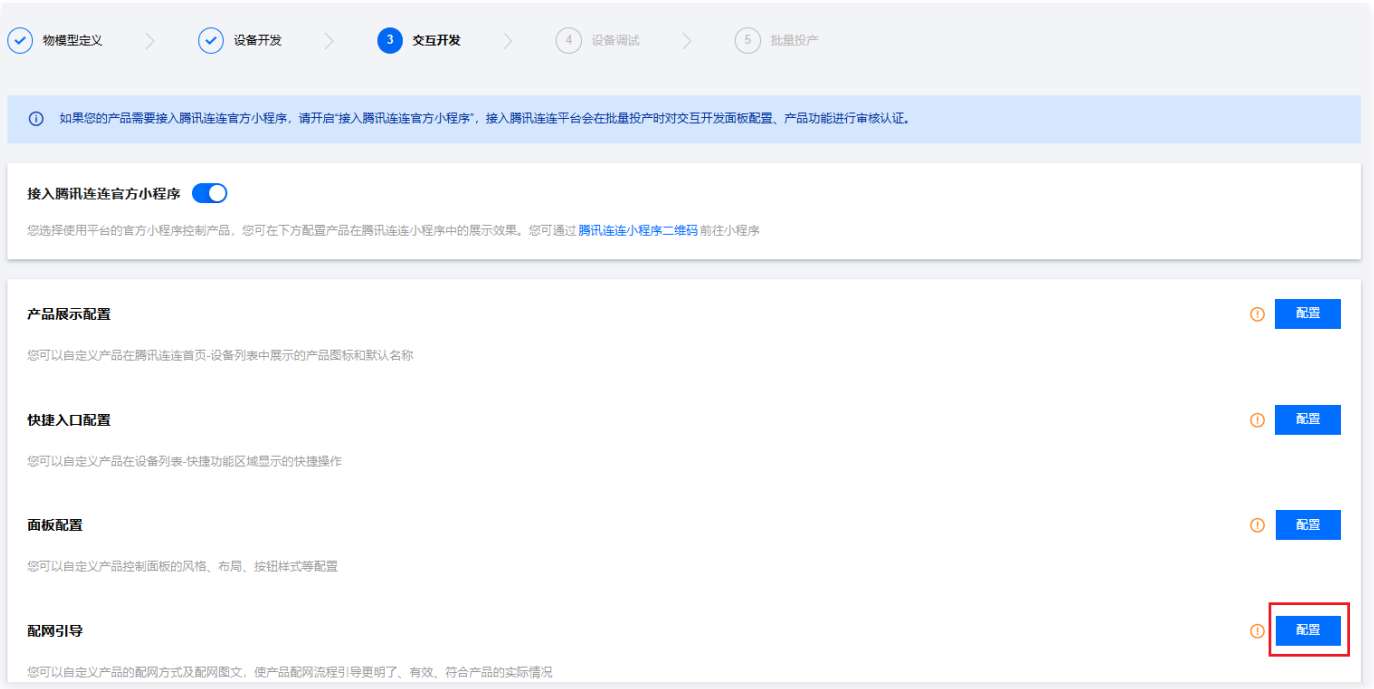
1. 硬件设备选择。
2. 控制台创建产品。
3. 获取 SDK。
4. 移植 SDK。
5. 验证蓝牙辅助配网功能。

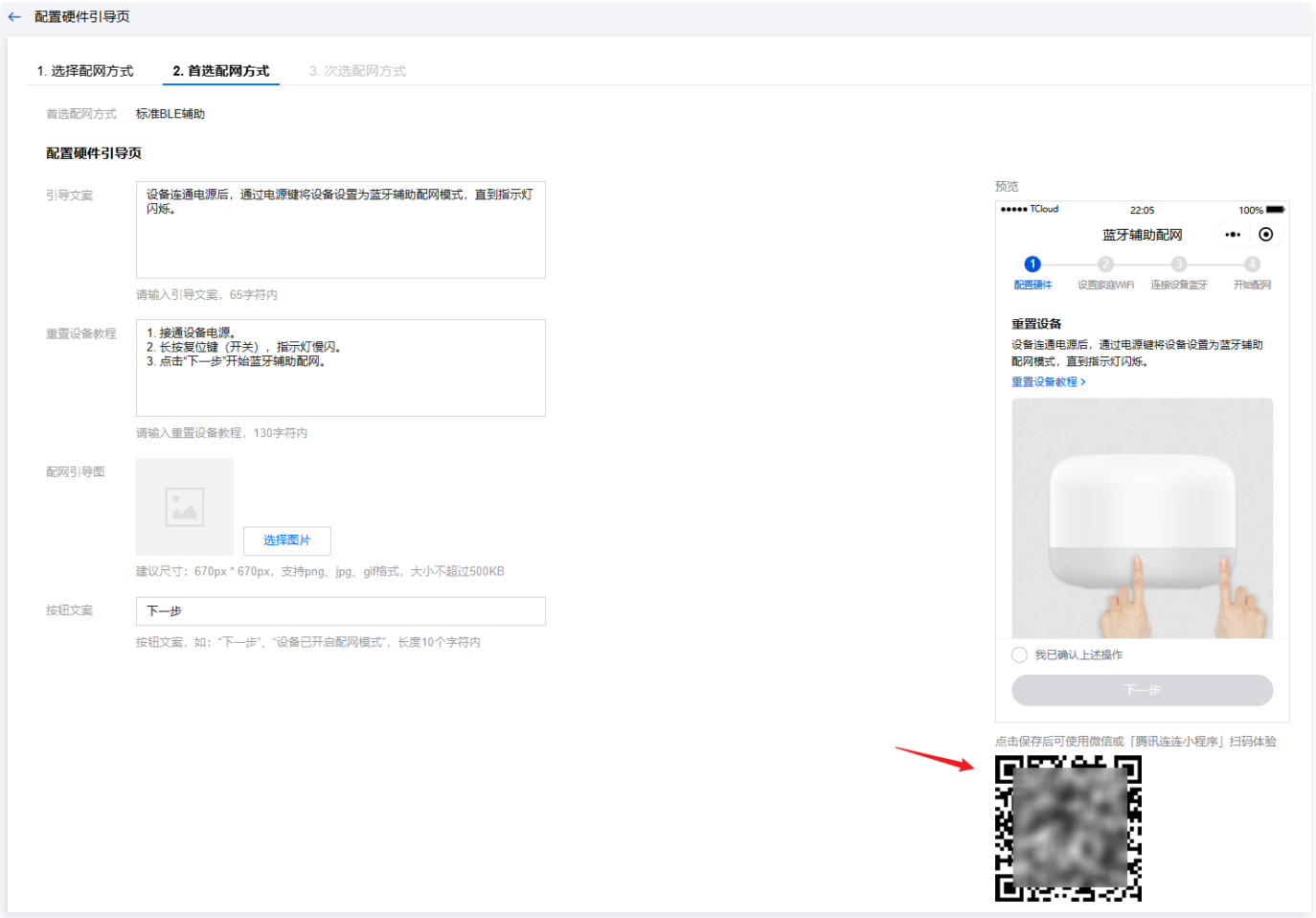
一、 硬件设备选择

需要您根据产品特性选择合适的 `Combo` 芯片或模组。

二、 控制台创建产品

1. 请参考 控制台操作 创建设备。
2. 在控制台选择设备配网方式，配网方式确定后可以通过扫描二维码的方式连接设备蓝牙进行配网。





三、获取 SDK

蓝牙辅助配网功能使用了 C SDK 和 LLSync SDK，请下载最新版本使用。您可以下载 `ESP32` 使用 LLSync 配网 示例程序 参考。

四、移植 SDK

蓝牙辅助配网功能使用了 `WI-Fi` 和 `BLE` 的通信能力，因此包括 `C SDK` 和 `LLSync SDK` 的移植。

1. `LLSync SDK` 移植主要是做配置编译、HAL 实现、API 调用。

1.1 配置编译

修改 `config/ble_qiot_config.h` 文件，对配网相关功能进行配置，不关心标准蓝牙功能。

```
// 配置为0，SDK 初始化后默认广播。配置为1，SDK 初始化不广播，需要通过设备 UI 操作触发广播（例如按键）。
#define BLE_QIOT_BUTTON_BROADCAST       (0)
```

```
#define __ORDER_LITTLE_ENDIAN__              (1234)
#define __ORDER_BIG_ENDIAN__                 (4321)
// 设备大小端定义。
#define __BYTE_ORDER__                       (__ORDER_LITTLE_ENDIAN__)

// 设备端串口输出函数。
#define BLE_QIOT_LOG_PRINT(...)              printf(__VA_ARGS__)
#define BLE_QIOT_USER_DEFINE_HEXDUMP     0

// 设备端和小程序通信过程中的单条数据最大长度，配网功能默认128即可。
#define BLE_QIOT_EVENT_MAX_SIZE              (128)

// 取MTU和BLE_QIOT_EVENT_MAX_SIZE中的较小值，LLSync会对数据分片后再发送。
#define BLE_QIOT_EVENT_BUF_SIZE              (23)

// 配置为1，建立蓝牙连接后小程序尝试设置MTU，通过ble_get_user_data_mtu_size接口获取；配置为0，默认MTU为23。
#define BLE_QIOT_REMOTE_SET_MTU              (1)

// 配置为1，使能LLSync配网功能
#define BLE_QIOT_LLSYNC_CONFIG_NET       (1)
```

LLSync 使用配网功能不需要 ota、数据模板等能力， `SDK` 会通过功能宏控制编译。

1.2 HAL 实现

`inc/ble_qiot_import.h` 中定义了 `LLSync SDK` 依赖的设备 `HAL` 实现，需要您在自己的硬件平台上进行实现。

```
    /* 获取设备MAC地址，示例:
    int ble_get_mac(char *mac)
    {
        char *address = (char *)esp_bt_dev_get_address();
        memcpy(mac, address, 6);
        return 0;
    }
    */
    int ble_get_mac(char *mac);

    /* 设置WIFI模式，当前支持wifi station，示例:
    ble_qiot_ret_status_t ble_combo_wifi_mode_set(BLE_WIFI_MODE mode)
    {
        if (mode != BLE_WIFI_MODE_STA){
            return ble_event_report_wifi_mode(-1)
        }
        ESP_ERROR_CHECK(esp_wifi_set_mode(mode);
        return ble_event_report_wifi_mode(0);
    }
    */
    ble_qiot_ret_status_t ble_combo_wifi_mode_set(BLE_WIFI_MODE mode);

    /* 设置WIFI SSID和PASSWORD，示例:
    ble_qiot_ret_status_t ble_combo_wifi_info_set(const char *ssid, uint8_t ssid_len, const char
*passwd, uint8_t passwd_len)
    {
         ......
         memcpy(wifi_config.sta.ssid, ssid, ssid_len);
        memcpy(wifi_config.sta.password, passwd, passwd_len);
        ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
        return ble_event_report_wifi_info(0);
    }
    */
```

```
    ble_qiot_ret_status_t ble_combo_wifi_info_set(const char *ssid, uint8_t ssid_len, const char
*passwd, uint8_t passwd_len);

    /* 使用设置的WIFI信息请求连接WIFI，示例：
    ble_qiot_ret_status_t ble_combo_wifi_connect()
    {
        ESP_ERROR_CHECK(esp_wifi_connect());
        return 0;
    }
    */
    ble_qiot_ret_status_t ble_combo_wifi_connect();
```

`inc/ble_qiot_import.h` 中定义了 `LLSync SDK` 依赖的蓝牙协议栈 `HAL` 实现，需要您在自己的硬件平台上进行实现。

```
    /* 通过ble_get_qiot_services获取蓝牙服务，将蓝牙服务添加到协议栈。配网功能使用的服务UUID是0xFFF0，特征值UUID
是0xFFE1和0xFFE3。您也可以选择其他方式将服务添加到协议栈，示例为ESP32上添加蓝牙服务代码：
    static uint8_t llsync_service_uuid[16] = {
        0xe2, 0xa4, 0x1b, 0x54, 0x93, 0xe4, 0x6a, 0xb5, 0x20, 0x4e, 0xd0, 0x65, 0xf0, 0xff, 0x00,
0x00,};
    static uint8_t llsync_device_info_uuid[16] = {
        0xe2, 0xa4, 0x1b, 0x54, 0x93, 0xe4, 0x6a, 0xb5, 0x20, 0x4e, 0xd0, 0x65, 0xe1, 0xff, 0x00,
0x00,};
    static uint8_t llsync_event_uuid[16] = {
        0xe2, 0xa4, 0x1b, 0x54, 0x93, 0xe4, 0x6a, 0xb5, 0x20, 0x4e, 0xd0, 0x65, 0xe3, 0xff, 0x00,
0x00,};
    static const esp_gatts_attr_db_t gatt_db[HRS_IDX_NB] = {
        [IDX_SVC] = {{ESP_GATT_AUTO_RSP},
                     {ESP_UUID_LEN_16, (uint8_t *)&primary_service_uuid, ESP_GATT_PERM_READ,
sizeof(llsync_service_uuid), sizeof(llsync_service_uuid), (uint8_t *)llsync_service_uuid}},
        [IDX_CHAR_A] = {{ESP_GATT_AUTO_RSP},
                        {ESP_UUID_LEN_16, (uint8_t *)&character_declaration_uuid, ESP_GATT_PERM_READ,
CHAR_DECLARATION_SIZE, CHAR_DECLARATION_SIZE, (uint8_t *)&char_prop_write}},
        [IDX_CHAR_VAL_A] = {{ESP_GATT_AUTO_RSP},
                            {ESP_UUID_LEN_128, (uint8_t *)llsync_device_info_uuid,
ESP_GATT_PERM_WRITE,
                             LLSYNC_CHAR_VAL_LEN_MAX, 0, NULL}},
        [IDX_CHAR_C] = {{ESP_GATT_AUTO_RSP},
                        {ESP_UUID_LEN_16, (uint8_t *)&character_declaration_uuid, ESP_GATT_PERM_READ,
CHAR_DECLARATION_SIZE, CHAR_DECLARATION_SIZE, (uint8_t *)&char_prop_notify}},
        [IDX_CHAR_VAL_C] = {{ESP_GATT_AUTO_RSP},
                            {ESP_UUID_LEN_128, (uint8_t *)llsync_event_uuid, 0,
LLSYNC_CHAR_VAL_LEN_MAX, 0, NULL}},
        [IDX_CHAR_CFG_C] = {{ESP_GATT_AUTO_RSP},
                            {ESP_UUID_LEN_16, (uint8_t *)&character_client_config_uuid,
                             ESP_GATT_PERM_READ | ESP_GATT_PERM_WRITE, sizeof(uint16_t),
sizeof(heart_measurement_ccc), (uint8_t *)heart_measurement_ccc}},
    }
    ……
    esp_err_t create_attr_ret = esp_ble_gatts_create_attr_tab(gatt_db, gatts_if, HRS_IDX_NB,
SVC_INST_ID);
    };
    */
    void ble_services_add(const qiot_service_init_s *p_service);

    /* 通过ble_get_my_broadcast_data获取广播数据，将其作为厂商数据开始广播，Compnay ID是0xFEE7。示例：
    ……
    adv_info_s my_adv_info;
    adv_data_len = ble_get_my_broadcast_data((char *)adv_data, sizeof(adv_data));
    my_adv_info.manufacturer_info.company_identifier = TENCENT_COMPANY_IDENTIFIER;
    my_adv_info.manufacturer_info.adv_data           = adv_data;
```

```
    my_adv_info.manufacturer_info.adv_data_len       = adv_data_len;
    ble_advertising_start(&my_adv_info);
    ……
    ble_qiot_ret_status_t ble_advertising_start(adv_info_s *adv)
    {
         static uint8_t raw_adv_data[32] = {0x02, 0x01, 0x06, 0x03, 0x03, 0xF0, 0xFF};
        uint8_t usr_adv_data[31] = {0};
        uint8_t len              = 0;
        uint8_t index            = 0;
        memcpy(usr_adv_data, &adv->manufacturer_info.company_identifier, sizeof(uint16_t));
        len = sizeof(uint16_t);
        memcpy(usr_adv_data + len, adv->manufacturer_info.adv_data, adv-
>manufacturer_info.adv_data_len);
        len += adv->manufacturer_info.adv_data_len;

        index                 = 7;
        raw_adv_data[index++] = len + 1;
        raw_adv_data[index++] = 0xFF;
        // 添加厂商广播数据
        memcpy(raw_adv_data + index, usr_adv_data, len);
        index += len;
        ……
        esp_log_buffer_hex("adv", raw_adv_data, index);
        esp_err_t ret = esp_ble_gap_config_adv_data_raw(raw_adv_data, index);
        if (ret) {
            ESP_LOGE(LLSYNC_LOG_TAG, "config adv data failed, error code = %x", ret);
        }
        adv_config_done |= ADV_CONFIG_FLAG;
        esp_ble_gap_start_advertising(&adv_params);
        return 0;
    }
    */
    ble_qiot_ret_status_t ble_advertising_start(adv_info_s *adv);

    /* 停止广播接口，示例：
    ble_qiot_ret_status_t ble_advertising_stop(void)
    {
        esp_ble_gap_stop_advertising();
        return 0;
    }
    */
    ble_qiot_ret_status_t ble_advertising_stop(void);

    /* 特征值UUID FFE3向小程序写数据的接口，示例：
    ble_qiot_ret_status_t ble_send_notify(uint8_t *buf, uint8_t len)
    {
        esp_ble_gatts_send_indicate(llsync_profile_tab[PROFILE_APP_IDX].gatts_if,
            llsync_profile_tab[PROFILE_APP_IDX].conn_id, llsync_handle_table[IDX_CHAR_VAL_C], len,
buf, false);
        return BLE_QIOT_RS_OK;
    }
    */
    ble_qiot_ret_status_t ble_send_notify(uint8_t *buf, uint8_t len);

    /* 用户指定MTU。当BLE_QIOT_REMOTE_SET_MTU设置为1时，小程序连接设备后会去修改MTU并通知设备。否则使用默认
MTU(23)进行通信。
    uint16_t ble_get_user_data_mtu_size(void)
    {
        return 128;
    }
    */
```

```
uint16_t ble_get_user_data_mtu_size(void);
```

1.3 API 调用

`inc/ble_qiot_export.h` 中定义了 `LLSync SDK` 对外提供的 `API` 。

```
/* 获取LLSync蓝牙服务，您可以在代码中获取蓝牙服务后将蓝牙服务添加到蓝牙协议栈*/
const qiot_service_init_s *ble_get_qiot_services(void);

/* LLSync SDK初始化接口，主要进行蓝牙服务添加。默认在start_device_btcomboconfig内已调用，您也可以选择在其他位置调用。*/
ble_qiot_ret_status_t ble_qiot_explorer_init(void);

/* LLSync广播启动接口，请您选择合适的位置调用。例如在蓝牙断开时重新开始广播。
……
ESP_LOGI(LLSYNC_LOG_TAG, "ESP_GATTS_DISCONNECT_EVT, reason = 0x%x", param->disconnect.reason);
ble_qiot_advertising_start();
……
*/
ble_qiot_ret_status_t ble_qiot_advertising_start(void);

/* LLSync广播停止接口，请您选择合适的位置调用。例如在配网结束时停止广播。
int stop_device_btcomboconfig(void)
{
    ble_qiot_advertising_stop();
    return QCLOUD_RET_SUCCESS;
}
*/
ble_qiot_ret_status_t ble_qiot_advertising_stop(void);

/* GAP连接时通知LLSync SDK。例如:
case ESP_GATTS_CONNECT_EVT:
     ……
     esp_ble_gap_update_conn_params(&conn_params);
     ble_gap_connect_cb();
    ……
*/
void ble_gap_connect_cb(void);

/* GAP断开时通知LLSync SDK。例如:
……
ESP_LOGI(LLSYNC_LOG_TAG, "ESP_GATTS_DISCONNECT_EVT, reason = 0x%x", param->disconnect.reason);
ble_gap_disconnect_cb();
……
*/
void ble_gap_disconnect_cb(void);

/* 小程序无法获取蓝牙MTU接口，设备端收到MTU修改通知后调用此接口通知小程序。例如:
case ESP_GATTS_MTU_EVT:
     ESP_LOGI(LLSYNC_LOG_TAG, "ESP_GATTS_MTU_EVT, MTU %d", param->mtu.mtu);
     ble_event_sync_mtu(param->mtu.mtu);
     break;
*/
ble_qiot_ret_status_t ble_event_sync_mtu(uint16_t llsync_mtu);

/* 特征值UUID FFE1数据处理接口，收到数据后调用此接口。例如:
case ESP_GATTS_WRITE_EVT:
     if (!param->write.is_prep) {
            // the data length of gattc write  must be less than LLSYNC_CHAR_VAL_LEN_MAX.
            if (param->write.handle == llsync_handle_table[IDX_CHAR_VAL_A]) {
                   ble_device_info_write_cb(param->write.value, param->write.len);}
```

```
                ……
        }
    */
    void ble_device_info_write_cb(const uint8_t *buf, uint16_t len);

    /* 设置WIFI Mode后，设备端给小程序回复设置结果。result 为0表示设置成功，其他表示错误。示例：
    ble_qiot_ret_status_t ble_combo_wifi_mode_set(BLE_WIFI_MODE mode)
    {
        if (mode != BLE_WIFI_MODE_STA){
            return ble_event_report_wifi_mode(-1)
        }
        ESP_ERROR_CHECK(esp_wifi_set_mode(mode);
        return ble_event_report_wifi_mode(0);
    }
    */
    ble_qiot_ret_status_t ble_event_report_wifi_mode(uint8_t result);

    /* 设置WIFI信息后，设备端给小程序回复设置结果。result 为0表示设置成功，其他表示错误。示例：
    ble_qiot_ret_status_t ble_combo_wifi_info_set(const char *ssid, uint8_t ssid_len, const char
*passwd, uint8_t passwd_len)
    {
         ......
         memcpy(wifi_config.sta.ssid, ssid, ssid_len);
        memcpy(wifi_config.sta.password, passwd, passwd_len);
        ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
        return ble_event_report_wifi_info(0);
    }
    */
    ble_qiot_ret_status_t ble_event_report_wifi_info(uint8_t result);

    /* WIFI连接结束后，将WIFI连接结果回复给小程序。WIFI连接成功后回复示例：
    static void bt_combo_report_wificonn_success()
    {
        wifi_config_t cfg;
        esp_wifi_get_config(WIFI_IF_STA, &cfg);
        ble_event_report_wifi_connect(BLE_WIFI_MODE_STA, BLE_WIFI_STATE_CONNECT,
(uint8_t)strlen((const char *)cfg.sta.ssid), (const char *)cfg.sta.ssid);
    }
    注意：WIFI连接失败也要回复小程序。
    */
    ble_qiot_ret_status_t ble_event_report_wifi_connect(BLE_WIFI_MODE mode, BLE_WIFI_STATE state,
uint8_t ssid_len, const char *ssid);
```

2. `C SDK` 移植请参考 C SDK 使用参考 。`C SDK` 已经实现了配网代码框架，您只需要根据需要实现相关的 `HAL` 接口即可。为进一步简化适配步骤，在 `qcloud-iot-explorer-BLE-sdk-embedded/lib/qcloud_iot_c_sdk` 目录下已经实现了部分 `C SDK` 的 HAL 接口，您需要拷贝文件覆盖 `qcloud-iot-explorer-sdk-embedded-c/platform` 和 `qcloud-iot-explorer-sdk-embedded-c/sdk_src/wifi_config` 目录下对应文件。

```
    cp qcloud-iot-explorer-BLE-sdk-embedded/lib/qcloud_iot_c_sdk/platform/* qcloud-iot-explorer-sdk-
embedded-c/platform/
    cp qcloud-iot-explorer-BLE-sdk-embedded/lib/qcloud_iot_c_sdk/sdk_src/* qcloud-iot-explorer-sdk-
embedded-c/sdk_src/wifi_config/
```

您还需要实现以下 HAL 接口以完成配网功能的适配工作。

```
    /* COMBO设备配网功能启动接口，默认调用了LLSync SDK初始化接口，您可以按照需要修改。示例
    int start_device_btcomboconfig(void)
    {
        // TODO other init（例如蓝牙协议栈初始化）
        ble_qiot_explorer_init();        // init llsync sdk
```

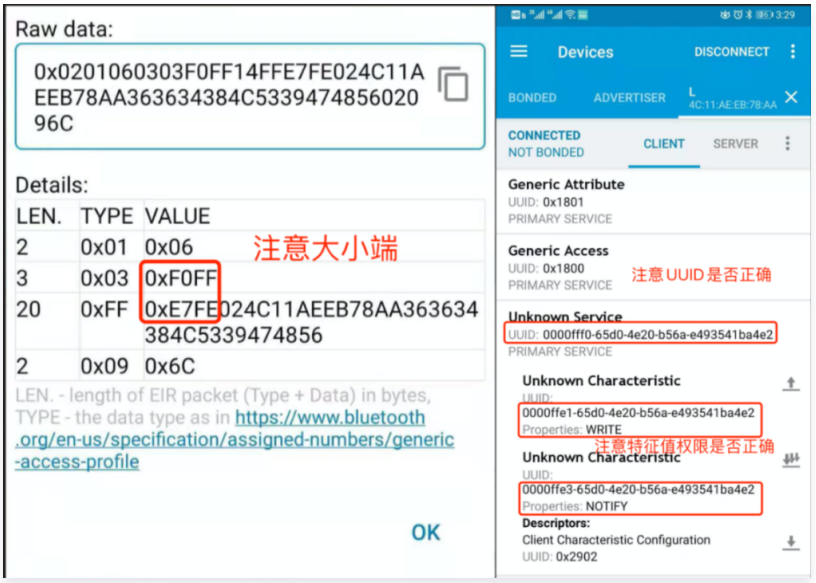
```
        return QCLOUD_RET_SUCCESS;
    }
    */
    int start_device_btcomboconfig(void);

    /* COMBO设备配网功能关闭接口，默认调用了LLSync停止广播接口，您可以按照需要修改。示例
    int stop_device_btcomboconfig(void)
    {
        ble_qiot_advertising_stop();
        // TODO other deinit
        return QCLOUD_RET_SUCCESS;
    }
    */
    int stop_device_btcomboconfig(void)

    /* WIFI连接成功后上报给小程序，您需要获取当前连接的WIFI信息并调用ble_event_report_wifi_connect接口上报。示例
    static void bt_combo_report_wificonn_success()
    {
        wifi_config_t cfg;
        esp_wifi_get_config(WIFI_IF_STA, &cfg);
        ble_event_report_wifi_connect(BLE_WIFI_MODE_STA, BLE_WIFI_STATE_CONNECT, (uint8_t)strlen((const
char *)cfg.sta.ssid), (const char *)cfg.sta.ssid);
    }
    */
    void bt_combo_report_wificonn_success();

    /* 设置 WIFI 配置状态，当 WIFI 信息设置后根据结果修改状态。示例
    ……
    ESP_LOGI(TAG, "wifi info %s, %s", wifi_config.sta.ssid,wifi_config.sta.password);
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    // WIFI 信息设置成功
    set_bt_combo_config_state(WIFI_CONFIG_SUCCESS);
    ……
    */
    void set_bt_combo_config_state(eWiFiConfigState state);

    /* 获取 WIFI 连接状态，建议在获取到IP后才认为WIFI连接成功。示例
    bool HAL_Wifi_IsConnected(void)
    {
        // TODO, retrun true when you got ip
        return false;
    }
    */
    bool HAL_Wifi_IsConnected(void);

    /* 配网错误日志操作接口，需要固化存储日志时可以实现以下接口。当出现错误时设备存储日志，设备重启时读取日志，防止错误信息
丢失。
    */
    int HAL_Wifi_read_err_log(uint32_t offset, void *log, size_t log_size);
    int HAL_Wifi_write_err_log(uint32_t offset, void *log, size_t log_size);
    int HAL_Wifi_clear_err_log(void);
```

`qcloud-iot-explorer-sdk-embedded-c/sdk_src/internal_inc/qcloud_wifi_config.h` 中进行选择配网功能配置。

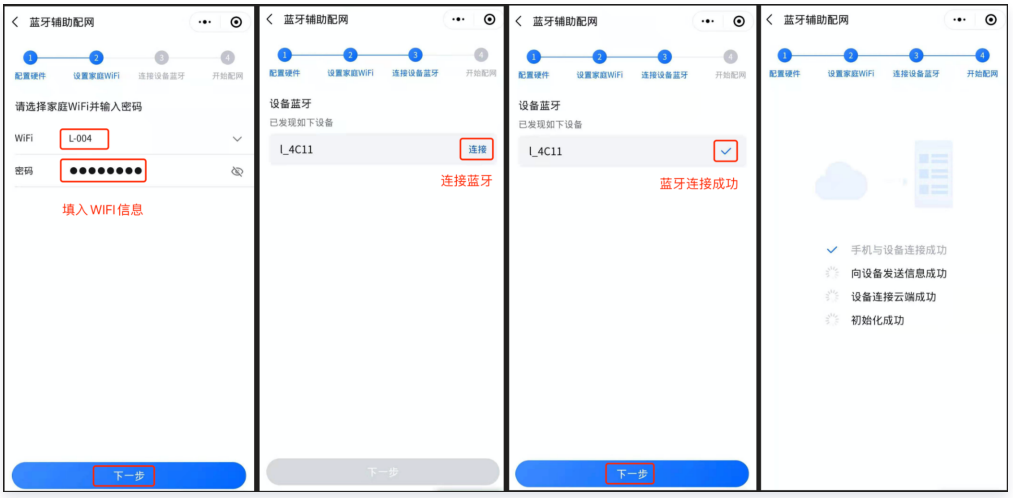
```
    #define WIFI_ERR_LOG_POST 1       // 配网出错时向小程序上报配网错误日志
    #define WIFI_LOG_UPLOAD   1     // 配网出错时向小程序上报配网过程日志
    #define WIFI_PROV_BT_COMBO_CONFIG_ENABLE 1    // 选择LLSync配网功能，soft ap，airkiss等配网选项需要关闭
```

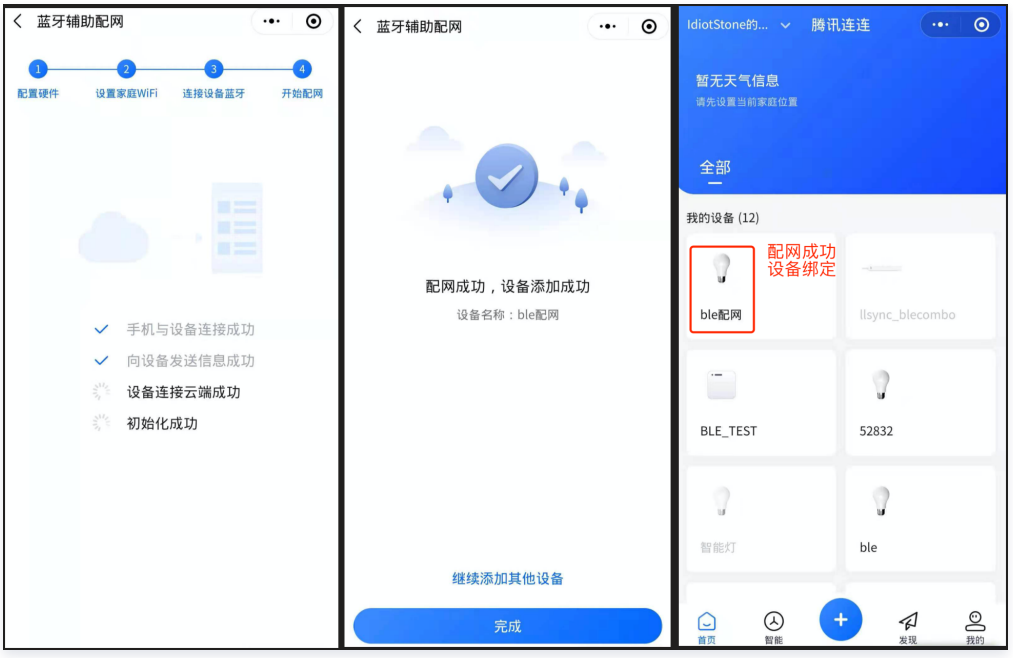
五、验证蓝牙辅助配网功能

1. BLE 适配完成后，可以通过 `Nrf Connect` 查看设备广播信息和蓝牙服务信息。



2. 小程序 LLSync 配网示例。


蓝牙辅助配网
配置硬件
设置家庭WiFi
连接设备蓝牙
开始配网
请选择家庭WiFi并输入密码
WiFi
L-004
密码
填入WIFI信息
下一步
设备蓝牙
已发现如下设备
l_4C11
连接
连接蓝牙
蓝牙连接成功
手机与设备连接成功
向设备发送信息成功
设备连接云端成功
初始化成功


蓝牙辅助配网
配置硬件
设置家庭WiFi
连接设备蓝牙
开始配网
手机与设备连接成功
向设备发送信息成功
设备连接云端成功
初始化成功
配网成功，设备添加成功
设备名称：ble配网
继续添加其他设备
完成
IdiotStone的...
腾讯连连
暂无天气信息
请先设置当前家庭位置
全部
我的设备 (12)
配网成功
设备绑定
ble配网
llsync_blecombo
BLE_TEST
52832
智能灯
ble
首页
智能
发现
我的

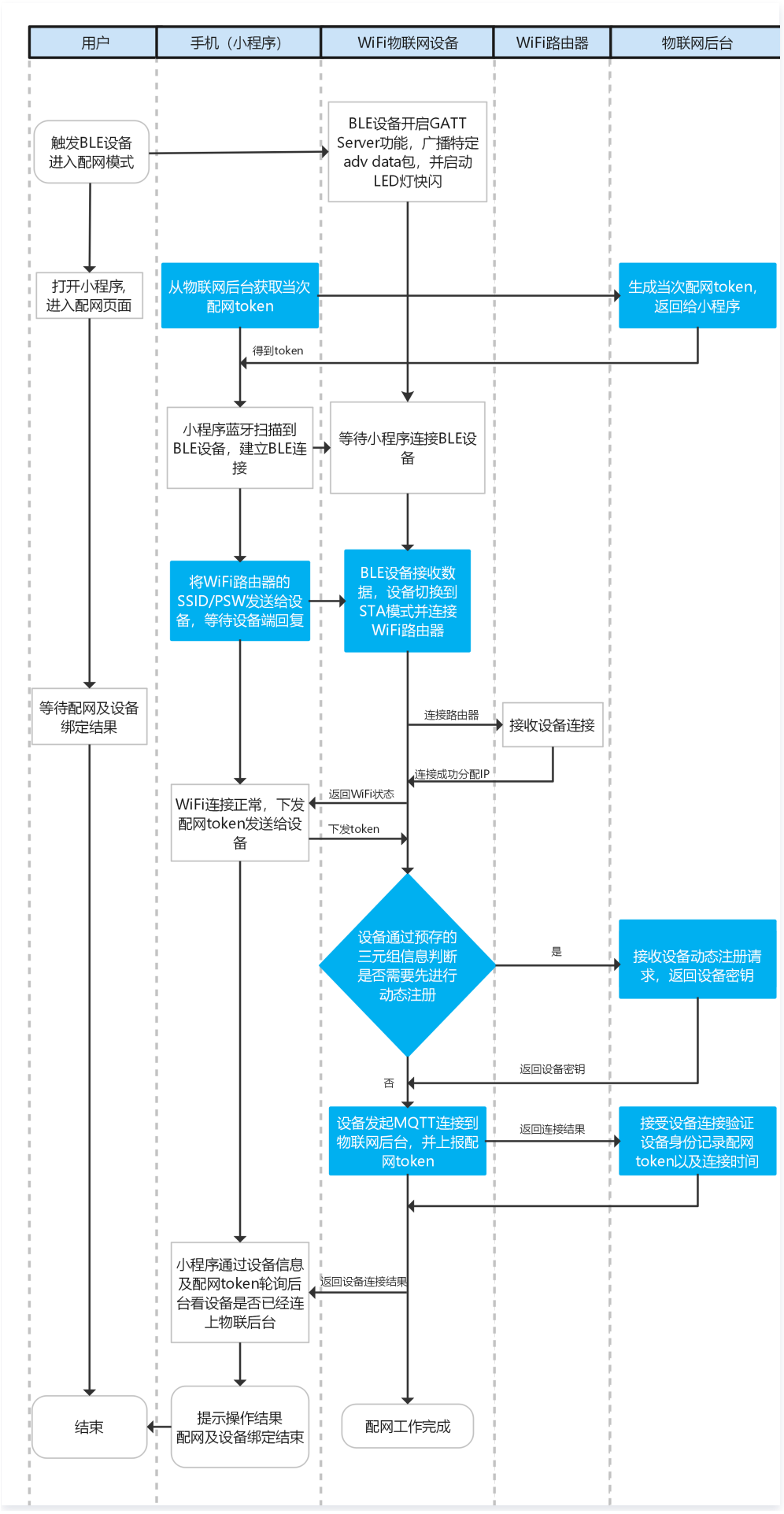
blufi 蓝牙辅助配网开发

最近更新时间：2024-08-26 14:55:51

操作场景

基本原理

blufi 蓝牙辅助配网是针对 ESP32 Combo 芯片的配网方案，通过 BLE 创建指定的 GATT 服务，手机连接该 GATT SERVER，利用 BLE 的无线通信能力，将物联网设备连接所需的 SSID、PSW 等信息传输给 Wi-Fi+BLE 的 Combo 芯片或模组，使设备顺利接入物联网平台，继而完成设备绑定等功能。
目前腾讯连连小程序已支持采用 blufi 蓝牙辅助配网协议进行配网 Demo 开发，并提供相应的 小程序 SDK 。
blufi 蓝牙辅助方式配网及设备绑定的示例流程图如下：


用户
手机（小程序）
WiFi物联网设备
WiFi路由器
物联网后台
触发BLE设备进入配网模式
BLE设备开启GATT Server功能，广播特定adv data包，并启动LED灯快闪
打开小程序，进入配网页面
从物联网后台获取当次配网token
生成当次配网token，返回给小程序
得到token
小程序蓝牙扫描到BLE设备，建立BLE连接
等待小程序连接BLE设备
将WiFi路由器的SSID/PSW发送给设备，等待设备端回复
BLE设备接收数据，设备切换到STA模式并连接WiFi路由器
等待配网及设备绑定结果
连接路由器
接收设备连接
连接成功分配IP
返回WiFi状态
WiFi连接正常，下发配网token发送给设备
下发token
设备通过预存的三元组信息判断是否需要先进行动态注册
是
接收设备动态注册请求，返回设备密钥
返回设备密钥
否
设备发起MQTT连接到物联网后台，并上报配网token
返回连接结果
接受设备连接验证设备身份记录配网token以及连接时间
小程序通过设备信息及配网token轮询后台看设备是否已经连上物联后台
返回设备连接结果
结束
提示操作结果配网及设备绑定结束
配网工作完成

操作步骤

蓝牙辅助配网协议示例

本示例基于 ESP32 腾讯云定制模组配合腾讯连连小程序。

1. 腾讯连连小程序进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关操作可以参考 生成 Wi-Fi 设备配网 Token 。
2. 使 Wi-Fi+BLE 设备进入蓝牙辅助配网模式，若设备有指示灯在快闪，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，再选择连接 Wi-Fi+BLE 设备蓝牙服务。
4. 手机连接 Wi-Fi+BLE 设备蓝牙服务成功后，小程序作为 GATT Client 会向 Wi-Fi+BLE 设备分别下发设置 Wi-Fi station 模式、下发 SSID 信息、下发 PSW 信息，最后下发 Wi-Fi+BLE 设备连接的 AP 指令。发送完成后，等待 Wi-Fi+BLE 设备回复 Wi-Fi 连接状态及 SSID 信息。
5. 如果步骤4收到设备回复，则说明设备端已收到 Wi-Fi 路由器的 SSID/PSW。此时小程序会将当次配网的token下发给 Wi-Fi+BLE 设备。
6. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将小程序发送的配网 Token，通过下面 MQTT 报文上报给后台服务：

```
topic: $thing/up/service/ProductID/DeviceName
payload: {"method":"app_bind_token","clientToken":"client-1234","params":
{"token":"6****345****234ee77****e528a0fd"}}
```

   设备端也可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 Token 上报的结果。
7. 设备端将 Token 的绑定结果上报至小程序，至此配网结束。
8. 如果设备成功上报了 Token，物联网后台服务已确认 Token 有效性，小程序会提示配网完成，设备添加成功。

ESP32 使用 蓝牙辅助 配网接口

配网协议在 ESP32 设备端的参考代码，请参见 GitHub 工程 esp-qcloud 。

二维码配网开发

最近更新时间：2023-10-24 09:35:51

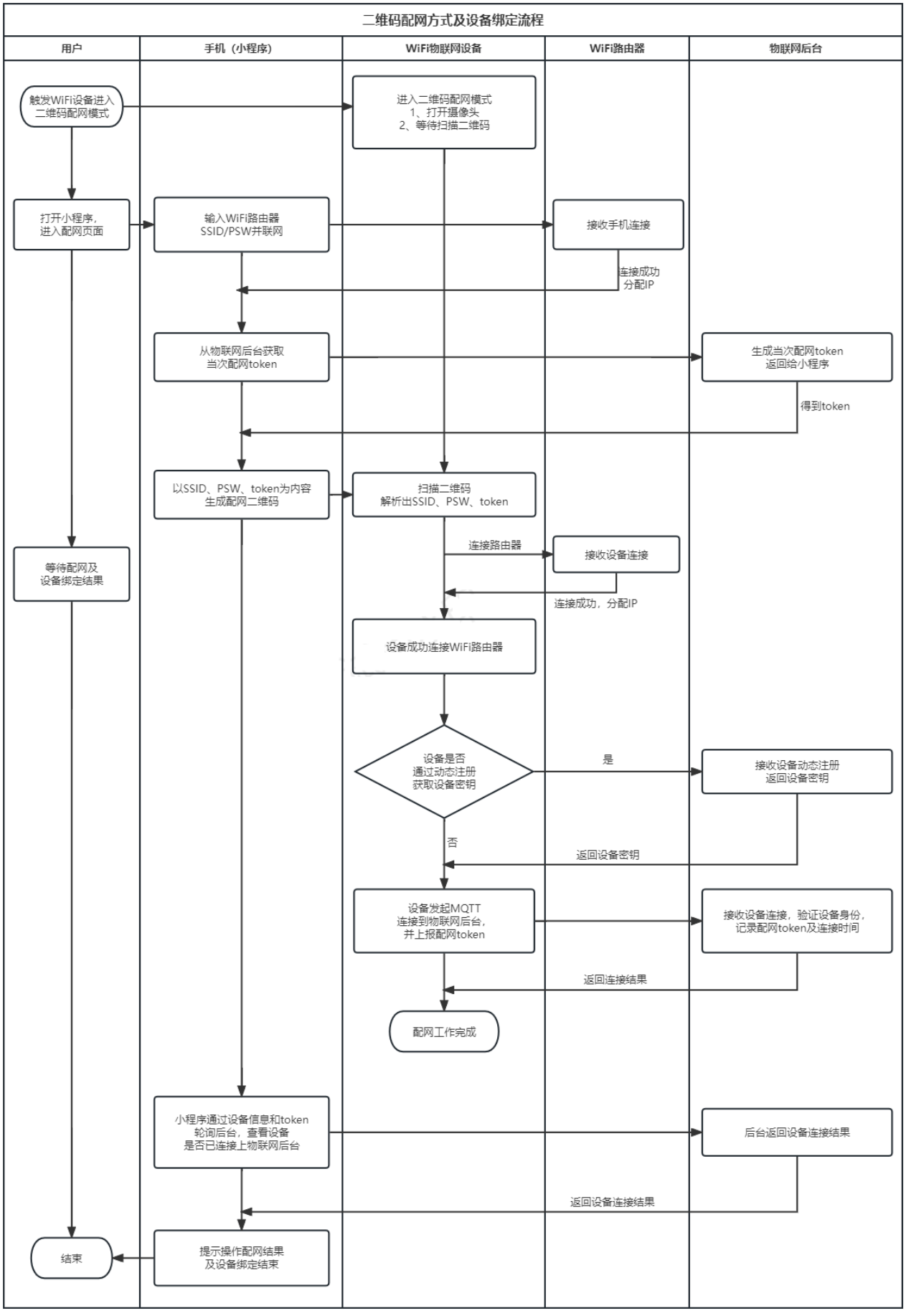
操作场景

基本原理

二维码配网是针对带有摄像头的产品提供的配网方案。在这种方案中，物联网设备通过读取和解析二维码信息，将连接所需的 SSID、PSW 等信息传输给设备，从而使设备能够顺利接入物联网平台。设备将后台提供的配网 Token 发送至物联网后台，后台可以依据 Token 进行设备绑定。本文档将主要指导您如何使用二维码配网开发。

腾讯连连小程序已经支持二维码配网。二维码数据格式：Wi-Fi 名字长度 + Wi-Fi 名字字符串 + 密码长度 + 密码字符串 + token 长度 + token 字符串。

二维码配网及设备绑定的示例流程图如下：


二维码配网方式及设备绑定流程
用户
手机（小程序）
WiFi物联网设备
WiFi路由器
物联网后台
触发WiFi设备进入二维码配网模式
进入二维码配网模式
1、打开摄像头
2、等待扫描二维码
打开小程序，进入配网页面
输入WiFi路由器SSID/PSW并联网
接收手机连接
连接成功
分配IP
从物联网后台获取当次配网token
生成当次配网token返回给小程序
得到token
以SSID、PSW、token为内容生成配网二维码
扫描二维码
解析出SSID、PSW、token
连接路由器
接收设备连接
等待配网及设备绑定结果
连接成功，分配IP
设备成功连接WiFi路由器
设备是否通过动态注册获取设备密钥
是
接收设备动态注册返回设备密钥
否
返回设备密钥
设备发起MQTT连接到物联网后台，并上报配网token
接收设备连接，验证设备身份，记录配网token及连接时间
返回连接结果
配网工作完成
小程序通过设备信息和token轮询后台，查看设备是否已连接上物联网后台
后台返回设备连接结果
返回设备连接结果
结束
提示操作配网结果及设备绑定结果

操作步骤

二维码配网协议示例

二维码配网设备端与腾讯连连小程序及后台交互的数据协议操作如下：

1. 腾讯连连小程序选择添加设备，进入配网模式后，则可以在物联网开发平台服务获取到当次配网的 Token。小程序相关操作可以参见 生成 Wi-Fi 设备配网 Token。
2. 使 Wi-Fi 设备进入二维码配网模式，若摄像头打开，则说明进入配网模式成功。
3. 小程序按照提示依次获取 Wi-Fi 列表，输入家里目标路由器的 SSID/PSW，单击下一步生成二维码。
4. Wi-Fi 设备读取小程序上的二维码，解析获取 SSID 信息、PSW 信息、Token，等待 Wi-Fi 设备回复 Wi-Fi 连接状态。
5. 如果步骤4收到设备回复，则说明设备端已收到 Wi-Fi 路由器的 SSID/PSW。
6. 设备端在成功连接 Wi-Fi 路由器后，需要通过 MQTT 连接物联网后台，并将获得的配网 Token，通过下面 MQTT 报文上报给后台服务：

```
topic: $thing/up/service/ProductID/DeviceName
payload: {"method":"app_bind_token","clientToken":"client-1234","params":
{"token":"6****345****234ee77****e528a0fd"}}
```

   设备端也可以通过订阅主题 $thing/down/service/ProductID/DeviceName 来获取 Token 上报的结果。
7. 设备端将 Token 的绑定结果上报至小程序，至此配网结束。
8. 如果设备成功上报了 Token，物联网后台服务已确认 Token 有效性，小程序会提示配网完成，设备添加成功。

已认证模组

最近更新时间：2022-08-01 11:15:33

简介

腾讯云物联网提供专用的腾讯云 IoT AT 指令集（例如：Wi-Fi 版，蜂窝版），如果通讯模组实现了该指令集，则设备接入和通讯更为简单，所需代码量更少，针对此场景，请参考面向腾讯云定制 AT 模组专用的 MCU AT SDK。
目前腾讯云和主流的模组厂商已进行深度合作，将 SDK 的核心协议移植到模组中，模组对外封装统一的腾讯云 AT 指令。已支持腾讯云定制 AT 指令的模组列表如下：

序号	模组商	模组型号	通信制式
1	博通	BL2028N	Wi-Fi+BLE
2	博通	BK3432	BLE
3	集贤	UAE051	BLE
4	集贤	UAE050	BLE
5	集贤	UAW026	Wi-Fi
6	力合微	PLM4010B	PLBUS PLC
7	力合微	LM010B-LC200	PLBUS PLC
8	力合微	LM011B-TG100	PLBUS PLC+Wi-Fi+BLE
9	力合微	LM980B-DC100	PLBUS PLC
10	力合微	LM980B-DC200	PLBUS PLC
11	力合微	LM980B-DC300	PLBUS PLC
12	乐鑫	ESP8266	Wi-Fi
13	乐鑫	ESP32	Wi-Fi+BLE
14	安信可	ESP-12S	Wi-Fi
15	安信可	PB-01/02	BLE
16	安信可	TB-02/03F/04	BLE
17	村田	MBN52832	BLE
18	村田	CMWC1ZZABR	Wi-Fi
19	朗国	XF.H3861.A	Wi-Fi+BLE
20	中移	M5310-A	NB-IoT
21	中移	M5311	NB-IoT
22	中移	M6315	2G
23	中移	M8321	4G
24	中移	ML302	LTE Cat.1
25	有方	N10	2G
26	有方	N21	NB-IoT
27	有方	N58	NB-IoT

28	有方	N720	4G
29	移远	EC600	LTE Cat.1
30	移远	FC41D	Wi-Fi
31	移柯	L206D	2G
32	移柯	L501C	LTE Cat.1
33	移柯	L620C	NB-IoT
34	广和通	L610	LTE Cat.1
35	高新兴	ME3616	NB-IoT
36	镁云	M-628	BLE
37	镁云	M-623	BLE
38	佳域顺芯	BC203/BC204	BLE
39	云希谷	XiaoGu	Wi-Fi
40	遥看	YKK221V	Wi-Fi
41	遥看	YKK-RF210A	Wi-Fi
42	遥看	YKK-RF510S	Wi-Fi
43	遥看	YKK-RF610S	Wi-Fi
44	爱联	WF-R710-RTS1	Wi-Fi
45	爱联	WF-R710-RPA1	Wi-Fi

基于 RT-Thread SDK 使用参考

最近更新时间：2022-06-10 15:15:07

Real Time-Thread（以下简称 RT-Thread）是一个嵌入式实时多线程操作系统，C SDK 支持以 软件包 的形式应用到 RT-Thread 操作系统，本文向您介绍如何使用 RT-Thread 快速接入腾讯云物联网开发平台。

操作步骤

RT-Thread 接入腾讯云物联网开发平台可以分为以下4个步骤。

步骤一：开发环境安装

安装 RT-Thread 开发环境，详情请参见 RT-Thread 开发环境搭建 。

步骤二：软件包下载

1. 执行 `scons --menuconfig` 命令打开配置面板。
   勾选 `【RT-Thread online packages】>【IoT - internet of things】>【IoT Cloud】>【tencent-iot-sdk】` 。

```
[ ] OneNET: China Mobile OneNet cloud SDK for RT-Thread  ----
[ ] GAgent: GAgent of Gizwits in RT-Thread  ----
[ ] Ali-iotkit: Aliyun cloud sdk 'iotkit-embedded' for RT-Thread  ----
[ ] Azure IoT SDK: Microsoft azure cloud SDK for RT-Thread  ----
[*] Tencent-IoT:  Tencent Cloud IoT Explorer Platform SDK for RT-Thread  --->
[ ] jiot-c-sdk: JIGUANG IoT Cloud Client SDK for RT_Thread  ----
[ ] ucloud_iot_sdk: Ucloud iot sdk for uiot-core platform.  ----
[ ] Joylink Cloud SDK for IoT platform  ----
```

2. 执行 `pkgs --update` 命令更新软件包，腾讯云物联网 C SDK 将被下载到 packages 目录。

步骤三：编译与运行

1. 执行 `scons --menuconfig` 命令打开配置面板，对 C SDK 进行配置。
   - 选择路径：`RT-Thread online packages => IoT - internet of things => IoT Cloud => tencent-iot-sdk` 。

```
--- Tencent-IoT:  Tencent Cloud IoT Explorer Platform SDK for RT-Thread
(0WUKPUCOTC) Config Product Id
(dev001) Config Device Name
(N6B8M91PB4YDTRCpqvOp4w==) Config Device Secret
[ ]   Enable dynamic register
[ ]   Enable err log upload
[ ]   Enable multi thread function
[*]   Enable TLS/DTLS
        Select Product Type (Data template protocol)  --->
[*]   Enable Event
[*]   Enable Action
[*]   Enable Smart_light Sample
[*]   Enable OTA
          Config OTA download by https or http (Download by http)  --->
[ ]   Enable GateWay
        Version (latest)  --->
```

   - 参数说明：
     - **Config Product Id**：配置产品 ID，平台创建生成。
     - **Config Device Name**：配置设备名，平台创建生成。
     - **Config Device Secret**：配置设备密钥，平台创建生成。考虑到嵌入式设备大多没有文件系统，暂时没有支持证书设备配置。
     - **Enable dynamic register**：是否使能动态注册功能及示例，若使能，需配置动态注册的产品密钥。
     - **Enable err log upload**：是否使能错误日志上传云端。

- **Enable TLS/DTLS**：是否使能 TLS，若使能，则会关联选中 mbedTLS 软件包。
- **Select Product Type**：产品类型为自定义或者数据模板协议产品。
- **Enable event**：选用数据模板的前提下，是否使能事件功能。
- **Enable Action**：选用数据模板的前提下，是否使能行为功能。
- **Enable Smart_light Sample**：是否使能智能灯场景示例。
- **Enable OTA**：是否使能 OTA 示例。若使能 OTA 可进一步选择下载使用 HTTPS 或者 HTTP。
- **Enable GateWay**：是否使能网关示例。
- **Version**：软件包版本选择，v3.1.2及其以后的版本只支持物联网开发平台，若需使用物联网通信平台，请选择 v3.0.2 及其之前的版本。

2. 编译并运行示例程序。
   本文以数据模板智能灯 + TLS 为例进行介绍，展示设备和物联网开发平台基于 数据模板协议 的通信示例使能 TLS。物联网开发平台下发控制灯为红色的命令，设备端收取消息，打印颜色，并上报对应消息。
   - 配置选项

```
--- Tencent-IoT:  Tencent Cloud IoT Explorer Platform SDK for RT-Thread
--- Tencent-IoT:  Tencent Cloud IoT Explorer Platform SDK for RT-Thread
    (0WUKPUCOTC) Config Product Id
    (dev001) Config Device Name
    (N6B8M91PB4YDTRCpqvOp4w==) Config Device Secret
    [ ]   Enable dynamic register
    [ ]   Enable err log upload
    [ ]   Enable multi thread function
    [*]   Enable TLS/DTLS
          Select Product Type (Data template protocol)  --->
    [*]   Enable Event
    [*]   Enable Action
    [*]   Enable Smart_light Sample
    [ ]   Enable OTA
    [ ]   Enable GateWay
          Version (latest)  --->
```

   - 运行示例

```
 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Jun 15 2020
 2006 - 2020 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[I/sal.skt] Socket Abstraction Layer initialize success.
[I/SDIO] SD card capacity 65536 KB.
hello rt-thread
msh />tc_data_template_light_example start
INF|20|qcloud_iot_device.c|iot_device_info_set(65): SDK_Ver: 3.1.1, Product_ID: 0WUKPUCOTC,
Device_Name: dev001
msh />DBG|20|HAL_TLS_mbedtls.c|HAL_TLS_Connect(228):  Connecting to
/0WUKPUCOTC.iotcloud.tencentdevices.com/8883...
DBG|21|HAL_TLS_mbedtls.c|HAL_TLS_Connect(233):  Setting up the SSL/TLS structure...
DBG|21|HAL_TLS_mbedtls.c|HAL_TLS_Connect(285): Performing the SSL/TLS handshake...
INF|21|mqtt_client.c|IOT_MQTT_Construct(127): mqtt connect with id: qPU1e success
DBG|21|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(141):
topicName=$thing/down/property/0WUKPUCOTC/dev001|packet_id=64736
DBG|21|data_template_client.c|_template_mqtt_event_handler(242): template subscribe success, packet-
id=64736
INF|21|light_data_template_sample.c|event_handler(321): subscribe success, packet-id=64736
INF|21|data_template_client.c|IOT_Template_Construct(785): Sync device data successfully
DBG|21|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(141):
topicName=$thing/down/event/0WUKPUCOTC/dev001|packet_id=64737
```

```
DBG|21|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(141):
topicName=$thing/down/action/0WUKPUCOTC/dev001|packet_id=64738
INF|21|light_data_template_sample.c|data_template_light_thread(650): Cloud Device Construct Success
DBG|21|light_data_template_sample.c|_usr_init(354): add your init code here
INF|21|light_data_template_sample.c|_register_data_template_property(432): data template
property=power_switch registered.
INF|21|light_data_template_sample.c|_register_data_template_property(432): data template
property=color registered.
INF|21|light_data_template_sample.c|_register_data_template_property(432): data template
property=brightness registered.
INF|21|light_data_template_sample.c|_register_data_template_property(432): data template
property=name registered.
INF|21|light_data_template_sample.c|data_template_light_thread(676): Register data template
propertys Success
INF|21|light_data_template_sample.c|_register_data_template_action(294): data template action=blink
registered.
INF|21|light_data_template_sample.c|data_template_light_thread(686): Register data template actions
Success
DBG|21|mqtt_client_publish.c|qcloud_iot_mqtt_publish(340): publish
packetID=0|topicName=$thing/up/property/0WUKPUCOTC/dev001|payload={"method":"report_info",
"clientToken":"0WUKPUCOTC-0", "params":
{"module_hardinfo":"ESP8266","module_softinfo":"V1.0","fw_ver":"3.1.1","imei":"11-22-33-
44","lat":"22.546015","lon":"113.941125", "device_label":{"append_info":"your self define info"}}}
DBG|21|data_template_client.c|_template_mqtt_event_handler(242): template subscribe success, packet-
id=64737
INF|21|light_data_template_sample.c|event_handler(321): subscribe success, packet-id=64737
DBG|21|data_template_client.c|_template_mqtt_event_handler(242): template subscribe success, packet-
id=64738
INF|21|light_data_template_sample.c|event_handler(321): subscribe success, packet-id=64738
DBG|21|data_template_client.c|_reply_ack_cb(169): replyAck=0
DBG|21|data_template_client.c|_reply_ack_cb(172): Received Json Document=
{"method":"report_info_reply","clientToken":"0WUKPUCOTC-0","code":0,"status":"success"}
DBG|21|mqtt_client_publish.c|qcloud_iot_mqtt_publish(340): publish
packetID=0|topicName=$thing/up/property/0WUKPUCOTC/dev001|payload={"method":"get_status",
"clientToken":"0WUKPUCOTC-1"}
DBG|21|data_template_client.c|_get_status_reply_ack_cb(185): replyAck=0
DBG|21|data_template_client.c|_get_status_reply_ack_cb(189): Received Json Document=
{"method":"get_status_reply","clientToken":"0WUKPUCOTC-1","code":0,"status":"success","data":
{"reported":{"power_switch":0,"color":0,"brightness":0,"name":"dev001"}}}
DBG|21|light_data_template_sample.c|data_template_light_thread(709): Get data status success
DBG|22|mqtt_client_publish.c|qcloud_iot_mqtt_publish(340): publish
packetID=0|topicName=$thing/up/property/0WUKPUCOTC/dev001|payload={"method":"report",
"clientToken":"0WUKPUCOTC-2", "params":{"power_switch":0,"color":0,"brightness":0,"name":"dev001"}}
INF|22|light_data_template_sample.c|data_template_light_thread(761): data template reporte success
INF|23|light_data_template_sample.c|OnReportReplyCallback(416): recv report_reply(ack=0):
{"method":"report_reply","clientToken":"0WUKPUCOTC-2","code":0,"status":"success"}
DBG|27|mqtt_client_publish.c|qcloud_iot_mqtt_publish(340): publish
packetID=0|topicName=$thing/up/property/0WUKPUCOTC/dev001|payload={"method":"report",
"clientToken":"0WUKPUCOTC-3", "params":{"power_switch":0,"color":0,"brightness":0,"name":"dev001"}}
```

步骤四：应用开发

详情请参见 软件包 中的示例程序进行开发。

SDK 使用参考

详情请参见 C SDK 使用参考 。

ESP8266 SDK 使用参考

最近更新时间：2022-06-10 15:14:27

qcloud-iot-esp-wifi 面向使用乐鑫 ESP Wi-Fi 芯片/模组（例如：ESP8266）来接入腾讯云物联网服务的开发者，包括使用腾讯云 IoT AT 指令 ESP8266 定制模组固件接入，以及使用 ESP8266 RTOS 平台进行 SoC 方式开发来接入腾讯云的用户，详情可参考 MCU+ 定制 MQTT AT模组（Wi-Fi 类）接入指引 和 MCU+ 通用 TCP AT 模组（FreeRTOS）接入指引 。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 ESP8266 SDK 。

移植指引

配网协议	参考文档
AirKiss	AirKiss 配网开发
SmartConfig	SmartConfig 配网开发
softAP	softAP 配网开发

直连设备开发
直连设备接入类型说明

最近更新时间：2022-09-02 10:11:16

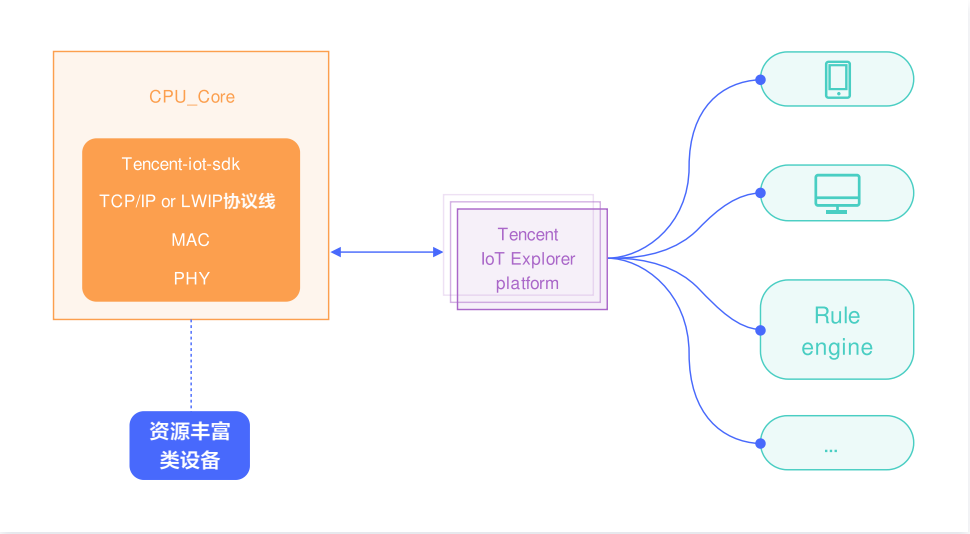
本文对接入腾讯云物联网开发平台 IoT Explorer 的直连设备接入类型进行说明，并介绍各直连设备类型如何接入 IoT Explorer 。

直连设备联网类型

直连设备指的是可以直接接入网络，访问物联网开发平台的设备。从 TCP/IP 协议栈承载的载体区分，直连设备分为资源丰富类设备和资源受限类设备。

资源丰富类设备

TCP/IP 协议栈运行在主芯片上，主芯片的处理能力和资源较丰富，例如：路由器，本文称作**资源丰富类设备**。



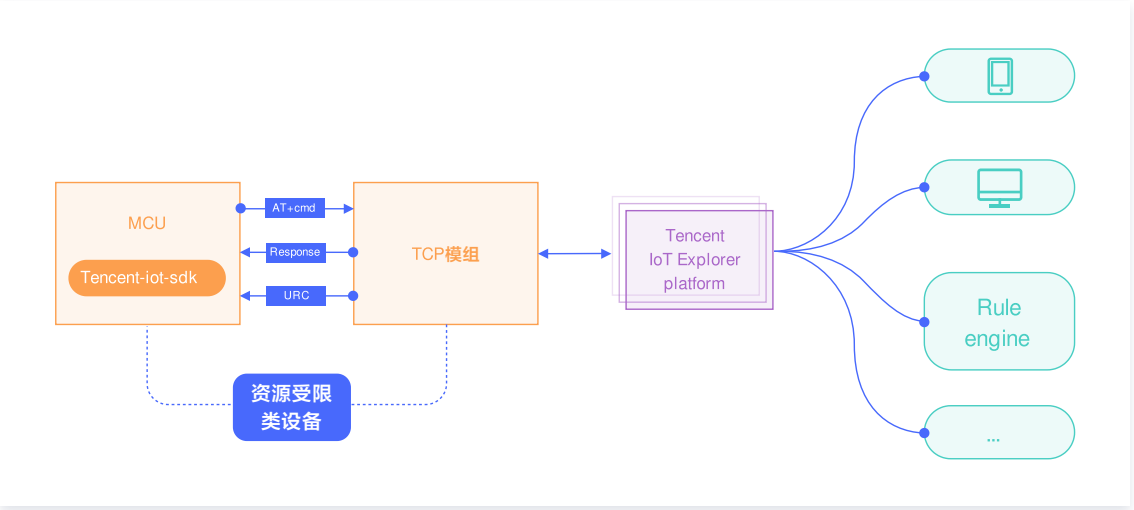
接入指引

资源丰富类设备根据开发平台可以分为以下5种，请参见相应指引文档：

开发平台	SDK	参考文档
Linux	qcloud-iot-explorer-sdk-embedded-c	Linux 平台接入指引
Windows	qcloud-iot-explorer-sdk-embedded-c	Windows 平台接入指引
Android	iot-device-java	Android 平台接入指引
Java	iot-device-java	Java 平台接入指引
FreeRTOS+lwIP	qcloud-iot-explorer-sdk-embedded-c	FreeRTOS+lwIP 平台接入指引
RT-Thread	qcloud-iot-explorer-sdk-embedded-c	RT-Thread 平台接入指引
其他平台	qcloud-iot-explorer-sdk-embedded-c	C SDK 移植接入指引

资源受限类设备

TCP/IP 协议栈运行在通信模组上，主芯片处理能力和资源有限，例如：STM32F103 系列，本文称作**资源受限类设备**。



接入指引

资源受限类设备，借助于通信模组实现网络访问，即 MCU + 模组方式。模组一般为蜂窝模组（2/3/4/5G）或者 Wi-Fi 模组，市场上可选的模组较多，AT 指令也各不相同，为此腾讯云物联网提供两种方式实现平台接入。

1. 基于 SDK 提供的 AT_Socket 框架和模组的通用 TCP 指令，参照 at_device 目录下已支持的模组，实现 AT_Device 驱动的结构体 at_device_op_t 对应的驱动接口即可，具体如下：

开发平台	SDK	文档
MCU+ 通用 AT 模组+FreeRTOS	qcloud-iot-explorer-sdk-embedded-c	MCU+ 通用 TCP AT 模组（FreeRTOS）接入指引
MCU+ 通用 AT 模组+nonOS	qcloud-iot-explorer-sdk-embedded-c	MCU+ 通用 TCP AT 模组（nonOS）接入指引

2. 腾讯云物联网与主流的模组厂商进行深度合作，将 SDK 的核心协议移植到模组中，模组对外封装统一的腾讯云物联网 AT 指令，并且为 MCU 提供实现和定制模组交互的 SDK。具体如下：

开发平台	SDK	文档
MCU+ 定制 AT 模组（蜂窝类）	qcloud-iot-sdk-tencent-at-based	MCU+ 定制 MQTT AT 模组（蜂窝类）接入指引
MCU+ 定制 AT 模组（Wi-Fi 类）	qcloud-iot-sdk-tencent-at-based	MCU+ 定制 MQTT AT 模组（Wi-Fi 类）接入指引

资源丰富类设备
Linux 平台接入指引

最近更新时间：2022-06-10 15:22:10

C SDK 已提供 Ubuntu Linux 基于 gcc 的适配，开发人员安装相应软件并根据指引快速接入腾讯云物联网开发平台。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

接入指引

Linux 平台接入腾讯云物联网开发平台可以分为以下3个步骤。

开发环境安装

> **说明：**
> 本文演示使用 Ubuntu 的版本为 v16.04。

SDK 需要 cmake 版本在 v3.5 以上，默认安装的 cmake 版本较低，若编译失败，请单击 下载 并参考 安装说明 进行 cmake 特定版本的下载与安装。

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

应用开发

可参考 SDK samples 目录下的例程进行开发。

编译与运行

1. 配置修改

修改 SDK 根目录下的 CMakeLists.txt 文件，并确保以下选项存在（以密钥认证设备为例）：

```
set(BUILD_TYPE                   "release")
set(COMPILE_TOOLS                "gcc")
set(PLATFORM                    "linux")
set(FEATURE_MQTT_COMM_ENABLED ON)
set(FEATURE_AUTH_MODE "KEY")
set(FEATURE_AUTH_WITH_NOTLS OFF)
set(FEATURE_DEBUG_DEV_INFO_USED  OFF)
```

2. 执行脚本编译

- 编译库和示例

```
./cmake_build.sh
```

- 只编译示例（完整编译后）

```
./cmake_build.sh samples
```

输出的库文件、头文件及示例在 `output/release` 文件夹中。

3. 填写设备信息

将在腾讯云物联网平台创建的设备的设备信息（以密钥认证设备为例），填写到 SDK 根目录下 device_info.json 中，示例代码如下：

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

```
"productId":"S3ExxxxxAZW",
"deviceName":"test_device",
"key_deviceinfo":{
    "deviceSecret":"vX6PQqa1****6f5SMfs6OA6y"
}
```

4. 运行示例

示例输出位于 `output/release/bin` 文件夹中，例如运行 data_template_sample 示例，输入 `./output/release/bin/data_template_sample` 即可。

SDK 使用参考

请参见 C SDK 使用参考 。

FreeRTOS+lwIP 平台接入指引

最近更新时间：2022-06-10 15:22:19

FreeRTOS 作为一个微内核系统，主要提供任务创建及调度和任务间通信等 OS 核心机制，在不同设备平台还需要搭配多个软件组件，包括 C 运行库（例如：newlib 或者 ARM CMSIS 库）和 TCP/IP 网络协议栈（例如：lwIP）才能形成完整的嵌入式运行平台。同时各个设备平台的编译开发环境也各不相同，因此在使用 C SDK 接入物联网开发平台时，需要根据不同设备的具体情况进行适配。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

> **说明：**
> SDK 在 **platform/os/freertos** 里提供了一个基于 **FreeRTOS+lwIP+newlib** 的参考实现，该实现已在乐鑫 ESP8266 平台上验证测试。

接入指引

FreeRTOS+lwip 平台接入腾讯云物联网开发平台可以分为以下两个步骤。

代码抽取

基于 RTOS 系统的平台编译方式各不相同，一般无法直接使用 SDK 的 cmake 或者 make 编译，因此 SDK 提供代码抽取功能，可根据需要将相关代码抽取到一个单独的文件夹，文件夹里面的代码层次目录简洁，方便用户拷贝集成到自己的开发环境。

1. 修改 CMakeLists.txt 中配置为 freertos 平台，并开启代码抽取功能：

```
set(BUILD_TYPE                  "release")
set(PLATFORM                    "freertos")
set(EXTRACT_SRC ON)
set(FEATURE_AT_TCP_ENABLED OFF)
```

2. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

3. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
    └── internal_inc
```

> **说明：**
> - include 目录：SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译宏。API 具体介绍请参考 接口及可变参数说明 。
> - platform 目录：平台相关的代码，可根据设备的具体情况进行修改适配。具体的函数说明请参考 C SDK 移植接入指引 。
> - sdk_src：SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 目录为 SDK 内部使用的头文件。

4. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

移植示例

在 Linux 开发环境基于乐鑫 ESP8266 RTOS 平台搭建一个工程示例。操作步骤如下：

1. 请参见 ESP8266_RTOS_SDK 获取 RTOS_SDK 和交叉编译器，并创建一个项目工程。
2. 将上一步骤抽取的 qcloud_iot_c_sdk 目录，拷贝到 components/qcloud_iot 下。
3. 在 components/qcloud_iot 下，新建一个编译配置文件 component.mk，内容如下：

```
#
# Component Makefile
#
COMPONENT_ADD_INCLUDEDIRS := \
qcloud_iot_c_sdk/include \
qcloud_iot_c_sdk/include/exports \
qcloud_iot_c_sdk/sdk_src/internal_inc
COMPONENT_SRCDIRS := \
qcloud_iot_c_sdk/sdk_src \
qcloud_iot_c_sdk/platform
```

至此，您可以将 qcloud_iot_c_sdk 作为一个组件进行编译了，之后在用户代码里面就可以调用 C SDK 的接口进行连接和收发消息。

SDK 使用参考

请参见 C SDK 使用参考 。

C SDK 移植接入指引

最近更新时间：2022-06-10 15:22:29

本文为您介绍如何将设备端 C SDK 移植到目标硬件平台。C SDK 采用模块化设计，分离核心协议服务与硬件抽象层，在进行跨平台移植时，一般只需对硬件抽象层进行修改适配即可。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

接入指引

非典型平台接入腾讯云物联网开发平台可以分为以下2个步骤。

硬件抽象层移植

HAL 层主要有几大块的移植，分别是 OS 相关、网络及 TLS 相关、时间及打印相关、设备信息相关。
SDK 在 platform/os 目录下示例了 Linux、Windows、FreeRTOS 及 nonOS 四个场景的硬件抽象层实现，详情可参考 直连设备接入类型说明 对应平台的接入指引。

OS 相关接口

序号	函数名	说明
1	HAL_Malloc	动态申请内存块。
2	HAL_Free	释放内存块。
3	HAL_ThreadCreate	线程创建。
4	HAL_ThreadDestroy	线程销毁。
5	HAL_MutexCreate	创建互斥锁。
6	HAL_MutexDestroy	销毁互斥锁。
7	HAL_MutexLock	mutex 加锁。
8	HAL_MutexUnlock	mutex 解锁。
9	HAL_SemaphoreCreate	创建信号量。
10	HAL_SemaphoreDestroy	销毁信号量。
11	HAL_SemaphoreWait	等待信号量。
12	HAL_SemaphorePost	释放信号量。
13	HAL_SleepMs	休眠。

网络及 TLS 相关的 HAL 接口

网络相关接口提供二选一的适配移植。对于具备网络通讯能力并且本身集成 TCP/IP 网络协议栈的设备，需要实现 POSIX_socket 的网络 HAL 接口，使用 TLS/SSL 加密通讯的还需要实现 TLS 相关的 HAL 接口；而对于 **MCU+ 通用 TCP_AT 模组** 的设备，则可以选择 SDK 提供的 AT_Socket 框架，并实现相关的 AT 模组接口。

- **基于 POSIX_socket 的 HAL 接口**

  其中 TCP/UDP 相关接口基于 POSIX socket 函数实现。TLS 相关接口依赖于 **mbedtls** 库，移植之前必须确保系统上有可用的 **mbedtls** 库。如果采用其他 TLS/SSL 库，可参考 platform/tls/mbedtls 相关实现进行移植适配。其中，UDP/DTLS 相关的函数仅在使能 **CoAP** 通讯的时候才需要移植。

序号	函数名	说明
1	HAL_TCP_Connect	建立 TCP 连接。
2	HAL_TCP_Disconnect	断开 TCP 连接。

3	HAL_TCP_Write	TCP 写。
4	HAL_TCP_Read	TCP 读。
5	HAL_TLS_Connect	建立 TLS 连接。
6	HAL_TLS_Disconnect	断开 TLS 连接。
7	HAL_TLS_Write	TLS 写。
8	HAL_TLS_Read	TLS 读。
9	HAL_UDP_Connect	建立 TCP 连接。
10	HAL_UDP_Disconnect	断开 TCP 连接。
11	HAL_UDP_Write	UDP 写。
12	HAL_UDP_Read	UPD 读。
13	HAL_DTLS_Connect	建立 DTLS 连接。
14	HAL_DTLS_Disconnect	断开 DTLS 连接。
15	HAL_DTLS_Write	DTLS 写。
16	HAL_DTLS_Read	DTLS 读。

- **基于 AT_socket 的 HAL 接口**

  通过使能编译宏 AT_TCP_ENABLED 选择 AT_socket，则 SDK 将会调用 `network_at_tcp.c` 的 `at_socket` 接口，at_socket 层不需要移植，需要实现 AT 串口驱动及 AT 模组驱动，AT 模组驱动只需要实现 AT 框架中 at_device 的驱动结构体 `at_device_op_t` 的驱动接口即可，详情可参考 at_device 目录下的已支持的模组。AT 串口驱动需要实现串口的中断接收，然后在中断服务程序中调用回调函数 `at_client_uart_rx_isr_cb` 即可，可以参考 `HAL_AT_UART_freertos.c` 实现目标平台的移植。

序号	函数名	说明
1	HAL_AT_Uart_Init	初始化 AT 串口
2	HAL_AT_Uart_Deinit	去初始化 AT 串口
3	HAL_AT_Uart_Send	AT 串口发送数据
4	HAL_AT_UART_IRQHandler	AT 串口接收中断服务程序

时间及打印相关的 HAL 接口

序号	函数名	说明
1	HAL_Printf	将格式化的数据写入标准输出流中。
2	HAL_Snprintf	将格式化的数据写入字符串。
3	HAL_UptimeMs	检索自系统启动以来已运行的毫秒数。
4	HAL_DelayMs	阻塞延时，单位毫秒。

设备信息相关的 HAL 接口

接入 IoT 平台需要在平台创建产品和设备信息，同时需要将产品及设备信息保存在设备侧的非易失存储介质。可以参考 platform/os/linux/HAL_Device_linux.c 示例实现。

序号	函数名	说明
1	HAL_GetDevInfo	设备信息读取。
2	HAL_SetDevInfo	设备信息保存。

应用开发

请参考 SDK samples 目录下的例程进行开发。

SDK 使用参考

请参考 C SDK 使用参考 。

Android 平台接入指引

最近更新时间：2024-08-26 14:55:51

Android SDK 依赖开源社区实现的 MQTT 协议 **eclipse.paho**，并在之上封装 数据模板协议 以便更好的接入腾讯云物联网开发平台。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 iot-device-java 。

接入指引

Android 平台接入腾讯云物联网开发平台可以分为以下两个步骤。

模块添加

在 App 的 build.gradle 中增加 eclipse.paho 的编译依赖：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "org.eclipse.paho:org.eclipse.paho.client.mqttv3"
}
```

应用开发

可根据 SDK 中提供的示例进行应用开发。

- 数据模板基本功能，可参见 `IoTDataTemplateFragment.java` 和 `DataTemplateSample.java` 。
- 直连设备接入，可参见 `IoTLightFragment.java` 和 `LightSample.java` 。
- 网关设备和子设备接入，可参见 `IoTGatewayFragment.java` 、 `GatewaySample.java` 、 `ProductLight.java` 和 `ProductAirconditioner.java` 。

SDK 使用参考

请参见 Android SDK 使用参考 。

Java 平台接入指引

最近更新时间：2022-06-10 15:22:47

Java SDK 依赖开源社区实现的 MQTT 协议 **eclipse.paho**，并在之上封装 数据模板协议 以便更好的接入腾讯云物联网开发平台。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 iot-device-java 。

接入指引

Java 平台接入腾讯云物联网开发平台可以分为以下两个步骤。

模块添加

在 App 的 build.gradle 中增加 eclipse.paho 的编译依赖：

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile "org.eclipse.paho:org.eclipse.paho.client.mqttv3"
}
```

应用开发

可根据 SDK 中提供的示例进行应用开发。

- 数据模板基本功能，可参见 `IoTDataTemplateFragment.java` 和 `DataTemplateSample.java` 。
- 直连设备接入，可参见 `IoTLightFragment.java` 和 `LightSample.java` 。
- 网关设备和子设备接入，可参见 `IoTGatewayFragment.java` 、 `GatewaySample.java` 、 `ProductLight.java` 和 `ProductAirconditioner.java` 。

SDK 使用参考

请参见 Java SDK 使用参考 。

Windows平台接入指引

最近更新时间：2022-06-10 15:22:58

C SDK 已提供 Windows 下基于 MSVC 的适配， 开发人员可以通过安装 Visual Studio 环境根据指引快速接入腾讯云物联网开发平台。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

接入指引

Windows 平台接入腾讯云物联网开发平台可以分为以下3个步骤。

开发环境安装

获取和安装 Visual Studio 2019开发环境，详情请参见 编译环境说明 。

应用开发

请参见 SDK samples 目录下的例程进行开发。

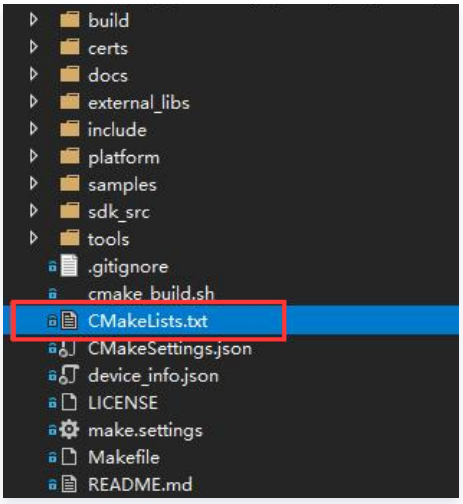
编译并运行

1. 运行 Visual Studio，单击**打开本地文件夹**，并选择下载的 C SDK 目录。

2. 将在腾讯云物联网通信控制台创建的设备的设备信息（以密钥认证设备为例），填写到 device_info.json 中，示例代码如下：

```
"auth_mode":"KEY",
"productId":"S3*****AZW",
"deviceName":"test_device",
"key_deviceinfo":{
    "deviceSecret":"vX6PQqa12****f5SMfs6OA6y"
}
```

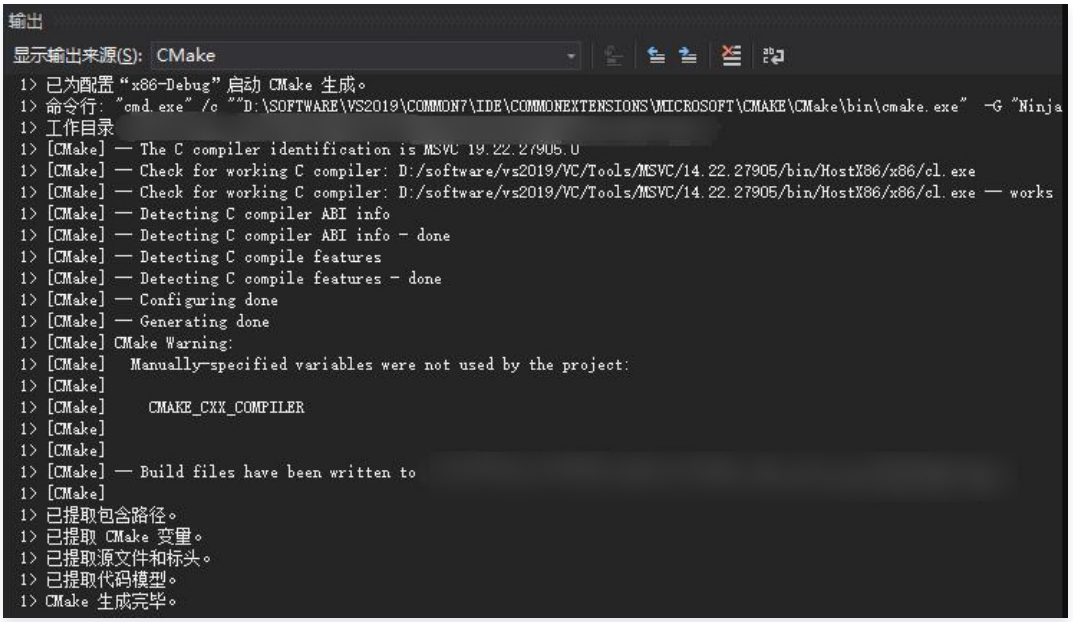
3. 双击打开根目录的 CMakeLists.txt，并确认编译工具链中设置的平台为 **Windows** 和编译工具为 **MSVC**。



示例代码如下：

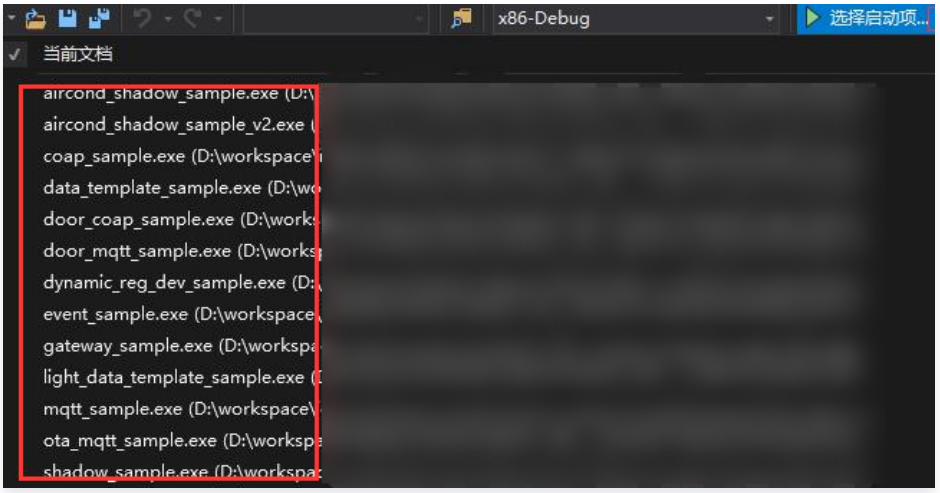
```
# 编译工具链
#set(COMPILE_TOOLS "gcc")
#set(PLATFORM   "linux")

set(COMPILE_TOOLS "MSVC")
set(PLATFORM   "windows")
```

4. Visual Studio 会自动生成 cmake 缓存，请等待 cmake 缓存生成完毕。



5. 缓存生成完毕后，单击**生成 > 全部生成**。

6. 选择相应的示例运行，示例应与用户信息相对应。



SDK 使用参考

请参见 C SDK 使用参考 。

资源受限类设备
MCU+ 定制 MQTT AT 模组（Wi-Fi 类）接入指引

最近更新时间：2024-08-26 14:55:51

腾讯云物联网与主流的模组厂商进行深度合作，将 SDK 的核心协议移植到 Wi-Fi 模组中，模组对外封装统一的腾讯云物联网 AT 指令，并提供配合使用的 AT SDK。

SDK 获取

在 IoT explorer 平台 创建产品和设备 后，选择基于 MQTT AT 定制模组开发的方式，将会自动生成 MCU 侧的 AT SDK 代码，并且把在平台创建的数据模板和事件生成了对应的配置及初始化代码。

接入指引

MCU+ 定制 MQTT AT 模组（Wi-Fi 类）接入腾讯云物联网开发平台可以分为以下四个步骤。

平台适配移植

根据所选的嵌入式平台，适配 hal_export.h 头文件对应的 HAL 层 API 的移植实现。主要有串口收发（中断接收）、模组开关机、任务/线程创建、动态内存申请/释放、时延、打印等 API。详细操作可参考基于 STM32+FreeRTOS 的 AT-SDK 移植示例。
移植部分需要实现的 HAL 层适配接口请参考 hal_export.h，需要实现串口的收发接口（中断接收），延时函数，模组上下电及 OS 相关接口适配（互斥锁、动态内存申请释放、线程创建），适配层接口单独剥离在 port 目录。

- **hal_export.h**

  该源文件主要提供 HAL 层对外的 API 接口以及线程栈大小设置。

序号	宏定义	说明
1	PARSE_THREAD_STACK_SIZE	串口 AT 解析线程栈大小。

- **config.h**

  该源文件主要提供相关宏开关控制。

序号	宏定义	说明
1	OS_USED	是否使用 OS。目前的 AT-SDK 是基于多线程框架的，所以 OS 是必须的。
2	AUTH_MODE_KEY	认证方式。证书认证或者密钥认证。
3	DEBUG_DEV_INFO_USED	默认使能该宏，设备信息使用调试信息，正式量产关闭该宏，并实现设备信息存取接口。

- **hal_at.c**

  该源文件主要实现 AT 串口初始化、串口收发、模组开关机。

序号	HAL_API	说明
1	module_power_on	模组开机，AT 串口初始化，必选实现。
1	module_power_off	模组关机，低功耗需要，可选实现。
2	AT_UART_IRQHandler	串口接收中断 ISR，将收取到的数据放入 ringbuff 中，AT 解析线程会实时解析数据，必选实现。
3	at_send_data	AT 串口发送接口。

- **module_api_inf.c**

  该源文件主要实现配网/注网 API 业务适配和基于腾讯云物联网 AT 指令的 MQTT 交互，但有一个关于联网/注网的 API（module_register_network）需要根据模组适配。代码基于 ESP8266 腾讯云物联网定制 AT 固件 实现 Wi-Fi 直连的方式连接网络，但更常用的场景是根据特定事件（例如：按键）触发配网（softAP/一键配网），这块的逻辑各具体业务逻辑需自行实现。

ESP8266 有封装配网指令和示例 App。对于蜂窝模组，则是使用特定的网络注册指令。请参考 module_handshake 函数应用 AT-SDK 的 AT 框架添加和模组的 AT 指令交互。

```
//模组联网（NB/2/3/4G注册网络）、wifi配网（一键配网/softAP）暂时很难统一，需要用户根据具体模组适配。
//开发者参照 module_handshake API使用AT框架的API和模组交互，实现适配。
eAtResault module_register_network(eModuleType eType)
{
    eAtResault result = AT_ERR_SUCCESS;
#ifdef MODULE_TYPE_WIFI
    if(eType == eMODULE_ESP8266)
    {
        #define WIFI_SSID "test_****"
        #define WIFI_PW "********"

        /*此处示例传递热点名字直接联网，通常的做法是特定产品根据特定的事件（譬如按键）触发wifi配网（一键配网/softAP）*/
        result = wifi_connect(WIFI_SSID, WIFI_PW);
        if(AT_ERR_SUCCESS != result)
        {
            Log_e("wifi connect fail,ret:%d", result);
        }
    }
#else
    if(eType == eMODULE_N21)
    {
        /*模组网络注册、或者wifi配网需要用户根据所选模组实现*/
        result = N21_net_reg();
        if(AT_ERR_SUCCESS != result)
        {
           Log_e("N21 register network fail,ret:%d", result);
        }
    }
#endif

     return result;
}
```

设备信息修改

调试时，在 hal_export.h 将设备信息调试宏定义打开。量产时需要关闭该宏定义，实现 hal-os 中序列 17-26 的设备信息存取 API。

```
#define DEBUG_DEV_INFO_USED
```

修改下面宏定义的设备信息，则系统将会使用调试信息。

```
#ifdef DEBUG_DEV_INFO_USED

static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1] = "03UKN1****";
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1] = "at****";

#ifdef AUTH_MODE_CERT
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME + 1]       =
"YOUR_DEVICE_NAME_cert.crt";
static char sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_KEY_FILE_NAME + 1] =
"YOUR_DEVICE_NAME_private.key";
#else
char sg_device_secret[MAX_SIZE_OF_DEVICE_SERC + 1] = "ttOARy0PjYgzd9OSs1****==";
#endif
```

```
#endif
```

上下行业务逻辑开发

自动生成的代码 data_template_usr_logic.c，已实现数据、事件收发及响应的通用处理逻辑。但是具体数据处理的业务逻辑需要用户自己根据业务逻辑添加，上下行业务逻辑添加的入口函数分别为 `deal_up_stream_user_logic` 、 `deal_down_stream_user_logic` 。

- **下行业务逻辑实现**

```
/*用户需要实现的下行数据的业务逻辑，pData除字符串变量已实现用户定义的所有其他变量值解析赋值，待用户实现业务逻辑*/
static void deal_down_stream_user_logic(ProductDataDefine   * pData)
{
    Log_d("someting about your own product logic wait to be done");
}
```

- **上行业务逻辑实现**

```
/*用户需要实现的上行数据的业务逻辑，此处仅供示例*/
static int deal_up_stream_user_logic(DeviceProperty *pReportDataList[], int *pCount)
{
    int i, j;

     for (i = 0, j = 0; i < TOTAL_PROPERTY_COUNT; i++) {
        if(eCHANGED == sg_DataTemplate[i].state) {
            pReportDataList[j++] = &(sg_DataTemplate[i].data_property);
            sg_DataTemplate[i].state = eNOCHANGE;
        }
    }
    *pCount = j;

    return (*pCount > 0)?AT_ERR_SUCCESS:AT_ERR_FAILURE;
}
```

应用开发

Sample 目录一共有3个示例，用户可以参考各示例根据业务逻辑进行应用开发，分别是 mqtt_sample.c、shadow_sample.c、light_data_template_sample.c。
各示例说明如下：

序号	示例名称	说明
1	mqtt_sample.c	MQTT 示例，该示例介绍了基于定制的 AT 指令如何便捷地接入腾讯物联网平台及收发数据。
2	shadow_sample.c	影子示例，基于 AT 实现的 MQTT 协议，进一步封装的影子协议。
3	light_data_template_sample.c	基于智能灯的控制场景，介绍具体的产品如何应用数据模板及事件功能。

更多详情请参考 数据模板协议 。

SDK 使用参考

请参考 AT SDK 使用参考 。

MCU+ 定制 MQTT AT 模组（蜂窝类）接入指引

最近更新时间：2024-09-30 17:54:51

腾讯云物联网与主流的模组厂商进行深度合作，将 SDK 的核心协议移植到蜂窝模组（2/3/4/5G）中，模组对外封装统一的腾讯云物联网 AT 指令，并提供配合使用的 AT SDK 。

SDK 获取

在 IoT explorer 平台 创建产品和设备 后，选择基于 MQTT AT 定制模组开发的方式，将会自动生成 MCU 侧的 AT SDK 代码，并且把在平台创建的数据模板和事件生成了对应的配置及初始化代码。

接入指引

MCU+ 定制 MQTT AT 模组（蜂窝类）接入腾讯云物联网开发平台可以分为以下4个步骤。

平台适配移植

根据所选的嵌入式平台，适配 hal_export.h 头文件对应的 HAL 层 API 的移植实现。主要有串口收发（中断接收）、模组开关机、任务/线程创建、动态内存申请/释放、时延、打印等 API。详细操作可参考基于 STM32+FreeRTOS 的 AT-SDK 移植示例 。
移植部分需要实现的 HAL 层适配接口请参考 hal_export.h，需要实现串口的收发接口（中断接收），延时函数，模组上下电及 OS 相关接口适配（互斥锁、动态内存申请释放和线程创建），适配层接口单独剥离在 port 目录。

- **hal_export.h**

  该源文件主要提供 HAL 层对外的 API 接口及 HAL 层宏开关控制。

序号	宏定义	说明
1	PARSE_THREAD_STACK_SIZE	串口 AT 解析线程栈大小。
2	OS_USED	是否使用 OS。目前的 AT-SDK 是基于多线程框架的，所以 OS 是必须的。
3	AUTH_MODE_KEY	认证方式。证书认证或者密钥认证。
4	DEBUG_DEV_INFO_USED	默认使能该宏，设备信息使用调试信息，正式量产关闭该宏，并实现设备信息存取接口。

- **hal_os.c**

  该源文件主要实现打印、延时、时间戳、锁、线程创建、设备信息存取等。

序号	HAL_API	说明
1	HAL_Printf	打印函数，log 输出需要，可选实现。
2	HAL_Snprintf	格式化打印，JSON 数据处理需要，必须实现。
3	HAL_Vsnprintf	格式化输出， 可选实现。
4	HAL_DelayMs	毫秒延时，必选实现。
5	HAL_DelayUs	微妙延时，可选实现。
6	HAL_GetTimeMs	获取毫秒数，必选实现。
7	HAL_GetTimeSeconds	获取时间戳，必选实现，时间戳不需要绝对准确，但不可重复。
8	hal_thread_create	线程创建，必选实现。
9	hal_thread_destroy	线程销毁，必选实现。
10	HAL_SleepMs	放权延时，必选实现。
11	HAL_MutexCreate	互斥锁创建，必选实现。
12	HAL_MutexDestroy	互斥锁销毁，必选实现。

13	HAL_MutexLock	获取互斥锁，必选实现。
14	HAL_MutexUnlock	释放互斥锁，必选实现。
15	HAL_Malloc	动态内存申请，必选实现。
16	HAL_Free	动态内存释放，必选实现。
17	HAL_GetProductID	获取产品 ID，必选实现。
18	HAL_SetProductID	设置产品 ID，必须存放在非易失性存储介质，必选实现。
19	HAL_GetDevName	获取设备名，必选实现。
20	HAL_SetDevName	设置设备名，必须存放在非易失性存储介质，必选实现。
21	HAL_GetDevSec	获取设备密钥，密钥认证方式为必选实现。
22	HAL_SetDevSec	设置设备密钥，必须存放在非易失性存储介质，密钥认证方式为必选实现。
23	HAL_GetDevCertName	获取设备证书文件名，证书认证方式为必选实现。
24	HAL_SetDevCertName	设置设备证书文件名，必须存放在非易失性存储介质，证书认证方式为必选实现。
25	HAL_GetDevPrivateKeyName	获取设备证书私钥文件名，证书认证方式为必选实现。
26	HAL_SetDevPrivateKeyName	设置设备证书私钥文件名，必须存放在非易失性存储介质，证书认证方式为必选实现。

- **hal_at.c**

  该源文件主要实现 AT 串口初始化、串口收发、模组开关机。

序号	HAL_API	说明
1	module_power_on	模组开机，AT 串口初始化，必选实现。
1	module_power_off	模组关机，低功耗需要，可选实现。
2	AT_UART_IRQHandler	串口接收中断 ISR，将收取到的数据放入 ringbuff 中，AT 解析线程会实时解析数据，必选实现。
3	at_send_data	AT 串口发送接口。

- **module_api_inf.c**

  该源文件主要实现配网/注网 API 业务适配和基于腾讯云物联网 AT 指令的 MQTT 交互，但有一个关于联网/注网的API（module_register_network）需要根据模组适配。代码基于 ESP8266 腾讯云物联网定制 AT 固件 实现 Wi-Fi 直连的方式连接网络，但更常用的场景是根据特定事件（例如按键）触发配网（softAP/一键配网），这块的逻辑各具体业务逻辑需自行实现。
  ESP8266 有封装配网指令和示例 App。对于蜂窝模组，则是使用特定的网络注册指令。请参考 module_handshake 函数应用 AT-SDK 的 AT 框架添加和模组的 AT 指令交互。

```
//模组联网（NB/2/3/4G注册网络）、wifi配网（一键配网/softAP）暂时很难统一，需要用户根据具体模组适配。
//开发者参照 module_handshake API使用AT框架的API和模组交互，实现适配。
eAtResault module_register_network(eModuleType eType)
{
    eAtResault result = AT_ERR_SUCCESS;
#ifdef MODULE_TYPE_WIFI
    if(eType == eMODULE_ESP8266)
    {
        #define WIFI_SSID "test_****"
        #define WIFI_PW "********"

        /*此处示例传递热点名字直接联网，通常的做法是特定产品根据特定的事件（例如按键）触发wifi配网（一键配网/softAP）*/
        result = wifi_connect(WIFI_SSID, WIFI_PW);
```

```
        if(AT_ERR_SUCCESS != result)
        {
            Log_e("wifi connect fail,ret:%d", result);
        }
    }
#else
    if(eType == eMODULE_N21)
    {
        /*模组网络注册、或者wifi配网需要用户根据所选模组实现*/
        result = N21_net_reg();
        if(AT_ERR_SUCCESS != result)
        {
           Log_e("N21 register network fail,ret:%d", result);
        }
    }
#endif

     return result;
}
```

设备信息修改

调试时，在 hal_export.h 将设备信息调试宏定义打开。量产时需要关闭该宏定义，实现 hal-os 中序列 17-26 的设备信息存取 API。

```
#define DEBUG_DEV_INFO_USED
```

修改下面宏定义的设备信息，则系统将会使用调试信息。

```
#ifdef DEBUG_DEV_INFO_USED

static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1] = "03UKN1****";
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1] = "at****";

#ifdef AUTH_MODE_CERT
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME + 1]      =
"YOUR_DEVICE_NAME_cert.crt";
static char sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_KEY_FILE_NAME + 1] =
"YOUR_DEVICE_NAME_private.key";
#else
char sg_device_secret[MAX_SIZE_OF_DEVICE_SERC + 1] = "ttOARy0PjYgzd9OSs1****==";
#endif

#endif
```

上下行业务逻辑开发

自动生成的代码 data_template_usr_logic.c，已实现数据、事件收发及响应的通用处理逻辑。但是具体的数据处理的业务逻辑需要用户自己根据业务逻辑添加，上下行业务逻辑添加的入口函数分别为 `deal_up_stream_user_logic` 、 `deal_down_stream_user_logic` 。

- 下行业务逻辑实现

```
/*用户需要实现的下行数据的业务逻辑,pData除字符串变量已实现用户定义的所有其他变量值解析赋值，待用户实现业务逻辑*/
static void deal_down_stream_user_logic(ProductDataDefine   * pData)
{
  Log_d("someting about your own product logic wait to be done");
}
```

- 上行业务逻辑实现

```
/*用户需要实现的上行数据的业务逻辑，此处仅供示例*/
static int deal_up_stream_user_logic(DeviceProperty *pReportDataList[], int *pCount)
{
  int i, j;

   for (i = 0, j = 0; i < TOTAL_PROPERTY_COUNT; i++) {
      if(eCHANGED == sg_DataTemplate[i].state) {
          pReportDataList[j++] = &(sg_DataTemplate[i].data_property);
          sg_DataTemplate[i].state = eNOCHANGE;
      }
  }
  *pCount = j;

  return (*pCount > 0)?AT_ERR_SUCCESS:AT_ERR_FAILURE;
}
```

应用开发

Sample 目录一共有3个示例，用户可以参考各示例根据业务逻辑进行应用开发，分别是 mqtt_sample.c、shadow_sample.c、light_data_template_sample.c。
各示例说明如下：

序号	示例名称	说明
1	mqtt_sample.c	MQTT 示例，该示例介绍了基于定制的 AT 指令如何便捷地接入腾讯物联网平台及收发数据。
2	shadow_sample.c	影子示例，基于 AT 实现的 MQTT 协议，进一步封装的影子协议。
3	light_data_template_sample.c	基于智能灯的控制场景，介绍具体的产品如何应用数据模板及事件功能。

更多详情请参考 数据模板协议 。

SDK 使用参考

请参考 AT SDK 使用参考 。

MCU+ 通用 TCP AT 模组（FreeRTOS）接入指引

最近更新时间：2022-06-10 15:23:27

对于不具备网络通讯能力的 MCU，一般采用 MCU+ 通讯模组的方式，通讯模组（包括 Wi-Fi/2G/4G/NB-IoT）一般提供基于串口的 AT 指令协议供 MCU 进行网络通讯。针对这种场景，C SDK 封装了 AT-socket 网络层，网络层之上的核心协议和服务层无须移植。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

接入指引

MCU+ 通用 TCP AT 模组（FreeRTOS）接入腾讯云物联网开发平台可以分为以下四个步骤。

SDK 功能配置

使用通用 TCP 模组编译配置选项配置如下：

名称	配置	说明
BUILD_TYPE	debug/release	根据需要设置。
EXTRACT_SRC	ON	使能代码抽取。
COMPILE_TOOLS	gcc/MSVC	根据需要设置，IDE 情况不关注。
PLATFORM	Linux/Windows	根据需要设置，IDE 情况不关注。
FEATURE_OTA_COMM_ENABLED	ON/OFF	根据需要设置。
FEATURE_AUTH_MODE	KEY	资源受限设备认证方式建议选密钥认证。
FEATURE_AUTH_WITH_NOTLS	ON/OFF	根据需要是否使能 TLS。
FEATURE_EVENT_POST_ENABLED	ON/OFF	根据需要是否使能事件上报。
FEATURE_AT_TCP_ENABLED	ON	AT 模组 TCP 功能开关。
FEATURE_AT_UART_RECV_IRQ	ON	AT 模组中断接受功能开关。
FEATURE_AT_OS_USED	ON	AT 模组多线程功能开关。
FEATURE_AT_DEBUG	OFF	默认关闭 AT 模组调试功能，有调试需要再打开。

代码抽取

1. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

2. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
    └── internal_inc
```

> **说明：**
> - include 目录：SDK 供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译。
> - platform 目录：平台相关的代码，可根据设备的具体情况进行修改适配。
> - sdk_src ： SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 为 SDK 内部使用的头文件。

3. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

HAL 层移植

请先参见 C SDK 移植接入指引 进行移植。
对于网络相关的 HAL 接口，通过本文编译选项已选择 SDK 提供的 `AT_Socket` 框架，SDK 将会调用 `network_at_tcp.c` 的 `AT_socket` 接口，`AT_socket` 层不需要移植，需要实现 AT 串口驱动及AT模组驱动，AT模组驱动只需要实现 AT 框架中 `AT_device` 的驱动结构体 `AT_device_op_t` 的驱动接口即可，可以参照 `AT_device` 目录下的已支持的模组。

> **说明：**
> 目前 SDK 针对物联网使用较广的 Wi-Fi 模组 ESP8266 提供了底层接口实现，供移植到其他通讯模组时作为参考。

业务逻辑开发

可参见 SDK samples 目录下的例程进行开发。

SDK 使用参考

请参见 C SDK 使用参考 。

MCU+ 通用 TCP AT 模组（nonOS）接入指引

最近更新时间：2023-05-22 15:15:01

对于不具备网络通讯能力的 MCU，一般采用 MCU+ 通讯模组的方式，通讯模组（包括 Wi-Fi/2G/4G/NB-IoT）一般提供基于串口的 AT 指令协议供 MCU 进行网络通讯。针对这种场景，C SDK 封装 AT-socket 网络层，网络层之上的核心协议和服务层无需移植。
相较于有 RTOS 场景，AT-socket 网络接收数据的处理会有差异，应用层需要周期性的调用 IOT_MQTT_Yield 来接收服务端下行数据，错过接收窗口则会存在数据丢失的情况，所以在业务逻辑较为复杂的场景建议使用 RTOS，通过配置 `FEATURE_AT_OS_USED = OFF` 选择无 OS 方式。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 C SDK 。

接入指引

MCU+ 通用 TCP AT 模组（nonOS）接入腾讯云物联网开发平台可以分为以下4个步骤。

SDK 功能配置

无 RTOS 使用通用 TCP 模组编译配置选项配置如下：

名称	配置	说明
BUILD_TYPE	debug/release	根据需要设置。
EXTRACT_SRC	ON	使能代码抽取。
COMPILE_TOOLS	gcc/MSVC	根据需要设置，IDE 情况不关注。
PLATFORM	Linux/Windows	根据需要设置，IDE 情况不关注。
FEATURE_OTA_COMM_ENABLED	ON/OFF	根据需要设置。
FEATURE_AUTH_MODE	KEY	资源受限设备认证方式建议选密钥认证。
FEATURE_AUTH_WITH_NOTLS	ON/OFF	根据需要是否使能 TLS。
FEATURE_EVENT_POST_ENABLED	ON/OFF	根据需要是否使能事件上报。
FEATURE_AT_TCP_ENABLED	ON	使能 AT-socket 组件。
FEATURE_AT_UART_RECV_IRQ	ON	使能 AT 串口中断接收。
FEATURE_AT_OS_USED	OFF	AT-socket 组件无 RTOS 环境使用。
FEATURE_AT_DEBUG	OFF	默认关闭 AT 模组调试功能，有调试需要再打开。

代码抽取

1. 在 Linux 环境运行以下命令：

```
mkdir build
cd build
cmake ..
```

2. 即可在 output/qcloud_iot_c_sdk 中，找到相关代码文件，目录层次如下：

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
```

```
└── internal_inc
```

说明：

- include 目录：SDK 提供用户使用的 API 及可变参数，其中 config.h 为根据编译选项生成的编译宏。
- platform 目录：平台相关的代码，可根据设备的具体情况进行修改适配。
- sdk_src ： SDK 的核心逻辑及协议相关代码，一般不需要修改，其中 internal_inc 为 SDK 内部使用的头文件。

3. 用户可将 qcloud_iot_c_sdk 拷贝到其目标平台的编译开发环境，并根据具体情况修改编译选项。

HAL 层移植

请先参见 C SDK 移植接入指引 进行移植。
对于网络相关的 HAL 接口，通过本文编译选项已选择 SDK 提供的 AT_Socket 框架，SDK 会调用 `network_at_tcp.c` 的 `AT-socket` 接口，`AT-socket` 层不需要移植，需要实现 AT 串口驱动及 AT 模组驱动，AT 模组驱动只需要实现 AT 框架中 `AT_device` 的驱动结构体 `AT_device_op_t` 的驱动接口即可，可以参照 `AT_device` 目录下的已支持的模组。AT 串口驱动需要实现串口的中断接收，然后在中断服务程序中调用回调函数 `AT_client_uart_rx_isr_cb` 即可，可以参考 `HAL_OS_nonos.c` 实现目标平台的移植。

业务逻辑开发

可参考 SDK samples 目录下的例程进行开发。

SDK 使用参考

请参见 C SDK 使用参考 。

蓝牙设备开发
LLSync SDK 接入指引

最近更新时间：2022-06-10 15:23:45

本文为您介绍如何将 LLSync SDK 移植到目标硬件平台。LLSync SDK 采用模块化设计，分离 LLSync 核心组件与硬件抽象层，在进行跨平台移植时，一般只需要您对硬件抽象层进行适配即可。

SDK 获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 LLSync SDK 。

接入指引

LLSync SDK 现已支持标准蓝牙功能和辅助配网功能。

- 标准蓝牙功能：主要用于单 BLE 芯片通过腾讯连连小程序和腾讯云物联网开发平台进行通信。
- 辅助配网功能：主要用于通过 BLE 给同时具有 `BLE + Wi-Fi` 能力的设备配置网络。

您可以根据需求选择使用 LLSync SDK 的不同能力，详情请参见 标准蓝牙功能详细接入指引 和 辅助配网功能详细接入指引 。
同时，腾讯云也提供了标准蓝牙功能和辅助配网功能接入的 示例程序 供您参考。

LLSync 协议

详情请参见 LLSync 协议说明 。

SDK 使用参考

详情请参见 LLSync SDK 使用参考 。

LLSync SDK 使用参考

最近更新时间：2022-06-29 16:11:06

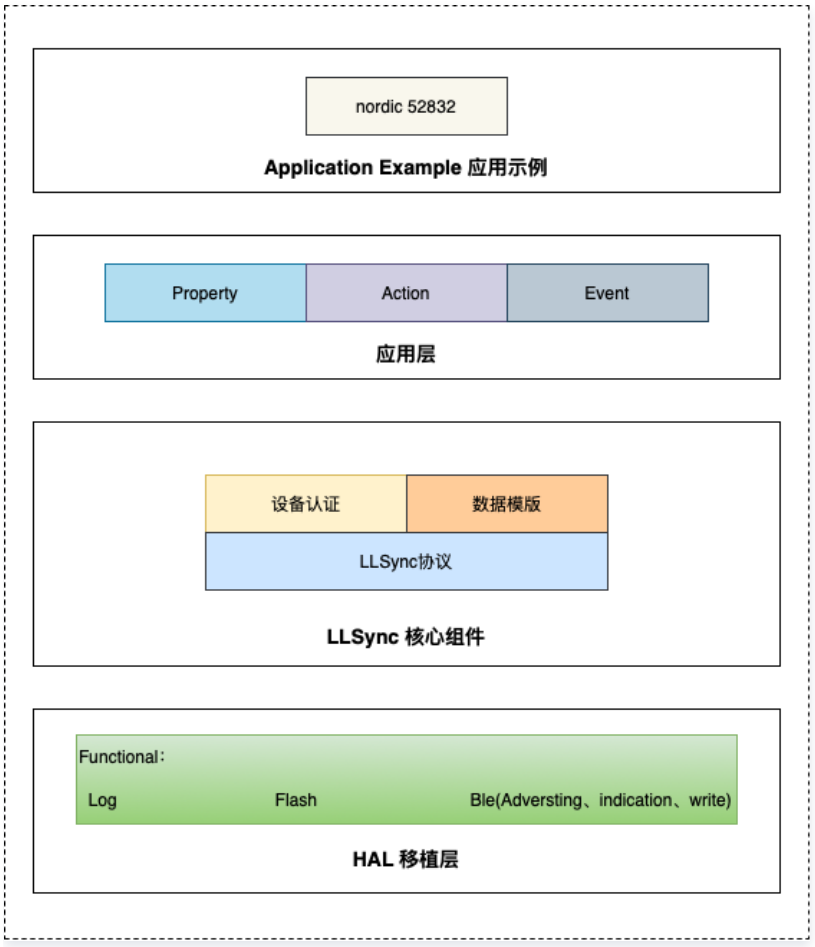
LLSync SDK 提供 LLSync 协议接入方案，打通了 **BLE 设备-App-物联网开发平台** 或 **BLE 设备-网关设备-物联网开发平台** 的数据链路，方便用户将 BLE 设备快速接入物联网开发平台。用户可以在物联网开发平台创建产品，通过云 API 下发数据模板操作来控制 BLE 设备。

SDK获取

SDK 使用 Github 托管，可访问 Github 下载最新版本设备端 LLSync SDK 。

软件架构

BLE 设备可以通过 LLSync 协议与移动端 App 或网关设备连接，通过 App 或网关设备接入物联网开发平台。LLSync SDK 结构框图见下图：



SDK 分三层设计，从上至下分别为应用层、LLSync 核心层、HAL 移植层。

- HAL 移植层：LLSync SDK 需要适配设备硬件和 BLE 协议栈，针对不同的设备和 BLE 协议栈用户需要进行移植和适配。
- LLSync 核心组件：定义了 BLE 设备和移动端 App 或网关设备之间的通信协议，实现身份认证，数据解析等功能，用户一般无需改动即可使用。
- 应用层：LLSync SDK 提供数据模板的操作函数，用户需要根据使用场景做具体实现。

目录结构

```
qcloud_iot_explorer_ble
  ├─config                          # 配置文件
  ├─docs                            # 协议文档
  ├─inc                             # 外部头文件
  ├─scripts                         # 脚本目录
  |   ├─interpret_json_dt           # json 文本转换 C 代码
  |       ├─config                  # ini 文件
  |       ├─src                     # 脚本文件
```

```
│           ├─dt_fixed_content      # C 代码中的固定内容，无需转换生成
└─src                               # LLSync 核心代码，用户一般无需修改
    ├─core                          # LLSync 源码
    ├─internal_inc                  # 内部头文件
    └─utils                         # 通用组件
```

移植指引

请参见 LLSync SDK 接入指引 。

设备端 OTA 功能开发指导

最近更新时间：2023-09-15 14:30:02

本文为您介绍如何配置 OTA 功能、上传固件以及在小程序进行验证等操作。

配置 OTA 功能

1. 打开您工程下的 `ble_qiot_config.h` 文件，找到 `BLE_QIOT_SUPPORT_OTA` 宏，设置为1开启 OTA 功能。

```
#define BLE_QIOT_SUPPORT_OTA 1
```

2. 修改当前软件的版本号，内容格式不限，用户可根据自己的需求任意设置，长度在1至32字节以内，不得包含除 `a-zA-Z0-9.-_` 以外的字符，固件版本号必须与控制台保持一致，详情可参见 固件升级 。

```
#define BLE_QIOT_USER_DEVELOPER_VERSION "0.0.1"
```

3. 您可以按需要设置是否开启断点续传功能， `BLE_QIOT_SUPPORT_RESUMING` 设置为1开启断点续传功能，如使用断点续传功能请同时设置 `BLE_QIOT_OTA_INFO_FLASH_ADDR` ，该地址用于在 flash 中保存断点续传信息，至少需要64字节。

```
#define BLE_QIOT_SUPPORT_RESUMING 1
#if (1 == BLE_QIOT_SUPPORT_RESUMING)
#define BLE_QIOT_OTA_INFO_FLASH_ADDR (BLE_QIOT_RECORD_FLASH_ADDR + 0x1000)
#endif
```

4. OTA 升级过程中小程序连续下发最大数据包数量，不得超过255字节，可根据实际情况修改。

```
#define BLE_QIOT_TOTAL_PACKAGES 0xFF
```

5. 每个 OTA 数据包长度，用户可根据设备所支持的 MTU 大小进行修改，但不得超过 "mtu - 3"，适当加大该数值一定程度上提升 OTA 升级的速度。

```
#define BLE_QIOT_PACKAGE_LENGTH 0x10
```

6. 两个数据包之间超时间隔，单位为秒，用户可根据实际情况进行修改。

```
#define BLE_QIOT_RETRY_TIMEOUT  2
```

7. OTA 升级重启等待时间，单位为秒，在此时间内小程序将持续尝试连接设备，超时后小程序认为升级失败，如果固件较大或设备升级流程较长可适当加长重启等待时间。

```
#define BLE_QIOT_REBOOT_TIME    20
```

8. 小程序下发两个数据包之间的时间间隔，单位为毫秒，可适当修改以提升 OTA 升级速度。

```
#define BLE_QIOT_PACKAGE_INTERVAL 0x05
```

9. 小程序发送的 OTA 数据将暂存在这个 buffer 中，buffer 满后会写入 flash 指定地址，用户可根据设备 RAM 空间进行修改，建议与 flash 的 page 大小相等或为其整数倍。

```
#define BLE_QIOT_OTA_BUF_SIZE (4096)
```

OTA 相关接口适配

1. 查看 `qcloud_iot_explorer_ble\inc\ble_qiot_import.h` 文件，用户需要实现以下几个接口。

- SDK 调用该接口将版本号传给用户，用户可以按照自己的规则检查版本号是否合法、是否高于上一个版本等，通过返回值通知 SDK 是否允许 OTA 升级。

```
uint8_t ble_ota_is_enable(const char *version);
```

- SDK 通过此接口获取 OTA 数据在 flash 中的保存地址，用户只需提供基址，在 OTA 升级过程中 SDK 会自动在基址的基础上计算偏移。

```
uint32_t ble_ota_get_download_addr(void);
```

- 用户提供写入 OTA 数据的接口，存储介质不限，可以是片内 ROM 或片外 flash 等。SDK 只负责写入数据，需用户自己在适当的时间进行擦除、磨损平衡等处理。

```
int ble_ota_write_flash(uint32_t flash_addr, const char *write_buf, uint16_t write_len);
```

2. 查看 `qcloud_iot_explorer_ble\inc\ble_qiot_export.h` 文件，用户需要实现以下几个回调函数。
   - 用于通知用户 OTA 升级开始。

```
typedef void (*ble_ota_start_callback)(void);
```

   - 用于通知用户 OTA 升级结束并返回结果，例如成功、CRC 校验错误、文件错误等。用户可以根据返回结果做进一步处理，例如返回成功则重启设备，通过 boot 程序将保存在外部 flash 的 OTA 固件写入片内进行升级；返回错误则擦除已经下载的 OTA 固件等。

```
typedef void (*ble_ota_stop_callback)(uint8_t result);
```

   - SDK 在接收完 OTA 数据并通过 CRC 校验后会调用该函数，用户可以在该函数内对固件做进一步校验，例如固件是否正确、是否为定制固件等，最终 OTA 升级的结果会通过 `ble_ota_stop_callback()` 告知用户。

```
typedef ble_qiot_ret_status_t (*ble_ota_valid_file_callback)(uint32_t file_size, char
*file_version);
```

   - 用户在自己工程代码合适的位置调用该接口，将以上回调函数注册至 SDK。

```
void ble_ota_callback_reg(ble_ota_start_callback start_cb, ble_ota_stop_callback
stop_cb,ble_ota_valid_file_callback valid_file_cb);
```

上传固件

详情请参见 固件升级，按操作步骤上传固件。

> ⚠ 注意：

因为蓝牙升级时需要腾讯连连小程序与设备保持连接，因此“升级确认”方式只有“**用户确认升级**”这一种。

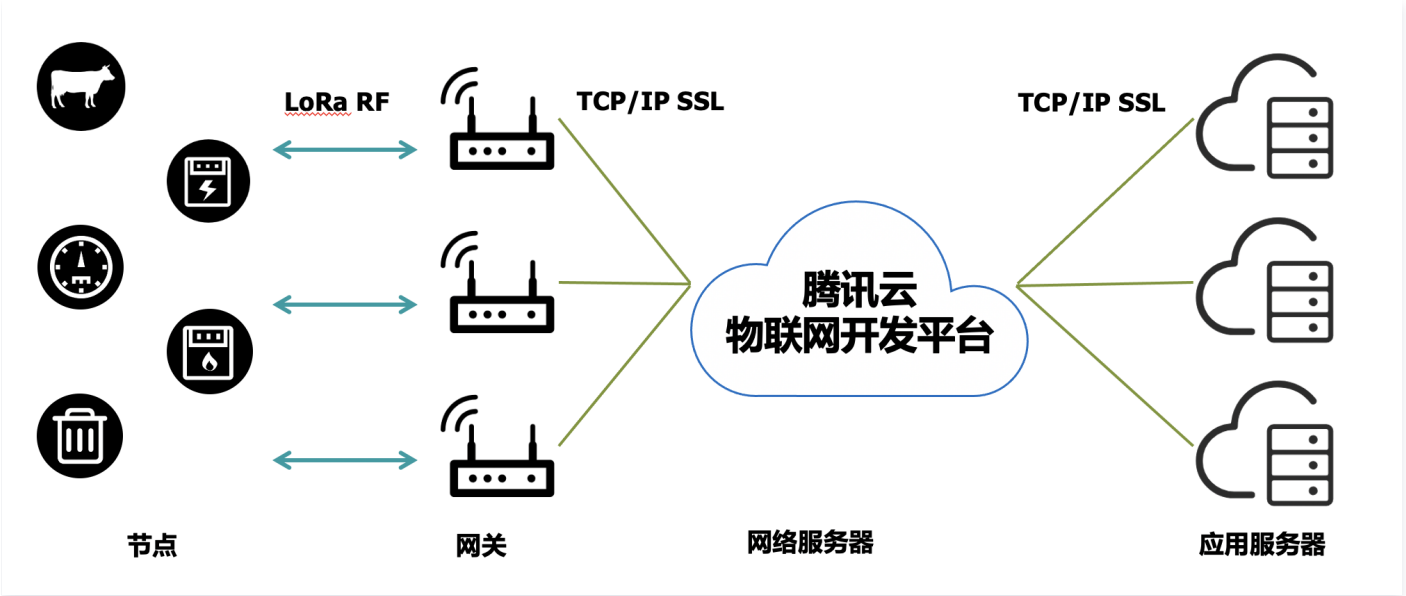
小程序操作

1. 打开**腾讯连连**小程序，单击需要升级的设备。
2. 蓝牙连接成功后点击左上角的“省略号”。
3. 单击**固件升级**，单击**立即升级**。
4. 小程序会自动下载固件并升级，升级过程会持续若干分钟（若设备支持断点续传功能小程序会自动继续升级）。
5. 升级完成后等待设备自动重启并连接，再次查看固件版本号，可看到新版本升级成功。

LoRaWan 设备接入开发
LoRaWan 产品简介

最近更新时间：2024-11-01 09:44:31

网络架构



频谱策略

请参考工信部于2019年11月28日发布的微功率技术要求《微功率短距离无线电发射设备目录和技术要求》公告 第四节民用计量仪表的使用要求，具体要求如下：

> （四）民用计量仪表
> 用于电力、热力、水务、燃气等公用计量业务。在建筑楼宇、住宅小区及村庄等小范围内组网应用，任意时刻限单个信道工作。
> 民用计量仪表设备应当具备“发射前搜寻”等干扰规避功能，且不能被用户调整或关闭。
> 若使用频率与当地声音、电视广播电台频率相同时，不得在当地使用；若对当地声音、电视广播接收产生干扰时，应立即停止使用，待消除干扰或调整到无干扰频率后方可重新使用。
> 1. 使用频率：470-510MHz。
> 2. 发射功率限值：50mW(e.r.p)。
> 3. 发射功率谱密度限值：占用带宽小于等于200kHz的,为50mW/200kHz(e.r.p)；占用带宽200-500kHz的，为10mW/100kHz(e.r.p)。
> 4. 单次发射持续时间：不超过1秒。
> 5. 占用带宽：不大于500kHz。
> 6. 频率容限：100 × 10^{-6}。

注意事项

基于上述要求，使用 LoRaWAN 时应注意以下几点：

1. 单终端及网关设备，不能同时使用两个及以上信道发送数据，但接收数据可以。
2. 发送功率加上天线的增益不能超过 17dBm，即 50mw。
3. 单次发送数据的空中时长 ToA 应不超过1s。

应用特征

LoRaWAN 并不适用每一种通信应用场景，使用者应根据 LoRaWAN 的特点来适配应用场景。

适用场景

- 长距离：根据不同的遮挡环境，可以覆盖几公里到几百公里，一般视距可达 10Km。

- 低功耗：典型场景下使用电池供电可以工作数年的时间。
- 低成本：通信模组成本可控制在 40RMB 以下，随着时间增长，成本会逐步降低。
- 低带宽：通信速率 250bits 到 5Kbits 不同的调制方式下均适用（SF7 到 SF12）。
- 按需部署：不需要依赖运营商的基站，可根据应用需求灵活部署网关。
- 安全：端到端128位 AES 加密。

不适用场景

- 实时数据：按秒为单位实时的上传应用数据的场景。
- 语音通信。
- 实时及高并发的下行控制场景。
- 大带宽：大数据包传输场景，LoRaWAN 物理层单个数据帧仅支持 242 字节传输。

关键功能及参数

工作模式

Class A

双向传输终端（**Class A**）：Class A 的终端在每次上行后都会紧跟两个短暂的下行接收窗口，以此实现双向传输。终端基于自身通信需求来安排传输时隙，在随机时间的基础上具有较小的变化（即 ALOHA 协议）。Class A 操作为应用提供了最低功耗的终端系统，只要求应用在终端上行传输后的很短时间内进行服务器的下行传输，服务器在其他任何时间进行的下行传输都需要等终端的下一次上行。

Class B

划定接收时隙的双向传输终端（**Class B**）：Class B 的终端有更多的接收时隙。除了 Class A 的随机接收窗口，Class B 设备还会在指定时间打开其他的接收窗口。为了让终端可以在指定时间打开接收窗口，终端需要从网关接收时间同步的信标（Beacon），使服务器知晓终端何时处于监听状态。

Class C

最大化接收时隙的双向传输终端（**Class C**）：Class C 的终端基本处于一直打开接收窗口的状态，只在发送时短暂关闭。Class C 的终端会比 Class A 和 Class B 更加耗电，但同时从服务器下发给终端的时延也是最短的。

> ⚠ **注意：**
> 目前腾讯 LoRaServer 支持 Class A、Class C 两种模式，Class B 暂未支持。

标识符

标识符	说明
DevEUI	64位设备终端标识符，设备唯一。
DevAddr	32位设备地址，设备非唯一。
AppEUI	应用标识符，目前腾讯 LoRaServer 未对 AppEUI 标识限制。
GatewayEUI	网关标识符，设备唯一。
NetID	腾讯使用 LoRaWAN 联盟分配地址前缀0x35（十进制53）。

网络接入模式

空中激活 OTAA

空中激活 OTAA 是目前推荐的连接方式，安全性更高，通过网络执行入网的过程，动态地生产会话密钥及 DevAddr。

本地激活 ABP

本地激活 ABP 接入网络的方式更为简单直接，无需入网流程，通过本地预存的会话密钥进行加解密，但存在一些安全性的问题如重放攻击，因此不推荐使用。

> ⚠ **注意：**
> 目前腾讯 LoRaServer 支持 OTAA、ABP 两种网络接入模式，用户需要妥善维护设备的密钥信息。

物理调制方式及速率

物理调制方式

基于 Chirp 扩频技术，LoRaWAN 使用 LoRa 的调制方式，即使在低功率下，也有很好的抗干扰特性，且多径衰弱及多普勒效应对 LoRaWAN 的影响也非常小。LoRaWAN 的速率取决于使用的带宽及扩频因子，目前在中国区可以使用125Khz带宽，SF7到SF12等6种速率，设备端决定使用扩频因子，不同的扩频因子极大的影响传输数据所需的时间和功耗。

> ⚠ **注意：**
> 在信号强度满足传输的条件下，请尽可能使用较高速率的扩频因子，以减少设备使用功耗，增加电池使用寿命及增加网关接入设备数量。

自适应速率 ADR

自适应速率 ADR 是一种用于优化网络速率、传输时间和功耗的机制，设备终端具有足够的稳定的信号强度下，需要启用 ADR。

- 静态终端：当设备长期处于静态状态，例如水表，此时 RF 环境相对稳定，应启用 ADR 功能，确保设备在合适的速率下运行。
- 移动终端：当设备处于移动的状态，例如包裹跟踪定位器，则在移动的状态下应禁用 ADR，使用固定的速率发送数据，确保数据能够被稳定的接收，当设备检测到设备处于静止的状态，应在此期间开启 ADR。

> ⚠ **注意：**
> 终端决定是否使用 ADR 功能，并非由网络或者应用来确定。ADR 开启期间，网络会根据最近多个数据包信号及丢包状态给出合适的速率建议。

LoRaWan 产品定义

最近更新时间：2024-12-26 18:07:33

操作场景

用户成功注册腾讯云账号后，通过物联网开发平台将 LoRaWAN 设备对接到腾讯云物联网平台时。您需要创建项目，并在项目下创建产品、定义产品的数据模板。本文档主要介绍如何使用开发平台创建项目并进行 LoRaWAN 产品定义。
LoRaWAN 产品定义与其他产品定义的操作步骤基本一致，只有创建产品时有不同，因此本文档中只对创建 LoRaWAN 产品这一步骤进行说明，其他步骤可参考通用的 产品管理 文档。

> ⚠ **注意：**
> LoRaWan服务需开通企业实例后才可使用。

操作步骤

产品相当于某一类设备的集合，用户通过产品管理其下的所有设备。

1. 登录 物联网开发平台控制台，单击某个已创建的项目。
2. 进入产品开发页，单击**新建产品**，开始定义您的产品。
3. 根据页面提示填写产品基本信息。产品基本信息设置如下：
   - 产品名称：名称支持中文、英文、数字、下划线、空格（非首尾字符）、中英文括号、-、@、\、/的组合，最多不超过40个字符。
   - 产品类型：选择您所创建产品的所属品类，不同类型产品的属性、事件等数据模板会有所不同。详情请参见 物模型定义 。
   - 设备类型：选择“设备”。
   - 认证方式：选择“密钥认证”。
   - 通信方式：LoRaWAN 产品的通信方式需要选择“LoRaWAN”。
   - 数据协议：产品默认采用自定义透传协议。
   - 描述：字数不能超过80个，您可以根据需要选填。

4. 信息填写完成后，单击**保存**即可。

LoRaWan 设备开发

最近更新时间：2024-12-24 17:18:32

操作场景

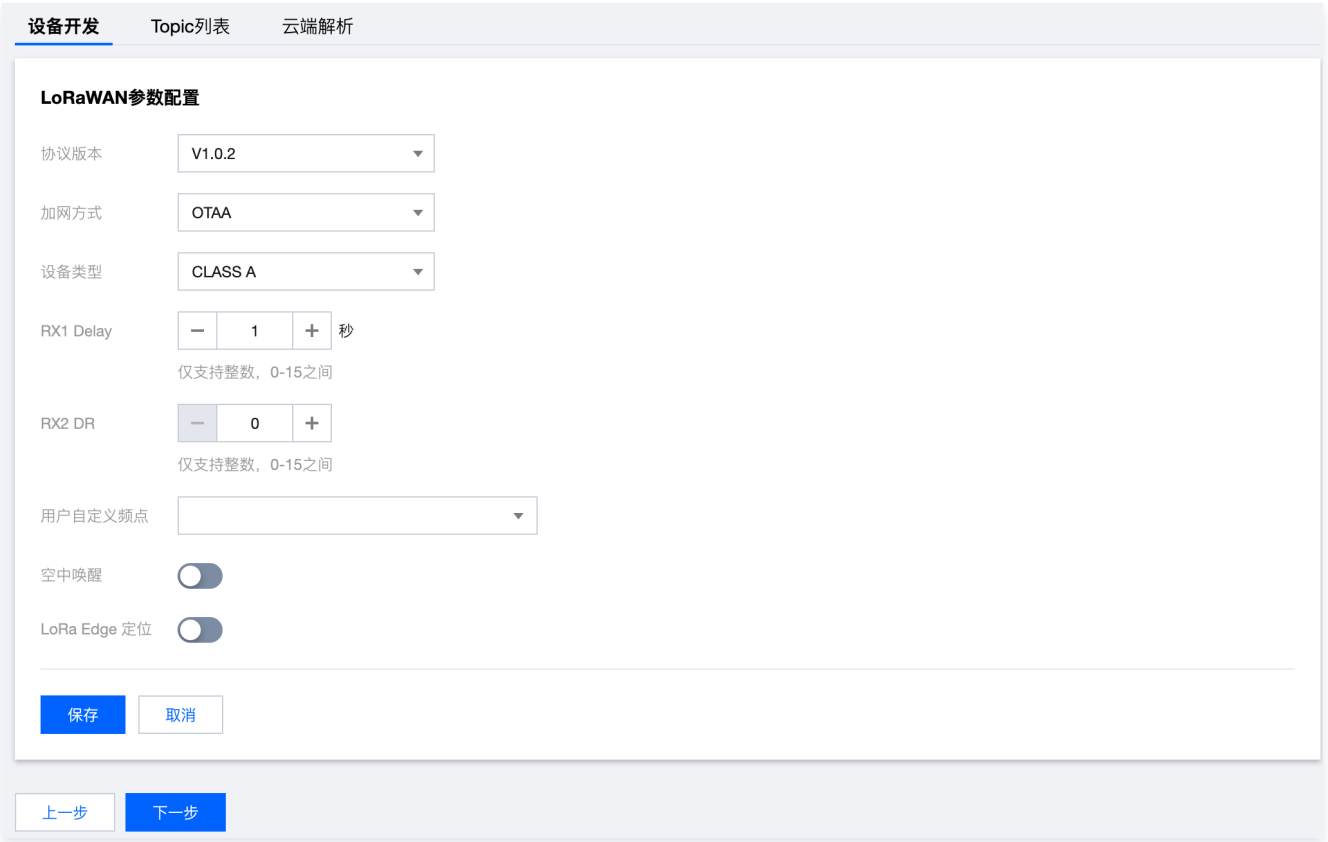
用户定义完产品与物模型后，需要按接入协议要求将 LoRaWAN 设备接入到平台。本文档主要介绍如何使用开发平台进行 LoRaWAN 设备开发。

前提条件

已创建完产品并定义产品的物模型。详情请参见 LoRaWAN 产品定义 。

操作步骤

1. 登录 物联网开发平台控制台 ，进入对应实例，单击已创建的产品名称，选择**设备开发**。
2. 选择设备开发方式，并单击右上角**编辑**，按照您的需求调整 LoRaWAN 参数配置。
3. 在弹出的页面中，调整完参数后，单击**保存**，完成 LoRaWAN 参数配置。



参数配置可选功能

功能	说明
协议版本	支持 LoRaWAN V1.0.0、V1.0.1、V1.0.2、V1.0.3。
加网方式	支持 OTAA、ABP 两种方式。
设备类型	支持 Class A、Class B、Class C 三种模式。
RX1 Delay	仅支持整数，0-15之间。
RX2 DR	仅支持整数，0-15之间。
用户自定义频点	用户自定义频点，设置适合用户应用场景的 Region 参数。
空中唤醒	开启后下行的数据会携带前导码，节点可通过前导码进行唤醒，以保证节省功耗同时保证实时性。

	注意：该功能仅支持已对接网关产品使用。

LoRaWan 设备数据云端解析

最近更新时间：2024-12-24 17:18:32

操作场景

由于 LoRaWAN 类资源有限设备不适合直接传输 JSON 格式数据，物联网开发平台提供了“设备数据解析”服务。用户通过编写自定义的解析脚本，可以将设备原始数据转化为产品定义的物模型协议数据。

操作步骤

编写脚本

登录 物联网开发平台控制台，进入目标产品的**设备开发 > 云端解析**页，在设备数据解析页签下，编写脚本。
上行数据解析的脚本主函数为 RawToProtocol，其带有 fPort、bytes 两个入参：

- fPort：设备上报的 LoRaWAN 协议数据的 FPort 字段。
- bytes：设备上报的 LoRaWAN 协议数据的 FRMPayload 字段。

脚本主函数的出参为产品数据模板协议格式的对象。
下行数据解析的脚本主函数为 ProtocolToRaw，其入参为产品数据模板协议格式的对象，其出参为至少3个字节的数组：

- 第1字节：下发给设备的 LoRaWAN 协议数据的 FPort 字段。
- 第2字节：bytes 为下发给设备的 LoRaWAN 协议数据的 MType（0 表示 Unconfirmed Data Down，1 表示 Confirmed Data Down）。
- 第3字节：开始为下发给设备的 LoRaWAN 协议数据的 FRMPayload 字段。

这里以温湿度传感器做示例说明，设备上行数据共4字节：

- 第1字节：温度。
- 第2字节：相对湿度。
- 第3、4字节：表示上报周期（单位秒）。
- 设备下行数据为2字节：上报周期（单位秒）。

在上行数据解析部分，javascript 示例代码如下：

```
function RawToProtocol(fPort, bytes) {
    var data = {
        "method": "report",
        "clientToken" : new Date(),
        "params" : {}
    };
    data.params.temperature = bytes[0];
    data.params.humidity = bytes[1];
    data.params.period = bytes[2] | (bytes[3] << 8);
    return data;
}
```

在下行数据解析部分，javascript 示例代码如下：

```
function ProtocolToRaw(obj) {
    var data = new Array();
    data[0] = 5;// fport=5
    data[1] = 0;// unconfirmed mode
    data[2] = obj.params.period & 0x00FF;
    data[3] = (obj.params.period >> 8) & 0x00FF;
    return data;
}
```

脚本模拟测试

您也可使用数据解析页面下方的模拟调试工具，如需开发更多的功能，请使用以下模拟脚本。
在设备上行数据的编辑框中填入模拟测试的 bytes 数据，为数组类型，编辑框的右上方可以填入模拟测试的 fPort 字段。

- **上行消息**
  假设设备上行原始数据为 0x11451E00，我们将其转化为数组，即上行模拟数据为：[17,69,30,0]，填入设备上行数据的编辑框中。单击**运行**，即可在模拟调试界面右侧查看结果。以温湿度传感器为例：

- **下行消息**
  在设备下行数据的编辑框中填入模拟测试的产品数据模板协议格式数据，为对象类型。
  假设设备下行原始数据如下，将其填入设备下行数据的编辑框中，以温湿度传感器为例：

```
{
"params": {
  "period": 15
}
}
```

  单击**运行**，即可在模拟调试界面右侧查看结果。

提交脚本

确认脚本可以正确解析数据后，单击**提交**，将该脚本提交到物联网开发平台，以供数据上下行时，物联网开发平台调用该脚本解析数据。

LoRaWan 设备调试

最近更新时间：2024-12-24 17:18:32

操作场景

设备开发完成后，需要进入设备调试阶段调试设备与云端的通信是否正常。设备调试提供了真实设备在线调试及虚拟设备调试，并可通过控制台查询设备上报的当前数据、历史通信日志、事件及上下线记录等。本文档主要介绍如何进行设备调试。
LoRaWAN 设备调试与其他产品的设备调试的操作步骤基本一致，只有新建设备时有不同，因此本文档中只对新建 LoRaWAN 设备这一步骤进行说明，其他步骤可参考通用的 LoRaWAN 设备调试 文档。

前提条件

已完成设备开发。详情请参见 LoRaWAN 设备开发 。

操作步骤

1. 登录 物联网开发平台控制台，设备开发完成后，单击**设备调试**。
2. 进入设备调试环节，单击**新建设备**，填写设备基本信息，单击**保存**，即可完成创建设备。
   - 设备名称：支持英文、数字、下划线的组合，最多不超过48个字符。
   - DevEUI：仅支持16进制字符，长度16位。
   - AppKey（仅限 OTAA 加网方式）：仅支持16进制字符，长度32位。
   - DevAddr（仅限 ABP 加网方式）：仅支持16进制字符，长度8位。
   - NwkSKey（仅限 ABP 加网方式）：仅支持16进制字符，长度32位。
   - AppSKey（仅限 ABP 加网方式）：仅支持16进制字符，长度32位。

   DevEUI、AppKey、DevAddr、NwkSKey、AppSKey 一般为 LoRaWAN 节点设备厂商提供。如果是自行开发协议栈，可以按需配置，只要平台和节点实际配置的内容一致即可。
   下图示例中为 OTAA 加网方式，如果需要切换到 ABP 加网方式，可以在**设备开发**界面中调整 “LoRaWAN 参数配置” 中的加网方式。

3. 创建成功后，您将会在“设备调试”列表页中，查看到新建成功的设备。

查看设备

1. 创建设备成功后，您将会在“设备调试”列表页中，查看到新建成功的设备。
2. 单击设备名称，可以查看设备相关信息、设备信息、设备日志、透传日志等。

设备信息

单击**设备信息**，即可查看设备基础的信息、名称、密钥、激活时间、最后上线时间等。
此外，在 ABP 模式下，可在设备信息中选择**禁用帧序号校验（临时调试使用）**以及**重置帧序号**功能。

> **说明：**
> 此功能建议用于设备临时调试，以解决设备上传序列号归零导致 MIC 校验错误的问题；不建议用户用于商业应用，存在设备重放攻击的安全隐患。

设备日志

单击设备日志，即可查看该设备上行到云端，并从云端接收的信息，可查看7天以内的设备日志内容。

- **上行：上行表示设备端向云端上报的数据。**
- **下行：下行表示云端向设备端发送的数据。**

设备透传日志

单击**设备日志（透传数据）**，即可查看数据透传到用户侧的数据内容，以及 LoRaWAN 网络侧内容数据。

示例

数据内容如下：

```
“eyJtZXRob2QiOiJyZXBvcnQiLCJjbGllbnRUb2tlbiI6IjIwMjEtMDEtMThUMDU6NTE6MDIuMDM4WiIsInBhcmFtcyI6eyJyYXdkYXRhIjoiMDAwMTAyMDMifSwibWV0YUxvUmEiOiJ7XCJmcmFtZVR5cGVcIjoyLFwiZlBvcnRcIjoyLFwiZkNudFwiOjQ2NCxcImZyZXF1ZW5jeVwiOjQ3MDMwMDAwMCxcImRyXCI6NSxcInJzc2lcIjotNTAsXCJzbnJcIjoyNixcInBheWxvYWRTaXplXCI6NH0ifQ==”
```

通过 BASE64 解析成文本的格式如下：

```
{"method":"report","clientToken":"2021-01-18T05:51:02.038Z","params":{"rawdata":"00010203"},"metaLoRa":"
{\"frameType\":2,\"fPort\":2,\"fCnt\":464,\"frequency\":470300000,\"dr\":5,\"rssi\":-50,\"snr\":26,\"paylo
adSize\":4}"}
```

在线调试

当您的真实设备已成功对接到开发平台后，则可使用在线调试对真实设备进行数据收发的测试，具体步骤如下：

1. 单击**在线调试**，即可进入在线调试功能。
2. 在线调试左侧的操控面板是根据设备所属产品的数据模板自动生成，设置需要下发的数据后，单击**发送**，系统会自动触发控制指令到设备端。

3. 设备端接收到指令后，会立刻返回数据到云端并显示在右侧的文本框中。

LoRaWan 网关管理

最近更新时间：2024-12-24 17:18:32

操作场景

用户创建完 LoRaWAN 设备后，需要通过 LoRaWAN 网关将设备接入到平台。本文档主要介绍如何使用物联网开发平台进行 LoRaWAN 网关管理。

操作步骤

使用腾讯 LoRaWAN 社区网络

1. 登录 物联网开发平台控制台，进入实例。
2. 选择左侧菜单**增值服务 > 网络管理**。导航栏右侧会展示出腾讯 LoRaWAN 社区网络的地图页面。
3. 您可以单击地图，并查看自己的附近是否有正在运行的 LoRaWAN 网关，这些网关为社区的开发者主动共享的，可以直接通过这些网关进行 LoRaWAN 设备的无线接入。

新建用户 LoRaWAN 网关

1. 登录 物联网开发平台控制台，进入实例。
2. 选择左侧菜单**增值服务 > 网络管理 > 用户网关**，单击**添加网关**。

3. 在新建网关页面，填写网关基本信息。
   - 网关名称：本示例中填写 GW1。
   - GwEUI：为网关唯一 ID。
   - 是否公开：
     - 选择“是”，表示社区开发者可在社区网络中看到该网关，并可通过这个网关进行 LoRaWAN 节点接入。
     - 选择“否”，则只有用户自己才能查看该网关。

4. 网关新建成功后，您可在网关列表页查看到“GW1”。

用户 LoRaWAN 网关操作

> **说明：**
> 用户的 LoRaWAN 网关需支持 Packet Forwarder 协议。

LoRaWAN 网关上的配置需做如下调整：

```
配置接入域名：loragw.things.qcloud.com
接入端口：1700
```

当网关按要求配置并重启运行之后，即可在网关列表页查看到网关的在线状态变为“在线”。

LoRaWan 用户自定义频点

最近更新时间：2024-12-24 17:18:32

背景介绍

- LoRaWAN 联盟在 lorawan_regional_parameters_v1.x 版本的 CN**470Mhz-510Mhz**，40Mhz频宽中，定义了上行96个信道，下行48个信道。
- LoRaWAN 联盟在 lorawan_regional_parameters_v2.x 版本的 CN**470Mhz-510Mhz**，40Mhz频宽中，定义了26M TypeA 和 TypeB、20M TypeA 和 TypeB 共4种频谱计划。

频谱计划	上行信道数	下行信道数	总频宽	上行频点对应模式
RP1.0	96	48	40M	异频
RP2.0 20M TypeA	64	64	20M	异频
RP2.0 20M TypeB	64	64	20M	同频
RP2.0 26M TypeA	48	24	26M	异频
RP2.0 26M TypeB	48	24	26M	异频

自定义功能说明

- 自定义频点功能兼容 RP1.0、RP2.0 各个版本的频谱计划，同时也可以支持用户自定义频谱计划，已满足不同环境下的要求。
- 自定义频点功能支持 MACCommand 中大部分功能，除了 linkadr 中 MASK 参数，设备回复 linkadrans 应作出相应的策略。

功能	说明
单信道频谱要求	网关工作在单个信道。
多信道频谱要求	网关工作在多个信道，支持2个 – 64个信道，自由选择可用信道。
全双工工作模式	网关上下行发送和接收数据同时工作。
上下同频	上下行共用同一组频点。
上下异频	上下行使用不同组频点。

操作步骤

新建频点计划

1. 登录 物联网开发平台控制台，进入企业实例。
2. 选择左侧菜单**增值服务 > 网络管理**，在页面上方选择**用户自定义频点**后，单击**添加频点**，进入“添加用户自定义频点”页面。
   - 频点配置名称：支持英文、数字、下划线的组合，最多不超过48个字符。
   - 数据信道配置：分别填入上行信道，下行 RX1 信道，下行 RX2 信道，单位Hz，仅支持数字，范围为470000000-510000000。
   - 入网信道配置：分别填入上行信道，下行 RX1 信道，下行 RX2 信道，单位Hz，仅支持数字，范围为470000000-510000000，若未配置入网信道，则按照数据信道入网。

示例

以双信道频点计划为例，配置两个信道，同频工作模式的频点计划：

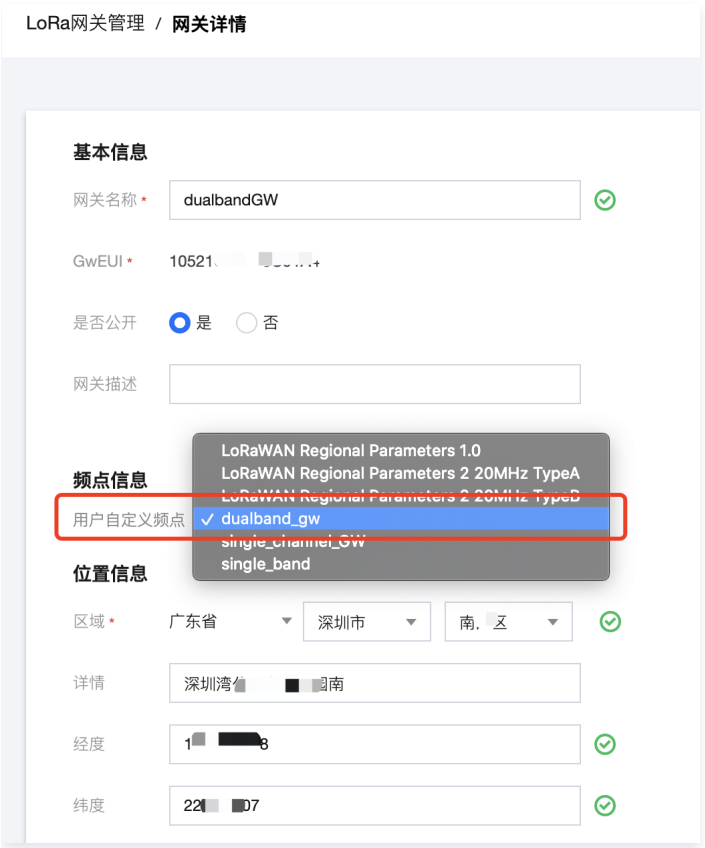
网关关联频点计划

1. 登录 物联网开发平台控制台，进入企业实例。
2. 选择左侧菜单**增值服务 > 网络管理**，在页面上方选择**用户网关**后，单击**添加网关**，进入新建或者编辑网关界面。
3. 在频点信息处选择用户自定义频点，关联与网关对应的频点计划。

> **说明：**
> 缺省用户频点为 RP1.0 频点计划，可选支持 RP2 20MHz TypeA、RP2 20MHz TypeB。

示例

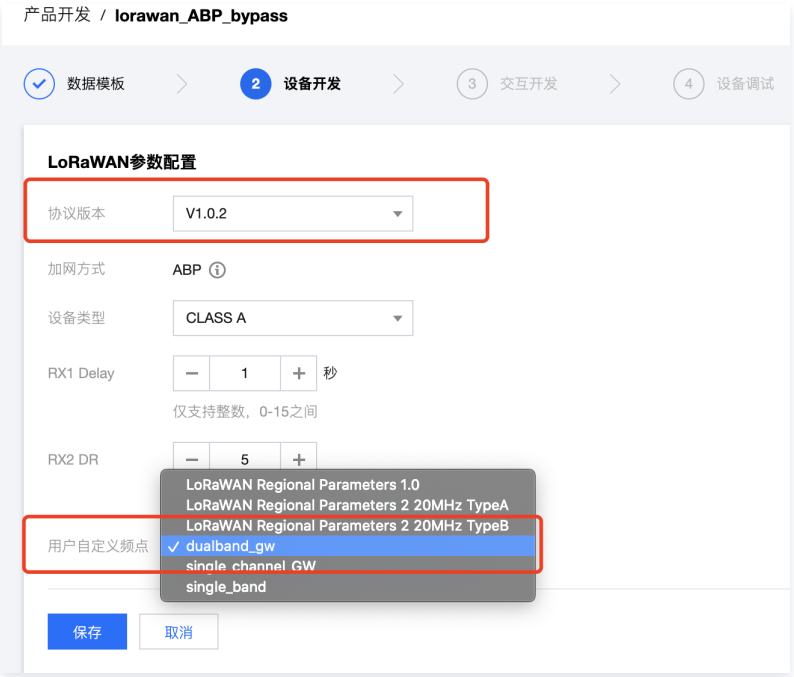
以网关关联双芯片频点计划为例：



设备关联频点计划

1. 登录 物联网开发平台控制台，进入企业实例。
2. 选择左侧菜单**产品开发**，选择需要打开的产品，在“物模型定义”页面下方，单击**下一步**进入“设备开发”页面。
3. 在 LoRaWAN 参数配置界面中，选择用户自定义频点，关联与设备对应的频点计划。

> **说明：**
> 同一个网络中，多个网关硬件使用相同的频点配置，若设备或者网关未关联对应的频点计划，则上行数据可以正常接收，下行数据发送只会选择关联频点计划的网关。


产品开发 / lorawan_ABP_bypass
数据模板
2 设备开发
3 交互开发
4 设备调试
LoRaWAN参数配置
协议版本
V1.0.2
加网方式
ABP
设备类型
CLASS A
RX1 Delay
1
秒
仅支持整数，0-15之间
RX2 DR
5
用户自定义频点
LoRaWAN Regional Parameters 1.0
LoRaWAN Regional Parameters 2 20MHz TypeA
LoRaWAN Regional Parameters 2 20MHz TypeB
dualband_gw
single_channel_GW
single_band
保存
取消