

# 阿里云 物联网平台

设备管理

文档版本：20200508

# 法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云文档中所有内容，包括但不限于图片、架构设计、页面布局、文字描述，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

## 通用约定

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

# 目录

法律声明.....	I
通用约定.....	I
<b>1 设备生命周期管理.....</b>	<b>1</b>
1.1 创建设备.....	1
1.2 设备上、下线.....	1
1.3 禁用和启用设备.....	2
1.4 删除设备.....	3
<b>2 物模型.....</b>	<b>4</b>
2.1 什么是物模型.....	4
2.2 单个添加物模型.....	7
2.3 批量添加物模型.....	19
<b>3 数据解析.....</b>	<b>22</b>
3.1 什么是数据解析.....	22
3.2 自定义Topic数据解析.....	25
3.2.1 概述.....	25
3.2.2 JavaScript脚本示例.....	27
3.2.3 Python脚本示例.....	28
3.2.4 PHP脚本示例.....	30
3.3 物模型数据解析.....	33
3.3.1 物模型数据解析使用示例.....	33
3.3.2 JavaScript脚本示例.....	38
3.3.3 Python脚本示例.....	41
3.3.4 PHP脚本示例.....	44
3.4 LoRaWAN设备数据解析.....	49
3.5 问题排查.....	55
<b>4 标签.....</b>	<b>58</b>
<b>5 设备分组.....</b>	<b>62</b>
<b>6 设备影子.....</b>	<b>65</b>
6.1 设备影子概览.....	65
6.2 设备影子JSON详解.....	66
6.3 设备影子数据流.....	69
<b>7 文件管理.....</b>	<b>77</b>
<b>8 NTP服务.....</b>	<b>79</b>
<b>9 网关与子设备.....</b>	<b>81</b>
9.1 网关与子设备.....	81
9.2 子设备管理.....	83
9.3 子设备上线.....	83

<b>10 CA证书管理.....</b>	<b>85</b>
<b>11 Alink协议.....</b>	<b>90</b>
11.1 Alink协议.....	90
11.2 设备身份注册.....	98
11.3 管理拓扑关系.....	102
11.4 子设备上下线.....	111
11.5 设备属性、事件、服务.....	116
11.6 设备期望属性值.....	132
11.7 子设备配置下发网关.....	136
11.8 子设备禁用、启用、删除.....	154
11.9 设备标签.....	157
11.10 TSL模板.....	159
11.11 固件升级.....	162
11.12 远程配置.....	167
11.13 设备日志上报.....	170
11.14 设备网络状态.....	175
11.15 设备端通用code.....	179
<b>12 设备端错误码.....</b>	<b>180</b>



# 1 设备生命周期管理

## 1.1 创建设备

通过物联网平台的设备管理功能，可查看和管理设备的生命周期。首先需在物联网平台上创建设备。您可以在控制台创建设备或调用云端API创建设备。

### 在控制台创建设备

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 产品**。
3. 在**产品管理**页，单击**创建产品**，填入产品信息，创建产品。

具体操作说明，请参见[#unique\\_5](#)。

4. 在左侧导航栏，选择**设备**。
5. 在**设备管理**页，添加设备。

- 单击**批量添加**，批量创建设备；
- 单击**添加设备**，单个创建设备。

具体操作说明，请参见[#unique\\_6](#)或[#unique\\_7](#)。

## 1.2 设备上、下线

设备上线，即设备端接入物联网平台，设备状态显示为**在线**；设备下线，即设备端断开与物联网平台的连接，设备状态显示为**离线**。

### 设备上线

开发设备端，设备接入物联网平台。



#### 说明：

以下是直连设备上线过程。子设备上线，请参见[子设备上线](#)。

### 1. 开发设备端。

物联网平台提供了多种语言的设备端SDK，这些SDK已封装了设备端与物联网平台的交互协议。使用物联网平台设备端SDK进行开发，请参见[#unique\\_10](#)。

开发设备端时，需在设备端上配置设备身份信息，用于设备接入物联网平台时，进行身份验证。

物联网平台支持的直连设备身份认证方案有：

- [#unique\\_11](#)：预先为每个设备烧录其唯一的设备证书（ProductKey、DeviceName和DeviceSecret）。
- [#unique\\_12](#)：同一产品下所有设备可以烧录相同固件（即烧录ProductKey和ProductSecret），并需在控制台上设备所属产品的[产品详情页](#)，打开动态注册开关。设备发送连接请求时，物联网平台验证产品证书。认证通过，下发该设备对应的DeviceSecret。

### 2. 安装设备端SDK到设备上。

### 3. 设备通电、连网后，接入物联网平台。

## 设备下线

设备下线后，该设备在物联网平台上的状态为**离线**。设备下线分为：

- 设备主动下线：设备端主动断开与物联网平台的连接。
- 设备被动下线：物联网平台主动断开与设备的连接。场景如：有其他设备使用相同的设备证书接入物联网平台，导致当前设备被迫下线；您在物联网平台上，删除或禁用了该设备等。

## 1.3 禁用和启用设备

禁用设备，即禁止设备接入物联网平台；启用设备，即重新启用已被禁用设备，允许设备重新接入物联网平台。本文介绍如何禁用和启用设备。

### 禁用设备



#### 说明：

设备被禁用后，物联网平台中，与该设备关联的信息依然保留。但是，该设备将不能接入物联网平台，您将无法执行与该设备有关的操作。

在控制台禁用设备的操作步骤如下。

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 设备**。
3. 在**设备列表**中，找到要禁用的设备，关闭设备对应的**启用状态**开关。



## 启用设备

禁用设备后，您还可以重新启用设备，即解除设备禁用。

在控制台启用设备的操作步骤如下。

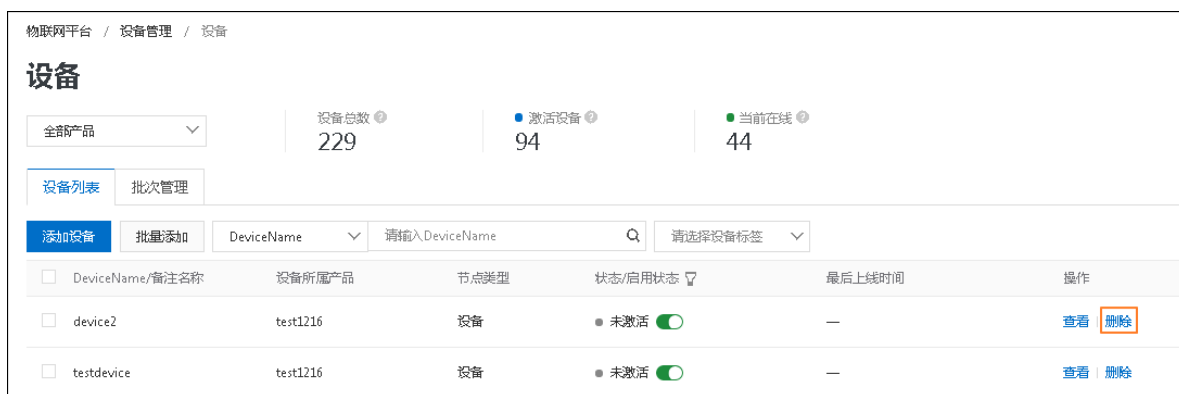
1. 在[物联网平台控制台](#) **设备管理**页的**设备列表**中，找到要启用的设备。
2. 打开设备对应的**启用状态**开关。

## 1.4 删除设备

将设备从物联网平台中删除。设备从物联网平台删除后，设备ID将失效，与该设备关联的其他信息也一并删除，您将无法通过物联网平台执行与该设备关联的任何操作。

### 操作步骤

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 设备**。
3. 在**设备列表**中，找到要删除的设备。
4. 单击对应的**删除**按钮，并确认删除。



### 预期结果

设备删除后，该设备证书失效，且不能恢复。

## 2 物模型

### 2.1 什么是物模型

物模型指将物理空间中的实体数字化，并在云端构建该实体的数据模型。在物联网平台中，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。物模型描述产品是什么、能做什么、可以对外提供哪些服务。

物模型TSL（Thing Specification Language）。是一个JSON格式的文件。它是物理空间中的实体，如传感器、车载装置、楼宇、工厂等在云端的数字化表示，从属性、服务和事件三个维度，分别描述了该实体是什么、能做什么、可以对外提供哪些信息。定义了这三个维度，即完成了产品功能的定义。

物模型将产品功能类型分为三类：属性、服务和事件。定义了这三类功能，即完成了物模型的定义。

功能类型	说明
属性（Property）	一般用于描述设备运行时的状态，如环境监测设备所读取的当前环境温度等。属性支持GET和SET请求方式。应用系统可发起对属性的读取和设置请求。
服务（Service）	设备可被外部调用的能力或方法，可设置输入参数和输出参数。相比于属性，服务可通过一条指令实现更复杂的业务逻辑，如执行某项特定的任务。
事件（Event）	设备运行时的事件。事件一般包含需要被外部感知和处理的通知信息，可包含多个输出参数。如，某项任务完成的信息，或者设备发生故障或告警时的温度等，事件可以被订阅和推送。

#### 物模型格式

您可以在产品的**功能定义**页面，单击**物模型TSL**，查看JSON格式的TSL。

物模型的JSON字段结构如下：

```
{
  "schema": "物模型结构定义的访问URL",
  "profile": {
    "productKey": "产品ProductKey"
  },
  "properties": [
    {
      "identifier": "属性唯一标识符（产品下唯一）",
      "name": "属性名称",
      "accessMode": "属性读写类型：只读（r）或读写（rw）。",
      "required": "是否是标准功能的必选属性",
      "dataType": {
```

```

    "type": "属性类型: int (原生)、float (原生)、double (原生)、text (原生)、
    date (String类型UTC毫秒)、bool (0或1的int类型)、enum (int类型)、struct (结构体类
    型, 可包含前面7种类型)、array (数组类型, 支持int/double/float/text/struct)",
    "specs": {
      "min": "参数最小值 (int、float、double类型特有)",
      "max": "参数最大值 (int、float、double类型特有)",
      "unit": "属性单位",
      "unitName": "单位名称",
      "size": "数组大小, 默认最大128 (数组特有)。",
      "step": "步长, 字符串类型。",
      "item": {
        "type": "数组元素的类型"
      }
    }
  },
  ],
  "events": [
    {
      "identifier": "事件唯一标识符 (产品下唯一, 其中post是默认生成的属性上报事件。)",
      "name": "事件名称",
      "desc": "事件描述",
      "type": "事件类型 (info、alert、error)",
      "required": "是否是标准功能的必选事件",
      "callType": "async (异步调用) 或sync (同步调用)",
      "outputData": [
        {
          "identifier": "参数唯一标识符",
          "name": "参数名称",
          "dataType": {
            "type": "属性类型: int (原生)、float (原生)、double (原生)、text (原
            生)、date (String类型UTC毫秒)、bool (0或1的int类型)、enum (int类型)、struct (结
            构体类型, 可包含前面7种类型)、array (数组类型, 支持int/double/float/text/struct)",
            "specs": {
              "min": "参数最小值 (int、float、double类型特有)",
              "max": "参数最大值 (int、float、double类型特有)",
              "unit": "属性单位",
              "unitName": "单位名称",
              "size": "数组大小, 默认最大128 (数组特有)。",
              "step": "步长, 字符串类型。",
              "item": {
                "type": "数组元素的类型"
              }
            }
          }
        }
      ]
    },
    {
      "method": "事件对应的方法名称 (根据identifier生成)"
    }
  ],
  "services": [
    {
      "identifier": "服务唯一标识符 (产品下唯一, 其中set/get是根据属性的accessMode默认
      生成的服务。)",
      "name": "服务名称",
      "desc": "服务描述",
      "required": "是否是标准功能的必选服务",
      "callType": "async (异步调用) 或sync (同步调用)",
      "inputData": [
        {
          "identifier": "入参唯一标识符",
          "name": "入参名称",
          "dataType": {

```

```

        "type": "属性类型: int (原生)、float (原生)、double (原生)、text (原生)、date (String类型UTC毫秒)、bool (0或1的int类型)、enum (int类型)、struct (结构体类型, 可包含前面7种类型)、array (数组类型, 支持int/double/float/text/struct)",
        "specs": {
            "min": "参数最小值 (int、float、double类型特有)",
            "max": "参数最大值 (int、float、double类型特有)",
            "unit": "属性单位",
            "unitName": "单位名称",
            "size": "数组大小, 默认最大128 (数组特有)。",
            "step": "步长, 字符串类型。",
            "item": {
                "type": "数组元素的类型"
            }
        }
    },
    ],
    "outputData": [
        {
            "identifier": "出参唯一标识符",
            "name": "出参名称",
            "dataType": {
                "type": "属性类型: int (原生)、float (原生)、double (原生)、text (原生)、date (String类型UTC毫秒)、bool (0或1的int类型)、enum (int类型)、struct (结构体类型, 可包含前面7种类型)、array (数组类型, 支持int/double/float/text/struct)",
                "specs": {
                    "min": "参数最小值 (int、float、double类型特有)",
                    "max": "参数最大值 (int、float、double类型特有)",
                    "unit": "属性单位",
                    "unitName": "单位名称",
                    "size": "数组大小, 默认最大128 (数组特有)。",
                    "step": "步长, 字符串类型。",
                    "item": {
                        "type": "数组元素的类型 (数组特有)"
                    }
                }
            }
        }
    ],
    "method": "服务对应的方法名称 (根据identifier生成)"
}
]
}

```

若产品的节点类型是设备，且接入网关协议为Modbus、OPC UA或自定义时，可以查看物模型扩展配置。

Modbus协议产品的扩展配置数据结构如下：

```

{
  "profile": {
    "productKey": "产品ProductKey"
  },
  "properties": [
    {
      "identifier": "属性唯一标识符 (产品下唯一)",
      "operateType": " (线圈状态/输入状态/保持寄存器/输入寄存器: coilStatus/inputStatus/holdingRegister/inputRegister)",
      "registerAddress": "寄存器地址",
      "originalDataType": {
        "type": "属性类型: int16、uint16、int32、uint32、int64、uint64、float、double、string、customized data (按大端顺序返回hex data)",
      }
    }
  ]
}

```

```
"specs": {
  "registerCount": "寄存器的数据个数，string和customized data特有。",
  "swap16": "把寄存器内16位数据的前后8个bits互换（byte1byte2 -> byte2byte10），除string和customized data外，其他数据类型特有。",
  "reverseRegister": "把原始数据32位数据的bits互换（byte1byte2byte3byte4 -> byte3byte4byte1byte2），除string和customized data外，其他数据类型特有。"
},
"scaling": "缩放因子",
"pollingTime": "采集间隔，单位是ms。",
"trigger": "数据的上报方式，目前有按时上报：1和变更上报：2。"
}
```

### 物模型使用流程

1. 在[物联网平台控制台](#)添加物模型，请参见[单个添加物模型](#)、[批量添加物模型](#)。
2. 参见[Link SDK文档](#)，进行设备端物模型开发。
3. 开发完成后，设备端可以上报属性和事件；云端可以向设备端发送设置属性和调用服务的指令。

## 2.2 单个添加物模型

单个添加物模型，即单个添加属性、事件和服务。本文介绍如何在物联网平台控制台定义物模型。

### 背景信息

- 如果产品已发布，不能新增或编辑物模型。如需新增或编辑物模型，需先撤销产品发布。
- 可以编辑物模型历史版本，生成新的版本。
- 同一物模型最多保存最近的10个版本，多余的历史版本将被覆盖。
- 编辑物模型后，需发布才会正式生效。

### 操作步骤

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 产品**。
3. 在**产品管理**页面产品列表中，单击产品对应的**查看**操作按钮。
4. 在**产品详情页**，选择**功能定义 > 编辑草稿**。
5. （可选）从**历史版本**下拉菜单选择要编辑的历史版本。

6. 添加标准功能。选择**添加标准功能**，然后在弹出对话框中，选择适用于该产品的物联网平台预定义的标准功能。



## 7. 添加自定义功能。

选择**添加自定义功能**。您可以为产品自定义属性、服务和事件。

- 自定义属性：在**添加自定义功能**对话框，选择功能类型为**属性**。设置参数完成后，单击**确认**。

属性

服务

事件

\* 功能名称 ?

当前电压

\* 标识符 ?

CurrentVoltage

\* 数据类型

double (双精度浮点型) ▾

\* 取值范围

0

~

36

\* 步长

0.01

单位

伏特 / V ▾

\* 读写类型




☒ 读写

☐ 只读


描述

请输入描述

属性参数设置说明如下表。

参数	描述
功能名称	<p>属性的名称，例如用电量。同一产品下功能名称不能重复。</p> <p>支持中文、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。</p> <p>如果您创建产品时选择了功能模板，输入功能名称时，将从标准功能库中筛选匹配的标准属性，供您选择。</p> <div>  <b>说明：</b>            当接入网关协议为Modbus时，不支持标准属性，仅支持自定义属性。         </div>
标识符	<p>属性唯一标识符，在产品中具有唯一性。即Alink JSON格式中的<b>identifier</b>的值，作为设备上报该属性数据的Key，云端根据该标识符校验是否接收数据。可包含英文、数字、下划线，长度不超过50个字符，例如PowerConsumption。</p> <div>  <b>说明：</b>            不能用以下系统保留参数作为标识符：            set、get、post、time、value。         </div>
数据类型	<ul style="list-style-type: none"> <li>- int32：32位整型。需定义取值范围、步长和单位符号。</li> <li>- float：单精度浮点型。需定义取值范围、步长和单位符号。</li> <li>- double：双精度浮点型。需定义取值范围、步长和单位符号。</li> <li>- enum：枚举型。定义枚举项的参数值和参数描述，例如1-加热模式、2-制冷模式等。</li> <li>- bool：布尔型。采用0或1来定义布尔值，例如0-关；1-开。</li> <li>- text：字符串。需定义字符串的数据长度，最长支持2048字节。</li> <li>- date：时间戳。格式为String类型的UTC时间戳，单位：毫秒。</li> <li>- struct：JSON对象。定义一个JSON结构体，新增JSON参数项，例如定义灯的颜色是由Red、Green、Blue三个参数组成的结构体。不支持结构体嵌套。</li> <li>- array：数组。需声明数组内元素的数据类型，可选择int32、float、double、text或struct。需确保同一个数组元素类型相同。数组内可包含1-512个元素。</li> </ul> <div>  <b>说明：</b>            当设备协议为Modbus时，无需设置该参数。         </div>
取值范围	数据类型为int32、float、double时，需设置属性值的取值范围。



参数	描述
步长	<p>属性值变化的最小粒度。数据类型为int32、float、double时，需要根据您的业务需要设置步长。</p> <p>例如为温度计产品定义温度属性时，将数据类型设置为int32，步长为2，单位为℃，取值范围0~100。即温度每变化两度，设备上报温度值，例如0℃、2℃、4℃、6℃、8℃.....。</p>
单位	单位可选择为无，或根据实际情况选择。
读写类型	<ul style="list-style-type: none"><li>- 读写：请求读写的方法支持GET（获取）和SET（设置）。</li><li>- 只读：请求只读的方法仅支持GET（获取）。</li></ul> <div> <b>说明：</b> 当接入网关协议为Modbus时，无需设置该参数。</div>
描述	输入文字，对该功能进行说明或备注。长度限制为100字。

参数	描述
扩展描述	<p>设备通信协议到标准物模型的映射关系。</p> <p>设备接入网关协议为自定义、OPC UA或Modbus时，需填写该参数。</p> <ul style="list-style-type: none"> <li>- 接入网关协议为自定义时，填写JSON格式的自定义配置信息，长度限制为1024字符。</li> <li>- 接入网关协议为OPC UA时，设置节点名称。节点名称需保证属性维度下唯一。</li> <li>- 接入网关协议为Modbus时，需设置以下参数： <ul style="list-style-type: none"> <li>■ 操作类型： <ul style="list-style-type: none"> <li>■ 线圈状态（只读，01）</li> <li>■ 线圈状态（读写，读取使用01，写入使用05）</li> <li>■ 线圈状态（读写，读取使用01，写入使用0F）</li> <li>■ 离散量输入（只读，02）</li> <li>■ 保持寄存器（只读，03）</li> <li>■ 保持寄存器（读写，读取使用03，写入使用06）</li> <li>■ 保持寄存器（读写，读取使用03，写入使用10）</li> <li>■ 输入寄存器（只读，04）</li> </ul> </li> <li>■ 寄存器地址：十六进制，必须以0x开头，且限制范围是0x0~0xFFFF，例如0xFE。</li> <li>■ 原始数据类型：支持int16、uint16、int32、uint32、int64、uint64、float、double、string、bool、自定义（原始数据）多种数据类型。</li> <li>■ 取值范围：这是原始数据经过缩放因子处理之后的取值范围。不在该取值范围内的数据会被丢弃。物联网平台已为各操作类型设置了默认取值范围： <ul style="list-style-type: none"> <li>■ 线圈状态类型：0~1</li> <li>■ 离散量输入类型：0~1</li> <li>■ 保持寄存器类型：-2147483648~2147483647</li> <li>■ 输入寄存器类型：-2147483648~2147483647</li> </ul> </li> <li>■ 交换寄存器内高低字节：是否把寄存器内16位数据的前后8个bits互换。 <ul style="list-style-type: none"> <li>■ true：互换。</li> <li>■ false：不互换。</li> </ul> </li> <li>■ 交换寄存器顺序：是否把原始数据32位数据的bits互换。 <ul style="list-style-type: none"> <li>■ true：互换。</li> <li>■ false：不互换。</li> </ul> </li> <li>■ 缩放因子：不能为0，默认为1，可以为负数。</li> <li>■ 采集间隔：数据采集间隔，单位ms，不能小于10。</li> <li>■ 数据上报方式：可选按时上报和变更上报。</li> </ul> </li> </ul>

- 自定义服务：在**添加自定义功能**对话框，选择功能类型为**服务**。设置参数完成后，单击**确认**。



**说明：**

接入网关的协议选择为Modbus时，不支持定义服务。

属性

服务

事件

\* 功能名称 ?

同步时间

\* 标识符 ?

SyncTime

\* 调用方式 ?

☐ 异步

☒ 同步

输入参数

+

参数名称: 本地时间

编辑 | 删除

+增加参数

输出参数

+

参数名称: 状态

编辑 | 删除




+增加参数


描述

请输入描述

0/100

服务参数设置说明如下表。

参数	描述
功能名称	<p>服务名称。</p> <p>支持中文、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。</p> <p>如果您创建产品时选择了功能模板，输入功能名称时，将从标准功能库中筛选匹配的标准服务，供您选择。</p> <div>  <b>说明：</b>            当接入网关协议为Modbus时，不支持自定义服务。         </div>
标识符	<p>服务唯一标识符，在产品下具有唯一性。即Alink JSON格式中该服务的 <b>identifier</b> 的值。可包含英文、数字、和下划线，长度不超过30个字符。</p> <div>  <b>说明：</b>            不能用以下系统保留参数作为标识符：            set、get、post、time、value。         </div>
调用方式	<ul style="list-style-type: none"> <li>- 异步：服务为异步调用时，云端执行调用后直接返回结果，不会等待设备的回复消息。</li> <li>- 同步：服务为同步调用时，云端会等待设备回复；若设备没有回复，则调用超时。</li> </ul>
输入参数	<p>设置该服务的入参，可选。</p> <p>单击<b>新增参数</b>，在弹窗对话框中添加服务入参。</p> <p>当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。</p> <div>  <b>说明：</b> <ul style="list-style-type: none"> <li>- 不能用以下系统保留参数作为输入参数的标识符：set、get、post、time、value。</li> <li>- 您可以使用某个属性作为入参，也可以自定义参数。例如在定义<b>自动喷灌</b>服务功能时，将已定义的属性<b>喷灌时间</b>和<b>喷灌量</b>作为自动喷灌服务的入参，则调用该参数时传入这两个参数，喷灌设备将按照设定的喷灌时间和喷灌量自动进行精准灌溉。</li> <li>- 一个服务最多支持定义 20 个入参。</li> </ul> </div>

参数	描述
输出参数	<p>设置该服务的出参，可选。</p> <p>单击<b>新增参数</b>，在弹窗对话框中添加服务出参。</p> <p>当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。</p> <div>  <b>说明：</b> <ul style="list-style-type: none"> <li>- 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。</li> <li>- 您可以使用某个属性作为出参，也可以自定义参数，例如将已定义的属性<b>土壤湿度</b>作为出参，则云端调用自动喷灌服务时，将返回当前土壤湿度的数据。</li> <li>- 一个服务最多支持定义20个出参。</li> </ul> </div>
扩展描述	<p>子设备接入网关协议为自定义协议或OPC UA时，需增加扩展描述，实现设备通信协议到标准物模型的映射关系。</p> <p>当接入网关协议为自定义时，需传入JSON格式的自定义配置，长度不超过1024字符。</p> <p>当接入网关协议为OPC UA时，设置节点名称。节点名称需保证服务维度下唯一。</p>
描述	输入文字，对该服务功能进行说明或备注。长度限制为100字。

- 自定义事件：在**添加自定义功能**对话框，选择功能类型为**事件**。设置参数完成后，单击**确认**。



**说明：**

接入网关的协议选择为Modbus时，不支持定义事件。

属性

服务

事件

\* 功能名称 ?

报警事件

\* 标识符 ?

alarmEvent

\* 事件类型 ?

☐ 信息

☒ 告警

☐ 故障

输出参数

+

参数名称: 报警类型

编辑 | 删除


+增加参数



描述

请输入描述

0/100

事件参数设置说明如下表。

参数	描述
功能名称	<div>事件的名称。</div> <div>支持中文、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。</div> <div><div> <b>说明：</b></div><div>当接入网关协议为Modbus时，不支持自定义事件。</div></div>

参数	描述
标识符	<p>事件唯一标识符，在产品下具有唯一性。即Alink JSON格式中该事件的 <b>identifier</b> 的值，作为设备上报该事件数据的Key，例如 <b>ErrorCode</b>。</p> <div>  <b>说明：</b>            不能用以下系统保留参数作为标识符：            set、get、post、time、value。         </div>
事件类型	<ul style="list-style-type: none"> <li>- 信息：指设备上报的一般性通知，例如完成某项任务等。</li> <li>- 告警：设备运行过程中主动上报的突发或异常情况，告警类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。</li> <li>- 故障：设备运行过程中主动上报的突发或异常情况，故障类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。</li> </ul>
输出参数	<p>该事件的出参。单击<b>增加参数</b>，在弹窗对话框中添加一个服务出参。您可以使用某个属性作为出参，也可以自定义参数。例如将已定义的属性<b>电压</b>作为出参，则设备上报该故障事件时，将携带当前设备的电压值，用于进一步判断故障原因。</p> <p>当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。</p> <div>  <b>说明：</b> <ul style="list-style-type: none"> <li>- 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。</li> <li>- 一个事件最多支持定义50个出参。</li> </ul> </div>
扩展描述	<p>子设备接入网关协议为自定义协议或OPC UA时，需增加扩展描述，实现设备通信协议到标准物模型的映射关系。</p> <p>当接入网关协议为自定义时，需传入JSON格式的自定义配置，长度不超过1024字符。</p> <p>当接入网关协议为OPC UA时，设置节点名称。节点名称需保证事件维度下唯一。</p>
描述	输入文字，对该事件功能进行说明或备注。长度限制为100字。



## 8. 发布物模型。

- a) 单击页面左下方的**发布上线**按钮。
- b) 在**发布物模型**对话框中，输入版本号和版本描述，单击**确定**。

参数	说明
版本号	设置当前物模型版本号。后期可根据版本号管理物模型。 版本号支持英文字母、数字和点号（.），长度限制 1~16 个字符。
版本描述	描述当前版本物模型。支持中文汉字、英文字母、数字和特殊符号。 长度限制为100个字符。一个中文汉字算一个字符。



### 说明：

- 发布后，物模型才会正式生效。
- 同一物模型最多保存最近的10个版本，多余的历史版本将被覆盖。

## 预期结果

物模型发布后，物联网平台为该产品生成正式版本的物模型。在**产品详情**页的**功能定义**页签下，您可以：

- 单击**物模型TSL**，查看JSON格式的物模型TSL。
- 单击**生成设备端代码**，下载物联网平台根据您定义的功能生成的设备端代码，用于设备端物模型功能开发。

## 2.3 批量添加物模型

在物联网平台控制台，通过导入物模型来批量添加属性、事件和服务，即将已编辑好的物模型JSON文件或其他产品的物模型导入为当前产品的物模型。

### 使用说明

- 导入物模型后，会覆盖该产品原有的功能定义。请谨慎使用。
- 设备协议为Modbus的产品，不支持导入物模型。
- 如果产品已发布，不能新增或编辑物模型。如需新增或编辑物模型，需先撤销产品发布。
- 如果导入其他产品的物模型，产品的**所属品类**必须相同。

### 导入物模型

- 登录[物联网平台控制台](#)。
- 在左侧导航栏中，选择**设备管理 > 产品**。

3. 在**产品管理**页的产品列表中，单击产品对应的**查看**操作按钮。
4. 在**产品详情**页，单击**功能定义 > 编辑草稿**。
5. 单击**快速导入**，然后在弹出的对话框中导入物模型。

导入物模型

注：导入的物模型会覆盖原来的功能。

拷贝产品

导入物模型

\* 选择产品

TestProduct

\* 选择版本

1578281903640 (正式版本)

确定

取消

可通过以下两种方法导入物模型。

- **拷贝产品**：选择物模型源产品和物模型版本，单击**确定**，即可将物模型导入到此产品中。

物模型导入后，如果需要修改某些功能，请在**功能定义**页签下，单击**编辑功能**，再单击功能对应的**编辑**按钮，即可修改该功能定义。

- **导入物模型**：上传物模型JSON文件，单击**确定**。

物模型文件编写，可参见[什么是物模型](#)中的物模型格式。



#### 说明：

- 物模型文件大小不能超过256 KB。
- 物模型文件中，参数**productKey**的值必须是当前产品的ProductKey。

## 6. 发布物模型。

- a) 单击页面左下方的**发布上线**按钮。
- b) 在**发布物模型**对话框中，输入版本号和版本描述，单击**确定**。

参数	说明
版本号	设置当前物模型版本号。后期可根据版本号管理物模型。 版本号支持英文字母、数字和点号（.），长度限制 1~16 个字符。
版本描述	描述当前版本物模型。支持中文汉字、英文字母、数字和特殊符号。 长度限制为100个字符。一个中文汉字算一个字符。



### 说明：

- 发布后，物模型才会正式生效。
- 同一物模型最多保存最近的10个版本，多余的历史版本将被覆盖。

## 3 数据解析

### 3.1 什么是数据解析

物联网平台定义的标准数据格式为Alink JSON。但是低配置且资源受限或者对网络流量有要求的设备，不适合直接构造JSON数据与物联网平台通信，可将原数据透传到物联网平台。物联网平台提供数据解析功能，可以根据您提交的脚本，将数据在设备自定义格式和JSON格式之间转换。

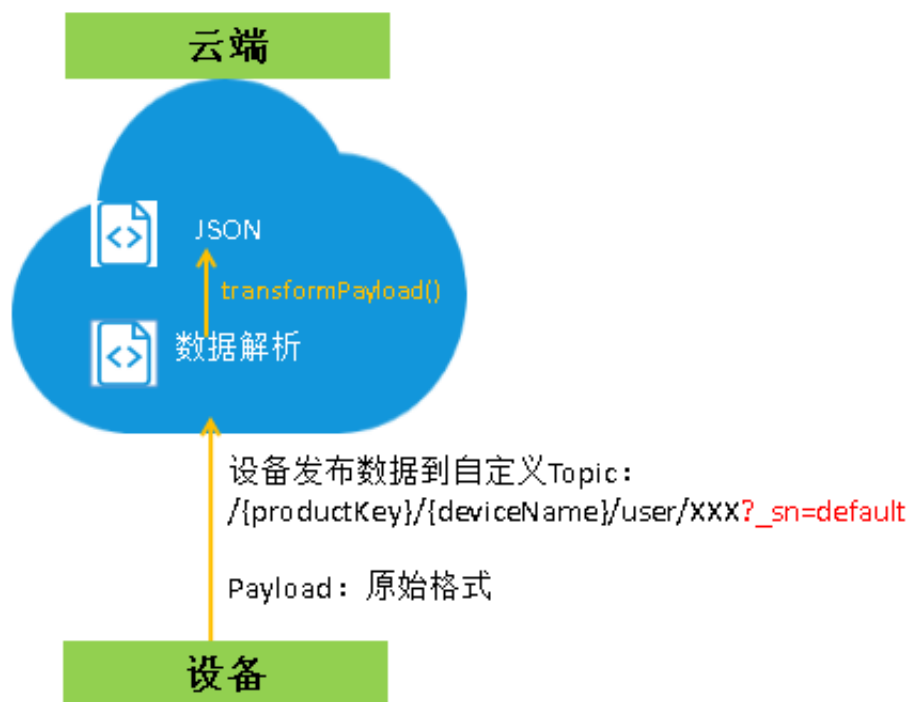
目前支持解析两类数据：

- 自定义Topic上行数据，即将设备通过自定义Topic上报给云端的自定义格式数据Payload解析为JSON格式。
- 上、下行物模型Topic的数据，即将设备上报给云端的自定义格式物模型数据解析为Alink JSON格式，和将云端下发的Alink JSON格式数据解析为设备自定义的格式。

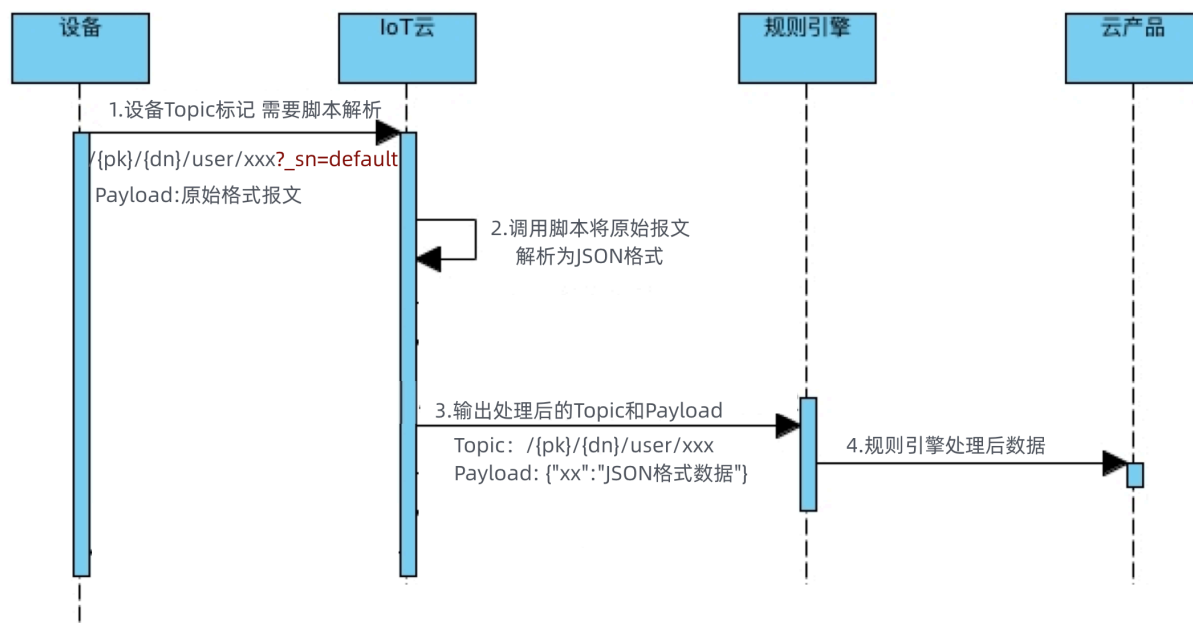
#### 自定义Topic数据解析

设备通过自定义Topic发布数据，且Topic携带解析标记（?\_sn=default）时，物联网平台接收数据后，先调用您在控制台提交的数据解析脚本，将设备上报的自定义格式数据的Payload解析为JSON结构体，再进行业务处理。

数据解析流程图：



设备上报自定义Topic的数据（上行数据）全流程图：



自定义Topic数据解析脚本编写方法，请参见：

[概述](#)

[JavaScript脚本示例](#)

[Python脚本示例](#)

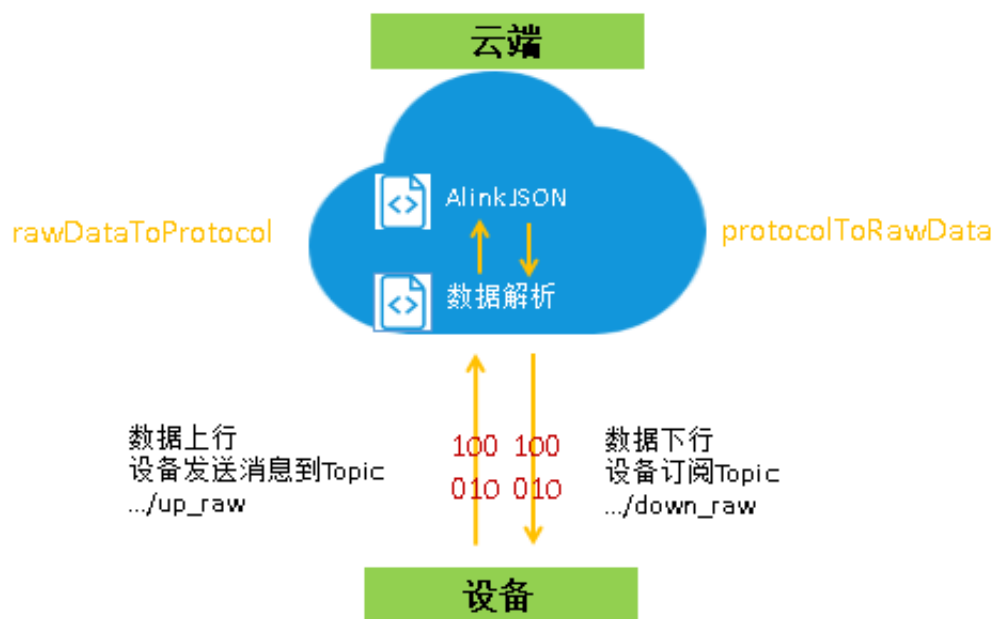
[PHP脚本示例](#)

## 物模型数据解析

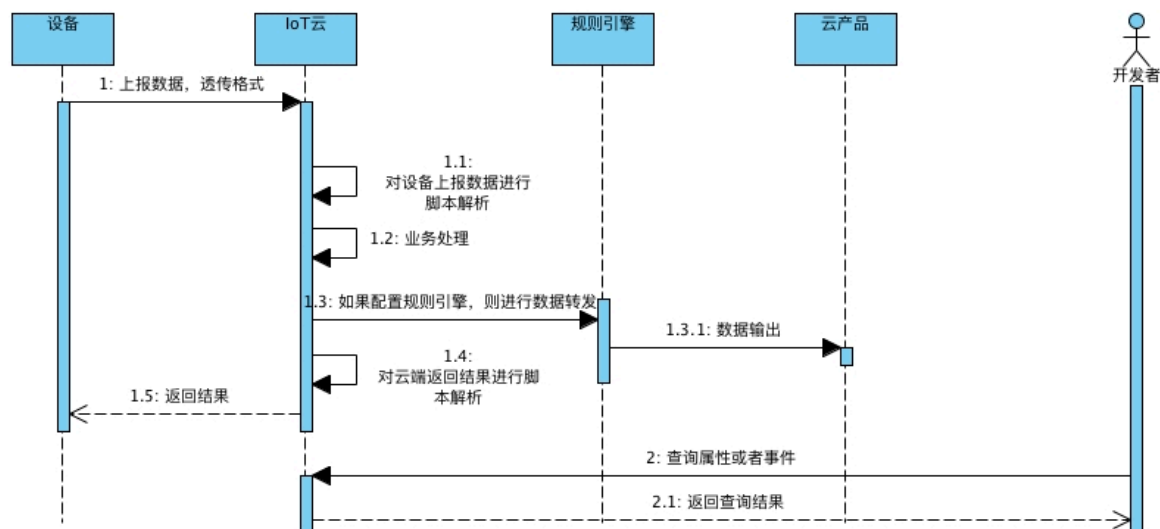
**数据格式为透传/自定义**的产品下的设备与云端进行物模型数据通信时，需要物联网平台调用您提交的数据解析脚本，将上、下行物模型数据分别解析为物联网平台定义的标准格式（Alink JSON）和设备的自定义数据格式。

物联网平台接收到来自设备的数据时，先运行解析脚本，将透传的数据转换成Alink JSON格式的数据，再进行业务处理；物联网平台下发数据给设备前，也会先通过脚本将数据转换为设备的自定义格式，再下发给设备。

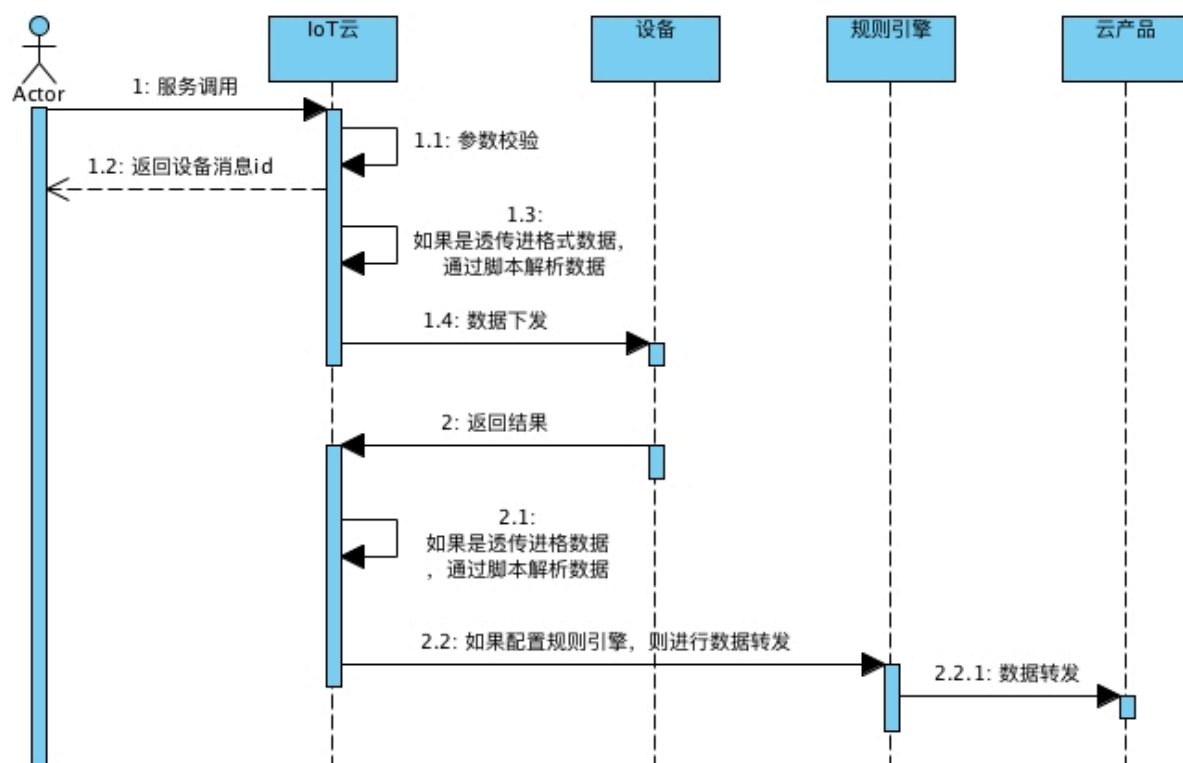
数据解析流程图：



设备上报透传格式的属性或事件（上行数据）全流程图：



调用设备服务或设置属性（下行数据）全流程图：



物模型数据解析脚本示例，请参见：

脚本编辑示例，请参见[物模型数据解析使用示例](#)、[JavaScript脚本示例](#)、[Python脚本示例](#)和[PHP脚本示例](#)。

若您的设备为LoRaWAN节点设备，请参见[LoRaWAN设备数据解析](#)。

若提交的脚本不能正常解析数据，请参见[问题排查](#)。

## 3.2 自定义Topic数据解析

### 3.2.1 概述

设备通过携带解析标记（?\_sn=default）的自定义Topic上报数据，物联网平台收到数据后，调用您在控制台提交的数据解析脚本，将自定义格式数据转换为JSON结构体，再流转给后续业务系统。

#### 说明

- 目前仅华东2（上海）地域支持自定义Topic数据解析。

- 配置设备端时，需在发布消息的自定义Topic后添加数据解析标记（?\_sn=default）。物联网平台仅解析设备通过携带标记的Topic发布的数据。

例如，设备发送到Topic `/${productKey}/${deviceName}/user/update`的数据需要解析为JSON格式。在开发设备端时，就需配置该Topic为：`/${productKey}/${deviceName}/user/update?_sn=default`。

**说明：**

在物联网平台创建自定义Topic时按正常Topic定义，不添加该解析标记。

- 仅解析设备上报云端的数据，不解析云端下行数据。
- 仅解析上报数据的Payload，并返回解析后的Payload。
- 解析前后，数据所在Topic不变。例如，设备发送到`/${productKey}/${deviceName}/user/update`的数据，解析后仍在该Topic中。

**提交数据解析脚本**

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 产品**。
3. 在**产品**页，单击产品对应的**查看**。
4. 在**产品详情页**，选择**数据解析**页签。
5. 选择脚本语言，然后在**编辑脚本**下的输入框中输入脚本。

目前支持三种脚本语言：JavaScript（ECMAScript 5）、Python 2.7和PHP 7.2。

脚本中需定义调用函数：

- JavaScript（ECMAScript 5）：**transformPayload()**
- Python 2.7：**transform\_payload()**
- PHP 7.2：**transformPayload()**

完整的示例代码，请参见[JavaScript脚本示例](#)、[Python脚本示例](#)和[PHP脚本示例](#)。

**说明：**

如果产品的**数据格式**为**透传/自定义**，还需编写物模型数据解析脚本。物模型数据解析脚本编写指导，请参见[物模型数据解析使用示例](#)。

6. 测试脚本。
  - a) **模拟输入**下，选择**模拟类型**为**自定义**，并选择设备和Topic。
  - b) 输入模拟的设备上报数据，单击**执行**。
7. 确认脚本可用后，单击**提交**，将脚本提交到物联网平台系统。



### 3.2.2 JavaScript脚本示例

本文提供JavaScript语言的自定义Topic数据解析脚本模板和示例。

#### 说明

- 目前仅华东2（上海）地域支持自定义Topic数据解析。
- 配置设备端时，需在发布消息的自定义Topic后添加数据解析标记（?\_sn=default）。物联网平台仅解析设备通过携带标记的Topic发布的数据。

例如，设备发送到Topic `/${productKey}/${deviceName}/user/update`的数据需要解析为JSON格式。在开发设备端时，就需配置该Topic为：`/${productKey}/${deviceName}/user/update?_sn=default`。



#### 说明：

在物联网平台创建自定义Topic时按正常Topic定义，不添加该解析标记。

- 仅解析设备上报云端的数据，不解析云端下行数据。
- 仅解析上报数据的Payload，并返回解析后的Payload。
- 解析前后，数据所在Topic不变。例如，设备发送到`/${productKey}/${deviceName}/user/update`的数据，解析后仍在该Topic中。

#### 脚本模板

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' //自定义Topic: /user/update
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' //自定义Topic: /user/update/error
/**
 * 将设备自定义Topic数据转换为JSON格式数据, 设备上报数据到物联网平台时调用
 * 入参: topic 字符串, 设备上报消息的Topic
 * 入参: rawData byte[]数组, 不能为空
 * 出参: jsonObj 对象, 不能为空
 */
function transformPayload(topic, rawData) {
    var jsonObj = {};
    return jsonObj;
}
```

#### 示例脚本



#### 说明：

如果产品的数据格式为透传/自定义，还需编写物模型数据解析脚本。物模型数据解析脚本编写指导，请参见[物模型数据解析使用示例](#)。

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' //自定义Topic: /user/update
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' //自定义Topic: /user/update/error
/*
 示例数据
*/
```

```

自定义Topic: /user/update上报数据
输入参数: topic: /{productKey}/{deviceName}/user/update和bytes: 0x00000000
0100320100000000
输出参数:
{
  "prop_float": 0,
  "prop_int16": 50,
  "prop_bool": 1,
  "topic": "/{productKey}/{deviceName}/user/update"
}
*/
function transformPayload(topic, bytes) {
  var uint8Array = new Uint8Array(bytes.length);
  for (var i = 0; i < bytes.length; i++) {
    uint8Array[i] = bytes[i] & 0xff;
  }
  var dataView = new DataView(uint8Array.buffer, 0);
  var jsonMap = {};

  if(topic.includes(SELF_DEFINE_TOPIC_ERROR_FLAG)) {
    jsonMap['topic'] = topic;
    jsonMap['errorCode'] = dataView.getInt8(0)
  } else if (topic.includes(SELF_DEFINE_TOPIC_UPDATE_FLAG)) {
    jsonMap['topic'] = topic;
    jsonMap['prop_int16'] = dataView.getInt16(5);
    jsonMap['prop_bool'] = uint8Array[7];
    jsonMap['prop_float'] = dataView.getFloat32(8);
  }

  return jsonMap;
}

```

### 3.2.3 Python脚本示例

本文提供Python语言的自定义Topic数据解析脚本模板和示例。

#### 说明

- 目前仅华东2（上海）地域支持自定义Topic数据解析。
- 配置设备端时，需在发布消息的自定义Topic后添加数据解析标记（?\_sn=default）。物联网平台仅解析设备通过携带标记的Topic发布的数据。

例如，设备发送到Topic `/${productKey}/${deviceName}/user/update`的数据需要解析为JSON格式。在开发设备端时，就需配置该Topic为：`/${productKey}/${deviceName}/user/update?_sn=default`。



#### 说明：

在物联网平台创建自定义Topic时按正常Topic定义，不添加该解析标记。

- 仅解析设备上报云端的数据，不解析云端下行数据。
- 仅解析上报数据的Payload，并返回解析后的Payload。
- 解析前后，数据所在Topic不变。例如，设备发送到`/${productKey}/${deviceName}/user/update`的数据，解析后仍在该Topic中。

## 脚本模板

```
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' #自定义Topic: /user/update
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' #自定义Topic: /user/update/
error

# 将设备自定义Topic数据转换为JSON格式数据, 设备上报数据到物联网平台时调用
# 入参: topic 字符串, 设备上报消息的Topic
# 入参: rawData 列表, 列表元素取值为int类型, 不能为空
# 出参: jsonObj 字典
def transform_payload(topic, rawData):
    jsonObj = {}
    return jsonObj
```

## 脚本示例



### 说明:

以下示例脚本仅用于解析自定义Topic数据。如果产品的数据格式为透传/自定义, 还需编写物模型数据解析脚本。物模型数据解析脚本编写指导, 请参见[物模型数据解析使用示例](#)。

```
# coding=UTF-8
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' #自定义Topic: /user/update
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' #自定义Topic: /user/update/
error

# 示例数据
# 自定义Topic: /user/update上报数据
# 输入参数: topic: /{productKey}/{deviceName}/user/update和bytes: 0x00000000
0100320100000000
# 输出参数:
# {
#   "prop_float": 0,
#   "prop_int16": 50,
#   "prop_bool": 1,
#   "topic": "/{productKey}/{deviceName}/user/update"
# }
def transform_payload(topic, bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)

    jsonMap = {}
    if SELF_DEFINE_TOPIC_ERROR_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['errorCode'] = bytes_to_int(uint8Array[0:1])

    elif SELF_DEFINE_TOPIC_UPDATE_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['prop_int16'] = bytes_to_int(uint8Array[5:7])
        jsonMap['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        jsonMap['prop_float'] = bytes_to_int(uint8Array[8:])

    return jsonMap

# byte数组转换为整型
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
```

```
return int("".join(data), 16)
```

### 3.2.4 PHP脚本示例

本文提供PHP语言的自定义Topic数据解析脚本模板和示例。

#### 说明

- 目前仅华东2（上海）地域支持自定义Topic数据解析。
- 配置设备端时，需在发布消息的自定义Topic后添加数据解析标记（?\_sn=default）。物联网平台仅解析设备通过携带标记的Topic发布的数据。

例如，设备发送到Topic `/${productKey}/${deviceName}/user/update`的数据需要解析为JSON格式。在开发设备端时，就需配置该Topic为：`/${productKey}/${deviceName}/user/update?_sn=default`。



#### 说明：

在物联网平台创建自定义Topic时按正常Topic定义，不添加该解析标记。

- 仅解析设备上报云端的数据，不解析云端下行数据。
- 仅解析上报数据的Payload，并返回解析后的Payload。
- 解析前后，数据所在Topic不变。例如，设备发送到`/${productKey}/${deviceName}/user/update`的数据，解析后仍在该Topic中。

#### 脚本模板

PHP脚本模版，您可以基于以下模版编写数据解析脚本。

```
<?php
/**
 * 将设备自定义Topic数据转换为JSON格式数据, 设备上报数据到物联网平台时调用
 * 入参: $topic 字符串, 设备上报消息的Topic
 * 入参: $rawData 普通数组, 数组元素为整数
 * 出参: $jsonObj 关联数组, 关联数组key取值为英文字符串不能是字符的数字如"10", 不能为空
 */
function transformPayload($topic, $rawData)
{
    $jsonObj = array();
    return $jsonObj;
}
```

#### 脚本编写注意事项

- 请避免使用全局变量或者static变量，否则会造成执行结果不一致。
- 脚本中，处理数据采用补码的方式，`[-128, 127]`补码范围为`[0, 255]`。例如，`-1`对应的补码为`255`（10进制表示）。

- 自定义协议解析的函数（transformPayload）的入参为整形数组。需要通过0xFF进行与操作，获取其对应的补码。返回结果为关联数组，要求key取值包含非数组字符（如数组key为“10”，PHP数组中会获取到整数10）。
- PHP执行环境对于异常处理会很严格，如发生错误会直接抛出异常，后续代码不会执行。保证代码的健壮性，对于异常需要捕获并进行处理。

## 示例脚本



### 说明：

如果产品的数据格式为透传/自定义，还需编写物模型数据解析脚本。物模型数据解析脚本编写指导，请参见[物模型数据解析使用示例](#)。

```
<?php
/*
  示例数据
  自定义Topic: /user/update上报数据
  输入参数: topic: /{productKey}/{deviceName}/user/update和bytes: 0x00000000
  0100320100000000
  输出参数:
  {
    "prop_float": 0,
    "prop_int16": 50,
    "prop_bool": 1,
    "topic": "/{productKey}/{deviceName}/user/update"
  }
*/
function transformPayload($topic, $bytes)
{
    $data = array();
    $length = count($bytes);
    for ($i = 0; $i < $length; $i++) {
        $data[$i] = $bytes[$i] & 0xff;
    }

    $jsonMap = array();

    if (strpos($topic, '/user/update/error') !== false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['errorCode'] = getInt8($data, 0);
    } else if (strpos($topic, '/user/update') !== false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['prop_int16'] = getInt16($data, 5);
        $jsonMap['prop_bool'] = $data[7];
    }

    return $jsonMap;
}

function getInt32($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1], $bytes[$index + 2], $bytes[$index + 3]);
    return hexdec(byteArrayToHexString($array));
}
```

```
function getInt16($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1]);

    return hexdec(byteArrayToHexString($array));
}

function getInt8($bytes, $index)
{
    $array = array($bytes[$index]);
    return hexdec(byteArrayToHexString($array));
}

function byteArrayToHexString($data)
{
    $hexStr = "";
    for ($i = 0; $i < count($data); $i++) {
        $hexValue = dechex($data[$i]);

        $tempHexStr = strval($hexValue);

        if (strlen($tempHexStr) === 1) {
            $hexStr = $hexStr . '0' . $tempHexStr;
        } else {
            $hexStr = $hexStr . $tempHexStr;
        }
    }

    return $hexStr;
}

function hexStringToByteArray($hex)
{
    $result = array();
    $index = 0;
    for ($i = 0; $i < strlen($hex) - 1; $i += 2) {
        $result[$index++] = hexdec($hex[$i] . $hex[$i + 1]);
    }
    return $result;
}

function concat($array, $data)
{
    return array_merge($array, $data);
}

function toHex($data)
{
    $var = dechex($data);
    $length = strlen($var);
    if ($length % 2 == 1) {
        $var = '0' . $var;
    }
    return $var;
}
```

```
}
```

## 3.3 物模型数据解析

### 3.3.1 物模型数据解析使用示例

本文以解析上、下行属性数据的脚本为例，介绍为**数据格式为透传/自定义**的产品下的物模型数据解析脚本编写方法。。

#### 步骤一：编辑脚本

1. 在[物联网平台控制台](#)，创建产品。

创建产品操作指导，请参见[#unique\\_5](#)。



**说明：**

**数据格式**选择为**透传/自定义**。

2. 为该产品定义物模型。请参见[单个添加物模型](#)中，自定义属性的操作说明。

本示例中定义了以下三个属性。

标识符 (identifer)	数据类型	取值范围	读写类型
prop_float	浮点单精度 float	-100~100	读写
prop_int16	整数型 int32	-100~100	读写
prop_bool	布尔型 bool	0: 开; 1: 关	读写

数据解析脚本将根据这里定义的物模型来编写，用于解析上下行物模型数据。

3. 在设备通信协议中做如下定义。

**表 3-1: 设备上报数据请求**

字段	字节数
帧类型	1字节
请求ID	4字节
属性prop_int16	2字节
属性prop_bool	1字节

字段	字节数
属性prop_float	4字节

表 3-2: 设备上报数据响应

字段	字节数
帧类型	1字节
请求ID	4字节
结果code	1字节

表 3-3: 设置属性请求

字段	字节数
帧类型	1字节
请求ID	4字节
属性prop_int16	2字节
属性prop_bool	1字节
属性prop_float	4字节

表 3-4: 属性设置响应

字段	字节数
帧类型	1字节
请求ID	4字节
结果code	1字节



#### 4. 编写脚本。

- 在**产品**页，单击产品对应的**查看**。
- 在**产品详情页**，选择**数据解析**页签。
- 选择脚本语言，然后在**编辑脚本**下的输入框中输入脚本。

目前支持两种脚本语言：JavaScript（ECMAScript 5）和Python 2.7。

脚本中需定义调用以下两个函数，分别用于解析上、下行物模型数据。

- 将Alink JSON格式数据转为设备自定义数据格式的函数：
  - JavaScript（ECMAScript 5）：**protocolToRawData**
  - Python 2.7：**protocol\_to\_raw\_data**
- 将设备自定义数据格式转Alink JSON格式数据的函数：
  - JavaScript（ECMAScript 5）：**rawDataToProtocol**
  - Python 2.7：**raw\_data\_to\_protocol**

完整的示例脚本Demo，请参见[JavaScript脚本示例](#)和[Python脚本示例](#)。



##### 说明：

您还需编写自定义Topic的上行数据解析脚本，相关脚本编写说明，请参见[概述](#)。

包含自定义Topic数据解析和物模型数据解析的完整示例脚本，请参见[#unique\\_33](#)和[#unique\\_34](#)。

#### 步骤二：在线测试脚本

脚本编辑完成后，在**模拟输入**下，选择**模拟类型**，输入模拟数据在线测试脚本。

- 模拟解析设备上报的属性数据。

选择模拟类型为**设备上报数据**，输入以下模拟的设备上报数据，然后单击**执行**。

```
0x00002233441232013fa00000
```

数据解析引擎会按照脚本规则，将透传数据转换为JSON格式数据。

单击**运行结果**，查看解析结果。

```
{
  "method": "thing.event.property.post",
  "id": "2241348",
  "params": {
    "prop_float": 1.25,
    "prop_int16": 4658,
    "prop_bool": 1
  },
  "version": "1.0"
```

```
}
```

- 模拟解析物联网平台下发的返回结果数据。

选择模拟类型为**设备接收数据**，输入以下JSON格式数据，然后单击**执行**。

```
{
  "id": "12345",
  "version": "1.0",
  "code": 200,
  "method": "thing.event.property.post",
  "data": {}
}
```

数据解析引擎会将JSON格式数据转换为以下数据。

```
0x0200003039c8
```

- 模拟解析物联网平台下发的属性设置数据。

选择模拟类型为**设备接收数据**，输入以下JSON格式数据，然后单击**执行**。

```
{
  "method": "thing.service.property.set",
  "id": "12345",
  "version": "1.0",
  "params": {
    "prop_float": 123.452,
    "prop_int16": 333,
    "prop_bool": 1
  }
}
```

数据解析引擎会将JSON格式数据转换为以下数据。

```
0x0100003039014d0142f6e76d
```

- 模拟解析设备返回的属性设置结果数据。

选择模拟类型为**设备上报数据**，输入以下数据，然后单击**执行**。

```
0x0300223344c8
```

数据解析引擎会将透传数据转换为以下JSON格式数据。

```
{
  "code": "200",
  "data": {},
  "id": "2241348",
  "version": "1.0"
}
```

### 步骤三：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平台系统，以供数据上下行时，物联网平台调用该脚本解析数据。

**说明：**

仅提交后的脚本才能被物联网平台调用；草稿状态的脚本不能被调用。

产品信息

Topic类列表

功能定义

数据解析

服务端订阅

编辑脚本 (当前展示为：脚本草稿) 脚本语言：JavaScript (ECMAScript 5)

```
89      payloadArray = payloadArray.concat(buffer_uint8(value));
90    }
91    return payloadArray;
92  }
93  //以下是部分辅助函数
94  function buffer_uint8(value) {
95    var uint8Array = new Uint8Array(1);
96    var dv = new DataView(uint8Array.buffer, 0);
97    dv.setUint8(0, value);
98    return [].slice.call(uint8Array);
99  }
100 function buffer_int16(value) {
101   var uint8Array = new Uint8Array(2);
102   var dv = new DataView(uint8Array.buffer, 0);
103   dv.setInt16(0, value);
104   return [].slice.call(uint8Array);
105 }
```

提交

▶ 执行

📁 保存

**步骤四：使用真实设备调试**

正式使用脚本之前，请使用真实设备与物联网平台进行上下行消息通信，以验证物联网平台能顺利调用脚本，解析上下行数据。

- 测试上报属性数据。

- 使用设备端上报设备属性数据，例如0x00002233441232013fa00000。
- 在物联网平台控制台，选择**设备管理 > 设备**。
- 单击设备对应的**查看**，然后在**设备详情页运行状态页**签下，查看是否有相应的属性数据。

- 测试下发属性数据。

- 在物联网平台控制台，选择**监控运维 > 在线调试**。
- 选择要调试的产品和设备，并选择**调试真实设备**，功能选择为要调试的属性identifier，如属性（prop\_int16），方法选择为**设置**，输入以下数据，单击**发送指令**。

```
{
  "method": "thing.service.property.set",
  "id": "12345",
  "version": "1.0",
  "params": {
    "prop_float": 123.452,
    "prop_int16": 333,
    "prop_bool": 1
  }
}
```

- 查看设备端是否收到该属性设置指令。
- 在该设备的**设备详情页运行状态页**签下，查看设备是否上报当前属性数据。

## 相关文档

- 查看JavaScript (ECMAScript 5) 脚本模板和示例, 请参见[JavaScript脚本示例](#)。
- 查看Python 2.7脚本模板和示例, 请参见[Python脚本示例](#)
- 了解数据解析流程等基本信息, 请参见[什么是数据解析](#)。
- 关于数据解析问题排查, 请参见[问题排查](#)。
- 有关自定义Topic数据解析说明, 请参见[概述](#)。

### 3.3.2 JavaScript脚本示例

本文提供JavaScript语言的物模型数据解析脚本模板和示例。

#### 脚本模板

以下为JavaScript脚本模版, 您可以基于以下模版编写物模型数据解析脚本。



#### 说明:

本模板仅适用于**数据格式为透传/自定义**的产品。

```
/**
 * 将Alink协议的数据转换为设备能识别的格式数据, 物联网平台给设备下发数据时调用
 * 入参: jsonObj 对象, 不能为空
 * 出参: rawData byte[]数组, 不能为空
 */
function protocolToRawData(jsonObj) {
    return rawData;
}

/**
 * 将设备的自定义格式数据转换为Alink协议的数据, 设备上报数据到物联网平台时调用
 * 入参: rawData byte[]数组, 不能为空
 * 出参: jsonObj 对象, 不能为空
 */
function rawDataToProtocol(rawData) {
    return jsonObj;
}
```

#### 脚本编写注意事项

- 请避免使用全局变量, 否则会造成执行结果不一致。
- 脚本中, 处理数据采用补码的方式,  $[-128, 127]$  补码范围为 $[0, 255]$ 。例如,  $-1$ 对应的补码为255 (10进制表示)。
- 解析设备上报数据的函数 (rawDataToProtocol) 的入参为整形数组。需要通过0xFF进行与操作, 获取其对应的补码。
- 解析物联网平台下发数据的函数 (protocolToRawData) 的返回结果为数组。数组元素为整形, 取值为 $[0, 255]$ 。

## 脚本示例

以下是基于[物模型数据解析使用示例](#)中定义的属性和通信协议编写的脚本。

```
var COMMAND_REPORT = 0x00; //属性上报
var COMMAND_SET = 0x01; //属性设置
var COMMAND_REPORT_REPLY = 0x02; //上报数据返回结果
var COMMAND_SET_REPLY = 0x03; //属性设置设备返回结果
var COMMAND_UNKNOWN = 0xff; //未知的命令
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //物联网平台Topic, 设备
上传属性数据到云端
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'; //物联网平台Topic, 云端下发
属性控制指令到设备端
var ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set'; //物联网平台Topic, 设
设备上报属性设置的结果到云端
/*
示例数据:
设备上报数据
传入参数 ->
    0x000000000100320100000000
输出结果 ->
    {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":50
    ,"prop_bool":1},"version":"1.0"}

属性设置的返回结果
传入参数 ->
    0x0300223344c8
输出结果 ->
    {"code":"200","data":{},"id":"2241348","version":"1.0"}
*/
function rawDataToProtocol(bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = new Object();
    var fHead = uint8Array[0]; // command
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD; //ALink JSON格式 - 属性上报
        topic
        jsonMap['version'] = '1.0'; //ALink JSON格式 - 协议版本号固定字段
        jsonMap['id'] = '' + dataView.getInt32(1); //ALink JSON格式 - 标示该次请求id值
        var params = {};
        params['prop_int16'] = dataView.getInt16(5); //对应产品属性中 prop_int16
        params['prop_bool'] = uint8Array[7]; //对应产品属性中 prop_bool
        params['prop_float'] = dataView.getFloat32(8); //对应产品属性中 prop_float
        jsonMap['params'] = params; //ALink JSON格式 - params标准字段
    } else if (fHead == COMMAND_SET_REPLY) {
        jsonMap['version'] = '1.0'; //ALink JSON格式 - 协议版本号固定字段
        jsonMap['id'] = '' + dataView.getInt32(1); //ALink JSON格式 - 标示该次请求id值
        jsonMap['code'] = '' + dataView.getUint8(5);
        jsonMap['data'] = {};
    }
    return jsonMap;
}
/*
示例数据:
属性设置
传入参数 ->
    {"method":"thing.service.property.set","id":"12345","version":"1.0","params":{"
    prop_float":123.452, "prop_int16":333, "prop_bool":1}}
```

输出结果 ->

0x0100003039014d0142f6e76d

设备上报的返回结果

传入数据 ->

```
{"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{}}
```

输出结果 ->

0x0200003039c8

\*/

```
function protocolToRawData(json) {
  var method = json['method'];
  var id = json['id'];
  var version = json['version'];
  var payloadArray = [];
  if (method == ALINK_PROP_SET_METHOD) // 属性设置
  {
    var params = json['params'];
    var prop_float = params['prop_float'];
    var prop_int16 = params['prop_int16'];
    var prop_bool = params['prop_bool'];
    //按照自定义协议格式拼接 rawData
    payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET)); // command字
    payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // ALink JSON格式 '
    payloadArray = payloadArray.concat(buffer_int16(prop_int16)); // 属性'prop_int16'的
    payloadArray = payloadArray.concat(buffer_uint8(prop_bool)); // 属性'prop_bool'的
    payloadArray = payloadArray.concat(buffer_float32(prop_float)); // 属性'prop_float
  } else if (method == ALINK_PROP_REPORT_METHOD) { //设备上报数据返回结果
    var code = json['code'];
    payloadArray = payloadArray.concat(buffer_uint8(COMMAND_REPORT_REPLY)); //
    payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // ALink JSON格式 '
    payloadArray = payloadArray.concat(buffer_uint8(code));
  } else { //未知命令, 对于有些命令不做处理
    var code = json['code'];
    payloadArray = payloadArray.concat(buffer_uint8(COMMAND_UNKOWN)); //command
    payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // ALink JSON格式 '
    payloadArray = payloadArray.concat(buffer_uint8(code));
  }
  return payloadArray;
}

//以下是部分辅助函数
function buffer_uint8(value) {
  var uint8Array = new Uint8Array(1);
  var dv = new DataView(uint8Array.buffer, 0);
  dv.setUint8(0, value);
  return [].slice.call(uint8Array);
}
function buffer_int16(value) {
  var uint8Array = new Uint8Array(2);
  var dv = new DataView(uint8Array.buffer, 0);
  dv.setInt16(0, value);
  return [].slice.call(uint8Array);
}
function buffer_int32(value) {
  var uint8Array = new Uint8Array(4);
  var dv = new DataView(uint8Array.buffer, 0);
}
```

```
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
```

### 3.3.3 Python脚本示例

本文提供Python语言的物模型数据解析脚本模板和示例。

#### 脚本模板

您可以基于以下Python脚本模版编写数据解析脚本。



#### 说明：

本模板仅适用于**数据格式为透传/自定义**的产品。

```
# 将设备的自定义格式数据转换为Alink协议的数据，设备上报数据到物联网平台时调用
# 入参: rawData 列表，列表元素取值为int类型，不能为空
# 出参: jsonObj 字典，不能为空
def raw_data_to_protocol(rawData):
    jsonObj = {}
    return jsonObj

# 将Alink协议的数据转换为设备能识别的格式数据，物联网平台给设备下发数据时调用
# 入参: jsonData 字典，不能为空
# 出参: rawData 列表，列表元素取值为int类型且大小为[0, 255]之间，不能为空
def protocol_to_raw_data(jsonData):
    rawData = []
    return rawData
```

#### 脚本编写注意事项

- 请避免使用全局变量，否则会造成执行结果不一致。
- 脚本中，处理数据采用补码的方式，[-128, 127]补码范围为[0, 255]。例如，-1对应的补码为255（10进制表示）。
- 解析设备上报数据的函数（raw\_data\_to\_protocol）的入参为整形数组。需要通过0xFF进行与操作，获取其对应的补码。
- 解析物联网平台下发数据的函数（protocol\_to\_raw\_data）的返回结果为数组。数组元素为整形，取值为[0, 255]。

#### 脚本示例

以下是基于[物模型数据解析使用示例](#)中定义的性质和通信协议编写的脚本。

```
# coding=UTF-8
import struct
```

```

COMMAND_REPORT = 0x00 # 属性上报
COMMAND_SET = 0x01 # 属性设置
COMMAND_REPORT_REPLY = 0x02 # //上报数据返回结果
COMMAND_SET_REPLY = 0x03 # //属性设置设备返回结果
COMMAND_UNKNOWN = 0xff # //未知的命令
ALINK_PROP_REPORT_METHOD = 'thing.event.property.post' # 物联网平台Topic, 设备上传
属性数据到云端
ALINK_PROP_SET_METHOD = 'thing.service.property.set' # //物联网平台Topic, 云端下发属
性控制指令到设备端
ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set' # //物联网平台Topic, 设备
上报属性设置的结果到云端

# 示例数据:
# 设备上报数据
# 传入参数 ->
# 0x000000000100320100000000
# 输出结果 ->
# {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":
50,"prop_bool":1},"version":"1.0"}
# 属性设置的返回结果
# 传入参数 ->
# 0x0300223344c8
# 输出结果 ->
# {"code":"200","data":{},"id":"2241348","version":"1.0"}

def raw_data_to_protocol(bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)

    fHead = uint8Array[0]
    jsonMap = {}
    if fHead == COMMAND_REPORT:
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        params = {}
        params['prop_int16'] = bytes_to_int(uint8Array[5:7])
        params['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        params['prop_float'] = bytes_to_int(uint8Array[8:])
        jsonMap['params'] = params
    elif fHead == COMMAND_SET_REPLY:
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        jsonMap['code'] = str(bytes_to_int(uint8Array[5:]))
        jsonMap['data'] = {}

    return jsonMap

# 示例数据:
# 属性设置
# 传入参数 ->
# {"method":"thing.service.property.set","id":"12345","version":"1.0",
# "params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1}}
# 输出结果 ->
# 0x0100003039014d0142f6e76d
# 设备上报的返回结果
# 传入数据 ->
# {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":
{}}
# 输出结果 ->
# 0x0200003039c8

```



```

def protocol_to_raw_data(json):
    method = json.get('method', None)
    id = json.get('id', None)
    version = json.get('version', None)
    payload_array = []

    if method == ALINK_PROP_SET_METHOD:
        params = json.get('params')
        prop_float = params.get('prop_float', None)
        prop_int16 = params.get('prop_int16', None)
        prop_bool = params.get('prop_bool', None)

        payload_array = payload_array + int_8_to_byte(COMMAND_SET)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_16_to_byte(prop_int16)
        payload_array = payload_array + int_8_to_byte(prop_bool)
        payload_array = payload_array + float_to_byte(prop_float)

    elif method == ALINK_PROP_REPORT_METHOD:
        code = json.get('code', None)
        payload_array = payload_array + int_8_to_byte(COMMAND_REPORT_REPLY)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)
    else:
        code = json.get('code')
        payload_array = payload_array + int_8_to_byte(COMMAD_UNKOWN)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)

    return payload_array

# 字节数组转换为整型
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16)

# 字节数组转换为浮点，不带精度
def bytes_to_float(bytes):
    data = []
    for i in bytes:
        t_value = '%02X' % i
        if t_value % 2 != 0:
            t_value += 0
        data.append(t_value)
    hex_str = ''.join(data)
    return struct.unpack('!f', hex_str.decode('hex'))[0]

# 8位整形转成字节数组
def int_8_to_byte(value):
    t_value = '%02X' % value
    if len(t_value) % 2 != 0:
        t_value += '0'

    return hex_string_to_byte_array(t_value)

# 32位整形转成字节数组
def int_32_to_byte(value):
    t_value = '%08X' % value
    if len(t_value) % 2 != 0:
        t_value += '0'

```

```

    return hex_string_to_byte_array(t_value)

# 16位整形转成byte数组
def int_16_to_byte(value):
    t_value = '%04X' % value
    if len(t_value) % 2 != 0:
        t_value += '0'

    return hex_string_to_byte_array(t_value)

# float转成整形数组
def float_to_byte(param):
    return hex_string_to_byte_array(struct.pack(">f", param).encode('hex'))

# 16进制字符串转成byte数组
def hex_string_to_byte_array(str_value):
    if len(str_value) % 2 != 0:
        return None

    cycle = len(str_value) / 2

    pos = 0
    result = []
    for i in range(0, cycle, 1):
        temp_str_value = str_value[pos:pos + 2]
        temp_int_value = int(temp_str_value, base=16)

        result.append(temp_int_value)
        pos += 2
    return result

```

### 3.3.4 PHP脚本示例

本文提供PHP语言的物模型数据解析脚本模板和示例。

#### 脚本模板

PHP脚本模版，您可以基于以下模版编写数据解析脚本。



#### 说明：

本模板仅适用于**数据格式为透传/自定义**的产品。

```

<?php
/**
 * 将Alink协议的数据转换为设备能识别的格式数据，物联网平台给设备下发数据时调用
 * 入参：$jsonObj 关联数组
 * 出参：$rawData 普通数组，数组元素为整数，取值范围为0~255，不能为空
 */
function protocolToRawData($jsonObj)
{
    $rawData = array();
    return $rawData;
}

/**

```

```

* 将设备的自定义格式数据转换为Alink协议的数据，设备上报数据到物联网平台时调用
* 入参：$rawData 普通数组，数组元素为整数
* 出参：$jsonObj 关联数组，关联数组key取值为英文字符串，不能是字符类型的数字如"10"，不能为空
*/
function rawDataToProtocol($rawData)
{
    $jsonObj = array();
    return $jsonObj;
}

/**
* 将设备自定义Topic数据转换为JSON格式数据，设备上报数据到物联网平台时调用
* 入参：$topic 字符串，设备上报消息的Topic
* 入参：$rawData 普通数组，数组元素为整数
* 出参：$jsonObj 关联数组，关联数组key取值为英文字符串，不能是字符的数字如"10"，不能为空
*/
function transformPayload($topic, $rawData)
{
    $jsonObj = array();
    return $jsonObj;
}

```

### 脚本编写注意事项

- 请避免使用全局变量或者static变量，否则会造成执行结果不一致。
- 脚本中，处理数据采用补码的方式，[-128, 127] 补码范围为[0, 255]。例如，-1对应的补码为255（10进制表示）。
- 解析设备上报数据的函数（rawDataToProtocol）的入参为整形数组。需要通过0xFF进行与操作，获取其对应的补码。返回结果为关联数组，要求key取值包含非数组字符（如数组key为“10”，PHP数组中会获取到整数10）。
- 解析物联网平台下发数据的函数（protocolToRawData）的返回结果为数组，要求为PHP普通数组。数组元素为整形，取值范围为0~255。
- 自定义协议解析的函数（transformPayload）的入参为整形数组。需要通过0xFF进行与操作，获取其对应的补码。返回结果为关联数组，要求key取值包含非数组字符（如数组key为“10”，PHP数组中会获取到整数10）。
- PHP执行环境对于异常处理会很严格，如发生错误会直接抛出异常，后续代码不会执行。保证代码的健壮性，对于异常需要捕获并进行处理。

### 脚本示例

以下是基于[物模型数据解析使用示例](#)中定义的性质和通信协议编写的脚本。

```

<?php
/*
示例数据：
设备上报数据
传入参数 ->
    0x00000000001003201
输出结果 ->

```

```

{"method":"thing.event.property.post","id":"1","params":{"prop_int16":50,"prop_bool":1
},"version":"1.0"}

```

属性设置的返回结果

传入参数 ->

0x0300223344c8

输出结果 ->

```

{"code":"200","id":"2241348","version":"1.0"}

```

\*/

```

function rawDataToProtocol($bytes)

```

```

{

```

```

    $data = [];

```

```

    $length = count($bytes);

```

```

    for ($i = 0; $i < $length; $i++) {

```

```

        $data[$i] = $bytes[$i] & 0xff;

```

```

    }

```

```

    $jsonMap = [];

```

```

    $fHead = $data[0]; // command

```

```

    if ($fHead == 0x00) {

```

```

        $jsonMap['method'] = 'thing.event.property.post'; //ALink JSON格式，属性上报topic

```

```

        $jsonMap['version'] = '1.0'; //ALink JSON格式，协议版本号固定字段

```

```

        $jsonMap['id'] = '' . getInt32($data, 1); //ALink JSON格式，标示该次请求id值

```

```

        $params = [];

```

```

        $params['prop_int16'] = getInt16($data, 5); //对应产品属性中 prop_int16

```

```

        $params['prop_bool'] = $data[7]; //对应产品属性中 prop_bool

```

```

        $jsonMap['params'] = $params; //ALink JSON格式，params标准字段

```

```

    } else if ($fHead == 0x03) {

```

```

        $jsonMap['version'] = '1.0'; //ALink JSON格式，协议版本号固定字段

```

```

        $jsonMap['id'] = '' . getInt32($data, 1); //ALink JSON格式，标示该次请求id值

```

```

        $jsonMap['code'] = getInt8($data, 5);

```

```

    }

```

```

    return $jsonMap;

```

```

}

```

/\*

示例数据：

属性设置

传入参数 ->

```

{"method":"thing.service.property.set","id":"12345","version":"1.0","params":{"

```

```

prop_int16":333,"prop_bool":1}}

```

输出结果 ->

0x013039014d01

设备上报的返回结果

传入数据 ->

```

{"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{}}

```

输出结果 ->

0x023039c8

\*/

```

function protocolToRawData($json)

```

```

{

```

```

    $method = $json['method'];

```

```

    $id = $json['id'];

```

```

    $version = $json['version'];

```

```

    $payloadArray = [];

```

```

    if ($method == 'thing.service.property.set') // 属性设置

```

```

    {

```

```

        $params = $json['params'];

```

```

        $prop_int16 = $params['prop_int16'];

```

```

        $prop_bool = $params['prop_bool'];

```

```

        //按照自定义协议格式拼接 rawData

```

```

    $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0x01))); //
command字段
    $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id
))); // ALink JSON格式 'id'
    $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($prop_int16
))); // 属性'prop_int16'的值
    $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($prop_bool
))); // 属性'prop_bool'的值
    } else if ($method == 'thing.event.property.post') { //设备上报数据返回结果
        $code = $json['code'];
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0x02))); //
command字段
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id
))); // ALink JSON格式 'id'
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($code)));
    } else { //未知命令, 对于有些命令不做处理
        $code = $json['code'];
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0xff))); //
command字段
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id
))); // ALink JSON格式 'id'
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($code)));
    }
    return $payloadArray;
}

/*
示例数据
自定义Topic: /user/update上报数据
输入参数: topic: /{productKey}/{deviceName}/user/update和bytes: 0x00000000
0100320100000000
输出参数:
{
    "prop_float": 0,
    "prop_int16": 50,
    "prop_bool": 1,
    "topic": "/{productKey}/{deviceName}/user/update"
}
*/
function transformPayload($topic, $bytes)
{
    $data = array();
    $length = count($bytes);
    for ($i = 0; $i < $length; $i++) {
        $data[$i] = $bytes[$i] & 0xff;
    }

    $jsonMap = array();

    if (strpos($topic, '/user/update/error') !== false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['errorCode'] = getInt8($data, 0);
    } else if (strpos($topic, '/user/update') !== false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['prop_int16'] = getInt16($data, 5);
        $jsonMap['prop_bool'] = $data[7];
    }

    return $jsonMap;
}

function getInt32($bytes, $index)
{

```

```
$array = array($bytes[$index], $bytes[$index + 1], $bytes[$index + 2], $bytes[$index + 3
]);
return hexdec(byteArrayToHexString($array));
}

function getInt16($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1]);
    return hexdec(byteArrayToHexString($array));
}

function getInt8($bytes, $index)
{
    $array = array($bytes[$index]);
    return hexdec(byteArrayToHexString($array));
}

function byteArrayToHexString($data)
{
    $hexStr = '';
    for ($i = 0; $i < count($data); $i++) {
        $hexValue = dechex($data[$i]);

        $tempHexStr = strval($hexValue);

        if (strlen($tempHexStr) === 1) {
            $hexStr = $hexStr . '0' . $tempHexStr;
        } else {
            $hexStr = $hexStr . $tempHexStr;
        }
    }

    return $hexStr;
}

function hexStringToByteArray($hex)
{
    $result = array();
    $index = 0;
    for ($i = 0; $i < strlen($hex) - 1; $i += 2) {
        $result[$index++] = hexdec($hex[$i] . $hex[$i + 1]);
    }
    return $result;
}

function concat($array, $data)
{
    return array_merge($array, $data);
}

function toHex($data)
{
    $var = dechex($data);
    $length = strlen($var);
    if ($length % 2 == 1) {
        $var = '0' . $var;
    }
    return $var;
}
```

```
}
```

## 3.4 LoRaWAN设备数据解析

LoRaWAN设备与物联网平台的通信数据格式为透传/自定义，因此需要使用数据解析脚本，解析上下行数据。本文以LoRaWAN温湿度传感器为例，介绍LoRaWAN设备数据解析脚本的编辑和调试方法。

### 步骤一：编辑脚本

1. 在[物联网平台控制台](#)，创建连网方式为LoRaWAN的产品。
2. 为该产品定义物模型。功能定义具体方法，请参见[单个添加物模型](#)。

本示例中，定义了以下属性、事件和服务。

表 3-5: 属性

标识符	数据类型	取值范围	读写类型
Temperature	int32	-40~55	读写
Humidity	int32	1~100	读写

表 3-6: 服务

标识符	调用方式	输入参数
SetTempHum iThreshold	异步	四个输入参数，数据类型均为int32： <ul style="list-style-type: none"><li>• 温度过高告警阈值（标识符：MaxTemp）</li><li>• 温度过低告警阈值（标识符：MinTemp）</li><li>• 湿度过高告警阈值（标识符：MaxHumi）</li><li>• 湿度过低告警阈值（标识符：MinHumi）</li></ul>

表 3-7: 事件

标识符	事件类型	输入参数
TempError	告警	温度Temperature
HumiError	告警	湿度Humidity

### 3. 编写脚本。

在物联网平台控制台，产品详情页的**数据解析**页签下，选择脚本语言，编写脚本。

脚本中需定义以下两个方法。

- 将Alink JSON格式数据转为设备自定义数据格式的函数：
  - JavaScript (ECMAScript 5) : **protocolToRawData**
  - Python 2.7: **protocol\_to\_raw\_data**
- 将设备自定义数据格式转Alink JSON格式数据的函数：
  - JavaScript (ECMAScript 5) : **rawDataToProtocol**
  - Python 2.7: **raw\_data\_to\_protocol**

本文示例的语言为JavaScript (ECMAScript 5)。

脚本中，解析下行数据的函数**protocolToRawData**中，需设定输出结果的起始三个字节，用于指定下行端口号和下行消息类型。否则，系统会丢掉下行帧。设备实际接收的数据中不会包含这三个字节。

表 3-8: 下行数据解析输出结果起始三字节

LoRa Downlink	字节数	说明
DFlag	1	固定为0x5D。
FPort	1	下行端口号。
DHDR	1	可选： <ul style="list-style-type: none"><li>• 0: 表示 “Unconfirmed Data Down” 数据帧。</li><li>• 1: 表示 “Confirmed Data Down” 数据帧。</li></ul>

例如，0x5D 0x0A 0x00表示下行帧端口号为10，数据帧为 “Unconfirmed Data Down”。

完整的示例脚本Demo，请参见本文附录：示例脚本。

#### 步骤二：在线测试脚本

在数据解析编辑器中，使用模拟数据测试脚本。



- 设备上报数据模拟解析。

在**模拟输入**框中，选择模拟类型为**设备上报数据**，输入模拟数据000102，单击**运行**。

运行结果为：

```
{
  "method": "thing.event.property.post",
  "id": "12345",
  "params": {
    "Temperature": 1,
    "Humidity": 2
  },
  "version": "1.1"
}
```

- 设备接收数据模拟解析。

选择模拟类型为**设备接收数据**，输入以下JSON格式的下行模拟数据，单击**运行**。

```
{
  "method": "thing.service.SetTempHumiThreshold",
  "id": "12345",
  "version": "1.1",
  "params": {
    "MaxTemp": 50,
    "MinTemp": 8,
    "MaxHumi": 90,
    "MinHumi": 10
  }
}
```

运行结果为：

```
0x5d0a000332085a0a
```

### 步骤三：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平台系统，以供数据上下行时，物联网平台调用该脚本解析数据。



#### 说明：

仅提交后的脚本才能被物联网平台调用；草稿状态的脚本不能被调用。

### 步骤四：使用真实设备调试

脚本提交后，正式使用之前，请使用真实设备进行测试。LoRaWAN节点设备如何发送和接收数据，请参见模组厂商的相关手册。

- 测试LoRaWAN设备上报温湿度属性。
  - 使用设备端发送数据，例如000102。
  - 在该设备的**设备详情页运行状态**页签下，查看设备上报的属性数据。

- 测试LoRaWAN设备上报事件。
  1. 使用设备端发送事件数据，如，温度告警数据0102，或湿度告警数据0202。
  2. 在该设备的**设备详情页事件管理**页签下，查看设备上报的事件数据。
- 测试调用LoRaWAN设备服务。
  1. 在物联网平台控制台，选择**监控运维 > 在线调试**。
  2. 选择要调试的产品和设备，并选择**调试真实设备**，功能选择为**温度湿度阈值 (SetTempHumiThreshold)**，输入以下数据后，单击**发送指令**。

```
{
  "MaxTemp": 50,
  "MinTemp": 8,
  "MaxHumi": 90,
  "MinHumi": 10
}
```

3. 检查设备端是否接收到服务调用命令。
4. 在该设备的**设备详情页服务调用**页签下，查看设备服务调用数据。

查看设备属性、事件和服务调用数据的路径如下图所示。



## 附录：示例脚本

根据以上产品和其功能定义，编辑的示例脚本如下。

```
var ALINK_ID = "12345";
var ALINK_VERSION = "1.1";
var ALINK_PROP_POST_METHOD = 'thing.event.property.post';
var ALINK_EVENT_TEMPERR_METHOD = 'thing.event.TempError.post';
var ALINK_EVENT_HUMIERR_METHOD = 'thing.event.HumiError.post';
var ALINK_PROP_SET_METHOD = 'thing.service.property.set';
var ALINK_SERVICE_THSET_METHOD = 'thing.service.SetTempHumiThreshold';
/*
* 示例数据：
* 传入参数 ->
* 000102 // 共3个字节
* 输出结果 ->
* {"method":"thing.event.property.post","id":"12345","params":{"Temperature":1,"Humidity":2},"version":"1.1"}
* 传入参数 ->
```

```

* 0102 // 共2个字节
* 输出结果 ->
* {"method":"thing.event.TempError.post","id":"12345","params":{"Temperature":2},"
version":"1.1"}
* 传入参数 ->
* 0202 // 共2个字节
* 输出结果 ->
* {"method":"thing.event.HumiError.post","id":"12345","params":{"Humidity":2},"version
":"1.1"}
*/
function rawDataToProtocol(bytes)
{
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++)
    {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var params = {};
    var jsonMap = {};
    var dataView = new DataView(uint8Array.buffer, 0);
    var cmd = uint8Array[0]; // command
    if (cmd === 0x00)
    {
        params['Temperature'] = dataView.getInt8(1);
        params['Humidity'] = dataView.getInt8(2);
        jsonMap['method'] = ALINK_PROP_POST_METHOD;
    }
    else if (cmd == 0x01)
    {
        params['Temperature'] = dataView.getInt8(1);
        jsonMap['method'] = ALINK_EVENT_TEMPERR_METHOD;
    }
    else if (cmd == 0x02)
    {
        params['Humidity'] = dataView.getInt8(1);
        jsonMap['method'] = ALINK_EVENT_HUMIERR_METHOD;
    }
    else
    {
        return null;
    }
    jsonMap['version'] = ALINK_VERSION;
    jsonMap['id'] = ALINK_ID;
    jsonMap['params'] = params;
    return jsonMap;
}
/*
* 示例数据:
* 传入参数 ->
* {"method":"thing.service.SetTempHumiThreshold", "id":"12345", "version":"1.1", "
params":{"MaxTemp":50, "MinTemp":8, "MaxHumi":90, "MinHumi":10}}
* 输出结果 ->
* 0x5d0a000332085a0a
*/
function protocolToRawData(json)
{
    var id = json['id'];
    var method = json['method'];
    var version = json['version'];
    var payloadArray = [];
    // 追加下行帧头部
    payloadArray = payloadArray.concat(0x5d);
    payloadArray = payloadArray.concat(0x0a);
    payloadArray = payloadArray.concat(0x00);

```

```
if (method == ALINK_SERVICE_THSET_METHOD)
{
    var params = json['params'];
    var maxtemp = params['MaxTemp'];
    var mintemp = params['MinTemp'];
    var maxhumi = params['MaxHumi'];
    var minhumi = params['MinHumi'];
    payloadArray = payloadArray.concat(0x03);
    if (maxtemp !== null)
    {
        payloadArray = payloadArray.concat(maxtemp);
    }
    if (mintemp !== null)
    {
        payloadArray = payloadArray.concat(mintemp);
    }
    if (maxhumi !== null)
    {
        payloadArray = payloadArray.concat(maxhumi);
    }
    if (minhumi !== null)
    {
        payloadArray = payloadArray.concat(minhumi);
    }
}
return payloadArray;
}
// 以下是部分辅助函数
function buffer_uint8(value)
{
    var uint8Array = new Uint8Array(1);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value)
{
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value)
{
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value)
{
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
```

## 相关文档

- 查看JavaScript（ECMAScript 5）脚本模板和示例，请参见[JavaScript脚本示例](#)。
- 查看Python 2.7脚本模板和示例，请参见[Python脚本示例](#)

- 了解数据解析流程等基本信息，请参见[什么是数据解析](#)。
- 关于数据解析问题排查，请参见[问题排查](#)。
- 有关自定义Topic数据解析说明，请参见[概述](#)。

## 3.5 问题排查

本文介绍在本地环境调试数据解析脚本的代码示例，和物联网平台不能正常使用脚本解析数据的排错方法。

### 本地环境调试脚本

目前，物联网平台数据解析支持在线测试脚本是否能解析数据，但不支持调试。建议先在本地编写脚本、调试完成后，再将脚本拷贝到物联网控制台的脚本编辑器中。可参见如下方式进行本地调试。

以下本地调试代码基于[物模型数据解析使用示例](#)中的示例脚本。您在实际使用时，请按照您的脚本需求进行具体参数设置。

```
// Test Demo
function Test()
{
    //0x001232013fa00000
    var rawdata_report_prop = new Buffer([
        0x00, //固定command头, 0代表是上报属性
        0x00, 0x22, 0x33, 0x44, //对应id字段, 标记请求的序号
        0x12, 0x32, //两字节 int16, 对应属性 prop_int16
        0x01, //一字节 bool, 对应属性 prop_bool
        0x3f, 0xa0, 0x00, 0x00 //四字节 float, 对应属性 prop_float
    ]);
    rawDataToProtocol(rawdata_report_prop);
    var setString = new String("{\"method\":\"thing.service.property.set\",\"id\":\"12345\",\"version\":\"1.0\",\"params\":{\"prop_float\":123.452, \"prop_int16\":333, \"prop_bool\":1}}");
    protocolToRawData(JSON.parse(setString));
}
Test();
```

### 线上问题排查

设备端连接物联网平台，上报属性数据后，若数据解析运行正常，在该设备的[设备详情页运行状态页](#)签下，将会显示设备上报的数据。

若设备已经上报了数据，但是[运行状态](#)下却没有显示对应的数据。如下图所示。



若出现这种情况，可在**监控运维 > 日志服务**中，选择**物模型数据分析**，查看属性相关的通信日志。

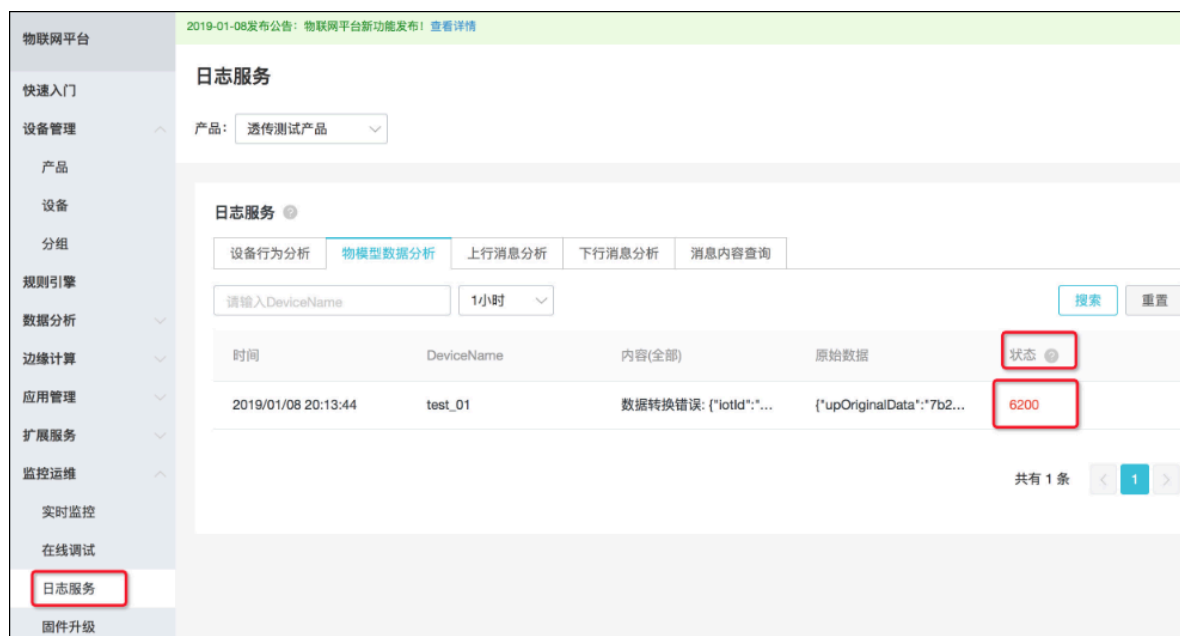
问题排查过程如下：

1. 在**日志服务**中，查看日志记录。日志中会显示脚本转化后的数据和原数据。
2. 结合**日志说明文档**，查看错误码的信息。
3. 按照错误码提示，结合脚本和设备上报的数据排查问题。

下面列举一些错误。

- 脚本不存在。

如下图，物模型数据分析日志中，显示错误码为6200。访问**日志说明文档**，查看错误的具体含义。错误码6200表示脚本不存在。请在控制台检查脚本是否已提交。



- Alink method不存在。

日志中显示错误码为6450。[日志说明文档](#)中有该错误码解释：错误码6450表示Alink协议格式的数据中method不存在。原因是设备上报的自定义/透传格式数据，经过脚本解析为Alink标准格式数据后无method。

快速入门

设备管理

产品

设备

分组

规则引擎

数据分析

边缘计算

应用管理

扩展服务

监控运维

2019-01-08发布公告：物联网平台新功能发布! [查看详情](#)

日志服务

产品: 透传测试产品

日志服务

设备行为分析 物模型数据分析 上行消息分析 下行消息分析 消息内容查询

1小时 搜索 重置

时间	DeviceName	内容(全部)	原始数据	状态
2019/01/08 20:21:21	test_01	数据转换错误: {"iotId": "...	{"upOriginalData": "7b2...	6450

日志内容如：

```
17:54:19.064, A7B02C60646B4D2E8744F7AA7C3D9567, upstream-error - bizType=OTHER_MESSAGE,params={"params":{}},result=code:6450,message:alink method not exist,...
```

可以从日志内容中看到，错误消息为alink method not exist，即Alink 协议格式的数据中method不存在。这是解析脚本中method定义有问题，需修改脚本。

## 4 标签

物联网平台的标签是您给产品、设备或分组自定义的标识。您可以使用标签功能来灵活管理产品、设备和分组。

### 概述

物联网往往涉及量级产品与设备的管理。如何区分不同批次的产品与设备，如何实现批量管理，成为一大挑战。阿里云物联网平台为解决这一问题提供了标签功能。您可以为不同产品、设备或设备分组贴上不同标签，然后根据标签实现分类统一管理。

标签包括产品标签、设备标签和分组标签。标签的结构为Key:Value。



#### 说明：

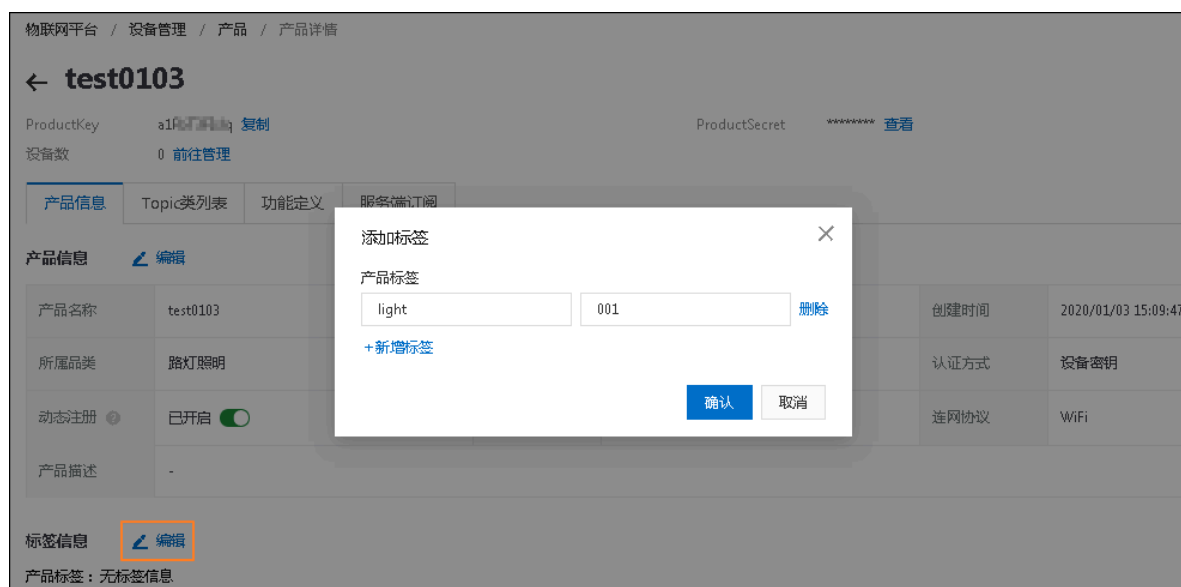
每个产品、设备或分组最多可有100个标签。

### 产品标签

产品标签通常描述的是对一个产品下所有设备所具有的共性信息。如产品的制造商、所属单位、外观尺寸、操作系统等。需在创建产品后，再为该产品添加产品标签。

在控制台添加产品标签操作步骤：

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 产品**。
3. 在**产品**页面，找到需要添加标签的产品，并单击对应操作栏中的**查看**。
4. 单击**标签信息**右侧的**编辑**按钮。





5. 在弹出的对话框中，输入标签的 **标签Key** 和 **标签Value**，然后单击**确认**。

参数	说明
标签Key	可包含英文大小写字母，数字和点号（.），长度不可超过30个字符。
标签Value	可包含中文汉字、英文字母、数字、下划线（_）、连接号（-）、英文冒号（:）和点号（.）。长度不可超过128字符。一个中文汉字算2字符。

## 设备标签

您可以根据设备的特性为设备添加特有的标签，方便对设备进行管理。例如，为房间 201 的智能电表定义一个标签为room:201。

设备标签信息会跟随设备在系统内部流转。并且，物联网平台可以基于规则引擎的数据流转功能，将设备标签添加到设备上报的消息体里，并发送给其它阿里云产品。

在控制台添加设备标签的操作步骤：

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 设备**。
3. 在**设备**页面，单击要添加标签的设备所对应的**查看**，进入**设备详情**页面。
4. 单击**标签信息**右侧的**编辑**按钮。



5. 在弹出的对话框中，输入标签的 **标签Key** 和 **标签Value**，然后单击**确认**。

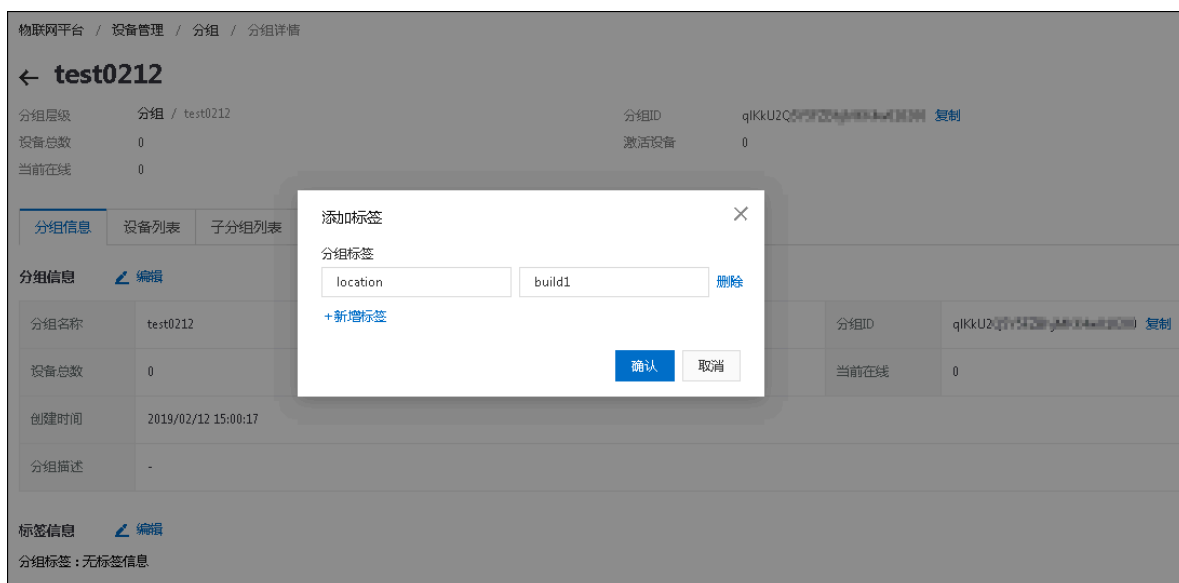
参数	说明
标签Key	可包含英文字母，数字和点号（.），长度不可超过30个字符。
标签Value	可包含中文汉字、英文字母、数字、下划线（_）、连接号（-）、英文冒号（:）和点号（.）。长度不可超过128字符。一个中文汉字算2字符。

## 分组标签

设备分组用于跨产品管理设备。分组标签通常描述的是一个分组下所有设备和子分组所具有的共性信息，如分组下的设备所在的地域、空间等。需在创建分组后，再为该分组添加标签。

添加分组标签操作步骤：

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > 分组**。
3. 在**分组**页面，找到需要添加标签的分组，并单击对应操作栏中的**查看**。
4. 单击**标签信息**右侧的**编辑**按钮。



5. 在弹出的对话框中，输入标签的 **标签Key** 和 **标签Value**，然后单击**确认**。

参数	说明
标签Key	可包含英文字母，数字和点号（.），长度不可超过30个字符。

参数	说明
标签Value	可包含中文汉字、英文字母、数字、下划线（_）和连接号（-）、英文冒号（:）和点号（.）。长度不可超过128字符。一个中文汉字算2字符。

### 批量操作标签

除在控制台创建、编辑和删除标签外，您还可以调用物联网平台提供的API来批量管理标签。此外，物联网平台还提供API，用于根据标签来查询产品、设备和分组。

请参见[API列表](#)。

## 5 设备分组

物联网平台提供设备分组功能。您可以通过设备分组来进行跨产品管理设备。本章节介绍如何在物联网平台控制台创建设备分组和管理分组。

### 背景信息

- 一个分组最多可包含100个一级子分组。
- 分组只支持三级嵌套，即分组>子分组>子子分组。
- 一个子分组只能隶属于一个父组。
- 分组的嵌套关系创建后不能修改，只能删除后重新创建。
- 分组下有子分组时，不能直接删除分组。需子分组全部删除后，才能删除父组。
- 搜索分组时，支持分组名称模糊搜索，包括在分组列表和子分组列表里的搜索。

### 操作步骤

1. 登录[物联网平台控制台](#)。
2. 单击**设备管理** > **分组**进入分组管理页面。
3. 单击**新建分组**，设置分组参数，并单击**保存**。



说明：

一个阿里云账号下最多可创建1,000个分组，包括分组和子分组。

新建分组 ?

父组 : ?  

请选择

\* 分组名称 : ?  

testGroup

分组描述 :  

测试

2/100

确认

取消

参数信息解释如下：

- **父组**：选择创建的分组类型。
  - 分组：创建的分组是一个父组。
  - 选择指定父组：以指定的分组为父组，创建子分组。
- **分组名称**：给该分组创建名称。分组名称支持中文汉字、英文字母、数字和下划线（-），长度限制4~30。分组名称必须为账号下唯一，且创建后不能修改。
- **分组描述**：输入文字，描述该分组。可为空。

4. 在**分组**页面，单击已创建分组对应的**查看**操作按钮，进入**分组详情**页面。

5. （可选）为分组添加标签，即自定义分组标识，以便灵活管理分组。

a) 单击**标签信息**栏的**编辑**，并输入标签的key和value。

- 标签key：可包含英文大小写字母，数字和点号（.），长度在2-32字符之间。
- 标签value：可包含中文、英文字母、数字、下划线（\_）和连字符（-）。长度不可超过128字符。一个中文汉字算2字符。

b) 单击**确认**添加标签。



**说明：**

一个分组最多可添加100个标签。

6. 单击**设备列表 > 添加设备到分组**。搜索并勾选设备，单击**确定**将选中设备添加至分组。

搜索设备：

- 若选择**全部产品**，将列出账号下全部产品下的所有设备。您可以输入设备名称的部分元素进行模糊搜索设备。如，输入test，可搜索出账号下名称为test1、test2、test3的设备。
- 选择某个产品，将列出该产品下的所有设备。您还可以输入该产品下设备名称的部分元素进行模糊搜索设备。



**说明：**

- 单次最多添加1000个设备。单个分组最多添加20,000个设备。
- 一个设备最多可以被添加到10个分组中。

**添加设备到分组**页面右上方的**全部**和**已选择**按钮说明：

- 单击**全部**，显示所有设备列表。
- 单击**已选择**，显示您已勾选的设备列表。

7. （可选）单击**子分组列表 > 新建分组**，为分组添加子分组。

子分组功能用于细化设备管理。例如，您可以在“智能家居”分组下，创建“智能厨房”、“智能卧室”等子分组，实现厨房设备和卧室设备的分开管理。具体操作如下：

- a) 选择该子分组的父组，输入子分组名称和描述，然后单击**保存**。
- b) 在**子分组列表**页面，单击子分组对应的**查看**操作按钮，进入该子分组的**分组详情页**。
- c) 单击**设备列表 > 添加设备到分组**，然后为该子分组添加设备。

创建子分组和添加子分组设备完成后，您可以对该子分组和其设备进行管理。您还可以在子分组下再创建子分组。

## 6 设备影子

### 6.1 设备影子概览

物联网平台提供设备影子功能，用于缓存设备状态。设备在线时，可以直接获取云端指令；设备离线后，再次上线可以主动拉取云端指令。

设备影子是一个 JSON 文档，用于存储设备上报状态、应用程序期望状态信息。

每个设备有且只有一个设备影子，设备可以通过 MQTT 获取和设置设备影子来同步状态，该同步可以是影子同步给设备，也可以是设备同步给影子。

#### 应用场景

- 场景1：网络不稳定，设备频繁上下线。

由于网络不稳定，设备频繁上下线。应用程序发出需要获取当前的设备状态请求时，设备掉线，无法获取设备状态，但下一秒设备又连接成功，应用程序无法正确发起请求。

使用设备影子机制存储设备最新状态，一旦设备状态产生变化，设备会将状态同步到设备影子。应用程序在请求设备当前状态时，只需要获取影子中的状态即可，不需要关心设备是否在线。

- 场景2：多程序同时请求获取设备状态。

如果设备网络稳定，很多应用程序请求获取设备状态，设备需要根据请求响应多次，即使响应的结果是一样的，设备本身处理能力有限，无法负载被请求多次的情况。

使用设备影子机制，设备只需要主动同步状态给设备影子一次，多个应用程序请求设备影子获取设备状态，即可获取设备最新状态，做到应用程序和设备的解耦。

- 场景3：设备掉线。

- 设备网络不稳定，导致设备频繁上下线，应用程序发送控制指令给设备时，设备掉线，指令无法下达到设备。

- 通过 QoS=1 或者 2 实现，但是该方法对于服务端的压力比较大，一般不建议使用。

- 使用设备影子机制，应用程序发送控制指令，指令携带时间戳保存在设备影子中。当设备掉线重连时，获取指令并根据时间戳确定是否执行。

- 设备真实掉线，指令发送失败。设备再上线时，设备影子功能通过指令加时间戳的模式，保证设备不会执行过期指令。

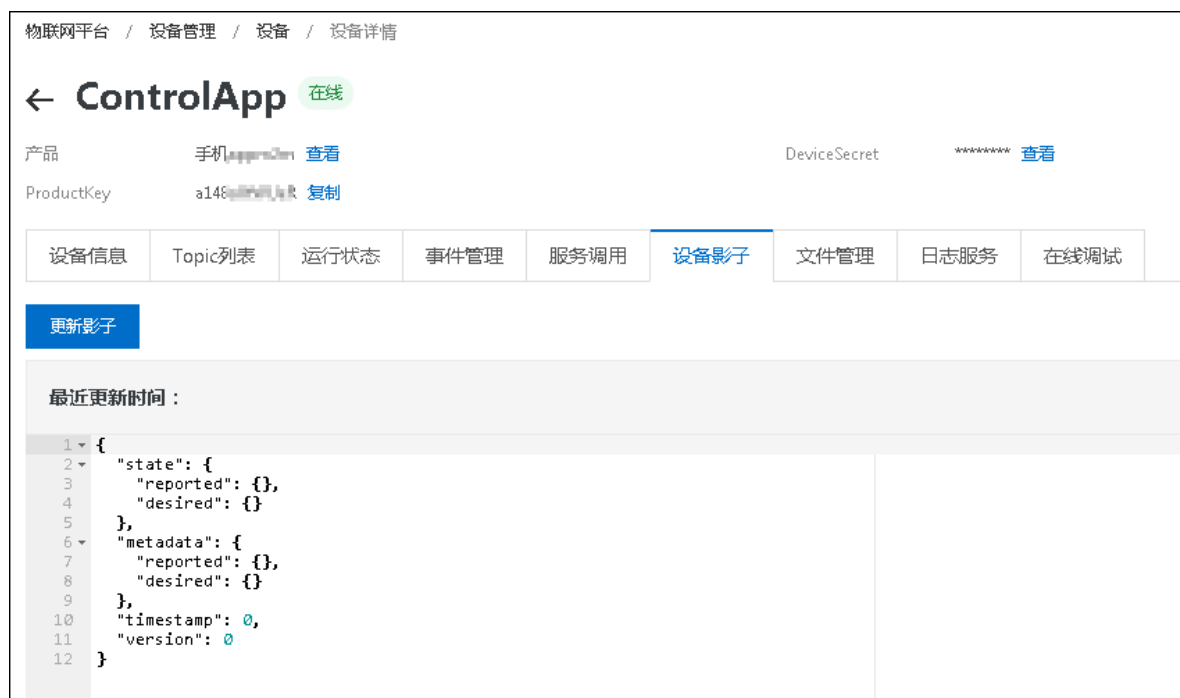
#### 查看与更新设备影子

您可以在控制台，查看设备影子信息，更新设备影子状态。

操作步骤：

1. 登录[物联网平台控制台](#)。
2. 单击[设备管理](#) > [设备](#)。
3. 单击对应设备的[查看](#)按钮，进入设备详情页。
4. 单击[设备影子](#)。

页面显示设备上报的影子状态。



5. 单击[更新影子](#)，在“desired”部分，填入期望设备状态。

设备影子文档格式，请参见[设备影子JSON详解](#)。

设备在线时，设备影子保存期望状态，设备通过订阅Topic直接获得期望状态。

设备离线时，设备影子缓存期望状态，设备上线后，主动从云端拉取最新期望状态。

## 相关API

获取设备影子：[#unique\\_42](#)

更新设备影子：[#unique\\_43](#)

## 6.2 设备影子JSON详解

本文档介绍设备影子的JSON格式表达方法。

设备影子JSON文档示例：

```
{
```



```
{
  "state": {
    "desired": {
      "color": "RED",
      "sequence": [
        "RED",
        "GREEN",
        "BLUE"
      ]
    },
    "reported": {
      "color": "GREEN"
    }
  },
  "metadata": {
    "desired": {
      "color": {
        "timestamp": 1469564492
      },
      "sequence": {
        "timestamp": 1469564492
      }
    },
    "reported": {
      "color": {
        "timestamp": 1469564492
      }
    }
  },
  "timestamp": 1469564492,
  "version": 1
}
```

JSON属性描述，如下表[表 6-1: JSON属性说明](#)所示。

**表 6-1: JSON属性说明**

属性	描述
<b>desired</b>	<p>设备的预期状态。仅当设备影子文档具有预期状态时，才包含<b>desired</b>部分。</p> <p>应用程序向<b>desired</b>部分写入数据，更新事物的状态，而无需直接连接到该设备。</p>
<b>reported</b>	<p>设备的报告状态。设备可以在<b>reported</b>部分写入数据，报告其最新状态。</p> <p>应用程序可以通过读取该参数值，获取设备的状态。</p> <p>JSON文档中也可以不包含<b>reported</b>部分，没有<b>reported</b>部分的文档同样为有效影子JSON文档。</p>
<b>metadata</b>	<p>当用户更新设备状态文档后，设备影子服务会自动更新<b>metadata</b>的值。</p> <p>设备状态的元数据的信息包含以 Epoch 时间表示的每个属性的时间戳，用来获取准确的更新时间。</p>

属性	描述
timestamp	影子文档的最新更新时间。
version	<p>用户主动更新版本号时，设备影子会检查请求中的<b>version</b>值是否大于当前版本号。</p> <p>如果大于当前版本号，则更新设备影子，并将<b>version</b>值更新到请求的版本中，反之则会拒绝更新设备影子。</p> <p>该参数更新后，版本号会递增，用于确保正在更新的文档为最新版本。</p> <p>version参数为long型。为防止参数溢出，您可以手动传入-1将版本号重置为0。</p>

**说明：**

设备影子支持数组。更新数组时必须全量更新，不能只更新数组的某一部分。

更新数组数据示例：

- 初始状态：

```
{
  "reported" : { "colors" : ["RED", "GREEN", "BLUE"] }
}
```

- 更新：

```
{
  "reported" : { "colors" : ["RED"] }
}
```

- 最终状态：

```
{
  "reported" : { "colors" : ["RED"] }
}
```

```
}
```

## 6.3 设备影子数据流

设备影子数据通过Topic进行流转。本文介绍设备影子数据流转，包括：设备上报状态到设备影子，应用程序更改设备状态，设备离线再上线后主动获取设备影子信息，和设备端请求删除设备影子中的属性信息。

### 设备影子Topic

物联网平台已为每个设备预定义了两个Topic，用于实现数据流转。您可以直接使用。

- `/shadow/update/${YourProductKey}/${YourDeviceName}`

设备和应用程序发布消息到此Topic。物联网平台收到该Topic的消息后，将消息中的状态更新到设备影子中。

- `/shadow/get/${YourProductKey}/${YourDeviceName}`

设备影子更新状态到该Topic，设备订阅此Topic获取最新消息。

### 使用示例

以下章节中，以某个具体灯泡设备为例，说明设备、设备影子以及应用程序之间的通信。

示例中，产品的ProductKey是a1PbRCF\*\*\*\*；设备名称DeviceName是lightbulb。设备以QoS=1发布消息和订阅两个设备影子Topic。

示例主要讲解四大部分内容：设备主动上报状态，应用程序改变设备状态，设备主动获取影子内容，和设备主动删除影子属性。

#### 一、设备主动上报状态

设备在线时，主动上报设备状态到影子，应用程序主动获取设备影子状态。

数据流转过程如下图所示。



1. 当灯泡lightbulb上线时，使用Topic /shadow/update/a1PbRCF\*\*\*\*/lightbulb上报最新状态到影子。

发送的JSON消息格式：

```
{
  "method": "update",
  "state": {
    "reported": {
      "color": "red"
    }
  },
  "version": 1
}
```

表 6-2: 上报参数说明

参数	说明
<b>method</b>	表示设备或者应用程序请求设备影子时的操作类型。  当执行更新操作时， <b>method</b> 为必填字段，设置为update。
<b>state</b>	表示设备发送给设备影子的状态信息。  <b>reported</b> 为必填字段，状态信息会同步更新到设备影子的 <b>reported</b> 部分。
<b>version</b>	表示设备影子检查请求中的版本信息。  只有当新版本大于当前版本时，设备影子才会接收设备端的请求，并更新设备影子版本。  如果 <b>version</b> 设置为-1时，表示清空设备影子数据，设备影子会接收设备端的请求，并将设备影子版本更新为0。

2. 设备影子接收到灯泡上报的状态数据后，更新影子文档。

```
{
  "state": {
    "reported": {
      "color": "red"
    }
  },
  "metadata": {
    "reported": {
      "color": {
        "timestamp": 1469564492
      }
    }
  },
  "timestamp": 1469564492,
  "version": 1
}
```

```
}
```

3. 影子文件更新后，设备影子会返回结果给设备（灯泡），即发送消息到设备订阅的Topic / shadow/get/a1PbRCF\*\*\*\*/lightbulb中。

- 若更新成功，发送到该Topic中的消息为：

```
{
  "method": "reply",
  "payload": {
    "status": "success",
    "version": 1
  },
  "timestamp": 1469564576
}
```

- 若更新失败，发送到该Topic中的消息为：

```
{
  "method": "reply",
  "payload": {
    "status": "error",
    "content": {
      "errorCode": "${errorCode}",
      "errorMessage": "${errorMessage}"
    }
  },
  "timestamp": 1469564576
}
```

表 6-3: 错误码说明

errorCode	errorMessage
400	不正确的JSON格式。
401	影子数据缺少 <b>method</b> 信息。
402	影子数据缺少 <b>state</b> 字段。
403	影子数据中 <b>version</b> 值不是数字。
404	影子数据缺少 <b>reported</b> 字段。
405	影子数据中 <b>reported</b> 属性字段为空。
406	影子数据中 <b>method</b> 是无效的方法。
407	影子内容为空。
408	影子数据中 <b>reported</b> 属性个数超过128个。
409	影子版本冲突。
500	服务端处理异常。

## 二、应用程序改变设备状态

应用程序通过调用云端API [#unique\\_43](#)下发期望状态给设备影子，设备影子再将文件下发给设备端。设备根据影子更新状态，并上报最新状态至影子。

数据流转流程如下图所示。



1. 应用程序调用云端API **UpdateDeviceShadow**，下发消息要求更改灯泡状态，例如要求将灯泡的 **color**属性值改为green。

调用API时，参数**ShadowMessage**的值为：

```
{
  "method": "update",
  "state": {
    "desired": {
      "color": "green"
    }
  },
  "version": 2
}
```

2. 设备影子接收到更新请求，更新其影子文档为：

```
{
  "state": {
    "reported": {
      "color": "red"
    },
    "desired": {
      "color": "green"
    }
  },
  "metadata": {
    "reported": {
      "color": {
        "timestamp": 1469564492
      }
    },
    "desired": {
      "color": {
        "timestamp": 1469564576
      }
    }
  },
  "timestamp": 1469564576,
}
```

```
"version": 2
}
```

3. 设备影子更新完成后，发送返回结果到Topic/shadow/get/a1PbRCF\*\*\*\*/lightbulb中。返回结果信息构成由设备影子决定。

```
{
  "method": "control",
  "payload": {
    "state": {
      "reported": {
        "color": "red"
      },
      "desired": {
        "color": "green"
      }
    },
    "metadata": {
      "reported": {
        "color": {
          "timestamp": 1469564492
        }
      },
      "desired": {
        "color": {
          "timestamp": 1469564576
        }
      }
    }
  },
  "version": 2,
  "timestamp": 1469564576
}
```

4. 如果设备灯泡在线，并且订阅了Topic/shadow/get/a1PbRCF\*\*\*\*/lightbulb，则会立即收到消息。

收到消息后，根据请求文档中**desired**的值，将灯泡颜色变成绿色。

灯泡更新完状态后，上报最新状态到物联网平台。

```
{
  "method": "update",
  "state": {
    "reported": {
      "color": "green"
    }
  },
  "version": 3
}
```



#### 说明：

如果有时间戳判断指令过期，也可以选择不更新。

5. 最新状态上报成功后，设备端和设备影子进行以下操作。

- 设备端发消息到Topic/shadow/update/a1PbRCFWfx/lightbulb中清空**desired**属性。消息如下：

```
{
  "method": "update",
  "state": {
    "desired": "null"
  },
  "version": 4
}
```

- 设备影子会同步更新影子文档，此时的影子文档如下：

```
{
  "state": {
    "reported": {
      "color": "green"
    }
  },
  "metadata": {
    "reported": {
      "color": {
        "timestamp": 1469564577
      }
    }
  },
  "desired": {
    "timestamp": 1469564576
  }
},
"version": 4
}
```

### 三、设备主动获取影子内容

若应用程序发送指令时，设备离线。设备再次上线后，将主动获取设备影子内容。

数据流转过程如下图所示。



- 灯泡主动发送以下消息到Topic/shadow/update/a1PbRCF\*\*\*\*/lightbulb中，请求获取设备影子中保存的最新状态。

```
{
  "method": "get"
}
```



```
}
```

2. 当设备影子收到这条消息后，发送最新状态到Topic/shadow/get/a1PbRCF\*\*\*\*/lightbulb。灯泡通过订阅该Topic获取最新状态。消息内容如下：

```
{
  "method": "reply",
  "payload": {
    "status": "success",
    "state": {
      "reported": {
        "color": "red"
      },
      "desired": {
        "color": "green"
      }
    },
    "metadata": {
      "reported": {
        "color": {
          "timestamp": 1469564492
        }
      },
      "desired": {
        "color": {
          "timestamp": 1469564492
        }
      }
    }
  },
  "version": 2,
  "timestamp": 1469564576
}
```

#### 四、设备主动删除影子属性

若设备端已经是最新状态，设备端可以主动发送指令，删除设备影子中保存的某条属性状态。

数据流转过程如下图所示。



设备发送以下内容到Topic/shadow/update/a1PbRCF\*\*\*\*/lightbulb中。

其中，**method**为delete，属性的值为null。

- 删除影子中某一属性。

```
{
  "method": "delete",
  "state": {
    "reported": {
      "color": "null",
      "temperature": "null"
    }
  },
  "version": 1
}
```

- 删除影子全部属性。

```
{
  "method": "delete",
  "state": {
    "reported": "null"
  },
  "version": 1
}
```

## 7 文件管理

物联网平台支持设备通过HTTP/2流通道方式，将文件上传至阿里云物联网平台服务器储存。设备上传文件后，您可以在物联网平台控制台进行下载、删除等管理操作。

### 前提条件

- 设备端成功连接到物联网平台。

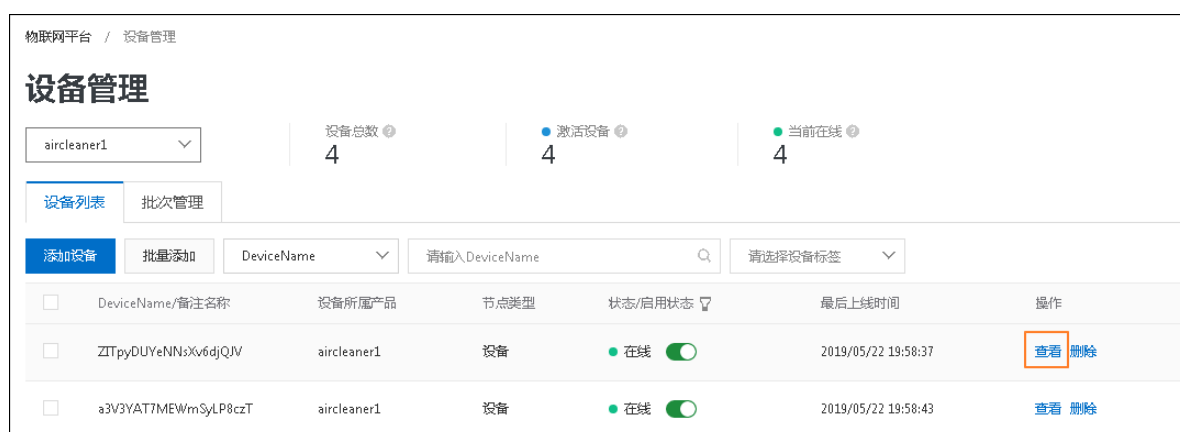
设备端SDK开发，请参见[Link SDK文档](#)。

- 设备端编译配置HTTP/2上传文件功能。

配置设备端上传文件功能，请参见Link SDK文档中[文件上传](#)章节。

### 操作步骤

- 登录[物联网平台控制台](#)。
- 在左侧导航栏，选择**设备管理 > 设备**，然后单击设备对应的**查看**按钮。



- 在**设备详情**页面，选择**文件管理**页签。

在**文件管理**页签下，您可以查看该设备通过HTTP/2通道上传的文件列表。



说明：

目前，一个阿里云账号下可存储在物联网平台服务器的文件总大小的上限是1G；每个设备下最多可以有1000个文件。

设备管理 / 设备详情

← AfhbeXsJmE3AQ8Kf8OfP

产品 路灯产品 [查看](#)

DeviceSecret \*\*\*\*\* [查看](#)

ProductKey  [复制](#)

设备信息

Topic列表

运行状态

事件管理

服务调用

设备影子

文件管理

日志服务

在线调试

📘 账号下已上传文件大小：200 KB / 1 GB

使用教程

文件名称	文件大小	上传时间	操作
testfile	100 KB	2019-12-02 14:00:41	<a href="#">下载</a> <a href="#">删除</a>

您可以对已上传的文件进行如下操作：

操作	描述
下载	下载该文件到本地。
删除	删除该文件。

除在控制台管理文件外，您还可以通过调用云端API查询或删除文件。请参见[#unique\\_46](#)、[#unique\\_47](#)、[#unique\\_48](#)。

## 8 NTP服务

物联网平台提供NTP服务，解决嵌入式设备资源受限，系统不包含NTP服务，端上没有精确时间戳的问题。

### 原理介绍

物联网平台借鉴NTP协议原理，将云端作为NTP服务器。设备端发送一个特定Topic给云端，payload中带上发送时间。云端回复时在payload中加上云端的接收时间和发送时间。设备端收到回复后，再结合自己本地当前时间，得出一共4个时间。一起计算出设备端与云端的时间差，从而得出端上当前的精确时间。



#### 说明：

只有设备端与云端成功建立连接之后，才能通过NTP服务进行校准。

如果嵌入式设备上电后没有准确时间，TLS建连过程中证书时间校验失败的问题，无法通过NTP服务解决，因为此时设备与云端尚未成功建立连接。

### 接入流程

请求Topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/request`

响应Topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`



#### 说明：

ProductKey和DeviceName是设备证书的一部分，可以从控制台获取。

1. 设备端订阅Topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`。
2. 设备端向Topic `/ext/ntp/${YourProductKey}/${YourDeviceName}/request`发送一条QoS=0的消息，payload中带上设备当前的时间戳，单位为毫秒。示例如下：

```
{
  "deviceSendTime": "1571724098000"
}
```



#### 说明：

- 时间戳数字的格式，支持Long和String。默认为Long类型
- NTP服务目前仅支持QoS=0的消息。

3. 设备端收到服务端回复的消息，payload中包含以下信息：

```
{
  "deviceSendTime": "1571724098000",
```

```
"serverRecvTime":"1571724098110",
"serverSendTime":"1571724098115",
}
```

#### 4. 设备端计算出当前精确的unix时间。

设备端收到服务端的时间记为 $\text{\${deviceRecvTime}}$ ，则设备上的精确时间为： $(\text{\${serverRecvTime}} + \text{\${serverSendTime}} + \text{\${deviceRecvTime}} - \text{\${deviceSendTime}}) / 2$

#### 使用示例



##### 说明：

设备端和服务端发送的时间戳数据的类型相同。例如，设备端传的时间戳是String类型，服务端返回的时间戳也是String类型。

例如，设备上时间是1571724098000，服务端时间是1571724098100，链路延时是10毫秒，服务端从接收到发送间隔为5毫秒。

-	设备端时间	服务端时间
设备发送	1571724098000 (deviceSendTime)	1571724098100
服务端接收	1571724098010	1571724098110 (serverRecvTime)
服务端发送	1571724098015	1571724098115 (serverSendTime)
设备端接收	1571724098025 (deviceRecvTime)	1571724098125

则设备端计算出的当前准确时间为  $(1571724098110 + 1571724098115 + 1571724098025 - 1571724098000) / 2 = 1571724098125$ 。

如果直接采用云端返回的时间戳，只能得到1571724098115，与服务端上的时间会有10毫秒的链路延时误差。

## 9 网关与子设备

---

### 9.1 网关与子设备

物联网平台支持设备直连，也支持设备挂载在网关上，作为网关的子设备，由网关直连。

#### 适用场景

适用于子设备不能直连或者需要拓扑关系管理的场景，如接入Wi-Fi网关、蓝牙网关、ZigBee网关等。

#### 方案优势

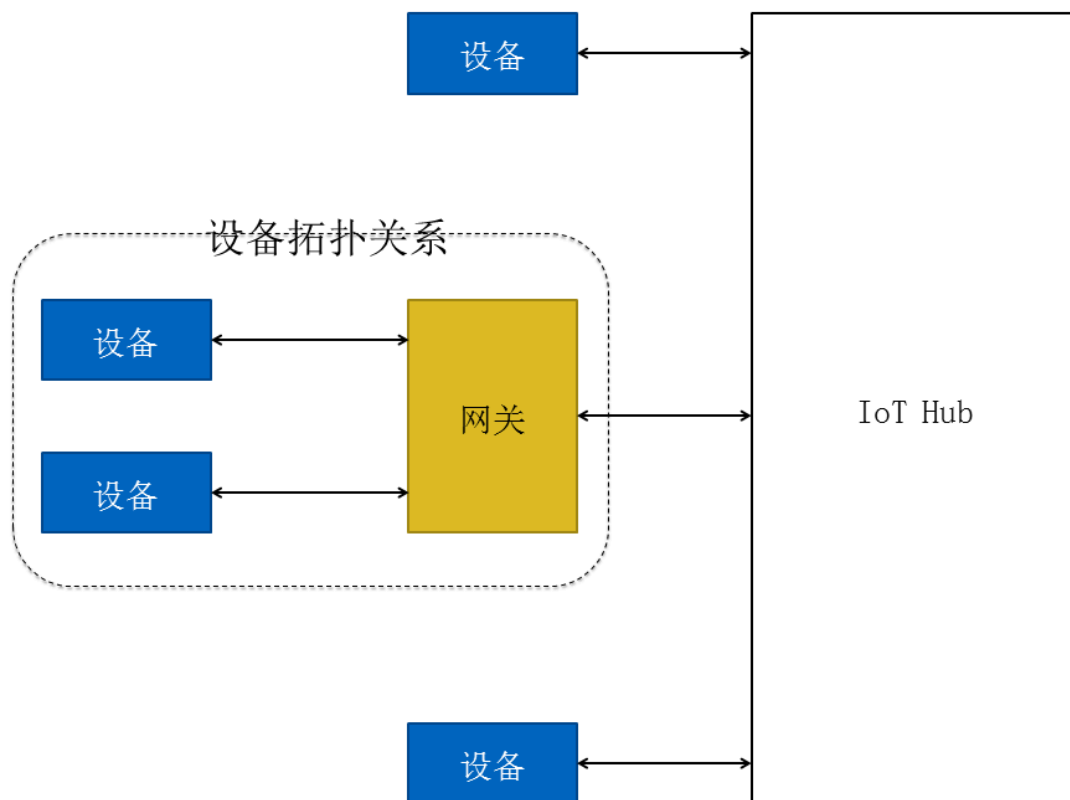
平台可以帮助您管理子设备、子设备与网关的拓扑关系，对子设备进行监控运维等，同时您的业务系统可以直接面向子设备收发消息，上层控制端给子设备下发消息时无需感知物理拓扑结构。

#### 网关与设备

创建产品与设备时，需要选择节点类型。平台目前支持两种节点类型：设备和网关。

- 设备：指不能挂载子设备的设备。设备可以直连物联网平台，也可以作为网关的子设备，由网关代理连接物联网平台。
- 网关：指可以挂载子设备的直连设备。网关可以管理子设备、可以维持与子设备的拓扑关系，并将该拓扑关系同步到云端。

网关与子设备的拓扑关系如下图所示。



## 接入方式

网关连接物联网平台后，将拓扑关系同步至云端，代理子设备进行设备认证、消息上传、指令接收等与物联网平台的通信。

- 网关接入物联网平台的方法与普通设备相同，具体请参见[Link SDK文档](#)。
- 子设备接入物联网平台有两种方式：
  - 使用[#unique\\_11](#)的认证方式。网关获取子设备的设备证书，由网关向物联网平台上报子设备证书信息（ProductKey、DeviceName和DeviceSecret）。
  - 使用[子设备动态注册](#)的认证方式。需在控制台，打开子设备的动态注册开关。网关获取子设备的ProductKey和DeviceName后，由网关代理子设备进行动态注册，云端校验子设备身份。校验通过后，动态下发DeviceSecret。然后子设备通过设备证书（ProductKey、DeviceName和DeviceSecret）接入物联网平台。



## 9.2 子设备管理

本文介绍在网关下面关联子设备。

### 操作步骤

1. 在物联网平台控制台左侧导航栏，选择**设备管理 > 设备**。
2. 在**设备**页面，找到对应网关，单击**查看**，进入**设备详情页**。
3. 单击**子设备管理 > 添加子设备**。
4. 在弹出页面上，选择要关联的子设备，然后单击**确定**。

参数	描述
产品	选择子设备对应的产品名称。
设备	选择子设备的设备名称。

### 预期结果

子设备信息配置完成后，可在**子设备管理**页签下，您可以：

- 单击子设备对应的**查看**按钮，进入**子设备详情页**查看子设备信息。
- 单击子设备对应的**删除**按钮，将该子设备从网关中删除。



#### 说明：

本操作仅删除子设备与网关的拓扑关系，并不删除设备本身。

## 9.3 子设备上线

子设备不直接连接物联网平台，而是通过网关与物联网平台建立连接，复用网关与物联网平台的通信通道。

### 背景信息

开发网关设备端时，需实现网关管理与子设备的拓扑关系、代理子设备上下线、代理子设备与物联网平台进行物模型通信等功能。

您可以使用阿里云提供的设备端SDK进行网关开发，详情请参见[Link SDK](#)文档。

如果您自行开发网关设备端，需在网关设备端封装子设备相关Alink协议数据。各功能Alink协议数据，请基于Alink协议开发目录下相关文档。

### 子设备上线过程

网关代理子设备上线过程如下。

1. 网关接入物联网平台。

2. 子设备接入网关。

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

网关发现子设备、网关获取子设备的物联网平台设备证书、发现子设备上下线和将来自物联网平台的消息发送给子设备等功能，均由网关厂商自行实现，或网关厂商与子设备厂商定义协议实现。

3. 网关查询与当前子设备是否有拓扑关系。

如果网关与子设备间已存在拓扑关系，则忽略第5步。

4. （可选）网关向物联网平台上报子设备的ProductKey和DeviceName，动态注册子设备。

此步骤仅适用于网关未获取到子设备的DeviceSecret，且您已在控制台开启了子设备的动态注册功能的场景。

5. （可选）网关添加与子设备的拓扑关系。

如果网关与子设备间目前不存在拓扑关系，需建立拓扑关系。

6. 网关代理子设备向物联网平台发起连接请求。

具体开发指南，请参见[Link SDK](#)中的子设备管理文档。

## 实践示例

网关接入物联网平台操作示例，请参见[子设备接入物联网平台](#)。

## 10 CA证书管理

物联网平台支持使用数字证书进行设备接入认证。使用数字证书，需先在物联网平台注册CA证书，然后将数字设备证书与设备身份相绑定。本文介绍如何在物联网平台注册CA证书和绑定设备证书。

### 前提条件

使用私有CA证书进行设备认证，需在创建产品时，选择**认证方式为X.509证书**；并在**使用私有CA证书**下选择**是**。

### ← 创建产品 (设备模型)

\* 产品名称

CA证书

\* 所属品类 ?

☐ 标准品类 ☒ 自定义品类

\* 节点类型

  
直连设备

  
网关子设备

  
网关设备

连网与数据

\* 连网方式

WiFi

\* 数据格式 ?

ICA 标准数据格式 (Alink JSON)

\* 认证方式 ?

X.509证书

\* 使用私有 CA 证书

☒ 是 ☐ 否

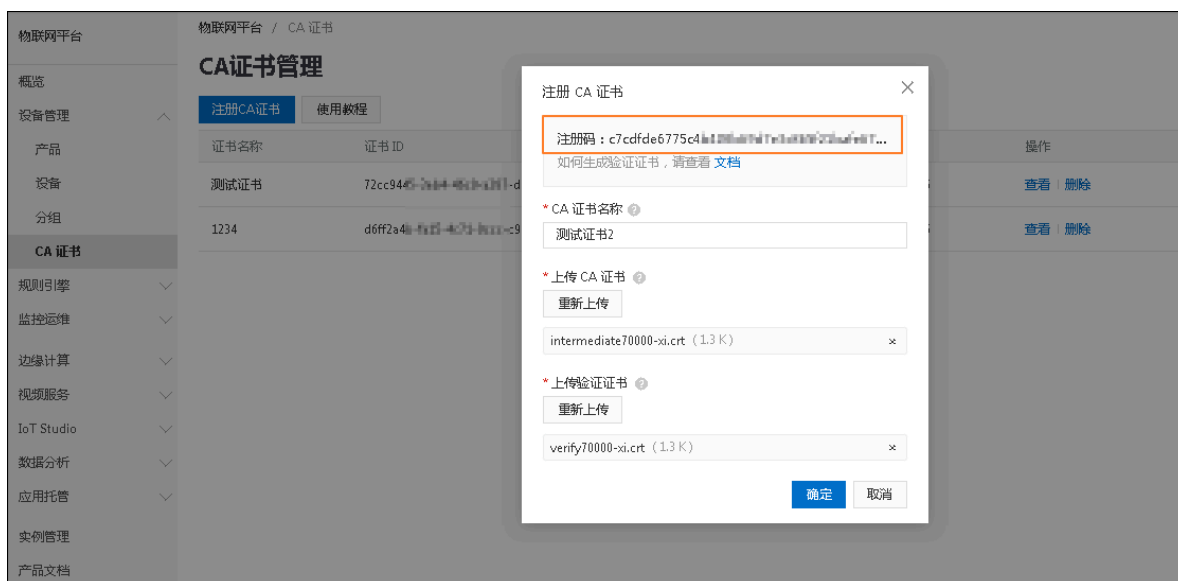
### 限制说明

- 仅MQTT协议直连的设备可使用私有CA证书。
- 目前仅华东2（上海）地域支持使用私有CA证书。

- 连网方式为LoRaWAN的产品不支持使用私有CA证书。
- 使用私有CA证书时，只支持RSA算法签名的设备证书。
- 一个阿里云账号最多可注册10个CA证书。

## 注册CA证书

1. 登录[物联网平台控制台](#)。
2. 在左侧导航栏，选择**设备管理 > CA证书**。
3. 在**CA证书管理**页，单击**注册CA证书**。
4. 在**注册CA证书**对话框中，输入证书名称，上传您的CA证书和验证证书，单击**确定**。



字段	说明
CA证书名称	支持中文汉字、英文字母、数字和下划线（_），长度限制4-30字符。
上传CA证书	上传您的CA证书。 CA证书文件仅支持.cer、.crt和.pem格式。

字段	说明
上传验证证书	<p>上传使用CA证书对应的私钥创建的验证证书，用来证明您拥有该CA证书。</p> <p>验证证书文件仅支持.cer、.crt和.pem格式。</p> <p>下面以使用OpenSSL为例，介绍创建验证证书的操作步骤。</p> <p><b>a. 生成私钥验证证书的密钥对。</b></p> <p>生成密钥对的命令如下：</p> <pre>openssl genrsa -out verificationCert.key 2048</pre> <p><b>b. 使用注册CA证书对话框上方的注册码创建CSR。</b></p> <p>创建CSR的命令如下：</p> <pre>openssl req -new -key verificationCert.key -out verificationCert.csr</pre> <p>从<b>注册CA证书</b>对话框中复制注册码，并粘贴在Common Name字段中。</p> <pre>..... Common Name (e.g. server FQDN or YOUR name) []: c7cdfde677 5c4b408b69d7e3c865f21baf67937dc9a483ebbf7e6997b7b**** .....</pre> <p><b>c. 使用由CA证书私钥签名的CSR创建验证证书。</b></p> <p>创建验证证书的命令如下：</p> <pre>openssl x509 -req -in verificationCert.csr -CA yourCA.cer -CAkey yourPrivateKey.key -CAcreateserial -out verificationCert.crt - days 300 -sha512</pre>

证书注册成功后，显示在**CA证书管理**页的CA证书列表中。

物联网平台 / CA证书				
<b>CA证书管理</b>				
<a href="#">注册CA证书</a> <a href="#">使用教程</a>				
证书名称	证书ID	生效时间	过期时间	操作
测试证书	72111405-1a64-44c1-a2d7-8504c3798004	2019-12-04 10:02:35	2029-10-22 10:02:35	<a href="#">查看</a> <a href="#">删除</a>
1234	d6072a4b-fa25-4e71-b0cc-c85a022a41b5	2019-12-04 10:02:35	2029-10-22 10:02:35	<a href="#">查看</a> <a href="#">删除</a>

单击证书对应的**查看**，进入**CA证书详情页**，可查看证书信息和绑定设备证书。

## 获取设备证书SN

您需要为每个设备签发对应的设备证书，获取设备证书SN。



**说明：**

- 设备证书SN必须在同一颁发者下具有唯一性。
- 证书签发后，请记录设备证书SN。将物联网平台设备与您的设备证书相绑定时，需使用该信息。

签发设备证书需遵循以下要求。

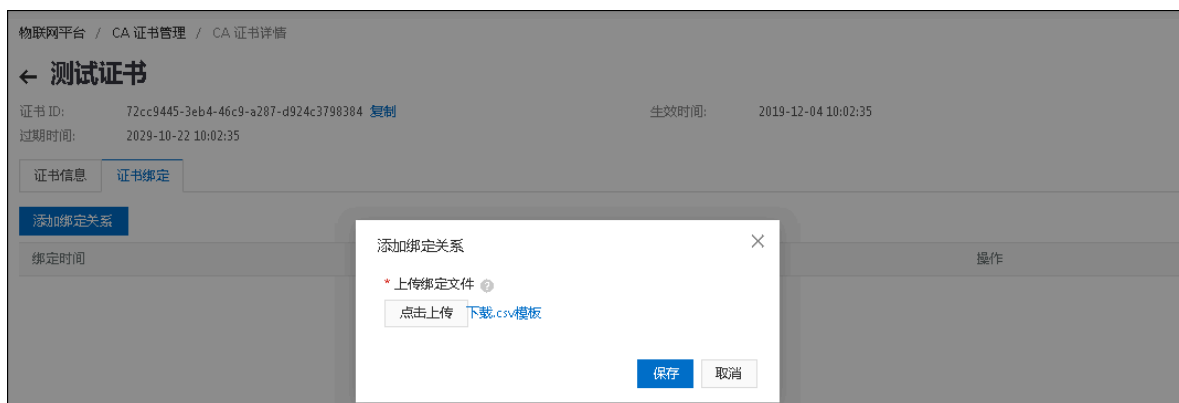
表 10-1: 设备证书格式要求

设备证书	<p>设备证书文件格式要求：</p> <ul style="list-style-type: none"><li>• 后缀名：.cer。</li><li>• 文件命格式：[ProductKey]_[DeviceName]_[CA_SN].cer。</li><li>• 内容格式： <pre>-----BEGIN CERTIFICATE----- 证书内容：Ascii字符串 -----END CERTIFICATE-----</pre></li></ul>
设备证书私钥	<p>设备证书私钥格式要求：</p> <ul style="list-style-type: none"><li>• 后缀名：.key。</li><li>• 文件名格式：[ProductKey]_[DeviceName]_[CA_SN].key。</li><li>• 内容格式： <pre>-----BEGIN RSA PRIVATE KEY----- 私钥内容：二进制字符串 -----END RSA PRIVATE KEY-----</pre></li></ul>

### 绑定设备与设备证书SN

CA证书注册成功后，需在该CA证书下，将您已获得的设备证书SN和物联网平台上的设备身份信息（ProductKey和DeviceName）相绑定。

1. 在**CA证书管理**页，单击证书对应的**查看**。
2. 在**CA证书详情页**，单击**证书绑定** > **添加绑定关系**。



3. 单击**下载.CSV模板**，下载设备证书信息填写模板。

4. 在模板文件中，填入您要绑定的设备证书信息，包括ProductKey、DeviceName和CertSN（即您的设备证书SN）。填写并保存文件后，单击**点击上传**上传文件。

设备证书SN是一个16进制的字符串，是您签发设备证书后，获取并记录下来的。您也可以通过读取您的设备证书的**SerialNumber**域获取。

**说明：**

- 一个文件中最多可包含10,000条绑定记录。
- 列表中的所有设备必须在同一个产品下。

5. 单击**保存**，完成文件上传。

物联网平台系统会对上传的文件进行解析和验证。验证通过后，才能进行绑定。

**配置设备接入物联网平台**

开发使用第三方CA证书进行身份验证的设备端时，无需配置设备的ProductKey和DeviceName信息，而需配置CA证书主题和设备证书SN。设备上线时，物联网平台根据设备上报的CA证书主题和设备证书SN进行身份验证。身份验证通过，则向设备下发ProductKey和DeviceName。具体设备端配置方法，请参见[#unique\\_56](#)中“设备端认证配置”。

# 11 Alink协议

---

## 11.1 Alink协议

物联网平台为设备端开发提供了SDK，这些SDK已封装了设备端与物联网平台的交互协议。您可以直接使用设备端SDK来进行开发。如果嵌入式环境复杂，已提供的设备端SDK不能满足您的需求，请参见本文，自行封装Alink协议数据，建立设备与物联网平台的通信。

物联网平台为设备端开发提供的各语言SDK，请参见[设备端SDK](#)。

Alink协议是针对物联网开发领域设计的一种数据交换规范，数据格式是JSON，用于设备端和物联网平台的双向通信，更便捷地实现和规范了设备端和物联网平台之间的业务数据交互。

以下为您介绍Alink协议下，设备的上线流程和数据上下行原理。

### 上线流程

设备上线流程，可以按照设备类型，分为直连设备接入与子设备接入。主要包括：设备注册、上线和数据上报三个流程。

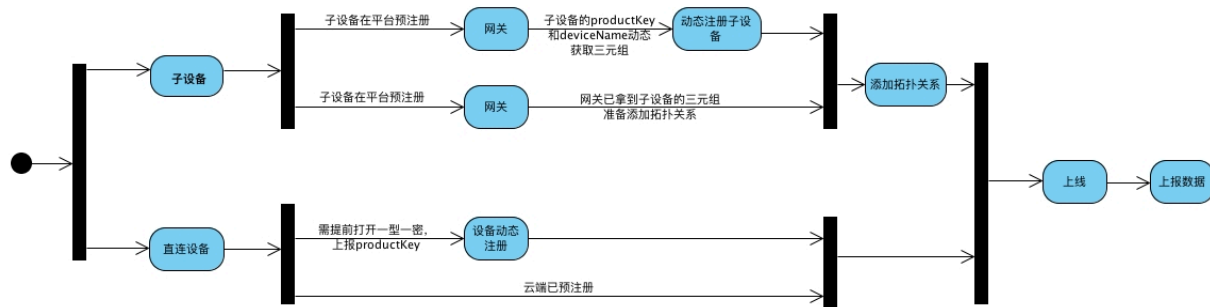
直连设备接入有两种方式：

- 使用[一机一密](#)方式提前烧录设备证书（ProductKey、DeviceName和DeviceSecret），注册设备，上线，然后上报数据。
- 使用[一型一密](#)动态注册提前烧录产品证书（ProductKey和ProductSecret），注册设备，上线，然后上报数据。

子设备接入流程通过网关发起，具体接入方式有两种：

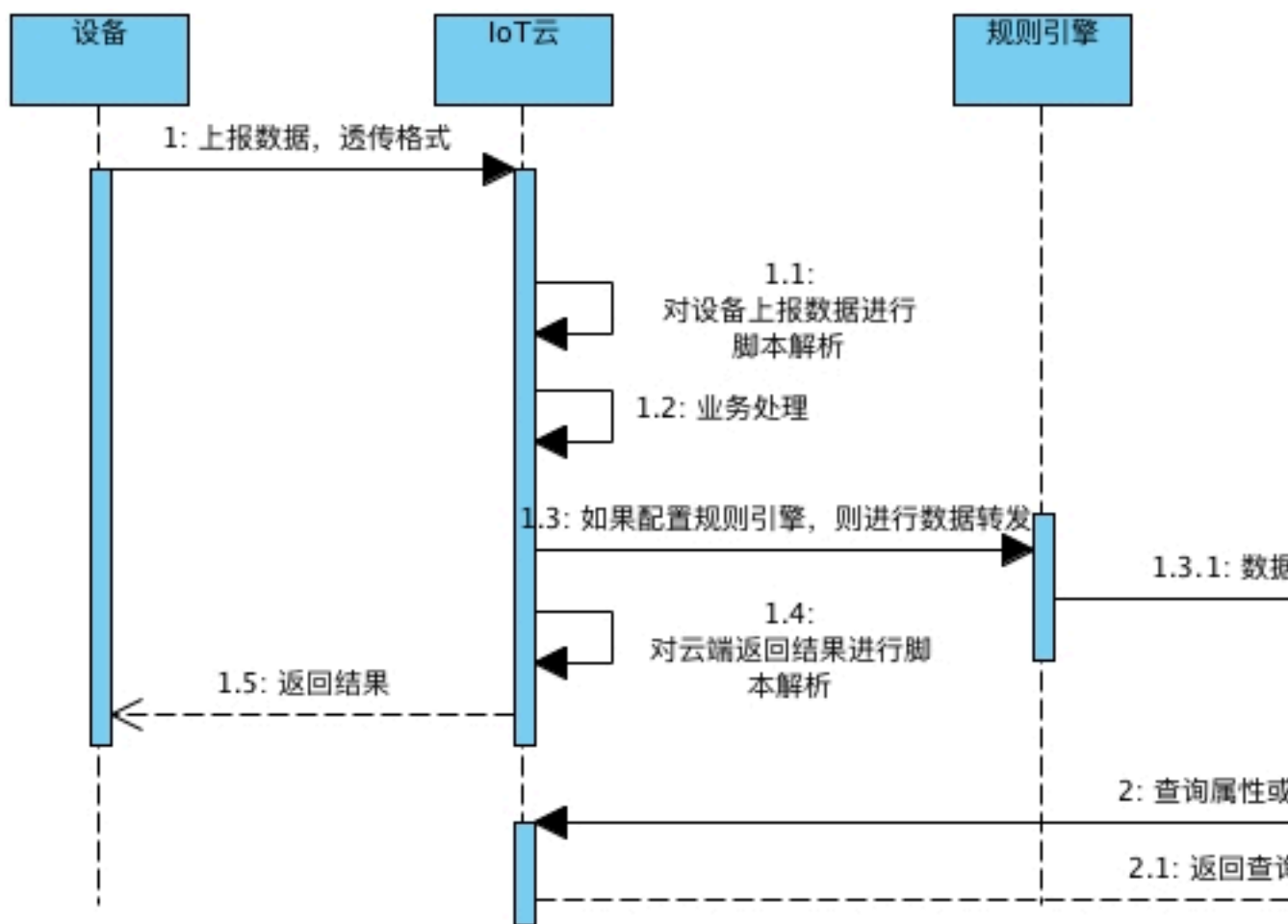
- 使用[一机一密](#)提前烧录设备证书（ProductKey、DeviceName和DeviceSecret），子设备上报设备证书给网关，网关添加拓扑关系，复用网关的通道上报数据。
- 使用动态注册方式提前烧录ProductKey，子设备上报ProductKey和DeviceName给网关，物联网平台校验DeviceName成功后，下发DeviceSecret。子设备将获得的设备证书信息上报网关，网关添加拓扑关系，通过网关的通道上报数据。





## 设备上报属性或事件

- 透传格式（透传/自定义）数据



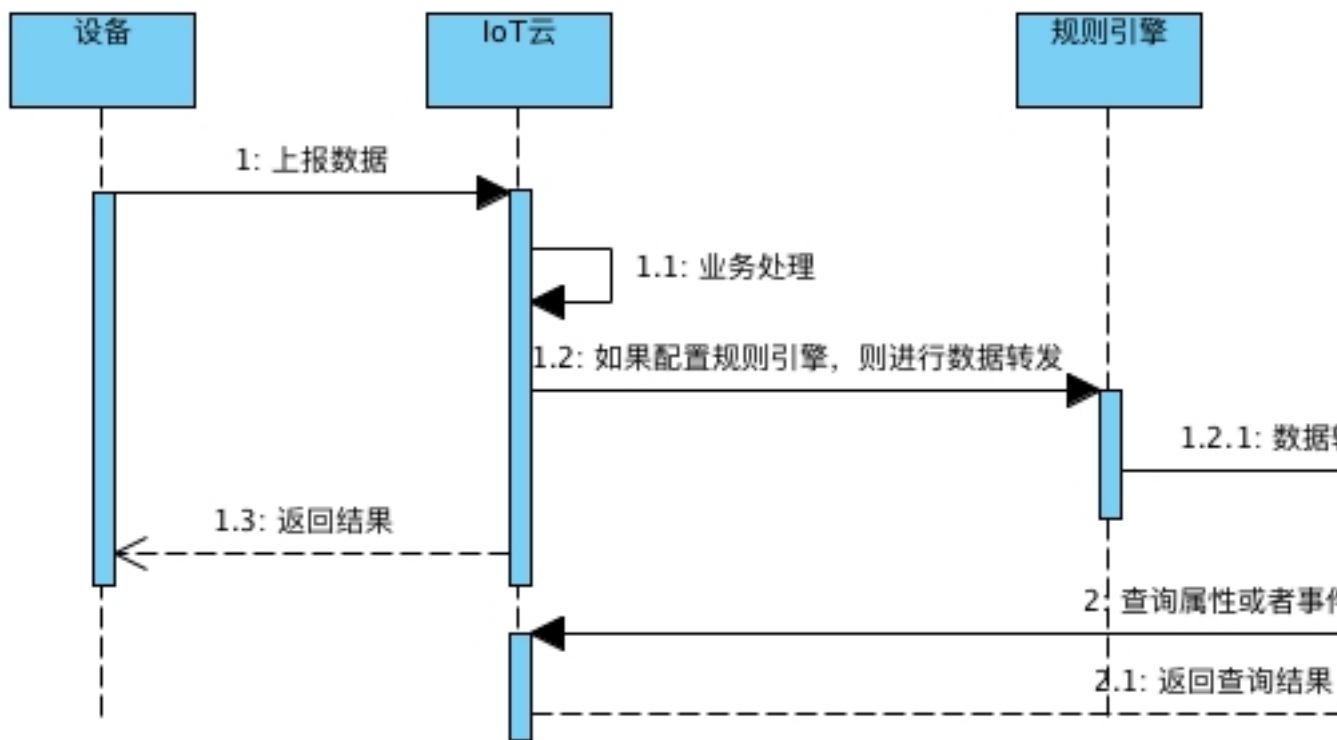
- 设备通过透传格式数据的Topic，上报透传数据。
- 物联网平台通过数据解析脚本先对设备上报的数据进行解析。调用脚本中的rawDataToProtocol方法，将设备上报的数据转换为物联网平台标准数据格式（Alink JSON格式）。
- 物联网平台使用转换后的Alink JSON格式数据进行业务处理。



### 说明：

如果配置了规则引擎数据流转，则会将数据流转到规则引擎配置的目的云产品中。

- 规则引擎获取到的数据是经过脚本解析之后的数据。
  - 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/property/post`，获取设备属性。
  - 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`，获取设备事件。
4. 调用数据解析脚本中的`protocolToRawData`方法，对结果数据进行格式转换，将数据解析为设备可以接收的数据格式。
  5. 推送解析后的返回结果数据给设备。
  6. 您可以通过[#unique\\_61](#)接口查询设备上报的属性历史数据，通过[#unique\\_62](#)接口查询设备上报的事件历史数据。
- 非透传格式（Alink JSON）数据



1. 设备使用非透传格式数据的Topic，上报数据。
2. 物联网平台进行业务处理。



#### 说明：

如果配置了规则引擎，则通过规则引擎将数据输出到规则引擎配置的目的云产品中。

- 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/property/post`，获取设备属性。

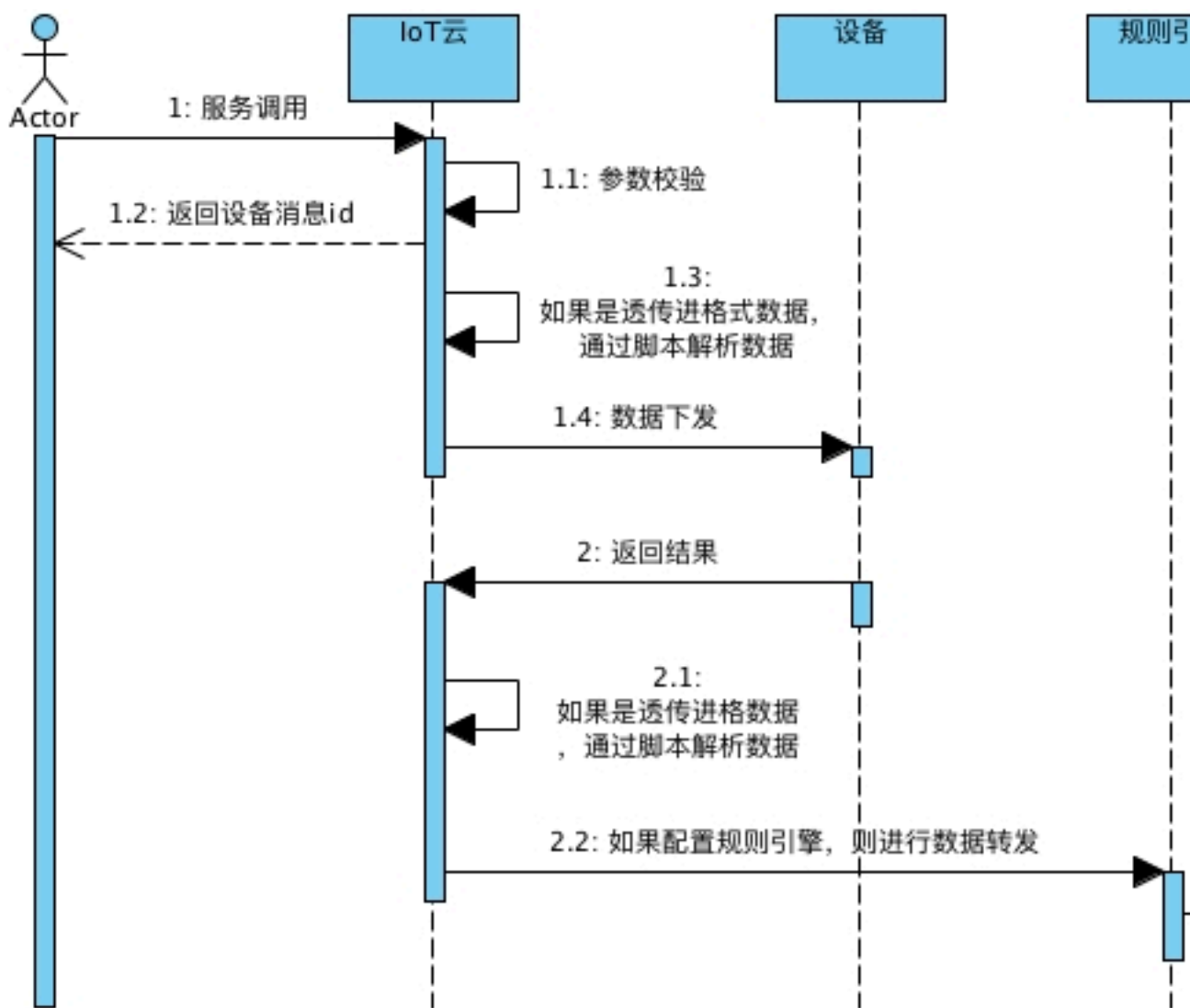
- 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`，获取设备事件。

3. 物联网平台返回处理结果。

4. 您可以通过[#unique\\_61](#)接口查询设备上报的属性历史数据，通过[#unique\\_62](#)接口查询设备上报的事件历史数据。

### 调用设备服务或设置属性

- 异步服务调用或属性设置



1. 设置属性或调用服务。



#### 说明:

- 设置属性：调用[#unique\\_63](#)接口为设备设置具体属性。

- 调用服务：调用#unique\_64接口来异步调用服务（定义服务时，调用方式选择为异步的服务即为异步调用）。

2. 物联网平台对您提交的参数进行校验。
3. 物联网平台采用异步调用方式下发数据给设备，并返回调用操作结果。若没有报错，则结果中携带下发给设备的消息ID。

**说明：**

对于透传格式（透传/自定义）数据，则会调用数据解析脚本中的protocolToRawData方法，对数据进行数据格式转换后，再将转换后的数据下发给设备。

4. 设备收到数据后，进行业务处理。

**说明：**

- 如果是透传格式（透传/自定义）数据，则使用透传格式数据的Topic。
- 如果是非透传格式（Alink JSON）数据，则使用非透传格式数据的Topic。

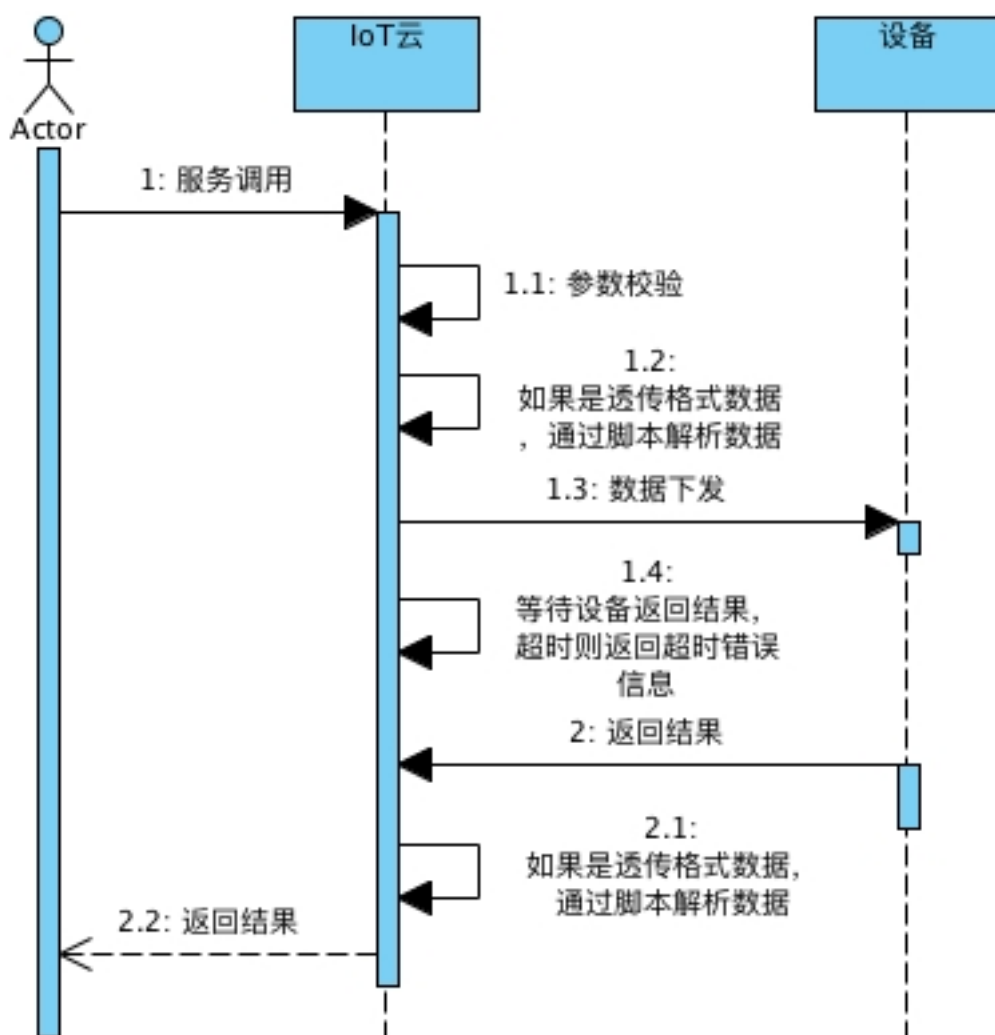
5. 设备完成业务处理后，返回处理结果给物联网平台。

6. 物联网平台收到处理结果的后续操作：

- 如果是透传格式（透传/自定义）数据，将调用数据解析脚本中的rawDataToProtocol方法，对设备返回的结果进行数据格式转换。
- 如果配置了规则引擎数据流转，则将数据流转到规则引擎配置的目的Topic或云产品中。

- 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：/sys/{productKey}/{deviceName}/thing/downlink/reply/message，获取异步调用的返回结果。
- 对于透传格式（透传/自定义）数据，规则引擎获取到的数据是经过脚本解析之后的数据。

- 同步服务调用



1. 通过调用[#unique\\_64](#)接口来调用同步服务（定义服务时，调用方式选择为同步的服务即为同步调用）。
2. 物联网平台对您提交的参数进行校验。
3. 使用同步调用方式，调用RRPC的Topic，下发数据给设备，物联网平台同步等待设备返回结果。

**说明：**

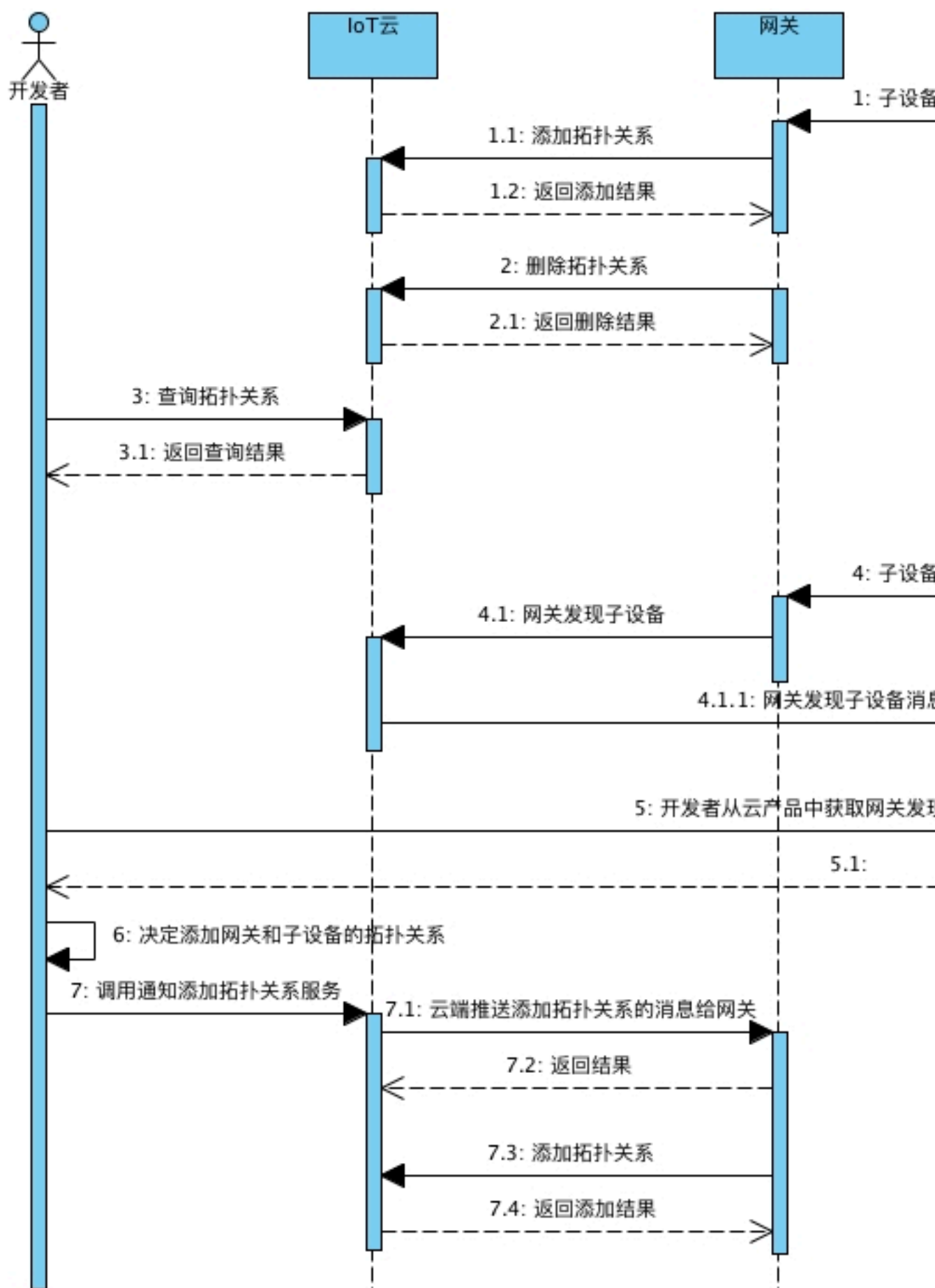
对于透传格式（透传/自定义）数据，则会先调用数据解析脚本中的protocolToRawData方法，对数据进行数据格式转换后，再将格式转换后的数据下发给设备。

4. 设备完成处理业务后，返回处理结果。若超时，则返回超时的错误信息。
5. 物联网平台收到设备处理结果后，返回结果给调用者。

**说明：**

对于透传格式（透传/自定义）数据，物联网平台会调用脚本中的rawDataToProtocol方法，对设备返回的结果数据进行格式转换后，再将转换后的结果数据发送给调用者。

## 拓扑关系



1. 子设备连接到网关后，网关通过添加拓扑关系Topic，添加拓扑关系，物联网平台返回添加的结果。
2. 网关可以通过删除拓扑关系的Topic，来删除网关和子设备的拓扑关系。
3. 开发者可以调用[#unique\\_65](#)接口来查询网关和子设备的拓扑关系。
4. 当添加拓扑关系需要第三方介入时，可以通过下面的步骤添加拓扑关系。
  - a. 网关通过发现设备列表的Topic，上报发现的子设备信息。
  - b. 物联网平台收到上报数据后，如果配置了规则引擎，可以通过规则引擎将数据流转对应的云产品中，开发者可以从云产品中获取该数据。
  - c. 当开发者从云产品中获取了网关发现的子设备后，可以决定是否添加与网关的拓扑关系。如果需要添加拓扑关系，可以调用[#unique\\_66](#)接口通知网关发起添加拓扑关系。
  - d. 物联网平台收到[#unique\\_66](#)接口调用后，会通知添加拓扑关系的Topic将命令推送给网关。
  - e. 网关收到通知添加拓扑关系的命令后，通过添加拓扑关系Topic，添加拓扑关系。

**说明：**

- 网关通过Topic：/sys/{productKey}/{deviceName}/thing/topo/add，添加拓扑关系。
- 网关通过Topic：/sys/{productKey}/{deviceName}/thing/topo/delete，删除拓扑关系。
- 网关通过Topic：/sys/{productKey}/{deviceName}/thing/topo/get，获取网关和子设备的拓扑关系。
- 网关通过发现设备列表的Topic：/sys/{productKey}/{deviceName}/thing/list/found，上报发现的子设备信息。
- 网关通过Topic：/sys/{productKey}/{deviceName}/thing/topo/add/notify，通知网关设备对子设备发起添加拓扑关系

## 11.2 设备身份注册

设备上线之前您需要对设备进行身份注册，标识您的设备。

接入物联网平台的设备身份注册有两种方式：

- 使用一机一密的方式。首先，在物联网平台注册设备，获取设备证书信息（ProductKey、DeviceName、DeviceSecret）做为设备唯一标识。然后，将设备证书信息预烧录到固件，固件在完成上线建连后即可向云端上报数据。



- 使用动态注册的方式，包括直连设备使用一型一密动态注册和子设备动态注册。
  - 直连设备使用一型一密动态注册的流程：
    1. 在物联网平台预注册设备，并获取产品证书（ProductKey和ProductSecret）。预注册设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
    2. 在控制台开启设备所属产品的动态注册开关。
    3. 将产品证书烧录至固件。
    4. 设备向云端发起身份认证。云端认证成功后，下发DeviceSecret。
    5. 设备使用设备证书与云端建立连接。
  - 子设备动态注册流程：
    1. 在物联网平台预注册子设备，获取ProductKey。预注册设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
    2. 在控制台开启子设备所属产品的动态注册开关。
    3. 将子设备ProductKey烧录至固件或网关上。
    4. 网关代替子设备向云端发起身份注册。

### 子设备的动态注册

数据上行。

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/sub/register`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/sub/register_reply`

请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.sub.register"
}
```

响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "iotId": "12344",
      "productKey": "1234556554",
      "deviceName": "deviceName1234",

```

```

    "deviceSecret": "xxxxxx"
  }
}
}

```

参数说明如下表。

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	List	设备动态注册的参数。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。
iotId	String	设备的唯一标识ID。
deviceSecret	String	设备密钥。
method	String	请求方法，取值thing.sub.register。
code	Integer	结果信息。

错误码说明如下表。

错误码	消息	描述
460	request parameter error	请求参数错误。
6402	topo relation cannot add by self	设备不能将自己添加为自己的子设备。
401	request auth error	签名校验失败。

### 直连设备使用一型一密动态注册

直连设备动态注册，通过HTTP发送请求。需先在控制台上，开通该产品的一型一密动态注册功能。

- URL模板：https://iot-auth.cn-shanghai.aliyuncs.com/auth/register/device
- HTTP方法：POST

请求数据格式：

```

POST /auth/register/device HTTP/1.1
Host: iot-auth.cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 123

```

```
productKey=1234556554&deviceName=deviceName1234&random=567345&sign=adfv123hdfdh&signMethod=HmacMD5
```

响应数据格式：

```
{
  "code": 200,
  "data": {
    "productKey": "1234556554",
    "deviceName": "deviceName1234",
    "deviceSecret": "adsfweafdsf"
  },
  "message": "success"
}
```

参数说明如下表。

参数	类型	说明
<b>Method</b>	String	POST
<b>Host</b>	String	Endpoint地址：iot-auth.cn-shanghai.aliyuncs.com。
<b>Content-Type</b>	String	设备发送给物联网平台的上行数据的编码格式。
<b>productKey</b>	String	产品唯一标识。
<b>deviceName</b>	String	设备名称。
<b>random</b>	String	随机数。
<b>sign</b>	String	签名。 加签方法： <ol style="list-style-type: none"><li>将所有提交给服务器的参数（<b>sign</b>、<b>signMethod</b>除外）按照字母顺序排序，然后将参数和值依次拼接（无拼接符号）。</li><li>通过<b>signMethod</b>指定的加签算法，使用产品的ProductSecret，对加签内容进行加签。</li></ol> 加签计算示例如下： <pre>hmac_sha1(productSecret, deviceName deviceName1234productKey1234556554random 123)</pre>
<b>signMethod</b>	String	签名方法，目前支持hmacmd5、hmacsha1、hmacsha256。
<b>code</b>	Integer	结果信息。
<b>deviceSecret</b>	String	设备密钥。

## 11.3 管理拓扑关系

子设备身份注册后，需由网关向物联网平台上报网关与子设备的拓扑关系，然后进行子设备上线。

子设备上线过程中，物联网平台会校验子设备的身份和与网关的拓扑关系。所有校验通过，才会建立并绑定子设备逻辑通道至网关物理通道上。子设备与物联网平台的数据上下行通信与直连设备的通信协议一致，协议上不需要露出网关信息。

删除拓扑关系后，子设备不能再通过网关上线。系统将提示拓扑关系不存在，认证不通过等错误。

### 添加设备拓扑关系

网关类型的设备，可以通过该Topic上行请求添加它和子设备之间的拓扑关系，返回成功添加拓扑关系的子设备。

数据上行

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/topo/add`
- 响应Topic: `sys/{productKey}/{deviceName}/thing/topo/add_reply`

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554",
      "sign": "xxxxxx",
      "signmethod": "hmacSha1",
      "timestamp": "1524448722000",
      "clientId": "xxxxxx"
    }
  ],
  "method": "thing.topo.add"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	List	请求入参。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。
sign	String	<p>签名。</p> <p>加签算法：</p> <ol style="list-style-type: none"> <li>1. 将所有提交给服务器的参数（<b>sign</b>，<b>signMethod</b>除外）按照字母顺序排序，然后将参数和值依次拼接（无拼接符号）。</li> <li>2. 对加签内容，需通过signMethod指定的加签算法，使用设备的DeviceSecret值，进行签名计算。</li> </ol> <p>签名计算示例：</p> <pre>sign= hmac_md5(deviceSecret, clientId23deviceNametestproductKey123timestamp1524448722000)</pre>
signmethod	String	签名方法，支持hmacSha1、hmacSha256、hmacMd5、Sha256。
timestamp	String	时间戳。
clientId	String	设备本地标记，非必填。可以设置为具体的productKey&deviceName。
method	String	请求方法，取值：thing.topo.add。

## 响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时返回的子设备信息，具体参数请参见下表data。

表 11-1: data

参数	类型	说明
deviceName	String	子设备的设备名称。
productKey	String	子设备所属产品的ProductKey。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6402	topo relation cannot add by self	设备不能把自己添加为自己的子设备。
401	request auth error	签名校验授权失败。

### 删除设备的拓扑关系

网关类型的设备，可以通过该Topic上行请求删除它和子设备之间的拓扑关系，返回成功删除拓扑关系的子设备。

数据上行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/topo/delete
- 响应Topic: /sys/{productKey}/{deviceName}/thing/topo/delete\_reply

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

}

## 请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	List	请求参数。
deviceName	String	子设备名称。
productKey	String	子设备产品Key。
method	String	请求方法。取值thing.topo.delete。

## 响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时返回的子设备信息，具体参数请参见下表data。

表 11-2: data

参数	类型	说明
deviceName	String	子设备的设备名称。
productKey	String	子设备所属产品的ProductKey。

## 错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

## 获取设备的拓扑关系

## 数据上行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/topo/get

- 响应Topic: /sys/{productKey}/{deviceName}/thing/topo/get\_reply

网关类型的设备，可以通过该Topic获取该设备和子设备的拓扑关系。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.topo.get"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，可为空。
method	String	请求方法，取值thing.topo.get。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。



## 错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。

## 发现设备列表上报

## 数据上行

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/list/found`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/list/found_reply`

在一些场景下，网关可以发现新接入的子设备。发现后，需将新接入子设备的信息上报云端，然后通过数据流转到第三方应用，选择将哪些子设备接入该网关。

## Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.list.found"
}
```

## Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

## 请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，可为空。
method	String	请求方法，取值thing.list.found。
deviceName	String	子设备的名称。

参数	类型	说明
productKey	String	子设备的产品Key。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6250	product not found	上报的子设备产品不存在。
6280	devicename not meet specs	上报的子设备的名称不符规范，设备名称支持英文字母、数字、连接号（-）、at符号（@）、_点号（.）和冒号（:），长度限制4~32。

## 通知网关添加设备拓扑关系

数据下行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify
- 响应Topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify\_reply

通知网关设备对子设备发起添加拓扑关系，可以配合发现设备列表上报功能使用。可以通过数据流转获取设备返回的结果，数据流转Topic为/{productKey}/{deviceName}/thing/downlink/reply/message。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.add.notify"
```

```
}

```

### Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

### 请求参数说明

参数	类型	说明
id	String	数据下行消息ID号，由物联网平台生成，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，可为空。
method	String	请求方法，取值thing.topo.add.notify。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。

### 响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。

### 通知网关拓扑关系变化

将拓扑关系变化通知网关。

网关订阅Topic： /sys/{productKey}/{deviceName}/thing/topo/change

操作	行为	通知方式
网关下添加子设备	添加网关与子设备的拓扑关系。	通知网关。
删除子设备	删除子设备与对应网关的拓扑关系。	

操作	行为	通知方式	
禁用子设备	禁用子设备，并禁用当前子设备与对应网关的拓扑关系。		
启用子设备	解除子设备禁用，恢复当前子设备和对应网关的拓扑关系。		

## 下行消息Alink数据格式

```
{
  "id":"123",
  "version":"1.0",
  "params":{
    "status":0, //0-创建 1-删除 2-恢复禁用 8-禁用
    "subList":[{
      "productKey":"a1hRrzd****",
      "deviceName":"abcd"
    }]
  },
  "method":"thing.topo.change"
}
```

## 参数说明

参数	类型	说明
id	String	数据下行消息ID号，由物联网平台生成，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
method	String	请求方法，取值thing.topo.change。
params	Object	请求参数，包含参数 <b>status</b> （拓扑关系状态）和 <b>sublist</b> （子设备列表）。
status	Integer	拓扑关系状态。 <ul style="list-style-type: none"> <li>0：创建</li> <li>1：删除</li> <li>2：解除禁用（启用）</li> <li>8：禁用</li> </ul>
deviceName	String	子设备的名称。
productKey	String	子设备所属产品的Key。

### Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "message": "success",
  "data": {}
}
```

## 11.4 子设备上下线

子设备上线之前，需在物联网平台为子设备注册身份，建立子设备与网关的拓扑关系。子设备上线时，物联网平台会根据拓扑关系进行子设备身份校验，以确定子设备是否具备使用网关通道的能力。



#### 说明：

子设备上下线消息，只支持QoS=0，不支持QoS=1。

### 子设备上线



#### 说明：

一个网关下，同时在线的子设备数量不能超过1500。若在线子设备数量达到1500个后，新的子设备上线请求将被拒绝。

### 数据上行

- 请求Topic: /ext/session/\${productKey}/\${deviceName}/combine/login
- 响应Topic: /ext/session/\${productKey}/\${deviceName}/combine/login\_reply



#### 说明：

因为子设备通过网关通道与物联网平台通信，以上Topic为网关设备的Topic。Topic中变量\${productKey}和\${deviceName}需替换为网关设备的对应信息。

### Alink请求数据格式

```
{
  "id": "123",
  "params": {
    "productKey": "al123455655",
    "deviceName": "device1234",
    "clientId": "al123455655&device1234",
    "timestamp": "1581417203000",
    "signMethod": "hmacmd5",
    "sign": "9B9C732412A4F84B981E1AB97CAB****",
    "cleanSession": "true"
  }
}
```

}

**说明:**

消息体中，参数`productKey`和`deviceName`的值是子设备的对应信息。

表 11-3: 请求参数说明

参数	类型	说明
<code>id</code>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<code>params</code>	Object	请求入参，包含的具体参数见下表params。

表 11-4: params

参数	类型	说明
<code>deviceName</code>	String	子设备的设备名称。
<code>productKey</code>	String	子设备所属的产品的ProductKey。
<code>sign</code>	String	<p>子设备签名。签名方法与直连设备签名方法相同。</p> <p>签名方法：</p> <ol style="list-style-type: none"> <li>将所有提交给服务器的参数（<code>sign</code>，<code>signMethod</code> 和 <code>cleanSession</code>除外）按照字母顺序排序，然后将参数和值依次拼接（无拼接符号）。</li> <li>对加签内容，通过<code>signMethod</code>指定的加签算法，并使用设备的<code>DeviceSecret</code>值，进行签名计算。</li> </ol> <p>计算结果作为<code>sign</code>的取值。</p> <p><code>sign</code>值计算方法示例如下：</p> <pre>hmac_md5(deviceSecret, clientId123455655&amp; device1234deviceNamedevice1234productKey al123455655timestamp1581417203000)</pre>
<code>signMethod</code>	String	签名方法，支持hmacSha1、hmacSha256、hmacMd5和Sha256。
<code>timestamp</code>	String	毫秒级时间戳。
<code>clientId</code>	String	设备端标识。可以为 <code>productKey&amp;deviceName</code> 。

参数	类型	说明
<b>cleanSession</b>	String	<ul style="list-style-type: none"> <li>如果取值是true，则清理所有子设备离线时的消息，即所有未接收的QoS1消息将被清除。</li> <li>如果取值是false，则不清理子设备离线时的消息。</li> </ul>

## Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "message": "success",
  "data": {
    "deviceName": "device1234",
    "productKey": "al123455655"
  }
}
```

表 11-5: 响应参数说明

参数	类型	说明
<b>id</b>	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<b>code</b>	Integer	返回结果，200代表成功。
<b>message</b>	String	结果信息。
<b>data</b>	Object	请求成功时返回的子设备信息，具体参数请参见下表data。

表 11-6: data

参数	类型	说明
<b>deviceName</b>	String	子设备的设备名称。
<b>productKey</b>	String	子设备所属产品的ProductKey。

## 错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
429	rate limit, too many subDeviceOnline msg in one minute	单个设备认证过于频繁被限流。
428	too many subdevices under gateway	网关下同时在线子设备过多。

错误码	消息	描述
6401	topo relation not exist	网关和子设备没有拓扑关系。
6100	device not found	子设备不存在。
521	device deleted	子设备已被删除。
522	device forbidden	子设备已被禁用。
6287	invalid sign	子设备密码或者签名错误。

## 子设备下线

### 数据上行

- 请求Topic: /ext/session/{productKey}/{deviceName}/combine/logout
- 响应Topic: /ext/session/{productKey}/{deviceName}/combine/logout\_reply



#### 说明:

因为子设备通过网关通道与物联网平台通信，以上Topic为网关设备的Topic。Topic中变量\${productKey}和\${deviceName}需替换为网关设备的对应信息。

### Alink请求数据格式

```
{
  "id": 123,
  "params": {
    "productKey": "al123455655",
    "deviceName": "device1234"
  }
}
```



#### 说明:

消息体中，参数productKey和deviceName的值是子设备的对应信息。

表 11-7: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
params	Object	请求入参，包含要下线的子设备信息。



表 11-8: params

参数	类型	说明
deviceName	String	子设备的设备名称。
productKey	String	子设备所属产品的ProductKey。

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "message": "success",
  "data": {
    "deviceName": "device1234",
    "productKey": "al123455655"
  }
}
```

表 11-9: 响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果，200代表成功。
message	String	结果信息。
data	Object	请求成功时，返回的下线的子设备信息。具体参数请参见下表data。

表 11-10: data

参数	类型	说明
deviceName	String	子设备的设备名称。
productKey	String	子设备所属产品的ProductKey。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
520	device no session	子设备会话不存在。

子设备接入配置，可参见[设备身份注册](#)，错误码可参见[设备端错误码](#)。

## 11.5 设备属性、事件、服务

如果产品定义了物模型，设备可以上报属性和事件信息，服务端可以下发设置属性和调用服务指令。

物模型（属性、事件、服务）数据格式，请参见[物模型格式](#)。

设备的数据上报方式有两种：ICA标准数据格式（Alink JSON）和透传/自定义。两者二选一，推荐您使用Alink JSON方式。

- ICA 标准数据格式（Alink JSON）：设备按照物联网平台定义的标准数据格式生成数据，然后上报数据。具体格式，请参见本文示例。
- 透传/自定义：设备上报原始数据如二进制数据流，阿里云物联网平台会运行您在控制台提交的数据解析脚本，将原始数据转成标准数据格式后，再进行业务处理。而云端返回的是标准Alink JSON格式，返回结果经数据解析后，再推送给设备。



### 设备上报属性

上行（透传）

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

请求数据为设备上报的原始报文。



#### 说明：

通过MQTT协议透传的数据为16进制格式。

示例如下：

```
0x00002233441232013fa00000
```



#### 说明：

透传的上报数据中，必须包含请求方法**method**参数，取值为`thing.event.property.post`。

云端返回数据格式如下：

```
{
  "id": "123",
```

```

    "code":200,
    "data":{}
  }

```

上行 (Alink JSON)

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/event/property/post`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/event/property/post_reply`

Alink请求数据格式:

```

{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": {
      "value": "on",
      "time": 1524448722000
    },
    "WF": {
      "value": 23.6,
      "time": 1524448722000
    }
  },
  "method": "thing.event.property.post"
}

```

表 11-11: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
method	String	请求方法。取值：thing.event.property.post。
params	Object	请求参数。如以上示例中，设备上报了两个属性Power和WF。具体属性信息，包含属性上报时间（time）和上报的属性值（value）。
time	Long	属性上报时间。该参数为可选字段。根据您的业务场景决定消息中是否带时间戳。如果消息频繁，需根据时间戳判断消息顺序，建议消息中带有时间戳。
value	object	上报的属性值。

Alink响应数据格式:

```

{
  "id": "123",
  "code": 200,

```

```
"data": {}
}
```

表 11-12: 响应参数说明

参数	类型	说明
id	String	消息ID号, String类型的数字, 取值范围0~4294967295, 且每个消息ID在当前设备中具有唯一性。
code	Integer	结果状态码。具体参考 <a href="#">设备端通用code</a> 。
data	String	请求成功时, 返回的数据。

表 11-13: 错误码

错误码	消息	描述
460	request parameter error	请求参数错误。
6106	map size must less than 200	一次最多只能上报200条属性。
6313	tsl service not available	物模型校验服务不可用。  物联网平台根据您定义的TSL中属性格式, 校验上报的属性信息。  校验后, 将过滤掉不合格的属性, 仅保留合格属性。即使全部属性都被过滤, 也代表着校验成功。  若校验服务不可用, 报6313错误码。

设备上报至平台的属性信息, 可以[使用规则引擎数据流转功能](#), 转发至其他云产品。具体Topic和数据格式, 请参见[设备属性上报](#)。

## 设置设备属性

通过调用[#unique\\_75](#)或[#unique\\_76](#)下发设置属性指令到设备。

下行 (透传)

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/model/down_raw`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/model/down_raw_reply`

下行 (Alink JSON)

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/service/property/set`

- 响应Topic: `/sys/{productKey}/{deviceName}/thing/service/property/set_reply`

Alink请求数据格式:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "temperature": "30.5"
  },
  "method": "thing.service.property.set"
}
```

表 11-14: 请求参数说明

参数	类型	说明
id	String	消息ID号, String类型的数字, 取值范围0~4294967295, 且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号, 目前协议版本号唯一取值为1.0。
params	Object	属性设置参数。如以上示例中, 设置属性: { "temperature": "30.5" }。
method	String	请求方法, 取值thing.service.property.set。

Alink响应数据格式:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 11-15: 响应参数说明

参数	类型	说明
id	String	消息ID号, String类型的数字, 取值范围0~4294967295, 且每个消息ID在当前设备中具有唯一性。
code	Integer	结果状态码, 具体参考 <a href="#">设备端通用code</a> 。
data	String	请求成功时, 返回的数据。

您可以[使用规则引擎数据流转功能](#), 将属性设置返回的结果, 转发至其它云产品。具体Topic和数据格式, 请参见[#unique\\_74/unique\\_74\\_Connect\\_42\\_section\\_mgr\\_2tl\\_b2b](#)。

## 设备事件上报

上行（透传）

- 请求Topic: /sys/{productKey}/{deviceName}/thing/model/up\_raw
- 响应Topic: /sys/{productKey}/{deviceName}/thing/model/up\_raw\_reply

请求数据为设备上报的原始报文，示例如下。

```
0xff0000007b00
```



### 说明：

透传的上报数据中，必须包含请求方法**method**参数，取值为thing.event.  
{tsl.event.identifier}.post。

云端返回数据格式如下：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

上行（Alink JSON）

- 请求Topic: /sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post
- 响应Topic: /sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post\_reply

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "Power": "on",
      "WF": "2"
    }
  },
  "time": 1524448722000
},
"method": "thing.event.{tsl.event.identifier}.post"
}
```

表 11-16: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。

参数	类型	说明
version	String	协议版本号，目前协议版本号唯一取值为1.0。
method	String	请求参数。取值：thing.event.{tsl.event.identifier}.post。
params	Object	上报事件的参数。
value	Object	具体的事件信息。如以上示例中 <pre> {   "Power": "on",   "WF": "2" } </pre>
time	Long	事件生成的时间戳，类型为UTC毫秒级时间。  该参数为可选字段。根据您的业务场景决定消息中是否带时间戳。如果消息频繁，需根据时间戳判断消息顺序，建议消息中带有时间戳。


Alink响应数据格式：

```

{
  "id": "123",
  "code": 200,
  "data": {}
}

```

表 11-17: 响应参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	结果状态码，具体参考 <a href="#">设备端通用code</a> 。  <div>  <b>说明：</b>            物联网平台会对设备上报的事件做校验。通过产品的TSL描述判断上报的事件是否符合定义的事件格式。不合格的事件会直接被过滤掉，并返回失败的错误码。         </div>
data	String	请求成功时，返回的数据。

Alink格式示例

假设产品中定义了一个alarm事件，它的TSL描述如下：

```
{
  "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json",
  "link": "/sys/${productKey}/airCondition/thing/",
  "profile": {
    "productKey": "${productKey}, 请替换为您的ProductKey",
    "deviceName": "airCondition, 请替换为您的Devicename"
  },
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",
      "desc": "风扇警报",
      "type": "alert",
      "required": true,
      "outputData": [
        {
          "identifier": "errorCode",
          "name": "错误码",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ]
    },
    {
      "method": "thing.event.alarm.post"
    }
  ]
}
```

当设备上报事件时，Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "errorCode": "error"
    }
  },
  "time": 1524448722000
},
{
  "method": "thing.event.alarm.post"
}
```

设备上报至平台的事件信息，可以[使用规则引擎数据流转功能](#)，转发至其他云产品。具体Topic和数据格式，请参见[设备上报事件](#)。

### 设备服务调用（异步调用）

物联网平台支持异同步调用和异步调用。[物模型定义服务](#)时，需设置此项。实现原理请参见[Alink协议中服务调用原理图](#)。



- 同步方式：通过[#unique\\_77](#)或[#unique\\_78](#)接口调用服务，物联网平台直接使用RRPC同步方式下行推送请求。此时，服务选择为同步调用方式，物联网平台订阅RRPC对应Topic。设备RRPC的集成方式，请参见[什么是RRPC](#)。
- 异步方式：通过[#unique\\_77](#)或[#unique\\_78](#)接口调用服务，物联网平台采用异步方式下行推送请求，设备也采用异步方式返回结果。此时，服务选择为异步调用方式，物联网平台订阅此处的异步响应Topic。

异步调用的结果，可以[使用规则引擎数据流转功能](#)获取。对应Topic和数据格式，请参见[#unique\\_74/unique\\_74\\_Connect\\_42\\_section\\_mgr\\_2tl\\_b2b](#)。

下行（透传）

- 请求Topic：/sys/{productKey}/{deviceName}/thing/model/down\_raw
- 响应Topic：/sys/{productKey}/{deviceName}/thing/model/down\_raw\_reply

下行（Alink JSON）


- 请求Topic：/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}
- 响应Topic：/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}\_reply

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": "on",
    "WF": "2"
  },
  "method": "thing.service.{tsl.service.identifier}"
}
```

表 11-18: 请求参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。

参数	类型	说明
params	Object	<p>服务调用参数。包含服务标识符和服务的值。如以上示例中：</p> <pre>{   "Power": "on",   "WF": "2" }</pre>
method	String	<p>请求方法：thing.service.{tsl.service.identifier}。</p> <div>  <b>说明：</b>  {tsl.service.identifier}为物模型中定义的服务标识符（Identifier）。请参见<a href="#">单个添加物模型</a>。 </div>

Alink响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 11-19: 返回参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	结果状态码，具体参考 <a href="#">设备端通用code</a> 。
data	String	<p>返回的结果信息。</p> <p>data参数的值和物模型定义相关。如果服务没有返回结果，则data的值为空。如果服务有返回结果，则返回的数据会严格遵循服务的定义。</p>

Alink格式示例

例如产品中定义了服务SetWeight，它的TSL描述如下：

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "productKey": "testProduct01"
  }
}
```

```

},
"services": [
  {
    "outputData": [
      {
        "identifier": "OldWeight",
        "dataType": {
          "specs": {
            "unit": "kg",
            "min": "0",
            "max": "200",
            "step": "1"
          },
          "type": "double"
        },
        "name": "OldWeight"
      },
      {
        "identifier": "CollectTime",
        "dataType": {
          "specs": {
            "length": "2048"
          },
          "type": "text"
        },
        "name": "CollectTime"
      }
    ],
    "identifier": "SetWeight",
    "inputData": [
      {
        "identifier": "NewWeight",
        "dataType": {
          "specs": {
            "unit": "kg",
            "min": "0",
            "max": "200",
            "step": "1"
          },
          "type": "double"
        },
        "name": "NewWeight"
      }
    ],
    "method": "thing.service.SetWeight",
    "name": "设置重量",
    "required": false,
    "callType": "async"
  }
]
}

```

当调用服务时，Alink请求数据格式：

```

{
  "method": "thing.service.SetWeight",
  "id": "105917531",
  "params": {
    "NewWeight": 100.8
  },
  "version": "1.0.0"
}

```

```
}
```

Alink响应数据格式：

```
{
  "id": "105917531",
  "code": 200,
  "data": {
    "CollectTime": "1536228947682",
    "OldWeight": 100.101
  }
}
```

## 网关批量上报数据

网关类型的设备可以批量上报属性和事件，也可以代其子设备批量上报属性和事件。



### 说明：

- 一次最多可上报200个属性，20个事件。
- 子设备数据条数限制为20。

上行（透传）

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

请求数据为设备上报的原始报文，示例如下。

```
0xff0000007b00
```



### 说明：

透传的上报数据中，必须包含请求方法**method**参数，取值为`thing.event.property.pack.post`。

云端返回数据格式如下：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

上行（Alink JSON）

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/event/property/pack/post`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/event/property/pack/post_reply`

Alink 请求数据格式：

```
{
  "id": "123",
```

```
"version": "1.0",
"params": {
  "properties": {
    "Power": {
      "value": "on",
      "time": 1524448722000
    },
    "WF": {
      "value": {},
      "time": 1524448722000
    }
  },
  "events": {
    "alarmEvent1": {
      "value": {
        "param1": "on",
        "param2": "2"
      },
      "time": 1524448722000
    },
    "alertEvent2": {
      "value": {
        "param1": "on",
        "param2": "2"
      },
      "time": 1524448722000
    }
  },
  "subDevices": [
    {
      "identity": {
        "productKey": "",
        "deviceName": ""
      },
      "properties": {
        "Power": {
          "value": "on",
          "time": 1524448722000
        },
        "WF": {
          "value": {},
          "time": 1524448722000
        }
      },
      "events": {
        "alarmEvent1": {
          "value": {
            "param1": "on",
            "param2": "2"
          },
          "time": 1524448722000
        },
        "alertEvent2": {
          "value": {
            "param1": "on",
            "param2": "2"
          },
          "time": 1524448722000
        }
      }
    }
  ]
},
"method": "thing.event.property.pack.post"
```

```
}

```

表 11-20: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数。
properties	Object	属性，包含属性标识符、属性值 <b>value</b> 和属性生成的时间 <b>time</b> 。  其中， <b>time</b> 参数为可选字段。根据您的业务场景决定消息中是否带时间戳。如果消息频繁，需根据时间戳判断消息顺序，建议消息中带有时间戳。
events	Object	事件，包含事件标识符、事件输出参数 <b>value</b> 和事件生成的时间 <b>time</b> 。  其中， <b>time</b> 参数为可选字段。根据您的业务场景决定消息中是否带时间戳。如果消息频繁，需根据时间戳判断消息顺序，建议消息中带有时间戳。
subDevices	Object	子设备信息。
productKey	String	子设备产品key。
deviceName	String	子设备名称。
method	String	请求参数。取值：thing.event.property.pack.post。


Alink 响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

```
}

```

表 11-21: 响应参数说明

参数	类型	说明
id	String	消息ID, String类型的数字, 取值范围0~4294967295, 且每个消息ID在当前设备中具有唯一性。
code	Integer	返回结果, 200代表成功。  <div>  <b>说明:</b>            系统会校验设备、拓扑关系、及上报的属性和事件都符合产品物模型（TSL）中的定义。如果其中任何一项校验不通过, 则上报数据失败。         </div>
data	Object	请求成功时的返回结果。

### 物模型历史数据上报

数据上行。

- 请求Topic: /sys/{productKey}/{deviceName}/thing/event/property/history/post
- 响应Topic: /sys/{productKey}/{deviceName}/thing/event/property/history/post\_reply

Alink请求数据格式:

```
{
  "id": 123,
  "version": "1.0",
  "method": "thing.event.property.history.post",
  "params": [
    {
      "identity": {
        "productKey": "",
        "deviceName": ""
      },
      "properties": [
        {
          "Power": {
            "value": "on",
            "time": 123456
          },
          "WF": {
            "value": "3",
            "time": 123456
          }
        }
      ]
    },
    {
      "Power": {
        "value": "on",
        "time": 123456
      }
    }
  ]
}
```

```
    },
    "WF": {
      "value": "3",
      "time": 123456
    }
  },
  ],
  "events": [
    {
      "alarmEvent": {
        "value": {
          "Power": "on",
          "WF": "2"
        },
        "time": 123456
      },
      "alertEvent": {
        "value": {
          "Power": "off",
          "WF": "3"
        },
        "time": 123456
      }
    }
  ],
  },
  {
    "identity": {
      "productKey": "",
      "deviceName": ""
    },
    "properties": [
      {
        "Power": {
          "value": "on",
          "time": 123456
        },
        "WF": {
          "value": "3",
          "time": 123456
        }
      }
    ],
    "events": [
      {
        "alarmEvent": {
          "value": {
            "Power": "on",
            "WF": "2"
          },
          "time": 123456
        },
        "alertEvent": {
          "value": {
            "Power": "off",
            "WF": "3"
          },
          "time": 123456
        }
      }
    ]
  }
]
```



}

表 11-22: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
method	String	请求方法，取值thing.event.property.history.post。
params	Object	请求参数。
identity	String	数据所属设备的身份证书信息，包含参数 <b>productKey</b> 和 <b>deviceName</b> 。 <div>  <b>说明：</b>            直连设备仅能上报自己的物模型历史数据。网关设备可以上报其子设备的物模型历史数据。网关上报子设备历史数据时，<b>identity</b>为子设备的信息。         </div>
properties	Object	属性，包含属性标识符、属性值 <b>value</b> 和属性生成的时间 <b>time</b> 。
events	Object	事件，包含事件标识符、事件输出参数 <b>value</b> 和事件生成的时间 <b>time</b> 。

Alink响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

设备上报至平台的物模型历史数据，可以[使用规则引擎数据流转功能](#)，转发至其他云产品。具体Topic和数据格式，请参见[物模型历史数据格式](#)。

## 11.6 设备期望属性值

在云端设置设备期望属性值后，若设备在线，将实时更新设备属性状态；若设备离线，期望属性值将缓存云端，待设备上线后，获取期望属性值，并更新属性状态。本文讲述设备期望属性值的数据格式。

### 获取期望属性值

上行（AlinkJSON）

设备向云端请求获取设备属性的期望值。

- 请求Topic: `/sys/{productKey}/{deviceName}/thing/property/desired/get`
- 响应Topic: `/sys/{productKey}/{deviceName}/thing/property/desired/get_reply`

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    "power",
    "temperature"
  ],
  "method": "thing.property.desired.get"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {
    "power": {
      "value": "on",
      "version": 2
    }
  }
}
```

表 11-23: 请求参数描述

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。


参数	类型	说明
params	List	要获取期望值的属性标识符（Identifier）列表。 如示例中列举了两个属性的标识符： <pre>[   "power",   "temperature" ]</pre>
method	String	请求方法，取值thing.property.desired.get。

表 11-24: 返回参数描述

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	结果信息，具体参考 <a href="#">设备端通用code</a> 。
data	Object	返回的期望值信息。 示例中，返回了属性power的期望值数据，包含期望值 <b>value</b> 和当前期望值版本 <b>version</b> 。 <pre>{   "power": {     "value": "on",     "version": 2   } }</pre>  <b>说明：</b> 若未在云端设置过该属性的期望值，或期望属性值已被清空，返回对象中不包含该属性的标识符。如示例中，属性temperature无期望值，返回数据中不包含该属性标识符。  <b>Data</b> 所包含的参数具体说明，请见下表data。

表 11-25: data

参数	类型	说明
key	String	key即属性的标识符。如示例中为power。
value	Object	期望属性值。

参数	类型	说明
version	Integer	<p>当前期望属性值的版本。</p> <div>  <b>说明：</b>            首次设置期望属性值后，期望值版本为1。以后每次设置后，期望值版本号自动加1。         </div>

### 清空期望属性值

上行（Alink JSON）

设备清除云端设备的期望属性值。

- 请求Topic: /sys/{productKey}/{deviceName}/thing/property/desired/delete
- 响应Topic: /sys/{productKey}/{deviceName}/thing/property/desired/delete\_reply

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "power": {
      "version": 1
    },
    "temperature": {}
  },
  "method": "thing.property.desired.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 11-26: 请求参数描述

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。

参数	类型	说明
params	List	<p>要清除期望值的属性信息列表。传入数据包含属性的标识符和期望值版本<b>version</b>。如：</p> <pre>{   "power": {     "version": 1   },   "temperature": {} }</pre> <p>params所包含的参数具体说明，请见下表<a href="#">params</a>。</p>
method	String	请求方法，取值thing.property.desired.delete。

表 11-27: params


参数	类型	说明
key	String	key即属性的标识符。如示例中，列出了power和temperature两个属性标识符。
version	Integer	<p>要删除期望属性值的版本号。</p> <div>  <b>说明：</b> <ul style="list-style-type: none"> <li><b>version</b>版本号可从Topic: /sys/{productKey}/{deviceName}/thing/property/desired/get 获取。</li> <li>如果指定<b>version</b>为2，则表示云端最新版本是2时执行清除。如果指定版本为2，但是云端最新版本是3，则忽略这个清除请求。</li> <li>若请求中，未指定要清除的期望值版本<b>version</b>，则不验证版本号，该属性的期望值将被清除。</li> </ul> </div>

表 11-28: 返回参数描述

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	结果信息，具体参考 <a href="#">设备端通用code</a> 。
data	String	返回数据。

## 11.7 子设备配置下发网关

云端将网关下所有子设备的连接配置和各子设备所属产品的物模型下发给网关。

### 配置下发

请求Topic: /sys/{productKey}/{deviceName}/thing/model/config/push

Alink配置推送数据格式:

```
{
  "id": 123,
  "version": "1.0",
  "method": "thing.model.config.push",
  "data": {
    "digest": "",
    "digestMethod": "",
    "url": ""
  }
}
```

参数说明:

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
method	String	请求方法，取值thing.model.config.push。
data	Object	数据。
digest	String	签名，用于校验从url中获取的数据完整性。
digestMethod	String	签名方法，默认sha256。
url	String	存储子设备配置数据的对象存储URL，用于从对象存储OSS中获取子设备配置数据。

### 子设备配置数据格式

对象存储URL中的子设备配置数据示例:

```
{
  "modelList": [
    {
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
```

```
"outputData": "",
"identifier": "AngleSelfAdaption",
"inputData": [
  {
    "identifier": "test01",
    "index": 0
  }
],
"displayName": "test01"
},
{
  "properties": [
    {
      "identifier": "identifier",
      "displayName": "test02"
    },
    {
      "identifier": "identifier_01",
      "displayName": "identifier_01"
    }
  ],
  "events": [
    {
      "outputData": [
        {
          "identifier": "test01",
          "index": 0
        }
      ],
      "identifier": "event1",
      "displayName": "abc"
    }
  ]
},
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "properties": [
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      },
      "identifier": "test01",
      "registerAddress": "0x03",
      "scaling": 1,
      "operateType": "inputStatus",
      "pollingTime": 1000,
      "trigger": 1
    },
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      }
    }
  ]
},
```

```

        "identifier": "test02",
        "registerAddress": "0x05",
        "scaling": 1,
        "operateType": "coilStatus",
        "pollingTime": 1000,
        "trigger": 2
    }
  ],
  {
    "profile": {
      "productKey": "a1ZlGQv****"
    },
    "properties": [
      {
        "identifier": "test_02",
        "customize": {
          "test_02": 123
        }
      },
      {
        "identifier": "test_01",
        "customize": {
          "test01": 1
        }
      }
    ]
  }
],
"serverList": [
  {
    "baudRate": 1200,
    "protocol": "RTU",
    "byteSize": 8,
    "stopBits": 2,
    "parity": 1,
    "name": "modbus01",
    "serialPort": "0",
    "serverId": "D73251B427****"
  },
  {
    "protocol": "TCP",
    "port": 8000,
    "ip": "192.168.0.1",
    "name": "modbus02",
    "serverId": "586CB066D****"
  },
  {
    "password": "XIJTginONohPEUAYZ****==",
    "secPolicy": "Basic128Rsa15",
    "name": "server_01",
    "secMode": "Sign",
    "userName": "123",
    "serverId": "55A9D276A7E****",
    "url": "tcp:00",
    "timeout": 10
  },
  {
    "password": "hAaX5s13gwX2JwyvUk****==",
    "name": "service_09",
    "secMode": "None",
    "userName": "1234",
    "serverId": "44895C63E3F****",
    "url": "tcp:00",

```



```

    "timeout": 10
  },
  "deviceList": [
    {
      "deviceConfig": {
        "displayNamePath": "123",
        "serverId": "44895C63E3FF4013924CEF31519A****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test_02"
    },
    {
      "deviceConfig": {
        "displayNamePath": "1",
        "serverId": "55A9D276A7****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test_03"
    },
    {
      "deviceConfig": {
        "slaveId": 1,
        "serverId": "D73251B4277****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test01"
    },
    {
      "deviceConfig": {
        "slaveId": 2,
        "serverId": "586CB066D6****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test02"
    }
  ],
  "tslList": [
    {
      "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
          "outputData": [],
          "identifier": "set",
          "inputData": [
            {
              "identifier": "test02",
              "dataType": {
                "specs": {
                  "unit": "mm",
                  "min": "0",
                  "max": "1"
                },
                "type": "int"
              },
              "name": "测试功能02"
            }
          ],
          "method": "thing.service.property.set",
          "name": "set",
          "required": true,

```

```
"callType": "async",
"desc": "属性设置"
},
{
  "outputData": [
    {
      "identifier": "test01",
      "dataType": {
        "specs": {
          "unit": "m",
          "min": "0",
          "max": "1"
        },
        "type": "int"
      },
      "name": "测试功能01"
    },
    {
      "identifier": "test02",
      "dataType": {
        "specs": {
          "unit": "mm",
          "min": "0",
          "max": "1"
        },
        "type": "int"
      },
      "name": "测试功能02"
    }
  ],
  "identifier": "get",
  "inputData": [
    "test01",
    "test02"
  ],
  "method": "thing.service.property.get",
  "name": "get",
  "required": true,
  "callType": "async",
  "desc": "属性获取"
}
],
"properties": [
  {
    "identifier": "test01",
    "dataType": {
      "specs": {
        "unit": "m",
        "min": "0",
        "max": "1"
      },
      "type": "int"
    },
    "name": "测试功能01",
    "accessMode": "r",
    "required": false
  },
  {
    "identifier": "test02",
    "dataType": {
      "specs": {
        "unit": "mm",
        "min": "0",
        "max": "1"
      },
      "type": "int"
    },
    "name": "测试功能02",
    "accessMode": "r",
    "required": false
  }
]
```

```

    },
    "type": "int"
  },
  {
    "name": "测试功能02",
    "accessMode": "rw",
    "required": false
  }
],
"events": [
  {
    "outputData": [
      {
        "identifier": "test01",
        "dataType": {
          "specs": {
            "unit": "m",
            "min": "0",
            "max": "1"
          },
          "type": "int"
        },
        "name": "测试功能01"
      },
      {
        "identifier": "test02",
        "dataType": {
          "specs": {
            "unit": "mm",
            "min": "0",
            "max": "1"
          },
          "type": "int"
        },
        "name": "测试功能02"
      }
    ],
    "identifier": "post",
    "method": "thing.event.property.post",
    "name": "post",
    "type": "info",
    "required": true,
    "desc": "属性上报"
  }
],
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "services": [
    {
      "outputData": [],
      "identifier": "set",
      "inputData": [
        {
          "identifier": "identifier",
          "dataType": {
            "specs": {
              "length": "2048"
            },
            "type": "text"
          },
          "name": "7614"
        }
      ]
    }
  ]
}

```

```

    },
    {
      "identifier": "identifier_01",
      "dataType": {
        "specs": {
          "length": "2048"
        },
        "type": "text"
      },
      "name": "测试功能1"
    }
  ],
  "method": "thing.service.property.set",
  "name": "set",
  "required": true,
  "callType": "async",
  "desc": "属性设置"
},
{
  "outputData": [
    {
      "identifier": "identifier",
      "dataType": {
        "specs": {
          "length": "2048"
        },
        "type": "text"
      },
      "name": "7614"
    },
    {
      "identifier": "identifier_01",
      "dataType": {
        "specs": {
          "length": "2048"
        },
        "type": "text"
      },
      "name": "测试功能1"
    }
  ],
  "identifier": "get",
  "inputData": [
    "identifier",
    "identifier_01"
  ],
  "method": "thing.service.property.get",
  "name": "get",
  "required": true,
  "callType": "async",
  "desc": "属性获取"
},
{
  "outputData": [],
  "identifier": "AngleSelfAdaption",
  "inputData": [
    {
      "identifier": "test01",
      "dataType": {
        "specs": {
          "min": "1",
          "max": "10",
          "step": "1"
        }
      }
    }
  ],

```

```

        "type": "int"
      },
      "name": "参数1"
    }
  ],
  "method": "thing.service.AngleSelfAdaption",
  "name": "角度自适应校准",
  "required": false,
  "callType": "async"
},
"properties": [
  {
    "identifier": "identifier",
    "dataType": {
      "specs": {
        "length": "2048"
      },
      "type": "text"
    },
    "name": "7614",
    "accessMode": "rw",
    "required": true
  },
  {
    "identifier": "identifier_01",
    "dataType": {
      "specs": {
        "length": "2048"
      },
      "type": "text"
    },
    "name": "测试功能1",
    "accessMode": "rw",
    "required": false
  }
],
"events": [
  {
    "outputData": [
      {
        "identifier": "identifier",
        "dataType": {
          "specs": {
            "length": "2048"
          },
          "type": "text"
        },
        "name": "7614"
      },
      {
        "identifier": "identifier_01",
        "dataType": {
          "specs": {
            "length": "2048"
          },
          "type": "text"
        },
        "name": "测试功能1"
      }
    ]
  },
  {
    "identifier": "post",
    "method": "thing.event.property.post",
    "name": "post",

```

```
    "type": "info",
    "required": true,
    "desc": "属性上报"
  },
  {
    "outputData": [
      {
        "identifier": "test01",
        "dataType": {
          "specs": {
            "min": "1",
            "max": "20",
            "step": "1"
          },
          "type": "int"
        },
        "name": "测试参数1"
      }
    ],
    "identifier": "event1",
    "method": "thing.event.event1.post",
    "name": "event1",
    "type": "info",
    "required": false
  }
],
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.json",
  "profile": {
    "productKey": "a1ZLGQv****"
  },
  "services": [
    {
      "outputData": [],
      "identifier": "set",
      "inputData": [
        {
          "identifier": "test_01",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "1"
            },
            "type": "int"
          },
          "name": "参数1"
        },
        {
          "identifier": "test_02",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "10"
            },
            "type": "double"
          },
          "name": "参数2"
        }
      ],
      "method": "thing.service.property.set",
      "name": "set",
```

```
"required": true,
"callType": "async",
"desc": "属性设置"
},
{
  "outputData": [
    {
      "identifier": "test_01",
      "dataType": {
        "specs": {
          "min": "1",
          "max": "100",
          "step": "1"
        },
        "type": "int"
      },
      "name": "参数1"
    },
    {
      "identifier": "test_02",
      "dataType": {
        "specs": {
          "min": "1",
          "max": "100",
          "step": "10"
        },
        "type": "double"
      },
      "name": "参数2"
    }
  ],
  "identifier": "get",
  "inputData": [
    "test_01",
    "test_02"
  ],
  "method": "thing.service.property.get",
  "name": "get",
  "required": true,
  "callType": "async",
  "desc": "属性获取"
}
],
"properties": [
  {
    "identifier": "test_01",
    "dataType": {
      "specs": {
        "min": "1",
        "max": "100",
        "step": "1"
      },
      "type": "int"
    },
    "name": "参数1",
    "accessMode": "rw",
    "required": false
  },
  {
    "identifier": "test_02",
    "dataType": {
      "specs": {
        "min": "1",
        "max": "100",
```

```

        "step": "10"
      },
      "type": "double"
    },
    {
      "name": "参数2",
      "accessMode": "rw",
      "required": false
    }
  ],
  "events": [
    {
      "outputData": [
        {
          "identifier": "test_01",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "1"
            },
            "type": "int"
          },
          "name": "参数1"
        },
        {
          "identifier": "test_02",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "10"
            },
            "type": "double"
          },
          "name": "参数2"
        }
      ],
      "identifier": "post",
      "method": "thing.event.property.post",
      "name": "post",
      "type": "info",
      "required": true,
      "desc": "属性上报"
    }
  ]
}

```

参数	类型	描述
modelList	Object	网关下面所有子设备的产品扩展信息。详见 modelList 说明。
serverList	Object	网关下面所有的子设备连接通道。详见 serverList 说明。
deviceList	Object	网关下面所有子设备的连接配置。详见 deviceList 说明。



参数	类型	描述
tslList	Object	网关下面所有子设备的TSL描述。请参见 <a href="#">#unique_83</a> 。

### modelList说明

设备协议目前支持Modbus、OPC UA和自定义协议，不同协议的扩展信息不一致。

- Modbus协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "properties": [
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        }
      },
      "type": "bool"
    },
    {
      "identifier": "test01",
      "registerAddress": "0x03",
      "scaling": 1,
      "operateType": "inputStatus",
      "pollingTime": 1000,
      "trigger": 1
    },
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        }
      },
      "type": "bool"
    },
    {
      "identifier": "test02",
      "registerAddress": "0x05",
      "scaling": 1,
      "operateType": "coilStatus",
      "pollingTime": 1000,
      "trigger": 2
    }
  ]
}
```

参数说明如下：

参数	类型	说明
identifier	String	属性、事件和服务的标识符。

参数	类型	说明
operateType	String	参数类型，取值可以为： <ul style="list-style-type: none"> <li>- 线圈状态：coilStatus</li> <li>- 输入状态：inputStatus</li> <li>- 保持寄存器：holdingRegister</li> <li>- 输入寄存器：inputRegister</li> </ul>
registerAddress	String	寄存器地址。
originalDataType	Object	数据原始类型。
type	String	取值如下： <p>int16, uint16, int32, uint32, int64, uint64, float, double, string, customized data</p>
specs	Object	描述信息。
registerCount	Integer	寄存器的数据个数。
swap16	Integer	把寄存器内16位数据的前后8个bits互换。0表示false、1表示true。
reverseRegister	Integer	把原始数据32位数据的bits互换。 <ul style="list-style-type: none"> <li>- 0：不互换（false）</li> <li>- 1：互换（true）</li> </ul>
scaling	Integer	缩放因子。
pollingTime	Integer	采集间隔。
trigger	Integer	数据的上报方式： <ul style="list-style-type: none"> <li>- 1：按时上报</li> <li>- 2：变更上报</li> </ul>

- OPC UA协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "services": [
    {
      "outputData": "",
      "identifier": "AngleSelfAdaption",
      "inputData": [
        {
          "identifier": "test01",
          "index": 0
        }
      ]
    }
  ]
}
```

```
{
  "displayName": "test01"
},
"properties": [
  {
    "identifier": "identifier",
    "displayName": "test02"
  },
  {
    "identifier": "identifier_01",
    "displayName": "identifier_01"
  }
],
"events": [
  {
    "outputData": [
      {
        "identifier": "test01",
        "index": 0
      }
    ],
    "identifier": "event1",
    "displayName": "abc"
  }
]
}
```

参数说明如下：

参数	类型	说明
services	Object	服务。
properties	Object	属性。
events	Object	事件。
outputData	Object	输出参数，如事件上报数据、服务调用的返回结果。
identifier	String	属性、事件或服务的标识符。
inputData	Object	输入参数，如服务的入参。
index	Integer	索引信息。
displayName	String	属性、事件或服务的名称。

- 自定义协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv*****"
  },
  "properties": [
    {
      "identifier": "test_02",
      "customize": {
        "test_02": 123
      }
    }
  ],
}
```

```
{
  "identifier": "test_01",
  "customize": {
    "test01": 1
  }
}
```

参数说明如下：

参数	类型	说明
productKey	String	产品Key，物联网平台为产品颁发的唯一标识符。
properties	Object	属性信息集合。
identifier	String	属性标识符。
customize	Object	属性的自定义扩展信息，即定义属性时，扩展描述中填入的数据。

### serverList说明

子设备连接通道的信息分为Modbus和OPC UA协议两种。

- Modbus协议下的连接通道数据示例：

```
[
  {
    "baudRate": 1200,
    "protocol": "RTU",
    "byteSize": 8,
    "stopBits": 2,
    "parity": 1,
    "name": "modbus01",
    "serialPort": "0",
    "serverId": "D73251B427****"
  },
  {
    "protocol": "TCP",
    "port": 8000,
    "ip": "192.168.0.1",
    "name": "modbus02",
    "serverId": "586CB066D****"
  }
]
```

参数	类型	说明
protocol	String	协议类型，TCP或者RTU。
port	Integer	端口号。
ip	String	IP地址。
name	String	子设备通道名称。

参数	类型	说明
serverId	String	子设备通道ID。
baudRate	Integer	波特率。
byteSize	Integer	字节数。
stopBits	Integer	停止位。
parity	Integer	奇偶校验位。 - E：偶校验 - O：奇校验 - N：无校验
serialPort	String	串口号。

- OPC UA协议下的连接通道数据示例：

```
{
  "password": "XIJTginONohPEUAYzx****==",
  "secPolicy": "Basic128Rsa15",
  "name": "server_01",
  "secMode": "Sign",
  "userName": "123",
  "serverId": "55A9D276A7E****",
  "url": "tcp:00",
  "timeout": 10
}
```

参数说明如下：

参数	类型	说明
password	String	密码，采用AES算法加密。具体说明见下文 OPC UA的password加密方式。
secPolicy	String	加密策略，取值：None、Basic128Rsa15、Basic256。
secMode	String	加密模式，取值：None、Sign、SignAndEncrypt。
name	String	子设备通道名称。
userName	String	用户名。
serverId	String	子设备通道ID。
url	String	服务器连接地址。

参数	类型	说明
timeout	Integer	超时时间。

OPC UA的password加密方式

加密算法采用AES算法，使用128位（16字节）分组，缺省的mode为CBC，缺省的padding为PKCS5Padding，密钥使用设备的DeviceSecret，加密后的内容采用BASE64进行编码。

加解密示例代码：



**说明：**

Java版本请采用Java 9、Java 8u161、Java 7u171、Java 6u181及以上版本。Java版本过低时运行以下示例会报错，此类报错具体解释，请参考[Java Bug Database](#)。

```
private static String instance = "AES/CBC/PKCS5Padding";

private static String algorithm = "AES";

private static String charsetName = "utf-8";
/**
 * 加密算法
 *
 * @param data      待加密内容
 * @param deviceSecret 设备的密钥
 * @return
 */
public static String aesEncrypt(String data, String deviceSecret) {
    try {
        Cipher cipher = Cipher.getInstance(instance);
        byte[] raw = deviceSecret.getBytes();
        SecretKeySpec key = new SecretKeySpec(raw, algorithm);
        IvParameterSpec ivParameter = new IvParameterSpec(deviceSecret.substring(0,
16).getBytes());
        cipher.init(Cipher.ENCRYPT_MODE, key, ivParameter);
        byte[] encrypted = cipher.doFinal(data.getBytes(charsetName));

        return new BASE64Encoder().encode(encrypted);
    } catch (Exception e) {
        e.printStackTrace();
    }

    return null;
}

public static String aesDecrypt(String data, String deviceSecret) {
    try {
        byte[] raw = deviceSecret.getBytes(charsetName);
        byte[] encrypted1 = new BASE64Decoder().decodeBuffer(data);
        SecretKeySpec key = new SecretKeySpec(raw, algorithm);
        Cipher cipher = Cipher.getInstance(instance);
        IvParameterSpec ivParameter = new IvParameterSpec(deviceSecret.substring(0,
16).getBytes());
        cipher.init(Cipher.DECRYPT_MODE, key, ivParameter);
        byte[] originalBytes = cipher.doFinal(encrypted1);
        String originalString = new String(originalBytes, charsetName);
        return originalString;
    }
}
```

```
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return null;
}

public static void main(String[] args) throws Exception {
    String text = "test123";
    String secret = "testTNmjyWHQzniA8wEkTNmjyWH****";
    String data = null;
    data = aesEncrypt(text, secret);
    System.out.println(data);
    System.out.println(aesDecrypt(data, secret));
}
```

## deviceList说明

- Modbus协议下子设备的连接配置数据示例：

```
{
  "deviceConfig": {
    "slaveId": 1,
    "serverId": "D73251B42777****"
  },
  "productKey": "a1ZlGQv****",
  "deviceName": "test01"
}
```

参数说明如下：

参数	类型	取值
deviceConfig	Object	设备连接配置信息。
slaveId	Integer	从站ID。
serverId	String	子设备通道ID。
productKey	String	子设备的产品Key。
deviceName	String	子设备名称。

- OPC UA协议

```
{
  "deviceConfig": {
    "displayNamePath": "123",
    "serverId": "44895C63E3FF4013924CEF31519A****"
  },
  "productKey": "a1ZlGQv****",
  "deviceName": "test_02"
}
```

```
}
```

参数说明如下：

参数	类型	说明
deviceConfig	Object	设备连接配置信息。
productKey	String	子设备的产品Key。
deviceName	String	子设备名称。
displayNamePath	String	自定义的展示名称。
serverId	String	子设备通道ID。

## 11.8 子设备禁用、启用、删除

网关类型设备可以禁用、启用（解除禁用）和删除子设备。

### 禁用设备

下行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/disable
- 响应Topic: /sys/{productKey}/{deviceName}/thing/disable\_reply

适用于网关类型设备，使用该功能通知网关禁用子设备。云端使用异步方式推送禁用设备的消息；网关设备订阅该Topic获取消息。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.disable"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明



参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，为空即可。
method	String	请求方法，取值thing.disable。
code	Integer	结果信息，请参见 <a href="#">设备端通用code</a> 。

## 解除禁用

下行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/enable
- 响应Topic: /sys/{productKey}/{deviceName}/thing/enable\_reply

适用于网关类型设备，使用该功能通知网关重新启用被禁用的子设备。云端使用异步方式推送启用子设备的消息；网关设备订阅该Topic获取消息。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.enable"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。

参数	取值	说明
params	Object	请求参数，为空即可。
method	String	请求方法，取值thing.enable。
code	Integer	结果信息，请参见 <a href="#">设备端通用code</a> 。

## 删除设备

下行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/delete
- 响应Topic: /sys/{productKey}/{deviceName}/thing/delete\_reply

适用于网关类型设备，使用该功能通知网关删除子设备。云端使用异步方式推送删除设备的消息；网关设备订阅该Topic获取消息。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，为空即可。
method	String	请求方法，取值thing.delete。
code	String	结果信息，请参见 <a href="#">设备端通用code</a> 。

## 11.9 设备标签

设备上报部分信息，如厂商、设备型号等，可以保存为设备标签。

### 标签信息上报

上行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/update
- 响应Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/update\_reply

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature",
      "attrValue": "36.8"
    }
  ],
  "method": "thing.deviceinfo.update"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明：

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，包含标签的键 <b>attrKey</b> 和值 <b>attrValue</b> 。 params元素个数不超过200个。
method	String	请求方法，取值thing.deviceinfo.update。

参数	类型	说明
attrKey	String	标签Key。 <ul style="list-style-type: none"> <li>长度不超过64字节。</li> <li>仅允许字符集为英文大小写字母、数字和下划线。</li> <li>首字符必须是字母或者下划线。</li> </ul>
attrValue	String	标签的值。可包含中文汉字、英文字母、数字、下划线（_）、连字符（-）和点号（.）。
code	Integer	结果信息，200表示成功。

## 错误码

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

## 删除标签信息

## 上行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/delete
- 响应Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/delete\_reply

## Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature"
    }
  ],
  "method": "thing.deviceinfo.delete"
}
```

## Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

## 参数说明：

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	请求参数，包含要删除的标签键attrKey参数。
method	String	请求方法，取值thing.deviceinfo.delete。
attrKey	String	要删除标签的Key。
code	Integer	结果信息，200表示成功。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

## 11.10 TSL模板

设备可以通过上行请求获取设备的TSL模板（包含属性、服务和事件的定义）。

- 请求Topic: /sys/{productKey}/{deviceName}/thing/dsltemplate/get
- 响应Topic: /sys/{productKey}/{deviceName}/thing/dsltemplate/get\_reply

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.dsltemplate.get"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {
    "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json",
    "link": "/sys/1234556554/airCondition/thing/",
    "profile": {
      "productKey": "1234556554",
      "deviceName": "airCondition"
    },
    "properties": [
      {
        "identifier": "fan_array_property",
```

```

    "name": "风扇数组属性",
    "accessMode": "r",
    "required": true,
    "dataType": {
      "type": "array",
      "specs": {
        "size": "128",
        "item": {
          "type": "int"
        }
      }
    }
  },
  ],
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",
      "desc": "风扇警报",
      "type": "alert",
      "required": true,
      "outputData": [
        {
          "identifier": "errorCode",
          "name": "错误码",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ]
    },
    {
      "method": "thing.event.alarm.post"
    }
  ],
  "services": [
    {
      "identifier": "timeReset",
      "name": "timeReset",
      "desc": "校准时间",
      "inputData": [
        {
          "identifier": "timeZone",
          "name": "时区",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "512"
            }
          }
        }
      ]
    },
    {
      "outputData": [
        {
          "identifier": "curTime",
          "name": "当前的时间",
          "dataType": {
            "type": "date",
            "specs": {}
          }
        }
      ]
    },
    {
      "method": "thing.service.timeReset"
    }
  ]
}

```

```

    },
    {
      "identifier": "set",
      "name": "set",
      "required": true,
      "desc": "属性设置",
      "method": "thing.service.property.set",
      "inputData": [
        {
          "identifier": "fan_int_property",
          "name": "风扇整数型属性",
          "accessMode": "rw",
          "required": true,
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "100",
              "unit": "g/ml",
              "unitName": "毫升"
            }
          }
        }
      ],
      "outputData": []
    },
    {
      "identifier": "get",
      "name": "get",
      "required": true,
      "desc": "属性获取",
      "method": "thing.service.property.get",
      "inputData": [
        "array_property",
        "fan_int_property",
        "batch_enum_attr_id",
        "fan_float_property",
        "fan_double_property",
        "fan_text_property",
        "fan_date_property",
        "batch_boolean_attr_id",
        "fan_struct_property"
      ],
      "outputData": [
        {
          "identifier": "fan_array_property",
          "name": "风扇数组属性",
          "accessMode": "r",
          "required": true,
          "dataType": {
            "type": "array",
            "specs": {
              "size": "128",
              "item": {
                "type": "int"
              }
            }
          }
        }
      ]
    }
  ]
}

```

}

参数说明：

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
params	Object	为空即可。
method	String	请求方法，取值thing.dsltemplate.get。
productKey	String	产品的Key，示例中取值为1234556554。
deviceName	String	设备名称，示例中取值为airCondition。
data	Object	设备的TSL描述，具体请参见 <a href="#">物模型文档</a> 。

错误码

错误码	消息	描述
460	request parameter error	请求参数错误。
6321	tsl: device not exist in product	设备不存在。

## 11.11 固件升级

物联网平台提供固件升级与管理服务。本文介绍固件升级数据上下行Topic和Alink协议下固件升级的上下行数据格式，包括设备上报固件版本、物联网平台推送固件信息、设备上报升级进度和设备请求获取最新固件信息。

关于固件升级开发设置流程，请参见[#unique\\_88](#)和[#unique\\_89](#)。

### 设备上报固件版本

数据上行。

Topic: /ota/device/inform/\${YourProductKey}/\${YourDeviceName}

设备通过这个Topic上报当前使用的固件版本信息。



**说明：**


本Topic只支持单个模块的固件版本上报。如果设备需要上报多个模块的固件版本，请分多次上报，每次上报一个模块的固件版本信息。



Alink请求数据格式：

```
{
  "id": "123",
  "params": {
    "version": "1.0.1",
    "module": "MCU"
  }
}
```

表 11-29: 参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	设备固件的版本信息。
module	String	固件所属的模块名。  <div>  <b>说明：</b> <ul style="list-style-type: none"> <li>上报默认（default）模块的固件版本号时，可以不上报<b>module</b>参数。</li> <li>设备的默认（default）模块的固件版本号等同于整个设备的固件版本号。</li> </ul> </div>

## 物联网平台推送固件信息

数据下行。

Topic: `/ota/device/upgrade/${YourProductKey}/${YourDeviceName}`


物联网平台通过这个Topic推送固件信息，设备订阅该Topic可以获得固件信息。

Alink请求数据格式：

```
{
  "code": "1000",
  "data": {
    "size": 432945,
    "version": "2.0.0",
    "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjlpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtlf3NpAusdsv03nWxT7v4flqFyTINVAEvYzJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfj%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltDUROfbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJoiTknxR7ARasaBqhelc4zqA%2FPPLWgAKvkXba7aloo01fv4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lL06iZy%2BVio2VSDZxshI5Z3McKARWct06MWV9ABA2TTXOio40BOxuq%2B3JGoABXC54TOlo7%2F1wTLtCUqzzeliXVOK8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmK"
  }
}
```

```
MQph2cKsr8y8UfWLC6IzVjsClXTnbJBMeuWlqo5zlynS1pm7gf%2F9N3hVc6%2BEeIk0xfl
2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
  "md5": "93230c3bde425a9d7984a594ac55ea1e",
  "sign": "93230c3bde425a9d7984a594ac55****",
  "signMethod": "Md5",
  "module": "MCU"
},
{id": "1507707025",
"message": "success"
}
```

表 11-30: 参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性
message	String	结果信息。
code	String	状态码。
version	String	设备固件的版本信息。
size	Long	固件大小，单位：字节。
url	String	固件在对象存储（OSS）上的存储地址。
sign	String	固件签名。
signMethod	String	签名方法，目前支持Md5，SHA256两种签名方法。
md5	String	当签名方法为Md5时，除了会给sign赋值外还会给md5赋值。
module	String	固件所属的模块名。  <div>  <b>说明：</b>            模块名为default时，云端不下发module参数。         </div>

## 设备上报升级进度

数据上行。

Topic: /ota/device/progress/\${YourProductKey}/\${YourDeviceName}

固件升级过程中，设备可以通过这个Topic上报固件升级的进度百分比。

Alink请求数据格式：


```
{
  "id": "123",
  "params": {
    "step": "-1",
    "desc": "固件升级失败，请求不到固件信息。",
  }
}
```

```

    "module": "MCU"
  }
}

```

表 11-31: 参数说明

参数	取值	说明
<b>id</b>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<b>step</b>	String	固件升级进度信息。  取值范围： <ul style="list-style-type: none"> <li>• [1, 100] 之间的数字：表示升级进度百分比。</li> <li>• -1：表示升级失败。</li> <li>• -2：表示下载失败。</li> <li>• -3：表示校验失败。</li> <li>• -4：表示烧写失败。</li> </ul>
<b>desc</b>	String	当前步骤的描述信息。如果发生异常，此字段可承载错误信息。
<b>module</b>	String	固件所属的模块名。  <div>  <b>说明：</b>            上报默认（default）模块的固件升级进度时，可以不上报<b>module</b>参数。         </div>

## 设备请求固件信息

数据上行。

请求Topic：/sys/{productKey}/{deviceName}/thing/ota/firmware/get

响应Topic：/sys/{productKey}/{deviceName}/thing/ota/firmware/get\_reply

Alink请求数据格式：


```

{
  "id": "123",
  "version": "1.0",
  "params": {
    "module": "MCU"
  },
  "method": "thing.ota.firmware.get"
}

```

}

表 11-32: 参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	Alink协议版本，目前协议版本号唯一取值为1.0。
params	String	请求参数。
module	String	固件所属的模块名。  <div>  <b>说明：</b>            不指定则表示请求默认（default）模块的固件信息。         </div>
method	String	请求方法，取值thing.ota.firmware.get。


物联网平台收到设备请求后，响应请求。

- 下发固件信息。返回数据格式如下：

```
{
  "id": "123",
  "code": 200,
  "data": {
    "size": 93796291,
    "sign": "f8d85b250d4d787a9f483d89a974****",
    "version": "1.0.1.9.20171112.1432",
    "url": "https://the_firmware_url",
    "signMethod": "Md5",
    "md5": "f8d85b250d4d787a9f483d89a9747348",
    "module": "MCU"
  }
}
```

表 11-33: 参数说明

参数	取值	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性
code	Integer	状态码，200表示成功。
version	String	设备固件的版本信息。
size	Long	固件大小，单位：字节。
url	String	固件在对象存储（OSS）上的存储地址。

参数	取值	说明
<b>sign</b>	String	固件签名。
<b>signMethod</b>	String	签名方法。取值： <ul style="list-style-type: none"> <li>- SHA256</li> <li>- Md5</li> </ul>
<b>md5</b>	String	当签名方法为Md5时，除了会给 <b>sign</b> 赋值外还会给 <b>md5</b> 赋值。
<b>module</b>	String	固件所属的模块名。 <div>  <b>说明：</b>            模块名为default时，云端不下发<b>module</b>参数。         </div>

- 无固件信息下发。返回数据格式如下：

```
{
  "id": "123",
  "code": 200,
  "data": {
  }
}
```

## 11.12 远程配置

本文档介绍设备主动请求配置信息和物联网平台推送配置信息的Topic及Alink数据格式。

远程配置的具体使用方法，请参见用户指南中[#unique\\_91](#)文档。

### 设备主动请求配置信息

上行

- 请求Topic：/sys/{productKey}/{deviceName}/thing/config/get
- 响应Topic：/sys/{productKey}/{deviceName}/thing/config/get\_reply

Alink请求数据格式

```
{
  "id": 123,
  "version": "1.0",
  "params": {
    "configScope": "product",
    "getType": "file"
  },
  "method": "thing.config.get"
```

```
}
```

## Alink响应数据格式

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjlpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtlf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2F2FMQBqEaXPS2MvVfj%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltDUOfbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTkxr7ARasaBqhelc4zqA%2FPPLWgAKvkXba7aloo01fV4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshl5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeliXVOK8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6lZvjsClXTnbJBMeuWlqo5zlynS1pm7gf%2F9N3hVc6%2BEelk0xfl2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  }
}
```

## 参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
configScope	String	配置范围，目前只支持产品维度配置。取值：product。
getType	String	获取配置类型。目前支持文件类型，取值：file。
method	String	请求方法，取值：thing.config.get。
configId	String	配置文件的ID。
configSize	Long	配置文件大小，按字节计算。
sign	String	签名。
signMethod	String	签名方法，仅支持Sha256。
url	String	存储配置文件的对象存储（OSS）地址。
code	Integer	结果码。返回200表示成功，返回其他状态码，表示失败。具体请参见 <a href="#">设备端通用code</a> 。

## 错误码

错误码	消息	描述
6713	thing config function is not available	产品的远程配置功能不可用，需要在控制台，远程配置页面打开配置开关。
6710	no data	没有配置的数据。

## 配置推送

## 下行

- 请求Topic: /sys/{productKey}/{deviceName}/thing/config/push
- 响应Topic: /sys/{productKey}/{deviceName}/thing/config/push\_reply

设备订阅该Topic后，您在物联网控制台批量推送配置信息时，物联网平台采用异步推送方式向设备推送信息。

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6Ft5B2yfSjlpK6MGsyN1Jx5jo6mVnfBglIPTvlt5D50Tz2IHtlf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfj%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltDUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTkxR7ARasaBqhelc4zqA%2FPPLWgAKvkXba7aloo01fv4jN5JXQfAU8KLO8trJofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDXshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54Tol07%2F1wTLtsCUqzzeliXVOK8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6IzvsClXTnbJBMeuWlqo5zlynS1pm7gf%2F9N3hVc6%2BEelk0xfl2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  },
  "method": "thing.config.push"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

## 参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性
version	String	协议版本号，目前协议版本号唯一取值为1.0。
configScope	String	配置范围，目前只支持产品维度配置。取值：product。
getType	String	获取配置类型，目前支持文件类型，取值：file。
configId	String	配置的ID。
configSize	Long	配置大小，按字节计算。
sign	String	签名。
signMethod	String	签名方法，仅支持sha256。
url	String	存储配置文件的对象存储（OSS）地址。
method	String	请求方法，取值：thing.config.push。
code	Integer	结果信息，具体请参见 <a href="#">设备端通用code</a> 。

您可以使用[规则引擎数据流转功能](#)，将设备返回的响应结果转发至其他Topic和其他阿里云服务中。具体的设备响应数据Topic和数据格式请参见[设备下行指令结果数据流转](#)。

## 11.13 设备日志上报

物联网平台支持设备将本地日志上报到云端，在控制台进行查询和故障分析。



### 说明：

- 仅使用了以下Link SDK的设备端支持本地日志上报：
  - [Android SDK](#)
  - [C SDK](#)
- 您在[设备详情页](#)，开启[设备本地日志上报](#)开关后，设备才能将本地日志上报到云端。

设备本地日志的具体查询方法，请参见[#unique\\_93](#)。

### 设备获取日志配置

数据上行

- 请求Topic：/sys/\${productKey}/\${deviceName}/thing/config/log/get
- 响应Topic：/sys/\${productKey}/\${deviceName}/thing/config/log/get\_reply



## Alink请求数据格式

```
{
  "id": 123,
  "version": "1.0",
  "params": {
    "configScope": "device",
    "getType": "content"
  },
  "method": "thing.config.log.get"
}
```

表 11-34: 请求参数说明

参数	类型	说明
<b>id</b>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<b>version</b>	String	协议版本号，目前协议版本号唯一取值为1.0。
<b>configScope</b>	String	配置范围，目前日志只有设备维度配置，默认为device。
<b>getType</b>	String	获取内容类型，默认为content。因日志配置内容较少，默认直接返回内容。
<b>method</b>	String	请求方法，取值thing.property.desired.get。

## Alink响应数据格式

```
{
  "id": "123"
  "version": "1.0"
  "code": 200
  "data": {
    "getType": "content",
    "content": {
      "mode": 0,
    }
  }
}
```

表 11-35: 响应参数说明

参数	类型	说明
<b>id</b>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<b>version</b>	String	协议版本号，目前协议版本号唯一取值为1.0。

参数	类型	说明
<b>code</b>	Integer	结果码。返回200表示成功，返回其他状态码，表示失败。 结果码6717、6718请参见下表错误信息。其他结果码请参见 <a href="#">设备端通用code</a> 。
<b>getType</b>	String	获取内容类型，默认为content。因日志配置内容较少，默认直接返回内容。
<b>content</b>	String	配置文本内容。
<b>mode</b>	Integer	设备日志上报模式，0表示设备SDK不上报日志，1表示设备SDK上报日志。

表 11-36: 错误信息

错误码	原因	排查方法
6717	请求参数中 <b>getType</b> 取值非法，设备日志只支持content。	在物联网平台控制台 <b>监控运维 &gt; 日志服务 &gt; 设备本地日志</b> ，或在设备本地日志中，查看上报数据中的 <b>getType</b> 取值。
6718	请求参数中configScope取值非法，设备日志只支持device	在物联网平台控制台 <b>监控运维 &gt; 日志服务 &gt; 设备本地日志</b> ，或在设备本地日志中，查看上报数据中的 <b>configScope</b> 取值。

### 设备接收订阅云端推送日志配置

数据下行

Topic: /sys/\${productKey}/\${deviceName}/thing/config/log/push

Alink配置推送数据格式

```
{
  "id": "123"
  "version": "1.0"
  "params": {
    "getType": "content",
    "content": {
      "mode": 0
    }
  }
}
```

```
}

```

表 11-37: 参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
getType	String	获取内容类型，默认为content。因日志配置内容较少，默认直接返回内容。
content	String	配置文本内容。
mode	Integer	设备日志上报模式，0表示设备SDK不上报日志，1表示设备SDK上报日志。

## 设备上报日志内容

数据上行

- 请求Topic: /sys/\${productKey}/\${deviceName}/thing/log/post
- 响应Topic: /sys/\${productKey}/\${deviceName}/thing/log/post\_reply

Alink请求数据格式

```
{
  "id" : 123,
  "version": "1.0",
  "params" : [{
    "utcTime": "2020-03-06T15:15:27.464+0800",
    "logLevel": "ERROR",
    "module": "ModuleA",
    "code" : "",
    "traceContext": "123456",
    "logContent": "some log content"
  }],
  "method" : "thing.log.post"
}
```

表 11-38: 请求参数说明

参数	类型	说明
id	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
version	String	协议版本号，目前协议版本号唯一取值为1.0。

参数	类型	说明
<b>params</b>	List	请求业务参数。数组元素最多40个。
<b>utcTime</b>	String	日志的采集时间，为设备本地UTC时间，包含时区信息，以毫秒计，格式为“yyyy-MM-dd'T'HH:mm:ss.SSSZ”。可上报其它字符串格式，但不利于问题排查，不推荐使用。
<b>logLevel</b>	String	日志级别。可以使用默认日志级别，也可以自定义日志级别。默认日志级别从高到低为： <ul style="list-style-type: none"> <li>FATAL</li> <li>ERROR</li> <li>WARN</li> <li>INFO</li> <li>DEBUG</li> </ul>
<b>module</b>	String	模块名称。 <ul style="list-style-type: none"> <li>当设备端使用Android SDK时，模块名称为ALK-LK。</li> <li>当设备端使用C SDK时，需自定义模块名称。</li> </ul>
<b>code</b>	String	结果状态码。 <ul style="list-style-type: none"> <li>当设备端使用Android SDK时，请参见<a href="#">Android SDK错误码</a></li> <li>当设备端使用C SDK时，请参见<a href="#">C SDK状态码</a>。</li> </ul>
<b>traceContext</b>	String	可选参数，上下文跟踪内容，设备端使用Alink协议消息的 <b>id</b> ，APP端使用 <b>TraceId</b> （追踪ID）。
<b>logContent</b>	String	日志内容详情。
<b>method</b>	String	请求方法，取值thing.log.post。

## Alink响应数据格式

```
{
  "id" : 123,
  "code":200,
  "data" : {}
}
```

表 11-39: 响应参数说明

参数	类型	说明
<b>id</b>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。

参数	类型	说明
code	String	结果码。返回200表示成功，返回其他状态码，表示失败，具体请参见 <a href="#">设备端通用code</a> 。

## 11.14 设备网络状态

使用Wi-Fi接入网络的设备可以主动将网络状态信息，通过指定Topic上报至云端。本文介绍设备上报网络状态的Topic、数据格式和网络错误数据说明。



### 说明：

如果设备使用AliOS Things 3.0版及以上系统，会自动检测和上报网络状态数据。

### 设备主动上报网络状态

数据上行。

请求Topic： `/sys/{productKey}/{deviceName}/_thing/diag/post`

响应Topic： `/sys/{productKey}/{deviceName}/_thing/diag/post_reply`

Alink请求数据格式如下。

- 当前数据：设备采集后立即上报的数据。

以下两种情况下，设备立即上报网络状态数据。

- 当网络出现异常时，设备采集到异常指标，会立即上报数据到云端。
- 您设置了定时采集，设备会在指定时间采集数据，并立即上报数据。

假设当前时间是2019-08-22 08:10:29，设备检测到网络问题，立即上报数据。网络异常的数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "p": {
      "wifi": {
        "rssi": 75,
        "snr": 20,
        "per": 10,
        "err_stats": "10,02,01;10,05,01"
      },
      "_time": 1566432629000
    },
    "model": "quantity=single|format=simple|time=now"
  }
}
```

}

**说明:**

定时检测上报的网络正常数据中，**err\_stats**为空。

- 历史数据：非立即上报的数据。设备在日常诊断中，采集到网络正常的指标数据可以延迟上报。设备可以批量上报历史数据。

数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "p": [
      {
        "wifi": {
          "rssi": 75,
          "snr": 20,
          "per": 10,
          "err_stats": "10,02,01;10,05,01"
        },
        "_time": 1566432629000
      }
    ],
    "model": "format=simple|quantity=batch|time=history"
  }
}
```

**说明:**

如果网络无报错，**err\_stats**为空。

**表 11-40: 请求参数说明**

参数	类型	说明
<b>id</b>	String	消息ID号。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。
<b>version</b>	String	协议版本号，目前协议版本号唯一取值为1.0。
<b>params</b>	Object	请求入参。
<b>wifi</b>	Object	设备的连网方式为Wi-Fi，该参数值由网络状态的四个指标组成。
<b>rssi</b>	integer	无线信号接收强度。
<b>snr</b>	integer	无线信号信噪比。
<b>per</b>	integer	数据丢包率。


参数	类型	说明
<b>err_stats</b>	String	<p>错误信息。仅当设备检测到网络异常后，上报数据包含该参数。</p> <p>取值格式："type,code,count;type,code,count"，如"10,02,01;10,05,01"。</p> <p>参数说明：</p> <ul style="list-style-type: none"> <li><b>type</b>：错误类型</li> <li><b>code</b>：错误原因</li> <li><b>count</b>：错误数量</li> </ul> <p>具体错误请参见下表err_stats。</p>
<b>_time</b>	Long	<p>时间戳。</p> <div>  <b>说明：</b>            时间戳可以为空。为空时，控制台上<b>设备网络状态</b>不展示采集时间。         </div>
<b>model</b>	String	<p>消息体模式。包含：</p> <ul style="list-style-type: none"> <li><b>format</b>：数据格式。仅支持simple，表示数据为精简格式。</li> <li><b>quantity</b>：数量。               <ul style="list-style-type: none"> <li>single：表示上报单条数据。</li> <li>batch：表示上报多条数据，仅用于上报历史数据。</li> </ul> </li> <li><b>time</b>：时间。               <ul style="list-style-type: none"> <li>上报当前数据，取值now。</li> <li>上报历史数据，取值history。</li> </ul> </li> </ul>

表 11-41: err\_stats

错误类型	含义	错误原因
0x00	无线环境参数。	<ul style="list-style-type: none"> <li>信号强度（RSSI）：0x01</li> <li>信噪比（SNR）：0x02</li> <li>丢包率（drop ratio）：0x03</li> </ul>

错误类型	含义	错误原因
0x10	设备与云端建立连接失败。	<ul style="list-style-type: none"> <li>路由器连接失败（Wi-Fi fail）：0x01</li> <li>DHCP失败，获取IP地址失败（DHCP fail）：0x02</li> <li>DNS失败，解析云端的域名失败（DNS fail）：0x03</li> <li>TCP握手失败（TCP fail）：0x04</li> <li>TLS握手失败（TLS fail）：0x05</li> </ul>
0x20	设备与云端的网络异常	<ul style="list-style-type: none"> <li>云端主动断开与设备的连接（CLOUD_REJECT）：0x01</li> <li>设备数据上下行失败（RW_EXCEPTION）：0x02</li> <li>设备与云端的PING操作异常（PING_EXCEPTION）：0x03</li> </ul>
0x30	设备运行异常。	<ul style="list-style-type: none"> <li>看门狗复位重启（WD_RST）：0x01</li> <li>设备存储异常重启（PANIC_ERR）：0x02</li> <li>设备掉电上电重启（RE-POWER）：0x03</li> <li>设备运行异常重启（FATAL_ERR）：0x04</li> </ul>
0x40	设备内存动态监控。	<ul style="list-style-type: none"> <li>内存总量（type of total size）：0x01</li> <li>空闲内存总量（type of free size）：0x02</li> </ul>
0x50	BLE异常	-

Alink响应数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {}
}
```

表 11-42: 响应参数说明

参数	类型	说明
id	String	消息ID。String类型的数字，取值范围0~4294967295，且每个消息ID在当前设备中具有唯一性。



参数	类型	说明
code	Integer	返回结果，200代表成功。
version	String	协议版本号，目前协议版本号唯一取值为1.0。
data	Object	值为空。

## 11.15 设备端通用code

设备通用code信息，用于表达云端下行推送时设备侧业务处理的返回结果。

错误码	消息	描述
200	success	请求成功。
400	request error	内部服务错误，处理时发生内部错误
460	request parameter error	请求参数错误，设备入参校验失败
429	too many requests	请求过于频繁，设备端处理不过来时可以使用
100000-110000	自定义的错误信息	从100000到110000的错误码用于设备自定义错误信息，和云端错误信息加以区分

## 12 设备端错误码


本文介绍设备端可能出现的错误码及说明。

### 公共错误码

表 12-1: 通用公共错误码

错误码	原因	解决办法
400	处理请求时出错。	提交工单。
429	请求过于频繁，触发系统限流。	提交工单。
460	设备上报的数据为空，或参数格式错误、参数的数量超过限制等原因。	按照 <a href="#">Alink协议</a> 下具体文档中的数据格式，检查参数信息。
500	系统发生未知异常。	提交工单。
5005	查询产品信息失败。	在控制台，查询产品信息，核对ProductKey。
5244	查询LoRaWAN类型产品的元信息失败。	提交工单。
6100	查询设备信息时，未查询到指定设备信息。	在控制台设备管理中，核对设备信息。
6203	解析Topic时失败。	提交工单。
6250	查询产品信息失败。	在控制台，查询产品信息，核对ProductKey。
6204	设备已被禁用，不能对设备进行操作。	在控制台设备管理中，查看设备状态。
6450	自定义/透传格式数据经过脚本解析为Alink标准格式数据后，无 <b>method</b> 。	在物联网平台控制台 <b>日志服务</b> 中，或设备本地日志中，检查设备上报的数据中是否有 <b>method</b> 参数。
6760	系统异常。	提交工单。

表 12-2: 数据解析公共错误码

错误码	原因	排查
26001	执行数据解析时，获取的脚本内容为空。	<p>在控制台，产品的<b>数据解析</b>页签下，确认脚本已提交。</p> <div>  <b>说明：</b> 未提交的脚本不能被调用。         </div>

错误码	原因	排查
26002	脚本执行正常，但脚本内容有问题，如脚本中语法错误。	使用相同的数据测试脚本。查看具体的错误信息，修改脚本。建议在本地详细的自验后，再提交到物联网平台。
26006	脚本执行正常，脚本内容有误。脚本中，要求有 <code>protocolToRawData</code> 和 <code>rawDataToProtocol</code> 这两个服务。如果缺失，会报错。	在控制台，产品的 <a href="#">数据解析</a> 页签下，查询脚本内容中 <code>protocolToRawData</code> 和 <code>rawDataToProtocol</code> 服务是否存在。
26007	脚本执行正常，但返回结果不符合格式要求。脚本中，要求有 <code>protocolToRawData</code> 和 <code>rawDataToProtocol</code> 这两个服务。 <code>protocolToRawData</code> 返回 <code>byte[]</code> 数组， <code>rawDataToProtocol</code> 要求返回JSON对象。如果返回结果不符合这两种格式，会报这个错。	在控制台或在本地测试脚本，并查看返回结果的格式是否符合要求。
26010	请求过于频繁，导致被限流。	提交工单。

表 12-3: TSL公共错误码

错误码	原因	排查
5159	TSL校验时，查询属性定义失败。	提交工单。
5160	TSL校验时，查询事件定义失败。	提交工单。
5161	TSL校验时，查询服务定义失败。	提交工单。
6207	设备上报的Alink数据格式，或者调用脚本解析后返回的数据格式，不是JSON格式。	请参见 <a href="#">设备属性</a> 、 <a href="#">事件</a> 、 <a href="#">服务</a> ，查看对应数据格式，并按格式要求上报数据。
6300	<code>method</code> 不存在。TSL校验时，设备上报的Alink（标准）格式数据，或自定义（透传）格式数据经过脚本转换后，没有Alink协议要求的 <code>method</code> 参数。	在控制台 <a href="#">日志服务</a> ，或者设备的本地日志中，查看上报数据中是否有 <code>method</code> 参数。
6301	TSL校验时，发现定义的数据为array类型，但上报的数据不是array类型。	在控制台，产品的 <a href="#">功能定义</a> 页签下，查看产品的TSL中对应数据格式，并按格式要求上报数据。
6302	TSL校验服务的入参时，发现缺少必需的参数。	在控制台，查看设备所属产品的功能定义，查询对应服务的入参，核对传入的参数。

错误码	原因	排查
6306	TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的参数取值范围不符合功能定义时设置的参数范围。</li> </ul>	在控制台，查看设备所属产品的功能定义，核对传入的参数类型和取值范围。
6307	传入的参数不符合TSL中32位浮点数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的参数取值范围不符合功能定义时设置的参数范围。</li> </ul>	
6308	传入的参数不符合TSL中布尔类型数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的参数取值范围不符合功能定义时设置的参数范围。</li> </ul>	
6310	传入的参数不符合TSL中字符类型数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的字符类型的参数长度超过限制。</li> </ul>	
6322	传入的参数不符合TSL中64位浮点数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的参数取值范围不符合功能定义时设置的参数范围。</li> </ul>	
6304	TSL校验时，发现传入的参数在结构体中不存在。	在控制台，查看设备所属产品的功能定义，核对传入的参数类型。
6309	传入的参数不符合TSL中枚举类型数据的规范。	

错误码	原因	排查
6311	传入的参数不符合TSL中日期类型数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或传入的字符类型不是UTC时间戳的字符格式。</li> </ul>	
6312	传入的参数不符合TSL中结构体类型数据的规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的参数类型和TSL中定义的类型不一致。</li> <li>或结构体类型中参数的个数和TSL中定义不一致。</li> </ul>	
6320	查询设备的TSL时，没有查询到设备的属性信息。	在控制台，查看设备所属产品的功能定义中是否存在该属性。若不存在，需增加属性定义。
6321	解析TSL时，发现属性、事件或者服务的标识符为空。	提交工单。
6317	TSL校验时，发现TSL中缺少关键信息，如 <b>type</b> ， <b>specs</b> 为空。	提交工单。
6324	传入的数组类型的参数不符合规范。TSL校验时，发现： <ul style="list-style-type: none"> <li>传入的数组类型的参数不符合TSL定义。</li> <li>或数组中参数个数超过了TSL中定义的最大个数。</li> </ul>	<ul style="list-style-type: none"> <li>在控制台产品详情中，查看设备所属产品的功能定义，检查对应数组的定义。</li> <li>查看设备上报的日志，检查设备上报的数据中数组内元素的个数。</li> </ul>
6325	传入的数组类型参数中有不支持的元素类型。目前，数组中元素的类型只支持整形、32位浮点类型、64位浮点类型、字符串类型、结构体类型。	检查传入的数组元素类型是否是支持的类型。
6326	TSL校验时，检查上报的数据中time字段格式时报错。	请参见 <a href="#">设备属性</a> 、 <a href="#">事件</a> 、 <a href="#">服务</a> ，查看对应数据格式，并按格式要求上报数据。
6328	TSL校验时，发现传入的参数不是数组类型。	在控制台，查看设备所属产品的功能定义，核对传入的对应参数是否是数组类型。
系统异常错误码		
6318	TSL解析时，系统异常。	提交工单。

错误码	原因	排查
6313		
6329		
6323		
6316		
6314		
6301		

### 设备身份注册相关错误码

- 直连设备身份注册

请求Topic: `/sys/{productKey}/{deviceName}/thing/sub/register`

错误码: 460、5005、5244、500、6288、6100、6619、6292、6203

以下为设备身份注册的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6288	设备动态注册开关未打开。	在控制台，设备所属的产品详情页，开启设备动态注册。
6619	设备已绑定到其它网关下。	在控制台，该子设备的详情页，查看该设备是否已绑定网关。

- 直连设备一型一密动态注册

错误码: 460、6250、6288、6600、6289、500、6292

以下为直连设备一型一密动态注册的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6288	设备动态注册开关未打开。	在控制台，设备所属的产品详情页，开启设备动态注册。
6292	校验签名时，发现传入的签名方法不支持。	请使用 <a href="#">设备身份注册</a> 中 <b>signMethod</b> 支持的签名方法。
6600	签名校验失败。	请按照 <a href="#">设备身份注册</a> 中的签名方法计算签名，并校验签名。
6289	一型一密动态注册直连设备时，发现设备已激活。	在控制台设备管理中，查看该设备的状态。

## 设备拓扑关系相关错误码

- 添加设备拓扑关系

请求Topic: `/sys/{productKey}/{deviceName}/thing/topo/add`

错误码: 460、429、6402、6100、401、6204、6400、6203

以下为添加设备拓扑关系的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
401	添加拓扑关系时，校验签名信息失败。	请按照 <a href="#">管理拓扑关系</a> 中的签名方法计算签名，并校验。
6402	网关与子设备是同一个设备。添加拓扑关系时，不能把当前网关作为子设备添加到当前网关下。	检查添加的子设备信息，是否有子设备信息和网关信息一致。
6400	为网关添加的子设备数量超过限制。	请参见 <a href="#">#unique_95</a> 查看相关限制；并在控制台，该网关设备的 <b>子设备管理</b> 页签下，查看已有子设备数量。

- 删除拓扑关系

请求Topic: `/sys/{productKey}/{deviceName}/thing/topo/delete`

错误码: 460、429、6100、6401、6203

以下为删除设备拓扑关系的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6401	检查拓扑关系时，拓扑关系不存在。	在控制台 <b>设备管理</b> ，网关设备的 <b>设备详情</b> 页 <b>子设备管理</b> 页签中，查看子设备信息。

- 获取拓扑关系

请求Topic: `/sys/{productKey}/{deviceName}/thing/topo/get`

错误码: 460、429、500、6203。错误码说明，请参见本文公共错误码章节。

- 网关上报发现子设备

请求Topic: `/sys/{productKey}/{deviceName}/thing/list/found`

错误码: 460、500、6250、6280、6203

以下为特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6280	网关上报的子设备名称不符合规范。设备名称字符仅支持中文汉字、英文字母、数字和下划线（_），长度范围4~32个字符，一个中文汉字算两个字符。	检查上报的设备名称是否符合规范。

### 子设备上下线相关错误码

- 子设备上线

请求Topic: `/ext/session/${productKey}/${deviceName}/combine/login`

错误码: 460、429、6100、6204、6287、6401、500

- 子设备主动下线异常

接收消息的网关Topic: `/ext/session/{productKey}/{deviceName}/combine/logout_reply`

错误码: 460、520、500

- 子设备被踢下线

接收消息的网关Topic: `/ext/error/{productKey}/{deviceName}`

错误码: 427、521、522、6401

- 子设备发送消息失败

接收消息的网关Topic: `/ext/error/{productKey}/{deviceName}`

错误码: 520

以下为子设备上、下线相关的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
427	设备重复登录。有使用相同设备证书信息的设备连接物联网平台，导致当前连接被断开。	在控制台设备的 <b>设备详情页</b> ，查看设备的上线时间，以此判断是否有其他设备使用相同的设备证书接入物联网平台。
428	单个网关下子设备数目超过最大值。目前一个网关下，最多可有1500个子设备。	请检查网关设备下的子设备数量。



错误码	原因	排查
521	设备已被删除。	请在控制台 <b>设备管理</b> 页搜索设备，确认设备是否已被删除。
522	设备已被禁用。	请在控制台 <b>设备管理</b> 页查看设备状态。
520	子设备会话错误。 <ul style="list-style-type: none"> <li>子设备会话不存在，可能子设备没有上线，也可能已经被下线。</li> <li>子设备会话在线，但是并不是通过当前网关会话上线的。</li> </ul>	
6287	按照产品或者设备的密钥校验签名失败。	请参见 <b>子设备上下线</b> 中的签名方法计算签名，并校验。

### 设备属性、事件、服务相关错误码

#### • 设备上报属性

- 请求Topic（透传数据格式）：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 请求Topic（Alink数据格式）：`/sys/{productKey}/{deviceName}/thing/event/property/post`

错误码：460、500、6250、6203、6207、6313、6300、6320、6321、6326、6301、6302、6317、6323、6316、6306、6307、6322、6308、6309、6310、6311、6312、6324、6328、6325、6200、6201、26001、26002、26006、26007

以下为上报属性的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6106	上报的属性数据过多。设备一次上报的有效属性个数不能超过200个。	在控制台， <b>监控运维 &gt; 日志服务</b> 中，或设备本地的日志中，检查上报的属性个数。

#### • 设备上报事件

- 请求Topic（透传格式数据）：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 请求Topic（Alink格式数据）：`/sys/{productKey}/{deviceName}/thing/event/{tsl.identifier}/post`

错误码：460、500、6250、6203、6207、6313、6300、6320、6321、6326、6301、6302、6317、6323、6316、6306、6307、6322、6308、6309、6310、6311、6312、6324、6328、6325、6200、6201、26001、26002、26006、26007

错误码说明，请参见本文公共错误码章节。

- 网关批量上报子设备数据

- 请求Topic（透传格式数据）：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 请求Topic（Alink格式数据）：`/sys/{productKey}/{deviceName}/thing/event/property/`  
`pack/post`

错误码：460、6401、6106、6357、6356、6100、6207、6313、6300、6320、6321、6326、6301、6302、6317、6323、6316、6306、6307、6322、6308、6309、6310、6311、6312、6324、6328、6325、6200、6201、26001、26002、26006、26007

以下为网关批量上报子设备数据失败的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6401	拓扑关系不存在。	在控制台，网关设备的 <b>子设备管理</b> 页签下，确认其子设备信息。
6106	上报的属性数据过多。设备一次上报的有效属性个数不能超过200个。	在控制台， <b>监控运维 &gt; 日志服务</b> 中，或设备本地的日志中，检查上报的属性个数。
6357	子设备数据过多。网关代替子设备上报数据，一次上报最多可包含20个子设备的数据。	查看设备本地日志中的上报数据。
6356	上报的事件数据过多。网关代替子设备上报数据，一次上报的事件个数不可超过200。	查看设备本地的日志中的上报数据。

#### 设备期望属性值相关错误码

- 设备获取期望属性值

请求Topic：`/sys/{productKey}/{deviceName}/thing/property/desired/get`

错误码：460、6104、6661、500

以下为设备期望属性值操作失败的特有错误码说明，其他错误码说明请参见以上公共错误码章节。

错误码	原因	排查
6104	请求中包含的属性个数过多。一次请求可包含的属性个数不能超过200个。	在控制台， <b>监控运维 &gt; 日志服务</b> 中，或者设备本地日志中，查看上报数据中的属性个数。
6661	查询期望属性失败。系统异常。	提交工单排查。

- 设备清空期望属性值

请求Topic: `/sys/{productKey}/{deviceName}/thing/property/desired/delete`

错误码: 460、6104、6661、500、6207、6313、6300、6320、6321、6326、6301、6302、6317、6323、6316、6306、6307、6322、6308、6309、6310、6311、6312、6324、6328、6325

#### 设备标签相关错误码

- 设备上报标签信息

请求Topic: `/sys/{productKey}/{deviceName}/thing/deviceinfo/update`

错误码: 460、6100

- 设备删除标签信息

请求Topic: `/sys/{productKey}/{deviceName}/thing/deviceinfo/delete`

错误码: 460、500

#### 获取TSL模板失败错误码

请求Topic: `/sys/{productKey}/{deviceName}/thing/dsltemplate/get`

错误码: 460、5159、5160、5161

#### 设备请求固件信息失败

请求Topic: `/ota/device/request/${YourProductKey}/${YourDeviceName}`



##### 说明:

- 固件升级 `AbstractAlinkJsonMessageConsumer` 类中, 只有 `DeviceOtaUpgradeReqConsumer` 会返回错误码; 其它接口不返回错误码信息, 返回数据为 `null`。
- 设备请求固件信息的Topic与返回数据Topic相同。

错误码: 429、9112、500

以下为设备请求固件信息的特有错误码, 其他错误码, 请参见以上公共错误码章节。

错误码	原因	排查
9112	未查询到指定的设备信息。	在控制台设备管理中, 确认设备信息是否正确。

**设备请求配置信息失败**

请求Topic: /sys/{productKey}/{deviceName}/thing/config/get

错误码: 460、500、6713、6710

以下为设备请求配置信息的特有错误码，其他错误码，请参见以上公共错误码章节。

错误码	原因	排查
6713	远程配置服务不可用。该产品的远程配置开关未打开。	在控制台， <b>监控运维 &gt; 远程配置</b> 中，打开该产品的远程配置开关。
6710	未查询到远程配置信息。	在控制台， <b>监控运维 &gt; 远程配置</b> 中，查看是否该产品编辑了远程配置文件。