ALIBABA CLOUD

阿里云

物联网平台

设备管理

文档版本：20220519

阿里云

法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	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。	危险 重置操作将丢失用户配置数据。
警告	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。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	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

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**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 产品**。
4. 在**产品**页，单击**创建产品**，填入产品信息，创建产品。

   具体操作说明，请参见创建产品。
5. 在左侧导航栏，选择**设备**。
6. 在**设备**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，请参见批量创建设备或单个创建设备。

1.2. 设备上线和下线

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Link 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Link SDK进行开发，请参见设备接入。

   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：
   - 一机一密：预先为每个设备烧录其唯一的设备证书（ProductKey、DeviceName和DeviceSecret）。
   - 一型一密：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。

  场景如下：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。

MQTT保活

MQTT连接心跳时间为30秒至1,200秒。心跳时间不在此区间内，服务器将会拒绝连接。 建议取值300秒以上。

从物联网平台发送CONNACK响应CONNECT消息时，开始心跳计时。收到PUBLISH、SUBSCRIBE、PING或 PUBACK消息时，会重置计时器。超过指定1.5倍心跳时间未收到消息（指定心跳时间乘以1.5），服务器将自动断开连接。

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**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 设备**。
4. 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. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 设备**。
4. 在**设备列表**中，找到要启用的设备，打开设备对应的**启用状态**开关。

1.4. 删除设备

您可以将设备从物联网平台中删除，不限设备当前状态。设备从物联网平台删除后，设备证书将失效，除已产生的云端运行日志外，与该设备关联的其他数据也一并删除。设备的云端运行日志仍可查询，但您将无法通过物联网平台执行与该设备关联的其他任何操作。

操作步骤

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 设备**。
4. 在**设备列表**中，找到要删除的设备。
5. 单击对应的**删除**按钮，并确认删除。

执行结果

设备删除后，该设备证书失效，且不能恢复。设备的云端运行日志仍可查询，但无法通过物联网平台执行与该设备关联的其他操作。

2.数字孪生

2.1. 数字孪生概述

物联网平台提供数字孪生服务，通过数字化形式动态呈现物理世界的业务模型。本文介绍数字孪生的相关概念和使用。

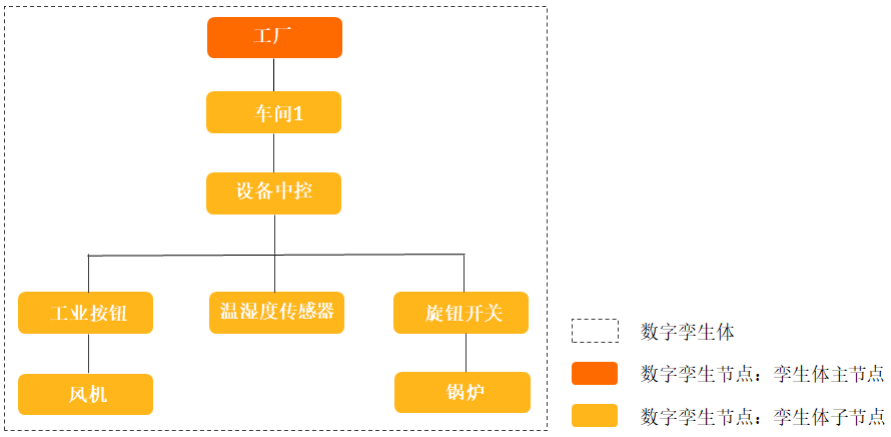
什么是数字孪生

> **说明** 目前仅华东2（上海）地域的企业版实例下，支持使用数字孪生功能。

数字孪生是物理世界的数字化呈现，可通过构建孪生体来描述设备、流程、系统、场景等业务模型，对物理世界实体信息进行实时采集、运算分析、监控统计等，助您更精准地掌握业务模型动态变化，进而实现对实际生产过程的提效和降本目的。

例如为工厂创建一个数字孪生体，工厂中每个设备作为孪生节点，使用孪生节点的物模型映射工厂设备的运行动态，通过设备运行数据，对设备进行故障预判和及时修复。

下图为一个工厂数字孪生体示例，描述一个车间中的IoT设备系统。您可在数字孪生节点配置物模型属性和孪生规则，通过数据映射能力，实现设备系统管理。



名词解释

名词	说明
数字孪生体	简称“孪生体”，由多个和多层级的数字孪生节点组成，用于构建物理世界的业务场景、业务流程和业务模型。
数字孪生节点	简称“孪生节点”，一个孪生节点可对应一个物理设备，非实体，无需连接物联网平台。删除孪生节点对物理设备无影响。 孪生节点提供物模型定义和孪生规则配置功能。
孪生体模板	可选中孪生体中一个或多个孪生节点生成孪生体模板。孪生体模板中新孪生节点，会拷贝源孪生节点间的关系、物模型定义和孪生规则，且新孪生节点信息支持单独修改。 您可引用或拷贝孪生体模板到数字孪生体中使用，快速构建完整的业务流程和模型。
物模型定义	在孪生体和孪生体模板中，支持为孪生节点配置物模型属性。属性数据可来自物理设备的数据映射，也可由孪生规则运算得出。 物模型定义的更多信息，请参见物模型。
孪生规则	用于动态计算孪生节点的属性值。 物理设备属性值变更，映射到孪生节点的属性上，会触发孪生节点配置的规则运算，得出通过规则定义的输出属性值。
数据映射	将物理设备上报的Topic数据，处理并传递给数字孪生节点的物模型属性。 支持的消息Topic有：设备自定义Topic和物模型属性上报Topic。

使用限制

限制项	描述	限制
数字孪生体	一个实例最多包含数字孪生体总数。	1,000
数字孪生节点	一个数字孪生体最多支持的层级总数（包含孪生体主节点）。	10
	一个数字孪生体最多包含孪生节点总数。	1,000
	一个数字孪生体模板最多包含孪生节点个数。	10
数据映射	一个数字孪生体最多包含数据映射总数。	1,000
	一个数据映射最多包含输出参数总数。	3,000
物模型属性	一个孪生节点最多包含的属性总数。	300
	一个属性最多作为多少孪生规则的输入参数。	10

限制项	描述	限制
孪生规则	一个孪生节点最多包含孪生规则总数。	300
	一个孪生规则最多包含输入参数总数。	5

使用流程

已在物联网平台接入物理设备，实现消息通信。具体内容，请参见通过设备Topic通信。

1. 添加孪生体。
2. 配置孪生节点。
   i. 配置功能属性。
   ii. （可选）配置孪生规则。
3. （可选）设置孪生体模板。
4. 添加数字映射。
5. （可选）设置数据流转规则或添加数据流转解析器。
   您可通过云产品流转功能将**孪生节点属性变更**数据，流转到其他云产品或转发到AMQP服务端订阅消费组。
6. 查看孪生体运行日志。

使用数字孪生管理园区环境

2.2. 添加孪生体

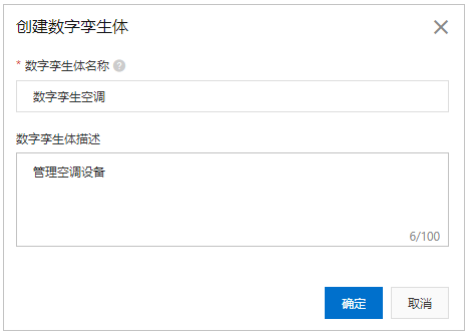
数字孪生体是由多个孪生节点，构建的一个完整的业务模型体系。本文介绍创建数字孪生体，并添加数字孪生节点的具体操作。

创建数字孪生体

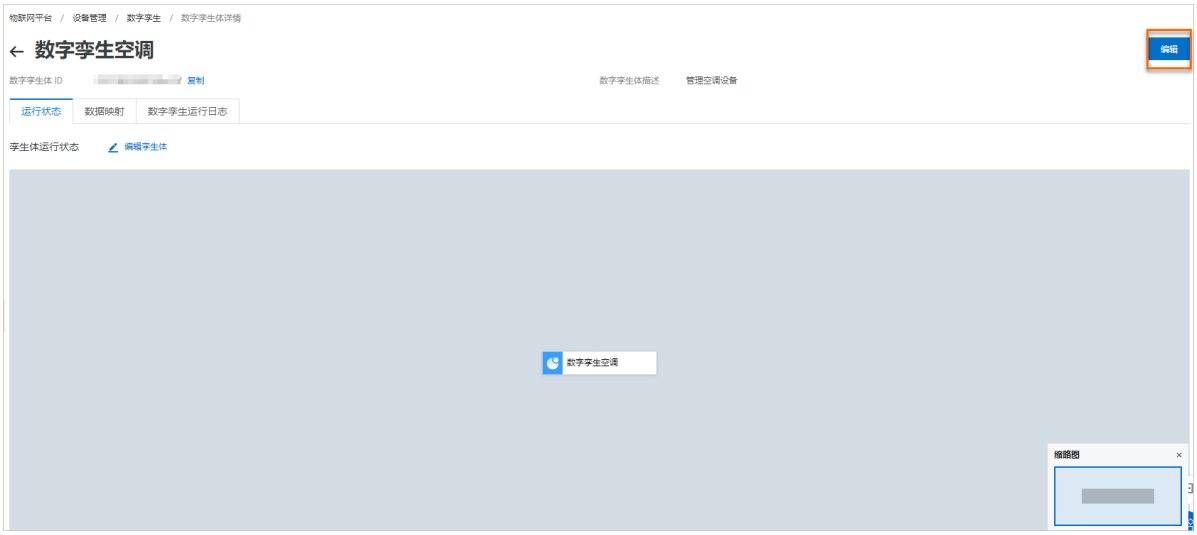
1. 登录物联网平台控制台。
2. 在**实例概览**页面左上方，选择**华东2（上海）**地域，找到对应的企业版实例，单击实例进入**实例详情**页面。

3. 在左侧导航栏，选择**设备管理 > 数字孪生**，单击**创建数字孪生体**。

4. 在**创建数字孪生体**对话框，输入数字孪生体名称和描述，单击**确定**。
   数字孪生体名称支持中文、英文字母、日文、数字、短划线（-）、下划线（_）、正斜线（/）和半角句号（.），必须以中文、英文字母或数字开头，长度不超过30个字符。



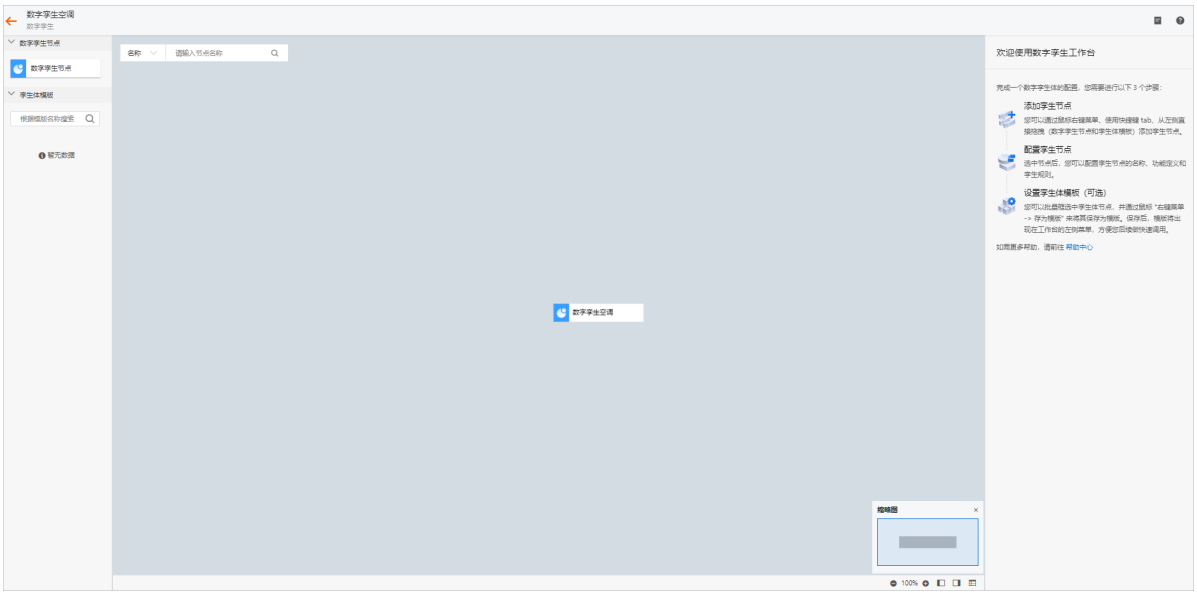
   创建数字孪生体成功后，根据页面提示，可直接进入**数字孪生体详情**页面。您可单击右上角**编辑**，修改数字孪生体名称和描述。



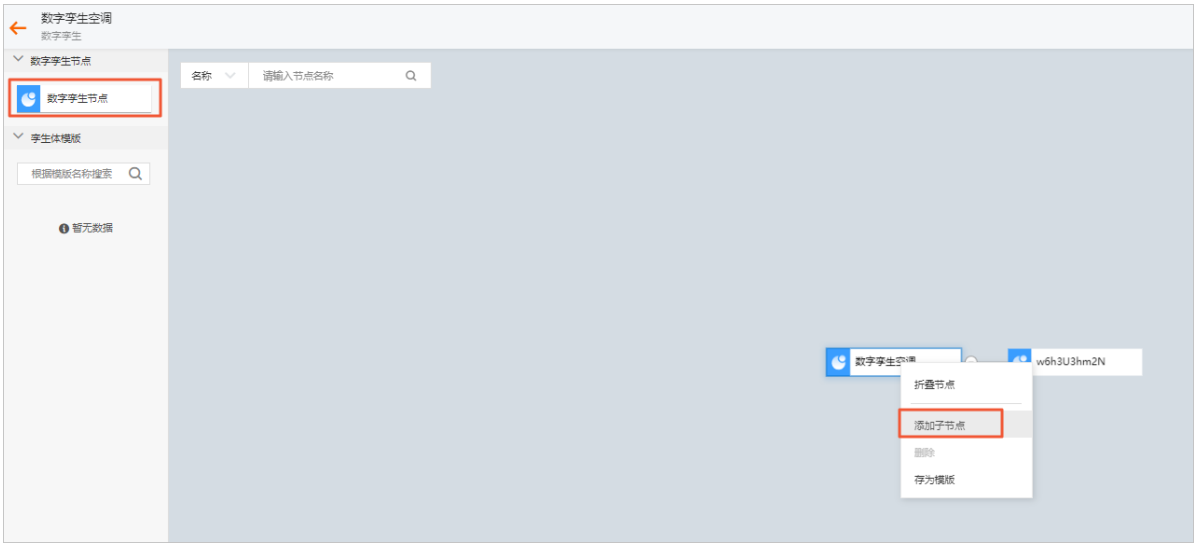
添加数字孪生节点

您可在**设备管理 > 数字孪生**页面的数字孪生列表，单击**操作**列的**查看**，进入对应的**数字孪生体详情**页面。

1. 在**数字孪生体详情**页面的**运行状态**页签，单击**编辑孪生体**，进入数字孪生工作台。
   工作台中默认添加了一个孪生主节点，不可删除。节点名称与孪生体名称相同，不可修改。

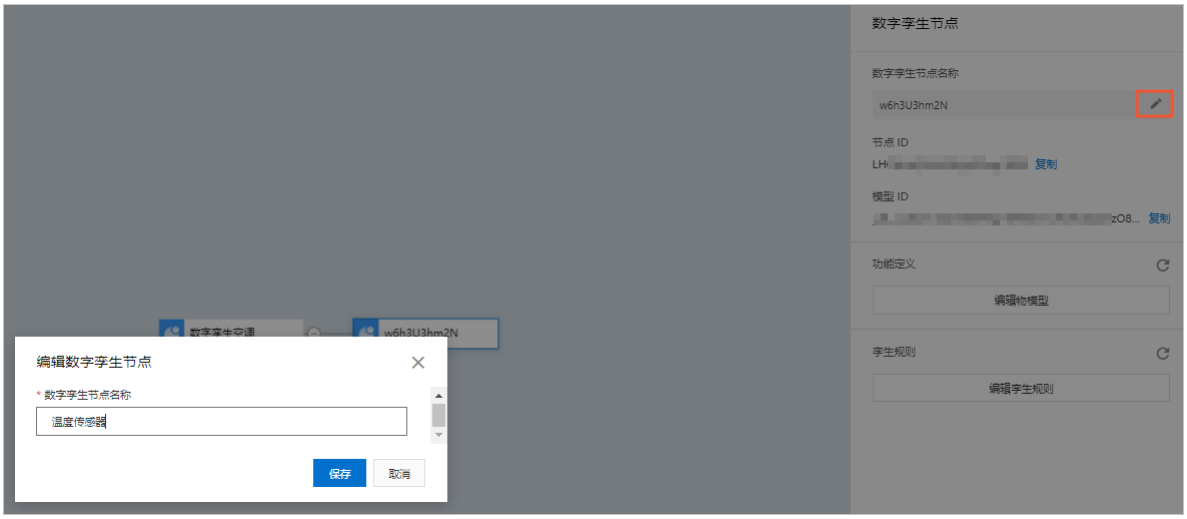


2. 将左侧的**数字孪生节点**拖拽到中间画布的指定节点上，为该指定节点添加子节点。
   您也可在画布中，右键单击指定节点，然后单击**添加子节点**，为该节点添加子节点，或单击**删除**，删除多余子节点。



3. （可选）单击画布中的孪生节点，在右侧配置页面，查看节点名称、节点ID和模型ID。

   您可双击画布中的子节点，或单击子节点名称的编辑图标，修改节点名称。您可在画布左上角输入节点名称，搜索指定孪生节点。



后续步骤

配置功能属性：为孪生节点添加物模型属性。

2.3. 配置孪生节点

2.3.1. 配置功能属性

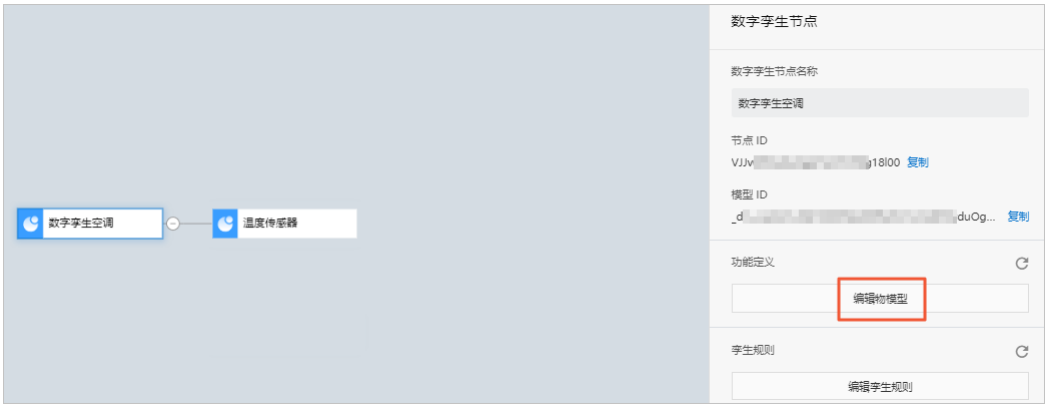
您需为已添加的数字孪生节点配置物模型属性，实现物理设备上报数据的数据映射。本文介绍孪生节点功能属性定义的具体操作。

前提条件

已创建孪生体，并完成添加孪生节点。具体操作，请参见添加孪生体。

编辑物模型

1. 在数字孪生工作台中，单击孪生节点，例如单击**温度传感器**，然后在**数字孪生节点**面板，单击**编辑物模型**。



2. 在弹出的**孪生节点物模型**面板，单击**添加功能**，配置以下属性参数，单击**确认**。

   您可重复该操作，添加多个功能属性，然后单击**物模型TSL**，查看物模型文件。

参数	说明
功能名称	属性的名称，例如：用电量。同一产品下功能名称不能重复。 支持中文、英文字母、日文、数字、短划线（-）、下划线（_）、正斜线（/）和半角句号（.），必须以中文、英文字母或数字开头，长度不超过30个字符。 输入功能名称时，将从标准功能库中筛选匹配的标准功能供您选择，您可以参考标准功能进行配置。
标识符	属性唯一标识符，在产品中具有唯一性。即Alink JSON格式中的identifier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。支持英文、数字和下划线（_），不超过50个字符，例如：PowerConsumption。 **说明** 不能用以下系统保留参数作为标识符：set、get、post、time、value。
数据类型	仅支持选择以下类型： ○ *int32*：32位整型。需定义取值范围、步长和单位符号。 ○ *float*：单精度浮点型。需定义取值范围、步长和单位符号。 ○ *double*：双精度浮点型。需定义取值范围、步长和单位符号。 ○ *enum*：枚举型。定义枚举项的参数值和参数描述，例如：1表示加热模式、2表示制冷模式。 ○ *bool*：布尔型。采用0或1来定义布尔值，例如：0表示关、1表示开。 ○ *text*：字符串。需定义字符串的数据长度，最长支持10240字节。 ○ *date*：时间戳。格式为String类型的UTC时间戳，单位：毫秒。
取值范围	数据类型为int32、float、double时，可设置属性值的取值范围。
步长	属性值变化的最小粒度。数据类型为int32、float、double时，可根据您的业务需要设置步长。 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℃等。
单位	单位可选择为无，或根据实际情况选择。
读写类型	此处设置为**读写**。 ○ *读写*：请求读写的方法支持GET（获取）和SET（设置）。 ○ *只读*：请求只读的方法仅支持GET（获取）。
描述	输入文字，对该功能进行说明或备注。长度限制为100个字符。

例如为**温度传感器**子节点，添加属性**当前温度**。

3. 添加功能属性完成后，单击**孪生节点物模型**面板左下角的**关闭**。
4. 重复以上步骤，为所有孪生节点配置属性功能。
   例如：为主节点添加功能属性**目标温度**（TargetTemperature）。

后续步骤

- （可选）配置孪生规则：为物模型属性添加孪生规则。
- （可选）设置孪生体模板：将多个孪生节点组成数字孪生体模板。
- 添加数字映射：为孪生节点的物模型属性，配置数据映射。

2.3.2. 配置孪生规则

孪生规则，是用来动态计算孪生节点的物模型属性。本文介绍配置孪生规则的具体操作。

前提条件

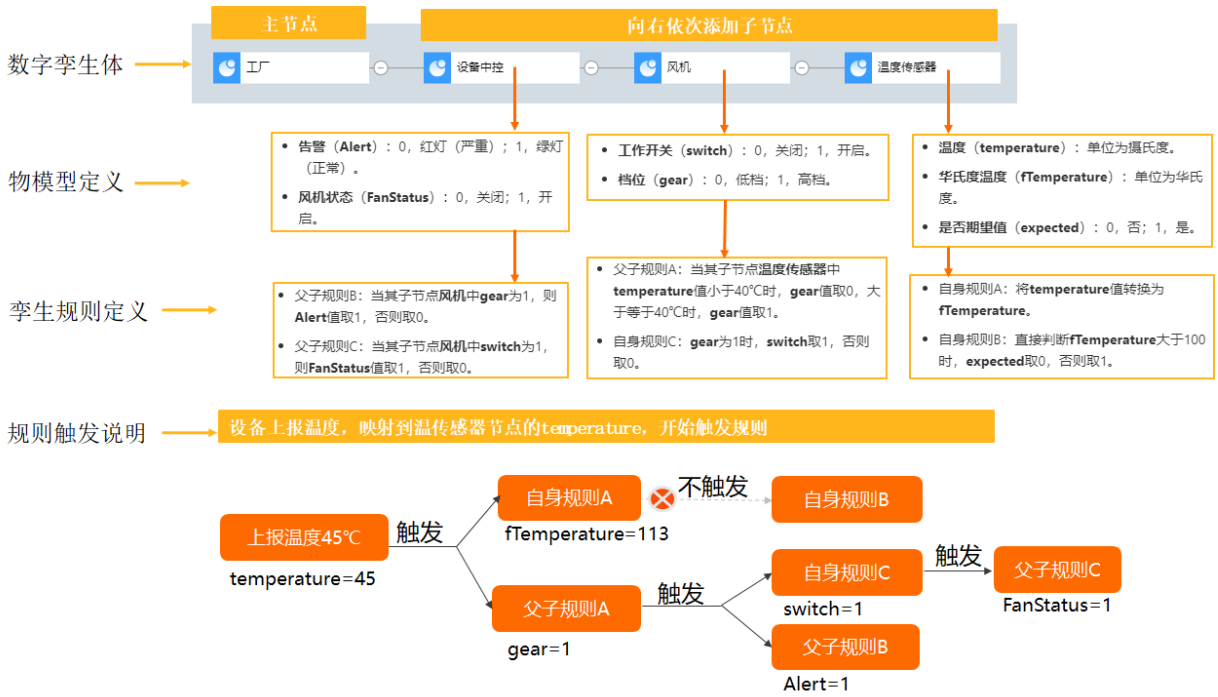
已为孪生节点配置功能属性。具体操作，请参见配置功能属性。

背景信息

孪生规则分为自身规则和父子规则，由输入参数、表达式和输出属性组成。输入参数对应孪生节点的物模型属性，属性运行时数据变更，会触发孪生规则更新输出属性。若输出属性是其他孪生规则的输入参数，会递归触发孪生规则执行，实现孪生节点间的运行时数据联动。

> 注意　自身规则的输出属性不可以触发另一个自身规则。

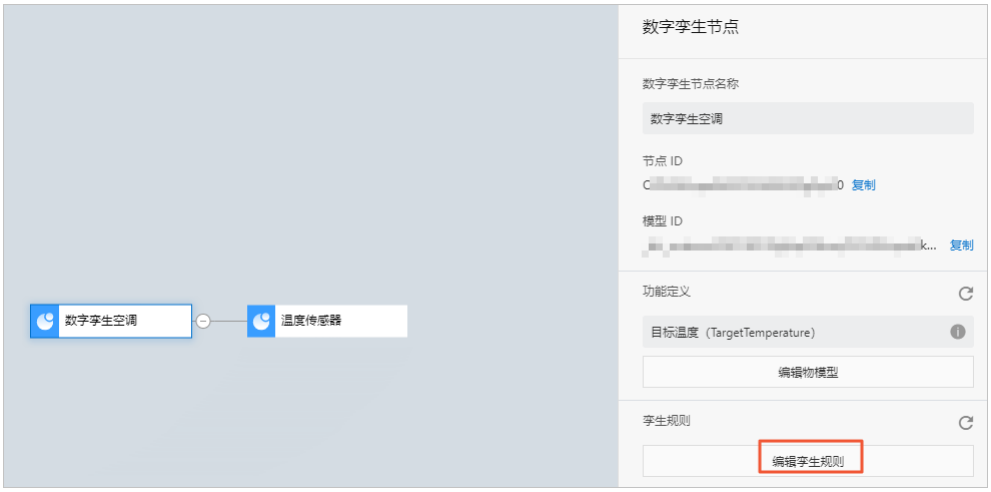
例如：在一个工厂数字孪生体中，依次逐级添加孪生节点**设备中控**、**风机**和**温度传感器**，孪生规则配置及触发流程如下图。



规则使用限制的详细内容，请参见使用限制中的孪生规则。

编辑孪生规则

1. 在数字孪生工作台中，单击孪生节点，然后在**数字孪生节点**面板，单击**编辑孪生规则**。



2. 在**孪生规则**面板，单个或批量添加孪生规则。

参照以下步骤，单个配置孪生规则。

i. 单击**添加孪生规则**，在弹出的对话框中，配置以下参数。

参数	说明
规则名称	自定义名称。支持中文、英文字母、日文、数字、短划线（-）、下划线（_）、正斜线（/）和半角句号（.），必须以中文、英文字母或数字开头，长度不超过30个字符。
属性引用类型	可选： ■ **父子规则**：**输入参数**的**属性来源**为当前孪生节点的子节点。 ■ **自身规则**：**输入参数**的**属性来源**为当前孪生节点。
输入参数	自定义**参数名称**，然后选择对应的**属性来源**（孪生节点）及其**属性**。 参数名称支持数字和英文字母，必须以英文字母开头，长度不超过20个字符。 单击**添加参数**，可添加多个输入参数，最多不超过5个。
表达式	在输入框内编辑运算表达式。支持的运算符和函数，请参见规则表达式。 表达式中变量的字段名称必须使用**输入参数**中的**参数名称**字段。 表达式中包含多个入参时，任何一个属性值变更，都会触发规则。执行规则时，其他入参的值，取对应节点属性的最新快照值。 例如：定义父子规则 `room1_temp+room2_temp=floor_temp` ，同时上报 `room1_temp` 和 `room2_temp` 时，会触发两次表达式的执行： ■ `room1_temp`**变更值**+`room2_temp`**的快照** 。 ■ `room1_temp`**的快照**+`room2_temp`**变更值** 。 **注意** 入参快照值为空时，表达式会计算异常。您可使用条件函数，设置入参为空时，返回默认值。函数说明，请参见Value条件函数。
输出属性	选择当前节点的功能属性，输出表达式计算结果。 若**属性引用类型**为**自身规则**，则**输入参数**中已添加属性，不再可选。 每个功能属性，仅作为一个孪生规则的输出属性。 **注意** 表达式输出的数据类型，都会转为输出属性的数据类型。 例如，表达式输出数据为Double类型，输出属性为整型，会截取Double数据的整数部分。

ii. 单击**确认**。

参照以下步骤，批量配置孪生规则。

i. 单击**批量添加**，在**批量添加孪生规则**对话框，单击**下载.xls模板**，获取以当前节点名称命名的 `.xls` 规则文件。

注意 对于每个孪生节点，都必须下载对应的规则文件，进行配置。

ii. 在规则文件中编辑规则内容，并保存。

如下图所示，规则文件中会根据对应孪生节点的配置，提供参数配置选项。参数配置说明，请参见上文*单个配置孪生规则*。

	A	B	C	D	E	F	G
1	Name （规则名称）	Expression （规则表达式）	ExpressionParamsType （规则类型）	Input.NodeName （入参节点名称）	Input.Identifier （入参功能定义标识符）	Input.Variable （入参名称）	Output.Identifier （出参功能定义标识符）
2						temp	
3							
4	test	1.8*temp+32					
5							
6							

iii. 在**批量添加孪生规则**对话框，单击**选择文件**，选择已保存的 `.xls` 文件，单击**打开**。

如果页面提示解析失败，请单击**下载不合法列表**，然后根据不合法列表文件中的错误提示信息，修正规则文件后，重新上传。

iv. 上传成功后，单击**确定**。

后续步骤

- （可选）设置孪生体模板：将多个孪生节点组成数字孪生体模板。
- 添加数字映射：为孪生节点的物模型属性，配置数据映射。

2.4. 设置孪生体模板

为方便快速构建孪生体业务模型，可选中孪生体中的多个孪生节点，将孪生节点的关系、物模型定义和孪生规则拷贝生成为孪生体模板。您可在任意孪生体中引用或拷贝孪生体模板，快速构建业务流程。本文介绍孪生体模板生成和使用的具体操作。

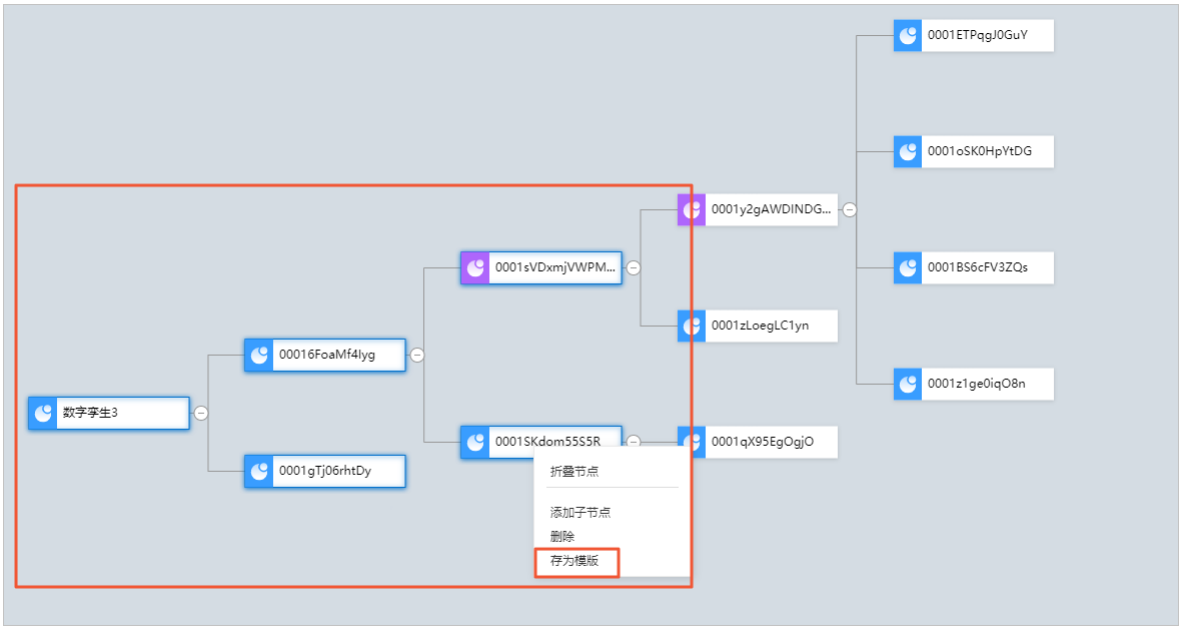
前提条件

已配置孪生节点。具体操作，请参见添加孪生体。

操作步骤

1. 在数字孪生工作台页面，按住键盘的Shift键，拖动鼠标指针，框选一个或多个孪生节点，然后右键单击选中的节点，在弹出的菜单栏中单击**存为模板**。

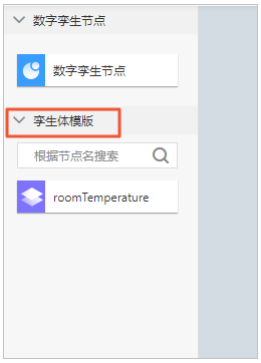
注意 最多可框选10个孪生节点。



2. 在弹出的对话框中，输入模板名称，单击**确定**。

   模板名称支持中文、英文字母、日文、数字、短划线（-）、下划线（_）、正斜线（/）和半角句号（.），必须以中文、英文字母或数字开头，长度不超过30个字符。

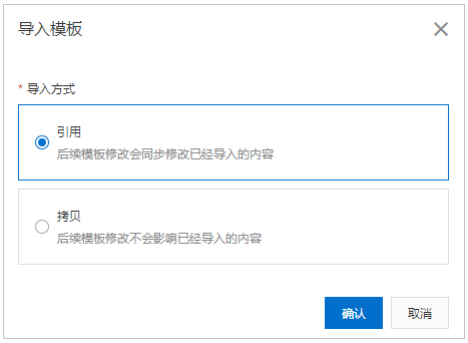
   模板生成完成后，在工作台左侧的**孪生体模板**列表下显示。该列表显示当前实例下，添加的所有孪生体模板。



3. 在任意数字孪生体的工作台中，使用拖拽方式，引用或拷贝孪生体模板。

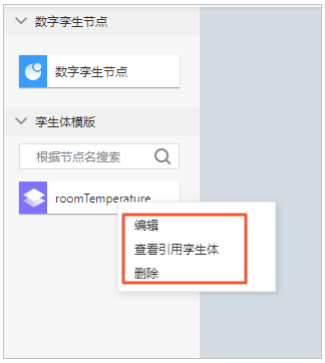
   在孪生体中，通过不同方式添加的孪生节点，对应功能的操作说明如下表，您可根据实际场景选择**导入方式**。

操作	引用方式	拷贝方式
删除孪生节点	支持直接在孪生体中操作，且与孪生体模板中操作互不影响	支持直接在孪生体中操作，且与孪生体模板中操作互不影响。
添加子节点		
编辑物模型	不支持在孪生体中操作。	
编辑孪生规则	必须在孪生体模板中操作，完成操作后，孪生体中孪生节点会同步生效。	



4. （可选）右键单击任意模板，然后单击以下按钮：

- **编辑**：进入模板编辑画布，支持删除、添加子节点，编辑物模型和孪生规则。
- **查看引用孪生体**：工作台右侧显示引用该模板的孪生体列表。您可单击**操作**列的**查看**，进入对应孪生体的详情页面。
- **删除**：仅支持删除未被引用的模板。



2.5. 添加数字映射

数据孪生体与物理设备之间是解耦关系，您可使用数据映射功能，将物理设备的原始数据映射到孪生体的业务模型中，即孪生节点的物模型属性上。本文介绍配置数据映射的具体操作。

前提条件

已为孪生节点配置功能属性。具体操作，请参见配置功能属性。

您需获取待添加数据映射的孪生节点ID，及其功能属性标识符，用于输出文件中数据映射的配置。

背景信息

数据映射由以下3部分组成：

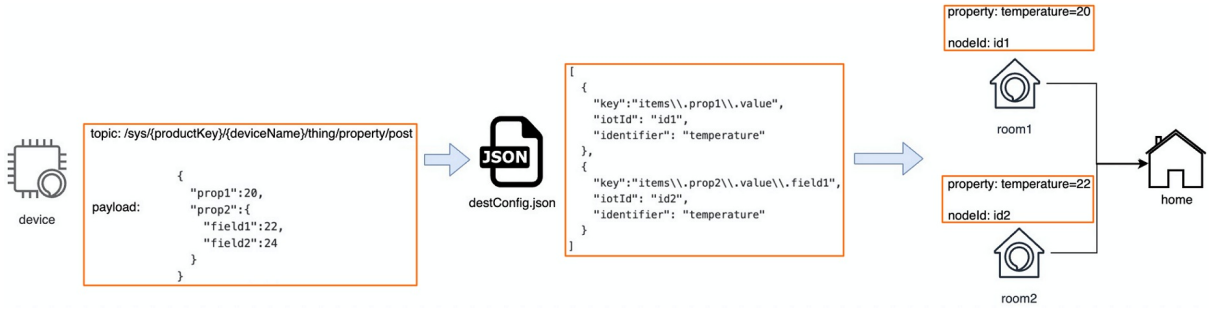
- 数据源：孪生节点数据所属的设备Topic，包含自定义Topic和物模型属性上报Topic。

  数据源作为脚本文件或输出文件的输入数据，其中自定义Topic数据是设备上报的原始数据格式，物模型属性上报Topic的数据格式，请参见设备属性上报。
- 脚本文件：使用JavaScript语言编辑脚本，解析处理设备Topic的原始数据，最后输出JSON格式数据。文件大小不超过128 KB。
- 输出文件：使用规定的JSON数据格式，将设备Topic数据或通过脚本处理后的数据，映射到指定孪生节点的物模型属性上。文件大小不超过256 KB。

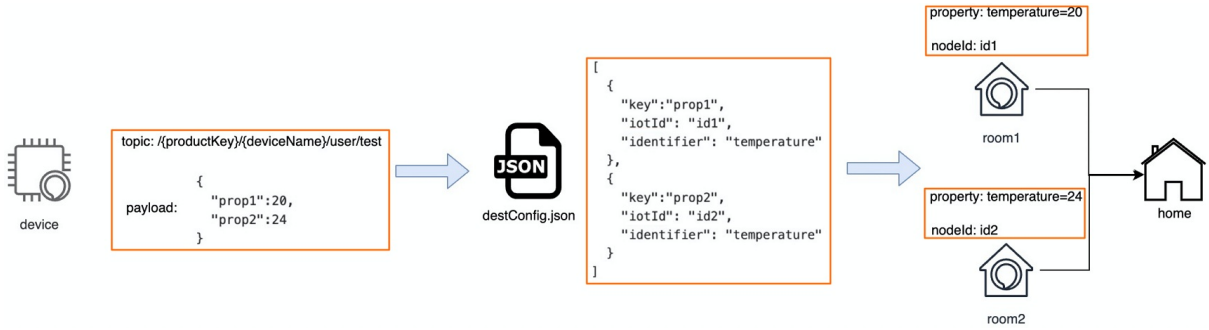
脚本和输出文件编辑方法，请参见脚本和输出语法。

数据映射使用场景如下：

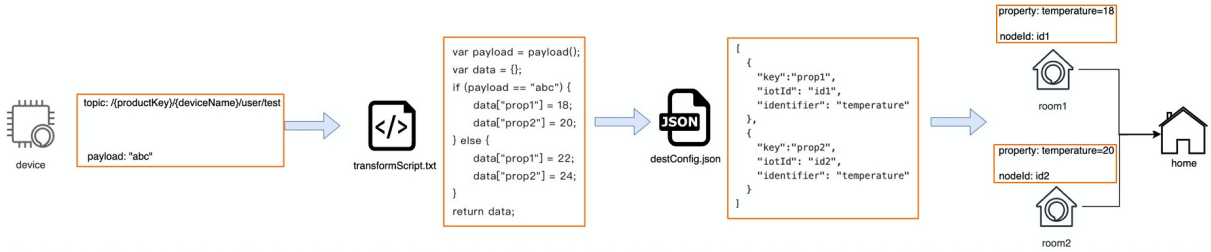
- 数据源为物模型属性Topic，无需脚本文件，使用输出文件完成数据映射，如下图所示。



- 数据源为自定义Topic，数据格式为JSON格式，无需脚本文件，使用输出文件完成数据映射，如下图所示。



- 数据源为自定义Topic，先使用脚本文件解析数据，再使用输出文件完成数据映射，如下图所示。



操作步骤

1. 在**数字孪生体详情**页面，单击**数据映射**页签，然后单击**添加数据映射**。
2. 在**添加数据映射**面板，配置以下参数，单击**确定**。

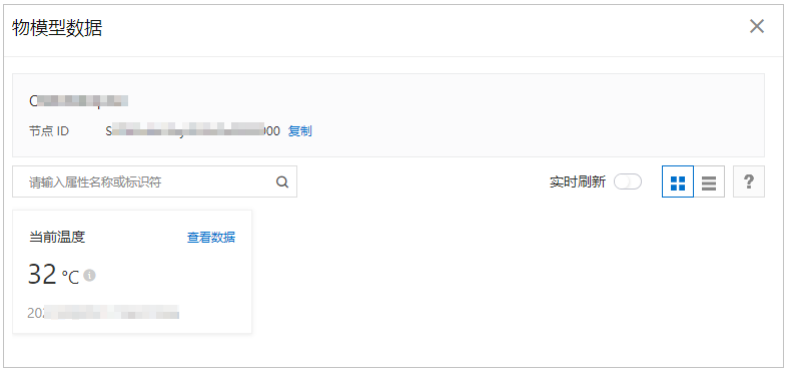
参数名称	说明
数据映射名称	自定义数据映射名称。支持中文、英文字母、数字、和特殊字符下划线（_）、短划线（-）、半角圆括号（()）和at符号（@），长度限制为4~30个字符，一个中文计为2个字符。
Topic	选择数据来源的消息Topic。 ○ **自定义**：指定消息源是自定义Topic时，支持使用通配符（+）和（#）。 ■ 全部设备（`+`）：指定产品下所有设备。 ■ `/user/#`：指定设备的所有自定义Topic。 自定义Topic说明，请参见自定义Topic。 ○ **物模型属性上报**：设备上报属性的Topic `thing/event/property/post`。
脚本	单击对应的**选择文件**，上传`.txt`格式的脚本文件、`.json`格式的输出文件。
输出	

3. 在数据映射列表，查看已添加数据映射脚本的解析状态。

   状态列显示**解析完成**，表示数据映射脚本添加成功。

   如果**状态**列显示**解析失败**，单击**下载失败原因**，根据文件中提示的错误信息，修正脚本。然后单击**操作**列的**编辑**，重新上传脚本。
4. 在**数字孪生体详情**页面，单击**运行状态**。
5. 在**运行状态**页签，单击孪生体节点，在**物模型数据**面板，可查看节点的属性数据。

   开启**实时刷新**开关，数据将实时刷新。



后续步骤

查看孪生体运行日志。

2.6. 查看孪生体运行日志

数字孪生体运行后，您可在物联网平台控制台，查看各孪生节点的运行日志，包括孪生节点数据更新、规则解析和数据映射的交互。本文介绍查看运行日志的具体操作，及日志中的错误码和排错方法。

查看运行日志

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 数字孪生**。
4. 在数字孪生列表，单击**操作**列的**查看**，进入**数字孪生体详情**页面。
5. 单击**数字孪生运行日志**页签，输入搜索条件，然后单击搜索图标，查看指定条件的日志。

   支持的搜索条件如下表。

搜索条件	说明
孪生节点ID	根据孪生节点ID，搜索该孪生节点的相关日志。
TraceId	根据追踪ID，搜索串联模块日志。
状态	查询某种结果状态的日志。可选择： ○ 全部状态 ○ 成功：状态码为200 ○ 失败：其他状态码
时间范围	选择要查询日志的时间范围。

日志字段说明

日志中包含的字段说明如下表。

参数	含义
时间	日志打印时间。
TraceId	追踪ID，可用于搜索串联模块。
数字孪生节点Id	孪生体中孪生节点ID，可用于搜索相关日志。
操作	显示触发孪生节点数据更新，对应的操作名称，包括： ● 更新孪生数据：孪生规则执行成功后，执行结果写到具体孪生节点的属性。 ● 孪生规则解析：孪生节点的数据变更，触发其配置的相关规则执行。 ● 数据映射：设备上报数据通过数据映射配置，映射到孪生体的节点中。
参数	当前操作对应的参数，内容由规则表达式、数据映射输入脚本决定。
结果	当前操作的处理结果，内容由规则表达式计算结果、数据映射输出参数决定。
状态码	结果码。200表示成功，其他表示失败。错误码说明，见下文。
状态信息	当前操作处理结果的信息。

错误码

数字孪生体物模型相关错误码含义，与设备物模型相关错误码含义相同，具体说明，请参见物模型相关错误码。

错误码	说明	排查
74109	孪生节点ID不存在。	检查孪生节点ID是否传错，或已被删除。

错误码	说明	排查
74200	表达式错误，或参数不合法，导致解析失败。	检查表达式的合法性，以及表达式参数的合法性。表达式说明，请参见规则表达式。
74202	表达式的输出类型无法转换到输出属性对应的类型。	检查表达式输出属性是否是基本类型，以及表达式的计算结果是否可以转换到输出属性。
74203	表达式参数错误，导致结果返回NaN。	检查NaN的产生原因，例如"0/0"或产生NaN。
74204	表达式参数错误，导致结果返回Infinity。	排查Infinity的产生原因，例如"1/0"或产生Infinity。
74205	表达式中包含无效变量，或计算结果为null，导致返回结果为null。	检查规则表达式是否正确，计算结果是否为null。
74400	父子规则在计算时，父节点ID不存在。	检查相关的节点是否被删除。

2.7. 规则表达式

数字孪生节点的孪生规则中支持多种运算符和函数，您可在编辑孪生规则时使用运算符和函数，实现数据处理的多样化。

运算符

运算符优先级依次递减顺序为：`( )`、`[ ]`、`.`、`**`、`!`、`~`、`*`、`/`、`%`、`+`、`-`、`<<`、`>>`、`>>>`、`<`、`<=`、`>`、`>=`、`==`、`!=`、`&`、`^`、`|`、`&&`、`||`。

数学函数

以下函数表达式中的入参n、n1和n2是数值型参数，且必填。函数返回值均为Double类型。

函数表达式	说明
abs(n)	返回n的绝对值。
acos(n)	返回n的反余弦值。
asin(n)	返回n的反正弦值。
atan(n)	返回n的反正切值。
ceil(n)	返回n最接近的整数。
cos(n)	返回n的余弦值。
cosh(n)	返回n的双曲余弦值。
cot(n)	返回n的余切值。
exp(n)	返回e的n次幂。
cbrt(n)	返回n的立方根。
expm1(n)	返回exp(n)-1的值。 n值较小时，使用此函数计算的结果比`exp(n)-1`更精确。
floor(n)	返回小于n的最近整数。
log(n)	返回log以e为底n的对数。
log1p(n)	返回log(1+n)的值。 n值较小时，使用此函数计算的结果比`log(1+n)`更精确。
log2(n)	返回log以2为底n的对数。
log10(n)	返回log以10为底n的对数。
pow(n1, n2)	返回n1的n2次幂。
rand()	返回[0, 1)之间的随机数。
signum(n)	返回n的符号。返回结果如下： • -1：负输入。 • 0：零输入。 • 1：正输入。
sin(n)	返回n的正弦值。
sinh(n)	返回n的双曲正弦值。
sqrt(n)	返回n的平方根。
tan(n)	返回n的正切值。
tanh(n)	返回n的双曲正切值。

条件函数

函数表达式	说明
condition(expression, resultIfTrue, resultIfFalse)	根据expression的计算结果，判断返回值。 expression为Boolean类型值，或计算结果为Boolean类型的表达式。 • expression为true，则返回resultIfTrue的结果。 • expression为false，则返回resultIfFalse的结果。 如果返回的resultIfTrue或resultIfFalse值为none，则表示不进行任何处理。 返回结果类型，取决于resultIfTrue或resultIfFalse的返回类型。
value(param, defaultValue)	如果param的值为空，则函数的返回结果为defaultValue，否则返回param。 param为基本数据类型，可为空，defaultValue为基本数据类型，不可为空。 返回结果类型，取决于defaultValue的返回类型。

字符函数

函数表达式	说明
contains(str1, str2)	判断字符串str1中是否包含字符串str2。 返回结果类型为Boolean。
endWith(str1, str2)	判断字符串str1中是否以字符串str2结尾。 返回结果类型为Boolean。
length(str)	返回字符串str的长度。 返回结果类型为Double。
startWith(str1, str2)	判断字符串str1中是否以字符串str2开头。 返回结果类型为Boolean。
substring(str, start, end)	返回字符串str从start（包括）到end（不包括）的子字符串。 • str：要操作的字符串。必填。 • start：起始下标。整数型，必填。 • end：结束下标。整数型，非必填。
join(param1, param2)	将param1和param2转换成字符串，并进行拼接。 param1和param2数据类型为数值型或字符串型。 返回结果类型为String。

2.8. 脚本和输出语法

脚本的解析能力包括获取消息内容、转换数据格式、处理字符串、组装JSON格式数据和处理二进制数据等。输出文件是使用固定的JSON数组格式，将设备上报数据或脚本解析后的数据，映射到数字孪生节点中。本文介绍如何编写脚本和输出文件。

背景信息

物联网平台是基于Topic中的数据格式来处理和传递数据的，数据格式的具体内容，请参见数据格式。

脚本和输出示例

本文以上报的属性数据为例，输入数据如下：

```
{
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "productKey":"a112345****",
    "deviceName":"deviceName1234",
    "gmtCreate":1510799670074,
    "deviceType":"Ammeter",
    "items":{
        "Power":{
            "value":"on",
            "time":1510799670074
        },
        "Position":{
            "time":1510292697470,
            "value":{
                "latitude":39.9,
                "longitude":116.38
            }
        }
    },
    "checkFailedData":{
    }
}
```

解析和处理数据的示例如下：

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var payload = payload("json");
//定义Map类型数据，存储键值对数据。
var data = {};
//获取位置的latitude值并且+1，将计算后的值存入data。
data["k1"] = payload.items.Position.value.latitude + 1;
//将Power的值进行转换，on转换成1，否则为0。
if (payload.items.Power.value == "on") {
  data["k2"] = 1;
} else {
  data["k2"] = 0;
}
//获取items的值。
data["items"] = payload.items;
//在data中添加一个常量stage1.stage2。
data["stage1.stage2"] = 1.3;
//返回解析后的数据data。
return data;
```

将脚本解析返回的data数据，作为输出文件的数据源，使用以下JSON格式，配置数据映射，示例如下：

```
[
  {
    //key值对应data数据中的字段。
    "key": "k1",
    //iotId值对应孪生节点ID。
    "iotId": "xx***",
    //孪生节点下的功能属性标识符，获取key对应字段的输出值，即节点“xx***”下属性id1值更新为k1的值。
    "identifier": "id1"
  },
  {
    "key": "k2",
    "iotId": "yy***",
    "identifier": "id2"
  },{
    "key": "items\\.Position\\.value\\.longitude",
    "iotId": "yy***",
    "identifier": "id3"
  },{
    "key": "stage1.stage2",
    "iotId": "zz***",
    "identifier": "id4"
  }
]
```

> 注意
>
> - 上报的JSON数据格式，为数组或者嵌套的JSON。脚本和输出文件中支持使用JSONPath获取其中的属性值。例如：
>   - 脚本文件中，使用 `payload.items.Position.value.latitude` 格式，获取到值39.9。
>   - 输出文件中，使用 `items\\.Position\\.value\\.longitude` 格式，获取到值116.38。
>
>   有关JSONPath的更多信息，请参见LanguageManual UDF。
> - 脚本中定义常量、变量或其他自定义字段的标识符时，半角句号（.）可作为正常字符使用。
> - 若设备上报数据是JSON格式，可不使用脚本解析数据，直接编辑输出文件，进行数据映射。

脚本编辑的更多语法说明，请参见脚本语法。

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。

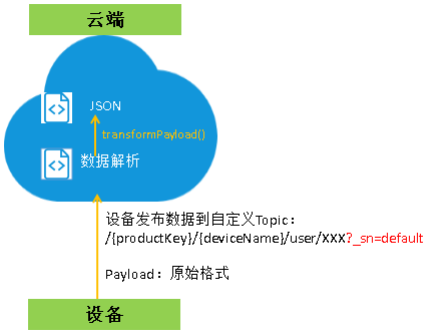
目前支持解析两类数据：

- 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。

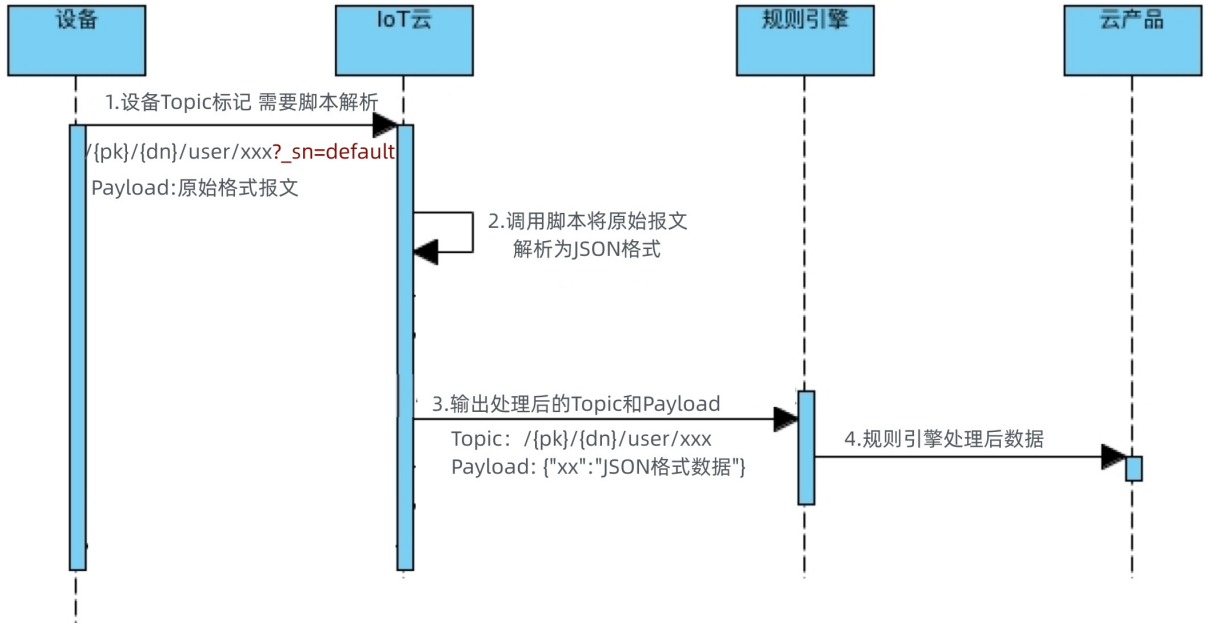
自定义Topic数据解析

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：

提交数据解析脚本

JavaScript脚本示例

Python脚本示例

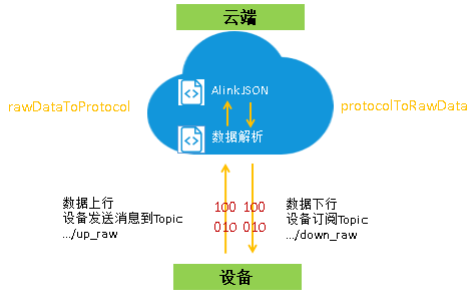
PHP脚本示例

物模型数据解析

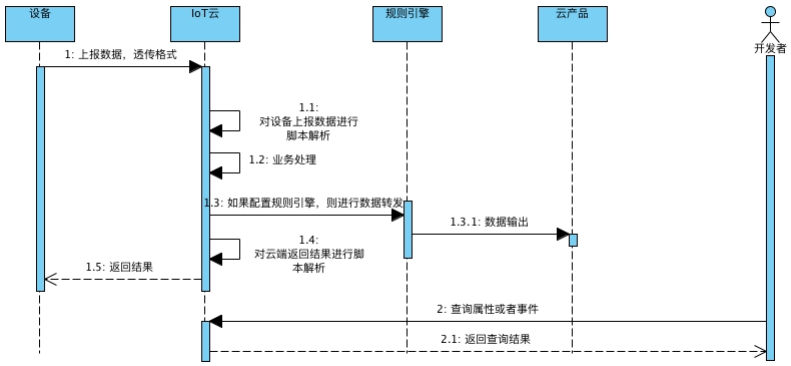
数据格式为**透传/自定义**的产品下的设备与云端进行物模型数据通信时，需要物联网平台调用您提交的数据解析脚本，将上、下行物模型数据分别解析为物联网平台定义的标准格式（Alink JSON）和设备的自定义数据格式。

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. 提交数据解析脚本

您需要在控制台提交数据解析脚本。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

- 仅华东2（上海）、华北2（北京）、华南1（深圳）、日本（东京）地域支持自定义Topic数据解析。
- 仅通过MQTT协议接入的设备支持自定义Topic数据解析。
- 仅解析设备上报云端的数据，不解析云端下行数据。
- 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。

解析标记

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。

操作步骤

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 产品**。
4. 在**产品**页，单击产品对应的**查看**。
5. 在**产品详情**页，选择**数据解析**页签。
6. 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。

7. 测试脚本。
   i. **模拟输入**下，选择**模拟类型**为**自定义**，并选择设备和Topic。
   ii. 输入模拟的设备上报数据，单击**执行**。
8. 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。

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** 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。
>
> 有关**透传/自定义**说明，请参见创建产品。

```
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
     bytes: 0x00000000010032010000000
  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

本文介绍Python语言的自定义Topic数据解析脚本模板和示例。

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。
>
> 有关**透传/自定义**说明，请参见创建产品。

```
# 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
#     bytes: 0x0000000001003201000000000
#  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。

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）的入参为整型数组。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** 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。
>
> 有关**透传/自定义**说明，请参见创建产品。

```
<?php
/*
  示例数据
  自定义Topic:
     /user/update，上报数据。
  输入参数:
     topic: /${productKey}/${deviceName}/user/update
     bytes: 0x00000000100320100000000。
  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，介绍在数据格式为透传或自定义的产品下，物模型数据解析脚本的编写方法。

> **说明** 本示例中的属性为默认模块属性，若使用自定义模块，标识符（identifier）的格式为 `模块标识符:属性标识符` 。例如， `model1:prop_int16` 。

步骤一：编辑脚本

1. 在物联网平台控制台，创建产品。

   具体操作，请参见创建产品。

   > **说明** 数据格式选择为**透传/自定义**。

2. 为该产品定义物模型。自定义属性的操作说明，请参见单个添加物模型。

   本示例中定义了以下三个属性：

标识符（identifier）	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. 本示例通信中参数值长度定义：

   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
属性prop_float	4字节

   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

   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

   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。
   i. 在**产品**页，单击产品对应的**查看**。
   ii. 在**产品详情**页，选择**数据解析**页签。

iii. 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
  - PHP 7.2：protocolToRawData
- 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
  - PHP 7.2：rawDataToProtocol

完整的示例脚本Demo，请参见JavaScript脚本示例、Python脚本示例和PHP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，请参见提交数据解析脚本。
>
> 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，请参见透传/自定义产品完整示例脚本（JavaScript）、透传/自定义产品完整示例脚本（Python）和透传/自定义产品完整示例脚本（PHP）。

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。

> **说明** 下文中传入和返回的十六进制字符串和JSON格式数据仅为示例，实际场景中，不可直接用于调试。

- 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**。

  > **说明** 以下传入参数模拟数据仅适用于JavaScript脚本，更多示例内容，请参见JavaScript脚本示例。
  >
  > Python、PHP脚本的传入参数模拟数据，请参见Python脚本示例、PHP脚本示例。

  您可使用字符串转十六进制工具，将待传入参数JSON格式数据转为十六进制格式数据。例如转化后为 `0x00002233441232013fa00000` ，则输入如下数据。

  ```
  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。

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。
  i. 使用设备端上报设备属性数据，例如 `0x000022334412320l3fa00000` 。
  ii. 在物联网平台控制台，选择**设备管理 > 设备**。
  iii. 单击设备对应的**查看**，然后在**设备详情**页**物模型数据 > 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。
  i. 在物联网平台控制台，选择**监控运维 > 在线调试**。
  ii. 选择要调试的产品和设备，并选择**默认模块**，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
      }
  }
  ```

  iii. 查看设备端是否收到该属性设置指令。
  iv. 在该设备的**设备详情**页**物模型数据 > 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
- 查看PHP 7.2脚本模板和示例，请参见PHP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，请参见提交数据解析脚本。
- 有关调用API向设备发送消息，请参见物模型使用相关API和消息通信相关API。

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）的入参为整型数组。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。数组元素为整型，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。

物模型中数据类型说明，请参见物模型支持的数据类型。物模型属性、事件上报数据后，物联网平台返回的Alink格式响应结果，会通过脚本解析转换后返回给设备。Alink数据格式说明，请参见设备属性、事件、服务。

```
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 COMMAD_UNKOWN = 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，设备上报属性设置的结果到云端。
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：
设备上报属性数据：
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：
云端下发属性设置指令：
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字段。
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); //ALink JSON格式 'id'。
        payloadArray = payloadArray.concat(buffer_int16(prop_int16)); //属性'prop_int16'的值。
        payloadArray = payloadArray.concat(buffer_uint8(prop_bool)); //属性'prop_bool'的值。
        payloadArray = payloadArray.concat(buffer_float32(prop_float)); //属性'prop_float'的值。
    } 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)); //command字段。
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); //ALink JSON格式'id'。
        payloadArray = payloadArray.concat(buffer_uint8(code));
    } else { //未知命令，对于这些命令不做处理。
        var code = json['code'];
        payloadArray = payloadArray.concat(buffer_uint8(COMMAD_UNKOWN)); //command字段。
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); //ALink JSON格式'id'。
        payloadArray = payloadArray.concat(buffer_uint8(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
     bytes: 0x0000000001003201000000000
  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;
}
//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）的入参为整型数组。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。数组元素为整型，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。

物模型中数据类型说明，请参见物模型支持的数据类型。物模型属性、事件上报数据后，物联网平台返回的Alink格式响应结果，会通过脚本解析转换后返回给设备。Alink数据格式说明，请参见设备属性、事件、服务。

```
# coding=UTF-8
import struct
import common_util
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。
COMMAD_UNKOWN = 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。
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：
# 设备上报属性数据：
# 传入参数：
#     0x0000000001003201000000000
# 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:
# 云端下发属性设置指令:
# 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
#  示例数据:
#  自定义Topic:
#      /user/update，上报数据。
#  输入参数:
#      topic:/{productKey}/{deviceName}/user/update
#      bytes: 0x0000000001003201000000000
#  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转成int。
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16)
# 8位整型转成byte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位整型转成byte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位整型转成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转成整型数组。
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

您可以基于以下PHP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。

```
<?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：$jsonObj，关联数组，数组key取值为英文字符串，不能是字符类型的数字，如"10"，不能为空。
 */
function rawDataToProtocol($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，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 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）的入参为整型数组。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。数组元素为整型，取值范围为0~255。

- 自定义协议解析的函数（transformPayload）的入参为整型数组。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属性和通信协议编写的脚本。

物模型中数据类型说明，请参见物模型支持的数据类型。物模型属性、事件上报数据后，物联网平台返回的Alink格式响应结果，会通过脚本解析转换后返回给设备。Alink数据格式说明，请参见设备属性、事件、服务。

```
<?php
/*
示例数据：
设备上报数据：
传入参数：
    0x0000000001003201
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 = $data[0]; //command字段。
    if ($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，对于这些命令不做处理。
        $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 $payloadArray;
}
/*
  示例数据：
  自定义Topic：
     /user/update，上报数据。
  输入参数：
     topic: /{productKey}/{deviceName}/user/update
     bytes: 0x0000000001003201000000000
  输出参数：
  {
     "prop_float": 0,
     "prop_int16": 50,
     "prop_bool": 1,
     "topic": "/{productKey}/{deviceName}/user/update"
   }
 */
function transformPayload($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. 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时的排错方法。

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。

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，在设备列表单击该设备对应的**查看**，进入**设备详情**页，在**物模型数据 > 运行状态**页签下，可以看到设备上报的数据。

若设备已经上报了数据，但是却没有显示对应的数据，可选择**监控运维 > 日志服务 > 云端运行日志**，通过日志排查问题。

问题排查过程如下：

1. 选择对应产品，输入DeviceName，单击**搜索**，业务类型选择*物模型上报*，查询该设备的相关日志。
2. 查看日志记录，日志中会显示脚本转化后的数据和原数据。
3. 结合日志说明文档，查看错误码的信息。
4. 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。

  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。

  日志内容如：

  ```
  17:54:19.064, A7B02C60646B4D2E8744F7AA7C3D9567, upstream-error - bizType=OTHER_MESSAGE,params={"params":{}},result=code:6450,message:alink method n
  ot 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

您可以使用标签功能为产品、设备或分组自定义标识，以便灵活管理产品、设备和分组；使用地理位置标签标记设备地理位置，为设备设置GeoLocation属性值。本文介绍添加标签的操作步骤。

前提条件

添加标签前，需先对应完成产品、设备或分组的创建，请参见：

- 创建产品。
- 创建设备。
- 创建分组。

背景信息

物联网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。

使用说明

- 产品标签、设备标签和分组标签的结构为 `Key:Value` 。
- 每个产品、设备或分组最多可有100个标签。

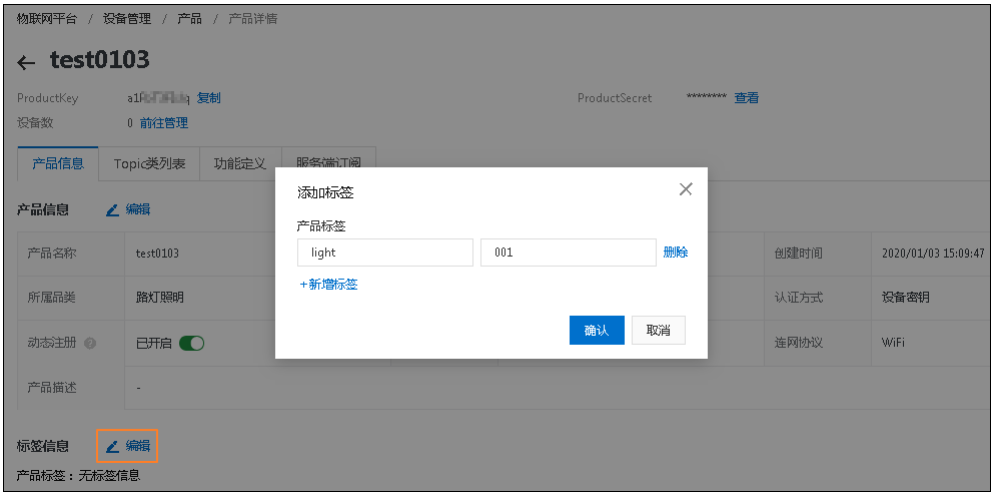
添加产品标签

产品标签通常描述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。

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   注意 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 产品**。
4. 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**。
5. 单击**标签信息**右侧的**编辑**。



6. 在**编辑标签**对话框中，输入标签的标签Key和标签Value，然后单击**确认**。

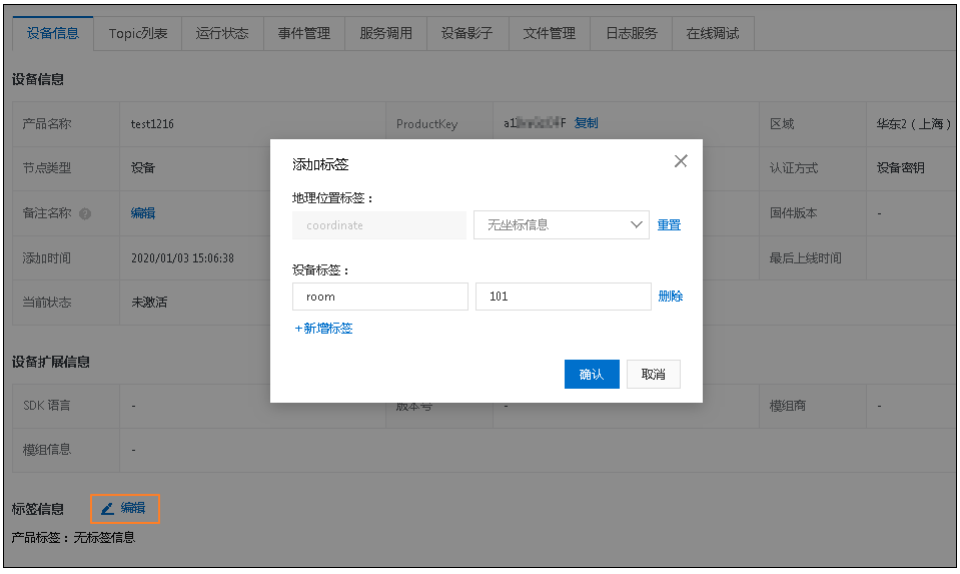
参数	说明
标签Key	可包含英文字母、数字和英文句号（.），长度不超过30个字符。
标签Value	可包含中文、英文字母、数字、日文、下划线（_）、短划线（-）、英文冒号（:）和英文句号（.），长度不超过128个字符，一个中文或日文占2个字符。

添加设备标签

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，流转的数据格式，请参见设备标签的Alink协议。
- 物联网平台可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。流转的数据格式，请参见设备标签变更。

1. 在目标实例的左侧导航栏，选择**设备管理 > 设备**。
2. 在**设备**页面，单击要添加标签的设备所对应的**查看**。
3. 在**设备详情**页面，单击**标签信息**右侧的**编辑**。



4. （可选）在**编辑标签**对话框中，单击**地理位置标签**的文本框，在弹出的地图中选择位置，可以按地址搜索。

> **说明** 地理位置标签用于标记设备的地理位置，结构为 `coordinate: 经度:纬度` 。
> 如果设备所属产品的物模型包含GeoLocation属性，设置地理位置标签后，该地理位置信息将同步为GeoLocation属性值。

5. 在**编辑标签**对话框中，单击**新增标签**，输入标签的标签Key和标签Value。

参数	说明
标签Key	可包含英文字母、数字、正斜线（/）、下划线（_）、短划线（-）、井号（#）、at（@）、百分号（%）、and（&）、星号（*）和英文句号（.），长度不超过30个字符。
标签Value	可包含中文、英文字母、数字、日文、下划线（_）和短划线（-）、井号（#）、at（@）、百分号（%）、and（&）、英文冒号（:）和英文句号（.），长度不超过128个字符，一个中文或日文占2个字符。

6. 单击**确认**。

添加分组标签

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地域、空间等。

1. 在目标实例的左侧导航栏，选择**设备管理 > 分组**。
2. 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**。
3. 单击**标签信息**右侧的**编辑**。

4. （可选）在**编辑标签**对话框中，单击**地理位置标签**的文本框，在弹出的地图中选择位置，可以按地址搜索。

> **说明** 地理位置标签用于标记设备的地理位置，结构为 `coordinate: 经度:纬度` 。
> 如果设备所属产品的物模型包含GeoLocation属性，设置地理位置标签后，该地理位置信息将同步为GeoLocation属性值。

5. 在**编辑标签**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、数字和英文句号（.），长度不超过30个字符。
标签Value	可包含中文、英文字母、数字、日文、下划线（_）、短划线（-）、英文冒号（:）和英文句号（.），长度不超过128个字符，一个中文或日文占2个字符。

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，或根据标签来查询产品、设备和分组。详细信息，请参见API列表。

5.高级搜索

当您需要在物联网平台搜索并下载指定条件的设备列表（包含ProductKey和DeviceName）时，可以使用高级搜索功能，通过类SQL语句快速搜索满足指定条件的设备，例如在线设备。本文介绍高级搜索操作，以及使用的类SQL语法。

使用限制

- 支持地域：华东2（上海）、华北2（北京）、华南1（深圳）。
- 功能限制：
  - 旧版公共实例：不支持通过物模型搜索设备和下载设备文件。
  - 企业版实例和新版公共实例：仅华东2（上海）地域下，高级搜索中通过物模型搜索设备的功能，仅可在**设备管理 > 设备**页面的**高级搜索**页签使用。

    使用物模型搜索功能，需先配置物模型索引，然后物联网平台基于设备运行时数据，检索设备最近一次上报的属性数据，搜索出符合条件设备。

    > **注意** 一个企业版实例下，最多可配置100个物模型属性的索引。

使用场景

物联网平台控制台的以下使用场景支持高级搜索：

- 管理设备：在**设备管理 > 设备**页面的**高级搜索**页签，从设备列表搜索出指定设备进行管理。
- 设备分组：在向设备分组添加设备时，搜索待添加的设备。
- OTA升级：在验证OTA升级包、创建批量升级批次时，搜索待升级的设备。

您也可以调用QueryDeviceBySQL API进行设备高级搜索。使用API进行高级搜索时，不局限于以上控制台使用场景。

操作步骤

本文以使用高级搜索功能向设备分组添加设备、在设备管理中通过物模型搜索设备为例，介绍物联网平台控制台的设备高级搜索操作。

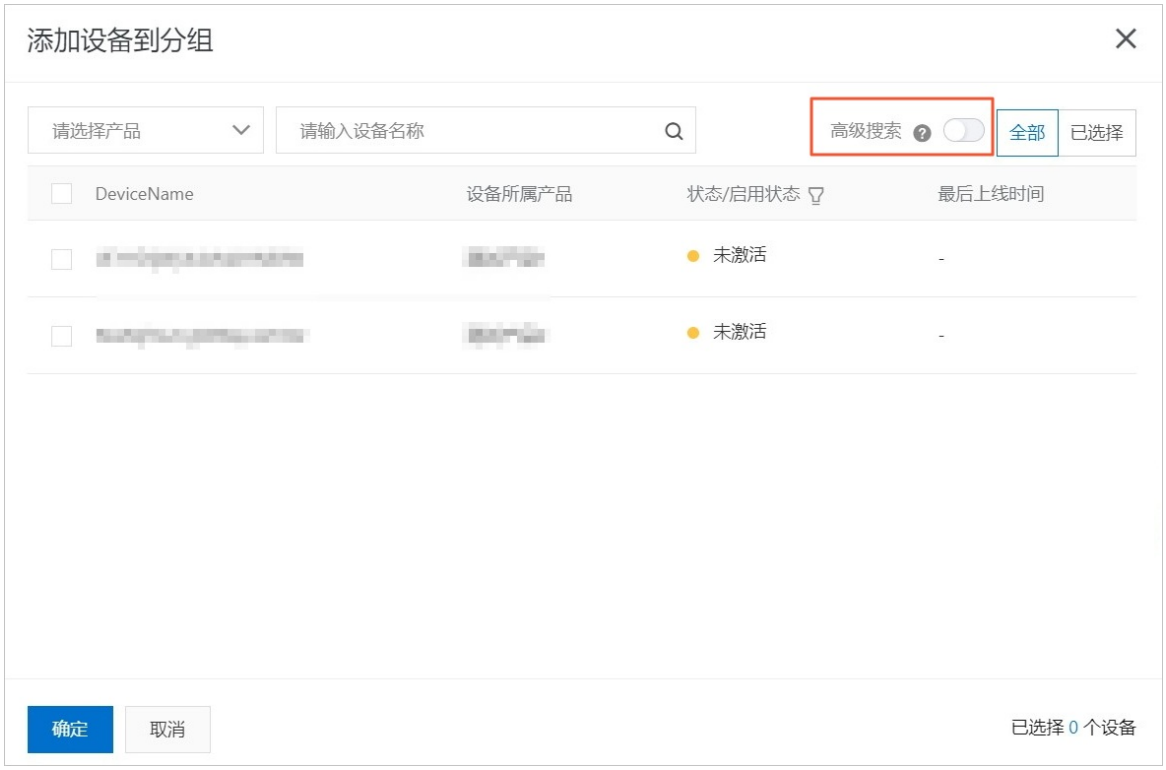
1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

   > **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 通过高级搜索功能，搜索符合条件设备。
   i. 在左侧导航栏，选择**设备管理 > 分组**。在**分组**页面，单击已创建分组对应的**查看**。

ii. 在**分组详情**页面，单击**设备列表 > 添加设备到分组**。在**添加设备到分组**面板，开启右上角**高级搜索**开关。

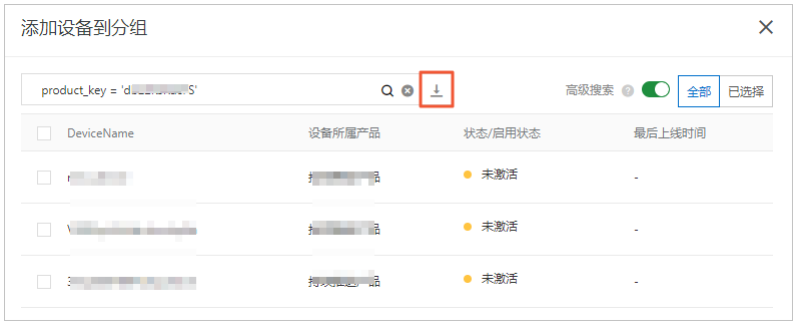


iii. 在搜索框中输入搜索条件的语句，单击搜索图标。

搜索条件语句语法的更多信息，请参见类SQL语法说明。

列表显示搜索结果，搜索框右侧显示下载图标。

您可单击下载图标，获取已搜索设备的列表文件，格式为CSV。



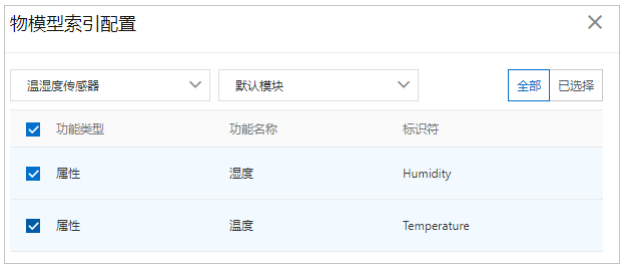
4. 通过物模型搜索设备。

i. 在左侧导航栏，选择**设备管理 > 设备**，单击**高级搜索**页签。

ii. 在**高级搜索**页签，单击**通过物模型数据搜索**右侧的**配置索引**。

iii. 在**物模型索引配置**面板，选择产品和物模型模块，然后选中待索引属性，单击**确定**。

> **说明**
>
> - 配置属性索引后，最多一分钟即可生效。
> - 配置属性索引后，如果您删除了产品下该物模型属性，仍可基于该属性最近一次上报数据检索设备。



iv. 在搜索框中输入物模型索引条件，单击**搜索**。

您可在**通过物模型数据搜索**下方，单击复制**property.floatText > 11**或**property.module.floatText > 11**，然后将floatText替换为已配置索引的属性标识符；module替换为对应的自定义模块标识符；**11**替换为您要搜索的条件值。

> **注意** 条件值的数据类型必须与物模型属性的数据类型保持一致。
>
> 例如，物模型属性Temperature（温度）为Double类型，需要搜索上报温度值大于11的设备，条件值必须设置11.0，否则无法进行搜索。

在**高级搜索**页签，显示搜索结果。如果有满足条件的设备，您可单击**下载搜索结果**，获取已搜索设备的列表文件，格式为CSV。

v. （可选）如果您需要取消属性的索引能力，在**高级搜索**页签，单击**重置**，回到搜索页面。然后单击**配置索引**，进入**物模型索引配置**面板。

您可先选择产品和物模型模块，或单击右上角**已选择**，然后取消选中的目标属性，单击**确定**。

如果产品下已删除属性，您可在**已选择**列表，找到要取消索引的属性。

> **注意** 任何情况下，如果您取消了属性的索引能力，都不再支持通过该属性搜索设备。

取消属性索引，会清空该属性的历史搜索数据。如果累计历史数据过多，清空操作会花费时间，属性索引状态不会立刻取消，即您重新打开**物模型索引配置**面板，该属性可能仍处于选中状态。此时，请您耐心等待，不可重复删除操作。

类SQL语法说明

在控制台使用高级搜索功能时，类SQL语句由WHERE子句、ORDER BY子句（可选）组成，省略SELECT子句、LIMIT子句以及WHERE子句的`WHERE`。长度限制为400个字符。

示例：

```
product_key = "a1*****" order by active_time
```

WHERE子句

格式：

```
[condition1] AND [condition2]
```

省略`WHERE`。

最多使用5个condition，且不支持嵌套，请参见下面的检索字段说明、运算符说明。

连接词支持AND、OR，最多使用5个连接词。

ORDER BY子句（可选）

ORDER BY子句用来实现自定义排序，可自定义排序的字段包括gmt_create、gmt_modified、active_time。

该子句可不填，不填时随机排序。

检索字段说明

字段名	类型	说明
product_key	text	设备所属产品ProductKey。
iot_id	text	设备标识符。默认返回iot_id。
name	text	设备名称。
active_time	date	设备激活时间。格式为yyyy-MM-dd HH:mm:ss.SSS，精确到毫秒。
nickname	text	设备备注名称。
gmt_create	date	设备创建时间。格式为yyyy-MM-dd HH:mm:ss.SSS，精确到毫秒。
gmt_modified	date	设备信息最后一次更新时间。格式为yyyy-MM-dd HH:mm:ss.SSS，精确到毫秒。
status	text	设备状态，取值： • ONLINE：在线 • OFFLINE：离线 • UNACTIVE：未激活 • DISABLE：已禁用

字段名	类型	说明
property.floatText	text	物模型属性标识符。 • property.floatText：默认模块属性。 • property.module.floatText：自定义模块属性。 仅企业版实例下**设备管理 > 设备**页面的**高级搜索**页签支持该字段检索。
property.module.floatText	text	
group.group_id	text	设备分组ID。
tag.tag_name	text	设备标签名。
tag.tag_value	text	设备标签值。
ota_module.name	text	OTA模块名称。 建议与ota_module.version配合使用，用于指定设备当前OTA版本号对应的OTA模块。
ota_module.version	text	设备当前OTA版本号。 建议与ota_module.name配合使用。

运算符说明

运算符	支持的字段数值类型
=	number、date、text、keyword
>	number、date
<	number、date
LIKE	text

其中，`LIKE`支持前缀匹配，不支持后缀匹配或通配符匹配。前缀必须满足以下条件：

- 前缀不得少于4个字符，且不能包含任何特殊字符，例如反斜线（\）、正斜线（/）、and（&）、加号（+）、短划线（-）、感叹号（!）、半角圆括号（()）、半角冒号（:）、波浪线（~）、方括号（[]）、大括号（{}）、星号（*）、半角问号（?）等。
- 前缀填写完成后，必须固定以`%`结尾。

示例：

```
product_key = "a1*********" and name LIKE "test%"
```

6.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进行跨产品管理设备。本文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

设备分组分为静态和动态两种类型，使用说明见下表。

分组类型	使用说明
静态分组	手动添加设备到分组，同时支持手动移出分组中设备。 支持添加子分组，使用限制如下： • 一个阿里云账号下最多可创建1,000个分组，包括所有的分组和子分组。 • 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：分组>子分组>子子分组。 • 一个子分组只能属于一个父组。 • 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。需全部删除子分组后，才能删除父组。 • 支持在分组列表和子分组列表，进行分组名称模糊搜索。
动态分组	配置分组条件规则，动态匹配符合条件的设备到分组，不支持手动添加和移出设备。 使用限制如下： • 当前仅华东2（上海）地域下，企业版实例和有ID的公共实例，支持动态分组功能。 • 一个阿里云账号下最多可创建10个动态分组。 • 动态分组默认为父组，且不支持添加子分组。 • 首次创建动态分组，配置分组规则后，规则最多允许匹配100,000设备，分组创建成功后，增量设备数无限制。

创建静态分组

1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 分组**，进入**分组**页面。
4. 单击**创建分组**，在弹出对话框中，设置分组参数，并单击**确认**。

参数	说明
分组类型	选择**默认**。仅华东2（上海）地域的企业版实例中，显示该参数。
父组	选择父组。 ◦ 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。

参数	说明
分组名称	设置分组名称。分组名称支持中文、英文字母、日文、数字和下划线（_），长度限制为4~30个字符，一个中文或日文占2个字符。 分组名称必须在账号下唯一，且创建后不能修改。
分组描述	描述该分组信息。

分组创建成功后，您可在**分组详情**页面，管理分组和设备。

创建动态分组

1. 登录物联网平台控制台。
2. 在控制台左上方，选择**华东2（上海）**地域，然后在**实例概览**页面，找到对应的实例，单击实例。
3. 在左侧导航栏，选择**设备管理 > 分组**，进入**分组**页面。
4. 单击**创建分组**，在弹出对话框中，按照以下步骤，完成动态分组配置。
   i. 选择**分组类型**为**动态**。

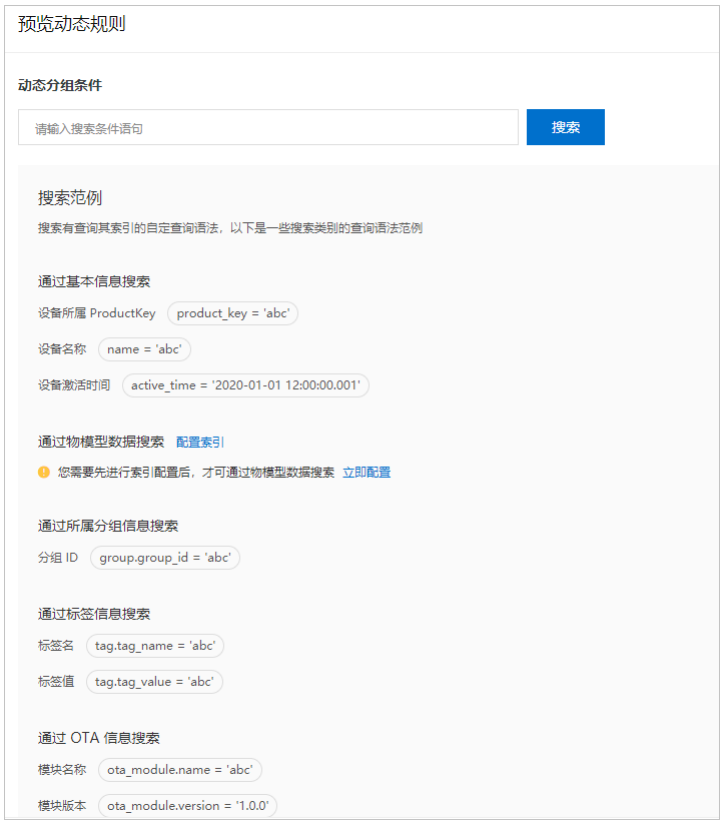
   > **说明** **分组类型**为**默认**，表示创建静态分组，具体配置，请参见上文创建静态分组。

   ii. 在**动态规则**下，输入查询设备的条件规则。

   您可单击**预览动态规则**，在右侧面板的**动态分组条件**下，输入查询条件，单击**确定**。

   分组的查询条件设置方法与设备高级搜索功能相同，详细说明，请参见高级搜索。

   > **注意**
   > - 通过物模型数据搜索设备时，需先配置物模型属性索引，并在一分钟后，再基于该物模型属性创建动态分组来检索设备。
   >
   >   一个动态分组中最多支持使用100个物模型属性。
   > - 通过设备OTA模块数据搜索设备时，若该动态分组用于设备OTA动态升级，则ota_module.name和ota_module.version必须结合使用。OTA升级更多信息，请参见OTA升级概述。



   iii. 输入**分组名称**和**分组描述**，单击**确认**。

   分组名称支持中文、英文字母、日文、数字和下划线（_），长度限制为4~30个字符，一个中文或日文占2个字符。

   分组名称必须在账号下唯一，且创建后不能修改。

动态分组创建成功后，最多一分钟生效。生效后，初始化存量设备最多需要20分钟。动态分组中仅包含符合条件的设备，您可在**分组详情**页面，查看设备列表。

> **说明** 动态分组创建成功后，不支持修改动态规则。

管理分组

1. 在**设备管理 > 分组**页面的分组列表中，单击对应分组操作列的**查看**，进入**分组详情**页面。
2. （可选）在**分组信息**页签，为分组添加标签，即自定义分组标识，以便灵活管理分组。具体操作，请参见添加分组标签。
3. 单击**设备列表**页签，查看或添加设备。
   - 添加设备。

     > **注意** 仅静态分组，支持此操作。
     > - 单次最多添加1,000个设备。单个分组最多添加100,000个设备。
     > - 一个设备最多可以被添加到10个分组中。

     单击**添加设备到分组**，在**添加设备到分组**面板，搜索并选中设备，单击**确定**。

     在**添加设备到分组**面板中，可单击**全部**查看所有设备列表，也可单击**已选择**，查看已选中设备的列表。

     在**添加设备到分组**面板，您可开启列表右上方的**高级搜索**，搜索设备，还可下载已搜索设备的CSV格式文件。更多信息，请参见高级搜索。
   - 搜索查看设备。

     您可查看**全部产品**下的所有设备，也可选择某个产品，查看该产品下的所有设备。

     您可输入至少5个字符，作为设备名称前缀模糊搜索设备。例如，输入*Dtest*，可搜索出账号下名称为Dtest1、Dtest2、Dtest3的设备。
4. （可选）单击**子分组列表 > 创建分组**，为分组添加子分组。

   > **说明** 仅静态分组支持此操作。

   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：

   i. 选择该子分组的父组，输入子分组名称和描述，然后单击**确认**。
   ii. 在**子分组列表**页签，单击子分组对应的**查看**。
   iii. 在子分组的**分组详情**页面，单击**设备列表**页签，然后单击**添加设备到分组**，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，还可以在子分组下再创建子分组。

7.设备任务

7.1. 设备任务概述

物联网平台支持设备任务功能，可同时向多个设备发起属性设置、异步服务调用、消息发送和自定义的任务。本文介绍设备任务的使用及流程。

使用说明

物联网平台的设备任务，是按照已定义的JSON格式的规则文件，在设备端实现对应的任务功能。

任务	说明
自定义任务	可根据业务需要，自行开发自定义任务规则内容和设备文件。 您需在设备端，开发任务功能，并根据物联网平台定义的Alink协议格式，开发任务实现逻辑。
设备批量属性设置任务	产品、设备或分组维度下的多个设备，包含相同的物模型属性标识符时，可通过本任务同时设置多个的设备的一个或多个属性值。
设备批量服务调用任务	产品、设备或分组维度下的多个设备，包含相同的物模型服务（异步调用）标识符时，可通过本任务同时调用多个设备的异步服务。
Pub批量消息推送任务	产品、设备或分组维度下的多个设备，其设备自定义Topic的后缀，配置均相同时，可通过本任务批量调用接口Pub，向多个设备发布消息。 自定义Topic格式为 `/${productKey}/${deviceName}/user/${TopicShortName}` ，其中 `${TopicShortName}` 为自定义Topic后缀。更多信息，请参见自定义Topic。

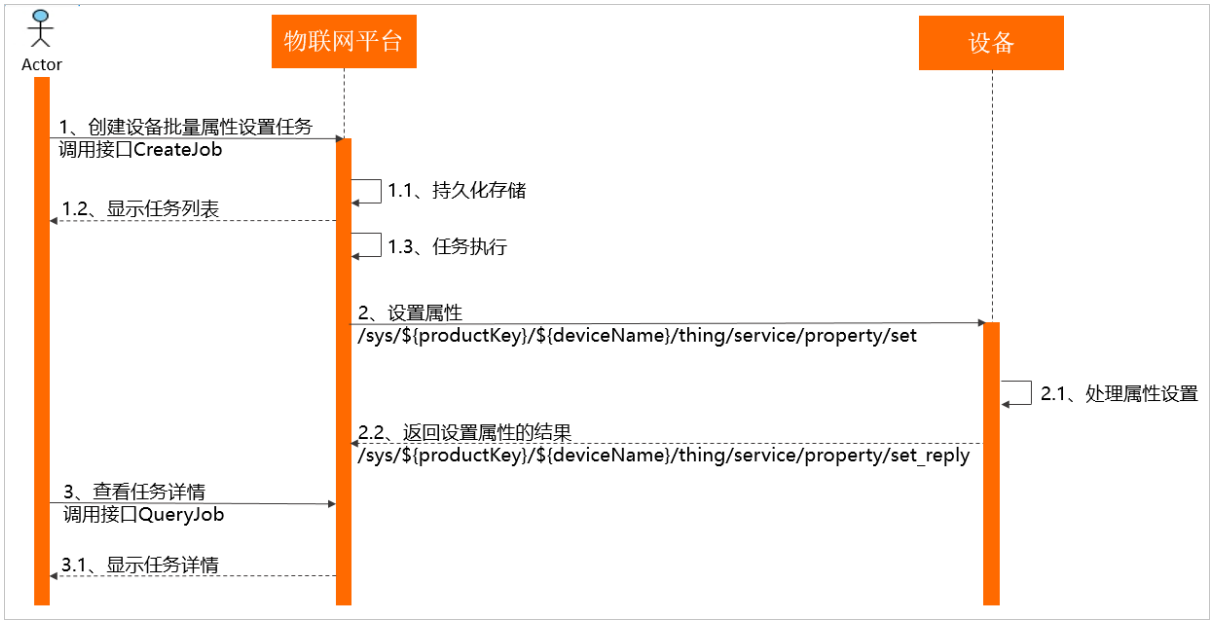
流程说明

不同类型设备任务，实现流程如下图所示。

- 自定义任务

相关Topic的使用说明，请参见设备任务的ALink协议说明。

- 设备批量属性设置任务

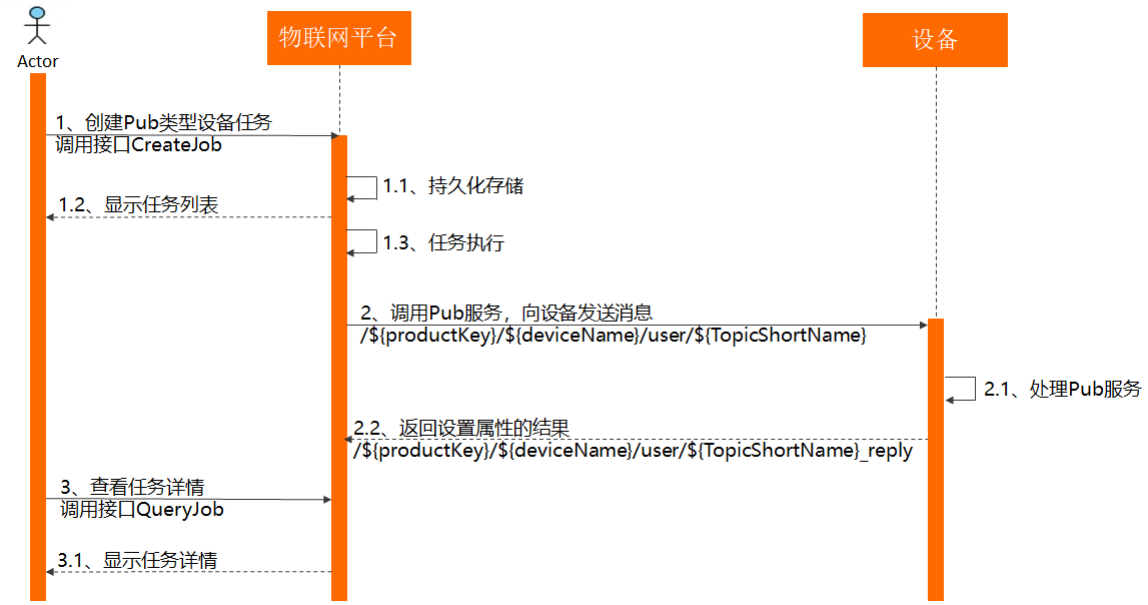


相关Topic的使用说明，请参见设置设备属性。

- 设备批量服务调用任务

相关Topic的使用说明，请参见设备服务调用（异步调用）。

- Pub批量消息推送任务



自定义Topic配置和使用说明，请参见自定义Topic。

您可根据实际业务需求，添加对应的设备任务。具体操作，请参见：

- 添加自定义任务
- 添加属性设置任务
- 添加服务调用任务
- 添加消息批量下发任务

7.2. 添加自定义任务

设备批量的属性设置和服务调用任务，无法满足设备端业务需求时，您可自定义任务规则，实现多功能场景任务。本文介绍该任务的创建方法、运行中Topic及其数据格式的说明，查看状态的具体操作。

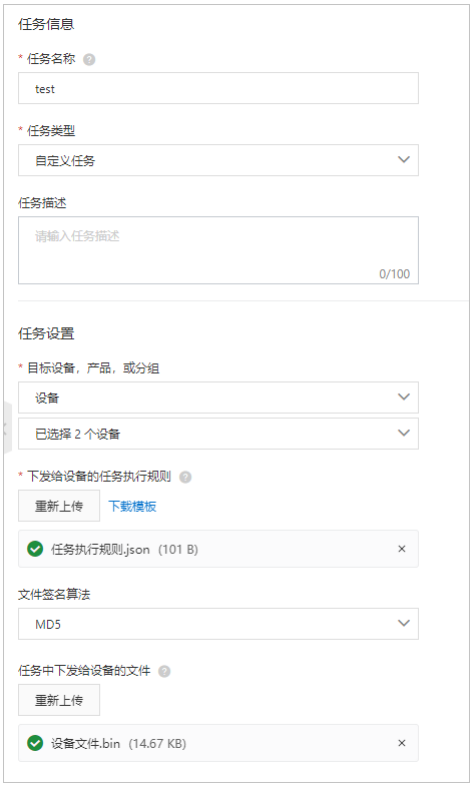
前提条件

已在设备端完成任务部署和管理功能开发。具体操作，请参见Link SDK的设备任务。

任务管理流程

1. 创建自定义任务。
   i. 在物联网平台控制台，对应实例下的**设备管理 > 任务**页面，单击**创建任务**。
   ii. 在**创建任务**页面，单击以下参数名称右侧的帮助图标，根据页面提示，配置任务和作业，单击**完成**。

- 任务配置



参数	说明
任务名称	输入符合规则的任务名称。可自定义。
任务类型	选择类型：**自定义任务**。
任务描述	输入该任务的用途等信息，便于您区分不同的任务。
目标设备、产品或分组	选择可执行任务的设备。
下发给设备的任务执行规则	上传规则文件。仅支持 `.json` 格式文件，文件大小不能超过64 KB。 您可单击**下载模板**，获取规则文件模板。内容如下： `{ "key":"value" }`
文件签名算法	支持**MD5**和**SHA256**。 可选配置，与**任务中下发给设备的文件**结合使用。
任务中下发给设备的文件	上传自定义任务的文件。 可选参数，与**文件签名算法**结合使用。 支持 `.bin` 、`.apk` 、`.tar` 、`.gz` 、`.zip` 、`.gzip` 、`.tar.gz` 格式文件，文件大小不能超过1,000 MB。

注意 您需根据实际需求，自定义规则和设备文件内容。任务规则和设备文件内容的实现逻辑，需您在设备端完成开发。

- 作业配置

参数	说明
作业执行推送配置	■ **每分钟作业执行数量**：根据您的业务需要，设置每分钟作业推送数量。 ■ **推送消息类型**：仅对自定义任务和Pub批量消息推送任务生效。 可选： ■ **QoS0**：最多发送一次。 ■ **QoS1**：最少发送一次。如果QoS1消息未接收到PUBACK消息，会在设备重连时，重新推送给设备。
作业执行的超时配置	可选配置。不设置表示不会超时。仅对自定义任务生效。 从设备任务进入IN_PROGRESS状态，开始计算时间。如果超过了超时时间，任务下作业仍未执行完成，作业状态将被自动设置为TIMED_OUT，作业停止执行。
作业开始调度时间	可选配置。 从当前设置操作的时间，开始计算时间。 设备任务创建成功后，先初始化，直至到达调度时间，才会开始调度执行。

2. 任务创建完成后，物联网平台通过Topic：`/sys/{productKey}/{deviceName}/thing/job/notify`，将任务信息推送给设备。

消息格式如下：

```
{
  "id": "7542940",
  "version": "1.0",
    "params": {
        "task": {
            "taskId": "i5Ks6***pF010101",
            "status": "SENT",
            "jobDocument": {
              },
      "jobFile":{
         "signMethod":"Md5",
         "sign":"wssxff56dhdsd***",
         "fileUrl": "https://iotx-***.aliyuncs.com/***.zip"
      }
        }
    }
}
```

其中jobDocument下为任务规则文件内容。

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	请求业务参数。
task	Object	任务下的作业参数。
taskId	String	任务下作业的ID。为全局唯一标识符。
status	String	任务下作业的状态。 ○ SENT：已调度 ○ REMOVED：已删除 ○ CANCELLED：已取消

参数	类型	说明
jobDocument	Object	任务文档，描述任务执行规则。 说明 status为REMOVED或CANCELLED时，该字段值为空。
jobFile	Object	创建自定义任务时，上传的文件信息。 ○ signMethod：签名方法，目前支持 Md5 和 Sha256 。 ○ sign：签名，根据相应的签名方法生成的签名参数。 ○ fileUrl：任务文件的下载地址。 说明 status为REMOVED或CANCELLED时，该字段值为空。

3. 设备端根据自定义任务逻辑，实现规则内容。

   如果当前设备处于离线状态，无法接收任务信息，设备上线后，可通过Topic： /sys/{productKey}/{deviceName}/thing/job/get 先获取可执行的任务列表，然后获取一个可执行任务信息，来完成任务。

   获取任务列表的消息格式如下：

```
{
    "id": "123",
    "version": "1.0",
    "params": {
        "taskId": "$list"
    }
}
```

   获取任务信息的消息格式：

```
{
    "id": "123",
    "version": "1.0",
    "params": {
        "taskId": "i5Ks***F010101"
    }
}
```

   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	请求业务参数。
taskId	String	三种取值方式，可返回不同状态的任务信息。 ○ 任务下作业的ID：返回作业ID对应任务的详细信息。 ○ $next ：返回一个可执行任务的信息。 ○ $list ：返回可执行的任务列表，默认最多返回10个。

   物联网平台收到请求后，通过响应Topic： /sys/{productKey}/{deviceName}/thing/job/get_reply ，向设备端返回结果。

   返回任务列表数据格式如下：

```
{
  "id": "1234",
  "code": 200,
  "data": {
    "taskId": "$list",
    "task":[
          {
          "taskId": "i5Ks***",
          "status": "IN_PROGRESS"
          },
          {
          "taskId": "i61s***",
          "status": "IN_PROGRESS"
          }
        ]
    }
}
```

   返回任务信息数据格式如下：

```
{
  "id": "1234",
  "code": 200,
  "data": {
    "taskId": "i5Ks***F010101",
    "task":{
          "taskId": "i5Ks***F010101",
          "status": "IN_PROGRESS",
          "jobDocument": {
       },
         "jobFile":{
              "signMethod":"Md5",
              "sign":"wssxff56dhdsd***",
              "fileUrl": "https://iotx-***.aliyuncs.com/***.zip"
      }
     }
    }
}
```

4. 任务进行过程中，设备端通过Topic：`/sys/{productKey}/{deviceName}/thing/job/update`，向物联网平台上报任务进度。

   消息格式如下：

```
{
    "id": "123",
    "version": "1.0",
    "params": {
        "taskId": "i5Ks***F010101",
        "status": "IN_PROGRESS",
        "statusDetails": {
            "key": "value"
        },
        "progress": 50
    }
}
```

   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	请求业务参数。
taskId	String	任务下作业的ID。为全局唯一标识符。
status	String	任务下作业的状态。可取值： ○ SUCCEEDED：成功 ○ FAILED：失败 ○ IN_PROGRESS：执行中 ○ REJECTED：已拒绝
statusDetails	Object	用户自定义的状态详情，内容可自定义。可在物联网平台控制台的**设备管理 > 任务 > 任务详情**页面查看。
progress	Integer	任务下作业执行进度的百分数。

5. 在物联网平台的**设备管理 > 任务**页面，查看已创建任务及当前状态。

   > **注意** 状态为**已超时**的任务，不可再被调度执行。
   >
   > 从任务创建完成开始计时，如果任务下作业未在7天内全部执行完成，任务状态显示为**已超时**。

   您可根据实际场景需要，执行以下操作：

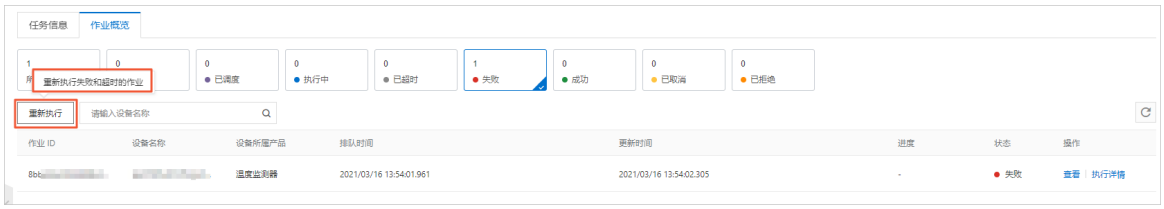
   - 在任务列表中，取消**执行中**状态的任务。
   - 单击目标任务对应的**查看**，进入**任务详情**页面。
     - 在**任务信息**页签，修改任务描述和作业配置，下载设备任务文件。

- 在**作业概览**页签，查看任务下各状态的作业统计。

  您可单击目标设备的**查看**，在**设备详情**页面，单击**任务**页签，查看该设备下的所有任务列表；单击**日志服务**的**前往查看**，在**云端运行日志**页签的**业务类型**列选择**云到设备消息**，查看设备任务相关日志。

  如果作业未执行成功，单击**执行详情**，可查看失败原因。

  如果作业执行已超时或失败，单击**已超时**或**失败**的状态按钮，可查看对应状态的作业列表。您可单击列表上方的**重新执行**，重新执行当前任务下所有已超时和失败的作业。



设备的自定义任务示例

7.3. 添加属性设置任务

如果您需要同时设置多个设备的属性值，可使用物联网平台的设备批量属性设置任务。本文介绍该任务的创建方法、运行中Topic及其数据格式的说明，查看状态的具体操作。

前提条件

- 设备所属产品下已添加物模型属性，且多个设备中包含相同属性标识符。具体操作，请参见添加物模型。
- 已在设备端完成物模型属性设置能力开发。具体操作，请参见物模型的设置属性。

任务管理流程

1. 创建设备批量属性设置任务。
   i. 在物联网平台控制台，对应实例下的**设备管理 > 任务**页面，单击**创建任务**。
   ii. 在**创建任务**页面，单击以下参数名称右侧的帮助图标，根据页面提示，配置任务和作业，单击**完成**。

- 任务配置

参数	说明
任务名称	输入符合规则的任务名称。可自定义。
任务类型	选择类型：**设备批量属性设置任务**。
任务描述	输入该任务的用途等信息，便于您区分不同的任务。
目标设备，产品，或分组	选择可执行任务的设备。
下发给设备的任务执行规则	上传规则文件。仅支持 `.json` 格式文件，文件大小不能超过64 KB。 您可单击**下载模板**，获取规则文件模板。 例如设备所属产品下的物模型属性温度，**标识符**为temperature），**数据类型**为**double**，代码示例如下： `{ "params": { "temperature": 30.5 } }` ■ params：属性设置参数。其下可包含多个属性。 ■ temperature、30.5：属性标识符及对应值。可在物联网平台控制台，设备所属产品的**功能定义**中查看属性标识符。 如果是自定义（非默认）模块 `testFb` 下属性 `temperature`，则参数为 `"testFb:temperature":30.5`。

- 作业配置

参数	说明
作业执行速率配置	根据您的业务需要，设置每分钟作业推送数量。必选配置。
作业执行的超时配置	仅对自定义任务生效。此处无需配置。
作业开始调度时间	可选配置。 从当前设置操作的时间，开始计算时间。 设备任务创建成功后，先初始化，直至到达调度时间，才会开始调度执行。

2. 任务创建完成后，物联网平台通过调用SetDeviceProperty或SetDevicesProperty接口，下发设置属性指令到设备，然后设备向物联网平台返回响应结果。

   消息格式如下：

   其中，params数据是任务规则文件的params数据。

   设置设备属性的Alink协议，请参见设置设备属性。

3. 在物联网平台的**设备管理 > 任务**页面，查看已创建任务及当前状态。

   > **注意** 状态为**已超时**的任务，不可再被调度执行。
   >
   > 从任务创建完成开始计时，如果任务下作业未在7天内全部执行完成，任务状态显示为**已超时**。

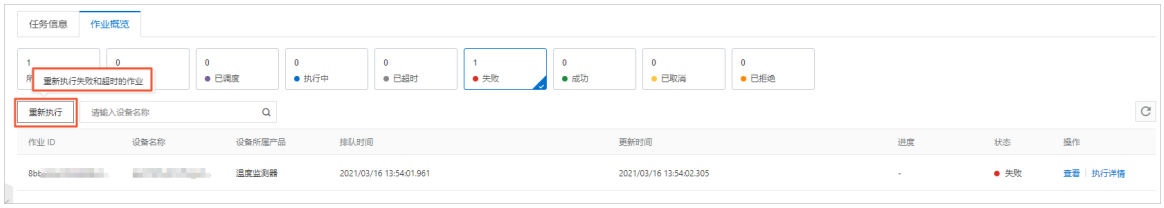
   您可根据实际场景需要，执行以下操作：

   - 在任务列表中，取消**执行中**状态的任务。
   - 单击目标任务对应的**查看**，进入**任务详情**页面。
     - 在**任务信息**页签，修改任务描述和作业配置，下载设备任务文件。
     - 在**作业概览**页签，查看任务下各状态的作业统计。

       您可单击目标设备的**查看**，在**设备详情**页面，单击**任务**页签，查看该设备下的所有任务列表；单击**日志服务**的**前往查看**，在**云端运行日志**页签的**业务类型**列选择**云到设备消息**，查看设备任务相关日志。

       如果作业未执行成功，单击**执行详情**，可查看失败原因。

       如果作业执行已超时或失败，单击**已超时**或**失败**的状态按钮，可查看对应状态的作业列表。您可单击列表上方的**重新执行**，重新执行当前任务下所有已超时和失败的作业。



7.4. 添加服务调用任务

如果您需要同时调用多个设备的异步服务，可使用物联网平台的设备批量服务调用任务。本文介绍该任务的创建方法、运行中Topic及其数据格式的说明，查看状态的具体操作。

前提条件

- 设备所属产品下已添加物模型服务（异步调用），且多个设备中包含相同服务标识符。具体操作，请参见添加物模型。
- 已在设备端完成物模型服务调用能力开发。具体操作，请参见物模型的调用服务。

任务管理流程

1. 创建设备批量服务调用任务。
   i. 在物联网平台控制台，对应实例下的**设备管理 > 任务**页面，单击**创建任务**。

ii. 在**创建任务**页面，单击以下参数名称右侧的帮助图标，根据页面提示，配置任务和作业，单击**完成**。

- 任务配置

参数	说明
任务名称	输入符合规则的任务名称。可自定义。
任务类型	选择类型：**设备批量服务调用任务**。
任务描述	输入该任务的用途等信息，便于您区分不同的任务。
目标设备，产品，或分组	选择可执行任务的设备。
下发给设备的任务执行规则	上传规则文件。仅支持 `.json` 格式文件，文件大小不能超过64 KB。 您可单击**下载模板**，获取规则文件模板。 例如设备所属产品下物模型服务数据计算，**标识符**为Operation_Service，**调用方式**为**异步**，输入参数为数值A（NumberA）和数值B（NumberB），参数**数据类型**均为**int32**，代码示例如下： `{ "serviceIdentifier": "Operation_Service", "params": { "NumberA": 32, "NumberB": 56 } }` ■ serviceIdentifier：服务唯一标识符。可在物联网平台控制台中，设备所属产品的**功能定义**中查看。 如果是自定义（非默认）模块 `testFb` 下服务 `Operation_Service`，则参数为 `"serviceIdentifier": "testFb:Operation_Service",`。 ■ params：服务的入参，可包含多个入参。

- 作业配置

参数	说明
作业执行速率配置	根据您的业务需要，设置每分钟作业推送数量。必选配置。
作业执行的超时配置	仅对自定义任务生效。此处无需配置。
作业开始调度时间	可选配置。 从当前设置操作的时间，开始计算时间。 设备任务创建成功后，先初始化，直至到达调度时间，才会开始调度执行。

2. 任务创建完成后，物联网平台通过InvokeThingService或InvokeThingsService接口调用服务，采用异步方式下行推送请求，然后设备也采用异步方式向物联网平台返回响应结果。

   消息格式如下：

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

   其中，params数据是任务规则文件中params数据；`{tsl.service.identifier}`是任务规则文件中serviceIdentifier的值。上文规则文件对应消息格式如下：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "NumberA": 32,
    "NumberB": 56
  },
  "method": "thing.service.Operation_Service"
}
```

   设备服务异步调用的Alink协议，请参见设备服务调用（异步调用）。

3. 在物联网平台的**设备管理 > 任务**页面，查看已创建任务及当前状态。

   > **注意** 状态为**已超时**的任务，不可再被调度执行。
   >
   > 从任务创建完成开始计时，如果任务下作业未在7天内全部执行完成，任务状态显示为**已超时**。

   您可根据实际场景需要，执行以下操作：

   - 在任务列表中，取消**执行中**状态的任务。
   - 单击目标任务对应的**查看**，进入**任务详情**页面。
     - 在**任务信息**页签，修改任务描述和作业配置，下载设备任务文件。

- 在**作业概览**页签，查看任务下各状态的作业统计。

  您可单击目标设备的**查看**，在**设备详情**页面，单击**任务**页签，查看该设备下的所有任务列表；单击**日志服务**的**前往查看**，在**云端运行日志**页签的**业务类型**列选择**云到设备消息**，查看设备任务相关日志。

  如果作业未执行成功，单击**执行详情**，可查看失败原因。

  如果作业执行已超时或失败，单击**已超时**或**失败**的状态按钮，可查看对应状态的作业列表。您可单击列表上方的**重新执行**，重新执行当前任务下所有已超时和失败的作业。

7.5. 添加消息批量下发任务

如果您需要同时向多个设备发送自定义Topic消息，可使用物联网平台的消息批量下发任务。本文介绍该任务的创建方法、运行中Topic及其数据格式的说明，查看状态的具体操作。

前提条件

- 已在物联网平台为设备添加具有**订阅**或**订阅和发布**权限的自定义Topic。具体操作，请参见自定义Topic。
- 已在设备端完成自定义Topic订阅，且未取消订阅。具体操作，请参见Link SDK的订阅Topic。

任务管理流程

1. 创建Pub服务任务。
   i. 在物联网平台控制台，对应实例下的**设备管理 > 任务**页面，单击**创建任务**。
   ii. 在**创建任务**页面，单击以下参数名称右侧的帮助图标，根据页面提示，配置任务和作业，单击**完成**。

- 任务配置

参数	说明
任务名称	输入符合规则的任务名称。
任务类型	选择类型：**Pub批量消息推送任务**。
任务描述	输入该任务的用途等信息，便于您区分不同的任务。
目标设备、产品或分组	选择可执行任务的设备。
下发给设备的任务执行规则	上传规则文件。仅支持 `.json` 格式文件，文件大小不能超过64 KB。 您可单击**下载模板**，获取规则文件模板。 例如多个设备的自定义Topic为： `/${productKey}/${deviceName}/user/get` ，代码示例如下： `{ "topicShortName": "get", "messageContent": "eyJ0ZXN0IjoidGFzayBiYXRjaHBlYiBicm9hZGNhc3QifQ==" }` ■ topicShortName：用于定义完整的自定义Topic： `/${productKey}/${deviceName}/user/${topicShortName}` 。即Topic： `/${productKey}/${deviceName}/user/get` 的topicShortName值为 `get` 。 ■ messageContent：要发送的消息主体。您需要将消息原文转换成二进制数据，并进行Base64编码，从而生成消息主体。

- 作业配置

参数	说明
作业执行推送配置	■ **每分钟作业执行数量**：根据您的业务需要，设置每分钟作业推送数量。 ■ **推送消息类型**：仅对自定义任务和Pub批量消息推送任务生效。 可选： ■ **QoS0**：最多发送一次。 ■ **QoS1**：最少发送一次。如果QoS1消息未接收到PUBACK消息，会在设备重连时，重新推送给设备。
作业执行的超时配置	可选配置。不设置表示不会超时。仅对自定义任务生效。 从设备任务进入IN_PROGRESS状态，开始计算时间。如果超过了超时时间，任务下作业仍未执行完成，作业状态将被自动设置为TIMED_OUT，作业停止执行。
作业开始调度时间	可选配置。 从当前设置操作的时间，开始计算时间。 设备任务创建成功后，先初始化，直至到达调度时间，才会开始调度执行。

2. 任务创建完成后，物联网平台通过调用Pub接口，向多个设备发送消息，然后设备向物联网平台返回响应结果。
   自定义消息Topic为 `/${productKey}/${deviceName}/user/${TopicShortName}` ，数据格式由用户自定义。
3. 在物联网平台的**设备管理 > 任务**页面，查看已创建任务及当前状态。

> **注意** 状态为**已超时**的任务，不可再被调度执行。
> 从任务创建完成开始计时，如果任务下作业未在7天内全部执行完成，任务状态显示为**已超时**。

您可根据实际场景需要，执行以下操作：

- 在任务列表中，取消**执行中**状态的任务。
- 单击目标任务对应的**查看**，进入**任务详情**页面。
  - 在**任务信息**页签，修改任务描述和作业配置，下载设备任务文件。
  - 在**作业概览**页签，查看任务下各状态的作业统计。

    您可单击目标设备的**查看**，在**设备详情**页面，单击**任务**页签，查看该设备下的所有任务列表；单击**日志服务**的**前往查看**，在**云端运行日志**页签的**业务类型**列选择**云到设备消息**，查看设备任务相关日志。

    如果作业未执行成功，单击**执行详情**，可查看失败原因。

    如果作业执行已超时或失败，单击**已超时**或**失败**的状态按钮，可查看对应状态的作业列表。您可单击列表上方的**重新执行**，重新执行当前任务下所有已超时和失败的作业。

8.设备影子

8.1. 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，可以直接获取物联网平台指令；设备离线后，再次上线可以主动拉取物联网平台指令。

什么是设备影子

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

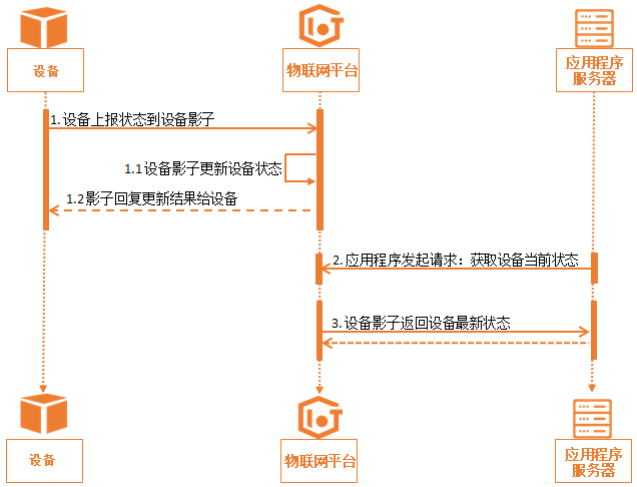
- 应用程序请求获取设备状态。

  场景描述：

  - 设备网络不稳定，设备频繁上下线，无法正常响应应用程序的请求。
  - 设备网络稳定，同时响应多个应用程序的请求，即使响应的结果一样，设备本身处理能力有限，也会无法负载多次请求。

  使用设备影子机制，设备状态变更，只需同步状态给设备影子一次，应用程序请求获取设备状态，不论应用程序请求数量，和设备是否联网在线，都可从设备影子中获取设备当前状态，实现应用程序与设备解耦。

  应用程序获取设备影子中状态的流程图如下，其中数据流转过程，请参见设备主动上报状态。

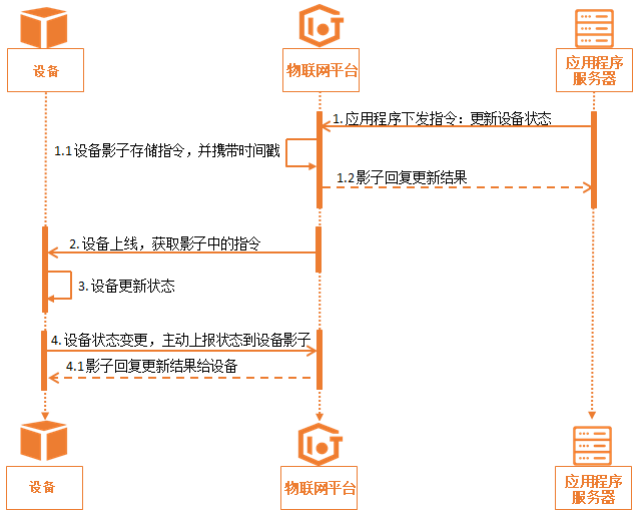


- 应用程序下发指令给设备，变更设备状态。

  场景描述：设备处于下线状态，或设备网络不稳定，设备频繁上下线，应用程序发送控制指令给设备，设备不在线，指令就会发送失败。

  使用设备影子机制，可以将应用程序下发的指令，携带时间戳存储到设备影子中。设备再上线时，获取设备影子中指令，并根据时间戳确定是否执行。

  应用程序更新设备状态的流程图如下，其中数据流转过程，请参见应用程序改变设备状态、设备主动获取影子内容、设备主动删除影子属性。



查看与更新设备影子

您需在设备端完成设备影子功能开发，具体内容，请参见设备影子。

完成设备开发和接入后，您可以在物联网平台控制台，查看设备影子信息，更新设备影子状态。

1. 登录物联网平台控制台。

2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 在中国地域，目前仅华东2（上海）地域开通了公共实例服务。

3. 在左侧导航栏，选择**设备管理 > 设备**。
4. 单击对应设备的**查看**按钮，进入**设备详情**页面。
5. 单击**设备影子**，页面显示设备上报的影子状态。

6. 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，主动从物联网平台拉取最新期望状态。

相关API

GetDeviceShadow：获取设备影子。

UpdateDeviceShadow：更新设备影子。

8.2. 设备影子数据流

设备影子数据通过Topic进行流转，主要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。本文介绍设备影子Topic及其数据格式。

设备影子Topic

物联网平台已为每个设备预定义了两个Topic，用于实现设备影子数据流转。

- `/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

本文以灯泡设备为例，说明设备、设备影子以及应用程序之间的通信，主要介绍设备主动上报状态、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。

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。

设备端开发设备影子能力的方法，请参见设备影子。

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
}
```

   上报参数说明

参数	说明
method	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，method为必填字段，设置为*update*。
state	表示设备发送给设备影子的状态信息。 reported为必填字段，状态信息会同步更新到设备影子的reported部分。
version	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version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
}
```

错误码说明

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

应用程序改变设备状态

应用程序通过调用云端API UpdateDeviceShadow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，下发消息更改灯泡状态，例如需将灯泡的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. 设备影子会返回响应结果给设备，发送消息到设备订阅的Topic `/shadow/get/a1PbRCF****/lightbulb` 中。详细说明，请参见设备上报状态的步骤3。
6. 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/a1PbRCF****/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
}
```

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。

1. 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
}
```

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
}
```

8.3. 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JSON属性说明所示。

JSON属性说明

属性	描述

属性	描述
desired	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desired部分。 应用程序向desired部分写入数据，更新事物的状态，而无需直接连接到该设备。
reported	设备的报告状态。设备可以在reported部分写入数据，报告其最新状态。 应用程序可以通过读取该参数值，获取设备的状态。 JSON文档中也可以不包含reported部分，没有reported部分的文档同样为有效影子JSON文档。
metadata	当用户更新设备状态文档后，设备影子服务会自动更新metadata的值。 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。
timestamp	影子文档的最新更新时间。
version	用户主动更新版本号时，设备影子会检查请求中的version值是否大于当前版本号。 如果大于当前版本号，则更新设备影子，并将version值更新到请求的版本中，反之则会拒绝更新设备影子。 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。 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` 。

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"
        ]
    }
}
```

9.NTP服务

物联网平台提供NTP服务，为资源受限的嵌入式设备，解决无法实时地获取服务端时间的问题。

原理介绍

物联网平台的NTP服务，借鉴NTP协议原理，将物联网平台作为NTP服务器。高精准度的时间校正流程如下：

1. 设备端通过指定Topic向物联网平台发送消息，会携带发送时间deviceSendTime。
2. 物联网平台接收设备端消息后，回复消息中，会增加接收消息的时间serverRecvTime和回复消息的时间serverSendTime。
3. 设备端接收到物联网平台回复，会根据本地时间，给出接收回复的时间deviceRecvTime。
4. 根据以上出现的4个时间，计算设备端与物联网平台的时间差，得出设备端获取到的，服务端当前的精确时间Time。

> **注意** 仅当设备端成功接入物联网平台后，才能通过NTP服务进行时间校准。
>
> 若嵌入式设备未接入物联网平台，设备上电后，无法通过NTP服务进行时间校准，则TSL建连过程中，证书时间校验会失败。

通信的Topic

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

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

其中，*${YourProductKey}*和*${YourDeviceName}*是设备证书中的ProductKey和DeviceName，您可在物联网平台控制台的**设备详情**页面获取。

设备端接入说明

目前仅C语言的设备端Link SDK支持配置NTP服务功能。请访问C Link SDK获取，下载开发代码Demo。

设备端配置NTP服务的流程和示例，请参见NTP服务。

NTP服务使用流程，及其Topic说明如下：

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，携带设备当前的时间戳，单位为毫秒。示例如下：

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

> **说明**
> - 时间戳数字，支持Long（默认）和String类型。
> - NTP服务目前仅支持QoS=0的消息。

3. 设备端通过Topic `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`，收到物联网平台回复的消息，包含以下信息：

```
{
    "deviceSendTime":"1571724098000",
    "serverRecvTime":"1571724098110",
    "serverSendTime":"1571724098115",
}
```

4. 设备端计算出服务端当前精确的Unix时间。

   设备端收到服务端的时间记为${deviceRecvTime}，则设备上的精确时间为：`(${serverRecvTime}+${serverSendTime}+${deviceRecvTime}-${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毫秒，服务端时间是1571724098100毫秒，链路延时是10毫秒，服务端从接收到发送间隔为5毫秒。

操作	设备端时间（毫秒）	服务端时间（毫秒）
设备端发送	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)`**毫秒**`÷2=1571724098125`**毫秒**。

如果直接采用物联网平台返回的时间戳，只能得到时间1571724098115毫秒，与服务端上的时间会有10毫秒的链路延时误差。

10.设备端接收的错误码

本文介绍物联网平台可能返回给设备端的错误码及说明。

公共错误码

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Alink协议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。	在物联网平台控制台，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。	在物联网平台控制台的设备管理中，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。	在物联网平台控制台的查询产品信息，核对ProductKey。
6204	设备已被禁用，不能对设备进行操作。	在物联网平台控制台的设备管理中，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method。	在物联网平台控制台的**日志服务**中，或设备本地日志中，检查设备上报的数据中是否有method参数。
6760	系统异常。	提交工单。

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。	在物联网平台控制台，产品的**数据解析**页签下，确认脚本已提交。 说明 未提交的脚本不能被调用。
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protocolToRawData和rawDataToProtocol这两个服务。如果缺失，会报错。	在物联网平台控制台，产品的**数据解析**页签下，查询脚本内容中protocolToRawData和rawDataToProtocol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protocolToRawData和rawDataToProtocol这两个服务。protocolToRawData返回byte[]数组， rawDataToProtocol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。	在物联网平台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。

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。	请参见设备属性、事件、服务，查看对应数据格式，并按格式要求上报数据。
6300	method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method参数。	在物联网平台控制台的**日志服务**，或者设备的本地日志中，查看上报数据中是否有method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。	在物联网平台控制台，产品的**功能定义**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。	在物联网平台控制台，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。
6306	TSL校验时，发现： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的参数取值范围不符合功能定义时设置的参数范围。	

错误码	原因	排查
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： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的参数取值范围不符合功能定义时设置的参数范围。	在物联网平台控制台，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。
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： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的参数取值范围不符合功能定义时设置的参数范围。	
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： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的字符类型的参数长度超过限制。	
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： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的参数取值范围不符合功能定义时设置的参数范围。	
6304	TSL校验时，发现传入的参数在结构体中不存在。	在物联网平台控制台，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。	
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： • 传入的参数类型和TSL中定义的类型不一致。 • 传入的字符类型不是UTC时间戳的字符格式。	
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： • 传入的参数类型和TSL中定义的类型不一致。 • 结构体类型中参数的个数和TSL中定义不一致。	
6320	查询设备的TSL时，没有查询到设备的属性信息。	在物联网平台控制台，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type，specs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： • 传入的数组类型的参数不符合TSL定义。 • 数组中参数个数超过了TSL中定义的最大个数。	• 在物联网平台控制台的产品详情中，查看设备所属产品的功能定义，检查对应数组的定义。 • 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。
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。	请参见设备属性、事件、服务，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。	在物联网平台控制台，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。
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`
  - 响应Topic： `/sys/${productKey}/${deviceName}/thing/sub/register_reply`

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。	在物联网平台控制台，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。	在物联网平台控制台，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。	在物联网平台控制台，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。	请使用设备身份注册中signMethod支持的签名方法。
6600	签名校验失败。	请按照设备身份注册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。	在物联网平台控制台的设备管理中，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`
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/topo/add_reply`

  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。	请按照管理拓扑关系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。	请参见使用限制，查看相关限制，并在物联网平台控制台，该网关设备的**子设备管理**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`
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/topo/delete_reply`

  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。	在物联网平台控制台**设备管理**，网关设备的**设备详情**页**子设备管理**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`
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/topo/get_reply`

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

  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

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

  错误码：460、429、6100、6204、6287、6401、500、9241、9240

- 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

- 子设备被迫下线

  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

- 网关代理子设备批量上报消息失败

  接收消息的网关Topic：`/sys/${productKey}/${deviceName}/proxy/batch_post_reply`

  错误码：9242

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

错误码	原因	排查
427	设备证书信息被其他设备使用，使设备被迫下线。 物联网平台仅以设备证书信息（productKey、deviceName、deviceSecret）来判断设备。 • 多个设备上烧录了相同的设备证书。 • 设备端网络或电源不稳定，发生了瞬间断网或断电重连。这种情况下，设备与物联网平台是连接的，不影响设备使用。	在物联网平台控制台设备的**设备详情**页，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。 限制说明，请参见使用限制的网关与子设备。	请检查网关设备下的子设备数量。
521	设备已被删除。	在物联网平台控制台的**设备管理**页搜索设备，确认设备是否已被删除。
522	设备已被禁用。	在物联网平台控制台的**设备管理**页查看设备状态。
520	子设备会话错误。 • 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。 • 子设备会话在线，但是并不是通过当前网关会话上线的。	
6287	按照产品或者设备的密钥校验签名失败。	请参见子设备上下线中的签名方法计算签名，并校验。
1914	单个批量上下线请求中，包含的子设备数量超过限制（5个）。	检查子设备批量上下线数量是否超过阈值。
1913	网关离线导致子设备被云端自动离线。	根据日志服务查询网关离线原因。
9242	一个批量上报消息请求中，网关代理子设备批量上报的消息条数超出限制（50条）。	检查子设备批量上报的消息条数是否超过阈值。
9241	网关代理子设备上线请求中，在线设备不能修改连接的设备类型。 连接的设备类型说明，请参见MQTT客户端直连中的conntype。	检查网关设备和子设备是否已在线。
9240	一个批量上下线请求中，状态相关网关设备下子设备数量超出限制（10,000）。	检查子设备批量上下线数量是否超过阈值。

设备属性、事件、服务相关错误码

- 设备上报属性
  - 透传数据格式：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`
  - Alink数据格式：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/event/property/post`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/event/property/post_reply`

  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。	在物联网平台控制台，**监控运维 > 日志服务**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
  - 透传数据格式：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`

- Alink格式数据：
  - 默认模块
    - 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`
  - 自定义模块：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:${tsl.event.identifier}/post`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:${tsl.event.identifier}/post_reply`

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
  - 透传数据格式：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`
  - Alink格式数据：
    - 请求Topic：`/sys/${productKey}/${deviceName}/thing/event/property/pack/post`
    - 响应Topic：`/sys/${productKey}/${deviceName}/thing/event/property/pack/post_reply`

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公共错误码章节。

错误码	原因	排查
6401	拓扑关系不存在。	在物联网平台控制台，网关设备的**子设备管理**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。	在物联网平台控制台，**监控运维 > 日志服务**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` 。
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/property/desired/get_reply` 。

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。	在物联网平台控制台，**监控运维 > 日志服务**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` 。
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/property/desired/delete_reply` 。

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`
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/update_reply`

错误码：460、6100

- 设备删除标签信息
  - 请求Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/delete`
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/delete_reply`

错误码：460、500

获取TSL模板相关错误码

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

- 响应Topic：`/sys/${productKey}/${deviceName}/thing/dsltemplate/get_reply`

错误码：460、5159、5160、5161

设备请求升级包信息相关错误码

请求Topic：`/ota/device/request/${productKey}/${deviceName}`

> **说明** 设备请求升级包信息的Topic与返回数据Topic相同。

错误码：429、9112、500

以下为设备请求升级包信息的特有错误码，其他错误码，请参见公共错误码章节。

错误码	原因	排查
9112	未查询到指定的设备信息。	在物联网平台控制台的设备管理中，确认设备信息是否正确。

设备请求配置信息相关错误码

- 请求Topic：`/sys/${productKey}/${deviceName}/thing/config/get`
- 响应Topic：`/sys/${productKey}/${deviceName}/thing/config/get_reply`

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。	在物联网平台控制台，**监控运维 > 远程配置**中，打开该产品的远程配置开关。
6710	未查询到远程配置信息。	在物联网平台控制台，**监控运维 > 远程配置**中，查看是否为该产品编辑了远程配置文件。

设备自定义任务相关错误码

- 设备获取自定义任务详情
  - 请求Topic：`/sys/${productKey}/${deviceName}/thing/job/get`。
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/job/get_reply`。

  错误码：71012、71034、71035
- 设备更新任务下作业状态
  - 请求Topic：`/sys/${productKey}/${deviceName}/thing/job/update`。
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/job/update_reply`。

  错误码：71018、71019、71034、71035

错误码	原因	排查
71012	设备获取任务详情中，发起请求的数据格式不是JSON格式。	请参见获取设备任务详情，查看正确的数据格式，并按照数据格式上报请求。
71018	任务已处于以下任一状态： • 已完成 • 已超时 • 取消中 • 已取消 • 删除中	在物联网平台控制台的**设备管理 > 任务**的任务列表中，确认目标任务的状态。
71019	任务下作业的ID不存在。	在物联网平台控制台的**设备管理 > 任务**页面，进入目标任务的**任务详情 > 作业概览**页签，确认作业ID是否存在。
71034	任务下作业ID不属于当前设备。	在物联网平台控制台的**设备管理 > 任务**页面，进入目标任务的**任务详情 > 作业概览**页签，确认当前设备是否包含该作业ID。

71035	• 获取任务详情时，请求数据中任务ID不是自定义任务的ID。
	• 任务下作业已处于最终状态：
	  ○ REMOVED：已删除
	  ○ TIMED_OUT：已超时
	  ○ FAILED：失败
	  ○ SUCCEEDED：成功
	  ○ CANCELLED：已取消
	  ○ REJECTED：已拒绝
		在物联网平台控制台的**设备管理 > 任务**页面，进入目标任务的**任务详情**页面：
		• 在**任务信息**页签，确认当前任务的**任务类型**是否为**自定义任务**。
		• 在**作业概览**页签，确认目标作业的状态。

设备上传文件相关错误码

上传文件的数据格式和参数说明，请参见文件上传。

- 设备请求上传文件
  - 请求Topic：`/sys/${productKey}/${deviceName}/thing/file/upload/mqtt/init`。

- 响应Topic：`/sys/${productKey}/${deviceName}/thing/file/upload/mqtt/init_reply`。

错误码：78117、78123、78124、78125

- 设备上传文件分片
  - 请求Topic：`/sys/${productKey}/${deviceName}/thing/file/upload/mqtt/send`。
  - 响应Topic：`/sys/${productKey}/${deviceName}/thing/file/upload/mqtt/send_reply`。

错误码：78118、78119、78120、78121、78122、78124、78125、78126

错误码	原因	排查
78117	设备请求上传文件的大小超出最大限制（16 MB）。	确认待上传文件大小是否符合要求。
78118	设备上传文件分片的大小超出最大限制（128 KB）。	确认上传文件的分片大小是否符合要求。详细信息，请参见上传文件。
78120	设备上传文件分片的大小不满足最小限制（256 B）。	
78119	设备上传后续文件分片时的offset值，与实际已上传到物联网平台云端的文件大小不一致。	确认设备上传文件分片时的offset的值是否正确。
78121	设备上传文件分片时，当前文件分片的文件完整性校验失败。	根据CRC算法校验的错误信息，排查原因。
78122	设备上传文件的任务不存在。	确认上传文件的标识ID是否正确或存在。
78123	同名文件已存在。	确认设备端设置的文件处理策略，是否支持上传同名文件。详细信息，请参见上传文件。
78124	文件上传任务已经完成。 注意 针对已完成上传的文件，设备端不能再进行上传文件分片和取消文件上传任务的操作。	确认物联网平台是否已存在上传的文件，具体操作，请参见查看文件。
78125	文件完整性校验失败。	确认物联网平台云端计算的文件完整性校验值，是否与设备上传文件时的完整性校验值相同。
78126	设备端并行上传了文件的同一分片数据。	确认设备端是否收到当前分片数据的响应消息后，再发送的下一个分片数据。

Paho-MQTT接入相关错误码

设备使用MQTT开源库Paho接入物联网平台后，连接断开时，可以看到MQTT断开相关的错误日志和相关的错误码。具体信息，请参见错误码。