阿里云
物联网平台

用户指南

文档版本：20190921

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 产品与设备

本章节将介绍如何使用控制台创建、管理产品与设备。

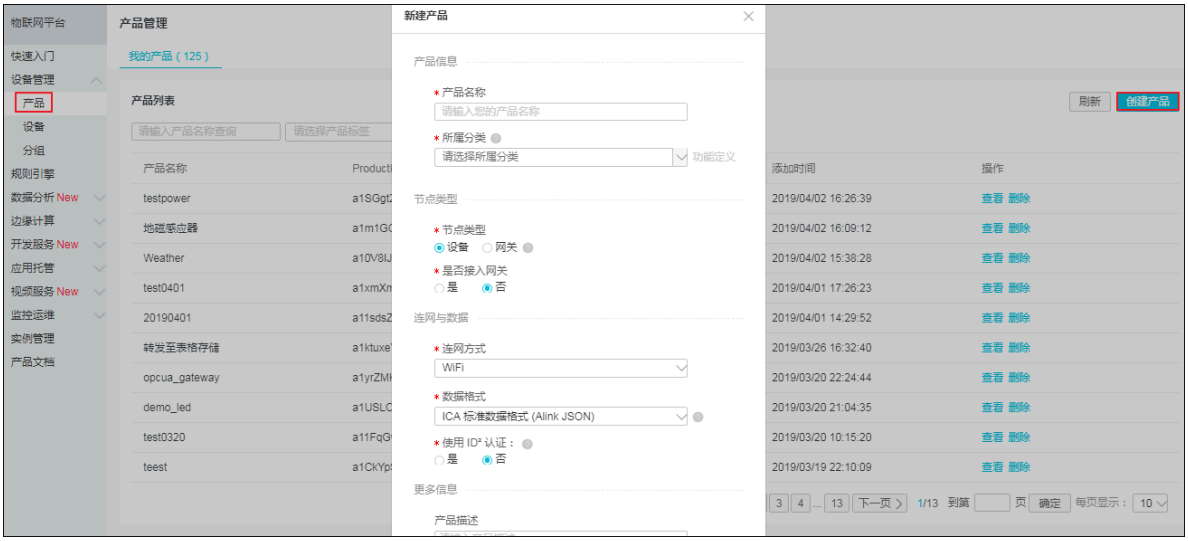
1.1 创建产品

使用物联网平台的第一步：在控制台创建产品。产品是设备的集合，通常是一组具有相同功能定义的设备集合。例如：产品指同一个型号的产品，设备就是该型号下的某个设备。

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品，单击创建产品。

3. 按照页面提示填写信息，然后单击完成。




产品信息
* 产品名称
测试的灯
* 所属分类
自定义品类
功能定义
节点类型
* 节点类型
设备
网关
* 是否接入网关
是
否
连网与数据
* 连网方式
WiFi
数据格式
ICA 标准数据格式 (Alink JSON)
* 使用 ID² 认证：
是
否
更多信息
产品描述
快速入门测试用灯
8/100

页面参数设置如下：

参数	描述
产品名称	为产品命名。产品名称在账号内具有唯一性。例如，可以填写为产品型号。支持中文、英文字母、数字和下划线，长度限制4~30，一个中文汉字算2位。
所属分类	选择品类，为该产品定义物模型。 可选择为： · 自定义品类：需根据实际需要，定义产品功能。 · 任一既有功能模板。 选择任一物联网平台预定义的品类，快速完成产品的功能定义。选择产品模板后，您可以在该模板基础上，编辑、修改、新增功能。 阿里云物联网平台提供多种品类，并为对应产品预定义了相关功能。如您选择智能城市 > 能源管理 > 电表设备类型模板中，已预定义用电量、电压、电流、总累积量等电表标准功能。
节点类型	产品下设备的类型。 · 设备：指不能挂载子设备的设备。这种设备可以直连物联网平台，也可以作为网关的子设备连接物联网平台 · 网关：指可以挂载子设备的直连设备。网关具有子设备管理模块，可以维持子设备的拓扑关系，并且将拓扑关系同步到云端。 网关与子设备的关系，请参见#unique_6。
是否接入网关	当节点类型为设备时出现的参数。表示该产品下的设备是否会接入网关设备，成为网关设备的子设备。 · 是：接入网关。需在连网与数据下，选择接入网关协议。 · 否：不接入网关。需在连网与数据下，选择连网方式。
接入网关协议	当是否接入网关选择为是时出现的参数。表示该产品下设备作为子设备与网关的通讯协议类型。 · 自定义：表示子设备和网关之间是其它标准或私有协议。 · Modbus：表示子设备和网关之间的通讯协议是 Modbus。 · OPC UA：表示子设备和网关之间的通讯协议是 OPC UA。 · ZigBee：表示子设备和网关之间的通讯协议是 ZigBee。 · BLE：表示子设备和网关之间的通讯协议是 BLE。

参数	描述
连网方式	当是否接入网关选择为否时出现的参数。设备连网方式： · WiFi · 蜂窝（2G/3G/4G） · 以太网 · LoRaWAN · 其他
授权	首次选择连网方式为LoRaWAN时，需要按照界面提示，授权IoT使用AliyunIOTAccessingLinkWANRole访问LinkWAN服务。
入网凭证	当连网方式选择为LoRaWAN时，需提供入网凭证名称。 若无凭证，请单击创建凭证，进入阿里云物联网络管理平台，添加专用凭证，并为凭证授权用户。 使用凭证创建的产品，将作为一个节点分组，自动同步到物联网络管理平台的节点分组列表中。
数据格式	设备上下行的数据格式。 · ICA 标准数据格式（Alink JSON）：是物联网平台为开发者提供的设备与云端的数据交换协议，采用 JSON 格式。 · 透传/自定义：如果您希望使用自定义的串口数据格式，可以选择为透传/自定义。 您需在控制台提交解析脚本，将自定义格式的数据通过#unique_7转换为 Alink JSON 格式，才能与云端进行通信。 说明: 使用LoRaWAN接入网关的产品仅支持透传/自定义。
使用ID²认证	该产品下的设备是否使用ID²认证。 ID²认证提供设备与物联网平台的双向身份认证能力，通过建立轻量化的安全链路（iTLS）来保障数据的安全性。 说明: · 仅在产品创建时能够启用ID²认证。并且，产品创建成功后，不能更改是否使用ID²认证的状态。 · 选择使用ID²认证，需购买ID²服务。请参见IoT设备身份认证(ID²)用户手册。

参数	描述
产品描述	可输入文字，用来描述产品信息。字数限制为100。

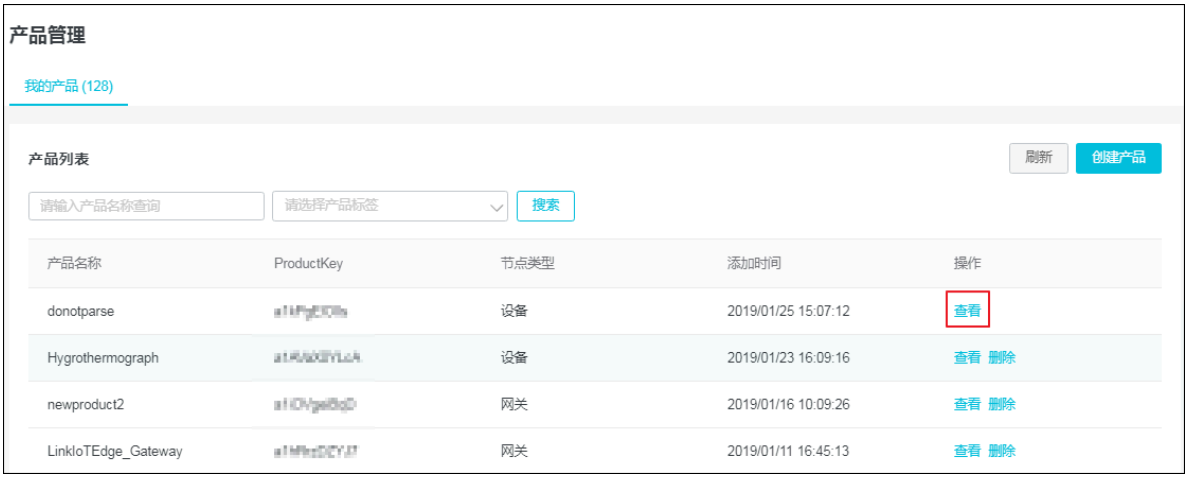
产品创建成功后，页面自动跳转回产品列表页面。

后续步骤

1. 在产品列表中，单击该产品的查看按钮，设置自定义Topic类、物模型（功能定义）、服务端订阅等。
2. 在设备页，创建设备。
3. 参考设备开发指南手册完成设备开发。
4. 在该产品的详情页中，单击发布按钮，发布产品。

发布前需确认：产品各项信息已设置完成、设备开发调试工作已完成、产品已具备上线发布条件。

产品发布后，产品状态变为已发布，此时产品信息仅支持查看，不支持修改和删除操作。



已发布的产品支持撤销发布。

1.2 创建设备

1.2.1 批量创建设备

产品指某一类设备，创建完产品后，需要为具体设备创建身份。您可以创建单个设备，也可以批量创建设备。本文为您讲述如何批量创建设备。

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，单击批量添加。
3. 选择产品。新创建的设备将继承该产品的功能和特性。

   说明:
   若该产品关联了其他平台，请确保您的账户下有足够的激活码用于创建设备。
4. 选择设备名称的添加方式。
   - 自动生成：无需为设备指定名称。填写设备数量后，系统将为每个设备自动生成由字母、数字随机组合成的DeviceName。
   - 批量上传：需要为所有设备指定名称。单击下载.csv模板下载表格模板，在模板中填写设备名称，然后将填好的表格上传至控制台。

     说明:
     填写模板时，请注意：
     - 设备名称长度为4-32个字符，可包含英文字母、数字和特殊字符，包括连接号（-）、下划线（_）、at符号（@）、点号（.）和英文冒号（:）。
     - 设备名称产品内具有唯一性。列表中的设备名称不可重复，且不可与该产品下已有设备的名称重复。
     - 一个文件中最多可包含1,000个名称。

- 文件大小不超过2 MB。

5. 单击确认，完成创建批量设备。

   若批量上传的设备名称列表中有不合法的名称，将出现错误提示。请单击下载不合法列表，查看不合法的设备名称。根据设备名称规范，修改设备名称，再重新上传文件。

6. 单击下载设备证书，下载本批次设备的设备证书。

预期结果

设备创建完成，获得设备证书。之后，您可以进入设备管理下的批次管理页签栏，

- 单击查看详情查看对应批次创建设备的详细信息。
- 单击下载CSV下载该批次设备的证书，方便产线统一烧录。

后续步骤

设备创建完成后，显示“待激活”状态。请参见设备端SDK开发文档开发设备端SDK，激活设备。

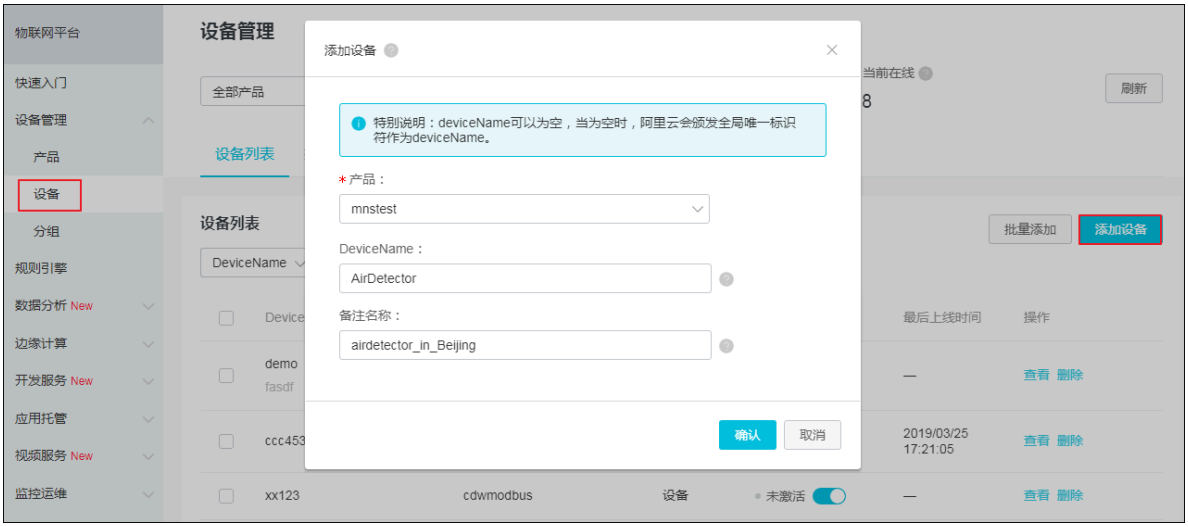
1.2.2 单个创建设备

产品指某一类设备，创建完产品后，需要为设备创建身份。您可以创建单个设备，也可以批量创建设备。本文为您讲述单个设备的创建。

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，再页面右侧添加设备。

3. 在添加设备对话框中，输入设备信息，单击确认。



参数	描述
产品	选择产品。新创建的设备将继承该产品定义好的功能和特性。 **说明:** 若该产品关联了其他平台，请确保您的账户下有足够的激活码用于创建设备。
DeviceName	设置设备名称。如果不填，系统将自动生成一个由数字和字母组成的设备名称。 · 设备名称在产品内具有唯一性。 · 设备名称长度为4-32个字符，可包含英文字母、数字和特殊字符，包括连接号（-）、下划线（_）、at符号（@）、点号（.）和英文冒号（:）。
备注名称	设置备注名称。备注名称长度为4-64个字符，可包含中文汉字、英文字母、数字和下划线（_）。一个中文汉字算2字符。

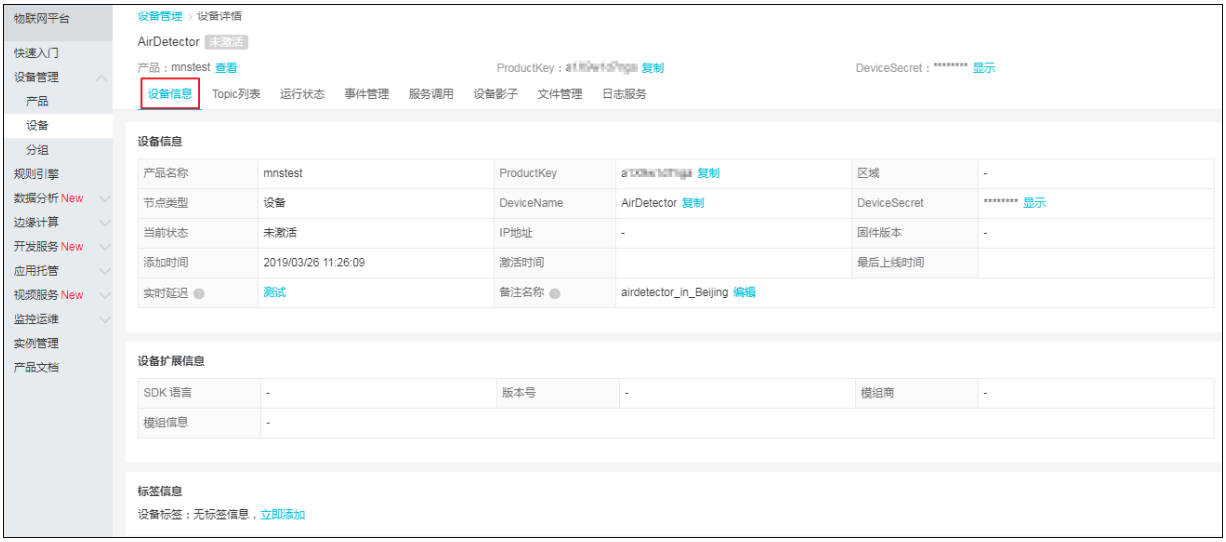
预期结果

设备创建成功后，将自动弹出查看设备证书弹框。您可以查看、复制设备证书信息。设备证书由设备 ProductKey、DeviceName、和 DeviceSecret组成，是设备与物联网平台进行通信的重要身份认证，建议您妥善保管。

参数	说明
ProductKey	设备所隶属产品的Key，即物联网平台为产品颁发的全局唯一标识符。
DeviceName	设备在产品内的唯一标识符。DeviceName与设备所属产品的ProductKey组合，作为设备标识，用来与物联网平台进行连接认证和通信。

参数	说明
DeviceSecret	物联网平台为设备颁发的设备秘钥，用于认证加密。需与DeviceName成对使用。

之后，您也可以在设备列表中，单击设备对应的查看按钮，进入设备详情页设备信息页签下，查看设备信息。



后续步骤

请参见设备端开发文档开发设备端SDK。

1.2.3 创建LoRa设备

物联网平台支持创建LoRa产品和设备。创建LoRa产品后，可以根据本文操作，创建LoRa设备。您可以单个创建LoRa设备，也可以批量操作。

前提条件

#unique_16

单个创建

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，单击添加设备。
3. 选择已创建的连网方式为LoRaWAN的产品。新创建的设备将继承该产品定义好的功能和特性。
4. 填入DevEUI和PIN Code，单击确认，完成设备创建。

> 说明:
>
> · DevEUI 是LoRa设备的唯一标识符，采用LoRaWAN协议标准规范，不可为空。请确保DevEUI产品下唯一，且已烧录到设备中。

· DevEUI和PIN Code一般印刷在设备外显标签上。

设备创建完成后，将自动弹出查看设备证书弹框。您可以查看、复制LoRa设备的证书信息，包括JoinEUI和DevEUI。

批量创建

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，单击批量添加。
3. 选择已创建的连网方式为LoRaWAN的产品。新创建的设备将继承该产品定义好的功能和特性。
4. 单击下载.csv模板下载表格模板，在模板中填写DevEUI和PIN Code，然后将填好的表格上传至控制台。

说明:

DevEUI 是LoRa设备的唯一标识符，采用LoRaWAN协议标准规范，不可为空。请确保DevEUI产品下唯一，且已烧录到设备中。

5. 单击确认。完成设备创建。

后续步骤

您可以参见物联网络管理平台文档搭建物联网所需的网络服务和开发设备端（即网关开发和节点开发）。

说明:

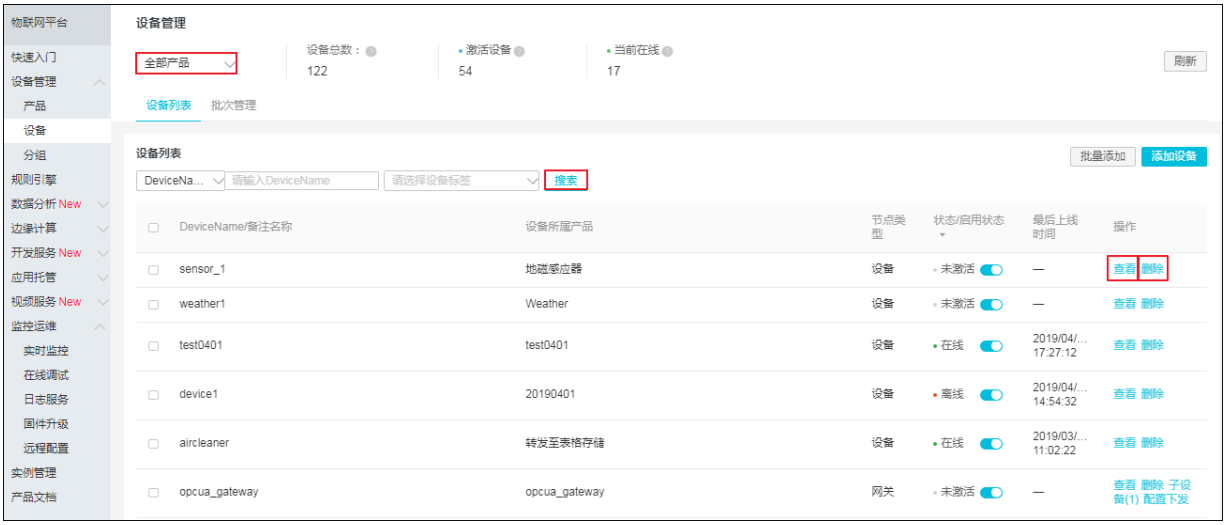
在物联网平台创建LoRa设备后，您无需再在物联网络管理平台录入节点（即设备）信息和配置数据流转。

1.2.4 管理设备

在物联网平台成功创建设备后，您可以在控制台管理、查看具体设备信息。

管理账号下的设备

在物联网平台左侧导航栏，单击设备管理 > 设备，进入设备管理页。



任务	操作步骤
查看具体产品下设备信息	在页面左上方选择某个产品。
搜索设备	输入设备名称、设备备注名称或设备标签搜索具体设备，支持模糊搜索。
查看具体设备信息	单击对应设备的查看按钮。

任务	操作步骤
删除某个设备	单击对应设备的删除按钮。 说明: 设备删除后，该设备证书信息将失效，该设备在物联网平台上的数据记录随之删除。

查看具体设备信息

在设备列表中，单击设备对应的查看按钮，进入设备详情页。

任务	操作步骤
激活设备	“未激活”状态表明设备未接入阿里云物联网平台。您可以参见#unique_11进行设备开发，将设备激活。
查看设备信息	查看设备基本信息，包括设备证书信息、固件信息、扩展信息、标签信息等内容。
查看设备数据	· 在运行状态页签下，查看设备上报的当前属性值，属性记录数据和期望属性值。 · 在事件管理页签下，查看设备上报的事件记录。 · 在服务调用页签下，查看设备服务调用记录。
查看设备日志	设备行为、上行消息、下行消息、物模型数据和QoS=1的设备消息内容，可以在日志服务页签下查看。日志具体说明，请参见日志服务文档。

1.3 物模型

1.3.1 什么是物模型

物模型指将物理空间中的实体数字化，并在云端构建该实体的数据模型。在物联网平台中，定义物模型即定义产品功能。完成功能定义后，系统将自动生成该产品的物模型。物模型描述产品是什么，能做什么，可以对外提供哪些服务。

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

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

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

物模型格式

您可以在产品的功能定义页面，单击查看物模型，查看JSON格式的 TSL。

物模型的JSON字段结构如下：

```
{
    "schema": "物的TSL描述schema",
    "link": "云端系统级uri,用来调用服务/订阅事件",
    "profile": {
        "productKey": "产品key"
    },
    "properties": [
        {
            "identifier": "属性唯一标识符(产品下唯一)",
            "name": "属性名称",
            "accessMode": "属性读写类型，只读(r)，读写(rw)",
            "required": "是否是标准功能的必选属性",
            "dataType": {
                "type": "属性类型: int(原生)，float(原生)，double(原生)，
text(原生)，date(String类型UTC毫秒)，bool(0或1的int类型)，enum(int类型)，
struct(结构体类型，可包含前面6种类型)，array(数组类型，支持int/double/float/
text)",
                "specs": {
```

```
                        "min": "参数最小值(int, float, double类型特有)",
                        "max": "参数最大值(int, float, double类型特有)",
                        "unit": "属性单位",
                        "unitName": "单位名称",
                        "size": "数组大小, 默认最大128(数组特有)",
                        "step": "步长, 字符串类型",
                        "item": {
                            "type": "数组元素的类型"
                        }
                    }
                }
            }
        ],
        "events": [
            {
                "identifier": "事件唯一标识符(产品下唯一, 其中post是默认生成的属性
上报事件)",
                "name": "事件名称",
                "desc": "事件描述",
                "type": "事件类型(info,alert,error)",
                "required": "是否是标准功能的必选事件",
                "callType": "async(异步调用), sync(同步调用)",
                "outputData": [
                    {
                        "identifier": "参数唯一标识符",
                        "name": "参数名称",
                        "dataType": {
                            "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                            "specs": {
                                "min": "参数最小值(int, float, double类型特
有)",
                                "max": "参数最大值(int, float, double类型特
有)",
                                "unit": "属性单位",
                                "unitName": "单位名称",
                                "size": "数组大小, 默认最大128(数组特有)",
                                "step": "步长, 字符串类型",
                                "item": {
                                    "type": "数组元素的类型"
                                }
                            }
                        }
                    }
                ],
                "method": "事件对应的方法名称(根据identifier生成)"
            }
        ],
        "services": [
            {
                "identifier": "服务唯一标识符(产品下唯一, 其中set/get是根据属性的
accessMode默认生成的服务)",
                "name": "服务名称",
                "desc": "服务描述",
                "required": "是否是标准功能的必选服务",
                "callType": "async(异步调用), sync(同步调用)",
                "inputData": [
                    {
                        "identifier": "入参唯一标识符",
                        "name": "入参名称",
                        "dataType": {
```

```
                            "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                            "specs": {
                                "min": "参数最小值(int, float, double类型特
有)",
                                "max": "参数最大值(int, float, double类型特
有)",
                                "unit": "属性单位",
                                "unitName": "单位名称",
                                "size": "数组大小, 默认最大128(数组特有)",
                                "step": "步长, 字符串类型",
                                "item": {
                                    "type": "数组元素的类型"
                                }
                            }
                        }
                    }
                ],
                "outputData": [
                    {
                        "identifier": "出参唯一标识符",
                        "name": "出参名称",
                        "dataType": {
                            "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                            "specs": {
                                "min": "参数最小值(int, float, double类型特
有)",
                                "max": "参数最大值(int, float, double类型特
有)",
                                "unit": "属性单位",
                                "unitName": "单位名称",
                                "size": "数组大小, 默认最大128(数组特有)",
                                "step": "步长, 字符串类型",
                                "item": {
                                    "type": "数组元素的类型(数组特有)"
                                }
                            }
                        }
                    }
                ],
                "method": "服务对应的方法名称(根据identifier生成)"
            }
        ]
}
```

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

Modbus协议产品的扩展配置数据结构如下：

```
{
  "profile": {
    "productKey": "产品key"
  },
  "properties": [
    {
      "identifier": "属性唯一标识符(产品下唯一)",
```

```
      "operateType": "(线圈状态/输入状态/保持寄存器/输入寄存器: coilStatus/
inputStatus/holdingRegister/inputRegister)",
      "registerAddress": "寄存器地址",
      "originalDataType": {
        "type": "属性类型:int16, uint16, int32, uint32, int64, uint64,
float, double, string, customized data(按大端顺序返回hex data)",
        "specs": {
          "registerCount": "寄存器的数据个数”，string, customized data特
有",
          "swap16": "把寄存器内16位数据的前后8个bits互换 (byte1byte2 ->
byte2byte1)，除 string, customized data外，其他数据类型特有",
          "reverseRegister": "Ex:把原始数据32位数据的bits互换 (byte1byte2
byte3byte4 ->byte3byte4byte1byte2”，除 string, customized data外，其他数
据类型特有"
        }
      },
      "scaling": "缩放因子",
      "pollingTime": "采集间隔，单位是ms",
      "trigger": "数据的上报方式，目前有 按时上报:1和变更上报:2"
    }
  ]
}
```

物模型使用流程

1. 使用控制台，#unique_22，或#unique_23。
2. 参见Link Kit SDK文档Link Kit SDK 文档，进行设备端物模型开发。
3. 开发完成后，设备端可以上报属性和事件；云端可以向设备端发送设置属性和调用服务的指令。

1.3.2 新增物模型

1.3.2.1 单个添加物模型

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

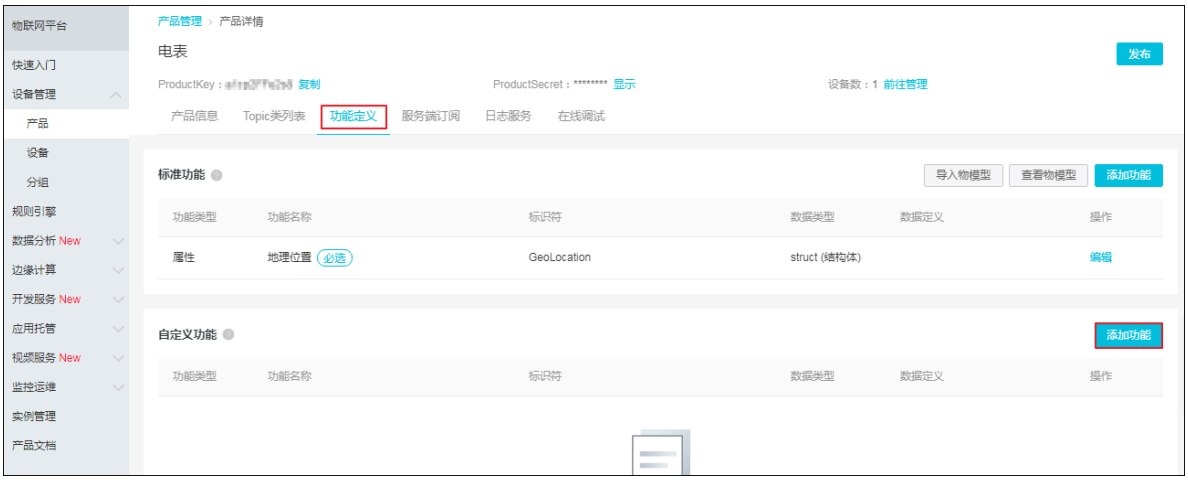
操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品。
3. 在产品管理页面产品列表中，单击产品所对应的查看操作按钮。
4. 单击功能定义。

> 说明:
> 已发布的产品不能添加和更新功能。

5. 添加标准功能。单击标准功能栏对应的添加功能按钮，然后在弹出对话框中，选择适用于该产品的物联网平台预定义的标准功能。

6. 添加自定义功能。单击自定义功能栏对应的添加功能为产品新增自定义功能。您可为产品自定义属性、服务和事件。



· 自定义属性。在添加自定义功能页面，选择功能类型为属性。设置参数完成后，单击确认。

属性参数设置说明如下表。

参数	描述
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、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。 如果您创建产品时选择了功能模板，输入功能名称时，将从标准功能库中筛选匹配的标准属性，供您选择。  说明: 当接入网关协议为Modbus时，不支持标准属性，仅支持自定义属性。
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。可包含英文、数字、下划线，长度不超过50个字符，例如PowerComsuption。  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	- 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-加热模式、2-制冷模式等。 - bool：布尔型。采用0或1来定义布尔值，例如 0-关；1-开。 - text：字符串。需定义字符串的数据长度，最长支持2048字节。 - date：时间戳。格式为String类型的UTC时间戳，单位：毫秒。 - struct：JSON对象。定义一个JSON结构体，新增JSON参数项，例如定义灯的颜色是由Red、Green、Blue三个参数组成的结构体。不支持结构体嵌套。 - array：数组。需声明数组内元素的数据类型，可选择int32、float、double、text或struct。需确保同一个数组元素类型相同。数组内可包含1-128个元素。  说明: 当设备协议为Modbus时，无需设置该参数。

参数	描述
步长	属性值和事件以及服务中输入输出参数值变化的最小粒度。数据类型为int32、float、double时，需要根据您的业务需要设置步长。 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℃……。
单位	单位可选择为无或根据实际情况选择。
读写类型	- 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）。 说明: 当接入网关协议为Modbus时，无需设置该参数。
描述	输入文字，对该功能进行说明或备注。长度限制为100字。

参数	描述
扩展描述	设备通信协议到标准物模型的映射关系。 设备接入网关协议为自定义、OPC UA或Modbus时，需填写该参数。 - 接入网关协议为自定义时，填写JSON格式的自定义配置信息，长度限制为1024字符。 - 接入网关协议为OPC UA时，设置节点名称。节点名称需保证属性维度下唯一。 - 接入网关协议为Modbus时，需设置以下参数： ■ 操作类型： ■ 线圈状态（只读，01） ■ 线圈状态（读写，读取使用01，写入使用05） ■ 线圈状态（读写，读取使用01，写入使用0F） ■ 离散量输入（只读，02） ■ 保持寄存器（只读，03） ■ 保持寄存器（读写，读取使用03，写入使用06） ■ 保持寄存器（读写，读取使用03，写入使用10） ■ 输入寄存器（只读，04） ■ 寄存器地址：十六进制，必须以`0x`开头，且限制范围是`0x0~0xFFFF`，例如`0xFE`。 ■ 原始数据类型：支持int16、uint16、int32、uint32、int64、uint64、float、double、string、bool、自定义（原始数据）多种数据类型。 ■ 取值范围：这是原始数据经过缩放因子处理之后的取值范围。不在该取值范围内的数据会被丢弃。物联网平台已为各操作类型设置了默认取值范围： ■ 线圈状态类型：0~1 ■ 离散量输入类型：0~1 ■ 保持寄存器类型：-2147483648 ~ 2147483647 ■ 输入寄存器类型：-2147483648 ~ 2147483647 ■ 交换寄存器内高低字节：是否把寄存器内16位数据的前后8个bits互换。 ■ `true`：互换。 ■ `false`：不互换。 ■ 交换寄存器顺序：是否把原始数据32位数据的bits互换。 ■ `true`：互换。 ■ `false`：不互换。 ■ 缩放因子：不能为0，默认为1，可以为负数。 ■ 采集间隔：数据采集间隔，单位ms，不能小于10。 ■ 数据上报方式：可选按时上报和变更上报。

- **自定义服务。在添加自定义功能页面，选择功能类型为服务。设置参数完成后，单击确认。**

说明:

接入网关的协议选择为Modbus时，不支持定义服务。

服务参数设置说明如下表。

参数	描述
功能名称	服务名称。 支持中文、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。 如果您创建产品时选择了功能模板，输入功能名称时，将从标准功能库中筛选匹配的标准服务，供您选择。  说明: 当接入网关协议为Modbus时，不支持自定义服务。
标识符	服务唯一标识符，在产品下具有唯一性。即 Alink JSON 格式中该服务的`identifier`的值。可包含英文、数字、和下划线，长度不超过30个字符。  说明: 不能用以下系统保留参数作为标识符： set、get、post、time、value。
调用方式	- 异步：服务为异步调用时，云端执行调用后直接返回结果，不会等待设备的回复消息。 - 同步：服务为同步调用时，云端会等待设备回复；若设备没有回复，则调用超时。
输入参数	设置该服务的入参，可选。 单击新增参数，在弹窗对话框中添加服务入参。 当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。  说明: - 不能用以下系统保留参数作为输入参数的标识符：set、get、post、time、value。 - 您可以使用某个属性作为入参，也可以自定义参数。例如在定义自动喷灌服务功能时，将已定义的属性喷灌时间和喷灌量作为自动喷灌服务的入参，则调用该参数时传入这两个参数，喷灌设备将按照设定的喷灌时间和喷灌量自动进行精准灌溉。 - 一个服务最多支持定义 20 个入参。

参数	描述
输出参数	设置该服务的出参，可选。 单击新增参数，在弹窗对话框中添加服务出参。 当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。 说明: - 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。 - 您可以使用某个属性作为出参，也可以自定义参数，例如将已定义的属性土壤湿度作为出参，则云端调用自动喷灌服务时，将返回当前土壤湿度的数据。 - 一个服务最多支持定义20个出参。
扩展描述	子设备接入网关协议为自定义协议或OPC UA时，需增加扩展描述，实现设备通信协议到标准物模型的映射关系。 当接入网关协议为自定义时，需传入JSON格式的自定义配置，长度不超过1024字符。 当接入网关协议为OPC UA时，设置节点名称。节点名称需保证服务维度下唯一。
描述	输入文字，对该服务功能进行说明或备注。长度限制为100字。

- 自定义事件。在添加自定义功能页面，选择功能类型为事件。设置参数完成后，单击确认。

说明:

接入网关的协议选择为Modbus时，不支持定义事件。

事件参数设置说明如下表。

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

参数	描述
标识符	事件唯一标识符，在产品下具有唯一性。即Alink JSON格式中该事件的identifier的值，作为设备上报该事件数据的Key，例如ErrorCode。 说明: 不能用以下系统保留参数作为标识符： set、get、post、time、value。
事件类型	- 信息：指设备上报的一般性通知，例如完成某项任务等。 - 告警：设备运行过程中主动上报的突发或异常情况，告警类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。 - 故障：设备运行过程中主动上报的突发或异常情况，故障类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。
输出参数	该事件的出参。单击增加参数，在弹窗对话框中添加一个服务出参。您可以使用某个属性作为出参，也可以自定义参数。例如将已定义的属性电压作为出参，则设备上报该故障事件时，将携带当前设备的电压值，用于进一步判断故障原因。 当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。 说明: - 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。 - 一个事件最多支持定义50个出参。
扩展描述	子设备接入网关协议为自定义协议或OPC UA时，需增加扩展描述，实现设备通信协议到标准物模型的映射关系。 当接入网关协议为自定义时，需传入JSON格式的自定义配置，长度不超过1024字符。 当接入网关协议为OPC UA时，设置节点名称。节点名称需保证事件维度下唯一。
描述	输入文字，对该事件功能进行说明或备注。长度限制为100字。

1.3.2.2 导入物模型

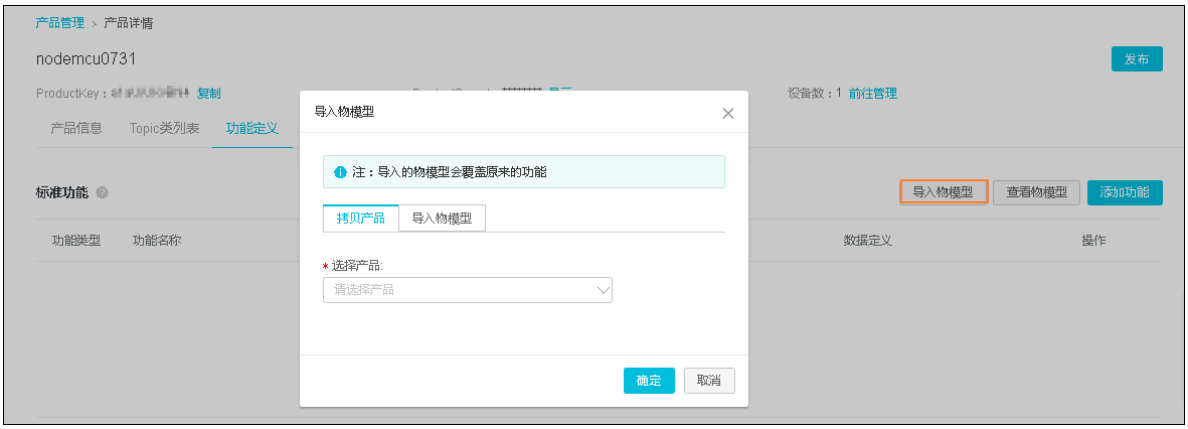
将物模型文件或其他产品的物模型导入为产品的物模型。

操作步骤

1. 登录物联网平台控制台。

2. 左侧导航栏选择设备管理 > 产品。
3. 在产品管理页面产品列表中，单击产品所对应的查看操作按钮。
4. 单击功能定义 > 导入物模型为产品导入物模型。

> **说明:**
>
> - 导入物模型后，会覆盖该产品原有的功能定义。请谨慎使用。
> - 设备协议为Modbus的产品，不支持导入物模型。



可通过以下两种方法导入物模型：

- 拷贝产品：选择已有产品，单击确定，即可将已有产品的物模型导入到此产品中。

  如果需要修改某些功能，单击功能对应的编辑按钮，即可修改该功能。
- 导入物模型：将物模型JSON文件上传至平台，然后单击确定。

> **说明:**
>
> 物模型文件大小不能超过64 KB。

1.3.3 物模型格式

物模型以JSON格式表达，简称为TSL（Thing Specification Language）。本文提供物模型的JSON字段说明。

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

物模型的 JSON 字段说明如下：

```
{
    "schema": "物的TSL描述schema",
    "link": "云端系统级uri,用来调用服务/订阅事件",
    "profile": {
        "productKey": "产品key"
    },
    "properties": [
        {
```

```
            "identifier": "属性唯一标识符(产品下唯一)",
            "name": "属性名称",
            "accessMode": "属性读写类型, 只读(r), 读写(rw)",
            "required": "是否是标准功能的必选属性",
            "dataType": {
                "type": "属性类型: int(原生), float(原生), double(原生),
text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int类型),
struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/float/
text)",
                "specs": {
                    "min": "参数最小值(int, float, double类型特有)",
                    "max": "参数最大值(int, float, double类型特有)",
                    "unit": "属性单位",
                    "unitName": "单位名称",
                    "size": "数组大小, 默认最大128(数组特有)",
                    "step": "步长, 字符串类型",
                    "item": {
                        "type": "数组元素的类型"
                    }
                }
            }
        }
    ],
    "events": [
        {
            "identifier": "事件唯一标识符(产品下唯一, 其中post是默认生成的属性
上报事件)",
            "name": "事件名称",
            "desc": "事件描述",
            "type": "事件类型(info,alert,error)",
            "required": "是否是标准功能的必选事件",
            "callType": "async(异步调用), sync(同步调用)",
            "outputData": [
                {
                    "identifier": "参数唯一标识符",
                    "name": "参数名称",
                    "dataType": {
                        "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                        "specs": {
                            "min": "参数最小值(int, float, double类型特
有)",
                            "max": "参数最大值(int, float, double类型特
有)",
                            "unit": "属性单位",
                            "unitName": "单位名称",
                            "size": "数组大小, 默认最大128(数组特有)",
                            "step": "步长, 字符串类型",
                            "item": {
                                "type": "数组元素的类型"
                            }
                        }
                    }
                }
            ],
            "method": "事件对应的方法名称(根据identifier生成)"
        }
    ],
    "services": [
        {
            "identifier": "服务唯一标识符(产品下唯一, 产品下唯一, 其中set/get
是根据属性的accessMode默认生成的服务)",
```

```
            "name": "服务名称",
            "desc": "服务描述",
            "required": "是否是标准功能的必选服务",
            "callType": "async(异步调用), sync(同步调用)",
            "inputData": [
                {
                    "identifier": "入参唯一标识符",
                    "name": "入参名称",
                    "dataType": {
                        "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                        "specs": {
                            "min": "参数最小值(int, float, double类型特
有)",
                            "max": "参数最大值(int, float, double类型特
有)",
                            "unit": "属性单位",
                            "unitName": "单位名称",
                            "size": "数组大小, 默认最大128(数组特有)",
                            "step": "步长, 字符串类型",
                            "item": {
                                "type": "数组元素的类型"
                            }
                        }
                    }
                }
            ],
            "outputData": [
                {
                    "identifier": "出参唯一标识符",
                    "name": "出参名称",
                    "dataType": {
                        "type": "属性类型: int(原生), float(原生), double
(原生), text(原生), date(String类型UTC毫秒), bool(0或1的int类型), enum(int
类型), struct(结构体类型, 可包含前面6种类型), array(数组类型, 支持int/double/
float/text)",
                        "specs": {
                            "min": "参数最小值(int, float, double类型特
有)",
                            "max": "参数最大值(int, float, double类型特
有)",
                            "unit": "属性单位",
                            "unitName": "单位名称",
                            "size": "数组大小, 默认最大128(数组特有)",
                            "step": "步长, 字符串类型",
                            "item": {
                                "type": "数组元素的类型(数组特有)"
                            }
                        }
                    }
                }
            ],
            "method": "服务对应的方法名称(根据identifier生成)"
        }
    ]
}
```

若该产品是设备，且接入网关协议为Modbus、OPC UA或自定义时，可查看物模型扩展配置。

Modbus协议产品的扩展配置示例如下：

```
{
  "profile": {
    "productKey": "产品key"
  },
  "properties": [
    {
      "identifier": "属性唯一标识符(产品下唯一)",
      "operateType": "(线圈状态/输入状态/保持寄存器/输入寄存器: coilStatus/
inputStatus/holdingRegister/inputRegister)",
      "registerAddress": "寄存器地址",
      "originalDataType": {
        "type": "属性类型:int16, uint16, int32, uint32, int64, uint64,
float, double, string, customized data(按大端顺序返回hex data)",
        "specs": {
          "registerCount": "寄存器的数据个数”，string, customized data特
有",
          "swap16": "把寄存器内16位数据的前后8个bits互换 (byte1byte2 ->
byte2byte10), 除 string, customized data外, 其他数据类型特有",
          "reverseRegister": "Ex:把原始数据32位数据的bits互换 (byte1byte2
byte3byte4 ->byte3byte4byte1byte2”, 除 string, customized data外, 其他数
据类型特有"
        }
      },
      "scaling": "缩放因子",
      "pollingTime": "采集间隔，单位是ms",
      "trigger": "数据的上报方式，目前有 按时上报:1和变更上报:2"
    }
  ]
}
```

1.4 数据解析

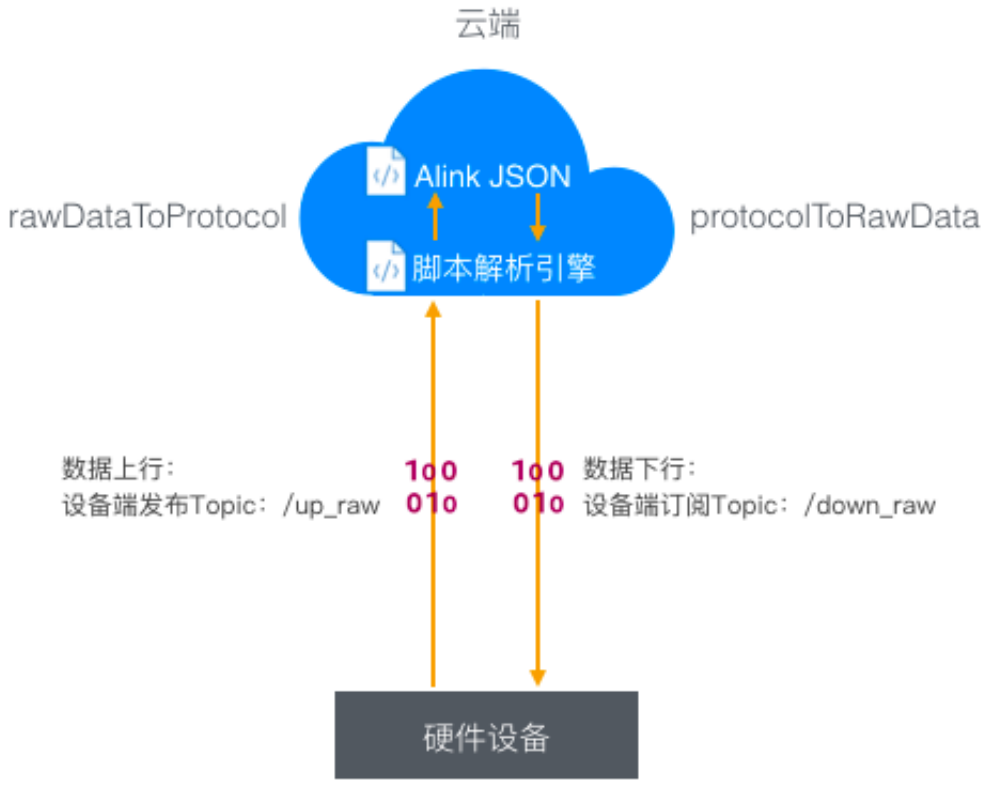
1.4.1 什么是数据解析

由于低配置且资源受限，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。您需在物联网平台控制台，编写数据解析脚本，用于将设备上下行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。

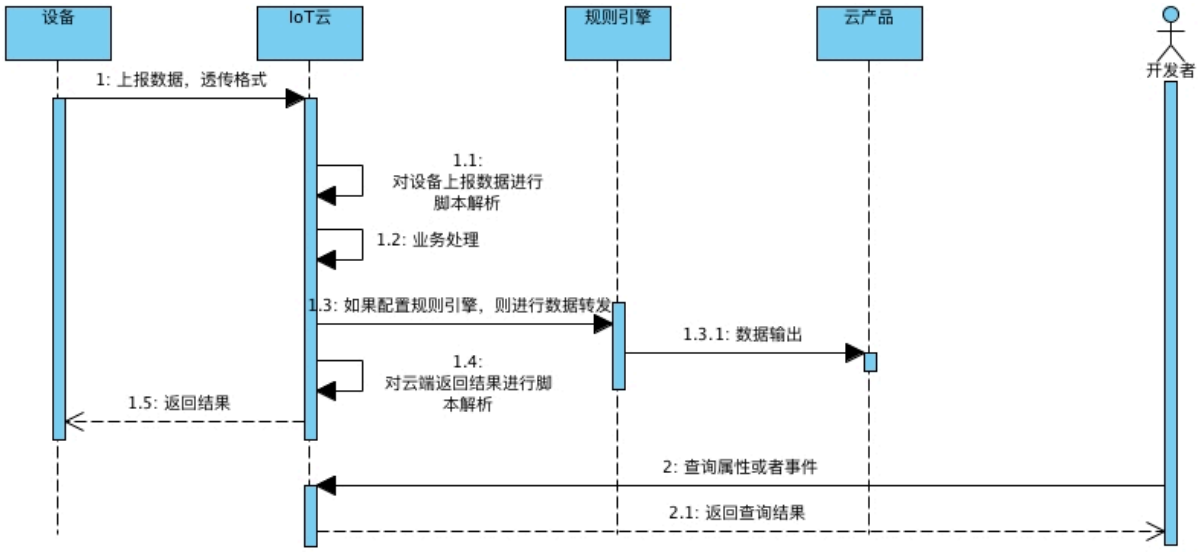
数据解析流程

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时，也会先通过脚本将数据转换为设备可以接收的数据格式，再下发给设备。

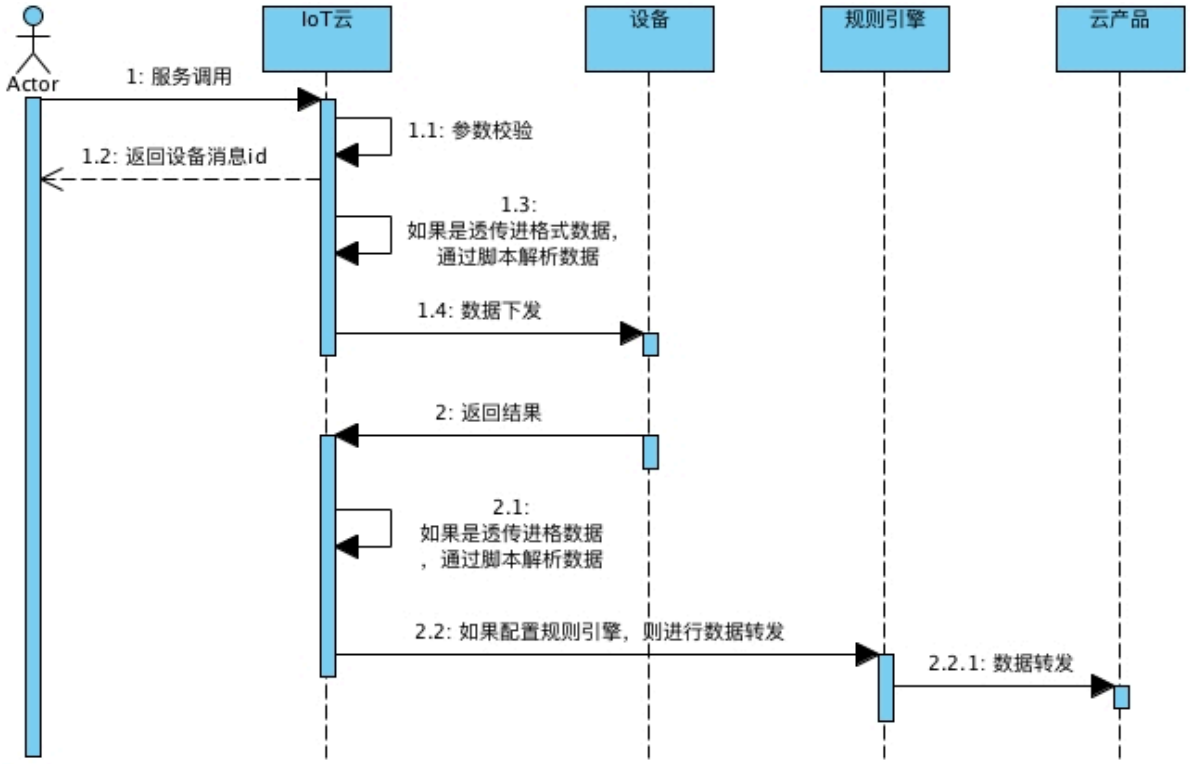
数据解析流程图：



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：



脚本格式

目前，只支持使用JavaScript语言编写解析脚本。物联网平台为您提供了在线脚本编辑器，用于编辑、提交脚本和模拟数据解析。

脚本中需定义支持以下两个方法：

- **Alink JSON格式数据转为设备自定义数据格式：**`protocolToRawData`。
- **设备自定义数据格式转Alink JSON格式数据：**`rawDataToProtocol`。

```
/**
 *  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;
```

```
}
```

脚本编辑

请单击以下链接，参见相关文档。

- 脚本编辑方法和测试方法，请参见#unique_30。
- 若您的设备为LoRaWAN节点设备，请参见#unique_31。
- 若提交的脚本不能正常解析数据，请参见#unique_32。

1.4.2 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，介绍数据解析脚本的编辑和测试过程，并提供一个脚本示例。

步骤一：编辑脚本

1. 在物联网平台控制台，创建产品，数据格式选择为透传/自定义格式。
2. 为该产品定义物模型。请参见#unique_22中，“自定义属性”章节。

   本示例中定义了以下三个属性。

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

3. 在设备通信协议中做如下定义：

   表 1-1: 设备上报数据请求

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

   表 1-2: 设备上报数据响应

字段	字节数
帧类型	1字节

字段	字节数
请求ID	4字节
结果code	1字节

表 1-3: 设置属性请求

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

表 1-4: 属性设置响应

字段	字节数
帧类型	1字节
请求ID	4字节
结果code	1字节

4. 编写脚本。

在物联网平台控制台，产品详情页的数据解析页签下，编写脚本。

脚本中需定义以下两个方法：

- 将Alink JSON格式数据转为设备自定义数据格式：`protocolToRawData`。
- 将设备自定义数据格式转Alink JSON格式数据：`rawDataToProtocol`。

完整的示例脚本Demo，请参见本文附录：示例脚本。

步骤二：在线测试脚本

脚本编辑完成后，使用模拟数据在线测试脚本。

- 模拟解析设备上报的属性数据。

选择模拟类型为设备上报数据，输入以下模拟的设备上报数据，然后单击运行。

```
0x00002233441232013fa00000
```

数据解析引擎会按照脚本规则，将透传数据转换为JSON格式数据。运行结果栏将显示解析结果。

```
{
    "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，然后单击运行。

```
{
  "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，然后单击运行。

```
{
    "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，然后单击运行。

```
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"
}
```

若脚本不正确，运行结果栏将显示报错信息。您需根据报错信息，查找错误，并修改脚本代码。

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。

调试方法：

· 测试上报属性数据

1. 使用设备端上报设备属性数据，如`0x00022334412320l3fa00000`。
2. 在物联网平台控制台，该设备的设备详情页运行状态页签下，查看是否有相应的属性数据。

· 测试下发属性数据

1. 在物联网平台控制台，选择监控运维 > 在线调试。
2. 选择要调试的产品和设备，并选择调试真实设备，功能选择为要调试的属性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
    }
}
```

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

附录：示例脚本

根据脚本示例Demo如下：

```
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
/*
示例数据:
设备上报数据
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"}
*/
```

```
function rawDataToProtocol(bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = new Object();
    var fHead = uint8Array[0]; // command
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD; //ALink JSON格式
 - 属性上报topic
        jsonMap['version'] = '1.0'; //ALink JSON格式 - 协议版本号固定字段
        jsonMap['id'] = '' + dataView.getInt32(1); //ALink JSON格式
 - 标示该次请求id值
        var params = {};
        params['prop_int16'] = dataView.getInt16(5); //对应产品属性中
prop_int16
        params['prop_bool'] = uint8Array[7]; //对应产品属性中 prop_bool
        params['prop_float'] = dataView.getFloat32(8); //对应产品属性中
prop_float
        jsonMap['params'] = params; //ALink JSON格式 - params标准字段
    } else if(fHead == COMMAND_SET_REPLY) {
        jsonMap['version'] = '1.0'; //ALink JSON格式 - 协议版本号固定字段
        jsonMap['id'] = '' + dataView.getInt32(1); //ALink JSON格式
 - 标示该次请求id值
        jsonMap['code'] = ''+ dataView.getUint8(5);
        jsonMap['data'] = {};
    }

    return jsonMap;
}
/*
示例数据:
属性设置
传入参数 ->
    {"method":"thing.service.property.set","id":"12345","version":"1.0
","params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1}}
输出结果 ->
    0x0100003039014d0142f6e76d

设备上报的返回结果
传入数据 ->
    {"method":"thing.event.property.post","id":"12345","version":"1.0
","code":200,"data":{}}
输出结果 ->
    0x0200003039c8
*/
function protocolToRawData(json) {
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD) // 属性设置
    {
        var params = json['params'];
        var prop_float = params['prop_float'];
        var prop_int16 = params['prop_int16'];
        var prop_bool = params['prop_bool'];
        //按照自定义协议格式拼接 rawData
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET
)); // command字段
        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_RE
PORT_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，对于有些命令不做处理
        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;
}
//以下是部分辅助函数
function buffer_uint8(value) {
    var uint8Array = new Uint8Array(1);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value) {
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
```

相关文档

- 了解数据解析流程和脚本格式等基本信息，请参见#unique_34。
- 关于数据解析问题排查，请参见#unique_32。

1.4.3 LoRaWAN设备数据解析

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

步骤一：编辑脚本

1. 在物联网平台控制台，创建连网方式为LoRaWAN的产品。
2. 为该产品定义功能。功能定义具体方法，请参见#unique_36。

   本示例中，定义了以下属性、事件和服务。

   表 1-5: 属性

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

   表 1-6: 服务

标识符	调用方式	输入参数
SetTempHumiThreshold	异步	四个输入参数，数据类型均为int32： · 温度过高告警阈值（标识符：MaxTemp） · 温度过低告警阈值（标识符：MinTemp） · 湿度过高告警阈值（标识符：MaxHumi） · 湿度过低告警阈值（标识符：MinHumi）

   表 1-7: 事件

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

3. 编写脚本。

在产品的产品详情页数据解析页签下编写脚本。

脚本中需定义以下两个方法：

- 将Alink JSON格式数据转为设备自定义格式数据：`protocolToRawData`。
- 将设备自定义数据格式转Alink JSON格式数据：`rawDataToProtocol`。

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

表 1-8: 下行数据解析输出结果起始三字节

LoRa Downlink	字节数	说明
DFlag	1	固定为`0x5D`。
FPort	1	下行端口号。
DHDR	1	可选： - `0`：表示“Unconfirmed Data Down”数据帧。 - `1`：表示“Confirmed Data Down”数据帧。

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

完整的脚本示例Demo，请参见本文附录章节。

步骤二： 在线测试脚本

在数据解析编辑器中，使用模拟数据测试脚本。

- **设备上报数据模拟解析**

  在模拟输入框中，输入模拟数据，如000102，选择模拟类型为设备上报数据，单击运行。右侧运行结果中将显示数据解析是否成功。

- **设备接收数据模拟解析**

  输入JSON格式的云端下行模拟数据，选择模拟类型为设备接收数据，单击运行。

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

步骤三：提交脚本

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

说明:

仅提交后的脚本才能被物联网平台调用；草稿状态的脚本不能被调用。

步骤四：使用真实设备调试

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

- 测试LoRaWAN设备上报温湿度属性。
  1. 使用设备端发送数据，如000102。
  2. 在该设备的设备详情页运行状态页签下，查看设备上报的属性数据。
- 测试LoRaWAN设备上报事件。
  1. 使用设备端发送事件数据，如，温度告警数据0102，或湿度告警数据0202。
  2. 在该设备的设备详情页事件管理页签下，查看设备上报的事件数据。
- 测试调用LoRaWAN设备服务。
  1. 在物联网平台控制台，选择监控运维 > 在线调试。
  2. 选择要调试的产品和设备，并选择调试真实设备，功能选择为温度湿度阈值（SetTempHumiThreshold），输入以下数据后，单击发送指令。

```
{
        "MaxTemp": 50,
        "MinTemp": 8,
        "MaxHumi": 90,
```

```
        "MinHumi": 10
    }
```

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

查看设备属性、事件和服务调用数据的路径如下图：

附录：示例脚本

根据以上产品和其功能定义，编辑的示例脚本如下：

```
var ALINK_ID = "12345";
var ALINK_VERSION = "1.1";
var ALINK_PROP_POST_METHOD     = 'thing.event.property.post';
var ALINK_EVENT_TEMPERR_METHOD = 'thing.event.TempError.post';
var ALINK_EVENT_HUMIERR_METHOD = 'thing.event.HumiError.post';
var ALINK_PROP_SET_METHOD      = 'thing.service.property.set';
var ALINK_SERVICE_THSET_METHOD = 'thing.service.SetTempHumiThreshold';
/*
 * 示例数据:
 *  传入参数 ->
 *      000102 // 共3个字节
 *  输出结果 ->
 *      {"method":"thing.event.property.post", "id":"12345", "params":
{"Temperature":1,"Humidity":2}, "version":"1.1"}
 *  传入参数 ->
 *      0102 // 共2个字节
 *  输出结果 ->
 *      {"method":"thing.event.TempError.post","id":"12345","params":
{"Temperature":2},"version":"1.1"}
 *  传入参数 ->
 *      0202 // 共2个字节
 *  输出结果 ->
 *     {"method":"thing.event.HumiError.post","id":"12345","params":{"
Humidity":2},"version":"1.1"}
 */
function rawDataToProtocol(bytes)
{
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++)
    {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var params = {};
```

```
    var jsonMap = {};
    var dataView = new DataView(uint8Array.buffer, 0);
    var cmd = uint8Array[0]; // command
    if (cmd === 0x00)
    {
        params['Temperature']  = dataView.getInt8(1);
        params['Humidity']     = dataView.getInt8(2);
        jsonMap['method']  = ALINK_PROP_POST_METHOD;
    }
    else if (cmd == 0x01)
    {
        params['Temperature']  = dataView.getInt8(1);
        jsonMap['method']  = ALINK_EVENT_TEMPERR_METHOD;
    }
    else if (cmd == 0x02)
    {
        params['Humidity']  = dataView.getInt8(1);
        jsonMap['method']  = ALINK_EVENT_HUMIERR_METHOD;
    }
    else
    {
        return null;
    }
    jsonMap['version'] = ALINK_VERSION;
    jsonMap['id']      = ALINK_ID;
    jsonMap['params']  = params;
    return jsonMap;
}
/*
 * 示例数据:
 *  传入参数 ->
 *      {"method":"thing.service.SetTempHumiThreshold", "id":"12345",
 "version":"1.1", "params":{"MaxTemp":50, "MinTemp":8, "MaxHumi":90, "
MinHumi":10}}
 *  输出结果 ->
 *      0x5d0a000332085a0a
 */
function protocolToRawData(json)
{
    var id  = json['id'];
    var method  = json['method'];
    var version = json['version'];
    var payloadArray = [];
    // 追加下行帧头部
    payloadArray = payloadArray.concat(0x5d);
    payloadArray = payloadArray.concat(0x0a);
    payloadArray = payloadArray.concat(0x00);
    if (method == ALINK_SERVICE_THSET_METHOD)
    {
        var params  = json['params'];
        var maxtemp = params['MaxTemp'];
        var mintemp = params['MinTemp'];
        var maxhumi = params['MaxHumi'];
        var minhumi = params['MinHumi'];
        payloadArray = payloadArray.concat(0x03);
        if (maxtemp !== null)
        {
            payloadArray = payloadArray.concat(maxtemp);
        }
        if (mintemp !== null)
        {
            payloadArray = payloadArray.concat(mintemp);
        }
        if (maxhumi !== null)
```

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

相关文档

- 了解数据解析流程和脚本格式等基本信息，请参见#unique_34。
- 关于数据解析问题排查，请参见#unique_32。

1.4.4 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的排错方法。

本地环境调试脚本

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

以下本地调试代码基于#unique_30中的示例脚本。您在实际使用时，请按照您的脚本需求进行具体参数设置。

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

线上问题排查

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

若设备已经上报了数据，但是运行状态下却没有显示对应的数据。如下图所示。

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

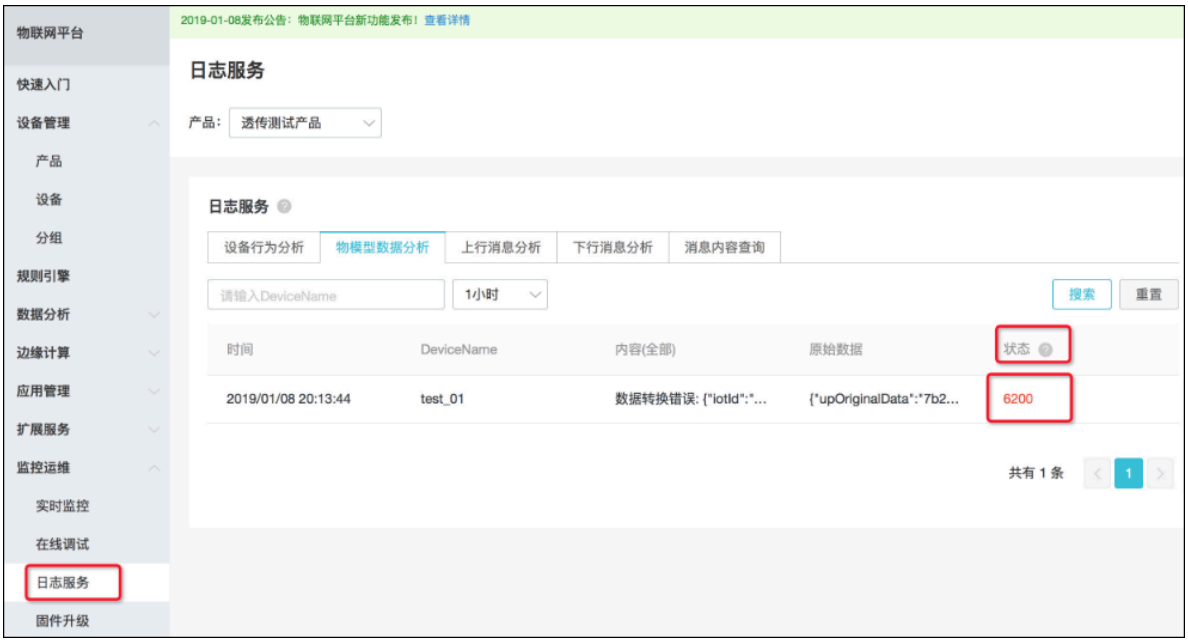
问题排查过程如下：

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

下面列举一些错误。

- 脚本不存在。

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



- Alink method不存在。

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

  日志内容如：

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

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

1.5 Topic

物联网平台中，云端和设备端通过 Topic 来实现消息通信。设备上报消息至指定的Topic中，并从Topic中订阅消息。云端将指令下发到Topic中，并订阅具体Topic来获取设备信息。

1.5.1 什么是Topic

物联网平台中，服务端和设备端通过 Topic 来实现消息通信。Topic是针对设备的概念，Topic类是针对产品的概念。产品的Topic类会自动映射到产品下的所有设备中，生成用于消息通信的具体设备Topic。

产品Topic类

为了方便海量设备基于海量Topic进行通信，简化授权操作，物联网平台增加了产品Topic类的概念。Topic类是一类Topic的集合。例如，产品的自定义Topic类`/${YourProductKey}/${YourDeviceName}/user/update`是具体Topic`/${YourProductKey}/device1/user/update`和`/${YourProductKey}/device2/user/update`的集合。

您创建设备后，产品的所有Topic类会自动映射到设备上。您无需单独为每个设备创建Topic。

图 1-1: Topic 自动生成示意图

关于Topic类的说明：

- Topic类中，以正斜线（/）进行分层，区分每个类目。其中，有两个类目为既定类目：`${YourProductKey}`表示产品的标识符ProductKey；`${YourDeviceName}`表示设备名称。
- 类目命名只能包含字母，数字和下划线（_）。每级类目不能为空。
- 设备操作权限：发布表示设备可以往该Topic发布消息；订阅表示设备可以订阅该Topic获取消息。

- Topic类是一个Topic模版配置，编辑更新某个Topic类后，可能对产品下所有设备使用该类Topic通信产生影响。建议在设备研发阶段设计好，设备上线后不再变更Topic类。
- 产品Topic类的订阅操作权限是定义产品（所有设备）对此类Topic是否有发起订阅指令（sub）的权限。 订阅（sub）和 取消订阅（unsub）都需由设备发起。设备发送sub指令订阅某个Topic后，该订阅永久生效；仅在设备发起unsub指令取消订阅该Topic后，订阅才会被取消。

  如果您需要管控单个设备的消息收发，请在控制台的设备列表页或服务端调用#unique_40接口，禁用该设备；或在业务上管控发送给设备的消息。

设备Topic

产品的Topic类不用于通信，只是定义Topic。用于消息通信的是具体的设备Topic。

- Topic格式和Topic类格式一致。区别在于Topic类中的变量`${YourDeviceName}`，在Topic中是具体的设备名称。
- 设备对应的Topic是从产品Topic类映射出来，根据设备名称而动态创建的。设备的具体Topic中带有设备名称（即`DeviceName`），只能被该设备用于消息通信。例如，Topic：`/${YourProductKey}/device1/user/update`归属于设备名为device1的设备，所以只能被设备device1用于发布或订阅消息，而不能被设备device2用于发布或订阅消息。

系统Topic和自定义Topic

物联网平台有两类Topic。

类别	说明
系统Topic	物联网平台预定义的Topic。 系统Topic包含展示在控制台产品、设备详情页下的Topic和各功能使用的Topic。具体功能使用的Topic请在对应功能的文档中查看。 例如，物模型相关的Topic一般以`/sys/`开头；固件升级相关的Topic以`/ota/`开头；设备影子的Topic以`/shadow/`开头。
自定义Topic	您可以根据业务需求，在产品的Topic类列表页，自定义Topic类。

Topic通配符

物联网平台支持使用两种通配符：

通配符	描述
#	#必须出现在Topic的最后一个类目，代表本级及下级所有类目。 例如，`/a1aycMA****/device1/user/#`表示设备Topic `/a1aycMA****/device1/user/update`和`/a1aycMA****/device1/user/update/error`。
+	代表本级所有类目。 例如，`/a1aycMA****/device1/user/+/error`，表示设备Topic `/a1aycMA****/device1/user/get/error`和`/a1aycMA****/device1/user/update/error`。

通配符可用于以下两种场合：

- 使用通配符自定义Topic，实现批量订阅。
- 编写规则引擎数据流转SQL时，指定数据源为一批Topic。

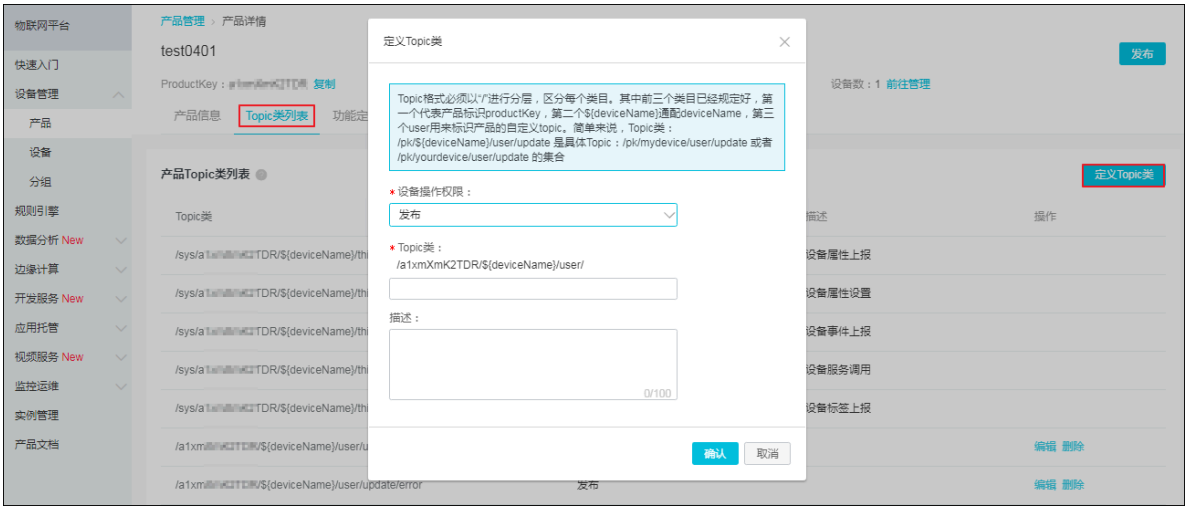
1.5.2 自定义Topic

本文介绍如何为产品自定义Topic类。自定义Topic类将自动映射到该产品下的所有设备中。

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏单击设备管理 > 产品。
3. 在产品管理页面，找到需要自定义Topic类的产品，并单击对应操作栏中的查看按钮。
4. 在产品详情页面，单击Topic类列表 > 定义Topic类。

5. 定义Topic类。



参数	描述
设备操作权限	设备对该Topic的操作权限，可设置为发布、订阅、发布和订阅。
Topic类	将Topic类填充完整。 **说明:** 只有设备操作权限为订阅时，才可以使用通配符+和#自定义Topic类。 +代表本级所有类目。 #代表本级及下级所有类目。它只能出现在Topic类的最后一个类目。
描述	写一些话描述该Topic类，可以为空。

6. 单击确认。

带通配符的自定义Topic

带通配符的Topic不支持在设备的Topic列表页面执行发布消息操作，仅支持订阅操作。

例如，某产品有一个自定义Topic类：`/a1aycMA****/${deviceName}/user/#`。DeviceName为Light的设备订阅`/a1aycMA****/Light/user/#`表示批量订阅了以/a1aycMA****/Light/user/为开头的全部Topic，包含`/a1aycMA****/Light/user/get`，`/a1aycMA****/Light/user/data`等。

例如，某产品有一个自定义Topic类：`/a1aycMA****/${deviceName}/user/+/error`。DeviceName为Robot的设备订阅`/a1aycMA****/Robot/user/+/error`，表示批量订阅

了 `/a1aycMA****/Robot/user/get/error`、`/a1aycMA****/Robot/user/update/error`等Topic。

自定义Topic通信

服务端调用#unique_44，可向指定的自定义Topic发布消息；设备通过订阅该Topic，接收来自服务端的消息。

使用自定义Topic通信的示例，请参见#unique_45。

1.6 标签

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

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

标签包括产品标签、设备标签和分组标签。标签的结构为`Key:Value`。

本文将详细讲解如何在控制台创建产品标签、设备标签与分组标签。

说明:

每个产品、设备或分组最多可有100个标签。

产品标签

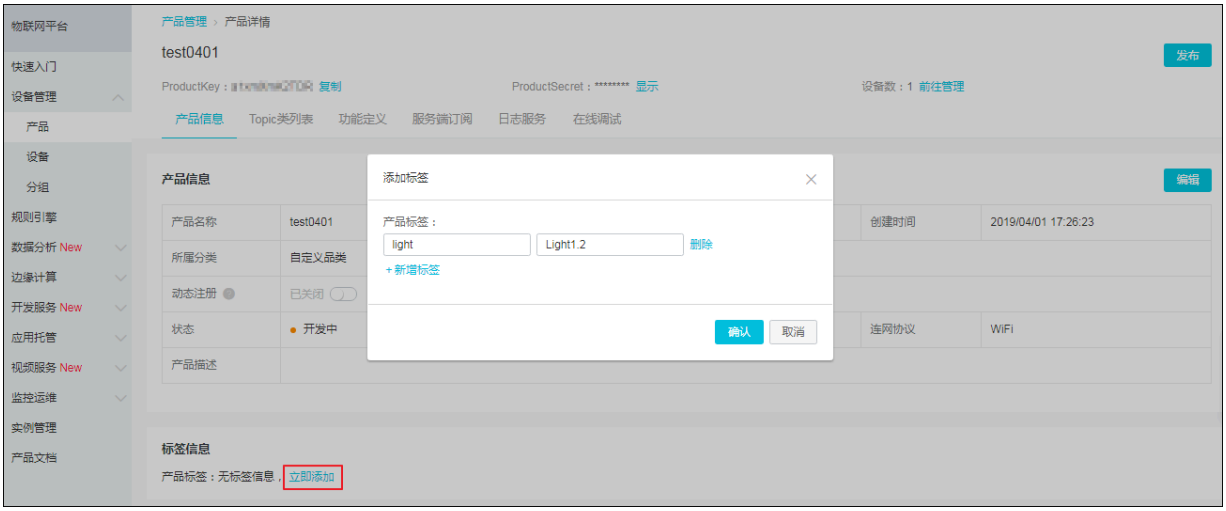
产品标签通常描述的是对一个产品下所有设备所具有的共性信息。如产品的制造商、所属单位、外观尺寸、操作系统等。需在创建产品后，再为该产品添加产品标签。

在控制台添加产品标签操作步骤：

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品。
3. 在产品管理页面，找到需要添加标签的产品，并单击对应操作栏中的查看。
4. 在标签信息部分，单击立即添加按钮。
5. 在弹出的对话框中，输入标签的 标签Key 和 标签Value，然后单击确认。

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

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



设备标签

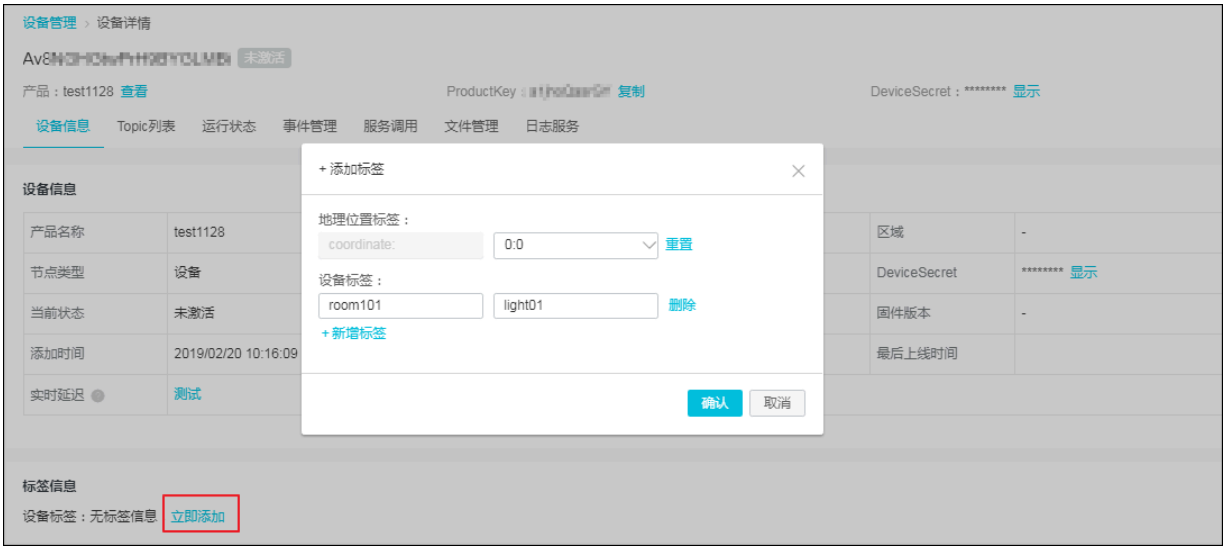
您可以根据设备的特性为设备添加特有的标签，方便对设备进行管理。例如，为房间 201 的智能电表定义一个标签为`room:201`。

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

在控制台添加设备标签的操作步骤：

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备。
3. 在设备管理页面，单击要添加标签的设备所对应的查看，进入设备详情页面。
4. 在标签信息部分，单击立即添加按钮。
5. 在弹出的对话框中，输入标签的 标签Key 和 标签Value，然后单击确认。

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



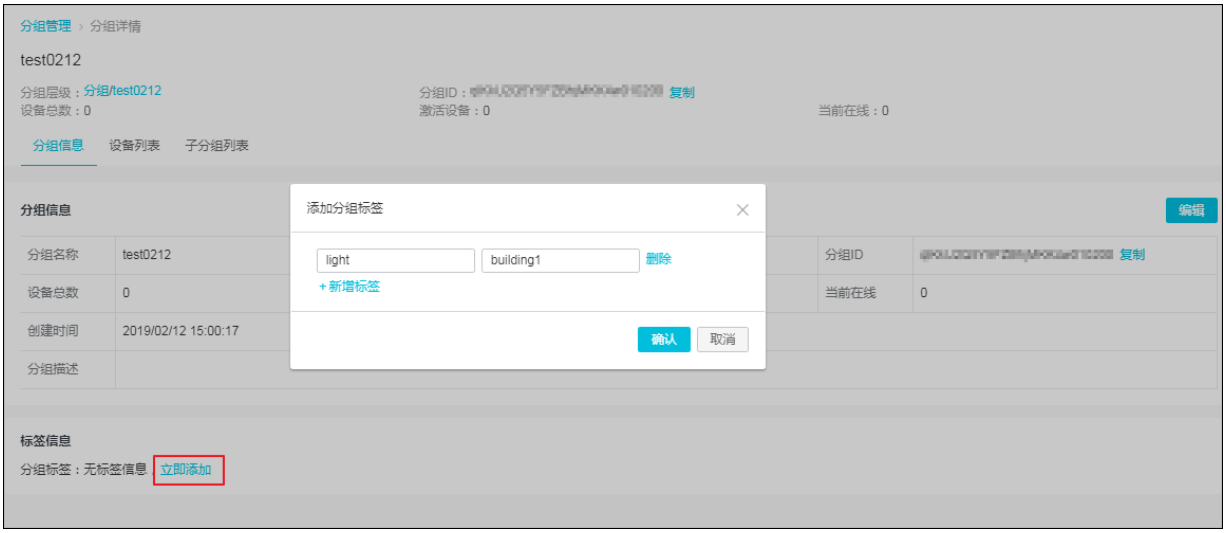
分组标签

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

添加分组标签操作步骤：

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 分组。
3. 在分组管理页面，找到需要添加标签的分组，并单击对应操作栏中的查看。
4. 在标签信息部分，单击立即添加按钮。
5. 在弹出的对话框中，输入标签的 标签Key 和 标签Value，然后单击确认。

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



批量操作标签

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

请参见API列表。

1.7 网关与子设备

1.7.1 网关与子设备

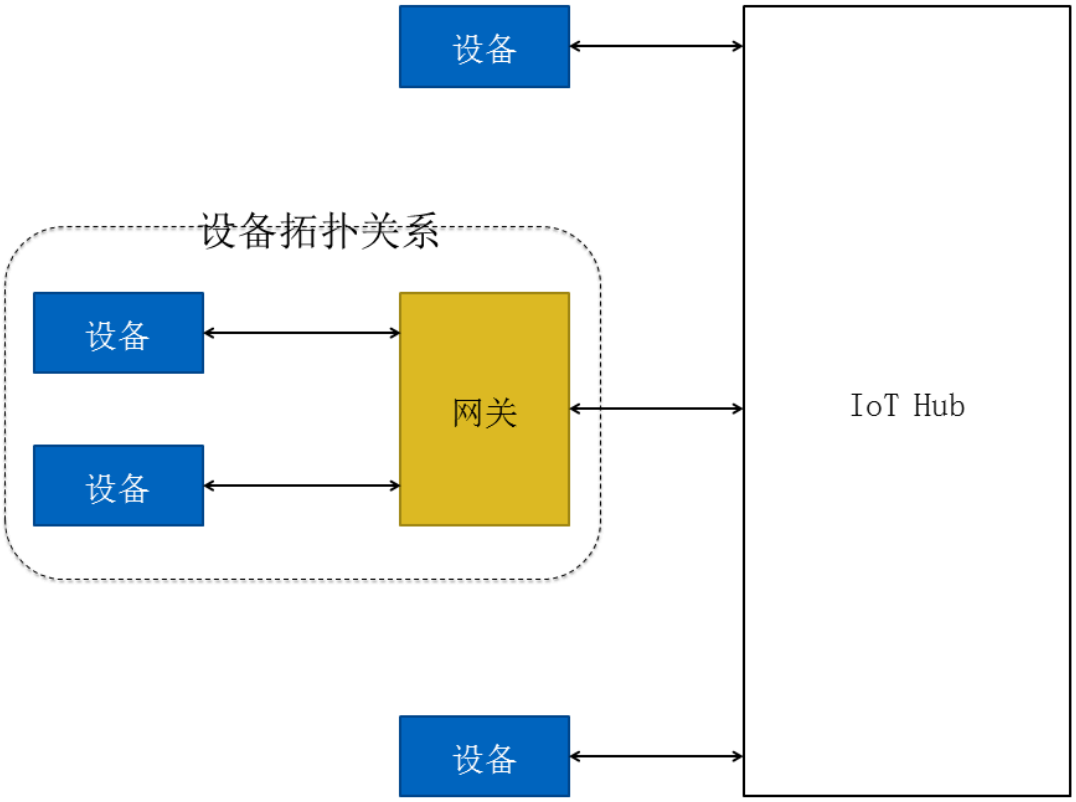
物联网平台支持设备直连，也支持设备挂载在网关上，作为网关的子设备，由网关直连。

网关与设备

创建产品与设备时，需要选择节点类型。平台目前支持两种节点类型：设备和网关。

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

网关与子设备的拓扑关系如下图所示。



接入方式

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

- 网关接入物联网平台的方法与普通设备相同，具体请参见Link Kit SDK文档Link Kit SDK文档。
- 子设备接入物联网平台有两种方式：
  - 使用一机一密的认证方式。网关获取子设备的设备证书，由网关向物联网平台上报子设备证书信息（ProductKey、DeviceName和DeviceSecret）。
  - 使用子设备动态注册的认证方式。需在控制台，打开子设备的动态注册开关。网关获取子设备的ProductKey和DeviceName后，由网关代理子设备进行动态注册，云端校验子设备身份。校验通过后，动态下发DeviceSecret。然后子设备通过设备证书（ProductKey、DeviceName和DeviceSecret）接入物联网平台。

1.7.2 子设备管理

本文介绍在网关下面关联子设备。

操作步骤

1. 左侧导航栏选择设备管理 > 设备。
2. 在设备管理页面，找到对应网关，单击查看，进入设备详情页。
3. 单击子设备管理 > 添加子设备。

4. 在弹出页面上，选择要关联的子设备，然后单击确定。

参数	描述
产品	选择子设备对应的产品名称。
设备	选择子设备的设备名称。

预期结果

子设备信息配置完成后，可在子设备管理页签下，您可以：

- 单击子设备对应的查看按钮，进入子设备详情页查看子设备信息。
- 单击子设备对应的删除按钮，将该子设备从网关中删除。

> **说明:**
> 本操作仅删除子设备与网关的拓扑关系，并不删除设备本身。

1.8 服务端订阅

1.8.1 什么是服务端订阅

服务端可以直接订阅产品下所有类型的消息：设备上报消息、设备状态变化通知、网关发现子设备上报消息、设备生命周期变更消息和设备拓扑关系变更消息。配置服务端订阅后，物联网平台会将产品下所有设备的已订阅类型的消息转发至您的服务端。目前支持两种订阅方式：通过HTTP/2通道进行消息流转和推送消息到消息服务（MNS）。

适用场景

服务端订阅适用于单纯的接收设备数据的场景，并且适用于高并发场景。如果您有多个服务器，服务端订阅的消息会随机转发至某个服务器。

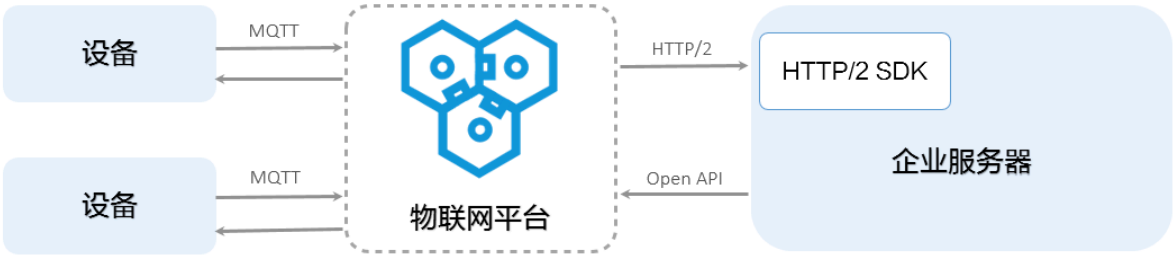
还需同时满足以下条件的场景：

- 服务端接收产品下全部设备的订阅数据。
- 设备数据流转性能要求不超过5000条/秒。

HTTP/2服务端订阅

目前，新版物联网平台通过HTTP/2通道进行消息流转。您配置HTTP/2服务端订阅后，物联网平台会将产品下所有设备的已订阅类型消息，通过HTTP/2通道推送至服务端。

HTTP/2通道服务端订阅消息流转流程图如下：



通过接入HTTP/2 SDK，企业服务器可以直接从物联网平台接收消息。HTTP/2 SDK提供身份认证、Topic订阅、消息发送和消息接收能力，并支持设备接入和云端接入能力。

- 服务端HTTP/2 SDK，用于物联网平台与企业服务器之间的大量消息流转。
- 设备端HTTP/2 SDK，用于设备与物联网平台之间的消息收发。

> 说明：
> 目前，只提供了Java和.NET语言的SDK。

HTTP/2通道的配置方法和示例，请参见：

- 使用限制
- 服务端Java HTTP/2 SDK

- 服务端.Net HTTP/2 SDK

服务端订阅与规则引擎数据流转的对比，请参见数据流转方案对比。

推送到MNS

物联网平台将订阅的消息推送到消息服务（MNS），您的服务端应用通过监听MNS队列接收设备消息。

使用消息服务订阅设备消息的具体配置方法，请参见使用消息服务（MNS）订阅设备消息

说明:

消息服务接收物联网平台推送的消息会收取费用。消息服务计费和使用，请参见消息服务文档消息服务文档。

1.8.2 开发指南（Java）

本文档为服务端订阅功能的开发指南，介绍了Java版本的开发方法。完成后，您可以成功使用SDK，通过HTTP/2通道，接收物联网平台推送过来的设备消息。

服务端订阅Demo：HTTP/2 SDK (Java) server side demo。

以下简单介绍服务端订阅的开发流程。

配置服务端订阅

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品。
3. 在产品列表中，搜索到要配置服务端订阅的产品，并单击该产品对应的查看按钮，进入产品详情页。
4. 单击服务端订阅 > 设置。

5. 选择推送的消息类型。

- 设备上报消息：指产品下所有设备 Topic 列表中，具有发布权限的 Topic 中的消息。勾选后，可以通过 HTTP/2 SDK 接收这些消息。

  设备上报消息，包括设备上报的自定义数据和物模型数据（属性上报、事件上报、属性设置响应和服务调用响应）。

  例如，一个产品有3个Topic类，分别是：

  - /${YourProductKey}/${YourDeviceName}/user/get，具有订阅权限。
  - /${YourProductKey}/${YourDeviceName}/user/update，具有发布权限。
  - /sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post，具有发布权限。

  那么，服务端订阅会推送具有发布权限的Topic类中的消息，即/${YourProductKey}/${YourDeviceName}/user/update和/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post中的消息。其中，/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post中的数据已经过系统处理。
- 设备状态变化通知：指一旦该产品下的设备状态变化时通知的消息，例如设备上线、下线的消息。设备状态消息的发送 Topic 为 /as/mqtt/status/${YourProductKey}/${YourDeviceName}。勾选后，可以通过 HTTP/2 SDK 接收设备状态变化的通知消息。
- 网关发现子设备上报：网关可以将发现的子设备信息上报给物联网平台。需要网关上的应用程序支持。网关产品特有消息类型。
- 设备拓扑关系变更：指子设备和网关之间的拓扑关系建立和解除消息。网关产品特有消息类型。
- 设备生命周期变更：包括设备创建、删除、禁用、启用等消息。

说明：

> 设备上报物模型消息、设备状态变化通知、网关发现子设备上报消息、设备拓扑关系变更消息和设备生命周期变更消息均默认为QoS=0；设备上报消息（除物模型相关消息外），您可以在设备端上设置为QoS=0或QoS=1。

接入 SDK

在Maven工程项目中添加以下依赖，安装阿里云IoT SDK。

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.3</version>
</dependency>

<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.7.1</version>
</dependency>
```

身份认证

使用服务端订阅功能，需要基于您的阿里云 AccessKey 进行身份认证并建立连接。

建立连接示例如下：

```
// 阿里云accessKey
String accessKey = "xxxxxxxxxxxxxx";
// 阿里云accessSecret
String accessSecret = "xxxxxxxxxxxxxx";
// regionId
String regionId = "cn-shanghai";
// 阿里云uid
String uid = "xxxxxxxxxxx";
// endPoint:  https://${uid}.iot-as-http2.${region}.aliyuncs.com
String endPoint = "https://" + uid + ".iot-as-http2." + regionId + ".
aliyuncs.com";

// 连接配置
Profile profile = Profile.getAccessKeyProfile(endPoint, regionId,
accessKey, accessSecret);

// 构造客户端
MessageClient client = MessageClientFactory.messageClient(profile);
// 数据接收
client.connect(messageToken -> {
    Message m = messageToken.getMessage();
    System.out.println("receive message from " + m);
    return MessageCallback.Action.CommitSuccess;
});
```

以上示例中账号相关信息的获取方法：

参数	获取途径
accessKey	您的账号AccessKey ID。 登录阿里云控制台，将光标移至账号头像上，然后单击accesskeys，跳转至用户信息管理页，即可获取。
accessSecret	您的账号AccessKey Secret。获取方式同accessKey。
uid	您的账号ID。 用主账号登录阿里云控制台，单击账号头像，跳转至账号管理控制台，即可获取账号UID。
regionId	您的物联网平台服务所在地域代码。 在物联网平台控制台页，右上方即可查看地域（Region）。RegionId的表达方法，请参见文档#unique_61。

调用连接相关的方法

方法	说明
connect方法	调用该方法建立服务端HTTP/2 SDK和物联网平台的连接。连接建立后，服务端便可以收到物联网平台推送的消息。 `void connect(MessageCallback callback);`
disconnect方法	调用该方法断开服务端HTTP/2 SDK与物联网平台连接，停止接收消息。 `void disconnect();`
isConnected方法	调用该方法判断HTTP/2 SDK是否已连接物联网平台。 `boolean isConnected();`

setConnectionCallback方法	调用该方法设置连接状态监听回调。
	`void setConnectionCallback(ConnectionCallback var1);`
	示例：
	`client.setConnectionCallback(new ConnectionCallback() {`
	`@Override`
	`public void onConnectionLost() {`
	`System.out.println("连接断开");`
	`}`
	`@Override`
	`public void onConnected(boolean isReconnected) {`
	`System.out.println("连接成功，是否为重连：" + isReconnected);`
	`}`
	`});`

设置消息接收接口

连接建立后，服务端会立即向SDK推送已订阅的消息。因此，建立连接时，需要提供消息接收接口，用于处理未设置回调的消息。建议在连接之前，调用 `setMessageListener` 设置消息回调。

您需要通过 `MessageCallback` 接口的consume方法，和调用messageClient的setMessageListener()方法来设置消息接收接口。

consume方法的返回值决定SDK是否发送 ACK（acknowledgement，即回复确认消息）。

设置消息接收接口的方法如下：

```
MessageCallback messageCallback = new MessageCallback() {
    @Override
    public Action consume(MessageToken messageToken) {
        Message m = messageToken.getMessage();
        log.info("receive : " + new String(messageToken.getMessage().
getPayload()));
        return MessageCallback.Action.CommitSuccess;
    }
};
messageClient.setMessageListener("/${YourProductKey}/#",messageCal
lback);
```

其中，

- 参数MessageToken指消息回执的消息体。通过MessageToken.getMessage()可获取消息体。MessageToken可以用于手动回复ACK。

  消息体包含的内容如下：

  ```
  public class Message {
      // 消息体
      private byte[] payload;
      // Topic
      private String topic;
      // 消息ID
      private String messageId;
      // QoS
      private int qos;
  }
  ```

  各类型消息的具体格式，请参见数据格式。

  说明：

  关于设备上下线状态，为避免消息时序紊乱造成影响，建议您根据消息中的lastTime字段来判断最终设备状态。

- 示例中，`messageClient.setMessageListener("/${YourProductKey}/#", messageCallback);`用于设置指定Topic回调。

  您可以设置为指定Topic回调，也可以设置为通用回调。

  - 指定Topic回调

    指定Topic回调的优先级高于通用回调。一条消息匹配到多个Topic时，按字典顺序优先调用，并且仅回调一次。

    设置回调时，可以指定带通配符的Topic，如/${YourProductKey}/${YourDeviceName}/#。

    示例：

    ```
    messageClient.setMessageListener("/alEddfaXXXX/device1/#",
    messageCallback);
    //当收到消息的Topic，如"/alEddfaXXXX/device1/update"，匹配指定Topic
    时，会优先调用该回调
    ```

  - 通用回调

    未指定Topic回调的消息，则调用通用回调。

    设置通用回调方法：

    ```
    messageClient.setMessageListener(messageCallback);
    //当收到消息topic未匹配到已定指的Topic 回调时，调用该回调
    ```

- 设置回复ACK。

  QOS=1的消息消费后，需要回复ACK。支持设置为自动回复ACK和手动回复ACK。默认为自动回复 ACK。本示例中未设置回复 ACK，则默认为自动回复。

  - 自动回复ACK：设置为自动回复ACK后，若`MessageCallback.consume`的返回值为`true`， SDK会发送ACK；返回`false`或抛出异常，则不会返回ACK。对于QOS=1且未回复ACK的消息，服务器会重新发送。
  - 手动回复ACK方法：

    ```
    CompletableFuture<Boolean> ack(MessageToken var1);
    ```

    调用该方法的请求参数为`MessageToken`，表示需要回复ACK的消息。该参数值从`MessageCallback`回调信息中获取。

    设置回复ACK示例：

    ```
    client.connect(messageToken -> {
        System.out.println(messageToken.getMessage());
        asyncHandleMessage(messageToken);
        return MessageCallback.Action.CommitAckManually;
    ```

```
});
public void asyncHandleMessage(MessageToken messageToken) {
  client.ack(messageToken)
}
```

相关文档

- 服务端订阅使用限制说明，请参见#unique_63
- 查看各类型消息的具体格式，请参见数据格式。

1.8.3 开发指南（.NET）

本文档为服务端订阅功能的开发指南，介绍了.NET版本的开发方法。完成后，您可以成功使用SDK，通过HTTP/2通道，接收物联网平台推送过来的设备消息.

服务端订阅Demo：HTTP/2 SDK(.NET) server side demo。

说明:

- 使用Demo程序时，需要使用NLog包。请访问NLog官网下载，建议版本号：4.5.11。
- 使用Demo程序时，请引入最新依赖包。使用依赖包时，需引入Newtonsoft.Json，建议版本号：12.0.2 。

以下简单介绍服务端订阅的开发流程。

配置服务端订阅

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品。
3. 在产品列表中，搜索到要配置服务端订阅的产品，并单击该产品对应的查看按钮，进入产品详情页。
4. 单击服务端订阅 > 设置。

5. 选择推送的消息类型。

- 设备上报消息：指产品下所有设备 Topic 列表中，具有发布权限的 Topic 中的消息。勾选后，可以通过 HTTP/2 SDK 接收这些消息。

  设备上报消息，包括设备上报的自定义数据和物模型数据（属性上报、事件上报、属性设置响应和服务调用响应）。

  例如，一个产品有3个Topic类，分别是：

  - /${YourProductKey}/${YourDeviceName}/user/get，具有订阅权限。
  - /${YourProductKey}/${YourDeviceName}/user/update，具有发布权限。
  - /sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post，具有发布权限。

  那么，服务端订阅会推送具有发布权限的Topic类中的消息，即/${YourProductKey}/${YourDeviceName}/user/update和/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post中的消息。其中，/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post中的数据已经过系统处理。
- 设备状态变化通知：指一旦该产品下的设备状态变化时通知的消息，例如设备上线、下线的消息。设备状态消息的发送 Topic 为/as/mqtt/status/${YourProductKey}/${YourDeviceName}。勾选后，可以通过 HTTP/2 SDK 接收设备状态变化的通知消息。
- 网关发现子设备上报：网关可以将发现的子设备信息上报给物联网平台。需要网关上的应用程序支持。网关产品特有消息类型。
- 设备拓扑关系变更：子设备和网关之间的拓扑关系建立和解除消息。网关产品特有消息类型。
- 设备生命周期变更：设备创建、删除、禁用、启用等消息。

说明：

设备上报物模型消息、设备状态变化通知、网关发现子设备上报消息、设备拓扑关系变更消息和设备生命周期变更消息均默认为QoS=0；设备上报消息（除物模型相关消息外），您可以在设备端上设置为QoS=0或QoS=1。

接入 SDK

在工程中添加依赖包。

身份认证

使用服务端订阅功能，需要基于您的阿里云 AccessKey 进行身份认证并建立连接。

建立连接示例如下：

```
//阿里云accessKey
string accessKey = "xxxxxxxxxxxxxx";
//阿里云accessSecret
string accessSecret = "xxxxxxxxxxxxxx";
//regionId
string regionId = "cn-shanghai";
//阿里云uid
string uid = "xxxxxxxxxxxxxx";
//domain
string domain = ".aliyuncs.com";
//endpoint
string endpoint = "https://" + uid + ".iot-as-http2." + regionId +
domain;

//连接参数配置
Profile profile = new Profile();
profile.AccessKey = accessKey;
profile.AccessSecret = accessSecret;
profile.RegionId = regionId;
profile.Domain = domain;
profile.Url = endpoint;
//清除堆积消息
profile.CleanSession = true;
profile.GetAccessKeyAuthParams();

//构造客户端
IMessageClient client = new MessageClient(profile);

//连接HTTP2通道，并接收消息
client.DoConnection(new DefaultHttp2MessageCallback());

//指定topic回调
client.SetMessageListener("/${YourProductKey}/#", new CustomHttp
2MessageCallback());
```

以上示例中账号相关信息的获取方法：

参数	获取途径
accessKey	您的账号AccessKey ID。 登录阿里云控制台，将光标移至账号头像上，然后单击accesskeys，跳转至用户信息管理页，即可获取。
accessSecret	您的账号AccessKey Secret。获取方式同accessKey。
uid	您的账号ID。 用主账号登录阿里云控制台，单击账号头像，跳转至账号管理控制台，即可获取账号UID。
regionId	您的物联网平台服务所在地域代码。 在物联网平台控制台页，右上方即可查看地域（Region）。RegionId的表达方法，请参见文档#unique_61。

设置消息接收接口

连接建立后，服务端会立即向 SDK 推送已订阅的消息。因此，建立连接时，需要实现消息接收接口。

消息接收接口如下：

```
public interface IHttp2MessageCallback
{
    ConsumeAction Consume(Http2ConsumeMessage http2ConsumeMessage);
}
```

您需要通过 IHttp2MessageCallback接口的consume方法，来设置消息接收接口。

设置消息接收接口的方法如下：

```
public class DefaultHttp2MessageCallback : IHttp2MessageCallback
    {
        public DefaultHttp2MessageCallback()
        {
        }

        public ConsumeAction Consume(Http2ConsumeMessage http2Consu
meMessage)
        {
            Console.WriteLine("receive : " + http2ConsumeMessage.
MessageId);
            //自动回复ACK
            return ConsumeAction.CommitSuccess;
        }
    }
```

其中，

- 参数 Http2ConsumeMessage 是消息回执的消息体。

  消息体包含的内容如下：

```
public class Http2ConsumeMessage
{
    //消息体
    public byte[] Payload { get; set; }
    //Topic
    public string Topic { get; set; }
    //消息ID
    public string MessageId { get; set; }
    //QoS
    public int Qos { get; set; }
    //连接体
    public Http2Connection Connection { get; set; }
}
```

  各类型消息具体格式，请参见数据格式。

  说明：

  关于设备上下线状态，为避免消息时序紊乱造成影响，建议您根据消息中的lastTime字段来判断最终设备状态。

- `messageClient.setMessageListener("/${YourProductKey}/#",messageCallback);`用于设置回调。本示例中，设置为指定 Topic 回调。

  您可以设置为指定 Topic 回调，也可以设置为通用回调。

  - 指定 Topic 回调

    指定 Topic 回调的优先级高于通用回调。一条消息匹配到多个 Topic 时，按字典顺序优先调用，并且仅回调一次。

    设置回调时，可以指定带通配符的 Topic，如/${YourProductKey}/${YourDeviceName}/# 。

    示例：

```
client.SetMessageListener("/alEddfaXXXX/device1/#",messageCallback);
```

```
//当收到消息的Topic，如"/alEddfaXXXX/device1/update"，匹配指定Topic
时，会优先调用该回调
```

- 通用回调

  未指定 Topic 回调的消息，则调用通用回调。

  设置通用回调方法：

```
new DefaultHttp2MessageCallback()
```

· 设置回复 ACK（acknowledgement，即回复确认消息）。

对于 QOS>0 的消息，消费后需要回复 ACK。SDK 支持自动回复 ACK 和手动回复 ACK。默认为自动回复 ACK。本示例中未设置回复 ACK，则默认为自动回复。

- 自动回复 ACK：设置为自动回复 ACK 后，若 `IHttp2MessageCallback.consume`的返回值为 `ConsumeAction.CommitSuccess` 则 SDK 会发送 ACK；返回 `ConsumeAction.CommitFailure` 或抛出异常，则不会返回 ACK。对于 QOS>0 且未回复 ACK 的消息，服务端会重新发送。
- 手动回复 ACK：通过`ConsumeAction.CommitFailure`设置手动回复 ACK。

  设置为手动回复 ACK 后，需要调用`MessageClient.DoAck()`方法回复 ACK，参数为`topic`、`messageId`和连接体。 这些参数可以在接收消息中获取到。

  手动回复 ACK 的方法：

```
client.DoAck(connection, topic, messageId, delegate);
```

相关文档

· 服务端订阅使用限制说明，请参见#unique_63

· 查看各类型消息的具体格式，请参见数据格式。

1.8.4 使用限制

当您使用服务端订阅时，请注意以下限制。

限制	描述
JDK版本	仅支持JDK8。
认证超时	连接建立之后，需要立刻发送认证请求。如果15秒内没有认证成功，服务器将主动关闭连接。

限制	描述
数据超时	连接建立之后，客户端需要定期发送PING包来维持连接。发送PING包的时间间隔可以在客户端设置，默认为30秒，最大60秒。 若超过60秒发送PING包或数据，服务端会关闭连接。 若超过设定的时间，客户端没有收到PING包响应或数据应答，SDK将主动断开重连，默认时间间隔为60秒。
推送超时	推送失败重试消息时，每次批量推送10条。若该批次消息在10秒后，仍未收到客户端回复的ACK，则认为推送超时。
失败推送重试策略	每60秒重新推送一次因客户端离线、消息消费慢等原因导致的堆积消息。
消息保存时长	QoS0的消息保存1天，QoS1的消息保存7天。
SDK实例个数	每个阿里云账号最多可以启动64个SDK实例。
单租户限流限制	默认单租户的限流限制为1,000 QPS。如果您有特殊需求，请提交工单。

1.8.5 使用消息服务订阅设备消息

物联网平台服务端订阅支持将设备消息发送至消息服务（MNS），云端应用通过监听MNS队列，获取设备消息。本章讲解使用MNS订阅设备消息的配置方法。

操作步骤

1. 在物联网平台控制台上，为产品配置服务端订阅，实现物联网平台将消息自动转发至MNS。

   a) 在产品的产品详情页，选择服务端订阅页签。

   b) 单击服务端订阅（推送MNS）对应的设置按钮。

   c) 在服务端订阅对话框中，选择要推送的消息类型后，单击保存。

   订阅完成后，物联网平台会自动创建MNS消息队列，名称格式为`aliyun-iot-${yourProductKey}`。如果已存在相关消息队列，则不再创建。该消息队列信息将显示在服务端订阅页。

2. 配置监听MNS队列，以接收设备消息。

   以下示例中，使用MNS Java SDK监听消息。

   下载MNS SDK Demo，请访问MNS文档MNS文档。

   a. 在pom.xml文件中，添加如下依赖安装MNS Java SDK。

```
<dependency>
    <groupId>com.aliyun.mns</groupId>
    <artifactId>aliyun-sdk-mns</artifactId>
    <version>1.1.8</version>
    <classifier>jar-with-dependencies</classifier>
```

```
</dependency>
```

b. 配置接收消息时，需填入以下信息。

```
CloudAccount account = new CloudAccount( $AccessKeyId, $AccessKeyS
ecret, $AccountEndpoint);
```

- $AccessKeyId和$AccessKeySecret需替换为您的阿里云账号访问API的基本信息。登录阿里云控制台，光标移至阿里云账号头像上，然后单击AccessKey管理，创建或查看AccessKey信息。
- $AccountEndpoint需填写实际的Endpoint值。在MNS控制台，单击获取Endpoint获取。

c. 填写接收设备消息的逻辑。

```
MNSClient client = account.getMNSClient();
CloudQueue queue = client.getQueueRef("aliyun-iot-a1xxxxxx8o9
"); //请输入IoT自动创建的队列名称

    while (true) {
    // 获取消息
    Message popMsg = queue.popMessage(10); //长轮询等待时间为10秒

    if (popMsg != null) {
        System.out.println("PopMessage Body: "+ popMsg.getMessage
BodyAsRawString()); //获取原始消息
        queue.deleteMessage(popMsg.getReceiptHandle()); //从队列中
删除消息
    } else {
        System.out.println("Continuing"); } }
```

d. 运行程序，完成对MNS队列的监听。

3. 启动设备，上报消息。

   设备端SDK开发，请参见Link Kit SDK文档Link Kit SDK文档。

4. 检查云端应用是否监听到设备消息。若成功监听，将获得如下所示消息代码。

```
{
"messageid":" ",//消息标识
"messagetype":"upload",
"topic":"//信息来源Topic,
"payload": //Base64编码后的数据
"timestamp": //时间戳
}
```

参数	说明
messageid	物联网平台生成的消息ID。

参数	说明
messagetype	消息类型。 · status：设备状态通知 · upload：设备上报消息 · device_lifecycle：设备生命周期变更通知 · topo_lifecycle：设备拓扑关系变更 · topo_listfound：网关发现子设备上报
topic	服务端监听到的信息来源的物联网平台Topic。
payload	Base64编码的消息数据。 payload数据格式，请参见数据格式。
timestamp	时间戳，以Epoch时间表示。

相关文档

数据格式

使用规则引擎，您需要基于Topic编写SQL处理数据。自定义Topic中数据格式由您自己定义，物联网平台不做处理。系统Topic中数据格式由物联网平台定义，此时您需要根据平台定义的数据格式处理数据。本文讲述了系统Topic中的数据格式。

相关文档

#unique_68

1.9 设备分组

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

背景信息

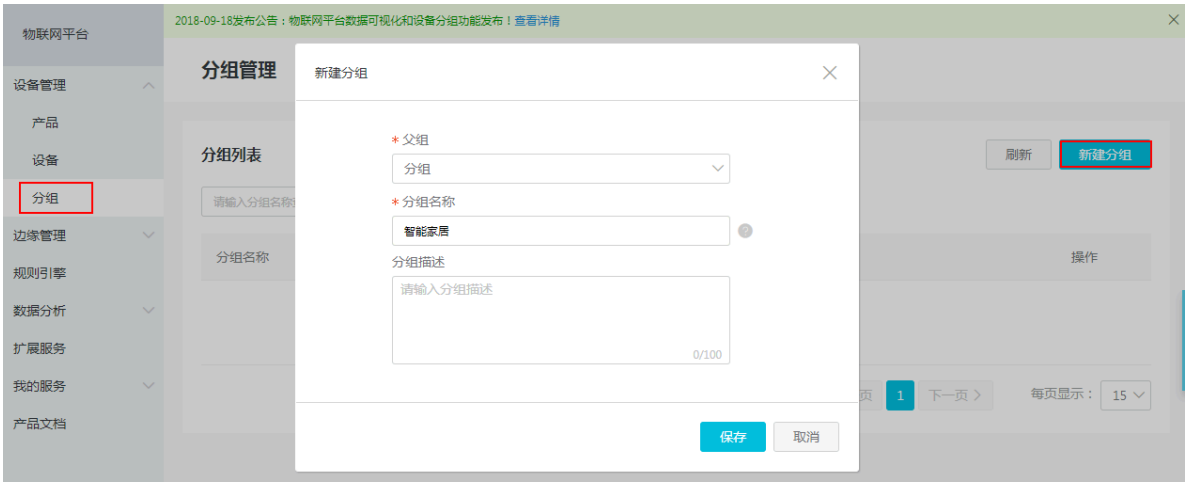
- 一个分组最多可包含100个子分组。
- 分组只支持三级嵌套，即分组>子分组>子子分组。
- 一个子分组只能隶属于一个父组。
- 分组的嵌套关系创建后不能修改，只能删除后重新创建。
- 分组下有子分组时，不能直接删除分组。需子分组全部删除后，才能删除父组。
- 搜索分组时，支持分组名称模糊搜索，包括在分组列表和子分组列表里的搜索。

操作步骤

1. 登录物联网平台控制台。
2. 单击设备管理 > 分组进入分组管理页面。

3. 单击新建分组，设置分组参数，并单击保存。

> **说明:**
> 一个阿里云账号下最多可创建1,000个分组，包括分组和子分组。



参数信息解释如下：

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

4. 在分组管理页面，单击已创建分组对应的查看操作按钮，进入分组详情页面。
5. （可选）为分组添加标签，即自定义分组标识，以便灵活管理分组。

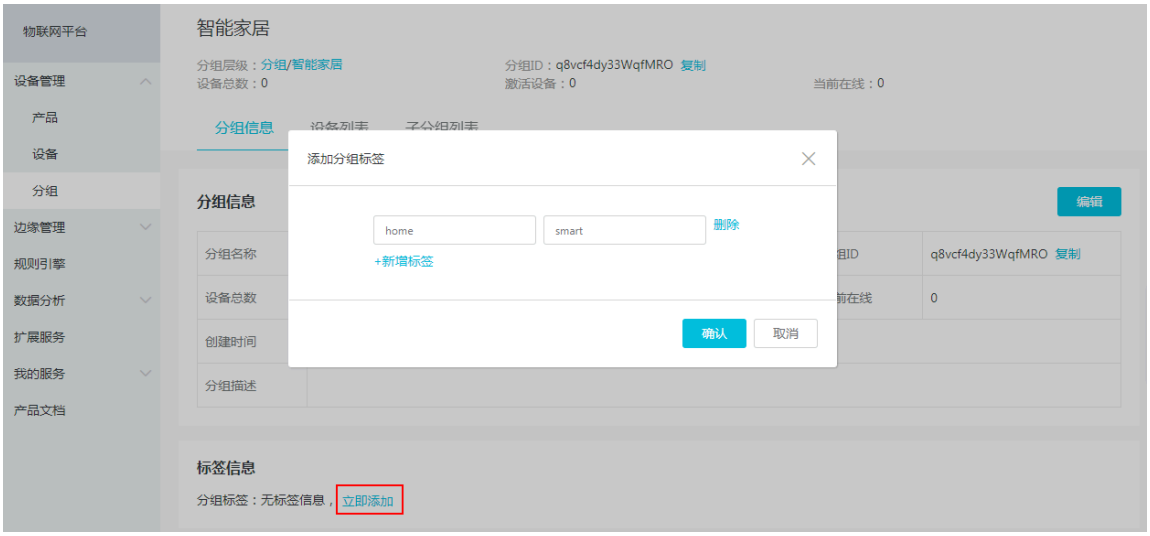
a) 单击标签信息栏下的立即添加，并输入标签的key和value。

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

b) 单击确认添加标签。

> **说明:**

一个分组最多可添加100个标签。



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

   搜索设备：

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

   说明：

   - 单次最多添加1000个设备。单个分组最多添加20,000个设备。

- 一个设备最多可以被添加到10个分组中。

添加设备到分组页面右上方的全部和已选择按钮说明：

- 单击全部，显示所有设备列表。
- 单击已选择，显示您已勾选的设备列表。

7. （可选）单击子分组列表 > 新建分组，为分组添加子分组。

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

a) 选择该子分组的父组，输入子分组名称和描述，然后单击保存。

b) 在子分组列表页面，单击子分组对应的查看操作按钮，进入该子分组的分组详情页。

c) 单击设备列表 > 添加设备到分组，然后为该子分组添加设备。

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

1.10 文件管理

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

前提条件

- 设备端成功连接到物联网平台。

  设备端SDK开发，请参见Link Kit SDK文档Link Kit SDK文档。

- 设备端编译配置HTTP/2上传文件功能。

  配置设备端上传文件功能，请参见Link Kit SDK文档中文件上传章节。

操作步骤

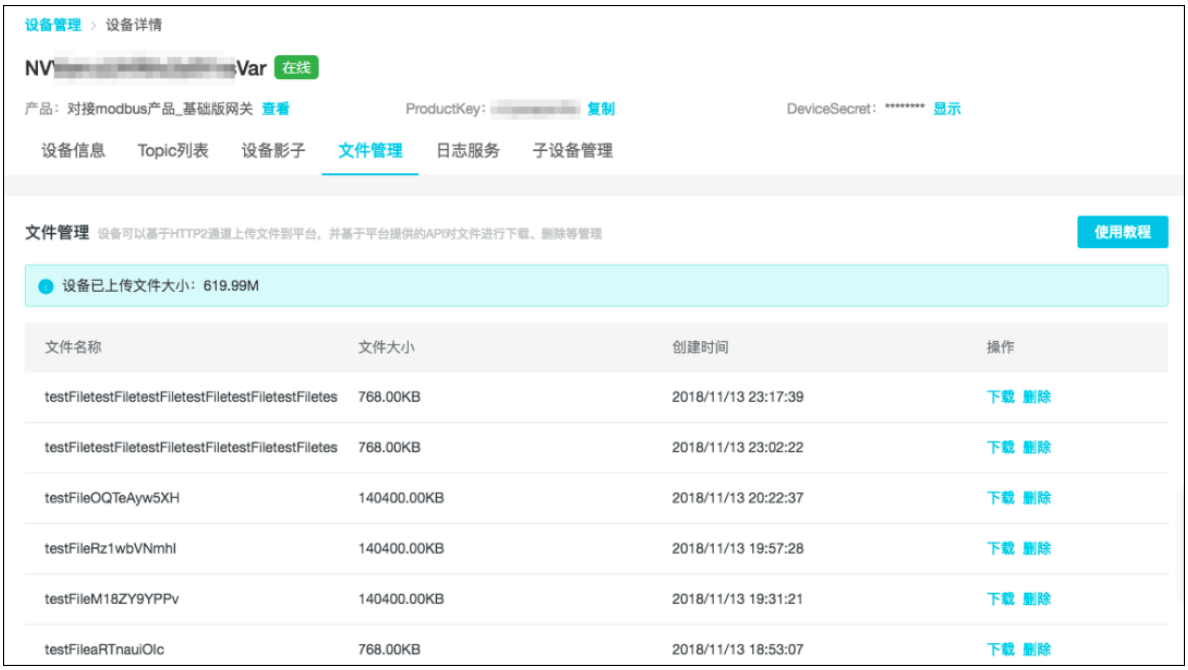
1. 登录物联网平台控制台。
2. 在左侧导航栏，选择设备管理 > 设备，然后单击设备对应的查看按钮。

3. 在设备详情页面，选择文件管理页签。

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

说明:

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



您可以对已上传的文件进行如下操作：

操作	描述
下载	下载该文件到本地。
删除	删除该文件。

除在控制台管理文件外，您还可以通过调用云端API查询或删除文件。请参见#unique_71、#unique_72、#unique_73。

2 规则引擎

2.1 数据流转

2.1.1 数据流转概览

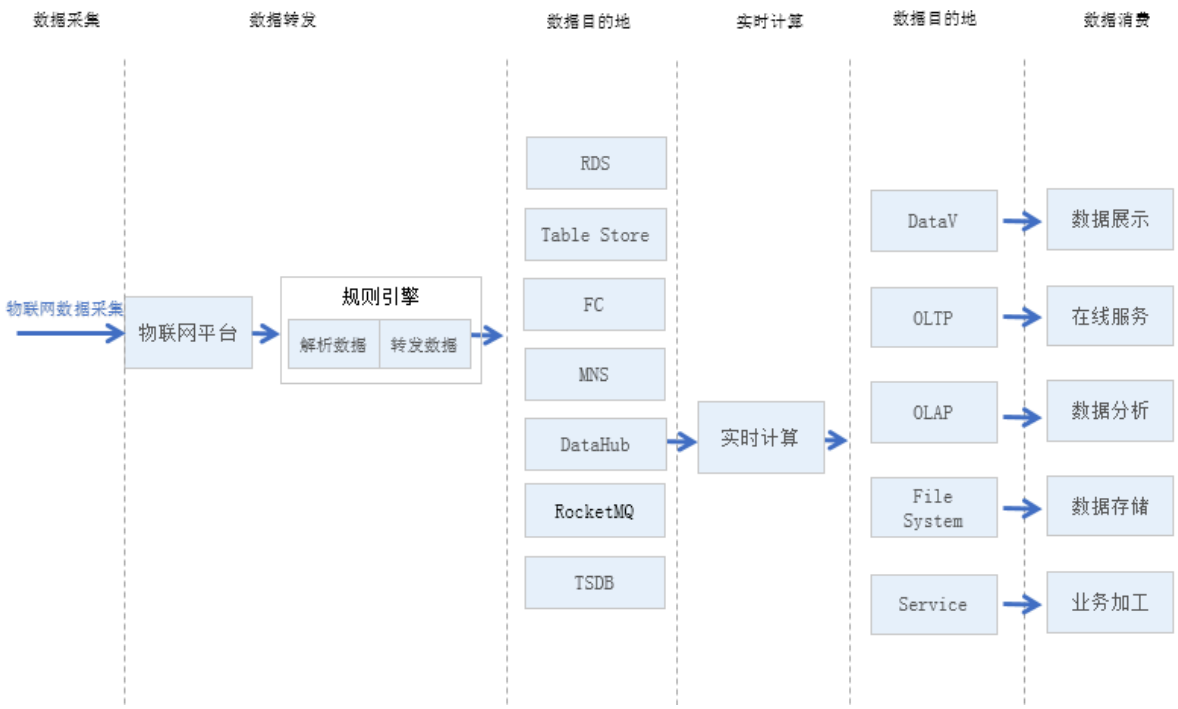
使用物联网平台规则引擎的数据流转功能，可将Topic中的数据消息转发至其他Topic或其他阿里云产品进行存储或处理。

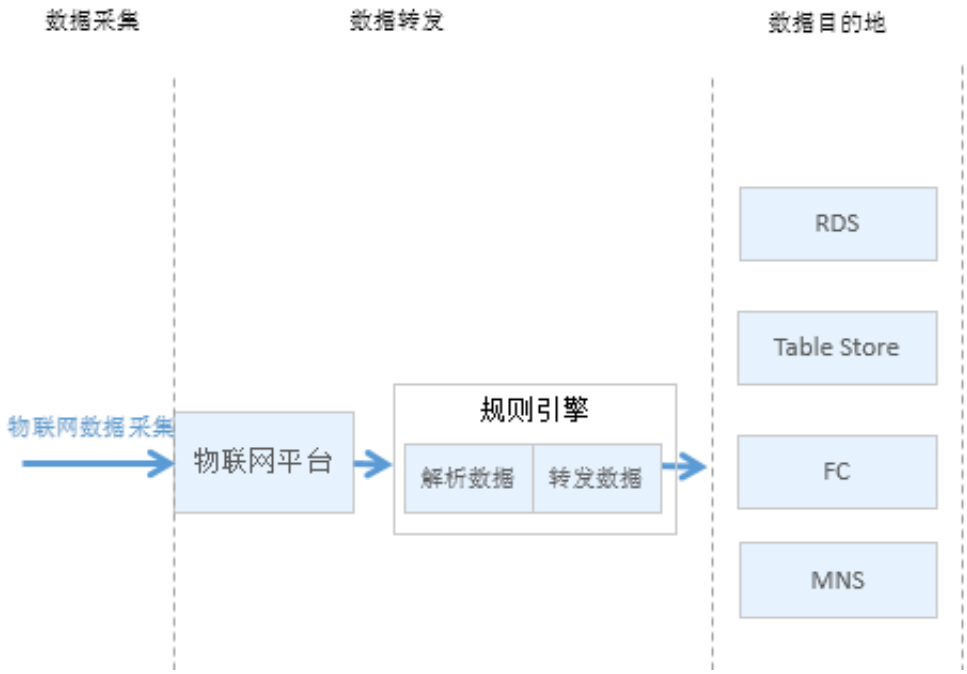
什么是数据流转

当设备基于Topic进行通信时，您可以在规则引擎的数据流转中，编写SQL对Topic中的数据进行处理，并配置转发规则将处理后的数据转发到其他Topic或阿里云其他服务。例如：

- 将数据转发到另一个Topic中以实现M2M通信。
- 将数据转发到RDSRDS、表格存储表格存储、TSDB中进行存储。
- 将数据转发到DataHub中，然后使用实时计算进行流计算，使用Maxcompute进行大规模离线计算。
- 将数据转发到函数计算函数计算进行事件计算。
- 可以转发到消息队列RocketMQ、消息服务消息服务实现高可靠消费数据·。

使用规则引擎的数据流转功能后，您无需购买服务器部署分布式架构，即可实现采集 + 计算 + 存储的全栈服务。





数据流转使用限制

- 数据流转基于Topic对数据进行处理。只有通过Topic进行通信时，才能使用数据流转。
- 通过SQL对Topic中的数据进行处理。
- SQL语法目前不支持子查询。
- 支持部分函数，例如使用`deviceName()`获取当前设备名称，具体函数请参见#unique_78文档。

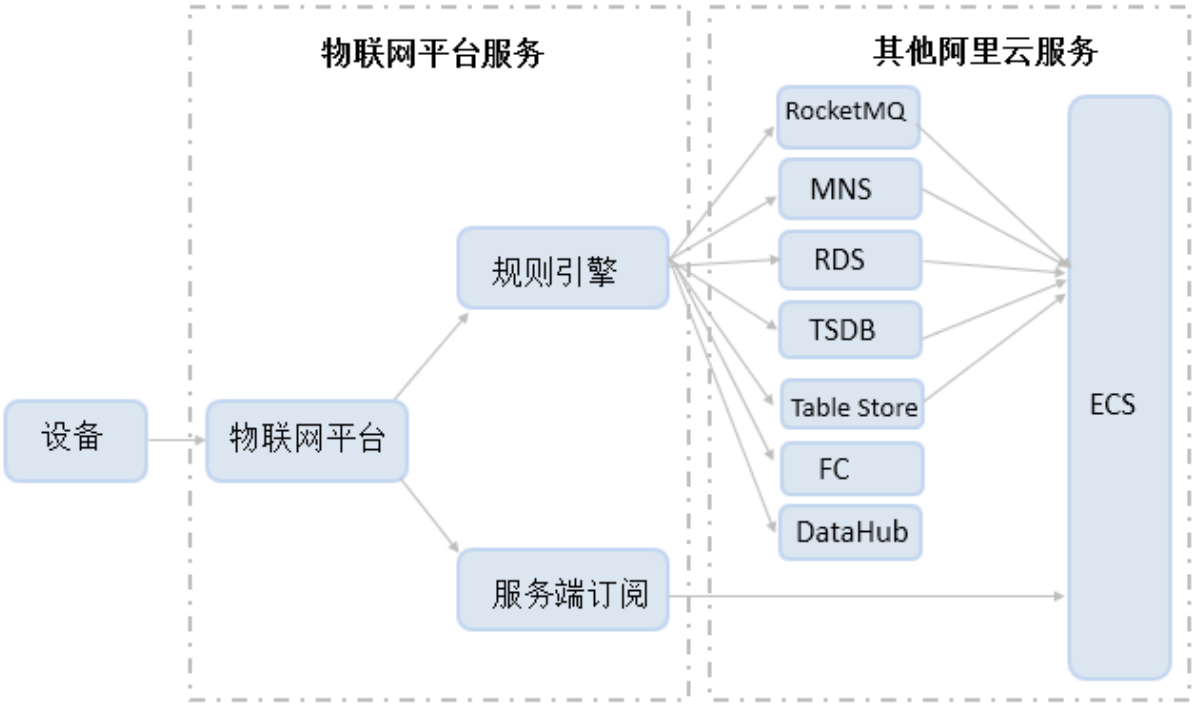
数据流转使用指南

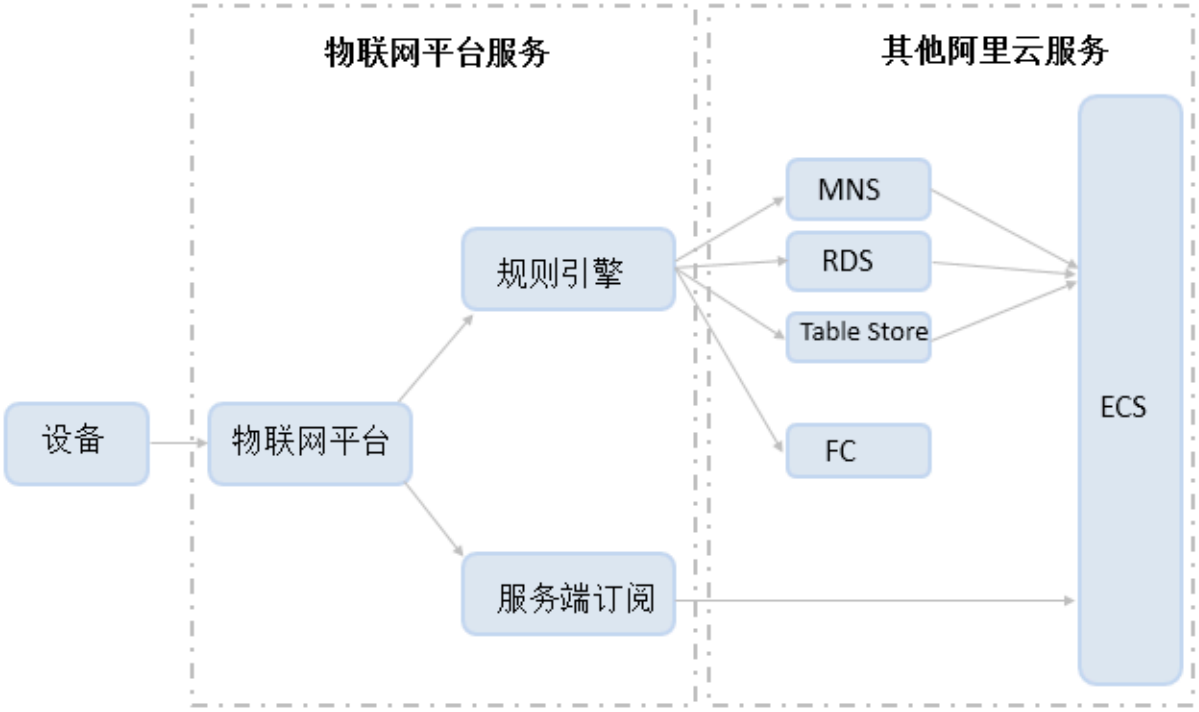
1. #unique_79：如何设置一条转发规则。
2. #unique_80：规则中SQL表达式的写法详解。
3. #unique_78：规则中SQL表达式支持的函数列表。
4. #unique_81：使用规则引擎进行数据流转的过程和各阶段的数据格式。
5. #unique_62：系统Topic消息经物模型解析后的数据格式。数据流转规则中，SQL字段需按照解析后的数据格式编写。
6. #unique_82：查看不同地域和可用区所支持的目的云产品。

2.1.2 数据流转方案对比

在许多场景中，您需要将设备上报给物联网平台的数据进行加工处理或用于业务应用。使用物联网平台提供的服务端订阅功能和规则引擎数据流转功能，均可实现设备数据流转。本文对比物联网平台支持的各流转方案及使用场景，帮助您选择合适的流转方案。

设备数据流转方案汇总





目前，流转方式有两类：

- **规则引擎数据流转：提供初级的数据过滤转换能力。支持对设备数据进行过滤并转换，然后再流转到其他阿里云云产品实例。**
- **服务端订阅：通过HTTP/2客户端直接获取设备消息。可快速地获取设备消息，无消息过滤和转换能力，功能较为单一，但是简单易用且高效。**

规则引擎 vs 服务端订阅

流转方式	适用场景	优缺点	使用限制
规则引擎	· 复杂场景。 · 海量吞吐量场景。	优点： · 功能相对完备。 · 支持在规则运行时，调整流转规则。 · 支持对数据进行简单过滤处理。 · 支持将数据流转至其他阿里云云产品。 数据流转至其他阿里云云产品简要对比，请参见下表：规则引擎各方案对比。 缺点： 需编写SQL和配置规则，使用相对复杂。	请参见规则引擎使用限制。
服务端订阅	· 单纯的接收设备数据的场景。 · 同时满足以下条件的场景： - 云端接收全部设备数据。 - 云端采用Java语言或.NET语言开发。 - 设备数据流转性能要求不超过5000条/秒。	优点： 相对简单易用。 缺点： · 缺少过滤能力。 · 目前仅支持Java和.NET语言SDK。	请参服务端订阅使用限制。

表 2-1: 规则引擎各方案对比

流转目标	适用场景	优点	缺点
消息队列 RocketMQ	要对设备数据进行复杂或精细化处理的海量设备场景。 设备消息量>1000 QPS的场景，推荐使用消息队列RocketMQ。	· 稳定可靠。 · 支持海量数据。	公网支持略差（铂金版公网性能较好）。

流转目标	适用场景	优点	缺点
消息服务（MNS）	公网环境场景下，对设备数据进行复杂或精细化处理。 设备消息量<1000 QPS的场景，推荐使用MNS。	· 采用HTTPS协议。 · 公网支持较好。	性能略低于消息队列RocketMQ。
云数据库RDS版	适合单纯的数据保存场景。	数据直接写入数据库。	-
时序时空数据库（TSDB）	适合根据设备数据进行业务分析和监控的场景。	数据直接写入时序数据库。	-
DataHub	适合需对数据进行分析处理的场景。	数据直接写入DataHub。	-
表格存储（Table Store）	适合单纯的数据存储场景。	数据直接写入表格存储实例。	-
函数计算（Function Compute）	需要简化设备开发过程，且对设备数据进行一定自由度的处理的场景。	· 数据处理自由度高。 · 功能多。 · 无需部署。	费用略高。

服务端订阅

服务端可以通过SDK订阅产品下配置的所有类型的消息，包含设备上报消息，上下线消息等。

使用限制	使用注意	相关文档
· 目前，仅支持Java SDK（需要JDK 8及以上环境）和 .NET SDK，其他语言暂不支持。 · 不支持细粒度的过滤及订阅功能，只能接收租户下全部的消息。 · 目前，流转的消息QPS限制为1000 QPS。如果业务需求超过该限制，需要通过工单申请的方式调整。 使用限制可能更新，详细的服务端订阅使用限制，请参见#unique_55。	· 适合最大消息量< 5000 QPS的流转场景。 · 请务必考虑消息丢失，以及消息延迟到达对业务系统的影响。对于重要的消息，请在业务层做好防护。 · 服务端订阅不能满足需对数据进行高级过滤以及精细化处理的场景，这类场景推荐使用规则引擎。	· 功能简介 · Java SDK开发 · .NET SDK开发 · 最佳实践

规则引擎 + 消息队列 RocketMQ

通过规则引擎数据流转功能，将物联网平台中指定Topic的消息流转到RocketMQ中的Topic，然后通过RocketMQ的SDK接收相应的消息。

- 因为RocketMQ性能突出，推荐通过RocketMQ接收设备消息。
- RocketMQ支持跨租户的Topic授权，可满足一定场景的跨租户的数据流转需求。

使用限制	使用注意	相关文档
· RocketMQ的公网测试集群不能用于生产环境。 · RocketMQ的非公网endpoint无法用于本地调试，需要在该地域的ECS中运行调试。	· RocketMQ的公网测试集群可以支持开发者本地接收消息，但是稳定性很差，请勿用于线上生产环境。 · 对于较大的设备消息量场景（> 5000 QPS）或者稳定性要求特别高的场景，推荐使用RocketMQ铂金版。 · 规则引擎数据流转失败，然后再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的影响防护。	· #unique_79 · #unique_87 · #unique_88 · RocketMQ使用指南

规则引擎 + 消息服务（MNS）

可以通过规则引擎数据流转功能，将指定Topic的消息流转到MNS的主题中，然后通过MNS的SDK接收相应的消息。MNS对公网环境支持友好。设备消息量不是特别大（< 1000QPS），推荐使用MNS。

使用限制	使用注意	相关文档
请参见消息服务使用限制文档。	规则引擎数据流转失败后，再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的防护。	· #unique_79 · #unique_89 · MNS使用指南 · MNS使用指南

规则引擎 + 函数计算

通过规则引擎数据流转，将指定Topic的消息转入到函数计算中，开发者可以进一步对消息进行处理。函数计算免部署，可以简化业务的开发。

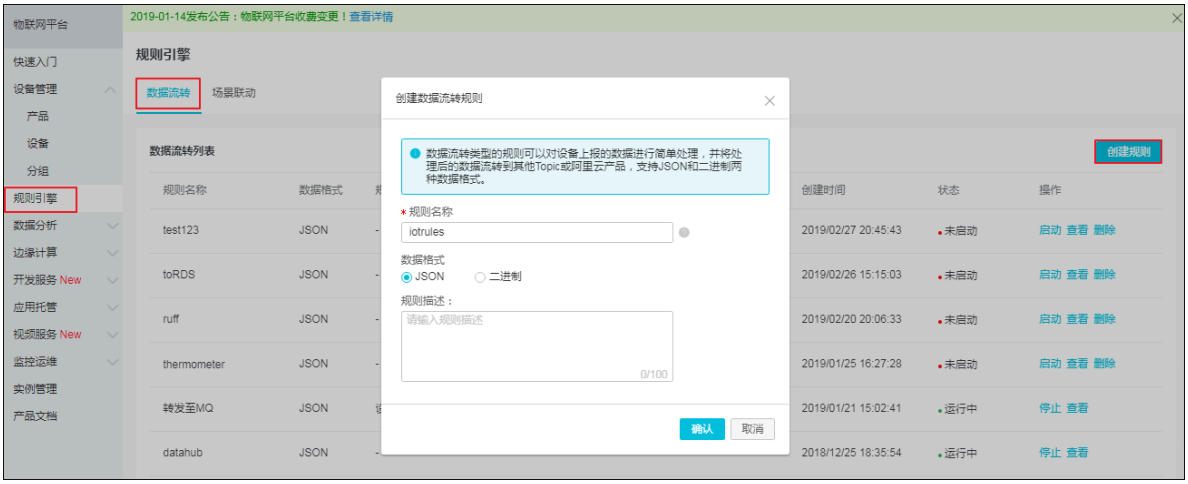
使用限制	使用注意	相关文档
请参见函数计算使用限制函数计算使用限制。	· 适用于对于设备消息有定制化的处理需求或者需简化开发运维的场景。 · 规则引擎数据流转失败后，再重试失败数次后，会丢弃消息。业务场景一定要做好消息丢失或者延迟送达的影响防护。	· #unique_79 · #unique_90 · 实践案例（温湿度计上报数据到钉钉群机器人） · 函数计算使用指南 · 函数计算使用指南

2.1.3 设置数据流转规则

通过规则引擎的数据流转功能，物联网平台可将指定Topic的数据流转至其他Topic和其他阿里云服务中。本文将为您详细讲解如何设置一条完整的数据流转规则。设置过程依次是创建规则、编写处理数据的SQL、设置数据流转目的地和设置流转失败的数据转发目的地。

操作步骤

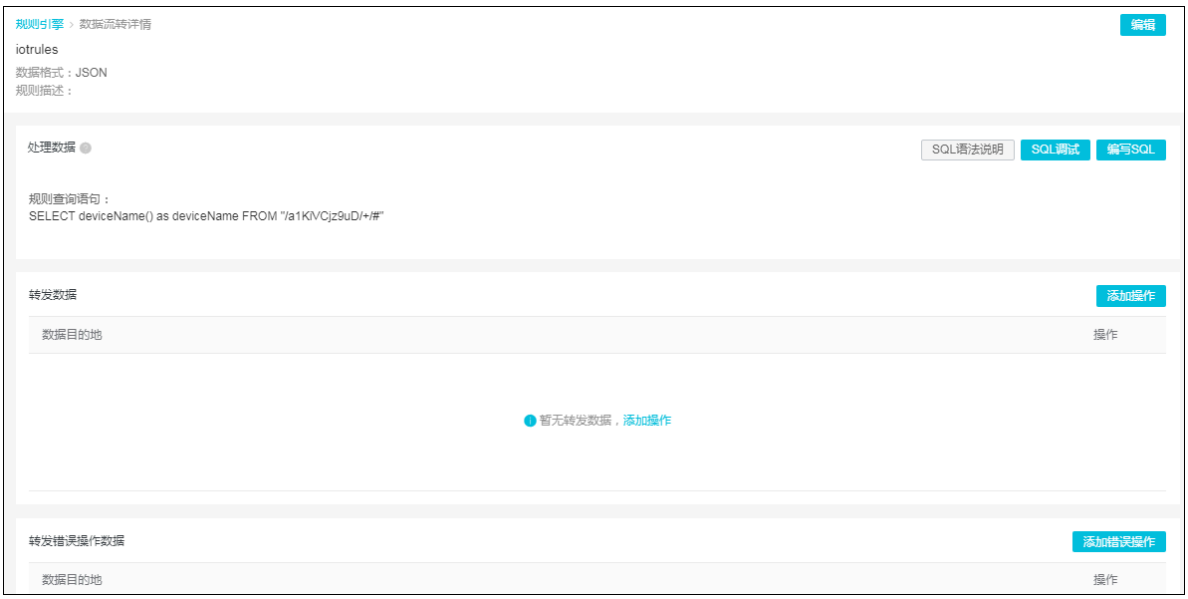
1. 在物联网平台控制台左侧导航栏，选择规则引擎。
2. 在数据流转页签栏下，单击创建规则。
3. 填写规则名称，选择数据格式后，单击确认。



参数	描述
规则名称	输入规则名称，用以区别各条规则。名称支持中文汉字、英文字母、数字、下划线和连字符。长度为1-30字节。一个中文汉字占二字节。

参数	描述
数据格式	选择该规则处理数据的格式。可选：JSON和二进制。 说明: · 因数据流转基于Topic处理数据，此处的数据格式需与被处理Topic中的数据格式保持一致。 · 若选择为二进制，该规则不能处理系统Topic的消息，且不能将数据转发至表格存储、时序时空数据库和云数据库RDS版。 · 若选择为二进制，该规则不能处理系统Topic的消息，且不能将数据转发至表格存储和云数据库RDS版。
规则描述	规则描述信息。

4. 规则创建成功后，页面将跳转到数据流转详情页。您需编辑处理消息数据的SQL、设置数据转发目的地和设置错误数据转发目的地。



a) 单击编写SQL，编写筛选消息字段的SQL。

例如：以下示例SQL可以从test1128产品下，全部设备的所有Topic消息中，取出deviceName字段内容。

说明:

二进制数据可使用`to_base64(*)`将原始数据转换成base64String，同时支持内置函数和条件筛选。

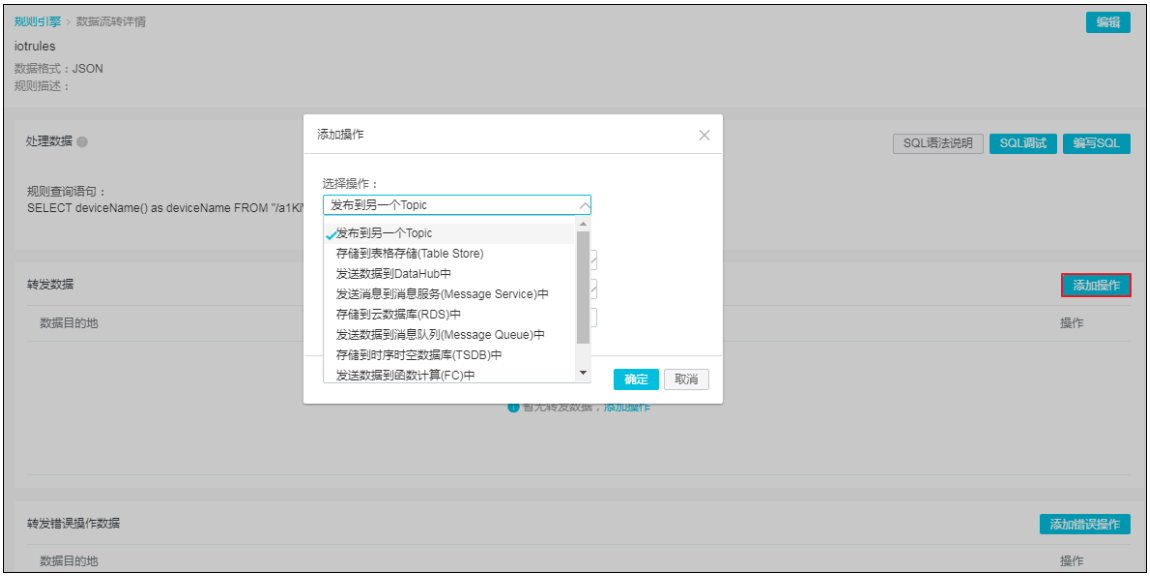
参数设置解释如下，具体可参考#unique_80和#unique_78。

参数	描述
规则查询语句	系统会在这里，根据您设置的字段、Topic和条件自动补充完整规则查询语句。
字段	指定要处理的消息内容字段。如，填入`deviceName() as deviceName`，则表示需筛选出消息中的deviceName字段内容。 有关消息内容数据格式，请参见数据格式。

参数	描述
Topic	选择需要处理的消息来源Topic。 可选类型： · 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(+)：表示指定产品下所有设备。 - /user/#：表示指定设备的所有Topic。 自定义Topic和通配符使用，请参见自定义Topic。 · 系统：规则数据类型是JSON时的可选项。系统Topic消息包含：设备上报属性和事件消息、设备生命周期变更消息、设备拓扑关系变更消息和网关发现子设备消息。相关消息格式，请参见数据格式。 支持使用通配符+代表指定产品下的所有设备。 · 设备状态：规则数据类型是JSON时的可选项。表示设备上下线状态变更消息Topic。设备上下线消息格式，请参见数据格式。
条件	设置规则触发条件。

b) 单击转发数据一栏的添加操作，将数据转发至目的云产品。数据转发的具体实例，请参考使用实例目录下的具体文档。

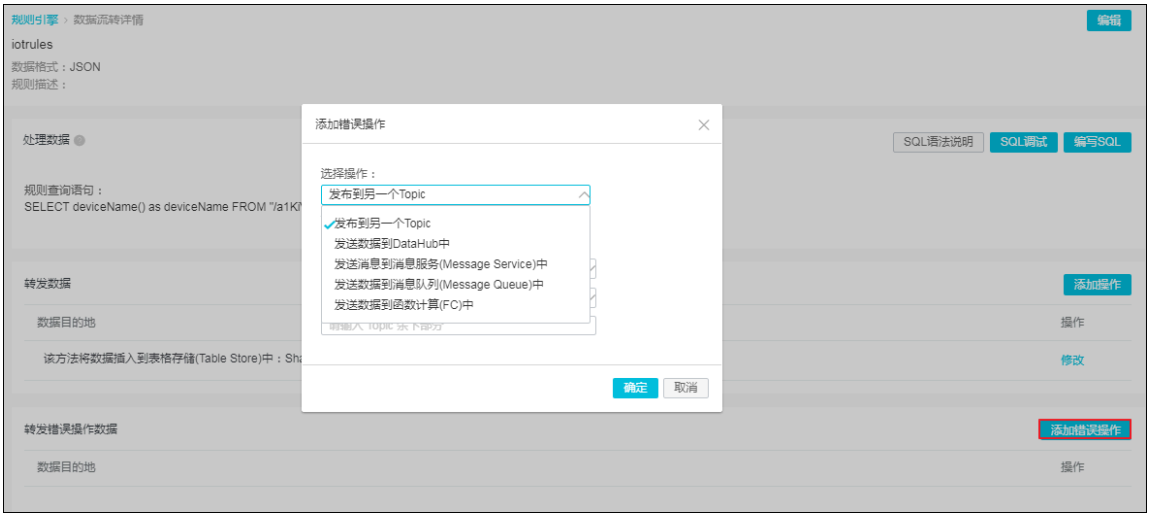
> 说明：
> 最多可为一个规则创建10个数据转发操作。



数据转发时，因选择的目的云产品出现异常情况导致转发失败，物联网平台会间隔1秒、3秒、10秒进行3次重试（重试策略可能会调整）。3次重试均失败后，消息会被丢弃。如果您

对消息可靠性要求比较高，可以进行下一步：添加错误操作。重试失败的消息可以通过错误操作转发到其它云产品中。

c) 单击转发错误数据一栏的添加错误操作，设置参数，将重试失败的错误消息转发至指定位置。



说明:

- 正常操作和错误操作的转发目的地云产品不能相同。比如不能同时转发到表格存储。
- 消息转发至云产品失败后，会进行重试。若重试失败，将根据错误操作数据转发的设置转发错误消息。
- 错误消息转发失败后，不会再进行重试。
- 这里的错误消息仅针对因其他云产品实例问题导致的规则引擎转发失败错误。
- 最多支持添加一个错误操作。
- 错误消息格式：

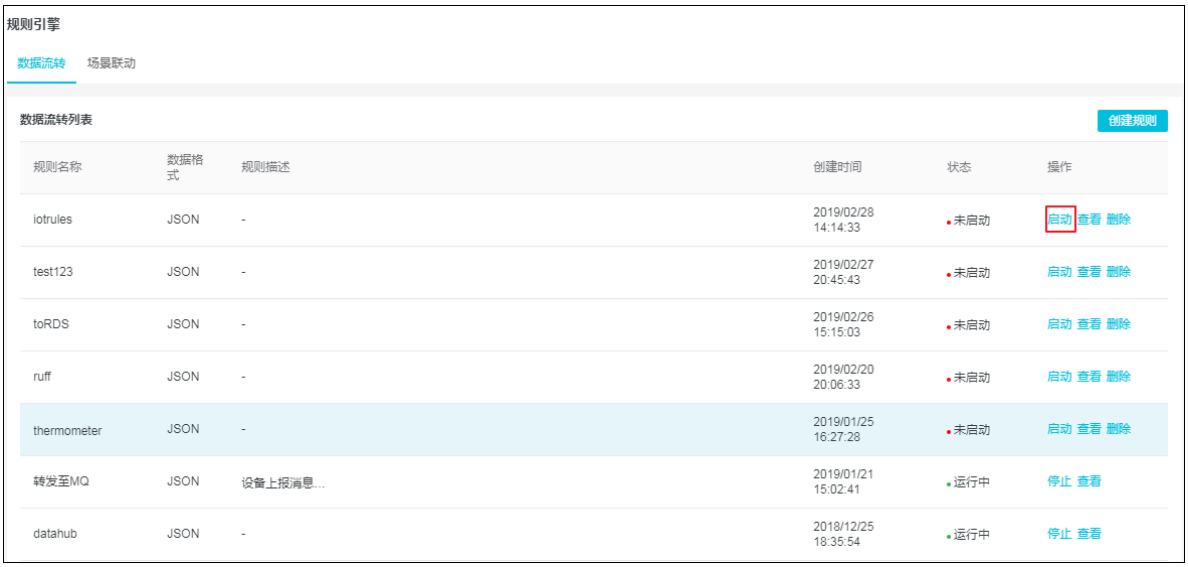
```
{
      "ruleName":"",
      "topic":"",
      "productKey":"",
      "deviceName":"",
      "messageId":"",
      "base64OriginalPayload":"",
      "failures":[
        {
          "actionType":"OTS",
          "actionRegion":"cn-shanghai",
          "actionResource":"table1",
          "errorMessage":""
        },
        {
          "actionType":"RDS",
          "actionRegion":"cn-shanghai",
          "actionResource":"instance1/table1",
          "errorMessage":""
        }
```

```
        ]
}
```

错误消息参数说明如下：

参数	说明
ruleName	规则名称。
topic	消息来源Topic。
productKey	产品Key。
deviceName	设备名称。
messageId	云端消息ID。
base64OriginalPayload	Base64编码后的原始数据。
failures	错误详情。可能会有多个。
actionType	出错操作的类型。
actionRegion	出错操作的地域。
actionResource	出错操作的目的资源。
errorMessage	错误信息。

5. 所有设置完成后，返回至规则引擎的数据流转页签栏，单击规则对应的启动按钮。规则启动后，数据即可按照规则进行转发。



您也可以：

- 单击查看进入数据流转详情页，修改规则的具体设置。
- 单击删除删除对应规则。

> 说明：
> 运行中的规则不可删除。

- 单击停止停止对应规则转发数据。

2.1.4 SQL表达式

创建数据流转规则时，需编写SQL来解析和处理设备上报的数据。二进制格式的数据不做解析，直接透传。本文主要介绍如何编写数据流转规则的SQL表达式。

SQL表达式

JSON数据可以映射为虚拟的表，其中Key对应表的列，Value对应列值，因此可以使用SQL来进行数据处理。数据流转规则的SQL表达示意图如下：

数据流转规则SQL示例：

- **处理自定义Topic数据的SQL示例。**

  某环境传感器可以采集温度、湿度及气压数据。设备上报到自定义Topic：/a1hRrzD**/+/user/update的数据如下：**

```
{
"temperature":25.1
"humidity":65
"pressure":101.5
"location":"xxx,xxx"
}
```

  设置温度大于38时触发规则，并筛选出设备名称、温度和位置信息，SQL语句如下：

```
SELECT temperature as t, deviceName() as deviceName, location FROM
 "/a1hRrzD****/+/user/update" WHERE temperature > 38
```

- **处理系统Topic数据的SQL示例。 系统Topic中的数据流转到规则引擎后，规则引擎会对数据进行解析，解析后的数据格式请参见#unique_93。**

  某温湿度传感器的属性如下：

功能类型	功能名称	标识符	数据类型	数据定义
属性	当前温度	CurrentTemperature	float (单精度浮点型)	取值范围：-40 ~ 120
属性	当前湿度	CurrentHumidity	float (单精度浮点型)	取值范围：0 ~ 100

  温湿度传感器上报的温度和湿度属性数据经规则引擎解析后，数据格式如下：

```
{
    "deviceType": "TemperatureHumidityDetector",
    "iotId": "N5KURkKdibnZvSls****000100",
    "productKey": "a15NNfl****",
```

```
    "gmtCreate": 1564569974224,
    "deviceName": "N5KURkKdibnZvSls3Yfx",
    "items": {
        "CurrentHumidity": {
            "value": 70,
            "time": 1564569974229
        },
        "CurrentTemperature": {
            "value": 23.5,
            "time": 1564569974229
        }
    }
}
```

设置温度大于38时触发规则，并筛选出设备名称、当前温度和当前湿度，SQL语句如下：

```
SELECT deviceName() as deviceName, items.CurrentHumidity.value as
 Humidity, items.CurrentTemperature.value as Temperature FROM "/
sysa15NNfl****/N5KURkKdibnZvSls3Yfx/thing/event/property/post" WHERE
 items.CurrentTemperature.value > 38
```

SELECT

- JSON数据格式

  SELECT语句中的字段，可以使用上报消息的payload解析结果，即JSON中的键值，也可以使用SQL内置的函数，如`deviceName()`。规则引擎内置的SQL函数，请参见#unique_78。

  支持`*`和函数的组合。不支持子SQL查询。

  上报的JSON数据格式，可以是数组或者嵌套的JSON。SQL语句支持使用JSONPath获取其中的属性值，如对于`{a:{key1:v1, key2:v2}}`，可以通过`a.key2`获取到值`v2`。使用变量时，需要注意单双引号区别：单引号表示常量，双引号或不加引号表示变量。如使用单引号`'a.key2'`，值为`a.key2`。

  以上SQL示例中，

  - 处理自定义Topic数据SQL的SELECT语句`SELECT temperature as t, deviceName() as deviceName, location`中，`temperature`和`loaction`来自于上报数据中的字段，`deviceName()`则使用了内置的SQL函数。
  - 处理上报属性Topic数据SQL的SELECT语句`SELECT deviceName() as deviceName, items.CurrentHumidity.value as Humidity, items.CurrentTemperature.value as Temperature`中，`items.CurrentHumidity.value`和`items.CurrentTemperature.value`来自于上报的属性数据中的字段，`deviceName()`则使用了内置的SQL函数。

- 二进制数据格式
  - 可填`*`直接透传数据。`*`后不能再使用函数。
  - 可使用内置函数，如`to_base64(*)`函数，将原始Payload二进制数据转成base64String提取出来；`deviceName()`函数，将设备名称信息提取出来。

说明:
SELECT语句中最多可包含50个字段。

FROM

FROM 可以填写为Topic，用于匹配需要处理的设备消息来源Topic。Topic中的设备名（deviceName）类目可以填写为通配符+，代表当前层级所有类目，即产品下的所有设备；指定为自定义Topic时，还可以使用统配符#，代表Topic中当前层级及之后的所有类目。通配符说明，请参见#unique_94。

当来自指定Topic的消息到达时，消息的payload数据以JSON格式解析，并根据SQL语句进行处理。如果消息格式不合法，将忽略此条消息。您可以使用`topic()`函数引用具体的Topic值。

以上SQL示例中，

- `FROM "/a1hRrzD****/+/user/update"`语句表示该SQL仅处理自定义Topic：/a1hRrzD****/+/user/update的消息。
- `FROM "/sys/a15NNfl****/N5KURkKdibnZvSls3Yfx/thing/event/property/post"`语句表示该SQL仅处理设备N5KURkKdibnZvSls3Yfx上报属性的Topic中的消息。

WHERE

- JSON数据格式

  规则触发条件，条件表达式。不支持子SQL查询。WHERE中可以使用的字段和SELECT语句一致，当接收到对应Topic的消息时，WHERE语句的结果会作为是否触发规则的判断条件。具体条件表达式，请参见以下章节：条件表达式支持列表。

  上文两个示例中，条件语句`WHERE temperature > 38`表示温度大于38时，才会触发该规则。
- 二进制数据格式

  目前二进制格式WHERE语句中，仅支持内置函数及条件表达式，无法使用payload中的字段。

SQL结果

SQL语句执行完成后，会得到对应的SQL结果，用于下一步转发处理。如果payload数据解析过程中出错，会导致规则运行失败。

如果要将数据流转到表格存储Table Store中，设置转发数据目的地时，需要使用变量格式 `${表达式}` 引用对应的值。

如果以上两个示例SQL对应的规则需将数据流转到表格存储的数据表中，主键对应的值可分别配置为

- *${t}*、*${deviceName}*和*${loaction}*。
- *${deviceName}*、*${Humidity}*和*${Temperature}*。

数组使用说明

数组表达式需要使用双引号，用`$.`表示取jsonObject，`$.`可以省略，`.`表示取jsonArray。

例如设备消息为：`{"a":[{"v":0},{"v":1},{"v":2}]}`，不同表达式结果如下：

- `"a[0]"` 结果为 `{"v":0}`
- `"$.a[0]"` 结果为`{"v":0}`
- `".a[0]"` 结果为 `[{"v":0}]`
- `"a[1].v"`结果为`1`
- `"$.a[1].v"`结果为`1`
- `".a[1].v"` 结果为`[1]`

条件表达式支持列表

操作符	描述	举例
=	相等	color = ‘red’
<>	不等于	color <> ‘red’
AND	逻辑与	color = ‘red’ AND siren = ‘on’
OR	逻辑或	color = ‘red’ OR siren = ‘on’
( )	括号代表一个整体	color = ‘red’ AND (siren = ‘on’ OR isTest)
+	算术加法	4 + 5
-	算术减	5 - 4

/	除	20 / 4
*	乘	5 * 4
%	取余数	20 % 6
<	小于	5 < 6
<=	小于或等于	5 <= 6
>	大于	6 > 5
>=	大于或等于	6 >= 5
函数调用	支持函数，详细列表请参见#unique_95。	deviceId()
JSON属性表达式	可以从消息payload以JSON表达式提取属性。	state.desired.color,a.b.c[0].d
CASE … WHEN … THEN … ELSE … END	Case 表达式（不支持嵌套）	CASE col WHEN 1 THEN ‘Y’ WHEN 0 THEN ‘N’ ELSE ‘’ END as flag
IN	仅支持枚举，不支持子查询。	例如， where a in(1,2,3)。不支持以下形式： where a in(select xxx)
like	匹配某个字符， 仅支持%通配符，代表匹配任意字符串。	例如， where c1 like ‘%abc’ , where c1 not like ‘%def%’

2.1.5 函数列表

规则引擎提供多种函数，您可以在编写SQL时使用这些函数，实现多样化数据处理。

规则引擎支持的函数如下表。

函数名	函数说明
abs(number)	返回绝对值。
asin(number)	返回number值的反正弦。
attribute(key)	返回key所对应的设备标签。如果设备没有该key对应的标签，则返回值为空。使用SQL调试时，因为没有真实设备及对应的标签，返回值为空。
concat(string1, string2)	用于连接字符串。返回连接后的字符串。 示例：`concat(field,'a')`。
cos(number)	返回number值的余弦。
cosh(number)	返回number值的双曲余弦（hyperbolic cosine）。

函数名	函数说明
crypto(field,String)	对field的值进行加密。 第二个参数String为算法字符串。可选：MD2、MD5、SHA1、SHA-256、SHA-384、SHA-512。
deviceName()	返回当前设备名称。使用SQL调试时，因为没有真实设备，返回值为空。
endswith(input, suffix)	判断input的值是否以suffix结尾。
exp(number)	返回指定数字的指定次幂。
floor(number)	返回一个最接近它的整数，它的值小于或等于这个浮点数。
log(n, m)	返回自然对数。 如果不传m值，则返回log(n)。
lower(string)	返回小写字符串。
mod(n, m)	n%m 余数。
nanvl(value, default)	返回属性值。 若属性值为null，则返回default。
newuuid()	返回一个随机UUID字符串。
payload(textEncoding)	返回设备发布消息的payload转字符串。 字符编码默认UTF-8，即 payload()默认等价于payload（‘utf-8’）。
power(n,m)	返回n的m次幂。
rand()	返回[0~1)之间的一个随机数。
replace(source, substring, replacement)	对某个目标列值进行替换，即用replacement替换resouce中的substring。 示例：replace(field,’a’,’1’)。
sin(n)	返回n值的正弦。
sinh(n)	返回n值的双曲正弦（hyperbolic sine）。
tan(n)	返回n值的正切。
tanh(n)	返回n值的双曲正切（hyperbolic tangent）。

函数名	函数说明
timestamp(format)	返回当前系统时间，格式化后返回当前地域的时间。 format为可选。如果为空，则返回当前系统时间戳毫秒值，例如，使用`timestamp()`，返回的时间戳为`1543373798943`；如果指定format，则根据指定格式，返回当前系统时间，如`timestamp('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'')`，返回的时间戳为`2018-11-28T10:56:38Z`。
timestamp_utc(format)	返回当前系统时间，格式化后返回UTC时间。 format为可选。如果format为空，则返回当前系统时间戳毫秒值，例如，使用`timestamp_utc()`，返回的时间戳为`1543373798943`；使用`timestamp_utc('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'')`，返回的时间戳为`2018-11-28T02:56:38Z`。
topic(number)	返回Topic分段信息。 如，有一个Topic： /alDbcLe****/TestDevice/user/set。使用函数`topic()`，则返回整个Topic `/alDbcLe****/TestDevice/user/set`； 使用`topic(1)`，则返回Topic的第一级类目`alDbcLe****`；使用`topic(2)`，则返回`TestDevice`。
upper(string)	将字符串中的小写字母转为大写字母。 示例：函数`upper(alibaba)`的返回结果是`ALIBABA`
to_base64(*)	当原始Payload数据为二进制数据时，可使用该函数，将所有二进制数据转换成base64String。
messageId()	返回物联网平台生成的消息ID。

函数名	函数说明
substring(target, start , end)	返回从start（包括）到end(不包括）的字符串。 · tartget：要操作的字符串。 必填。 · start：起始下标。必填。 · end：结束下标。非必填。 说明: · 目前，仅支持String和Int类型数据。Int类型数据会被转换成String类型后再处理。 · 常量字符串，请使用单引号。双引号中的数据，将视为Int类型数据进行解析。 · 如果传入的参数错误，例如参数类型不支持，会导致SQL解析失败，而放弃执行规则。 字符串截取示例： · substring('012345', 0) = "012345" · substring('012345', 2) = "2345" · substring('012345', 2.745) = "2345" · substring(123, 2) = "3" · substring('012345', -1) = "012345" · substring(true, 1.2) error · substring('012345', 1, 3) = "12" · substring('012345', -50, 50) = "012345" · substring('012345', 3, 1) = ""

使用示例

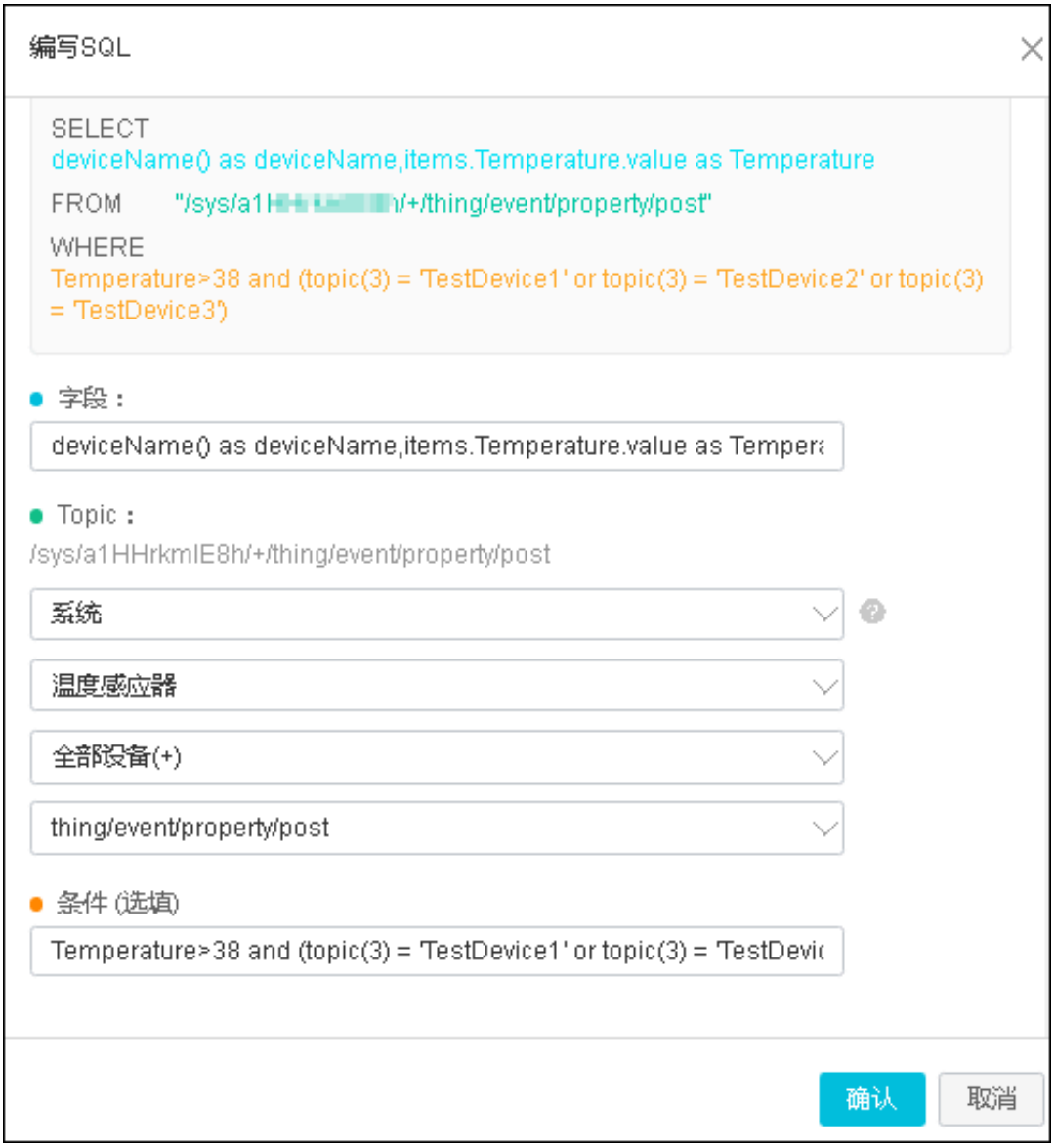
您可以在数据流转SQL语句的SELECT字段和WHERE条件字段中，使用函数获取数据或者对数据做处理。

例如，某温度感应器产品有一个温度属性（Temperature）。设备上报的温度属性数据经物模型转化后的数据格式如下。

```
{
    "deviceType": "Custom",
    "iotId": "H5KURkKdibnZvSls****000100",
    "productKey": "a1HHrkm****",
    "gmtCreate": 1564569974224,
    "deviceName": "TestDevice1",
    "items": {
        "Temperature": {
            "value": 23.5,
            "time": 1564569974229
        }
    }
```

```
}
```

该温度感应器产品下有多个设备，但只有当设备TestDevice1、TestDevice2或TestDevice3上报的温度大于38时，需将温度属性数据流转到函数计算中进行计算处理。设置筛选设备上报数据的规则SQL如下图。



SQL语句：

```
SELECT deviceName() as deviceName,items.Temperature.value as
Temperature FROM "/sys/a1HHrkmIE8h/+/thing/event/property/post" WHERE
Temperature>38 and (topic(3) = 'TestDevice1' or topic(3) = 'TestDevice
2' or topic(3) = 'TestDevice3')
```

以上示例中，使用了以下两个函数。

- **deviceName()：在SELECT字段使用该函数，表示从数据中筛选出设备名称。**

- topic(number)：在条件字段使用该函数，并指定为`(topic(3) = 'TestDevice1' or topic(3) = 'TestDevice2' or topic(3) = 'TestDevice3')`。其中，`topic(3)`表示Topic：/sys/a1HHrkmIE8h/+/thing/event/property/post的第三级类目，即设备名称类目。通配符加号（+）表示该产品下所有的设备名称。本示例中，仅当设备名称类目的值为TestDevice1、TestDevice2或TestDevice2中的其中一个时，才会进行数据流转。

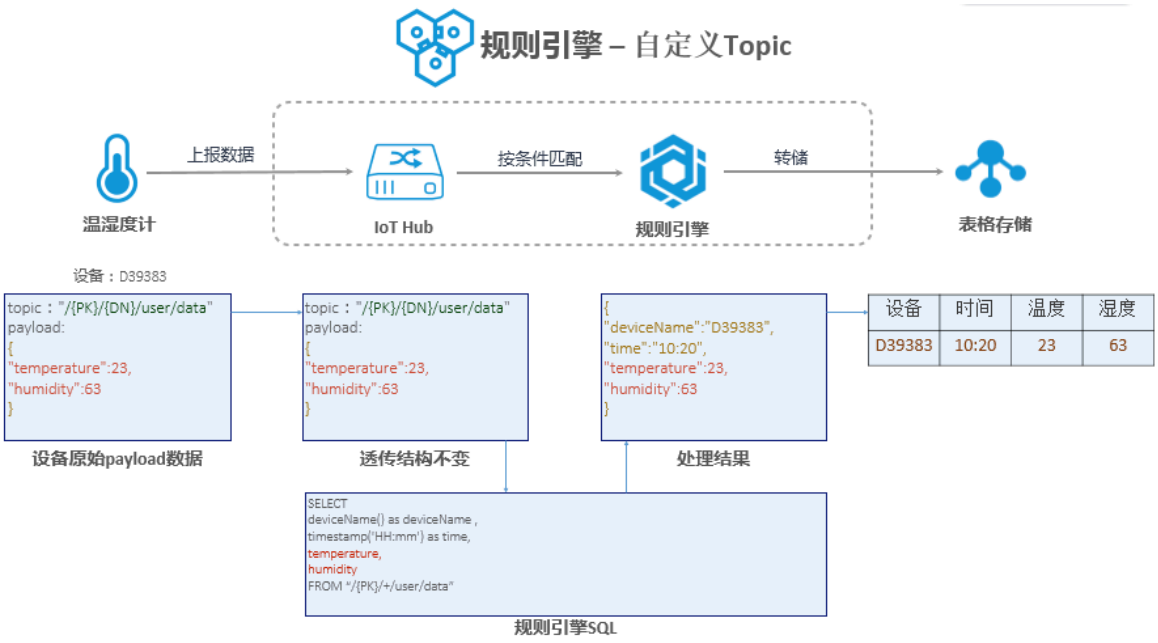
规则SQL中，SELECT字段和WHERE条件字段的具体编写方法，和规则引擎支持的条件表达式列表，请参见SQL表达式。

2.1.6 数据流转过程

规则引擎数据流转仅能处理发送至Topic的数据。本文讲解使用数据流转时，数据的流转过程和不同阶段的数据格式。

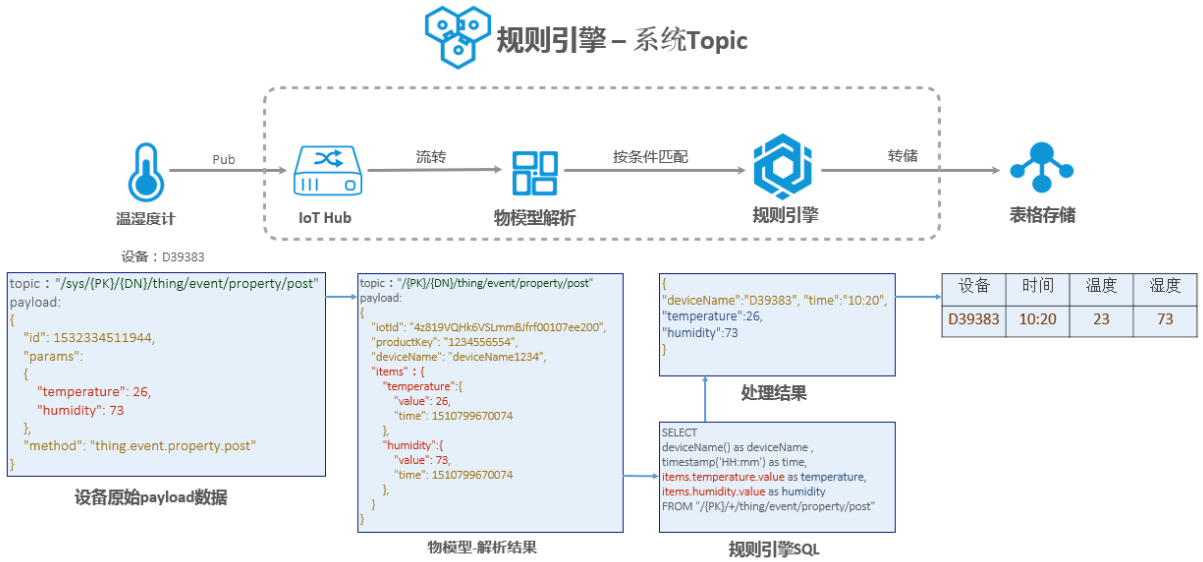
自定义Topic

自定义Topic中的设备数据直接透传至物联网平台，数据结构不变。数据流转示例图如下：



系统Topic

系统Topic中的数据为Alink JSON格式，数据流转时，SQL处理的是经物模型解析后的数据，具体数据格式请参见#unique_62。数据流转示例图如下：



说明:

上图示例中，payload中参数`params`，物模型解析之后，在数据流转中转变为参数`items`。

2.1.7 数据格式

使用规则引擎，您需要基于Topic编写SQL处理数据。自定义Topic中数据格式由您自己定义，物联网平台不做处理。系统Topic中数据格式由物联网平台定义，此时您需要根据平台定义的数据格式处理数据。本文讲述了系统Topic中的数据格式。

设备上下线状态

通过该Topic获取设备的上下线状态。

数据流转Topic：`/as/mqtt/status/{productKey}/{deviceName}`

数据格式：

```
{
    "status":"online|offline",
    "productKey":"12345565569",
    "deviceName":"deviceName1234",
    "time":"2018-08-31 15:32:28.205",
    "utcTime":"2018-08-31T07:32:28.205Z",
    "lastTime":"2018-08-31 15:32:28.195",
    "utcLastTime":"2018-08-31T07:32:28.195Z",
    "clientIp":"123.123.123.123"
}
```

参数说明：

参数	类型	说明
status	String	设备状态，`online`：上线；`offline`：离线。

参数	类型	说明
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
time	String	发送通知的时间点。
utcTime	String	发送通知的UTC时间点。
lastTime	String	状态变更前最后一次通信的时间。  说明: 为避免消息时序紊乱造成影响，建议您根据lastTime来维护最终设备状态。
utcLastTime	String	状态变更前最后一次通信的UTC时间。
clientIp	String	设备公网出口IP。

设备属性上报

通过该Topic获取设备上报的属性信息。

Topic：`/sys/{productKey}/{deviceName}/thing/event/property/post`

数据格式：

```
{
    "iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "productKey":"1234556554",
    "deviceName":"deviceName1234",
    "gmtCreate":1510799670074,
    "deviceType":"Ammeter",
    "items":{
        "Power":{
            "value":"on",
            "time":1510799670074
        },
        "Position":{
            "time":1510292697470,
            "value":{
                "latitude":39.9,
                "longitude":116.38
            }
        }
    }
}
```

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识。
productKey	String	设备所属产品的唯一标识。

参数	类型	说明
deviceName	String	设备名称。
deviceType	String	设备类型。
items	Object	设备数据。
Power	String	属性名称。产品所具有的属性名称请参见产品的TSL描述。
Position	String	属性名称。产品所具有的属性名称请参见产品的TSL描述。
value	根据TSL定义	属性值。
time	Long	属性产生时间，如果设备没有上报默认采用云端生成时间。
gmtCreate	Long	数据流转消息产生时间。

设备事件上报

通过该topic获取设备上报的事件信息。

Topic：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`

数据格式：

```
{
    "identifier":"BrokenInfo",
    "name":"损坏率上报",
    "type":"info",
    "iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "productKey":"X5eCzh6fEH7",
    "deviceName":"5gJtxDVeGAkaEztpisjX",
    "gmtCreate":1510799670074,
    "value":{
        "Power":"on",
        "Position":{
            "latitude":39.9,
            "longitude":116.38
        }
    },
    "time":1510799670074
}
```

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识。
productKey	String	设备所属产品的唯一标识。

参数	类型	说明
deviceName	String	设备名称。
type	String	事件类型，事件类型参见产品的TSL描述。
value	Object	事件的参数。
Power	String	事件参数名称。
Position	String	事件参数名称。
time	Long	事件产生时间，如果设备没有上报默认采用远端时间。
gmtCreate	Long	数据流转消息产生时间。

设备生命周期变更

通过该Topic获得设备创建、删除、禁用、启用等消息。

Topic：`/sys/{productKey}/{deviceName}/thing/lifecycle`

数据格式：

```
{
    "action" : "create|delete|enable|disable",
    "iotId" : "4z819VQHk6VSLmmxxxxxxxxxxee200",
    "productKey" : "X5eCxxxxEH7",
    "deviceName" : "5gJtxDVeGAkaEztpisjX",
    "deviceSecret" : "",
    "messageCreateTime": 1510292739881
}
```

参数说明：

参数	类型	说明
action	String	· create：创建设备。 · delete：删除设备。 · enable：启用设备。 · disable：禁用设备。
iotId	String	设备在平台内的唯一标识。
productKey	String	产品的唯一标识。
deviceName	String	设备名称。
deviceSecret	String	设备密钥，仅在`action`为`create`时包含。
messageCreateTime	Integer	消息产生时间戳，单位毫秒。

设备拓扑关系变更

通过该Topic获得子设备和网关之间拓扑关系建立和解除信息。

Topic：`/sys/{productKey}/{deviceName}/thing/topo/lifecycle`

数据格式：

```
{
    "action" : "add|remove|enable|disable",
    "gwIotId": "4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey": "1234556554",
    "gwDeviceName": "deviceName1234",
    "devices": [
        {
          "iotId": "4z819VQHk6VSLmmxxxxxxxxxxee201",
          "productKey": "1234xxxx569",
          "deviceName": "deviceName1234"
       }
    ],

    "messageCreateTime": 1510292739881
}
```

参数说明：

参数	类型	说明
action	String	· add：新增拓扑关系。 · remove：移除拓扑关系。 · enable：启用拓扑关系。 · disable：禁用拓扑关系。
gwIotId	String	网关设备在平台内的唯一标识。
gwProductKey	String	网关产品的唯一标识。
gwDeviceName	String	网关设备名称。
devices	Object	变更的子设备列表。
iotId	String	子设备在平台内的唯一标识。
productKey	String	子设备产品的唯一标识。
deviceName	String	子设备名称
messageCreateTime	Integer	消息产生时间戳，单位毫秒。

网关发现子设备

在一些场景中网关能够检测到子设备，并将检测到的子设备信息上报。此时可以通过该Topic获取到上报的信息。

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

数据格式：

```
{
    "gwIotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey":"1234556554",
    "gwDeviceName":"deviceName1234",
    "devices":[
        {
            "iotId":"4z819VQHk6VSLmmxxxxxxxxxxee201",
            "productKey":"1234xxxx569",
            "deviceName":"deviceName1234"
        }
    ]
}
```

参数说明：

参数	类型	说明
gwIotId	String	网关设备在平台内的唯一标识。
gwProductKey	String	网关产品的唯一标识。
gwDeviceName	String	网关设备名称。
devices	Object	发现的子设备列表。
iotId	String	子设备在平台内的唯一标识。
productKey	String	子设备产品的唯一标识。
deviceName	String	子设备名称。

设备下行指令结果

通过该Topic可以获取，通过异步方式下发指令给设备，设备进行处理后返回的结果信息。如果下发指令过程中出现错误，也可以通过该Topic得到指令下发的错误信息。

Topic：`/sys/{productKey}/{deviceName}/thing/downlink/reply/message`

数据格式：

```
{
    "gmtCreate":1510292739881,
    "iotId":"4z819VQHk6VSLmmxxxxxxxxxxee200",
    "productKey":"123xxxx554",
    "deviceName":"deviceName1234",
    "requestId":1234,
    "code":200,
    "message":"success",
    "topic":"/sys/123xxxx554/deviceName1234/thing/service/property/set
",
    "data":{

    }
```

```
}
```

参数说明：

参数	类型	说明
gmtCreate	Long	UTC时间戳。
iotId	String	设备在平台内的唯一标识。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
requestId	Long	阿里云产生和设备通信的信息ID。
code	Integer	调用的结果信息。
message	String	结果信息说明。
data	Object	设备返回的结果。Alink格式数据直接返回设备处理结果，透传格式数据则需要经过脚本转换。

返回信息：

参数	类型	说明
200	success	请求成功。
400	request error	内部服务错误， 处理时发生内部错误。
460	request parameter error	请求参数错误， 设备入参校验失败。
429	too many requests	请求过于频繁。
9200	device not actived	设备没有激活。
9201	device offline	设备不在线。
403	request forbidden	请求被禁止，由于欠费导致。

2.1.8 地域和可用区

物联网平台使用规则引擎将设备数据转发至其他阿里云产品。在转发时，需确认目的云产品已经在该地域和可用区上线，并且支持相应格式数据的转发。

表 2-2: 地域和可用区列表

数据转发目标云产品	华东2（上海）		新加坡		日本（东京）		美国（硅谷/弗吉尼亚）		德国（法兰克福）	
	JSON	二进制	JSON	二进制	JSON	二进制	JSON	二进制	JSON	二进制

表格存储（Table Store）	√	-	√	-	√	-	√	-	√	-
DataHub	√	√	√	√	-	-	-	-	-	-
云数据库RDS版（RDS）	√	-	√	-	√	-	√	-	√	-
消息服务（Message Service）	√	√	√	√	√	√	√	√	√	√
消息队列（Message Queue）	√	√	√	√	-	-	-	-	-	-
时序时空数据库（TSDB）	√	-	-	-	-	-	-	-	-	-
函数计算（FC，Function Compute）	√	√	√	√	√	√	-	-	-	-

表 2-3: 地域和可用区列表

数据转发目标云产品	华东2（上海）		新加坡		日本（东京）		美国（硅谷/弗吉尼亚）		德国（法兰克福）	
	JSON	二进制	JSON	二进制	JSON	二进制	JSON	二进制	JSON	二进制
表格存储（Table Store）	√	-	√	-	√	-	√	-	√	-
云数据库RDS版（RDS）	√	-	√	-	√	-	√	-	√	-
消息服务（Message Service）	√	√	√	√	√	√	√	√	√	√
函数计算（FC，Function Compute）	√	√	√	√	-	-	-	-	-	-

2.2 数据流转使用示例

2.2.1 数据转发到另一Topic

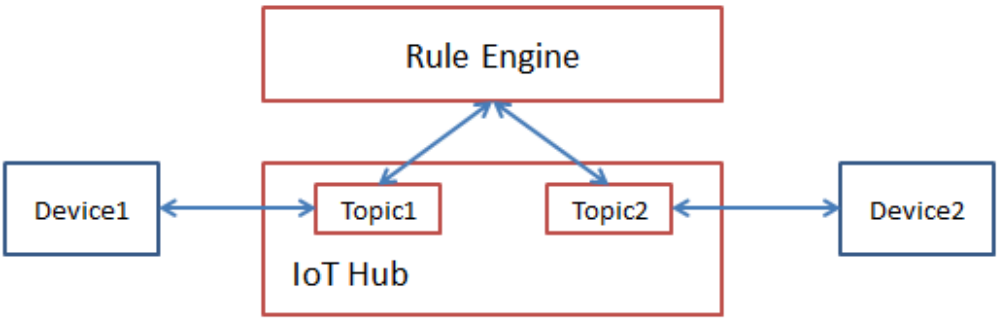
您可以将依据SQL规则处理完的数据，设置转发到另一个Topic中，实现M2M或者更多其他场景。

前提条件

在设置转发之前，您需要参考#unique_79编写SQL完成对数据的处理。

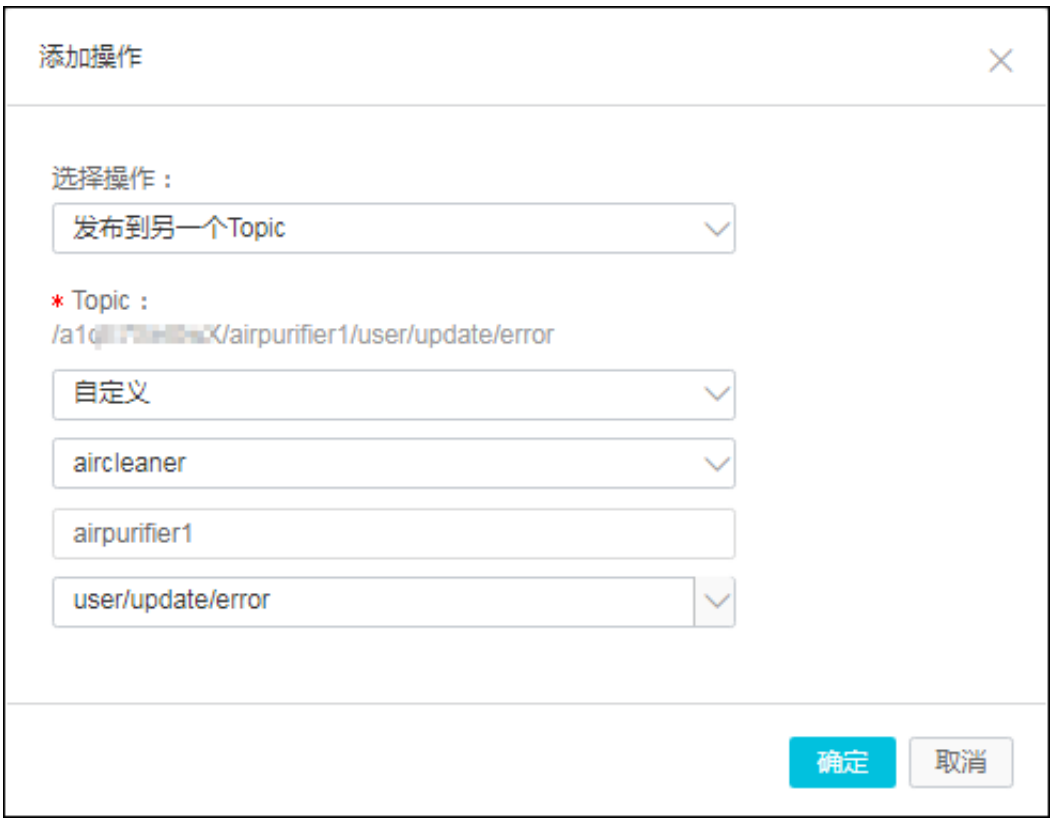
背景信息

本文将教您如何设置数据从Topic1中依照规则引擎设置转发到Topic2内：



操作步骤

1. 单击数据转发一栏的添加操作。出现添加操作页面。



2. 按照页面提示，设置参数。

   - 选择操作：此处选择发布到另一个Topic。
   - Topic：选择您需要把数据转发到哪一个Topic中。
     - 自定义：填写您自定义的产品Topic。在选择产品后，还需补充完整该Topic。您可以使用`${}`表达式引用上下文值。例如，填写`${devicename}/get`表示从消息中筛选出devicename信息，转发到后缀为get的Topic中。
     - sys：选择系统定义的Topic。在选择产品后，还需要选择设备，选择系统定义的某个Topic。

2.2.2 数据转发到消息队列RocketMQ

您可以使用规则引擎，将物联网平台数据转发到消息队列（RocketMQ）中存储。从而实现消息从设备、物联网平台、RocketMQ到应用服务器之间的全链路高可靠传输能力。

前提条件

在设置转发之前，您需

- 参见#unique_79，创建规则和编写用于筛选转发数据的SQL。
- 购买消息队列（RocketMQ）实例，并创建用于接收数据Topic。RocketMQ使用方法，请参见RocketMQ订阅消息RocketMQ订阅消息。

操作步骤

1. **在规则详情页，单击数据转发一栏的添加操作。**
2. **在添加操作对话框中，选择操作为发送数据到消息队列（RocketMQ）中，并按照界面提示，设置参数。**

参数	说明
选择操作	数据转发目标云产品。选择为发送数据到消息队列（RocketMQ）中。
地域	选择RocketMQ Topic所在地域。

参数	说明
实例	选择RocketMQ Topic所属的实例。 若无RocketMQ 实例，单击创建实例到消息队列控制台创建实例。请参见消息队列文档消息队列文档。
Topic	选择写入物联网平台备数据的RocketMQ Topic。 若无RocketMQ Topic，单击创建Topic在消息队列控制台，创建用于接收物联网平台数据的Topic。
Tag	（可选）设置标签。 设置标签后，所有通过该操作流转到RocketMQ对应Topic里的消息都会携带该标签。您可以在RocketMQ消息消费端，通过标签进行消息过滤。 标签长度限制为128字节。可以输入常量或变量。变量格式为${key}，代表SQL处理后的JSON数据中key对应的value值。
授权	授予物联网平台数据写入RocketMQ的权限。 如您还未创建相关角色，请单击创建RAM角色进入RAM控制台，创建角色和授权策略。如需帮助，请参见管理 RAM 角色管理 RAM 角色。

3. 回到数据流转列表页签，单击规则对应的启动按钮启动规则。
   规则启动之后，物联网平台即可将数据写入RocketMQ的Topic。
4. 测试。

   向规则SQL中定义的Topic发布一条消息，然后到RocketMQ控制台消息查询页，查看是否成功接收到消息。

   #unique_102

2.2.3 数据转发到表格存储

您可以使用规则引擎数据流转功能，将数据转发到表格存储（Table Store）中存储。

前提条件

在设置转发之前，

- 在物联网平台中，创建规则和编写处理消息数据的SQL。

  请参见#unique_79。

  本文示例的规则，定义了如下SQL语句：

  ```
  SELECT deviceName as deviceName, items.PM25.value as PM25, items.
  WorkMode.value as WorkMode
  FROM "/sys/a1ktuxe****/aircleanerthing/event/property/post" WHERE
  ```

- 在表格存储中，创建接收数据的实例和数据表。

  表格存储使用指南，请参见表格存储文档表格存储文档。

操作步骤

1. 在规则的数据流转详情页，单击数据转发一栏的添加操作。选择存储到表格存储（Table Store）。

   说明：

不支持二进制格式的数据转发至表格存储。

2. 按照界面提示设置参数，然后单击确定。

参数	说明
选择操作	选择为存储到表格存储（Table Store）。
地域	选择接收数据的表格存储实例的地域。
实例	选择接收数据的表格存储实例。
数据表	选择接收数据的数据表。

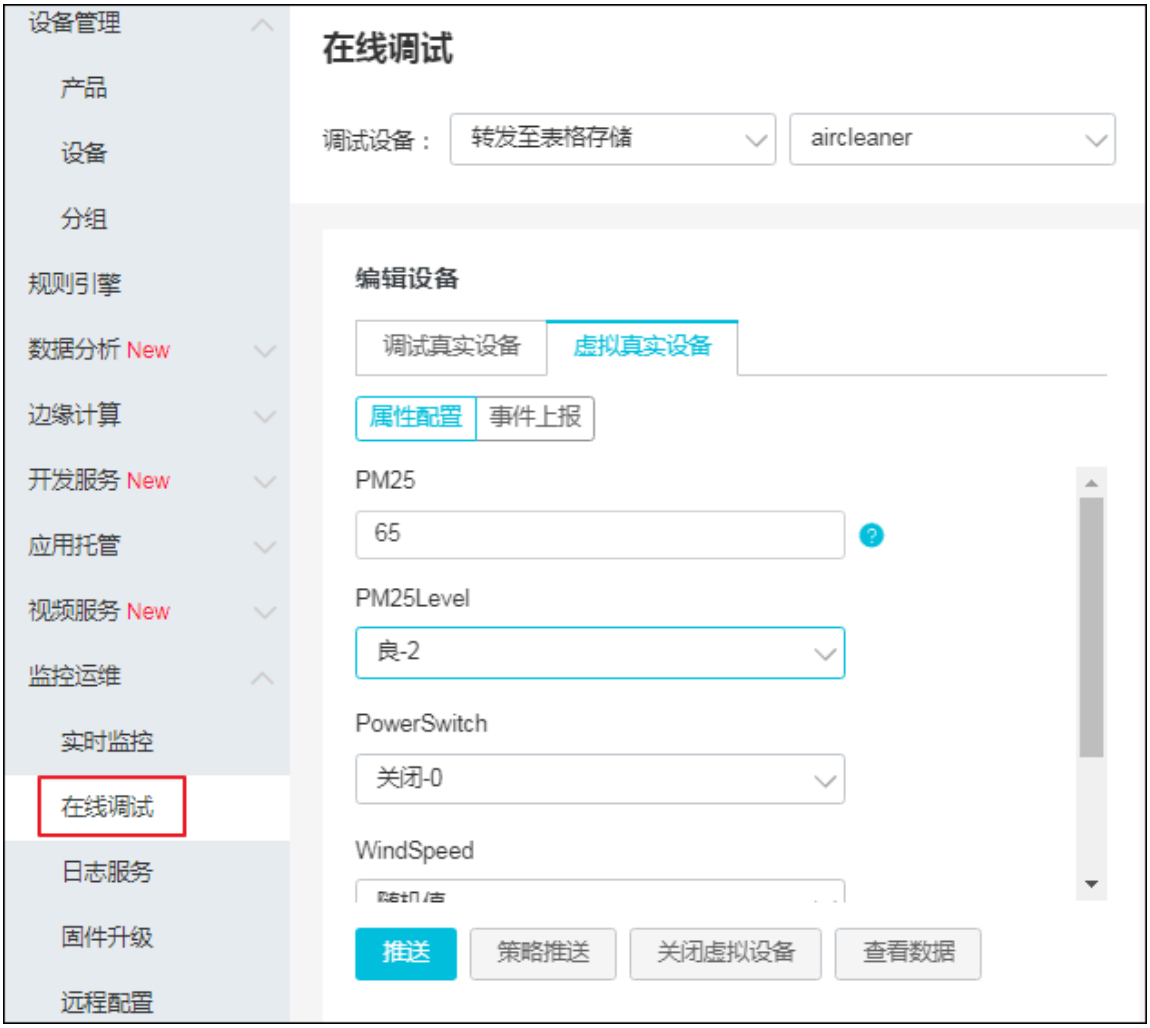
参数	说明
主键	配置表格存储数据表主键对应的值，需设置为规则SQL中SELECT的某字段值。数据流转时，该值将被存为主键对应的值。 说明: · 支持配置为变量格式`${}`，如${deviceName}，表示该主键对应的值为消息中`deviceName`的值。 · 如果主键类型是自增列，这一列主键无需填值，表格存储会自动生成这一主键列的值。所以，自增列主键值，系统已自动设置为`AUTO_INCREMENT`，且不能编辑。 更多自增列主键说明，请参见主键列自增主键列自增。
角色	授权物联网平台将数据写入表格存储。 需要您进入访问控制RAM控制台创建一个具有表格存储写入权限的角色，然后将该角色赋予给物联网平台。

3. 数据转发操作配置完成后，回到数据流转列表页，单击规则对应的启动操作按钮。
   规则启动后，SQL语句中定义的Topic有消息发布时，SELECT字段定义的消息数据将被流转至表格存储数据表中。

4. 模拟推送数据，测试数据流转。

a) 在物联网平台控制台左侧导航栏，选择监控运维 > 在线调试。

b) 选择调试用的设备，使用虚拟真实设备推送模拟数据。请参见#unique_104。



c) 数据推送成功后，在表格存储接收数据的数据表的数据管理页，查看是否成功接收到指定数据。

2.2.4 数据转发到DataHub

您可以使用规则引擎将数据转到DataHub上，再由DataHub将数据流转至实时计算、MaxCompute等服务中，以实现更多计算场景。

前提条件

在设置转发之前，您需要参见#unique_79编写SQL，用于处理数据。

操作步骤

1. 单击数据转发一栏的添加操作，出现添加操作页面。选择发送数据到DataHub中。

2. 按照界面提示，设置参数。

   - 选择操作：选择发送数据到DataHub中。
   - 选择DataHub对应的地域、Project和Topic。选完Topic后，规则引擎会自动获取Topic中的Schema，规则引擎筛选出来的数据将会映射到对应的Schema中。

     说明:

     - 将数据映射到Schema时，需使用${}，否则存入表中的将会是一个常量。
     - Schema与规则引擎的的数据类型必须保持一致，不然无法存储。
   - 角色：授权物联网平台将数据写入DataHub。此处需要您先创建一个具有DataHub写入权限的角色，然后将该角色赋予给规则引擎。这样，规则引擎才可以将处理完成的数据写入DataHub中。

     #unique_106

2.2.5 数据转发到云数据库RDS

您可以设置规则引擎数据流转，将处理后的数据转发到云数据库RDS版（以下简称RDS）的VPC实例中。

使用须知

- 只支持同地域转发，不支持跨地域转发。例如，华东2节点的物联网平台数据只能转发到华东2的RDS数据表中。
- 只支持转发到专有网络RDS实例。
- 支持MySQL实例和SQL Server实例。
- 支持普通数据库和高权限数据库的转发。
- 不支持二进制格式数据转发至RDS。

准备工作

- 请参见#unique_79创建规则和编写用于处理数据的SQL。
- 购买与您的物联网服务地域相同的RDS实例，并创建数据库和数据表。

操作步骤

1. 单击数据转发一栏的添加操作，出现添加操作页面。选择存储到云数据库（RDS）中。

2. 按照界面提示，设置参数。

参数	描述
选择操作	选择存储到云数据库（RDS）中。
RDS实例	需选择与您的物联网服务地域相同的专有网络RDS实例。

参数	描述
数据库	输入数据库名。  说明: 如果是高权限数据库，需要您手动输入数据库名称。
账号	输入RDS实例的用户账号。此账号应具有该数据库的读写权限，否则规则引擎无法将数据写入RDS。  说明: 规则引擎获得账号后，仅将规则匹配的数据写进数据库中，不会做其他操作。
密码	输入登录RDS实例的密码。
表名	输入数据库中已建立的数据表名。规则引擎将把数据写入这张表上。
键	输入RDS数据表的一个字段。规则引擎将把数据存入该字段中。
值	输入您在数据处理SQL中指定的Topic中的消息的一个字段，作为输入数据表字段（键）的值。  说明: · 值与键的数据类型需保持一致，否则无法存储成功。 · 可输入一个变量，如`${deviceName}`。
角色	授权物联网平台向RDS数据表写入数据的角色。 若您还没有创建相关角色，请单击创建RAM角色进入访问控制（RAM）控制台创建。

3. 在数据流转列表中，单击该规则对应的启动按钮启动规则。

4. 在RDS控制台数据安全性页，设置和查看白名单。配置完成后，规则引擎为了连接RDS，会在RDS的白名单中添加下列IP。若这些IP未出现，请手动添加。

   - 华东2：100.104.123.0/24
   - 亚太东南1（新加坡）：100.104.106.0/24
   - 美国（硅谷）：100.104.8.0/24
   - 美国（弗吉尼亚）：100.104.133.64/26
   - 德国（法兰克福）：100.104.160.192/26
   - 日本（东京）：100.104.160.192/26

2.2.6 数据转发到消息服务

您可以使用规则引擎数据流转功能，将设备数据转发到消息服务主题中，服务端再从消息服务主题中订阅消息，实现设备端与服务端之间高性能的消息闭环传输。

数据转发流程

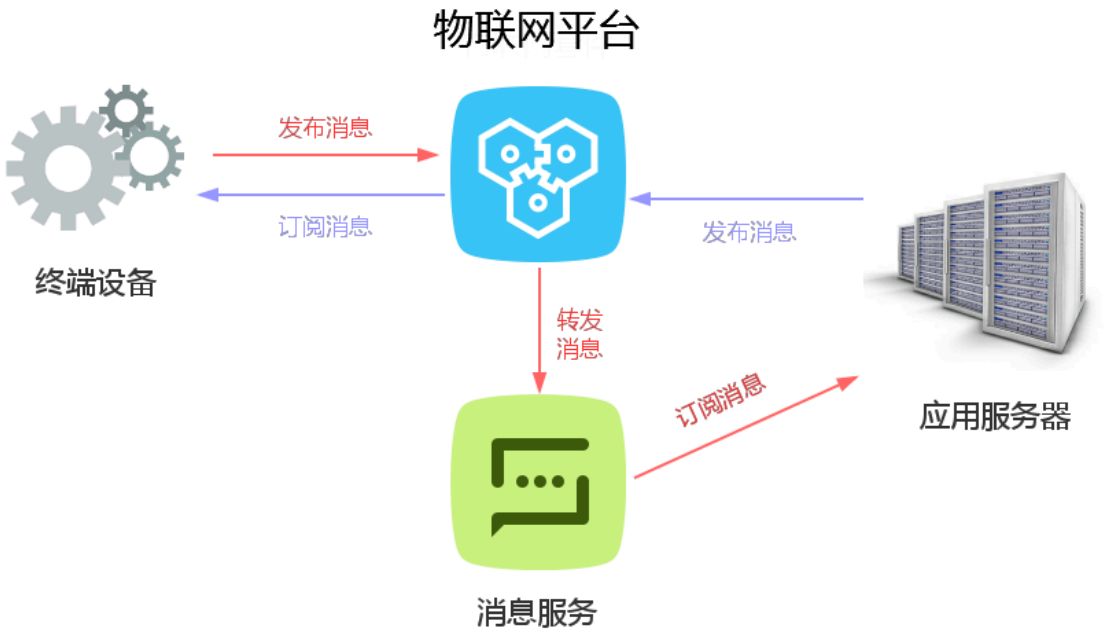
- 设备发送数据到服务端。

  设备发布消息到物联网平台中，物联网平台通过规则引擎将消息进行处理并转发到消息服务的主题中。然后，您的应用服务器调用消息服务的接口订阅消息。

  这种方式优势是消息服务可以保证消息的可靠性，避免了服务端不可用时导致消息丢失。同时，消息服务在处理大量消息并发时，有削峰填谷的作用，保证服务端不会因为突然的并发压力导致服务不可用。物联网平台与消息服务的结合，可以实现设备端与服务端之间高性能的消息闭环传输。

- 服务端发送数据到设备。

  您的应用服务器调用物联网平台的云端 API，发布数据到物联网平台中，然后设备从物联网平台中订阅消息。



操作步骤

1. 在访问控制（RAM）控制台，创建一个授权物联网平台数据写入消息服务的权限角色。

   物联网平台必须经过您的授权才能对您的消息服务主题写入数据。所以，您需要创建一个具有消息服务写入数据权限的角色，然后将该角色授予给物联网平台。这样规则引擎才能将处理过后的数据写入消息服务中。

   授权相关说明，请参见管理 RAM 角色管理 RAM 角色。

2. 在消息服务控制台，创建接收消息的主题。

   a) 在消息服务控制台中，单击主题 > 创建主题。

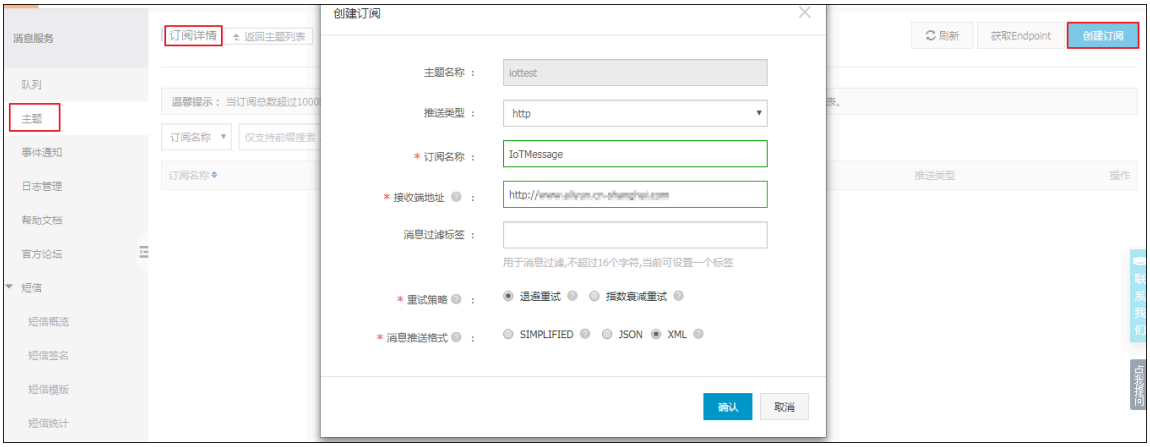
   b) 在创建主题对话框中，输入主题信息，然后单击确定。

   c) 在主题列表，找到创建的主题，单击其对应的订阅详情操作按钮。

   d) 在订阅详情页，单击创建订阅。

   e) 为这个主题创建订阅者。消息服务会将接收到的物联网平台的消息发送给您配置的订阅者。

> **说明:**
> 目前，物联网平台推送至消息服务主题的消息，仅可以通过队列和HTTP两种方式订阅。



   您可以根据自己的业务需求创建多个订阅者。

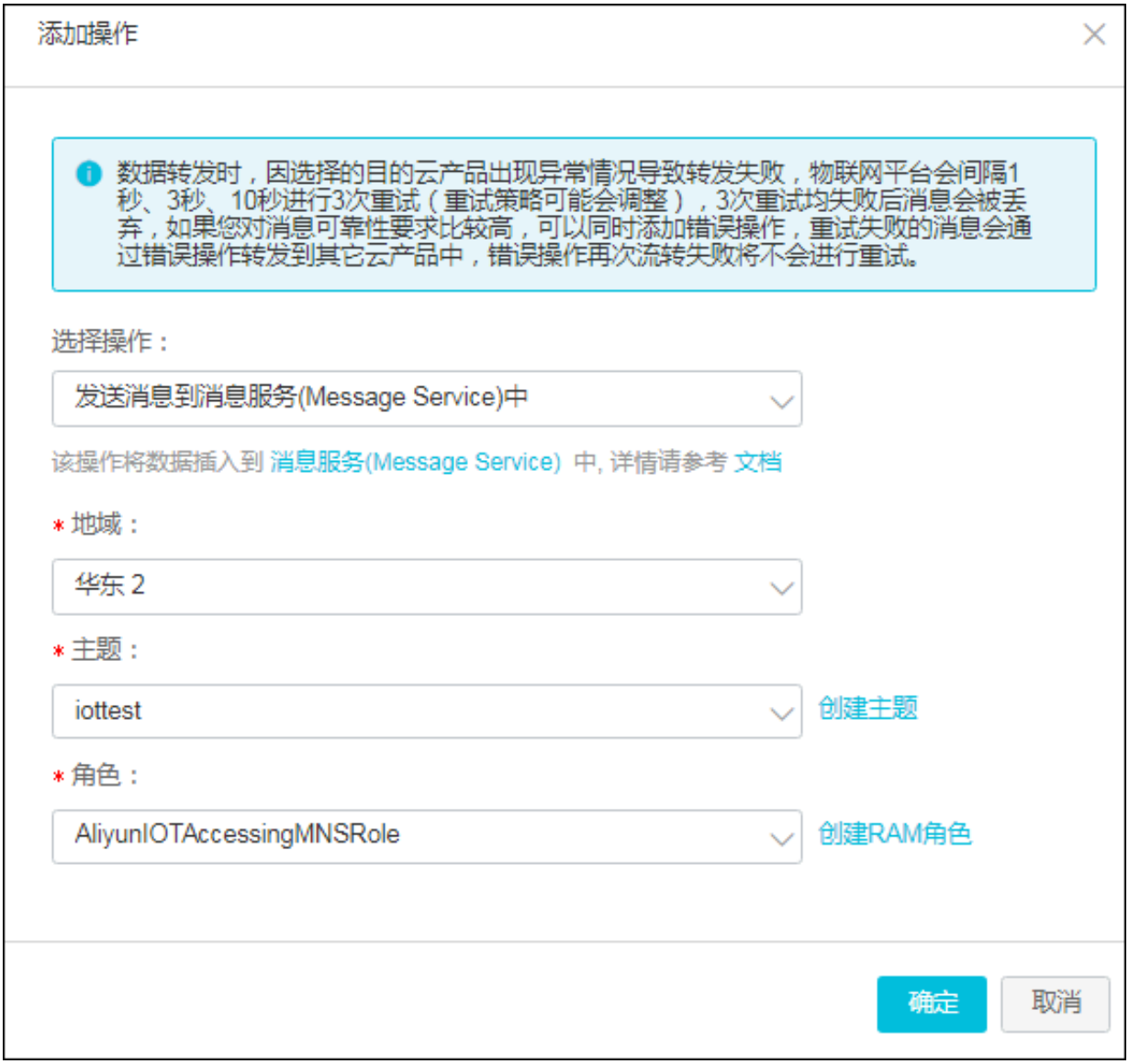
   关于消息服务的使用方法，请参见消息服务产品文档消息服务产品文档。

3. 在物联网平台控制台规则引擎页，单击创建规则，创建一个规则。

4. 在规则详情页，为该规则编写用于处理数据的SQL。SQL编写指导，请参见设置规则引擎和#unique_80。

5. 在规则详情页，单击数据转发栏对应的添加操作。

6. 在添加操作对话框中，输入数据转发目标的消息服务主题信息。



参数说明：

参数	描述
选择操作	选择物联网平台数据转发目标云产品。此处选择为发送消息到消息服务（Message Service）中。
地域	选择数据转发目标消息服务主题所在地域。
主题	选择接收数据的消息服务主题。
角色	授予物联网平台访问消息服务的权限角色。物联网平台扮演该角色，便可将数据写入消息服务。

7. 返回规则引擎页，在数据流转规则列表下，单击该规则对应的启动操作按钮，启动规则。

   该规则启动后，物联网平台便可将数据转发至设定的主题中，您便可以通过消息服务接收和处理来自物联网平台的数据。

2.2.7 数据转发到时序时空数据库

您可以配置数据流转规则，将处理过的数据转发到时序时空数据库（TSDB）的实例中存储。

使用须知

- 目前转发至TSDB仅发布在华东2节点。
- 只支持专有网络（VPC）下TSDB实例。
- 只支持数据在同地域内转发。例如：华东2地域的设备数据只能转发到华东2的TSDB实例中。
- 只支持JSON格式数据转发。
- 不支持时序数据库InfluxDB版。
- 不支持时空数据库。
- 转发的消息中，除了配置为`timestamp`和tag值的字段外，其他字段都将作为`metric`写入时序时空数据库。`metric`的值只能为数值类型，否则会导致写入数据库失败。

更多TSDB相关信息，请参见时序时空数据库（TSDB）使用指南。

准备工作

在设置转发之前，请参见#unique_79，创建数据转发规则和编写处理数据的SQL。

操作步骤

1. 在规则详情页，单击数据转发一栏的添加操作。

2. 按照界面提示，设置参数。

参数	描述
选择操作	选择存储到时序时空数据库（TSDB）中。
VPC实例	选择数据转发目标为您已创建的专有网络（VPC）中的TSDB实例。
timestamp	时间戳。支持： · 使用转义符`${}`表达式，如`${time}`，表示timestamp的值为规则SQL中指定Topic对应数据包中time字段对应的值。 · 使用数据流转函数`timestamp()`，表示timestamp的值为数据流转服务器的时间戳。 · 输入值，必须为Unix时间戳，如1404955893000。

参数	描述
tag	设置标记数据的标签名。支持中文汉字、英文字母、数字、和特殊字符包括：半角冒号（:）、逗号（,）、点号（.）、单引号（'）、正斜线（/）、连字符（-）、下划线（_）、圆括号()、方括号[]。
值	设置标签值。支持： · 使用转义符`${}`表达式，如`${city}`，表示标签值为规则SQL中指定Topic对应数据包中city字段对应的值。建议使用此方式。 · 使用数据流转函数规定的一些函数，如`deviceName()`，表示标签值为设备名称。 · 输入常量，例如beijing。支持输入中文汉字、英文字母、数字、和特殊字符包括：半角冒号（:）、逗号（,）、点号（.）、单引号（'）、正斜线（/）、连字符（-）、下划线（_）、圆括号()、方括号[]。 说明: · 最多可添加8个tag键-值对。 · 需保证TSDB能够获取到配置的tag键-值对，如果获取不到任意一个tag键-值对，会导致写入数据库失败。
角色	授予物联网平台向TSDB写数据的权限。 物联网平台会向TSDB实例中添加网络白名单，用于物联网平台访问您的TSDB数据库， 请勿删除这些IP段。 若TSDB实例白名单中未出现物联网平台IP：100.104.76.0/24，请手动添加。

示例

· 规则的SQL如下：

```
SELECT time,city,power,distance, FROM "/myproduct/myDevice/update"`;
```

以上SQL从Topic `/myproduct/myDevice/update`的消息中，筛选出time、city、power、distance字段内容，作为转发的消息内容。

· 通过以上SQL处理后的转发消息示例如下：

```
{
"time": 1513677897,
"city": "beijing",
"distance": 8545,
"power": 93.0
}
```

· 配置的数据转发规则，请见上一章节操作步骤中截图示例。

- 根据数据转发规则，向TSDB中写入的两条数据：

```
数据: timestamp:1513677897, [metric:power value:93.0]
tag: cityName=beijing
```

```
数据: timestamp:1513677897, [metric:distance value:8545]
tag: cityName=beijing
```

写入TSDB的数据说明：

以上转发的消息中，除了配置为`timestamp`的time字段和配置为tag值的city字段外，其他字段（power和distance）都作为`metric`写入时序时空数据库。

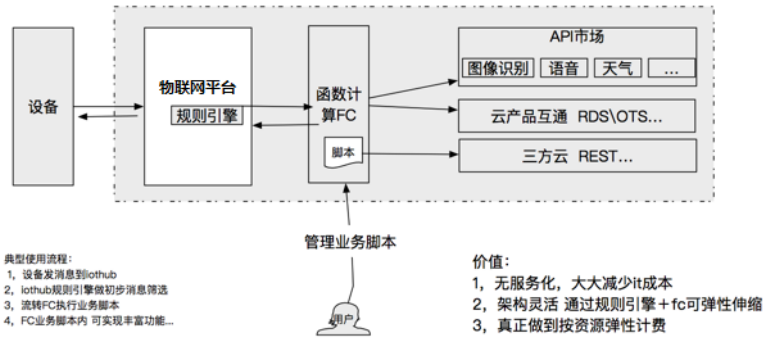
实践案例

#unique_111

2.2.8 数据转发到函数计算

您可以使用规则引擎数据流转，将数据转发至函数计算（FC）中，然后由函数计算运行函数脚本进行业务处理。

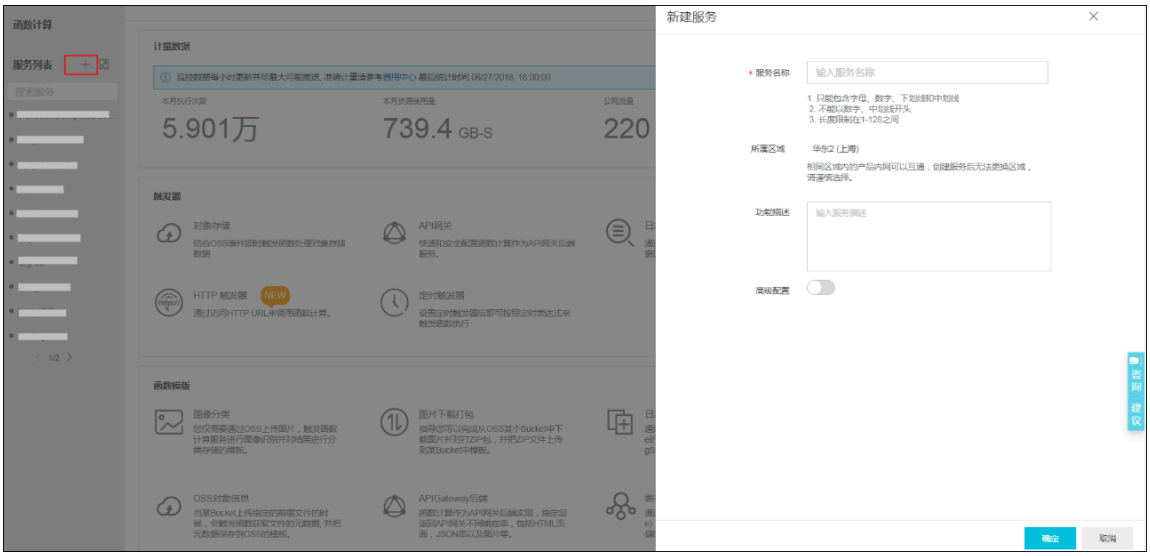
示例流程如下图：



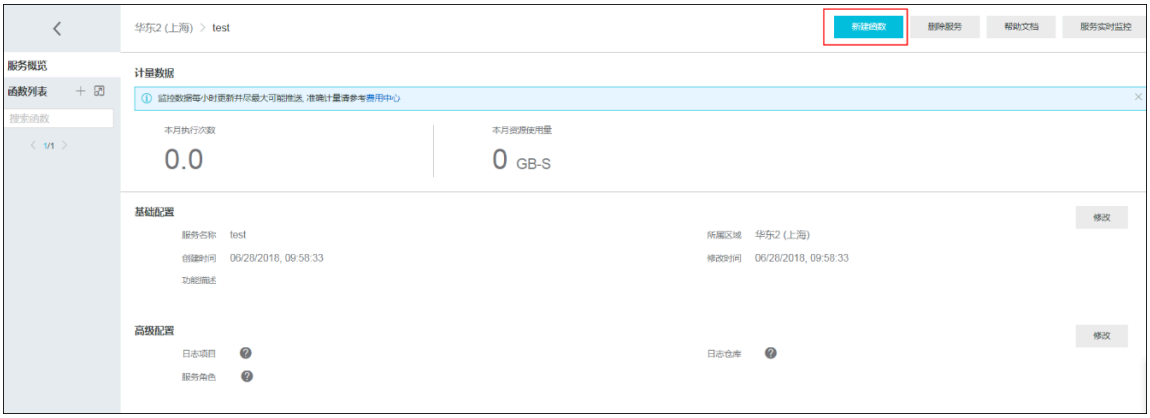
操作步骤

1. 登录函数计算控制台，创建服务与函数。

   a. 创建服务。其中，服务名称必须填写，其余参数请根据您的需求设置。



   b. 创建服务成功后，创建函数。



   c. 选择函数模板，此处以空白函数模板为示例。

   d. 设置函数参数。

此示例设置函数的逻辑为直接在函数计算中显示获取的数据。

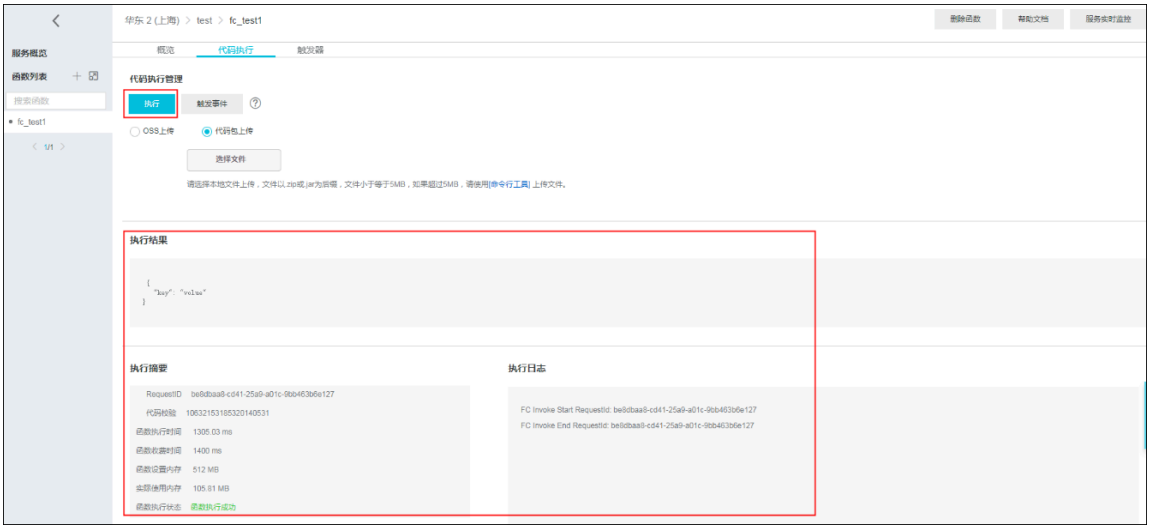
参数	说明
所在服务	选择函数所在的服务。
函数名称	设置函数名称。
运行环境	设置函数运行的环境，此示例中选择java8。
代码配置	上传代码。
函数入口	设置您的函数在函数计算系统运行的调用入口。此示例中设置为`com.aliyun.fc.FcDemo::handleRequest`。
函数执行内存	根据您的业务情况设置函数执行内存。

参数	说明
超时时间	设置超时时间。

更详细的函数计算使用帮助，请参见函数计算函数计算设置。

e. 函数执行验证。

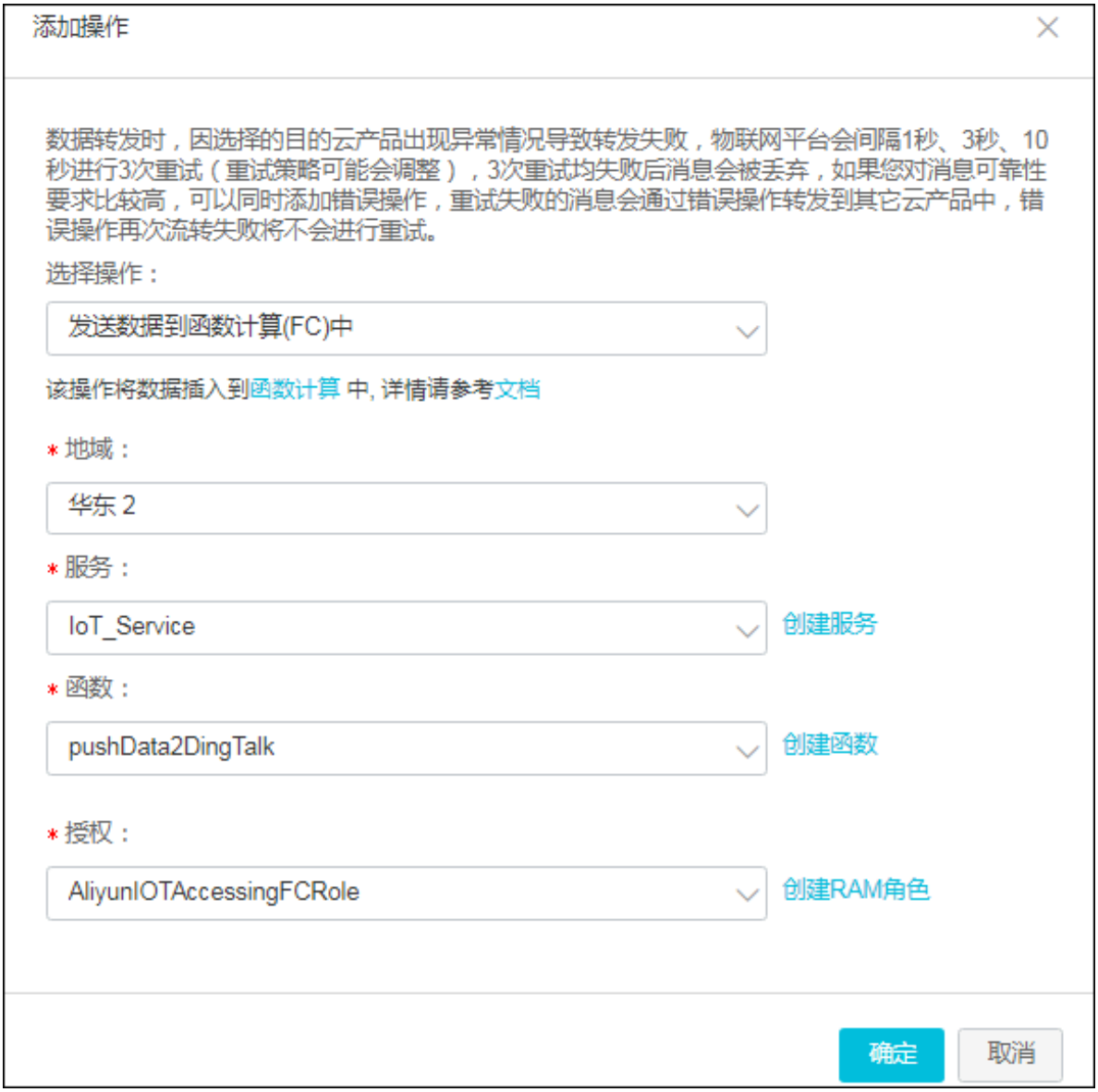
函数成功创建后，可以直接在函数计算的控制台执行，以验证函数执行情况。函数计算会直接将函数的输出和请求的相关信息打印在控制台上。



2. 在物联网平台控制台，创建规则，编写筛选消息数据的SQL。请参见#unique_113。
3. 在规则的数据流转详情页，单击数据转发一栏的添加操作，并在弹出的添加操作页面中设置参数。

说明:

JSON格式和二进制格式都支持转发到函数计算中



参数	说明
选择操作	选择发送数据到函数计算（FC）中。
地域	选择接收数据的函数计算服务地域。 **说明:** 目前支持转发数据至函数计算的地域包括：中国（上海）、新加坡和日本（东京）。
服务	选择接收数据的函数计算服务。
函数	选择接收数据的函数。

参数	说明
授权	授予物联网平台向函数计算写入数据的权限。 如您还未创建相关角色，请单击创建RAM角色进入RAM控制台，创建角色和授权策略。如需帮助，请参见管理 RAM 角色管理 RAM 角色。

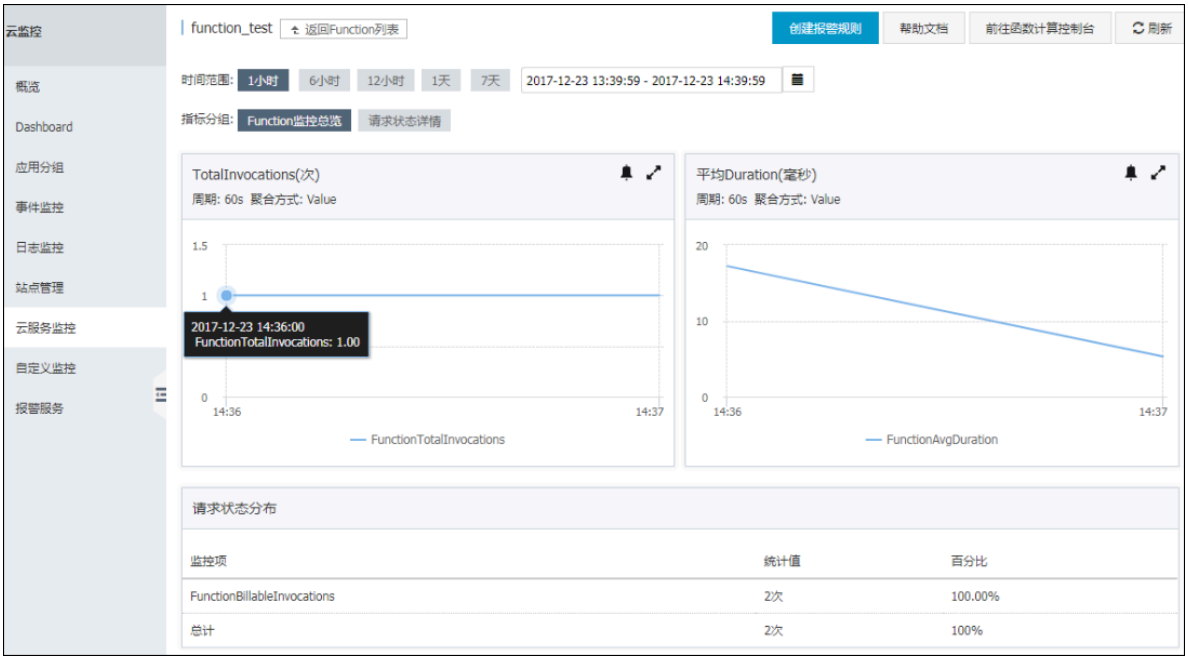
4. 在数据流转列表页签下，单击规则对应的启动按钮启动规则。

   规则启动后，物联网平台就会将相关数据发送至函数计算。

5. 测试。

   向规则SQL中定义的Topic发送一条测试消息，然后再到函数计算控制台去查看是否有相关记录。

   函数计算控制台针对函数的执行情况有监控统计。统计有大概5分钟的延时，可以通过云监控大盘查询函数的执行情况。



实践案例

#unique_114

#unique_115

2.3 场景联动

2.3.1 云端场景联动

场景联动类型的规则是一种开发自动化业务逻辑的可视化编程方式，可以通过设备或时间维度的条件触发，经过执行条件的过滤，执行预定的业务逻辑，输出数据到设备或者其他规则，实现海量设备根据场景的联动。

创建场景联动

1. 单击物联网平台控制台左侧导航栏中的规则引擎 > 场景联动。
2. 单击创建规则。

3. 设置参数，然后单击确认。

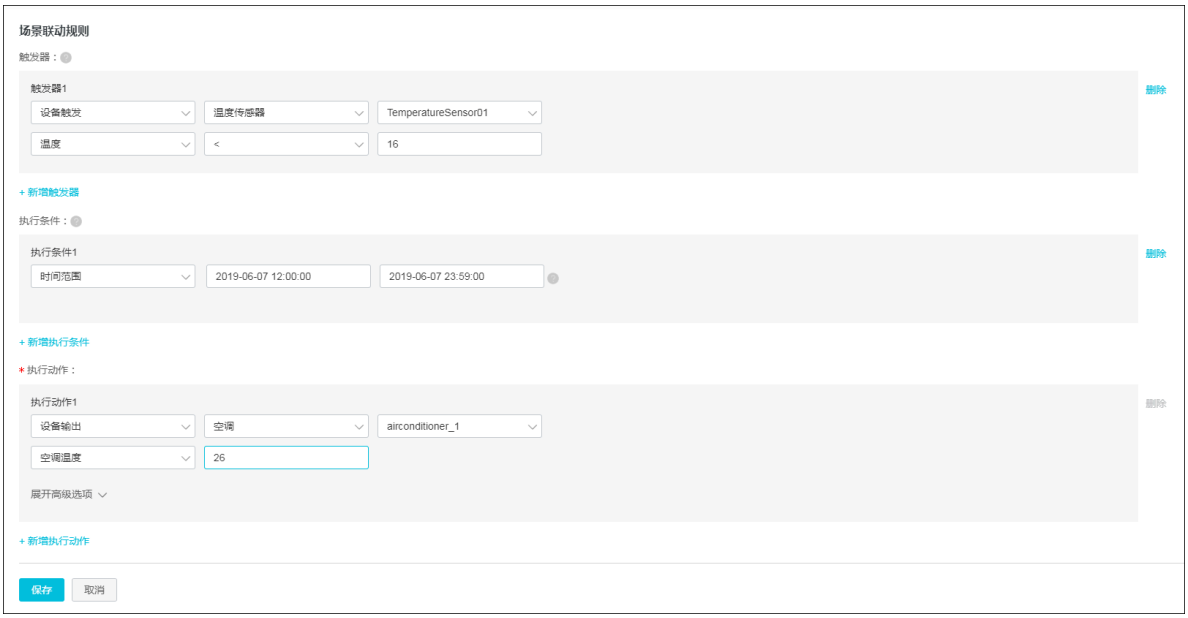
参数	描述
规则名称	设置具体规则的名称。支持中文、英文字母、数字、下划线和短划线，长度限制 1 ~ 30个字符，中文字算两位字符。
规则描述	为规则添加描述，可以为空。

4. 完成场景联动的创建后在弹窗中单击前往编辑，管理配置场景联动。

   您也可以在场景联动名称右侧单击管理，管理配置场景联动。

   以空调设备自动化为例：在 12:00 至 23:59 之间，当温度传感器上报的室内温度低于16摄氏度时，空调设备执行控制室内温度为26摄氏度。

   具体参数设置，请见下图。



   单击页面右上角编辑，可更改场景联动规则名称，其余参数说明请见下表。

参数	描述
触发器	即规则入口。可设置为设备触发或定时触发。当设备上报的数据或当前时间满足设定的触发器时，触发执行条件判断。可以为一个规则创建多个触发器，触发器之间是或（or）关系。 · 设置为设备触发，则需选择已创建的产品名称、设备名称、和设备属性或事件。 · 设置为定时触发，则需填写时间点。时间点格式为 cron 表达式。cron表达式的构成：分、小时、日、月、一周内的某天（0或7表示周日，1-6分别表示周一至周六），每项之间用空格隔开。如，每天18点整的cron表达式为：0 18 * * *（其中星号（*）是通配符）；每周五18点整的表达式为：0 18 * * 5。cron表达式具体写作方法，请参见 CRONTAB 网页。 上图示例中，设置为设备触发：以温度传感器上报的室内温度低于16摄氏度作为触发器。

参数	描述
执行条件	执行条件集。只有满足执行条件的数据，才能触发执行动作。可设置为设备状态或时间范围。可以为一个规则创建多个执行条件，执行条件之间是和（and）关系。 · 设置为设备状态，则需选择已创建的产品名称、设备名称、和设备功能中的属性或事件。 · 设置为时间范围，则需设置起始时间和结束时间。 上图示例中，设置为时间范围：时间在 12:00 至 23:59 之间，则触发执行动作。
执行动作	需执行的动作。可设置为设备输出或规则输出。您可以设置多个动作。某一动作执行失败时，不影响其他动作。 · 设置为设备输出，则需选择已创建的产品名称、设备名称、和设备属性或服务（只有可写的属性或服务才能被设为执行动作）。当触发器和执行条件均被满足时，执行已定义的设备属性或服务的相关动作。 · 设置为规则输出，则需嵌套另外一个规则，即调用其他规则。被调用规则中的触发器将被跳过，直接进行执行条件检查。若执行条件满足，则执行该规则中定义的执行动作。 上图示例中，设置为设备输出：指定的空调设备执行控制温度为26摄氏度。

运行场景联动

场景联动创建成功后，您可在规则引擎 > 场景联动页面中，启动此场景联动。

启动场景联动操作：

1. 单击物联网平台控制台左侧导航栏中规则引擎 > 场景联动。
2. 找到要启动的场景联动，单击右侧操作栏中的启动，使规则状态为运行中。

当设备有数据上报，并且上报数据满足触发器时，该场景联动便会在云端运行。

启动场景联动后：

- 若场景联动在云端运行，则需要为场景联动中的设备配置消息路由，使得设备的属性和事件能够发送到IoT Hub（云端）。消息路由的配置请参考#unique_118。
- 若场景联动在边缘端运行，则需要先停止其在云端的运行，再将场景联动分配到边缘实例中，分配方法请参考本文下方场景联动其他操作。

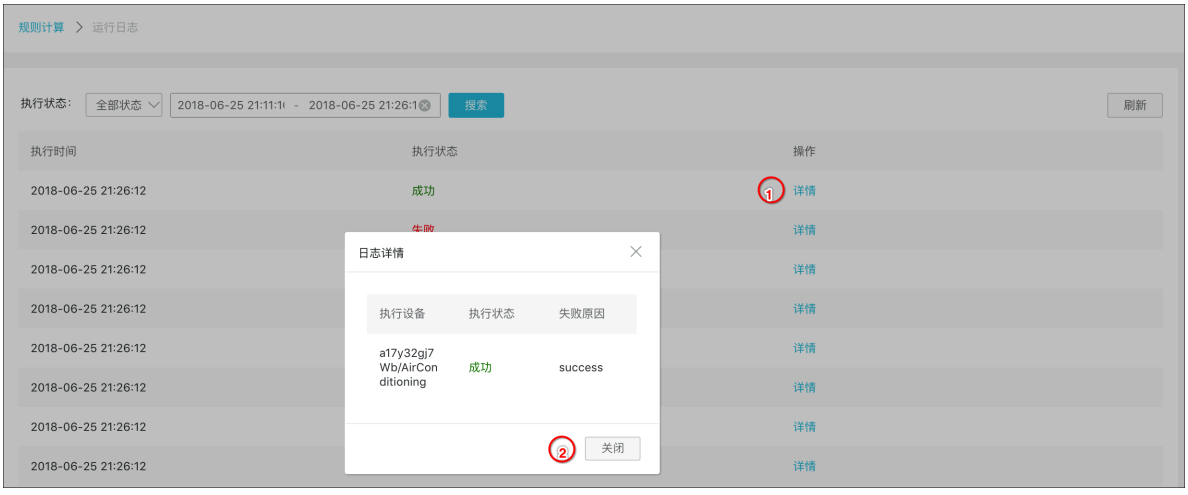
查看日志

您可以查看该场景联动的日志，并且可在详情中查看运行结果。

> **说明:**
> 若某条场景联动即在云端运行又在边缘端运行，那么在物联网平台控制台规则引擎 > 场景联动中，查看到的日志为云端运行日志和边缘端运行日志。

1. 单击物联网平台控制台左侧导航栏中规则引擎 > 场景联动。
2. 找到要查看日志的场景联动，单击右侧操作栏中的日志。

3. 单击详情，查看该条日志的详情信息。



场景联动其他操作

- 删除场景联动：

  1. 在规则引擎 > 场景联动页面中，找到需要删除的场景联动规则名称。
  2. 单击规则名称右侧的删除，删除该条场景联动规则。

- 触发场景规则：

  在启动场景联动规则后，方可显示触发操作按钮。

  1. 在规则引擎 > 场景联动页面中，找到需要已启动，并需要触发的场景联动规则名称。
  2. 单击规则名称右侧的触发，表示手动触发规则一次，即忽略已管理配置的触发器，直接执行所有执行条件和执行动作。

- 在边缘实例中运行场景联动：

  您需要根据如下步骤，将场景联动部署到边缘实例中。

  > **注意：**
  > 请确保已停止场景联动在云端的运行。

  1. 左侧导航栏选择边缘计算 > 边缘实例，在环境搭建中完成的边缘实例右侧单击查看。
  2. 在实例详情页面，选择场景联动，单击分配场景。
  3. 在分配场景页面，单击待分配场景联动规则名称后的分配，然后单击关闭。

  4. 分配场景联动后，重新部署边缘实例。

2.3.2 什么是场景联动

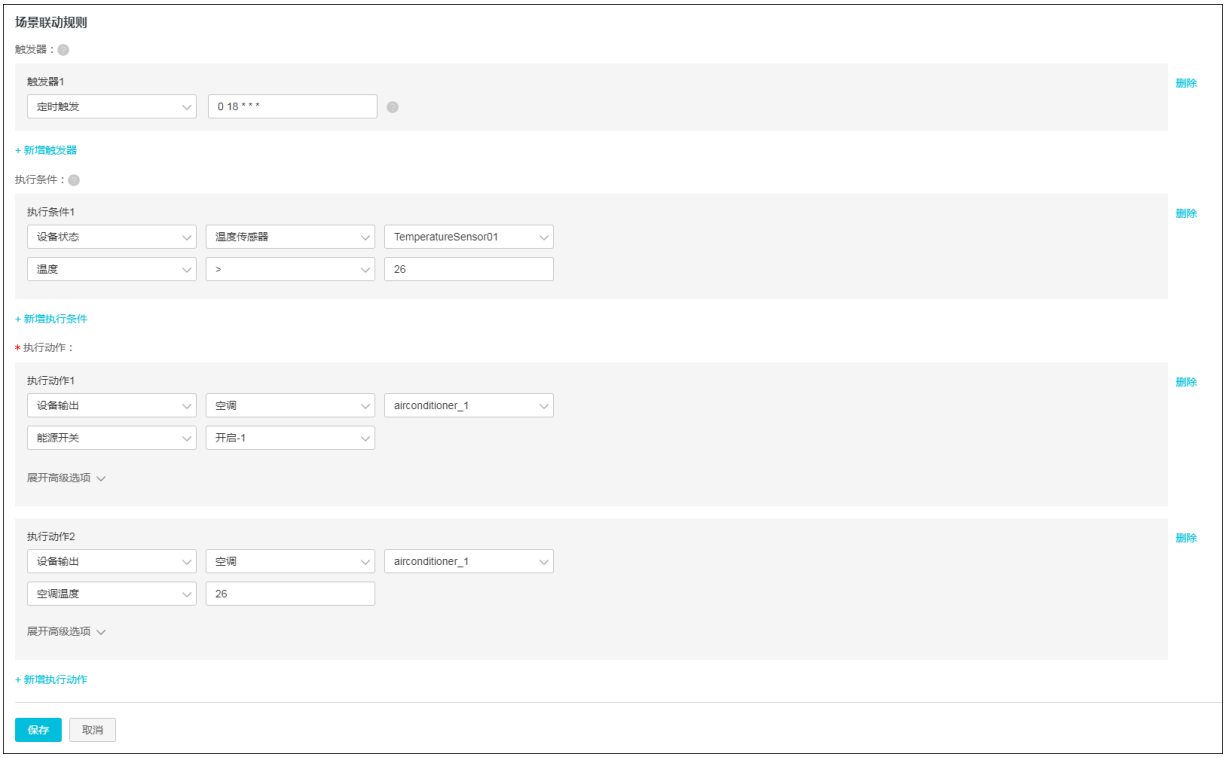
场景联动是规则引擎中，一种开发自动化业务逻辑的可视化编程方式，您可以通过可视化的方式定义设备之间联动规则，并将规则部署至云端或者边缘端。

您需在物联网平台控制台，规则引擎 > 场景联动页面中创建场景联动规则。每个场景联动规则由触发器（Trigger）、执行条件（Condition）、执行动作（Action）三个部分组成。这种规则模型称为 TCA 模型。

当触发器指定的事件或属性变化事件发生时，系统通过判断执行条件是否已满足，来决定是否执行规则中定义的执行动作。如果满足执行条件，则直接执行定义的执行动作；反之则不执行。

例如，您每天18:00下班回家。在炎热的夏天，您希望您到家后，家里的温度是凉爽、舒适的。您可以创建一条规则，使空调设备自动化，实现这个需求。

参数设置如下图。



参数说明如下：

参数	描述	并列关系
触发器	定时为每天18:00。时间的cron表达式写作方法，请参见CRONTAB网页。	或（||）
执行条件	温度传感器探测到室内温度高于 26 摄氏度。	与（&&）
执行动作	空调开关设置为打开；空调控制温度设置为 26 摄氏度。	与（&&）

创建场景联动规则的更多设置说明，请参见云端管理场景联动。

3 监控运维

3.1 实时监控

3.1.1 查看实时监控数据

物联网平台实时监控页，实时展示您的在线设备数量、上行/下行消息量和规则引擎消息流转次数。

查看实时数据

1. 登录物联网平台控制台。
2. 在左侧导航栏中，选择监控运维 > 实时监控。
3. 选择需要查看数据的产品和时间范围。

表 3-1: 时间选择说明

时间	说明
1小时	将展示1小时内，每1分钟为一个采集周期，每个周期的统计数据。
1天	将展示24小时内，每5分钟为一个采集周期，每个周期的统计数据。
1周	将展示7天内，每15分钟为一个采集周期，每个周期的统计数据。

说明:

实时监控页上显示的横坐标并不是统计周期。

数据统计说明

实时监控页展示的统计数据说明见下表：

数据类	说明
实时在线设备	与物联网平台建立长连接的设备数量。 数据采集有一定延迟，显示的数值是每个采集周期内的平均值。按设备与物联网平台的通信协议类型进行区分显示。
发送到平台的消息量	设备发送到物联网平台的消息数量。 数据采集有一定延迟，显示的数值是每个采集周期内的累加值。按设备与物联网平台的通信协议类型进行区分显示。
平台发出的消息量	从物联网平台发送到设备和服务端的消息。 数据采集有一定延迟，显示的数值是每个采集周期内的累加值。按设备与物联网平台的通信协议类型进行区分显示。
规则引擎消息流转次数	规则引擎数据流转功能流转消息的次数。 数据采集有一定延迟，显示的数值是每个采集周期内的累加值。按数据流转的目标云产品类型进行区分显示。

相关文档

在实时监控页，您可以单击报警配置，进入云监控控制台，设置阈值报警规则和事件报警规则。报警规则配置和报警消息说明，请参见：

配置报警规则

#unique_127

3.1.2 配置报警规则

物联网平台支持您使用云监控服务，设置阈值报警规则和事件报警规则，监控您的物联网平台资源使用情况，并及时接收报警信息。

创建阈值报警规则

阈值报警，即在指定时间周期内，您的物联网平台资源使用情况或操作执行失败次数超过指定阈值后，云监控将根据报警规则发送报警通知。

目前，物联网平台支持以下类型阈值报警：

- 通过MQTT协议接入物联网平台的实时在线设备数量。
- 设备通过MQTT、CoAP、HTTP、HTTP/2或LoRa协议发送到物联网平台的消息数量。
- 物联网平台通过MQTT、HTTP/2或LoRa协议发送到设备的消息数量。

- 规则引擎流转消息到物联网平台Topic（REPUBLISH）、DataHub、函数计算（FC）、消息服务（MNS）、消息队列（RocketMQ）、表格存储（OTS）、云数据库（RDS）、或时序时空数据库（TSDB）的次数。
- 设备上报属性失败数量。
- 设备上报事件失败数量。
- 设备服务调用失败数量。
- 设备属性设置失败数量。

1. 登录物联网平台控制台。
2. 在左侧导航栏中，选择监控运维 > 实时监控。
3. 单击实时监控页上的报警配置按钮。
4. 在创建报警规则页，设置报警规则的具体信息，单击确认完成规则创建。

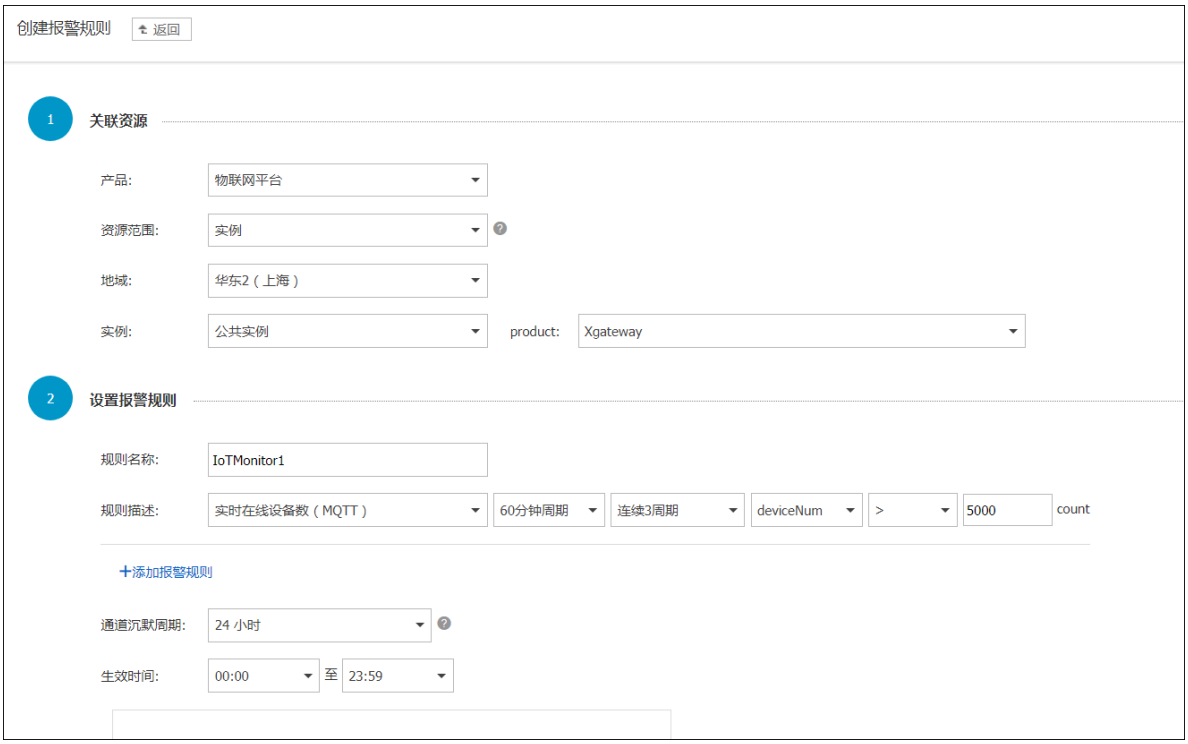


表 3-2: 阈值报警规则参数说明

参数	说明
产品	选择物联网平台。
资源范围	资源范围选择： · 全部资源：您的物联网平台服务下，任何实例满足报警规则描述时，都会发送报警通知。 · 实例：指定实例中的指定产品满足报警规则描述时，才会发送报警通知。

参数	说明
地域	资源范围选择为实例时，出现的参数。表示物联网平台实例所在地域。
实例	选择要监控的物联网平台实例和产品。产品可多选。
规则名称	设置报警规则名称。
规则描述	定义在监控数据满足何种条件时，触发报警规则。设置项包括： · 选择该规则监控的类目。 · 选择规则执行扫描的周期。如，选择为60分钟周期，表示以60分钟为周期进行扫描。 · 设置报警条件，如连续3个周期内设备数量大于5000时触发报警。
通道沉默时间	指报警发生后，如果未恢复正常，间隔多久再次发送报警通知。
生效时间	报警规则的执行时间范围。报警规则只在生效时间内发送报警通知。
通知方式	设置接收报警的联系组、接收方式等。

如需了解更多阈值报警规则设置，请参见#unique_128/unique_128_Connect_42_section_wfl_pv4_mgb。

创建事件报警规则

云监控事件报警规则可监控的物联网平台事件包括：

· 任一设备每分钟最大连接请求数达到上限。
· 任一设备上行消息QPS达到上限。
· 任一设备下行消息QPS达到上限。
· 当前账号每秒最大连接请求数达到上限。
· 当前账号上行消息QPS达到上限。
· 当前账号下行消息QPS达到上限。
· 当前账号规则引擎数据流转QPS达到上限。

操作步骤：

1. 登录物联网平台控制台。
2. 在左侧导航栏中，选择监控运维 > 实时监控。
3. 单击实时监控页上的报警配置按钮。

4. 在创建报警规则页，单击事件报警迁说明中的查看详情链接。

5. 单击创建事件报警按钮，然后在右侧对话框中，配置报警规则，再单击确定。

创建事件报警页：

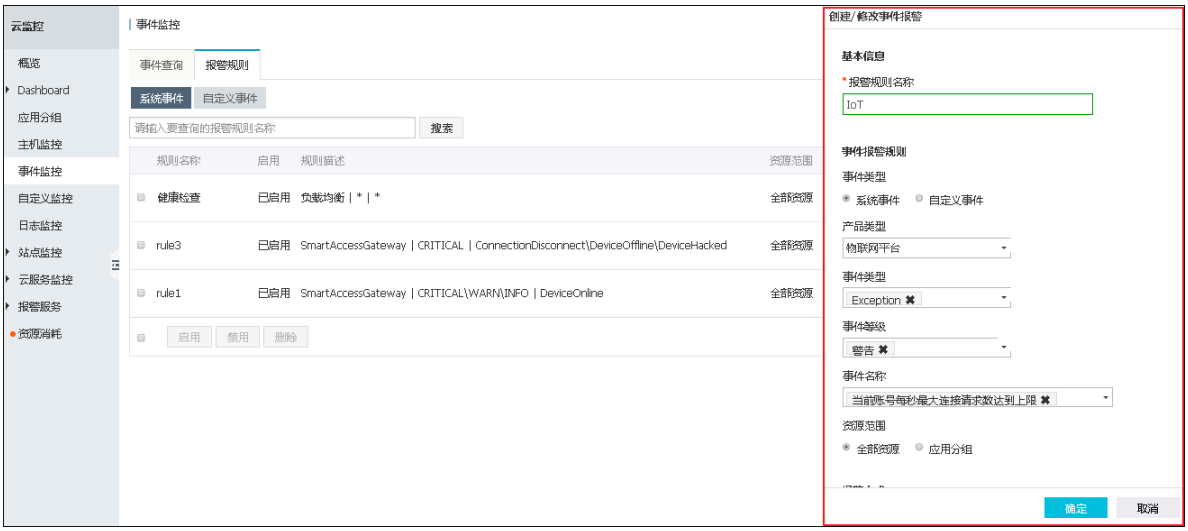


表 3-3: 事件报警规则参数说明

参数	说明
报警规则名称	按提示的规范设置规则名称。
事件类型	选择系统事件。
产品类型	选择物联网平台。
事件类型	选择为全部类型或Exception。
事件等级	可以选择为全部级别或具体事件等级（可多选）。
事件名称	选择该规则要监控的事件，可多选。
资源范围	选择为全部资源。
报警方式	设置报警联系人组和通知方式。

如需了解更多事件报警规则设置，请参见云监控文档#unique_129。

相关文档

#unique_127

3.1.3 报警信息说明

当您的物联网平台资源使用达到报警规则中设置的值后，将触发报警。阿里云将发送报警信息到您设置的联系人通知组。本文介绍报警信息内容。

阈值报警信息

当阈值报警规则被触发后，通知联系人将收到类似如下报警信息邮件。

表 3-4: 报警信息内容说明

字段	说明
物联网平台实例	报警触发对象信息。包含触发报警规则的产品Key（productKey）、实例ID（instanceId）和所在地域ID（regionId）。
报警类目	显示为报警类目名称代码，指阈值报警规则描述中选择的报警类目。 如示例中，“MessageCountForwardedThroughRuleEngine_MNS”对应类目为“规则引擎消息流转次数（MNS）”，表示在一定时间内，规则引擎流转数据到消息服务（MNS）的次数超过了报警规则中设置的阈值。 具体类目名称代码说明，请参见下表：阈值报警类目说明表。
告警时间	发生告警的时间。

字段	说明
计数值	规则所监控类目的消息量、消息流转次数、或在线设备数量。
持续时间	数量超过阈值的持续时长。
规则详情	您在云监控控制台设置的阈值报警规则详情。

表 3-5: 阈值报警类目说明表

类目代码	说明
MessageCountForwardedThroughRuleEngine_DATAHUB	规则引擎消息流转次数（DATAHUB），即规则引擎流转数据到DataHub的次数。
MessageCountForwardedThroughRuleEngine_FC	规则引擎消息流转次数（FC），即规则引擎流转数据到函数计算（Function Compute）的次数。
MessageCountForwardedThroughRuleEngine_MNS	规则引擎消息流转次数（MNS），即规则引擎流转数据到消息服务（Message Service）的次数。
MessageCountForwardedThroughRuleEngine_MQ	规则引擎消息流转次数（MQ），即规则引擎流转数据到消息队列（RocketMQ）的次数。
MessageCountForwardedThroughRuleEngine_OTS	规则引擎消息流转次数（OTS），即规则引擎流转数据到表格存储（Table Store）的次数。
MessageCountForwardedThroughRuleEngine_RDS	规则引擎消息流转次数（RDS），即规则引擎流转数据到云数据库RDS版的次数。
MessageCountForwardedThroughRuleEngine_REPUBLISH	规则引擎消息流转次数（REPUBLISH），即规则引擎流转数据到物联网平台其他Topic的次数。
MessageCountForwardedThroughRuleEngine_TSDB	规则引擎消息流转次数（TSDB），即规则引擎流转数据到时序时空数据库（TSDB）的次数。
MessageCountSentFromIoT_HTTP_2	平台发出的消息量（HTTP/2），即使用物联网平台通过HTTP/2协议发出的消息数量。
MessageCountSentFromIoT_LoRa	平台发出的消息量（LoRa），即使用物联网平台通过LoRa协议发出的消息数量。
MessageCountSentFromIoT_MQTT	平台发出的消息量（MQTT），即使用物联网平台通过MQTT协议发出的消息数量。
MessageCountSentToIoT_CoAP	发送到平台的消息量（CoAP），即使用物联网平台通过CoAP协议发出的消息数量。
MessageCountSentToIoT_HTTP	发送到平台的消息量（HTTP），即设备通过HTTP协议发送到物联网平台的消息数量。
MessageCountSentToIoT_HTTP/2	发送到平台的消息量（HTTP/2），即设备通过HTTP/2协议发送到物联网平台的消息数量。

类目代码	说明
MessageCountSentToIoT_LoRa	发送到平台的消息量（LoRa），即设备通过LoRa协议发送到物联网平台的消息数量。
MessageCountSentToIoT_MQTT	发送到平台的消息量（MQTT），即设备通过MQTT协议发送到物联网平台的消息数量。
OnlineDevicesCount_MQTT	实时在线设备数（MQTT），即使用MQTT协议接入物联网平台的在线设备数量。
DeviceEventReportError	设备事件上报失败数。
DevicePropertyReportError	设备属性上报失败数。
DevicePropertySettingError	设备属性设置失败数。
DeviceServiceCallError	设备服务调用失败数。

事件报警信息

当事件报警规则被触发后，通知联系人将收到类似如下报警信息邮件。


阿里云 首页 | 产品服务 | 备案专区 | 管理控制台 | 用户中心 | 帮助中心

尊敬的[redacted],您好:

IoT 系统事件发生如下报警:

事件名称	报警对象	事件等级	发生时间	事件状态	详情
Device_Connect_QPM_Limit 事件含义	resourceId:acs:iot:cn-shanghai::instance/iot-public/product/a[redacted]/device/T[redacted] 资源名：iot-public 分组id:[]	WARN	20190629T083607.521+0800	Fail	{"productKey":"a[redacted]","deviceName":"[redacted]","instanceId":"iot-public","regionId":"cn-shanghai"}

阿里云计算有限公司

表 3-6: 报警信息内容说明

字段	说明
事件名称	显示为报警事件名称代码。如示例中“Device_Connect_QPM_Limit”表示“任一设备每分钟最大连接请求数达到上限”。 具体事件名称代码对应的事件名称，请参见下表：事件类目说明表。
报警对象	报警触发的资源。 · resourceId：资源ID。信息结构： `acs:iot:$regionid::instance/$instanceId/product/$productKey/device/$deviceName` · 资源名：实例ID。iot-public即公共实例。 · 分组id：设备所属分组ID。若无分组，该字段值为空。
事件等级	目前均为WARN（告警）。
发生事件	事件报警的发生时间。
事件状态	目前均为Fail，表示连接请求数或消息发送QPS达到上限后，后续请求失败。
详情	触发报警的资源信息，JSON格式。包含地域ID（regionId）、实例ID（instanceId）、 产品Key（productKey）和 设备名称（deviceName）。其中，productKey和deviceName仅出现在设备每分钟最大连接请求数达到上限和设备上、下行消息QPS达到上限的报警信息中。

表 3-7: 事件类目说明表

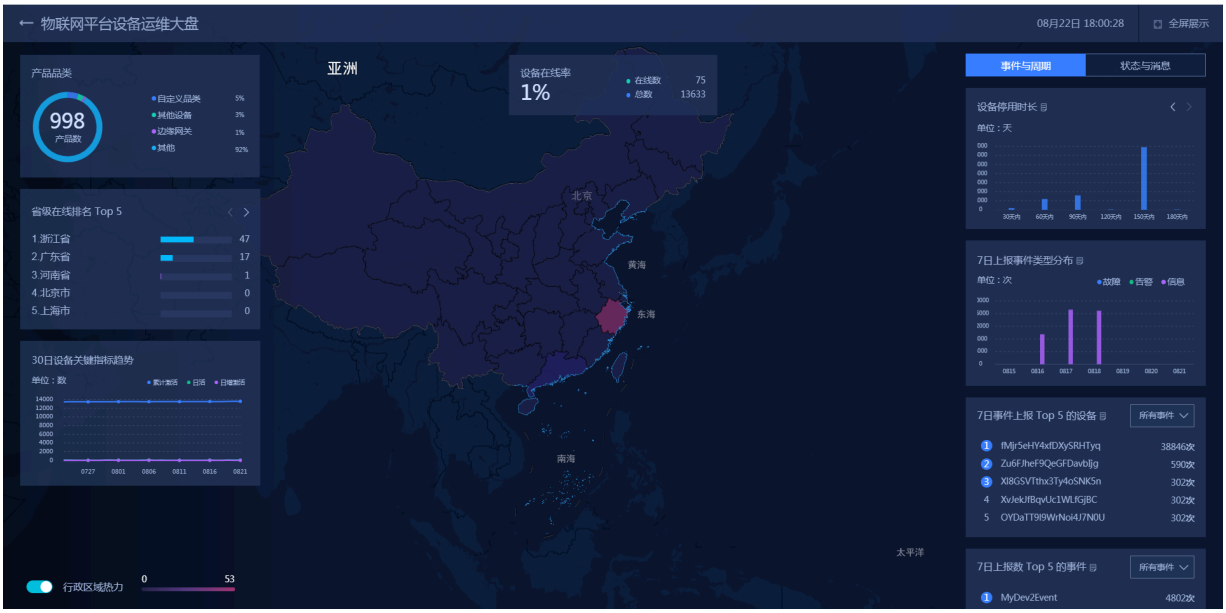
事件名称	说明
Device_Connect_QPM_Limit	任一设备每分钟最大连接请求数达到上限。
Device_Uplink_QPS_Limit	任一设备上行消息QPS达到上限。
Device_Downlink_QPS_Limit	任一设备下行消息QPS达到上限。
Account_Connect_QPS_Limit	当前账号每秒最大连接请求数达到上限。
Account_Uplink_QPS_Limit	当前账号上行消息QPS达到上限。
Account_Downlink_QPS_Limit	当前账号下行消息QPS达到上限。
Account_RuleEngine_DataForward_QPS_Limit	当前账号规则引擎数据流转QPS达到上限。

3.2 运维大盘

物联网平台提供运维大盘，展示设备的全国热力图分布，方便您直观、快速地了解所有设备的状态。

登录物联网平台控制台，左侧导航栏选择监控运维 > 运维大盘，单击设备运维大盘，系统跳转到物联网平台设备运维大盘页面。

物联网平台设备运维大盘通过地图和图标展示您的阿里云主账号下，物联网平台中所有设备的相关实时数据。



设备数据

大盘左侧和上方显示物联网平台设备的各种数据统计。

表 3-8: 数据说明

数据	描述
产品数	物联网平台中的所有产品数量。
产品品类	所有产品的所属分类的种类数量。 说明: 请在产品详情页面查看所属分类。
省级/市级在线排名 Top 5	显示在线设备数量Top 5的省/市。

数据	描述
30日设备关键指标趋势	分别显示近30日内如下三种关键指标的数据，您可以观察数据的趋势。 · 累计激活：当日已经激活过的设备总数量，与设备当前是否在线无关。 · 日活：设备当日上线的次数，与当前是否在线无关。 · 日增激活：当日新激活的设备数量。
设备在线率	（当前在线设备数/物联网平台中的所有设备数）*100%

设备属性与状态数据

数据大盘右侧，展示了设备的事件与周期、状态与消息。

您可以单击相应模块名称右边的  图标，跳转至数据开发页面，查看实现该模块功能的SQL任务模板。开发SQL任务详情请参见数据开发。

表 3-9: 数据说明

数据	描述
事件与周期	
设备停用/使用时长	从设备激活到当前时间，设备累计在线/离线天数在1~30（含）天内、30~60（含）天内、60~90（含）天内等的设备数量。 该数据可用于管理您的设备生命周期。若设备在线时间过长，表示该设备上线已久，您可能需要更新或升级设备。若设备停用时间过长，表示可能不需要在此处布置设备或该设备电源异常， 您可以合理优化设备的使用。
7日上报事件类型分布	显示近7天内，每天设备的故障、告警、信息上报量。
7日事件上报 Top 5 的设备	展示近7天内，上报事件最多的TOP 5设备。可选择显示如下五种事件的TOP 5设备。 · 所有事件； · 故障事件； · 告警事件； · 信息事件。

数据	描述
7日上报数 Top 5 的事件	显示近7天内，被上报最多的TOP 5事件类型。可选择显示所有事件、故障事件、告警事件、信息事件的类型。  说明: 事件类型的说明请参见物模型中的事件（Event）相关解释。
状态与消息	
设备日上下线次数 Top 5	显示当日上线和下线次数最多的Top 5设备。  说明: 设备日上下线次=设备当日上线次数+设备当日下线次数
设备日平均延迟 Top 5	显示当日平均通信延迟时间最长的Top 5设备。  说明: 通信延迟时间表示从物联网平台发送与设备建立连接的消息时间，到物联网平台收到设备的ACK （Acknowledge character）确认字符的时间。 设备的平均通信延迟时间=（与该设备的第一次通信延迟时间+第二次通信延迟时间+……+第n次通信延迟时间）/n次通信
设备日消息量 Top 5	当日订阅或发布消息最多的Top 5 Topic类。  说明: Topic类详细说明请参见#unique_133。

3.3 在线调试

3.3.1 调试真实设备

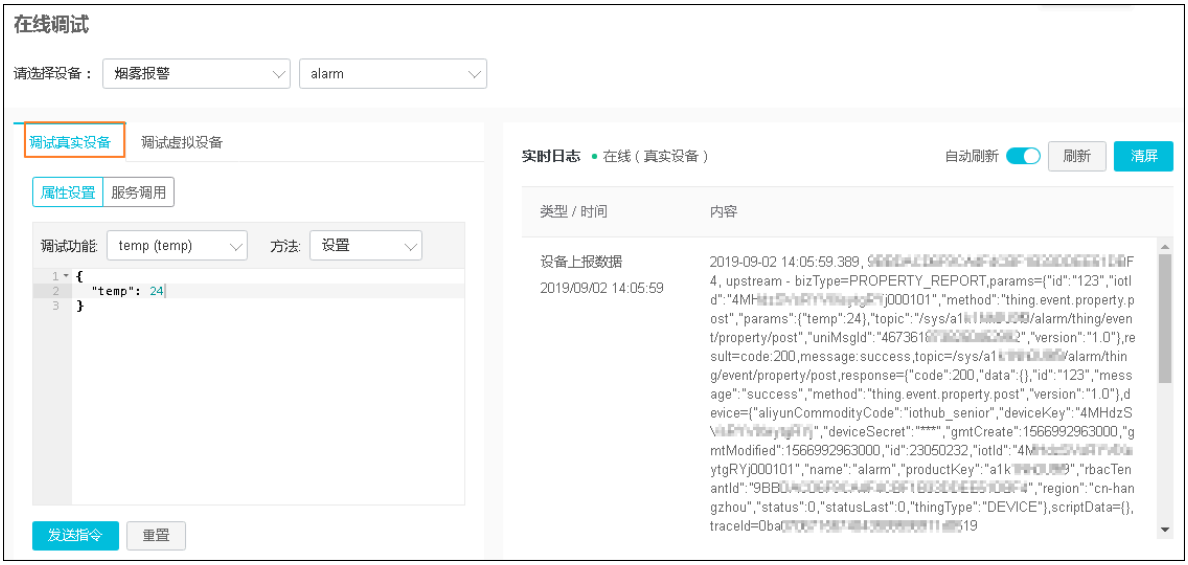
设备端开发完成后，您可以使用物联网平台的在线调试功能，从控制台下发指令给设备端进行功能测试。真实设备调试仅支持MQTT连接。

操作步骤

1. 在物联网控制台左侧导航栏，选择监控运维 > 在线调试。
2. 在在线调试页，选择本次调试的设备。

   选择设备后，页面会自动跳转至调试设置页。
3. 选择调试真实设备。

4. 推送调试指令。



可在真实设备上调试的物模型通信如下表。

调试项目	操作步骤
设置属性	从云端下发设置属性值的指令给设备。设备收到指令后，设置属性值，并将最新属性值上报给云端。 a. 单击属性设置。 b. 从调试功能选项表中，选择要调试的属性，并选择方法为设置。 选择完成后，输入框中将自动显示该属性的数据格式，如`{"Temperature":0}`。 c. 设置一个属性值，单击发送指令。
获取属性	从设备上获取指定属性的值。 a. 单击属性设置。 b. 从调试功能下拉选项中，选择要调试的属性，并选择方法为获取。 c. 单击发送指令。 **说明:** 调试获取属性时，无需在输入框中输入任何数据。 指令发送成功后，输入框中将显示获取到的最新属性数据。如果设备上没有该属性的数据，则数据为空。

调试项目	操作步骤
调用服务	a. 单击服务调用。 b. 从调试功能下拉选项中，选择要调试的服务。 c. 在输入框中，输入调用服务的入参，单击发送指令。 输入的服务入参数据，需为标准的JSON格式，如`{"Switch":0}`。

预期结果

推送指令后，可在页面右侧实时日志下查看操作日志。

3.3.2 调试虚拟设备

物联网平台提供虚拟设备功能，供云端应用开发测试使用。使用虚拟设备可调试：上报属性、上报事件、设置属性和调用服务。并且，虚拟设备调试支持数据格式为透传/自定义的设备。

背景信息

物联网正常开发流程是：设备端开发完成，设备上报数据，云端接收数据，云端开始开发工作。这样的开发流程战线较长，耗时较久。物联网平台提供虚拟设备功能，虚拟设备模拟真实设备与物联网平台建立连接，上报属性及事件处理。您可以根据虚拟设备的数据，完成应用的开发调试。

使用限制：

- 连续推送的最小时间间隔为1秒。
- 最多连续推送1000条消息。
- 每天最多可使用100次推送按钮推送调试信息。
- 如果设备的数据格式为透传/自定义，输入二进制数据Base64编码后的字符串，长度不超过4096字符。
- 真实设备在线或设备被禁用时，不能启动虚拟设备。真实设备上线后，虚拟设备会自动下线。

操作步骤

1. 登录物联网平台控制台。
2. 在左侧导航栏，选择监控运维 > 在线调试。
3. 在在线调试页，选择本次调试的设备。

   选择设备后，页面会自动跳转至调试设置页。
4. 单击调试虚拟设备 > 启动虚拟设备。

5. 推送调试指令。

调试项目	操作步骤
上报属性	a. 单击属性上报。 b. 在属性对应的输入框中，输入属性值。 可输入符合属性数据类型和取值范围的值，或使用random()函数生成随机值。 c. 推送指令。 可选择推送方式： · 推送：立即推送数据。 · 策略推送： 设置推送策略。 - 定时推送：在设置好的时间推送数据，仅推送一次。 - 连续推送：在设置好的时间段内，按照固定时间间隔，推送数据。时间间隔单位为秒。
上报事件	a. 单击事件上报。 b. 在事件对应的输入框中，输入事件值。 可输入符合事件数据类型和取值范围的值，或使用random()函数生成随机值。 c. 策略推送： 设置推送策略。 · 定时推送：在设置好的时间推送数据，仅推送一次。 · 连续推送：在设置好的时间段内，按照固定时间间隔，推送数据。时间间隔单位为秒。

调试项目	操作步骤
设置属性	从云端下发设置属性值的指令给设备。设备收到指令后，设置属性值，并将最新属性值上报给云端。 a. 单击属性设置。 b. 从调试功能选项表中，选择要调试的属性，并选择方法为设置。 选择完成后，输入框中将自动显示该属性的数据格式，如`{"Temperature":0}`。 c. 设置一个属性值，单击发送指令。
获取属性	从设备上获取指定属性的值。 a. 单击属性设置。 b. 从调试功能下拉选项中，选择要调试的属性，并选择方法为获取。 c. 单击发送指令。 说明: 调试获取属性时，无需在输入框中输入任何数据。 指令发送成功后，输入框中将显示获取到的最新属性数据。如果设备上没有该属性的数据，则数据为空。
调用服务	a. 单击服务调用。 b. 从调试功能下拉选项中，选择要调试的服务。 c. 在输入框中，输入调用服务的入参，单击发送指令。 输入的服务入参数据，需为标准的JSON格式，如`{"Switch":0}`。

预期结果

推送数据后，可在页面右侧实时日志下查看操作日志。

数据推送成功后，可前往设备的设备详情页。在运行状态页签下，查看设备上报的属性信息；在事件管理页签下，查看上报的事件信息。在服务调用页签下，查看服务调用记录。

说明:
选择策略推送时，在设定的时间推送数据后，相应的页面上才会显示对应日志、属性或事件信息。

3.4 日志服务

物联网平台提供日志服务功能。您可以在物联网平台控制台日志服务页，查询设备日志。本文主要介绍设备日志中的错误码和排错方法。

查询设备日志

日志分为四类：

- 设备行为分析
- 上行消息分析
- 下行消息分析
- 物模型数据分析

查询设备日志途径：

1. 登录物联网平台控制台。
2. 在左侧导航栏，单击监控运维 > 日志服务。
3. 选择产品、日志类型，输入设备名称等搜索条件，然后单击搜索。

支持的搜索条件：

搜索条件	说明
DeviceName	输入设备名称。可以根据设备名称，搜索出该设备的相关日志。
内容关键字	输入日志内容关键字搜索具体日志。
MessageID	消息ID，物联网平台为消息生成的唯一标识符。可以用MessageID追踪到这条消息全链路的流转状况，和查询消息内容。
状态	查询某种结果状态的日志。可选择： · 全部状态 · 成功 · 失败
时间范围	选择要查询日志的时间范围。

说明：

日志状态码说明：200表示成功，其余表示失败。失败状态码说明，请参见以下各章节。

设备行为分析

设备行为主要有设备上线（online）和设备下线（offline）的日志。

错误码	含义	原因	排查
400	请求错误。	原因可能是： · 被使用相同设备证书的设备踢下线。 物联网平台仅以设备证书信息（productKey、deviceName、deviceSecret）来判断设备。 具体原因可能是： - 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。 · 设备在云端已被删除。 · 设备在云端已被禁用	· 在控制台，设备的设备详情页，查看设备的激活时间，根据时间判断是否有设备使用相同证书接入物联网平台。 · 在控制台设备列表下，搜索设备，查看设备是否已被删除。 · 在控制台，查看对应的设备状态，是否显示为已禁用。

上、下行消息分析

· 上行消息日志主要包括：设备发送消息到Topic，消息流转到规则引擎，和规则引擎数据流转功能转发消息到其他Topic和阿里云其他云产品。
· 下行消息日志主要是云端发送消息到设备的日志。

错误码	含义	原因	排查
1901	受限于网络环境（如tcp write buffer拥堵等），消息发送失败。	设备端接受消息的通道阻塞，可能由于网络慢或者设备端消息能力不足，导致了服务端发送消息失败。	检查网络情况和设备端消息消费能力。
1902	消息写入网络时，发生异常。	网络异常导致发送失败。	检查网络情况。
1903	Topic格式错误。	消息Topic格式错误。	核对Topic格式。

错误码	含义	原因	排查
1904	云端收到无效的RRPC响应。	云端收到的RRPC响应没有对应的RRPC请求。可能是之前的请求已经超时失败。	检查设备端回复的RRPC响应，是否已经超时。
1905	云端等待设备响应RRPC超时。	云端下发的RRPC请求，没有在超时时间内收到设备端的RRPC响应。	检查设备端上收的RRPC请求，是否已及时响应。
1950	消息写入时，网络连接发生异常。	网络错误导致发送消息失败。	检查网络状况。
1951	未知的响应类型。	设备端向云端发送了未知类型的消息。	检查设备发送的消息类型。如果您使用的是阿里云设备端SDK，请联系客服或提交工单处理。
9200	设备未激活。	设备没有在物联网平台激活。新设备注册后，需要设备成功接入物联网平台，并进行数据上报才会激活设备。	在控制台，查看设备状态。
9201	设备离线。	设备不在线。	在控制台，查看设备状态。
9236	Topic鉴权失败。	发布或订阅消息的Topic对应的权限不一致。	在控制台，设备的Topic列表中，Topic权限是否正确，即用于发布消息的Topic，权限须为发布；订阅消息的Topic，权限须为订阅。
9324	限流。	设备或者租户流转请求过多。	降低消息发送频率，或者咨询客服。
9321	参数非法。	传递的请求参数不合法，包括其他云产品参数。	根据提示检查对应的参数设置。
9320	payload非法	设备发送的消息体的格式不合法。	检查消息体的格式是否规范。
9331	消息流转的目标云产品内部错误。	数据流转目标云产品的内部发生错误。	根据内容中的错误码，到对应云产品官网上查询或者联系客服。
9332	云产品配置异常。	设置消息流转时，配置有误，导致连接目标云产品服务时出错。	检查数据流转规则，查看目标产品的配置是否正确，资源是否存在。根据内容中的错误码，到对应云产品官网上查询原因和处理方法。

错误码	含义	原因	排查
9333	云产品授权错误。	授予物联网平台访问目标云产品的权限可能有误。	检查您的阿里云RAM授权策略。
9399	服务器内部未知错误。	物联网平台内部错误。	请联系客服或提交工单。

物模型数据分析

物的模型数据分析日志，包含上报属性或事件日志、属性设置日志、服务调用日志、属性或服务调用的回复日志。

透传/自定义数据格式产品下的物模型数据分析，除了展示日志的内容之外，还会显示设备上报的原始数据的16进制字符串格式内容。

日志格式说明

参数	说明
id	Alink协议的消息ID，用于标识云端和设备端通信的消息。
params	请求参数。
code	返回的结果代码。
method	请求方法。
type	消息上下行类型。 upstream上行，downstream下行。
scriptData	数据格式为透传（自定义）时，展示数据解析转换的入参和出参。
downOriginalData	数据格式为透传（自定义）时，需进行数据解析的下行原始数据，格式为Alink JSON。
downTransformedData	数据格式为透传（自定义）时，下行数据经数据解析后的数据，格式为十六进制字符串。
upOriginalData	数据格式为透传（自定义）时，需进行数据解析的上行原始数据，格式为十六进制字符串。
upTransformedData	数据格式为透传（自定义）时，上行数据经数据解析后的数据，格式为Alink JSON。

调用服务和设置属性失败错误码：

调用服务时，物联网平台会通过设备的模型描述TSL校验该服务的入参是否符合该服务在TSL中的定义。

错误码	含义	原因	排查
9201	设备已下线。	设备不在线时，会报这个错。	在控制台，查看设备的在线状态。
9200	设备没有激活。	设备没有在物联网平台激活。新设备注册后需要进行数据上报。	在控制台，查看设备的在线状态。
6208	设备被禁用。	设备被禁用时，属性设置、服务调用被禁止使用。	在控制台，查看设备的状态。如果设备被禁用，启用该设备然后重试操作即可。
6300	TSL校验时，method不存在。	TSL校验时，设备上报的Alink（标准）格式数据，或者自定义（透传）格式数据经过脚本转换后，没有Alink协议要求的method参数。	查看设备属性上报的日志，查看上报的数据。 或者查看设备的本地日志，查看上报的数据。
6206	查询服务定义出错。	调用服务时，会查询服务的定义信息，如果服务不存在会报这个错误。	在控制台产品详情中，查看设备所属产品的功能定义，查看传入的服务是否存在。如果存在，请校验传入的参数中是否包含不可见字符。
6200	脚本不存在。	对于透传（自定义）格式的产品，下行服务调用时，会调用产品脚本进行数据的转换。如果脚本不存在，会报这个错误。	在控制台产品详情中，查看产品的脚本是否存在。如果存在，请重新保存脚本后再尝试操作。
6201	脚本转换后数据为空。	脚本执行正常，但是脚本中返回的数据为空。如rawDataToProtocol返回null， protocolToRawData返回null或者空数组。	查看脚本的内容，确认在什么情况下返回数据为空。
6207	数据的格式错误。	下行同步调用时，或者设备上报数据时出现。 下行同步调用时，可能有如下原因： · 设备返回的数据格式错误。 · 对自定义/透传格式数据进行解析后的数据格式错误。 · 服务调用传入的参数格式不正确。	参见API接口文档及TSL，查看服务需要的数据格式。同时参见Alink协议文档，查看对应的数据格式。

错误码	含义	原因	排查
系统异常错误码			
5159	获取TSL中属性信息时报错。	系统异常。	可以通过提交工单排查。
5160	获取TSL中事件信息时报错。		
5161	获取TSL中服务信息时报错。		
6661	查询租户信息时异常。		
6205	下行服务调用异常。		

上报属性和上报事件失败错误码：

属性上报、事件上报时会通过物的模型描述TSL校验，校验属性是否符合TSL中属性的定义，事件的传入参数是否符合TSL中事件的定义。

错误码	含义	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。	查看设备属性上报的日志，检查上报的属性个数。或者查看设备本地的日志，查看上报的数据。
6300	TSL校验时，method不存在。	TSL校验时，设备上报的Alink（标准）格式数据，或者自定义（透传）格式数据经过脚本转换后，没有Alink协议要求的method参数。	查看设备属性上报的日志，查看上报的数据。 或者查看设备的本地日志，查看上报的数据。
6320	TSL校验时，属性信息不存在。	查询设备的TSL时，没有查询到设备的属性信息。	在控制台产品详情中，查看设备所属产品的功能定义，查看属性定义是否存在。不存在时，定义相应的属性。
6450	Alink协议格式的数据中method不存在。	设备上报的Alink标准格式数据，或者自定义/透传格式数据经过脚本解析为Alink标准格式数据后无method。	查看设备属性上报的日志，检查设备上报的数据中是否有method参数。或者查看设备本地的日志。

错误码	含义	原因	排查
6207	数据的格式错误。	下行同步调用时，或者设备上报数据时出现。 设备上报数据时，可能原因为：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Alink协议文档，查看对应数据格式，并按格式要求上报数据。
系统异常错误码			
6452	限流。	请求量过大，触发限流。	可以通过提交工单排查。
6760	租户的空间存储超出配额。	系统异常。	可以通过提交工单排查。

调用服务和设置属性的reply失败错误码：

错误码	含义	原因	排查
通用错误码			
460	参数错误。	请求的参数错误。	可以通过提交工单排查。
500	系统内部异常。	系统发生未知异常。	可以通过提交工单排查。
400	服务请求时报错。	调用服务时发生未知错误。	可以通过提交工单排查。
429	请求过于频繁。	请求过于频繁，触发系统限流时会报这个错。	可以通过提交工单排查。
系统异常错误码			
6452	限流。	请求量过大，触发限流。	可以通过提交工单排查。

TSL公共错误码

调用服务、上报属性、事件上报时，会通过设备的模型描述TSL校验，校验该服务的入参是否符合该服务的定义，属性是否符合属性的定义，事件的传入参数是否符合事件的定义。

错误码	含义	原因	排查
6321	TSL中，属性对应的标识符identifier不存在。	系统异常。	可以通过提交工单排查。

错误码	含义	原因	排查
6317	TSL模型有错误。	系统异常。	可以通过提交工单排查。
6302	参数不存在。	TSL校验服务的入参时，服务要求的参数没有传。	在控制台产品详情中，查看设备所属产品的功能定义。服务查询调用、属性设置，查询对应服务的入参，核对传入的参数。
6306	传入的参数，不符合TSL整形数据的规范。	TSL校验时： · 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。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型。
6307	传入的参数，不符合TSL中32位浮点数据的规范。	TSL校验时： · 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。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型和参数的取值范围。
6322	传入的参数，不符合TSL中64位浮点数据的规范。	TSL校验时： · 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。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型和参数的取值范围。
6308	传入的参数，不符合TSL中布尔类型数据的规范。	TSL校验时： · 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。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，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。	TSL校验时，参数类型和TSL中定义的类型不一致。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。

错误码	含义	原因	排查
6310	传入的参数，不符合TSL中字符类型数据的规范。	TSL校验时： · 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。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6311	传入的参数，不符合TSL中日期类型数据的规范。	TSL校验时： · 传入的参数类型，需要和TSL中定义的类型完全一致。 · 传入的日期类型判断不是UTC时间戳的字符格式时会报错。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
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： · 传入的参数类型，需要和TSL中定义的类型完全一致。 · 结构体类型中参数的个数和TSL中定义不一致时会报这个错。	在控制台产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6304	校验的参数，在TSL结构体中不存在。	TSL校验时，传入的参数在结构体中不存在。	在控制台产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6324	校验参数时，数组类型的参数不符合规范。	TSL校验时： · 传入的数组类型的参数不符合TSL定义时，会报这个错。 · 数组中参数个数超过了TSL中定义的最大个数。	· 在控制台产品详情中，查看设备所属产品的功能定义，检查对应数组的定义。 · 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。
6328	校验参数时，传入的参数不是数组类型。	TSL校验时，传入的参数如果不是数组类型，会报这个错。	在控制台中产品详情中查看设备所属产品的功能定义，查看对应服务的入参，查询类型为数组的参数，然后检查传入的对应参数是否是数组类型。

错误码	含义	原因	排查
6325	校验参数时，传入的数组类型参数中的元素类型，目前不支持该类型。	TSL校验参数时报错，数组中元素的类型目前只支持整形、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目前支持的类型。
系统异常错误码			
6318	TSL解析时系统异常。	系统异常。	可以通过提交工单排查。
6329	校验参数时，TSL中数组规范解析出错。		
6323	TSL中参数规范格式错误。		
6316	TSL中解析参数报错。		
6314	TSL不支持的数据类型。		
6301	通过TSL校验参数格式时报错。		
数据解析脚本相关			
26010	请求过于频繁被限流。	请求过于频繁。	可以通过提交工单排查。
26001	脚本内容为空。	执行脚本时获取脚本内容，不存在。	在控制台查询产品的脚本是否存在。如果存在，则是否正常保存。应是正式的脚本，不是草稿。
26002	脚本执行时异常。	脚本执行正常，但脚本编写有问题，如脚本中语法错误。	在控制台使用相同的参数去执行脚本，查看具体的错误信息，修改脚本。注意控制台只提供了脚本的基础运行环境，并不会对脚本进行详细的校验。建议脚本需在本地经过详细的自验后，再进行保存。

错误码	含义	原因	排查
26006	脚本执行时必要的方法不存在。	脚本执行正常，脚本内容有误。脚本编写要求有protocolToRawData和rawDataToProtocol这两个服务，如果执行时不存在，会报错。	在控制台查询脚本的内容，查看protocolToRawData和rawDataToProtocol服务是否存在。
26007	脚本执行时返回的结果格式不正确。	脚本执行正常，但返回的结果不符合格式要求。 脚本编写要求有protocolToRawData和rawDataToProtocol这两个服务。 protocolToRawData返回byte[]数组， rawDataToProtocol要求返回JSON对象。如果脚本返回的结果不符合这两种格式，返回时会报这个错。 如设备上报数据后，会返回结果给设备。返回的结果也会经过脚本进行解析，如果对于返回结果不做处理，可能会返回不符合要求的类型。	在控制台查看脚本，获取脚本内容。按照输入参数，在本地执行脚本并查看返回结果的格式是否符合要求。

3.5 固件升级

3.5.1 推送固件到设备端

物联网平台提供固件升级与管理服务。首先确保设备端支持OTA服务，然后在控制台上传新的固件，并将固件升级消息推送给设备，设备即可在线升级。本文介绍如何在物联网平台控制台，新增固件、验证固件和向设备批量推送固件。

前提条件

使用固件升级功能前，请确保设备端支持OTA升级服务。

- 如果您使用设备端SDK，请参见#unique_146。
- 如果您的设备搭载AliOS Things芯片，请参见AliOS Things技术文档。

背景信息

固件升级相关限制说明如下：

- 一个阿里云账号下最多可有500个固件。
- 固件文件大小不能超过1,000 MB，且仅支持bin、tar、gz、tar.gz、zip、gzip类型的文件。

- 升级批次限制说明。

  升级批次：物联网平台将已发起的各个升级任务展示为不同的升级批次。您可以在固件详情页的批次管理页签下，查看该固件的升级批次。

  - 使用一个固件，可同时发起多个批次升级。
  - 一个设备同时只能在一个正在进行的升级批次中（设备处于待升级或正在升级状态）。
  - 使用同一个固件，只能对同一个待升级版本发起一个动态升级批次。
  - 使用不同固件，可对同一个待升级版本发起多个动态升级批次。但是，如果一个设备同时满足多个动态升级策略时，仅执行最新发起的升级策略。
- 仅支持通过MQTT协议接入物联网平台的设备使用固件升级功能。
- 设备在线时可以立即接收到升级通知；不在线的设备下次上线时，系统会再次推送升级通知。

操作步骤

1. 登录物联网平台控制台。
2. 在左侧导航栏，选择监控运维 > 固件升级。

   说明:
   为提供更好的服务，物联网平台改版了原固件升级，新增了产品版本统计功能。首次进入改版后的固件升级页面时，您需要手动将之前上传的固件与产品进行关联。固件与产品一一对应，一个固件只能关联一个产品。详情请参见控制台指引。
3. （可选）若您的设备搭载了AliOS Things芯片，可以开启安全升级功能。

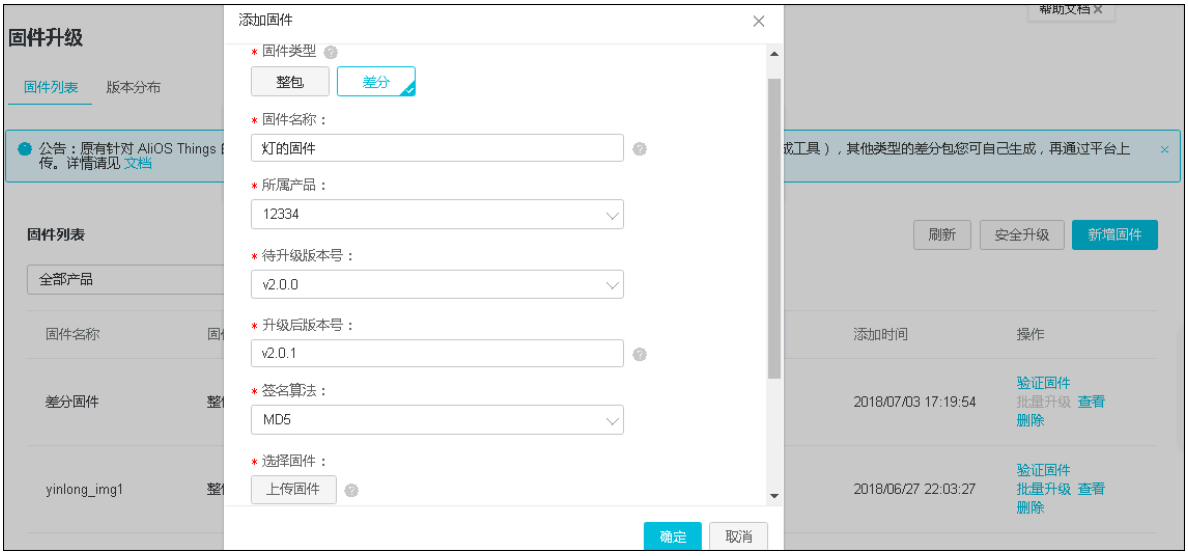
   安全升级是保证固件完整性、机密性的一种方式，建议打开。使用安全升级功能，设备端需配合对固件和固件的签名进行验证。具体请参见AliOS Things技术文档。

   a) 在固件升级页，单击安全升级。

   b) 在对话框中，将待升级产品对应的安全升级按钮设置为开。

   当安全升级功能为开时，可单击对应的复制按钮，复制公钥，用于设备端签名。
4. 在固件升级页，单击新增固件。

5. 在添加固件对话框中，输入固件信息，上传固件文件。



参数	描述
固件类型	· 整包：您上传的固件文件是完整的固件文件，将推送整包固件给设备进行升级。 · 差分：您上传的固件文件仅包含新版本固件与之前版本的差异部分，仅推送差异部分至设备进行升级。您需自行生成差分升级包。 差分升级可有效减少升级对设备资源的占用，和减少下发固件的流量消耗。 如果设备使用AliOS-Things芯片，差分升级包的生成方法，可参见OTA差分工具使用指南。
固件名称	设置固件名称。仅支持中文、英文字母、数字和下划线（_），且不能以下划线（_）开头。长度限制为4~32字符。
固件版本号	设置该固件的版本号。仅支持英文字母、数字、点号、连字符（-）和下划线（_）。长度限制为1~64字符。 固件类型选择为整包时，需设置的参数。
待升级版本号	选择待升级的固件版本号。下拉选项框中，将展示当前产品下所有设备的固件版本号，选择一个或者多个待升级的固件版本。 固件类型选择为差分时，需设置的参数。
升级后版本号	设置升级后的固件版本号。 固件类型选择为差分时，需设置的参数。
所属产品	选择固件所属产品。
签名算法	仅支持MD5和SHA256。

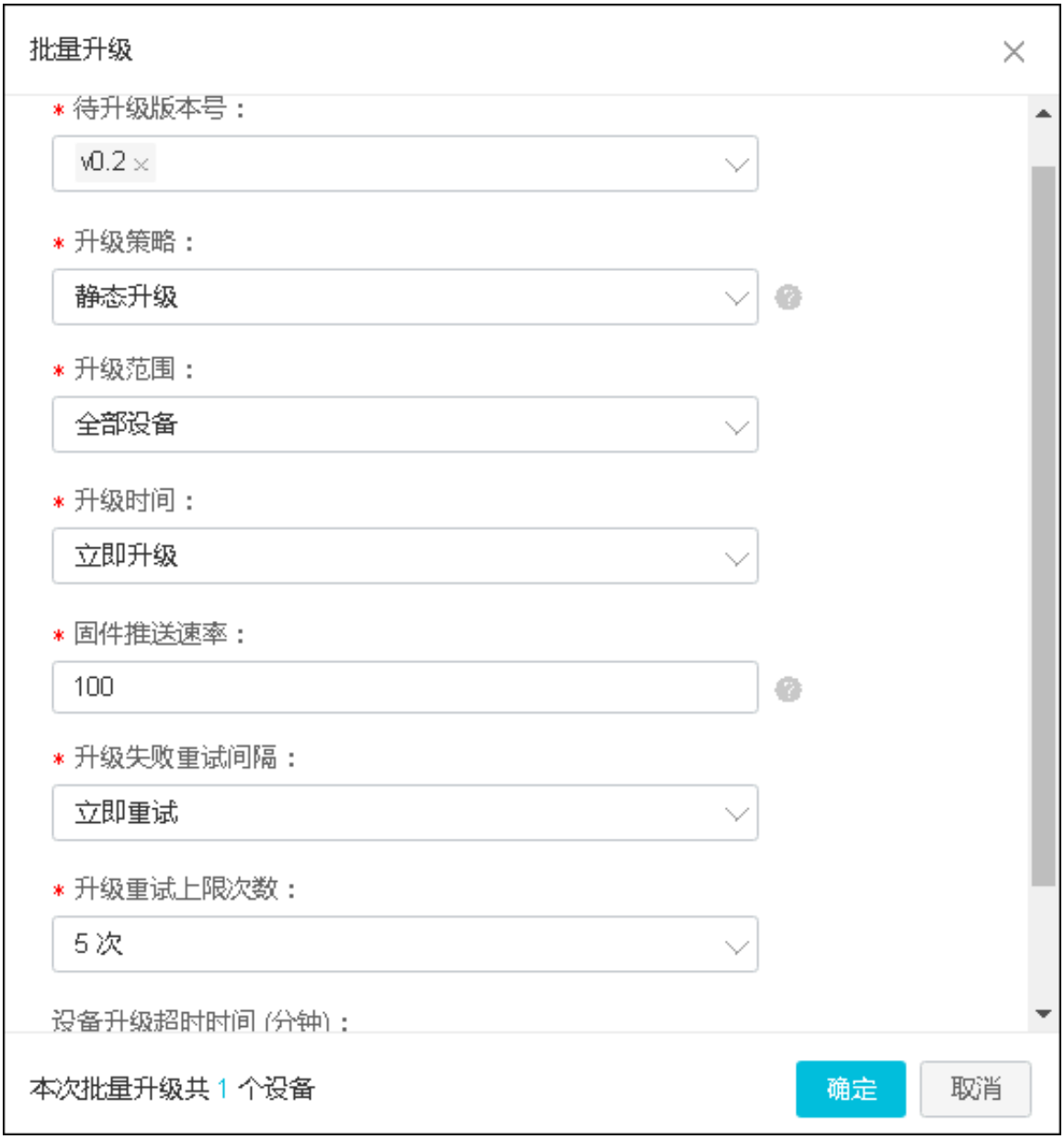
参数	描述
上传固件	上传固件文件。文件大小不能超过1,000 MB，仅支持bin、tar、gz、tar.gz、zip、gzip类型的文件。

6. 在固件列表中，单击固件对应的验证固件按钮，然后在一个或多个设备上进行固件测试。

> 说明:
> 固件上传至物联网平台后，必须使用少量设备对固件进行验证。确认测试设备升级成功后，才能批量升级。

参数	描述
待升级版本号	下拉选项框中，展示当前产品下所有设备的固件版本号，选择一个或者多个待升级的固件版本。 选择待升级版本号后，使用这些固件版本的设备将展示在待验证设备列表中。
待验证设备	选择用于此次测试的设备。
设备升级超时时间	设置单个设备的升级超时时间，即多长时间之后，升级未完成则为超时。从设备第一次上报升级进度开始计算时间。可选值范围：1~1440分钟。

7. 固件验证通过后，单击批量升级按钮，设置参数，批量向设备推送升级通知。



参数	描述
待升级版本号	下拉选项框中，展示当前产品下所有设备的当前固件版本号，从中选择待升级的固件版本。 整包升级时，需设置该参数。

参数	描述
升级策略	· 静态升级：仅升级满足指定条件，且已激活的设备。 · 动态升级：动态升级将持续维护需升级的设备，包括当前已经上报固件版本号的设备和新激活的设备。  说明: 一个固件下只能有一个动态升级批次。如果固件下已有一个动态升级批次，将不能创建新的动态升级，需先取消原有动态升级批次。
升级范围	· 全部设备：升级该产品下的全部设备。 · 定向升级：选择为定向升级后，下方出现设备范围选项框。单击选项框，在右侧弹出的对话框中，选择要升级的设备。仅升级被选中的设备。  说明: 定向升级的待升级版本为多选。默认选中您之前已输入的待升级版本号。 · 区域升级：升级实际地理位置在指定区域的设备。选择为区域升级后，下方出现指定区域的省份和城市选项框。 · 灰度升级：即局部升级。升级策略选择为静态升级时出现的可选项。 选择为灰度升级后，下方出现灰度范围输入框，需针对已选择的设备，设置灰度百分比。物联网平台根据设置的灰度百分进行计算，计算结果向下取整。灰度升级的设备至少为1个。
升级时间	指定设备固件升级的时间。 · 立即升级：立即进行固件升级。 · 定时升级：需设定升级时间。定时时间范围是5分钟~7天。  说明: 仅当升级策略为静态升级时，支持定时升级。
固件推送速率	设置每分钟向多少个设备推送固件下载URL。取值范围：10~1000。

参数	描述
升级失败重试间隔	如果升级失败，在什么时候进行重试升级。可选： · 不重试 · 立即重试 · 10分钟后重试 · 30分钟后重试 · 1小时候重试 · 24小时候重试
升级重试上限次数	选择升级失败后，最多可重试几次。可选： · 1次 · 2次 · 5次
设备升级超时时间	设置单个设备的升级超时时间，即多长时间之后，升级未完成则为超时。从设备第一次上报升级进度开始计算时间。可选值范围：1~1440分钟。

预期结果

批量升级提交后，单击该固件的查看，然后在批次管理页签下，查看升级状态。

- 待升级：已设置设备固件升级，但固件升级未开始。两种待升级状态：待升级（设备离线）和待升级（排队中）。
- 升级中：设备收到升级通知，并已上传升级进度。
- 升级成功：本次升级成功的设备。
- 升级失败：本次升级失败的设备及升级失败原因简述。

  以下原因可能造成设备升级失败：

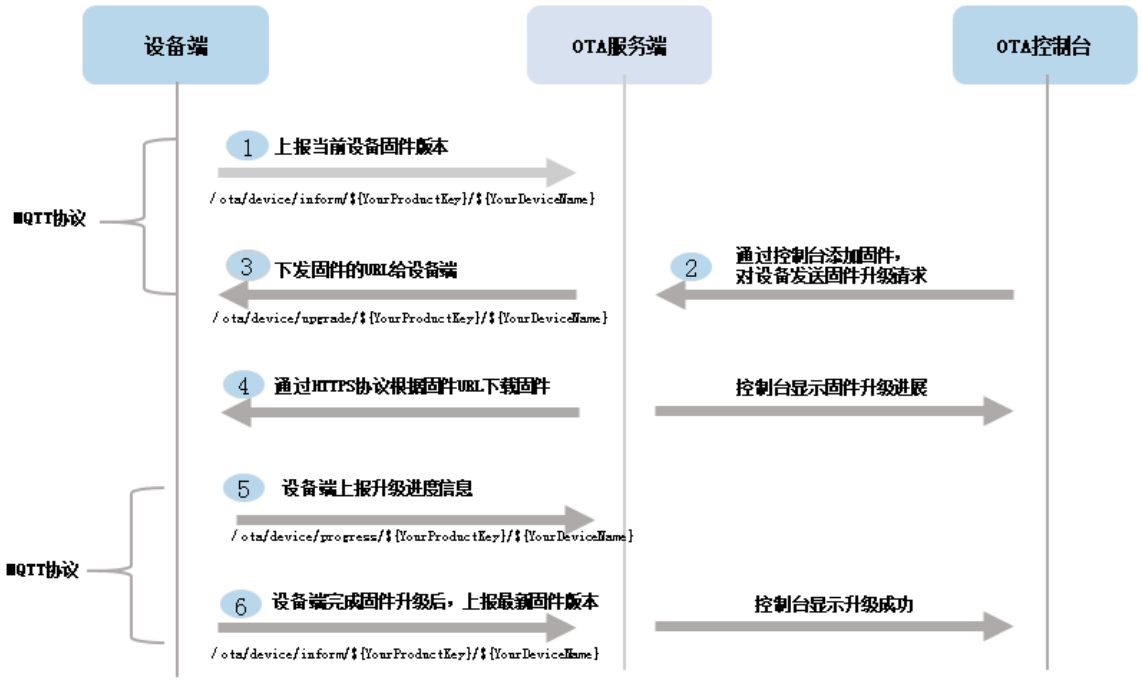
  - 若设备未结束上一次升级动作，又推送新的升级，会升级失败。可以等设备完成上一次升级动作后，再次尝试。
  - 设备在实际升级过程中，出现如下载失败、校验失败、解压失败等错误，可以尝试再次升级。

3.5.2 设备端OTA升级

OTA（Over-the-Air Technology）即空中下载技术。物联网平台支持通过OTA方式进行设备固件升级。本文以MQTT协议下的固件升级为例，介绍OTA固件升级流程、数据流转使用的Topic和数据格式。

OTA固件升级流程

MQTT协议下固件升级流程如下图所示。



固件升级过程使用的Topic如下列表。

- 设备端通过以下Topic上报固件版本给物联网平台 。

```
/ota/device/inform/${YourProductKey}/${YourDeviceName}
```

- 设备端订阅以下Topic接收物联网平台的固件升级通知 。

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

- 设备端通过以下Topic上报固件升级进度 。

```
/ota/device/progress/${YourProductKey}/${YourDeviceName}
```

说明:

- 设备固件版本号只需要在系统启动过程中上报一次即可，不需要周期循环上报。
- 从物联网平台控制台发起批量升级后，设备升级操作记录状态是待升级。

  实际升级以物联网平台OTA系统接收到设备上报的升级进度开始。设备升级操作记录状态是升级中。
- 根据版本号来判断设备端OTA升级是否成功。
- 设备离线时，不能接收服务端推送的升级消息。

  通过MQTT协议接入物联网平台的设备再次上线后，主动通知服务端上线消息。OTA服务端收到设备上线消息，验证该设备是否需要升级。如果需要升级，再次推送升级消息给设备， 否则，不推送消息。

数据格式说明

设备端OTA开发流程和代码示例，请参见Link Kit SDK文档Link Kit SDK文档中，各语言SDK文档中关于设备OTA开发章节。

1. 设备连接OTA服务，必须上报版本号。

   设备端通过MQTT协议推送当前设备固件版本号到Topic：`/ota/device/inform/${YourProductKey}/${YourDeviceName}`。消息内容格式如下：

```
{
  "id": 1,
  "params": {
    "version": "xxxxxxxx"
  }
}
```

   - id：消息ID号。
   - version：设备当前固件版本号。

2. 在物联网平台控制台上，添加固件、验证固件和发起批量升级固件。

   具体操作，请参见#unique_18。

3. 您在控制台触发升级操作之后，设备会收到物联网平台OTA服务推送的固件的URL地址。

   设备端订阅Topic：`/ota/device/upgrade/${YourProductKey}/${YourDeviceName}`。控制台对设备发起固件升级请求后，设备端会通过该Topic收到固件的URL。消息格式如下：

```
{
  "code": "1000",
  "data": {
    "size": 432945,
    "version": "2.0.0",
    "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com
/nopoll_0.4.4.tar.gz?    Expires=1502955804&OSSAccessKeyId=
XXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&
security-  token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5
D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasu
mZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%
2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K
%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTknxR7ARasaBq
helc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNz
BJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McKARWct06MWV
9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK8CfNOkfTuc
MGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6IzvJsClXTnbJ
BMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2FoaGk6BAF8
hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e"
  },
  "id": 1507707025,
  "message": "success"
```

```
}
```

- size：固件文件大小。
- md5：固件内容的MD5签名值，32位的hex String。
- url：固件的URL。时效为24小时。超过这个时间后，服务端拒绝下载。
- version：固件的目的版本号。

4. 设备收到URL之后，通过HTTPS协议根据URL下载固件。

说明:

设备需在固件URL下发后的24小时内下载固件，否则该URL失效。

下载固件过程中，设备端向服务端推送升级进度到Topic：`/ota/device/progress/${YourProductKey}/${YourDeviceName}`。消息格式如下：

```
{
  "id": 1,
  "params": {
    "step":"1",
    "desc":" xxxxxxxx "
  }
}
```

- id：消息ID号。
- step：
  - [1, 100] 下载进度比。
  - -1：代表升级失败。
  - -2：代表下载失败。
  - -3：代表校验失败。
  - -4：代表烧写失败。
- desc：当前步骤的描述信息。如果发生异常，可以用此字段承载错误信息。

5. 设备端完成固件升级后，推送最新的固件版本到Topic：`/ota/device/inform/${YourProductKey}/${YourDeviceName}`。如果上报的版本与OTA服务要求的版本一致就认为升级成功，反之失败。

说明:

升级成功的唯一判断标志是设备上报正确的版本号。即使升级进度上报为100%，如果不上报新固件版本号，也视为升级失败。

常见下载固件错误

- 签名错误。如果设备端获取的固件的URL不全或者手动修改了URL内容，就会出现如下错误：
- 拒绝访问。URL过期导致。目前，URL有效期为24小时。

3.6 远程配置

使用远程配置功能，可在不用重启设备或中断设备运行情况下，在线远程更新设备的系统参数、网络参数等配置信息。

前提条件

- 已在物联网平台控制台开通远程配置服务。如果未开通，登录物联网平台的控制台，选择监控运维 > 远程配置，然后单击开通服务。
- 设备端SDK已开启支持远程配置服务。需要在设备端SDK中定义 `FEATURE_SERVICE_OTA_ENABLED = y`。SDK提供接口`linkkit_cota_init`，用于初始化远程配置（Config Over The Air，COTA）。

远程配置说明

很多场景下，开发者需要更新设备的配置信息，包括设备的系统参数、网络参数、本地策略等。通常情况下，是通过#unique_149更新设备的配置信息。但是，这将加大固件版本的维护工作，并且需要设备中断运行以完成更新。为了解决上述问题，物联网平台提供远程配置更新功能，设备无需重启或中断运行即可在线完成配置信息更新。

物联网平台远程配置功能支持：

- 开启或关闭远程配置。
- 在线编辑配置文件，并管理版本。
- 从云端推送配置文件，批量更新设备配置信息。
- 设备主动请求更新配置信息。

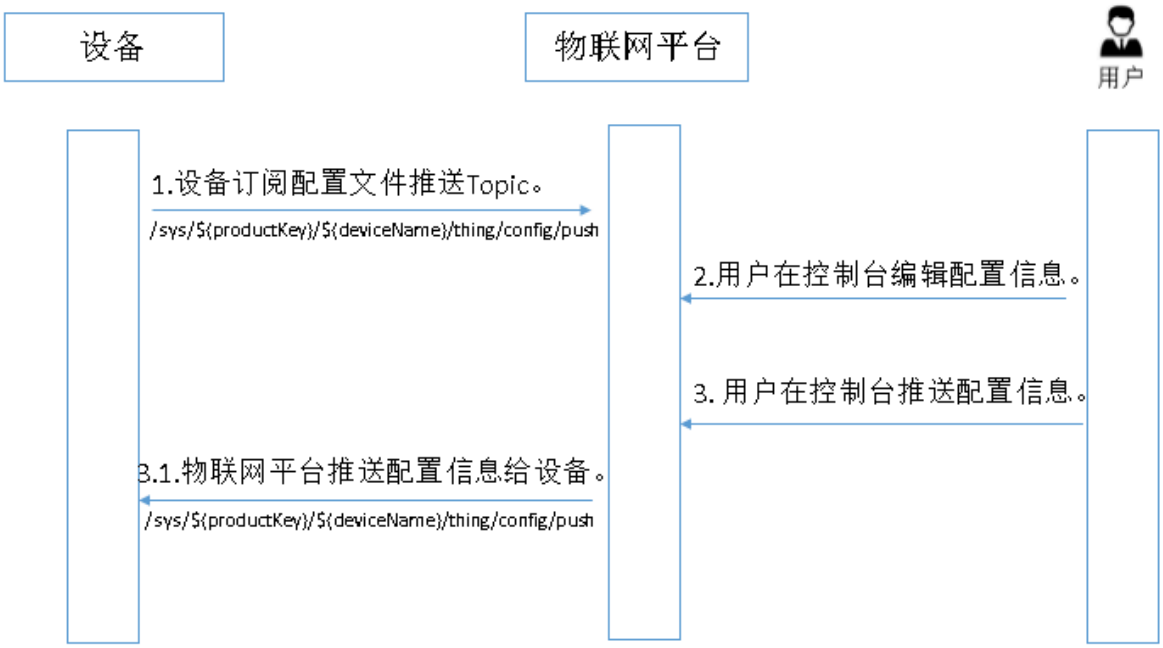
远程配置使用分为两种场景：

- 云端推送配置信息给设备端：您在物联网平台控制台批量推送配置信息给设备。设备接收后，修改本地配置文件。
- 设备主动请求配置信息：设备主动向云端请求新的配置文件，并进行更新。

以下两章节介绍这两种场景的操作流程。

场景一：云端推送配置信息给设备端

在物联网平台控制台，向某一产品下的所有设备批量推送配置文件。



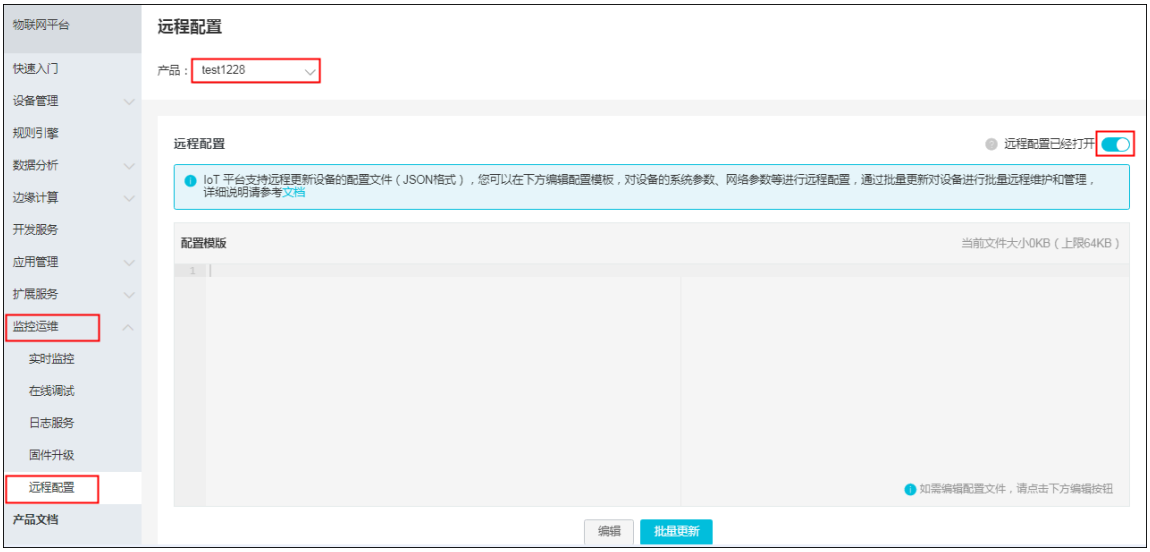
1. 设备上线。

> **说明:**
> 开发设备端时，已配置设备端订阅推送配置信息的Topic：/sys/${productKey}/${deviceName}/thing/config/push。

2. 在物联网平台控制台中，编辑配置文件。

a) 登录物联网平台的控制台，选择监控运维 > 远程控制。

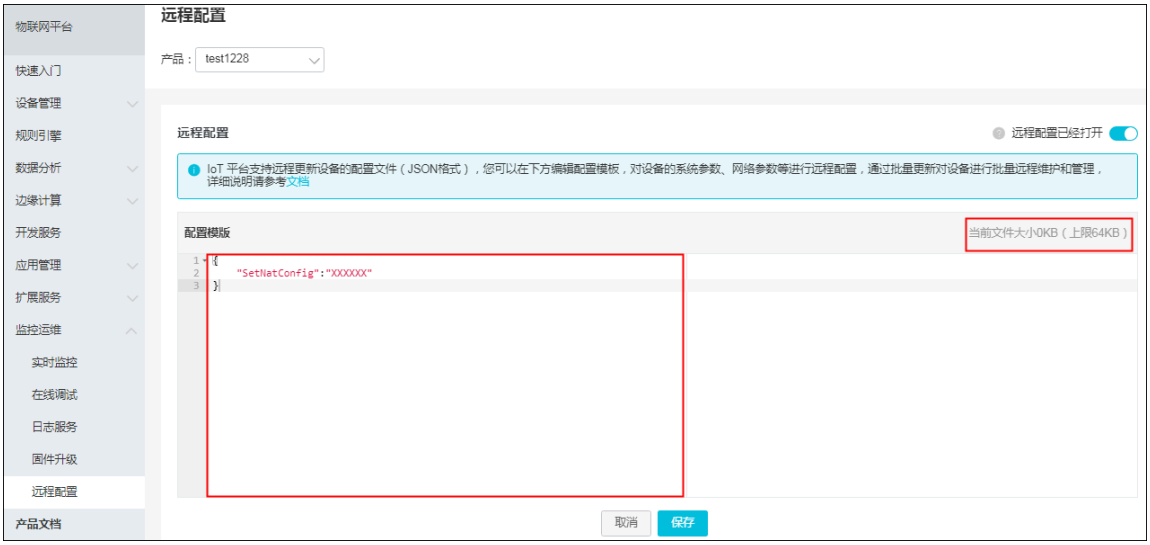
b) 选择产品，打开远程配置开关。



说明:

- 必须开启产品的远程配置功能后，才可以编辑配置信息。
- 切换为关闭状态，即关闭该产品的远程配置功能。
- 产品配置模板适用于该产品下的所有设备。目前，不支持在控制台向单个设备推送配置文件。

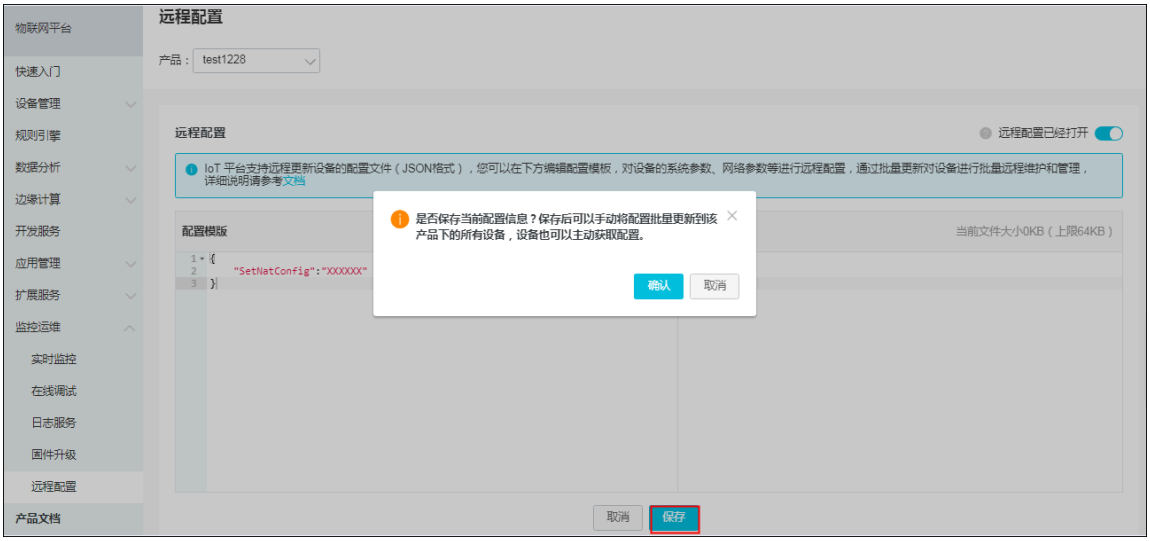
c) 单击编辑，然后在配置模板下的编辑区，编写或粘贴JSON格式的配置信息。



说明:

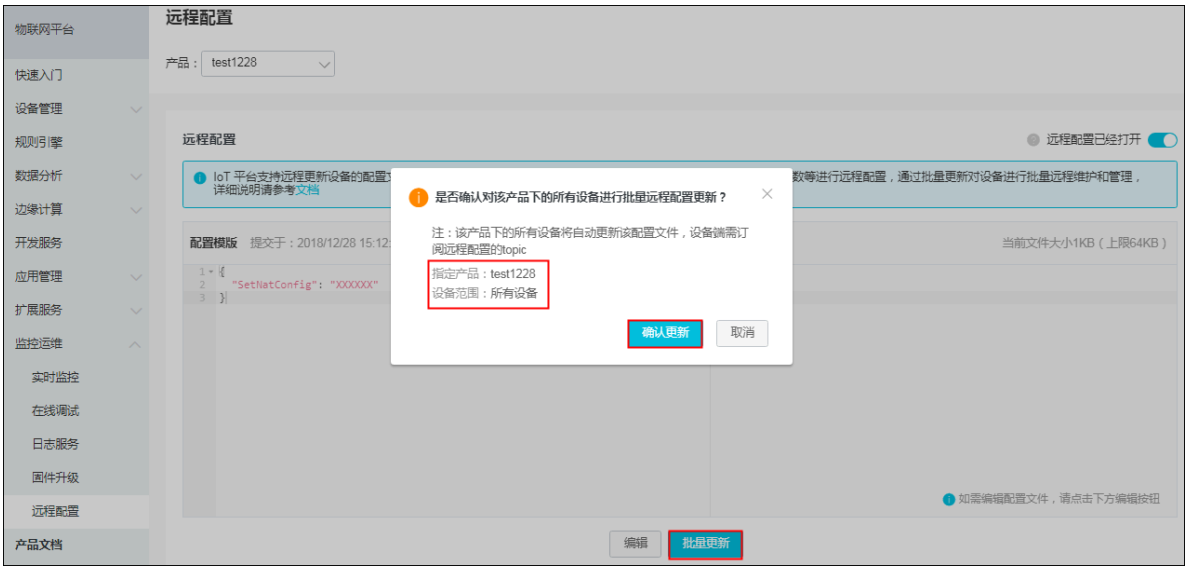
- 远程配置文件为JSON格式。物联网平台对配置内容没有特殊要求，但系统会对提交的配置文件进行JSON格式校验，避免错误格式引起配置异常。
- 配置文件最大支持64 KB。编辑框右上角将实时显示当前文件的大小。超过64KB的配置文件无法提交。

d) 编辑完成配置信息后，单击保存，将生成正式的配置文件。设备可主动请求更新该配置信息。



3. 单击批量更新，物联网平台会向该产品下的所有设备批量推送配置文件。

   您单击批量更新后，如果系统判断不是可信环境，会发起短信验证。您需完成短信验证后，系统才会向设备下发配置文件。



说明:

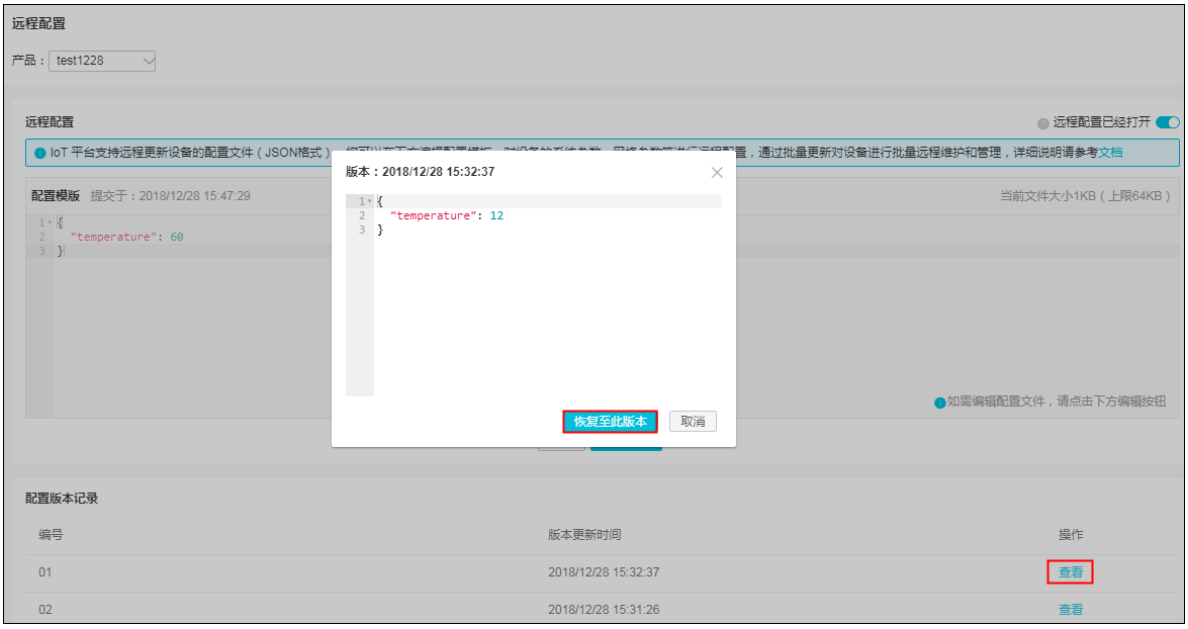
- 批量更新频率限制：一小时内仅允许操作一次 。

- 如果您希望停止批量更新，请关闭该产品的远程配置开关。关闭远程配置后，系统将停止所有更新推送，并且拒绝设备主动请求更新。

4. 设备端接收云端下发的配置信息后，自动更新配置。
5. （可选）查看和管理配置文件版本。

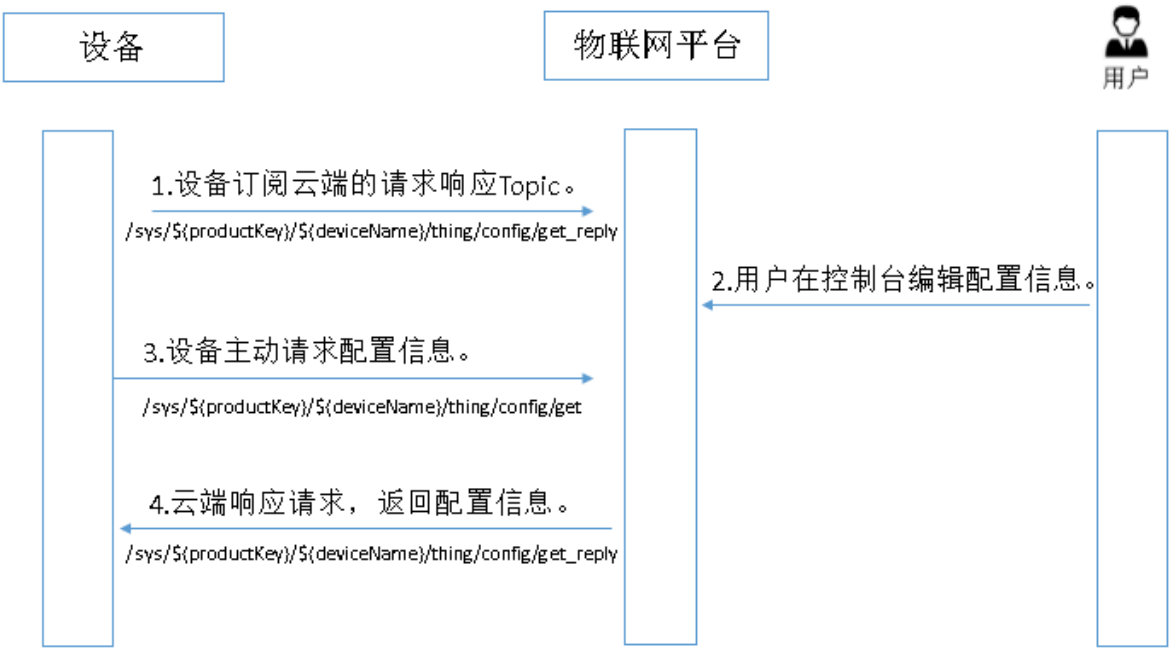
   远程配置默认保存最近5次的配置修改记录。重新编辑并提交配置文件后，上一次的配置版本信息将显示在版本记录列表中。您可以查看版本更新时间和配置内容，方便追溯。

   单击查看，将显示该版本的配置内容。再单击恢复至此版本，将所选版本的内容恢复至编辑区中，您可以编辑修改内容，然后批量更新。



场景二：设备主动请求配置信息

在一些场景中，设备需要主动查询配置信息。



1. 设备上线。

   > **说明:**
   > 开发设备端时，已配置设备端订阅云端响应设备请求配置信息的Topic：/sys/${productKey}/${deviceName}/thing/config/get_reply。

2. 在物联网平台控制台中，开启远程配置，编辑配置信息。详细步骤请参见以上场景一的开启远程配置操作描述。
3. 设备端使用接口`linkkit_invoke_cota_get_config`来触发远程配置请求。
4. 设备通过Topic `/sys/${productKey}/${deviceName}/thing/config/get`主动查询最新的配置信息。
5. 接收到设备的请求后，云端会返回最新的配置信息到Topic：/sys/${productKey}/${deviceName}/thing/config/get_reply。
6. 设备端在回调函数`cota_callback`中，处理远程配置下发的配置文件。

4 泛化协议

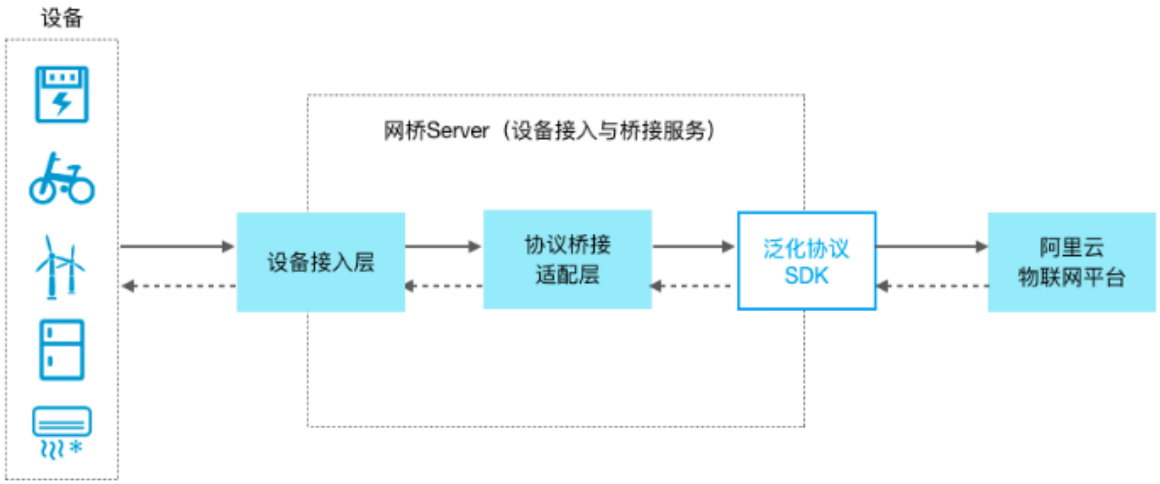
4.1 什么是泛化协议SDK

阿里云物联网平台支持基于MQTT、CoAP和HTTP协议的通信，其他类型协议，如消防协议GB/T 26875.3-2011、Modbus、JT808等暂未接入。在特定场景下，有些设备可能无法直接接入物联网平台。此时，您需要使用泛化协议SDK，快速构建桥接服务，搭建设备或平台与阿里云物联网平台的双向数据通道。

服务架构

泛化协议SDK是协议自适应的框架，用以构建与阿里云物联网平台进行高效双向通信的桥接服务。

服务架构如下图所示。



适用场景

泛化协议SDK面向的目标场景包括：

- 由于网络环境或者硬件限制，设备无法直接接入物联网平台。
- 设备只支持某种类型协议，而这种协议目前物联网平台不支持。
- 设备与您的服务器（Server）之间已有通信网络，您希望在不修改设备和协议的情况下，将设备接入物联网平台。
- 设备直接接入到您的服务器，且需要做一些其他的处理逻辑。

主要功能

泛化协议SDK使得网桥Server具备与物联网平台进行通信的能力。

基础功能：

- 提供基于配置文件的静态配置管理能力。
- 提供设备连接管理能力。
- 提供上行通信能力。
- 提供下行通信能力。

进阶功能如下：

- 提供基于接口的动态配置管理能力。
- 已封装属性、事件、标签数据上报接口供您调用。

名词解释

名词	描述
设备	您的真实物联网场景设备，该设备无法直接使用物联网平台所支持的协议直接与云端通信。
网桥Server	您的设备接入服务器。该服务器使用特定类型协议与设备通信，使用泛化协议SDK与物联网平台通信。
原始协议	设备与网桥Server之间使用的特定类型协议。泛化协议SDK不关心原始协议的具体定义和实现。
原始身份标识符	设备与网桥Server使用原始协议通信时的唯一标识符。泛化协议SDK接口参数中，用`originalIdentity`表示设备的原始身份标识符。
设备证书	在物联网平台注册设备后，获得的设备证书信息，包括ProductKey、DeviceName、DeviceSecret。使用泛化协议的场景下，不将设备证书烧录到设备上；而是配置泛化协议SDK文件*devices.conf*，由网桥将设备原始身份标识符`originalIdentity`映射到设备证书信息。
网桥证书	在物联网平台注册网桥设备后，获得的网桥设备证书信息，包括ProductKey、DeviceName、DeviceSecret，用于在云端标识网桥的身份。

开发和部署

1. 创建产品与设备。

   在物联网平台控制台，创建产品和设备。请参见创建产品和#unique_153或#unique_154。

   获取网桥设备证书信息。在泛化协议SDK配置时，需配置网桥设备证书信息。

   > 说明：
   > 网桥是个虚拟概念，您可以使用任意设备的证书信息作为网桥的证书信息。

2. 配置泛化协议SDK。

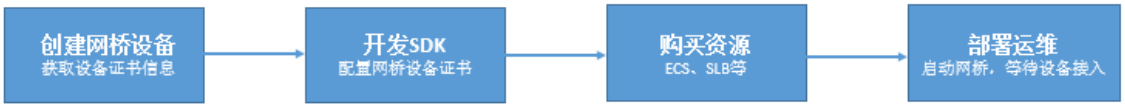
   目前，仅提供Java语言的泛化协议SDK，支持JDK 1.8及以上版本。

   泛化协议SDK配置细节，请参见#unique_155和#unique_156。

3. 部署服务。

   已开发完成的桥接服务，可以使用阿里云ECS和SLB等服务，以高度可扩展的方式部署至阿里云上；也可以直接部署到本地环境中，以保证可信通信环境。

   以基于阿里云云服务器ECS为例，上线流程如下。



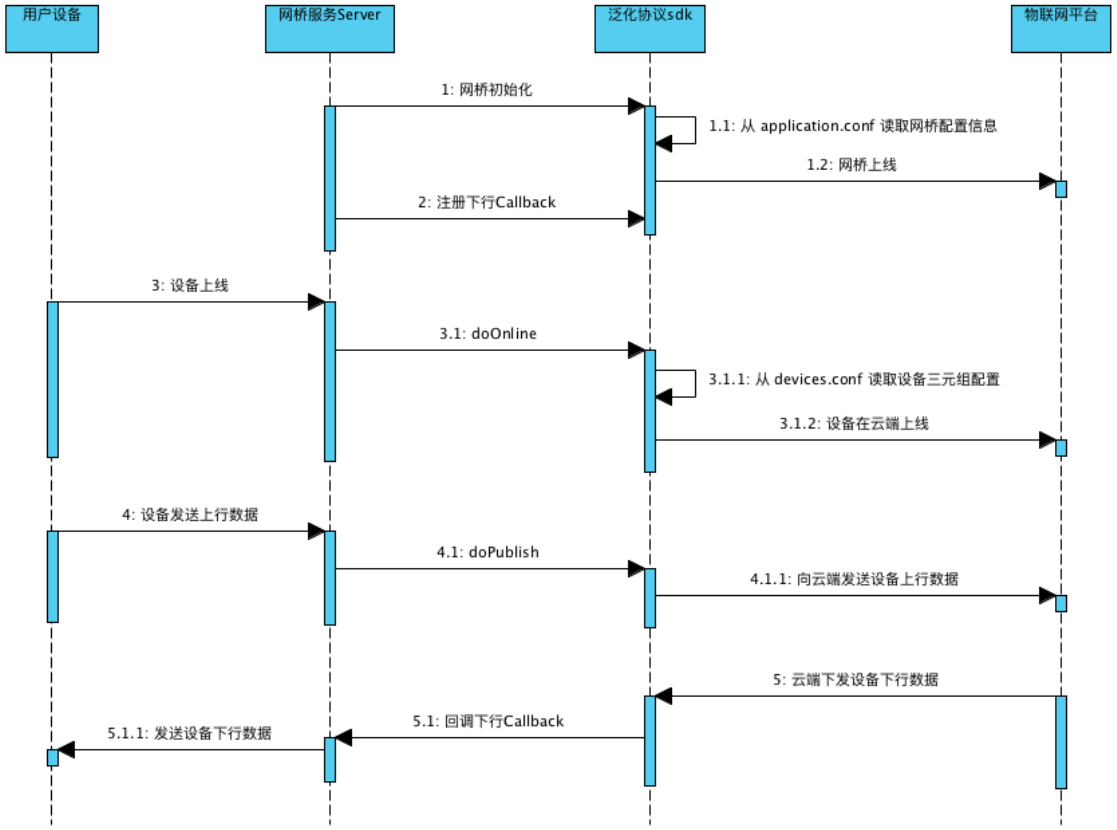
4.2 基础用法

基于泛化协议SDK，通过桥接服务，您的设备可以接入阿里云物联网平台，与物联网平台通信。本文介绍如何配置泛化协议SDK，实现设备上下线和消息上下行等基础能力。

物联网平台提供泛化协议SDK Demo，请访问泛化协议SDK Demo GitHub地址查看。

流程图

使用泛化协议SDK，桥接设备与物联网平台的整体流程图如下。



部署开发环境

部署Java SDK开发环境，并添加泛化协议SDK的项目Maven依赖。

```
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>iot-as-bridge-sdk-core</artifactId>
  <version>2.0.1</version>
</dependency>
```

初始化

- 初始化SDK。

  需要创建一个BridgeBootstrap对象实例，并调用bootstrap方法。泛化协议SDK初始化工作完成后，读取网桥信息，并向云端发起网桥设备上线请求等。

  此外，可以在调用bootstrap方法的同时，向泛化协议SDK注册一个DownlinkChannelHandler回调，用于接收云端下行消息。

  代码示例如下。

  ```
  BridgeBootstrap bridgeBootstrap = new BridgeBootstrap();
  bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
      @Override
      public boolean pushToDevice(Session session, String topic, byte
  [] payload) {
  ```

```
        //get message from cloud
        String content = new String(bytes);
        log.info("Get DownLink message, session:{}, {}, {}", session
, topic, content);
        return true;
    }

    @Override
    public boolean broadcast(String topic, byte[] payload) {
        return false;
    }
});
```

- 配置网桥信息。

  网桥配置默认使用配置文件方式。默认从Java工程默认资源文件路径（一般是*src/main/resources/*）下的*application.conf*中读取配置文件，格式支持HOCON（JSON超集）。泛化协议SDK使用typesafe.config解析配置文件。

  支持两种网桥配置方法：指定网桥设备和动态注册网桥设备。本文中仅提供指定网桥设备的配置示例；动态注册网桥设备的具体方法，请参见#unique_156/unique_156_Connect_42_section_xmx_dyi_nok。

  网桥配置参数说明如下表。

参数	是否必需	说明
productKey	是	网桥所属产品的ProductKey。
deviceName	否	网桥的DeviceName。 - 您预先注册了网桥设备，使用指定网桥设备证书信息进行配置，此参数为必需。 - 若您未预先注册网桥设备，而是根据服务器MAC地址作为设备名称，动态注册网桥设备，则不传入此参数。

参数	是否必需	说明
deviceSecret	否	网桥的DeviceSecret。 - 使用指定网桥设备证书信息进行配置，则需传入此参数。 - 若您未预先注册网桥设备，而是要动态注册网桥设备，则不传入此参数。

参数	是否必需	说明
http2Endpoint	是	HTTP2网关服务地址。网桥和云端通过HTTP2协议建立长连接通道。HTTP2网关服务地址的结构为：`${productKey}.iot-as-http2.${RegionId}.aliyuncs.com:443`。 变量*${productKey}*需替换成您的网桥所属产品的ProductKey。 变量*${RegionId}*需替换成您的服务所在地域代码。RegionId的表达方法，请参见#unique_160。 如，某用户的网桥设备的productKey为alabcabc12域为上海，HTTP2网关服务地址是`alabcabc123.iot-as-http2.cn-shanghai.aliyuncs.`

参数	是否必需	说明
authEndpoint	是	设备认证服务地址。 设备认证服务地址结构为：`https://iot-auth.${RegionId}.aliyuncs.com/auth/bridge`。 其中，变量*${RegionId}*需替换成您的服务所在地域代码。RegionId的表达方法，请参见#unique_160。 如，地域为上海，则认证服务地址就是`https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`。

参数	是否必需	说明
popClientProfile	否	根据MAC地址动态注册网桥设备时需传入的参数。 具体参数配置，请参见#unique_156/unique_156_Connect_42_s

指定网桥设备证书信息的配置格式如下。

```
# Server endpoint
http2Endpoint = "https://a1tN7OBmTcd.iot-as-http2.cn-shanghai.
aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/
bridge"

# Gateway device info, productKey & deviceName & deviceSecret
productKey = ${bridge-ProductKey-in-Iot-Plaform}
deviceName = ${bridge-DeviceName-in-Iot-Plaform}
deviceSecret = ${bridge-DeviceSecret-in-Iot-Plaform}
```

设备认证并上线

- 配置设备上线。

  泛化协议SDK中的设备上线接口设置如下。

```
/**
 * 设备认证
 * @param newSession 设备Session信息，下行回调的时候会将这个Session传递回
来。
 * @param originalIdentity 设备原始身份标识符。
 * @return
 */
public boolean doOnline(Session newSession, String originalIdentity
);
```

  设备上线时，需要传Session。下行消息回调时，会把Session回调给网桥。Session中包含设备的原始身份标识符字段，以便网桥判断消息属于哪个设备。

  此外，Session中还有一个可选的channel字段，设计上可以用来存放设备的连接信息。例如，您的网桥Server是基于Netty构建的，这里可以存放设备长连接对应的channel对象，消息下行的时候就可以直接从Session中获取channel进行操作。channel的数据类型是Object。

泛化协议SDK不会对channel数据做任何处理。您也可以根据使用场景，在channel中存放任何设备相关的信息。

设备上线代码示例如下。

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
//创建Session
Object channel = new Object();
Session session = Session.newInstance(originalIdentity, channel);
//设备上线
boolean success = uplinkHandler.doOnline(session, originalIdentity);
if (success) {
    // 设备上线成功，网桥接受后续设备通信请求。
} else {
    // 设备上线失败，网桥可以拒绝后续设备通信请求，如断开连接。
}
```

- 配置映射设备证书信息。

  配置设备原始身份标识符和设备证书信息的映射关系。默认使用配置文件方式，默认从Java工程的默认资源文件路径（一般是*src/main/resources/*）下的*devices.conf*中读取配置文件，格式支持HOCON（JSON超集）。泛化协议SDK使用typesafe.config解析配置文件。

  设备证书信息配置文件内容格式如下。

```
${device-originalIdentity} {
  productKey : ${device-ProductKey-in-Iot-Plaform}
  deviceName : ${device-DeviceName-in-Iot-Platform}
  deviceSecret : ${device-DeviceSceret-in-Iot-Platform}
}
```

参数	是否必需	说明
productKey	是	设备所属产品的ProductKey。
deviceName	是	设备的DeviceName。
deviceSecret	是	设备的DeviceSecret。

设备发送上行数据

泛化协议SDK的设备上报消息接口设置如下。

```
/**
 * 发送设备上行消息，同步调用接口。
 * @param originalIdentity 设备原始身份标识符。
 * @param protocolMsg 待发送消息，包含Topic、消息体、QoS等信息。
 * @param timeout 超时时间，单位秒。
 * @return 超时时间内是否发送成功。
 */
boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg
, int timeout);
/**
 * 发送设备上行消息，异步调用接口。
 * @param originalIdentity 设备原始身份标识符。
 * @param protocolMsg 待发送消息，包含Topic、消息体、QoS等信息。
```

```
 * @return 调用后立即返回CompletableFuture，调用者可进一步处理该future。
 */
CompletableFuture<ProtocolMessage> doPublishAsync(String originalId
entity,
                                                  ProtocolMessage
protocolMsg);
```

接口调用代码示例如下。

```
DeviceIdentity deviceIdentity =
    ConfigFactory.getDeviceConfigManager().getDeviceIdentity(
originalIdentity);
ProtocolMessage protocolMessage = new ProtocolMessage();
protocolMessage.setPayload("Hello world".getBytes());
protocolMessage.setQos(0);
protocolMessage.setTopic(String.format("/%s/%s/update",
        deviceIdentity.getProductKey(), deviceIdentity.getDeviceName
()));
//同步发送
int timeoutSeconds = 3;
boolean success = upLinkHandler.doPublish(originalIdentity, protocolMe
ssage, timeoutSeconds);
//异步发送
upLinkHandler.doPublishAsync(originalIdentity, protocolMessage);
```

网桥推送下行数据给设备

网桥在调用bootstrap方法时，向泛化协议SDK注册了DownlinkChannelHandler。当有下行消息的时候，泛化协议SDK就会回调DownlinkChannelHandler的pushToDevice方法。可以在pushToDevice中配置网桥处理下行消息。

 说明:

pushToDevice方法中不要做耗时逻辑，否则会阻塞下行消息接收的线程。如果有耗时或者IO逻辑，如收到云端下行消息后通过网络长连接发给子设备，请采用异步处理。

代码示例如下。

```
private static ExecutorService executorService  = new ThreadPool
Executor(
    Runtime.getRuntime().availableProcessors(),
    Runtime.getRuntime().availableProcessors() * 2,
    60, TimeUnit.SECONDS,
    new LinkedBlockingQueue<>(1000),
    new ThreadFactoryBuilder().setDaemon(true).setNameFormat("bridge-
downlink-handle-%d").build(),
    new ThreadPoolExecutor.AbortPolicy());
public static void main(String args[]) {
    //Use application.conf & devices.conf by default
    bridgeBootstrap = new BridgeBootstrap();
    bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
        @Override
        public boolean pushToDevice(Session session, String topic,
byte[] payload) {
            //get message from cloud
            //get downlink message from cloud
```

```
            executorService.submit(() -> handleDownLinkMessage(session
, topic, payload));
            return true;
        }
        @Override
        public boolean broadcast(String s, byte[] bytes) {
            return false;
        }
    });
}
private static void handleDownLinkMessage(Session session, String
topic, byte[] payload) {
    String content = new String(payload);
    log.info("Get DownLink message, session:{}, topic:{}, content:
{}", session, topic, content);
    Object channel = session.getChannel();
    String originalIdentity = session.getOriginalIdentity();
    //for example, you can send the message to device via channel, it
depends on you specific server implementation
}
```

参数	说明
Session	Session是设备在doOnline的时候传递进来的，可以用来区分下行消息是发给哪个设备的。
topic	下行消息的Topic。
payload	二进制格式的下行消息的消息体数据。

设备下线

设备下线分为两种情况：

- 网桥与云端之间的连接断开时，则所有的设备也会自动从云端离线。
- 网桥主动向云端上报某个设备下线的消息。

  网桥上报设备下线的接口定义如下。

```
/**
 * 向云端上报某个设备下线。
 * @param originalIdentity 设备原始标识符。
 * @return 是否成功上报。
 */
```

```
boolean doOffline(String originalIdentity);
```

调用下线接口代码示例如下。

```
upLinkHandler.doOffline(originalIdentity);
```

4.3 进阶用法

本文介绍泛化协议SDK的一些进阶能力的用法，包括自定义配置文件路径、配置动态创建网桥设备、调用泛化协议SDK中封装的数据上报接口上报属性、事件和标签。

自定义配置管理

默认情况下，网桥的配置文件和设备证书的映射关系配置文件，都是从固定路径的固定文件名（分别是*application.conf*和*devices.conf*）中读取的。泛化协议SDK提供了自定义配置管理的能力，您只需要在调用bootstrap方法之前，先调用*ConfigFactory.init*方法，自定义配置文件的路径，也可以自定义实例实现对应的接口。

自定义配置代码示例：

```
//Define config
//You can specify the location path of config files
//or you can create an instance and implement the corresponding
interface
//Config.init() must be called before bridgeBootstrap.bootstrap()
ConfigFactory.init(
    ConfigFactory.getBridgeConfigManager("application-self-define.conf
"),
    selfDefineDeviceConfigManager);
bridgeBootstrap.bootstrap();

private static DeviceConfigManager selfDefineDeviceConfigManager = new
 DeviceConfigManager() {
    @Override
    public DeviceIdentity getDeviceIdentity(String originalIdentity) {
        //Suppose you dynamically get deviceInfo in other ways
        return devicesMap.get(originalIdentity);
    }

    @Override
    public String getOriginalIdentity(String productKey, String
deviceName) {
        //you can ignore this
        return null;
    }
};
```

动态创建网桥设备

当您需要在大量的服务器上部署网桥应用，如果为每一个网桥服务器指定不同的网桥设备信息会比较繁琐。您可以配置网桥信息文件*application.conf*动态创建网桥设备。您需在配置

文件中，传入参数productKey和popClientProfile，泛化协议SDK将调用物联网平台开放API，以服务器MAC地址作为设备名称，新创建一个网桥设备。

说明:

- 采用动态创建网桥设备方法，仅需要修改网桥配置文件，调用代码与#unique_162一致。
- 网桥配置文件中，如果已经指定了网桥设备信息，不会再动态创建设备。仅当配置文件中deviceName和deviceSecret配置为空，且popClientprofile所有配置齐全的情况下，泛化协议SDK才会尝试调用物联网平台API，以服务器MAC地址作为设备名称动态创建设备。如果当前MAC地址已经创建过设备，则会直接使用这个设备作为网桥设备。
- 如果采用这种方式配置网桥，不建议您使用生产环境的配置直接在本地机器上调试。因为在多个本地PC上调试程序，每次都会将当前机器的MAC地址注册为网桥，并将设备信息配置文件*devices.conf*中的所有设备与该网桥关联。建议在调试阶段使用专门用于测试的设备，以免干扰生产环境。

表 4-1: 配置参数说明

参数	是否必需	说明
productKey	是	网桥所属产品的 ProductKey。
http2Endpoint	是	HTTP2网关服务地址。 网桥和云端通过HTTP2协议建立长连接通道。 HTTP2网关服务地址的结构为：`${productKey}.iot-as-http2.${RegionId}.aliyuncs.com:443`。 变量*${productKey}*需替换成您的网桥所属产品的ProductKey。 变量*${RegionId}*需替换成您的服务所在地域代码。RegionId 的表达方法，请参见地域和可用区地域和可用区文档。 如，某网桥设备的productKey为alabcabc123，地域为上海，则HTTP2网关服务地址是 `alabcabc123.iot-as-http2.cn-shanghai.aliyuncs.com:443`。

参数	是否必需	说明
authEndpoint	是	设备认证服务地址。 设备认证服务地址结构为：`https://iot-auth.${RegionId}.aliyuncs.com/auth/bridge`。 其中，变量*${RegionId}*需替换成您的服务所在地域代码。RegionId 的表达方法，请参见地域和可用区地域和可用区文档。 如，地域为上海，则认证服务地址就是 `https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`。
popClientProfile	是	配置此参数，泛化协议SDK将调用阿里云端开放接口自动创建一个网桥设备。 具体参数配置见下表popClientProfile。

表 4-2: popClientProfile

参数	是否必需	描述
accessKey	是	您的阿里云账号的AccessKey ID。 请登录阿里云控制台，单击您的账号头像，进入AccessKey管理中，创建或查看AccessKey。
accessSecret	是	您的阿里云账号的AccessKey Secret。
name	是	将要创建网桥设备的物联网平台服务地域，即`productKey`值所代表的产品所属的地域。 地域的表达方法，请参见地域和可用区地域和可用区文档。
region	是	将要创建网桥设备的物联网平台服务地域ID，即`productKey`值所代表的产品所属的地域。 表达方法同`name`。
product	是	产品名称，请配置为 `Iot`。

参数	是否必需	描述
endpoint	是	调用指定地域API的节点地址。 节点地址结构为：`iot.${RegionId}.aliyuncs.com`。 其中，变量*${RegionId}*需替换成您的服务所在地域代码。RegionId 的表达方法，请参见地域和可用区地域和可用区文档。 如上海节点的 endpoint 为：`iot.cn-shanghai.aliyuncs.com`。

动态创建网桥设备配置示例：

```
# Server endpoint
http2Endpoint = "https://${YourProductKey}.iot-as-http2.cn-shanghai.
aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge"

# Gateway device info
# You can also specify productKey only, and dynamic register
deviceName & deviceSecret in runtime
productKey = ${YourProductKey}

# If you dynamic register gateway device using your mac address, you
have to specify 'popClientProfile'
# otherwise you can ignore it
popClientProfile = {
    accessKey = ${YourAliyunAccessKey}
    accessSecret = ${YourAliyunAccessSecret}
    name = cn-shanghai
    region = cn-shanghai
    product = Iot
    endpoint = iot.cn-shanghai.aliyuncs.com
}
```

调用物模型数据上报接口

为了方便使用，减少您的封装操作，泛化协议SDK中已封装部分数据上报接口，包括属性上报接口`reportProperty`、事件上报接口`fireEvent`和更新设备标签接口`updateDeviceTag`。设备可以通过这些接口向物联网平台上报相应消息。

接口使用前提和说明：

- 调用`reportProperty`和`fireEvent`上报属性值和事件前，您需先在物联网平台控制台设备所属产品的产品详情页的功能定义页签下，定义属性和事件。请参见#unique_36。
- 调用`updateDeviceTag`接口上报的设备标签，如果您已经在物联网平台控制台设备对应的设备详情页，添加了该标签，则更新标签值（value）；若没有对应的标签，则新建标签。

接口调用示例：

```
TslUplinkHandler tslUplinkHandler = new TslUplinkHandler();
//report property
//Property 'testProp' is defined in IoT Platform Web Console
String requestId = String.valueOf(random.nextInt(1000));
tslUplinkHandler.reportProperty(requestId, originalIdentity, "testProp
", random.nextInt(100));

//fire event
//Event 'testEvent' is defined in IoT Platform Web Console
requestId = String.valueOf(random.nextInt(1000));
HashMap<String, Object> params = new HashMap<String, Object>();
params.put("testEventParam", 123);
tslUplinkHandler.fireEvent(originalIdentity, "testEvent", ThingEvent
Types.INFO, params);

//update device tag
//'testDeviceTag' is a tag key defined in IoT Platform Web Console
requestId = String.valueOf(random.nextInt(1000));
tslUplinkHandler.updateDeviceTag(requestId, originalIdentity, "
testDeviceTag", String.valueOf(random.nextInt(1000)));
```

以上示例中，接口调用的参数说明：

参数	说明
requestId	请求消息ID。
originalIdentity	设备的原始身份标识符。
testProp	属性的identifier。本示例的前提条件是：在物联网平台控制台上为产品定义功能时，定义了一个identifier为testProp属性。本示例代码为上报属性testProp的值。
random.nextInt(100)	上报的属性值。属性值的取值范围也在定义属性时定义。在本示例中，使用`random.nextInt(100)`表示取小于100的整形随机值。
testEvent	事件的identifier。本示例的前提条件是：在物联网平台控制台为产品定义功能时，定义了一个identifier为testEvent的事件。本示例代码为上报事件testEvent。
ThingEventTypes.INFO	事件类型。`ThingEventTypes`参数表示事件类型，`INFO`表示事件类型取值为INFO（信息）。 本示例的前提条件是：在物联网平台控制台上定义事件testEvent时，选择的事件类型为信息（即INFO）。 如果事件类型定义为故障，则该参数为`ThingEventTypes.ERROR`
params	事件的输出参数。事件输出参数的identifier、数据类型、取值范围等也在定义事件时定义。本示例中，上报事件的出参identifier是testEventParam，参数值是123。

参数	说明
testDeviceTag	设备标签键（key），String类型。本示例中为testDeviceTag。 实际使用时，请根据设备标签键规范和您的需求设置。请参见#unique_163/unique_163_Connect_42_section_igy_bvb_wdb文档。
String.valueOf(random.nextInt(1000))	设备标签值（value），String类型。本示例中，用`String.valueOf(random.nextInt(1000))`表示取值为一个小于1000的随机值。实际使用时，请根据设备标签值规范和您的需求设置。 请参见#unique_163/unique_163_Connect_42_section_igy_bvb_wdb文档。

5 RRPC

5.1 什么是RRPC

MQTT协议是基于PUB/SUB的异步通信模式，不适用于服务端同步控制设备端返回结果的场景。物联网平台基于MQTT协议制定了一套请求和响应的同步机制，无需改动MQTT协议即可实现同步通信。物联网平台提供API给服务端，设备端只需要按照固定的格式回复PUB消息，服务端使用API，即可同步获取设备端的响应结果。

名词解释

- RRPC：Revert-RPC。RPC（Remote Procedure Call）采用客户机/服务器模式，用户不需要了解底层技术协议，即可远程请求服务。RRPC则可以实现由服务端请求设备端并能够使设备端响应的功能。
- RRPC 请求消息：云端下发给设备端的消息。
- RRPC 响应消息：设备端回复给云端的消息。
- RRPC 消息ID：云端为每次RRPC调用生成的唯一消息ID。
- RRPC 订阅Topic：设备端订阅RRPC消息时传递的Topic，含有通配符。

RRPC原理

1. 物联网平台收到来自用户服务器的RRPC调用，下发一条RRPC请求消息给设备。消息体为用户传入的数据，Topic为物联网平台定义的Topic，其中含有唯一的RRPC消息ID。
2. 设备收到下行消息后，按照指定Topic格式（包含之前云端下发的唯一的RRPC消息ID）回复一条RRPC响应消息给云端，云端提取出Topic中的消息ID，和之前的RRPC请求消息匹配上，然后回复给用户服务器。
3. 如果调用时设备不在线，云端会给用户服务器返回设备离线的错误；如果设备没有在超时时间内（8秒内）回复RRPC响应消息，云端会给用户服务器返回超时错误。

Topic格式

不同Topic格式使用方法不同。

- 系统Topic使用方法参见#unique_166。
- 自定义Topic使用方法参见#unique_167。

RRPC调用实践示例，请参见#unique_168。

5.2 调用系统Topic

RRPC支持调用系统Topic与云端通信。本文介绍RRPC通信的系统Topic和接入方法。

系统Topic

RRPC调用的系统Topic格式如下：

- RRPC请求消息Topic：/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/${messageId}
- RRPC响应消息Topic：/sys/${YourProductKey}/${YourDeviceName}/rrpc/response/${messageId}
- RRPC订阅Topic：/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/+

其中，*${YourProductKey}*是您的设备所属产品的ProductKey，*${YourDeviceName}*是您的设备的名称，*${messageId}*是云端生成的唯一的RRPC消息ID。

RRPC接入

- 云端发送RRPC消息。

  服务端调用云端API RRpc接口向设备发送消息。接口调用方法，请参见RRpc。

  以使用Java SDK为例，调用方式：

```
RRpcRequest request = new RRpcRequest();
request.setProductKey("testProductKey");
request.setDeviceName("testDeviceName");
request.setRequestBase64Byte(Base64.getEncoder().encodeToString("
hello world"));
request.setTimeout(3000);
RRpcResponse response = client.getAcsResponse(request);
```

- 设备端返回RRPC响应的Topic。

  设备端收到RRPC请求之后，需要根据RRPC请求Topic的格式，返回响应消息到对应的响应Topic。

  设备端从收到消息的Topic（/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/${messageId}）中提取出`messageId`，然后拼装出对应的RRPC响应Topic，发送响应给云端。

  说明:

  目前，仅支持设备端返回QoS=0的RRPC响应消息。

5.3 调用自定义Topic

RRPC支持调用自定义Topic与云端通信，且相关Topic中包含了完整的您自定义的Topic。

自定义Topic

RRPC调用自定义Topic的格式如下：

- RRPC请求消息Topic：/ext/rrpc/${messageId}/${topic}
- RRPC响应消息Topic：/ext/rrpc/${messageId}/${topic}
- RRPC订阅Topic：/ext/rrpc/+/${topic}

其中*${messageId}*是云端生成的唯一的RRPC消息ID，*${topic}*是您的自定义Topic。

RRPC接入

1. 云端发送RRPC消息。

   服务端调用云端API RRpc接口向设备发送消息。接口调用方法，请参见RRpc。

   以使用Java SDK为例，调用方式：

```
RRpcRequest request = new RRpcRequest();
request.setProductKey("testProductKey");
request.setDeviceName("testDeviceName");
request.setRequestBase64Byte(Base64.getEncoder().encodeToString("
hello world"));
request.setTopic("/testProductKey/testDeviceName/user/get");//如果是
自定义Topic调用方式，在这里传递自定义Topic
request.setTimeout(3000);
RRpcResponse response = client.getAcsResponse(request);
```

   使用自定义Topic格式时，您需要确保您的云端Java SDK（aliyun-java-sdk-iot）版本为6.0.0及以上版本。

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-iot</artifactId>
    <version>6.0.0</version>
```

```
</dependency>
```

2. 设备端接入。

   从云端下发自定义格式Topic的RRPC调用命令到设备端时，设备端必须在进行MQTT CONNECT协议设置时，在clientId中增加ext=1参数。设备端通过MQTT协议接入物联网平台操作指导，请参见MQTT-TCP连接通信。

   例如，原来传递的clientId为：

   ```
   mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=
   132323232|"
   ```

   则添加ext=1参数后，传递的clientId为：

   ```
   mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=
   132323232,ext=1|"
   ```

   > 说明：
   > 云端和设备端之间使用自定义Topic进行RRPC通信的条件：
   >
   > - 云端传递的Topic字段不为空。
   > - 设备端在建立连接（connect）时传递了ext=1参数。

3. 设备端返回RRPC响应的Topic。

   RRPC请求Topic和响应Topic格式一样，直接将请求Topic作为响应Topic即可。

   > 说明：
   > 目前，仅支持设备端返回QoS=0的RRPC响应消息。

6 设备影子

6.1 设备影子概览

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

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

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

应用场景

- 场景1：网络不稳定，设备频繁上下线。

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

  使用设备影子机制存储设备最新状态，一旦设备状态产生变化，设备会将状态同步到设备影子。应用程序在请求设备当前状态时，只需要获取影子中的状态即可，不需要关心设备是否在线。
- 场景2：多程序同时请求获取设备状态。

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

  使用设备影子机制，设备只需要主动同步状态给设备影子一次，多个应用程序请求设备影子获取设备状态，即可获取设备最新状态，做到应用程序和设备的解耦。
- 场景3：设备掉线。
  - 设备网络不稳定，导致设备频繁上下线，应用程序发送控制指令给设备时，设备掉线，指令无法下达到设备。
    - 通过QoS=1或者2实现，但是该方法对于服务端的压力比较大，一般不建议使用。
    - 使用设备影子机制，应用程序发送控制指令，指令携带时间戳保存在设备影子中。当设备掉线重连时，获取指令并根据时间戳确定是否执行。
  - 设备真实掉线，指令发送失败。设备再上线时，设备影子功能通过指令加时间戳的模式，保证设备不会执行过期指令。

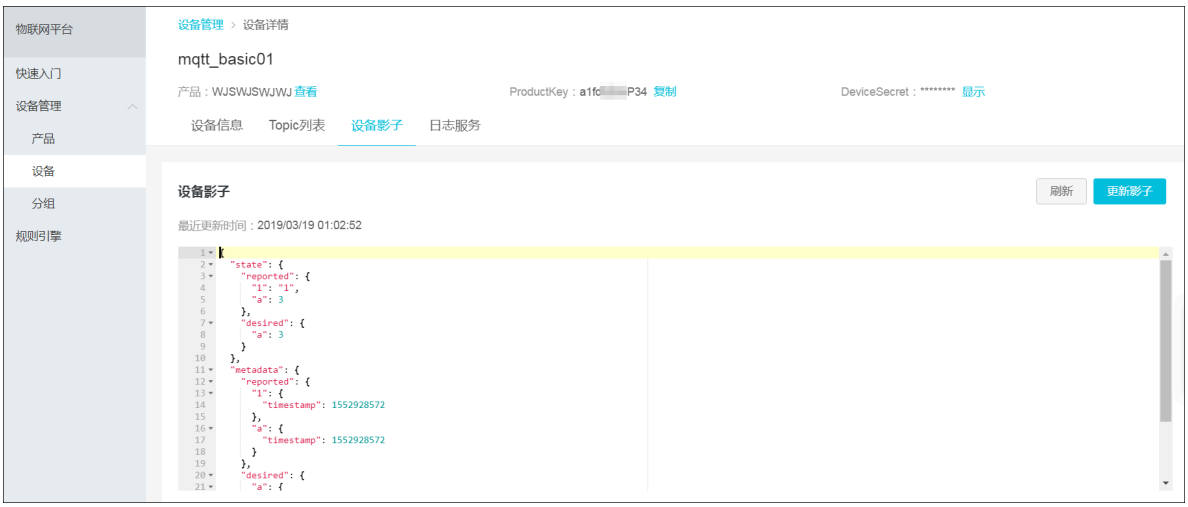
查看与更新设备影子

您可以在控制台，查看设备影子信息，更新设备影子状态。

操作步骤：

1. 登录物联网平台控制台。
2. 单击设备管理 > 设备。
3. 单击对应设备的查看按钮，进入设备详情页。
4. 单击设备影子。

   页面显示设备上报的影子状态。



5. 单击更新影子，在“desired”部分，填入期望设备状态。

   设备影子文档格式，请参见设备影子JSON详解。

   设备在线时，设备影子保存期望状态，设备通过订阅Topic直接获得期望状态。

   设备离线时，设备影子缓存期望状态，设备上线后，主动从云端拉取最新期望状态。

相关API

获取设备影子：#unique_176

更新设备影子：#unique_177

6.2 设备影子JSON详解

本文档介绍设备影子的JSON格式表达方法。

设备影子JSON文档示例：

```
{
    "state": {
        "desired": {
            "color": "RED",
            "sequence": [
                "RED",
                "GREEN",
                "BLUE"
```

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

JSON属性描述，如下表表 6-1: JSON属性说明所示。

表 6-1: JSON属性说明

属性	描述
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。

属性	描述
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"] }
}
```

6.3 设备影子数据流

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

设备影子Topic

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

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

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

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

  设备影子更新状态到该Topic，设备订阅此Topic获取最新消息。

使用示例

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

示例中，产品的ProductKey是a1PbxxxxWfX；设备名称DeviceName是lightbulb。设备以QoS=1发布消息和订阅两个设备影子Topic。

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

一、设备主动上报状态

设备在线时，主动上报设备状态到影子，应用程序主动获取设备影子状态。

流程图：

1. 当灯泡lightbulb联网时，使用Topic `/shadow/update/a1PbRCFQWfX/lightbulb`上报最新状态到影子。

   发送的JSON消息格式：

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

```
}
```

表 6-2: 上报参数说明

参数	说明
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。 当version为-1时，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. 当设备影子接收到灯泡上报状态时，成功更新影子文档。

```
{
  "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/a1PbRCFQWfX/lightbulb`中。

   · 若更新成功，发送到该Topic中的消息为：

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

```
}
```

- 若更新失败，发送到该Topic中的消息为：

```
{
  "method": "reply",
  "payload": {
    "status": "error",
    "content": {
      "errorcode": "${errorcode}",
      "errormessage": "${errormessage}"
    }
  },
  "timestamp": 1469564576
}
```

表 6-3: 错误码说明

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

二、应用程序改变设备状态

应用程序下发期望状态给设备影子，设备影子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。

流程图：

1. 应用程序发消息到Topic `/shadow/update/a1PbRCFQWfX/lightbulb/`中，要求更改灯泡状态。

```
{
  "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/a1PbRCFQWfX/lightbulb`中。返回结果信息构成由设备影子决定。

```
{
  "method": "control",
  "payload": {
    "status": "success",
```

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

4. 因为设备灯泡在线，并且订阅了Topic`/shadow/get/a1PbRCFQWfX/lightbulb`，所以会收到消息。

   收到消息后，根据请求文档中`desired`的值，将灯泡颜色变成绿色。

   灯泡更新完状态后，上报最新状态到物联网平台。

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

说明:

如果有时间戳判断指令过期，也可以选择不更新。

5. 最新状态上报成功后，

   - 设备端发消息到Topic`/shadow/update/a1PbRCFQWfX/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，此时的影子文档如下：

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

三、设备主动获取影子内容

若应用程序发送指令时，设备离线。设备再次上线后，将主动获取设备影子内容。

流程图

1. 灯泡主动发送以下消息到Topic`/shadow/update/a1PbRCFQWfX/lightbulb`中，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时，发送最新状态到Topic`/shadow/get/a1PbRCFQWfX/lightbulb`。灯泡通过订阅该Topic获取最新状态。消息内容如下：

```
{
  "method": "reply",
  "payload": {
    "status": "success",
    "state": {
      "reported": {
```

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

四、设备主动删除影子属性

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

流程图

发送以下内容到Topic`/shadow/update/a1PbRCFQWfX/lightbulb`**中。**

其中，`method`**为**`delete`**，属性的值为**`null`**。**

- **删除影子中某一属性：**

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

```
}
```

- **删除影子全部属性：**

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

7 NTP服务

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

原理介绍

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

> 说明:
> 只有设备端与云端成功建立连接之后，才能通过NTP服务进行校准。
>
> 举个例子，嵌入式设备上电后没有准确时间，TLS建连过程中证书时间校验失败的问题，无法通过NTP服务解决，因为此时设备与云端尚未成功建立连接。

接入流程

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

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

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

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

```
{
    "deviceSendTime":"100"
}
```

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

3. 设备端收到服务端回复的消息，payload中包含以下信息：

```
{
```

```
    "deviceSendTime":"100",
    "serverRecvTime":"1010",
    "serverSendTime":"1015",
}
```

4. 设备端计算出当前精确的unix时间。

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

使用示例

说明:

本示例为了使数值差异一目了然，使用的数值不是实际的时间戳。

实际时间戳数据格式可以为Long或String类型，且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100，服务端时间是1000，链路延时是10，服务端从接收到发送经过了5。

-	设备端时间	服务端时间
设备发送	100（deviceSendTime）	1000
服务端接收	110	1010（serverRecvTime）
服务端发送	115	1015（serverSendTime）
设备端接收	125（deviceRecvTime）	1025

则设备端计算出的当前准确时间为（1010 + 1015 + 125 - 100）/ 2 = 1025。

与云端当前时间相同对比下，如果直接采用云端返回的时间戳，只能得到1015，端上的时间会有一个链路延时的误差。

8 账号与登录

本章将详细介绍IoT控制台账号、登录的操作。

8.1 使用阿里云主账号登录控制台

阿里云主账号具有该账号下所有资源的完全操作权限，并且可以修改账号信息。

使用主账号登录 IoT 控制台

使用阿里云主账号登录物联网控制台，建议您首先完成实名认证，以获取对物联网平台所有操作的完全权限。

1. 访问阿里云官网阿里云官网。
2. 单击控制台。
3. 使用阿里云账号和密码登录。

> 说明:
> 若忘记账号或密码，请单击登录框中忘记会员名或忘记密码，进入账号或密码找回流程。

4. 在控制台中，单击产品与服务，页面显示所有阿里云产品和服务名称。
5. 搜索物联网平台，并单击搜索结果中的物联网平台产品名，进入物联网控制台。

> 说明:
> 如果您还没有开通物联网平台服务，物联网控制台主页会展示相关提示，您只需单击立即开通便可快速开通物联网平台服务。

进入物联网控制台后，您便可以进行产品管理、设备管理、规则管理等操作。

使用主账号创建访问控制

因为主账号具有账号的完全权限，主账号泄露会带来极严重的安全隐患。因此，若需要授权其他人访问您的阿里云资源，请勿将您的阿里云账号及密码直接泄露出去。应该通过访问控制 RAM 创建子账号，并给子账号授予需要的访问权限。非账号所有者或管理员的其他人通过子账号访问资源。有关子账号访问的具体方法，请参见#unique_183和#unique_184。

8.2 RAM授权管理

本章节将详细介绍物联网平台账号权限控制相关事宜。

8.2.1 RAM 和 STS 介绍

RAM 和 STS 是阿里云提供的权限管理系统。

了解 RAM 和 STS 的详情，请参见访问控制产品帮助文档访问控制产品文档。

RAM 的主要作用是控制账号系统的权限。通过使用 RAM，创建、管理子账号，并通过给子账号授予不同的权限，控制子账号对资源的操作权限。

STS 是一个安全凭证（Token）的管理系统，为阿里云子账号（RAM 用户）提供短期访问权限管理。通过 STS 来完成对临时用户的访问授权。

背景介绍

RAM 和 STS 解决的一个核心问题是如何在不暴露主账号的 AccessKey 的情况下，安全地授权他人访问。因为一旦主账号的 AccessKey 被泄露，会带来极大的安全风险：获得该账号 AccessKey 的人可任意操作该账号下所有的资源，盗取重要信息等。

RAM 提供的是一种长期有效的权限控制机制。通过创建子账号，并授予子账号相应的权限，将不同的权限分给不同的用户。子账号的 AccessKey 也不能泄露。即使子账号泄露也不会造成全局的信息泄露。一般情况下，子账号长期有效。

相对于 RAM 提供的长效控制机制，STS 提供的是一种临时访问授权。通过调用 STS，获得临时的 AccessKey 和 Token。可以将临时 AccessKey 和 Token 发给临时用户，用来访问相应的资源。从 STS 获取的权限会受到更加严格的限制，并且具有时间限制。因此，即使出现信息泄露的情况，影响相对较小。

使用场景示例，请参见使用示例。

基本概念

使用 RAM 和 STS 涉及以下基本概念：

- 子账号：在 RAM 控制台中，创建的用户，每个用户即一个子账号。创建时或创建成功后，均可为子账号生成独立的 AccessKey。创建后，需为子账号配置密码和权限。使用子账号，可以进行已获授权的操作。子账号可以理解为具有某种权限的用户，可以被认为是一个具有某些权限的操作发起者。
- 角色（Role）：表示某种操作权限的虚拟概念，但是没有独立的登录密码和 AccessKey。子账号可以扮演角色。扮演角色时，子账号拥有的权限是该角色的权限。
- 授权策略（Policy）：用来定义权限的规则，如允许子账号用户读取或者写入某些资源。
- 资源（Resource）：代表子账号用户可访问的云资源，如表格存储所有的 Instance、某个 Instance 或者某个 Instance 下面的某个 Table 等。

子账号和角色可以类比为个人和其身份的关系。如，某人在公司的角色是员工，在家里的角色是父亲。同一人在不同的场景扮演不同的角色。在扮演不同角色的时候，拥有对应角色的权限。角色本身并不是一个操作的实体，只有用户扮演了该角色之后才是一个完整的操作实体。并且，一个角色可以被多个不同的用户同时扮演。

使用示例

为避免阿里云账号的 AccessKey 泄露而导致安全风险，某阿里云账号管理员使用 RAM 创建了两个子账号，分别命名为 A 和 B，并为 A 和 B 生成独立的 AccessKey。A 拥有读权限，B 拥有写权限。 管理员可以随时在 RAM 控制台取消子账号用户的权限。

现在因为某些原因，需要授权给其他人临时访问物联网平台接口的权限。这种情况下，不能直接把 A 的 AccessKey 透露出去，而应该新建一个角色 C，并给这个角色授予读取物联网平台接口的权限。但请注意，目前角色 C 还无法直接使用。因为并不存在对应角色 C 的 AccessKey，角色 C 仅是一个拥有访问物联网平台接口权限的虚拟实体。

需调用 STS 的 AssumeRole 接口，获取访问物联网平台接口的临时授权。在调用 STS 的请求中，RoleArn 的值需为角色 C 的 Arn。如果调用成功，STS 会返回临时的 AccessKeyId、AccessKeySecret和 SecurityToken 作为访问凭证（凭证的过期时间，在调用 AssumeRole 的请求中指定）。将这个凭证发给需要访问的用户，该用户就可以获得访问物联网平台接口的临时权限。

为什么 RAM 和 STS 的使用这么复杂?

虽然 RAM 和 STS 的概念和使用比较复杂，但这是为了账号的安全性和权限控制的灵活性而牺牲了部分易用性。

将子账号和角色分开，主要是为了将执行操作的实体和代表权限集合的虚拟实体分开。如果某用户需要使用多种权限，如读/写权限，但是实际上每次操作只需要其中的一部分权限，那么就可以创建两个角色。这两个角色分别具有读或写权限。然后，创建一个可以扮演这两个角色的用户子账号。当用户需要读权限的时候，就可以扮演其中拥有读权限的角色；使用写权限的时候同理。这样可以降低每次操作中权限泄露的风险。而且，通过扮演角色，可以将角色权限授予其他用户，更加方便了协同使用。

STS 对权限的控制更加灵活。如按照实际需求设置有效时长。但是，如果需要一个长期有效的临时访问凭证，则可以只适用 RAM 子账号管理功能，而无需使用 STS。

在后面的章节中，我们将提供一些 RAM 和 STS 的使用指南和使用示例。如果您需要了解更多 RAM 和 STS 的代码详情，请参见 RAM APIRAM API和 STS APISTS API 。

8.2.2 自定义权限

权限指在某种条件下，允许 (Allow) 或 拒绝 (Deny) 对某些资源执行某些操作。

权限的载体是授权策略。自定义权限，即在自定义授权策略时定义某些权限。在 RAM 控制台，策略管理页面，单击新建授权策略开始创建自定义授权策略。创建自定义授权策略时，请选择模板为空白模板。

授权策略是 JSON 格式的字符串，需包含以下参数：

- `Action`：表示要授权的操作。IoT 操作都以*iot:* 开头。定义方式和示例，请参见本文档中Action 定义。
- `Effect` ： 表示授权类型，取值：`Allow`、`Deny`。
- `Resource` ：物联网平台目前暂不支持资源粒度的授权，请填写`*`。
- `Condition` ：表示鉴权条件。详细信息，请参见本文档中 Condition 定义。

Action 定义

`Action`是 API 的名称。在创建 IoT 的授权策略时，每个 Action 前缀均为*iot:*，多个 Action 以逗号分隔。并且，支持使用星号通配符。IoT API 名称定义，请参见#unique_188 。

下面介绍一些典型的 Action 定义示例。

- 定义单个 API。

```
"Action": "iot:CreateProduct"
```

- 定义多个API。

```
"Action": [
"iot:UpdateProduct",
"iot:QueryProduct"
]
```

- 定义所有只读 API。

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "iot:Query*",
        "iot:List*",
        "iot:Get*",
        "iot:BatchGet*",
        "iot:Check*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:DescribeDBInstances",
```

```
        "rds:DescribeDatabases",
        "rds:DescribeAccounts",
        "rds:DescribeDBInstanceNetInfo"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:ListRoles",
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mns:ListTopic",
        "mns:GetTopicRef"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "dhs:ListProject",
        "dhs:GetProject",
        "dhs:ListTopic",
        "dhs:GetTopic"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ots:ListInstance",
        "ots:GetInstance",
        "ots:ListTable",
        "ots:DescribeTable"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ons:OnsRegionList",
        "ons:OnsInstanceInServiceList",
        "ons:OnsTopicList",
        "ons:OnsTopicGet"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "hitsdb:DescribeRegions",
        "hitsdb:DescribeHiTSDBInstanceList",
        "hitsdb:DescribeHiTSDBInstance"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "fc:ListServices",
        "fc:GetService",
        "fc:GetFunction",
```

```
        "fc:ListFunctions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListShards",
        "log:ListLogStores",
        "log:ListProject"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "cms:QueryMetricList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "iot:Query*",
        "iot:List*",
        "iot:Get*",
        "iot:BatchGet*",
        "iot:Check*"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDatabases",
        "rds:DescribeAccounts",
        "rds:DescribeDBInstanceNetInfo"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:ListRoles",
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mns:ListTopic",
        "mns:GetTopicRef"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
```

```
        "ots:ListInstance",
        "ots:GetInstance",
        "ots:ListTable",
        "ots:DescribeTable"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "fc:ListServices",
        "fc:GetService",
        "fc:GetFunction",
        "fc:ListFunctions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListShards",
        "log:ListLogStores",
        "log:ListProject"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "cms:QueryMetricList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

· **定义所有读写 API。**

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "iot:*",
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDatabases",
        "rds:DescribeAccounts",
        "rds:DescribeDBInstanceNetInfo",
        "rds:ModifySecurityIps"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:ListRoles",
      "Resource": "*",
      "Effect": "Allow"
    },
```

```
    {
      "Action": [
        "mns:ListTopic",
        "mns:GetTopicRef"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "dhs:ListProject",
        "dhs:ListTopic",
        "dhs:GetProject",
        "dhs:GetTopic"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ots:ListInstance",
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetInstance"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ons:OnsRegionList",
        "ons:OnsInstanceInServiceList",
        "ons:OnsTopicList",
        "ons:OnsTopicGet"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "hitsdb:DescribeRegions",
        "hitsdb:DescribeHiTSDBInstanceList",
        "hitsdb:DescribeHiTSDBInstance",
        "hitsdb:ModifyHiTSDBInstanceSecurityIpList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "fc:ListServices",
        "fc:GetService",
        "fc:GetFunction",
        "fc:ListFunctions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListShards",
        "log:ListLogStores",
        "log:ListProject"
      ],
```

```
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "acs:Service": "iot.aliyuncs.com"
        }
      }
    },
    {
      "Action": [
        "cms:QueryMetricList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": "iot:*",
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "rds:DescribeDBInstances",
        "rds:DescribeDatabases",
        "rds:DescribeAccounts",
        "rds:DescribeDBInstanceNetInfo",
        "rds:ModifySecurityIps"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:ListRoles",
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "mns:ListTopic",
        "mns:GetTopicRef"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ots:ListInstance",
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetInstance"
      ],
      "Resource": "*",
```

```
      "Effect": "Allow"
    },
    {
      "Action": [
        "fc:ListServices",
        "fc:GetService",
        "fc:GetFunction",
        "fc:ListFunctions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListShards",
        "log:ListLogStores",
        "log:ListProject"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "acs:Service": "iot.aliyuncs.com"
        }
      }
    },
    {
      "Action": [
        "cms:QueryMetricList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Condition 定义

目前 RAM 授权策略支持访问 IP 限制、是否通过 HTTPS 访问、是否通过 MFA（多因素认证）访问、访问时间限制等多种鉴权条件。物联网平台的所有 API 均支持这些条件。

访问 IP 限制

访问控制可以限制访问 IoT 的源 IP 地址，并且支持根据网段进行过滤。以下是典型的使用场景示例。

- 限制单个 IP 地址和 IP 网段。例如，只允许 IP 地址为 10.101.168.111 或 10.101.169.111/24 网段的请求访问。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
```

```
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.168.111",
            "10.101.169.111/24"
          ]
        }
      }
    }
  ],
  "Version": "1"
}
```

- **限制多个 IP 地址。例如，只允许 IP 地址为 10.101.168.111 和 10.101.169.111 的请求访问。**

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.168.111",
            "10.101.169.111"
          ]
        }
      }
    }
  ],
  "Version": "1"
}
```

HTTPS 访问限制

访问控制可以限制是否通过 HTTPS 访问。

示例：限制必须通过 HTTPS 请求访问。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "acs:SecureTransport": "true"
        }
      }
    }
  ],
  "Version": "1"
}
```

MFA 访问限制

访问控制可以限制是否通过 MFA（多因素认证）访问。MFA访问适用于控制台登录，使用API访问无需MFA码。

示例：限制必须通过 MFA 请求访问。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "Bool": {
          "acs:MFAPresent ": "true"
        }
      }
    }
  ],
  "Version": "1"
}
```

访问时间限制

访问控制可以限制请求的访问时间，即只允许或拒绝在某个时间点范围之前的请求。

示例：用户可以在北京时间 2019 年 1 月 1 号凌晨之前访问，之后则不能访问。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "DateLessThan": {
          "acs:CurrentTime": "2019-01-01T00:00:00+08:00"
        }
      }
    }
  ],
  "Version": "1"
}
```

典型使用场景

结合以上对 Action、Resource 和 Condition 的定义，下面介绍一些典型使用场景的授权策略定义和授权方法。

允许访问的授权策略示例

场景：定义访问 IP 地址为 10.101.168.111/24 网段的用户访问 IoT 的权限，且要求只能在 2019-01-01 00:00:00 之前访问和通过 HTTPS 访问。

```
{
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.168.111/24"
          ]
        },
        "DateLessThan": {
          "acs:CurrentTime": "2019-01-01T00:00:00+08:00"
        },
        "Bool": {
          "acs:SecureTransport": "true"
        }
      }
    }
  ],
  "Version": "1"
}
```

拒绝访问的授权策略示例

场景：拒绝访问 IP 地址为 10.101.169.111 的用户对 IoT 执行读操作。

```
{
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "iot:Query*",
        "iot:List*",
        "iot:Get*",
        "iot:BatchGet*"
      ],
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.169.111"
          ]
        }
      }
    }
  ],
  "Version": "1"
}
```

授权策略创建成功后，在访问控制 RAM 控制台，用户管理页面，将此权限授予子账号用户。获得授权的子账号用户就可以进行权限中定义的操作。创建子账号和授权操作帮助，请参见#unique_183。

8.2.3 IoT API 授权映射表

定义授权策略，为RAM用户授予具体某些API的访问权限。

为RAM用户授权的具体方法，请参见#unique_184。

下表中列举的物联网平台 API 名称，即您在创建物联网平台相关授权策略时，参数 Action 的可选值。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
CreateProduct	iot:CreateProduct	*	创建产品。
UpdateProduct	iot:UpdateProduct	*	修改产品。
QueryProduct	iot:QueryProduct	*	查询产品信息。
QueryProductList	iot:QueryProductList	*	查询产品列表。
DeleteProduct	iot:DeleteProduct	*	删除产品。
CreateProductTags	iot:CreateProductTags	*	创建产品标签。
UpdateProductTags	iot:UpdateProductTags	*	更新产品标签。
DeleteProductTags	iot:DeleteProductTags	*	删除产品标签。
ListProductTags	iot:ListProductTags	*	查询产品标签。
ListProductByTags	iot:ListProductByTags	*	根据标签查询产品。
RegisterDevice	iot:RegisterDevice	*	注册设备。
QueryDevice	iot:QueryDevice	*	查询指定产品下的所有设备列表。
DeleteDevice	iot:DeleteDevice	*	删除设备。
QueryPageByApplyId	iot:QueryPageByApplyId	*	查询批量注册的设备信息。
BatchGetDeviceState	iot:BatchGetDeviceState	*	批量获取设备状态。
BatchRegisterDeviceWithApplyId	iot:BatchRegisterDeviceWithApplyId	*	根据ApplyId批量申请设备。
BatchRegisterDevice	iot:BatchRegisterDevice	*	批量注册设备（随机生成设备名）。
QueryBatchRegisterDeviceStatus	iot:QueryBatchRegisterDeviceStatus	*	查询批量注册设备的处理状态和结果。
BatchCheckDeviceNames	iot:BatchCheckDeviceNames	*	批量自定义设备名称。
QueryDeviceStatistics	iot:QueryDeviceStatistics	*	获取设备的统计数量。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
QueryDeviceEventData	iot:QueryDeviceEventData	*	获取设备的事件历史数据。
QueryDeviceServiceData	iot:QueryDeviceServiceData	*	获取设备的服务记录历史数据。
SetDeviceProperty	iot:SetDeviceProperty	*	设置设备的属性。
SetDevicesProperty	iot:SetDevicesProperty	*	批量设置设备属性。
InvokeThingService	iot:InvokeThingService	*	调用设备的服务。
InvokeThingsService	iot:InvokeThingsService	*	批量调用设备服务。
QueryDevicePropertyStatus	iot:QueryDevicePropertyStatus	*	查询设备的属性快照。
QueryDeviceDetail	iot:QueryDeviceDetail	*	查询设备详情。
DisableThing	iot:DisableThing	*	禁用设备。
EnableThing	iot:EnableThing	*	解除设备的禁用状态。
GetThingTopo	iot:GetThingTopo	*	查询设备拓扑关系。
RemoveThingTopo	iot:RemoveThingTopo	*	移除设备拓扑关系。
NotifyAddThingTopo	iot:NotifyAddThingTopo	*	通知云端增加设备拓扑关系。
QueryDevicePropertyData	iot:QueryDevicePropertyData	*	获取设备的属性历史数据。
QueryDevicePropertiesData	iot:QueryDevicePropertiesData	*	批量查询指定设备的属性上报数据。
GetGatewayBySubDevice	iot:GetGatewayBySubDevice	*	根据挂载的子设备信息查询对应的网关设备信息。
SaveDeviceProp	iot:SaveDeviceProp	*	为指定设备设置标签。
QueryDeviceProp	iot:QueryDeviceProp	*	查询指定设备的标签列表。
DeleteDeviceProp	iot:DeleteDeviceProp	*	删除设备标签。
QueryDeviceByTags	iot:QueryDeviceByTags	*	根据标签查询设备。
CreateDeviceGroup	iot:CreateDeviceGroup	*	创建分组。
UpdateDeviceGroup	iot:UpdateDeviceGroup	*	更新分组信息。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
DeleteDeviceGroup	iot:DeleteDeviceGroup	*	删除分组。
BatchAddDeviceGroupRelations	iot:BatchAddDeviceGroupRelations	*	添加设备到分组。
BatchDeleteDeviceGroupRelations	iot:BatchDeleteDeviceGroupRelations	*	将设备从分组中删除。
QueryDeviceGroupInfo	iot:QueryDeviceGroupInfo	*	查询分组详情。
QueryDeviceGroupList	iot:QueryDeviceGroupList	*	查询分组列表。
SetDeviceGroupTags	iot:SetDeviceGroupTags	*	添加或更新分组标签。
QueryDeviceGroupTagList	iot:QueryDeviceGroupTagList	*	查询分组标签列表。
QueryDeviceGroupByDevice	iot:QueryDeviceGroupByDevice	*	查询指定设备所在的分组列表。
QueryDeviceListByDeviceGroup	iot:QueryDeviceListByDeviceGroup	*	查询分组中的设备列表。
QuerySuperDeviceGroup	iot:QuerySuperDeviceGroup	*	根据子分组ID查询父分组信息。
QueryDeviceGroupByTags	iot:QueryDeviceGroupByTags	*	根据标签查询设备分组。
StartRule	iot:StartRule	*	启动规则。
StopRule	iot:StopRule	*	暂停规则。
ListRule	iot:ListRule	*	查询规则列表。
GetRule	iot:GetRule	*	查询规则详情。
CreateRule	iot:CreateRule	*	创建规则。
UpdateRule	iot:UpdateRule	*	修改规则。
DeleteRule	iot:DeleteRule	*	删除规则。
CreateRuleAction	iot:CreateRuleAction	*	创建规则中的数据转发方法。
UpdateRuleAction	iot:UpdateRuleAction	*	修改规则中的数据转发方法。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
DeleteRuleAction	iot:DeleteRuleAction	*	删除规则中的数据转发方法。
GetRuleAction	iot:GetRuleAction	*	查询规则中的数据转发方法的详细信息。
ListRuleActions	iot:ListRuleActions	*	获取规则中的数据转发方法列表。
Pub	iot:Pub	*	发布消息。
PubBroadcast	iot:PubBroadcast	*	向订阅了指定产品广播Topic的所有设备发送消息。
RRpc	iot:RRpc	*	发送消息给设备并得到设备响应。
CreateProductTopic	iot:CreateProductTopic	*	创建产品Topic类。
DeleteProductTopic	iot:DeleteProductTopic	*	删除产品Topic类。
QueryProductTopic	iot:QueryProductTopic	*	查询产品Topic类列表。
UpdateProductTopic	iot:UpdateProductTopic	*	修改产品Topic类。
CreateTopicRouteTable	iot:CreateTopicRouteTable	*	新建Topic间的消息路由关系。
DeleteTopicRouteTable	iot:DeleteTopicRouteTable	*	删除Topic路由关系。
QueryTopicReverseRouteTable	iot:QueryTopicReverseRouteTable	*	查询指定Topic订阅的源Topic。
QueryTopicRouteTable	iot:QueryTopicRouteTable	*	查询向指定Topic订阅消息的目标Topic。
GetDeviceShadow	iot:GetDeviceShadow	*	查询设备的影子信息。
UpdateDeviceShadow	iot:UpdateDeviceShadow	*	修改设备的影子信息。
SetDeviceDesiredProperty	iot:SetDeviceDesiredProperty	*	为指定设备批量设置期望属性值。
QueryDeviceDesiredProperty	iot:QueryDeviceDesiredProperty	*	查询指定设备的期望属性值。
BatchUpdateDeviceNickname	iot:BatchUpdateDeviceNickname	*	批量更新设备备注名称。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
QueryDeviceFileList	iot:QueryDeviceFileList	*	查询指定设备上传到物联网平台的所有文件列表。
QueryDeviceFile	iot:QueryDeviceFile	*	查询指定设备上传到物联网平台的指定文件信息。
DeleteDeviceFile	iot:DeleteDeviceFile	*	删除指定设备上传到物联网平台的指定文件。
QueryLoRaJoinPermissions	iot:QueryLoRaJoinPermissions	*	查询LoRaWAN入网凭证列表。
CreateLoRaNodesTask	iot:CreateLoRaNodesTask	*	生成批量注册LoRaWAN设备的任务。
GetLoraNodesTask	iot:GetLoraNodesTask	*	查询批量注册LoRaWAN设备任务的状态。

8.2.4 子账号访问

用户可以使用RAM子账号访问物联网平台资源。本文介绍如何创建子账号，如何授予子账号访问物联网平台资源的权限，和子账号用户如何登录物联网平台控制台。

您需先创建子账号，并通过授权策略授予子账号访问物联网平台的权限。创建自定义授权策略的方法，请参见#unique_184。

创建子账号

如果您已有子账号，请忽略此操作。

1. 用主账号登录访问控制 RAM 控制台。
2. 在左侧导航栏人员管理菜单下，单击用户。
3. 单击新建用户。
4. 输入登录名称和显示名称。
5. 在访问方式区域下，选择控制台密码登录或编程访问，并设置具体的登录信息。

   说明:
   为了保障账号安全，建议仅为RAM用户选择一种登录方式，避免RAM用户离开组织后仍可以通过访问密钥访问阿里云资源。
6. 单击确认。

7. 身份验证。阿里云可能会进行用户身份验证，并向您的账号预留的联系手机号中发送验证码。请将收到的验证码填入验证对话框中。

子账号创建完成后，子账号用户便可通过子用户登录链接登录阿里云官网和控制台。子用户的登录地址，请在访问控制 RAM 控制台的概览页面查看。

但是，在获得授权之前，该子账号无法访问您的阿里云资源。您需为子账号授予物联网平台的访问权限。

授权子账号访问物联网平台

在访问控制 RAM 控制台中，您可以在用户页，为单个子账号进行授权；也可以在用户组页，为整个群组授予相同的权限。下面我们以为单个子账号授权为例，介绍授权操作流程。

1. 用主账号登录访问控制 RAM 控制台。
2. 在左侧导航栏人员管理菜单下，单击用户。
3. 勾选要授权的子账号，单击下方添加权限。
4. 在授权对话框中，选中您要授予该子账号的物联网平台授权策略，再单击确定。

> 说明:
> 如果您要为子账号授予自定义权限，请先创建授权策略。授权策略的创建方法，请参见#unique_184。

授权成功后，子账号用户便可访问授权策略中定义的资源，和进行授权策略中定义的操作。

子账号登录控制台

阿里云主账号登录是从阿里云官网主页直接登录，但是子账号需从子用户登录页登录。

1. 获取子用户登录页的链接地址。

   用主账号登录访问控制 RAM 控制台，在概览页的账号管理区域下，查看用户登录地址，并将链接地址分发给子账号的用户。
2. 子账号用户访问子用户登录页进行登录。

   子账号用户登录方式有以下三种。

   - 方式一：`<$username>@<$AccountAlias>.onaliyun.com`。例如：`username@company-alias.onaliyun.com`。

     > 说明:

RAM用户登录账号为UPN（User Principal Name）格式，即RAM控制台用户列表中所见的用户登录名称。<$username>为RAM用户名称，<$AccountAlias>.onaliyun.com为默认域名。

- 方式二：`<$username>@<$AccountAlias>`。例如：`username@company-alias`。

  说明：
  <$username>为RAM用户名称，<$AccountAlias>为账号别名。

- 方式三：如果创建了域别名，也可以使用域别名登录，格式为：`<$username>@<$DomainAlias>`。

  说明：
  <$username>为RAM用户名称，<$DomainAlias>为域别名。

3. 单击页面右上角控制台按钮，进入管理控制台。
4. 单击产品与服务，选择物联网平台，即可进入物联网控制台。

子账号用户登录物联网控制台后，便可在控制台中，进行已获授权的操作。

8.2.5 进阶使用STS

STS权限管理系统是比访问控制（RAM）更为严格的权限管理系统。使用STS权限管理系统进行资源访问控制，需通过复杂的授权流程，授予子账号用户临时访问资源的权限。

子账号和授予子账号的权限均长期有效。删除子账号或解除子账号权限，均需手动操作。发生子账号信息泄露后，如果无法及时删除该子账号或解除权限，可能给您的阿里云资源和重要信息带来危险。所以，对于关键性权限或子账号无需长期使用的权限，您可以通过STS权限管理系统来进行控制。

图 8-1: 子账号获得临时访问权限的操作流程

步骤一：创建角色

RAM角色是一种虚拟用户，是承载操作权限的虚拟概念。

1. 使用阿里云主账号登录访问控制 RAM 控制台。
2. 单击RAM角色管理 > 新建RAM角色，进入角色创建流程。
3. 选择可信实体类型为阿里云账号，单击下一步。

4. 输入角色名称和备注，选择云账号为当前云账号或其他云账号，单击完成。

> **说明:**
> 若选择其他云账号，需要填写其他云账号的ID。

步骤二：创建角色授权策略

角色授权策略，即定义要授予角色的资源访问权限。

1. 在访问控制 RAM 控制台左侧导航栏，单击权限管理 > 权限策略管理。
2. 单击新建权限策略。
3. 输入授权策略名称、策略模式和策略内容，单击确认。

   如果策略模式选择为脚本配置，授权策略内容的编写方法请参见语法结构。

   IoT资源只读权限的授权策略内容示例如下：

```
{
    "Version":"1",
    "Statement":[
        {
            "Action":[
                "rds:DescribeDBInstances",
                "rds:DescribeDatabases",
                "rds:DescribeAccounts",
                "rds:DescribeDBInstanceNetInfo"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":"ram:ListRoles",
            "Effect":"Allow",
            "Resource":"*"
        },
        {
            "Action":[
                "mns:ListTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "dhs:ListProject",
                "dhs:ListTopic",
                "dhs:GetTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "ots:ListInstance",
                "ots:ListTable",
                "ots:DescribeTable"
            ],
```

```
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "log:ListShards",
                "log:ListLogStores",
                "log:ListProject"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Effect":"Allow",
            "Action":[
                "iot:Query*",
                "iot:List*",
                "iot:Get*",
                "iot:BatchGet*"
            ],
            "Resource":"*"
        }
    ]
}
```

IoT资源读写权限的授权策略内容示例如下：

```
{
    "Version":"1",
    "Statement":[
        {
            "Action":[
                "rds:DescribeDBInstances",
                "rds:DescribeDatabases",
                "rds:DescribeAccounts",
                "rds:DescribeDBInstanceNetInfo"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":"ram:ListRoles",
            "Effect":"Allow",
            "Resource":"*"
        },
        {
            "Action":[
                "mns:ListTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "dhs:ListProject",
                "dhs:ListTopic",
                "dhs:GetTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
```

```
                "ots:ListInstance",
                "ots:ListTable",
                "ots:DescribeTable"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "log:ListShards",
                "log:ListLogStores",
                "log:ListProject"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Effect":"Allow",
            "Action":"iot:*",
            "Resource":"*"
        }
    ]
}
```

授权策略创建成功后，您就可以将该授权策略中定义的权限授予角色。

步骤三：为角色授权

角色获得授权后，才具有资源访问权限。您可以在RAM角色管理页，单击角色对应的添加权限按钮，为单个角色授权。同时为多个角色授权，请参见以下步骤。

1. 在访问控制 RAM 控制台页左侧导航栏，单击权限管理 > 授权。
2. 单击新增授权。
3. 在授权对话框中，被授权主体下，输入RAM角色名称，选中要授权的策略，再单击确定。

下一步，为子账号授予可以扮演该角色的权限。

步骤四：授予子账号角色扮演权限

虽然经过授权后，该角色已拥有了授权策略定义的访问权限，但角色本身只是虚拟用户，需要子账号用户扮演该角色，才能进行权限允许的操作。若任意子账号都可以扮演该角色，也会带来风险，因此只有获得角色扮演权限的子账号用户才能扮演角色。

授权子账号扮演角色的方法：先新建一个Resource参数值为角色ID的自定义授权策略，然后用该授权策略为子账号授权。

1. 在访问控制 RAM 控制台左侧导航栏，单击权限管理 > 权限策略管理。
2. 单击新建权限策略。
3. 输入授权策略名称，选择策略模式为脚本配置，输入策略内容，单击确认。

说明:

> 授权策略内容中，参数Resource 的值需为角色Arn。在RAM角色管理页面，单击角色名称，进入基本信息页，查看角色的Arn 。

角色授权策略示例：

```
{
    "Version":"1",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":"iot:QueryProduct",
            "Resource":"角色Arn"
        }
    ]
}
```

4. 授权策略创建成功后，返回访问控制 RAM 控制台主页。
5. 单击左侧导航栏中的人员管理 > 用户。
6. 在子账号列表中，勾选要授权的子账号，并单击下方的添加权限按钮。
7. 在授权对话框中，选中刚新建的角色授权策略，再单击确定。

授权完成后，子账号便有了可以扮演该角色的权限，就可以使用STS获取扮演角色的临时身份凭证，和进行资源访问。

步骤五：子账号获取临时身份凭证

获得角色授权的子账号用户，可以通过直接调用API或使SDK 来获取扮演角色的临时身份凭证：AccessKeyId、AccessKeySecret和SecurityToken。STS API和STS SDK详情，请参见访问控制文档中STS API和STS SDK。

使用API和SDK获取扮演角色的临时身份凭证需传入以下参数：

- RoleArn：需要扮演的角色Arn。
- RoleSessionName：临时凭证的名称（自定义参数）。
- Policy：授权策略，即为角色增加一个权限限制。通过此参数限制生成的Token的权限。不指定此参数，则返回的Token将拥有指定角色的所有权限。
- DurationSeconds：临时凭证的有效期。单位是秒，最小为900，最大为3600，默认值是3600。
- id 和secret：指需要扮演该角色的子账号的AccessKeyId和AccessKeySecret。

获取临时身份凭证示例

API示例：子账号用户通过调用STS的AssumeRole接口获得扮演该角色的临时身份凭证。

```
https://sts.aliyuncs.com?Action=AssumeRole
&RoleArn=acs:ram::1234567890123456:role/iotstsrole
&RoleSessionName=iotreadonlyrole
&DurationSeconds=3600
```

```
&Policy=<url_encoded_policy>
&<公共请求参数>
```

SDK示例：子账号用户使用STS的Python命令行工具接口获得扮演该角色的临时身份凭证。

```
$python ./sts.py AssumeRole RoleArn=acs:ram::1234567890123456:role/
iotstsrole RoleSessionName=iotreadonlyrole Policy='{"Version":"1","
Statement":[{"Effect":"Allow","Action":"iot:*","Resource":"*"}]}'
DurationSeconds=3600 --id=id --secret=secret
```

请求成功后，将返回扮演该角色的临时身份凭证：AccessKeyId、AccessKeySecret和SecurityToken。

步骤六：子账号临时访问资源

获得扮演角色的临时身份凭证后，子账号用户便可以在调用SDK的请求中传入该临时身份凭证信息，扮演角色。

Java SDK示例：子账号用户在调用请求中，传入临时身份凭证的AccessKeyId、AccessKeySecret和SecurityToken参数，创建IAcsClient对象。

```
IClientProfile  profile = DefaultProfile.getProfile("cn-hangzhou",
AccessKeyId,AccessSecret) ;
RpcAcsRequest request.putQueryParameter("SecurityToken", Token);
IAcsClient client = new DefaultAcsClient(profile);
AcsResponse response = client.getAcsResponse(request);
```