设备接入

用户指南

文档版本 1.0

发布日期 2022-08-30

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

目 录

1 概述

物联网平台提供海量设备的接入和管理能力，配合华为云其他产品同时使用，帮助快速构筑物联网应用，简化海量设备管理复杂性，节省人工操作，提升管理效率。使用设备接入控制台，可以实现对产品的创建、开发、调试，设备的注册、管理、鉴权、软固件升级。在设备接入控制台，可以创建规则引擎，满足用户实现设备联动和数据转发的需求；还可以存储产品和设备数据及生成相应统计报表，方便用户监控设备的各种状态。

功能	简介
产品	某一类具有相同能力或特征的设备的集合称为一款产品。您可以基于控制台快速进行产品模型和插件的开发，同时提供在线调试、自定义Topic等多种能力，端到端指引物联网开发，帮助开发者提升集成开发效率、缩短物联网解决方案建设周期。
产品模型	又称Product Model，用于定义一款接入设备所具备的属性（如颜色、大小、采集的数据、可识别的指令或者设备上报的事件等信息）。产品模型可以在设备接入控制台直接创建。
设备	归属于某个产品下的设备实体，每个设备具有一个唯一的标识码。设备可以是直连物联网平台的设备，也可以是代理子设备连接物联网平台的网关。
设备鉴权	设备接入物联网平台时，需要对接入平台的设备进行鉴权认证。目前华为物联网平台支持设备使用密钥和X.509两种方式进行鉴权认证。待平台验证通过，设备成功连接到物联网平台后，就可以进行数据通信。
群组与标签	群组是一系列设备的集合，用户可以对资源空间下所有设备，根据区域、类型等不同规则进行分类建立群组，以便处理对海量设备的批量管理和操作。 物联网平台支持定义不同的标签，并对设备打标签。
软固件升级	用户可以通过OTA的方式对LwM2M协议和MQTT协议的设备进行软固件升级。
设备影子	设备影子是一个JSON文件，用于存储设备的在线状态、设备最近一次上报的设备属性、应用服务器期望下发的配置。每个设备有且只有一个设备影子，设备可以获取和设置设备影子以此来同步状态，这个同步可以是影子同步给设备，也可以是设备同步给影子。

功能	简介
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，通过网关进行数据转发。
规则引擎	用户可以在物联网平台上对接入平台的设备设定相应的规则，在条件满足所设定的规则后，平台会触发相应的动作来满足用户需求。包含设备联动和数据转发两种类型。
监控运维	提供查看统计报表、在线调试、消息跟踪、当前告警、运行日志等监控运维功能。用户可以使用这些功能，监控设备运行状态、设备消息通讯、用户操作，快速追查定位故障，保障设备的可靠性及安全性。
资源空间	可以理解为在物联网平台中为您的业务划分的一个资源空间，您在平台中创建的资源（如产品、设备等）都需要归属到某个资源空间，您可以基于资源空间实现多业务应用的分域管理。
IoTDA实例	为满足物联网不同设备规模的企业客户诉求，设备接入服务提供基础版（原共享实例）、标准版（标准实例）、企业版（专享实例）三种产品形态。您可以可以根据企业业务场景、设备规模和数据采集频率，购买最合适的实例类型和实例规格。
数据上报	当设备完成和物联网平台对接后，一旦设备上电，设备基于在设备定义上的业务逻辑进行数据采集和上报，可以是基于周期或者事件触发。
数据转发	数据转发功能用于提供IoTDA与其他第三方以及华为云服务的连接通道，从而实现将设备数据平滑流转至消息中间件、存储、数据分析、业务应用。
命令下发	为能有效地对设备进行管理，设备的产品模型中定义了物联网平台可向设备下发的命令，应用服务器可以调用物联网平台开放的API接口向设备下发命令，以实现对设备的远程控制。

使用限制

为保证良好的显示效果和易用性体验，请使用兼容性良好的浏览器，对于浏览器的要求如下：

浏览器类型	版本要求	分辨率
Microsoft Edge	支持和测试最新的3个稳定版本。	推荐1366 x 768分辨率
Firefox	支持和测试最新的3个稳定版本。	
Google Chrome	支持和测试最新的3个稳定版本。	

2 IoTDA 实例

2.1 实例介绍

华为物联网平台提供实例助力解决数据和资源隔离等问题，当前华为云物联网平台提供标准版（标准实例）。

- **标准版**

  标准版提供免费试用，您可以在每日限定消息数和设备数范围内免费体验标准版的所有功能。

  如需更多设备数和消息数，您可以在标准版实例详情界面进行升级配置，根据需要选择相应的规格单元。

购买实例的约束说明

购买实例存在一些约束，当您登录后无法购买实例，或者购买实例后创建失败，请参考以下约束说明进行检查，并解除限制。

- 购买标准实例的约束说明
  - 同一个区域下，每个租户只能开通一个免费试用实例。

2.2 购买实例

操作步骤

步骤1 配置访问授权后，在侧导航栏，选择“IoTDA实例”，单击企业版右侧的“购买实例”。

步骤2 在页面中填写实例配置信息，系统会根据您选择的“实例规格”和“购买时长”自动计算费用。

参数名称	参数说明
计费模式	选择实例的计费模式，当前仅支持“包年/包月”。

参数名称	参数说明
区域	设备接入服务部署的区域，当前支持“华北-北京四”、“华东-上海一”、“华南-广州”。 **说明** 不同区域之间的云服务产品内网互不相通，请就近选择靠近您业务的区域，可降低网络时延，提高访问速度。
网络	选择虚拟私有云和子网。 如果需要创建新的虚拟私有云，请创建**虚拟私有云**。
安全组	选择实例所关联的安全组，请提前创建**安全组**。
公网接入	提供设备接入公网接入能力，请根据需要配置，避免不必要的浪费。
私网接入	• 勾选：购买实例时会自动购买VPC终端节点，并自动分配接入地址 • 未勾选：仍然需要私网接入，可以自行购买**VPC终端节点**对接。
接入端口	支持接入端口可配置，并提供了默认端口，支持可配置的端口如下： 应用接入：HTTPS(443)、AMQPS(5671) 设备接入：CoAP(5683)、CoAPS(5684)、MQTT(1883)、MQTTS(8883)、HTTPS(8943)
实例版本	企业版。
企业项目	该参数仅对开通企业项目的企业客户账号显示。企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。了解更多企业项目相关信息，请查看**企业管理**。
标签	标签以键值对的形式表示，用于标识实例，便于对实例进行分类和搜索。此处的标签仅用于实例的过滤和管理。详情可参考**实例标签管理**。
实例名称	设备接入实例的名字，方便根据名字管理实例。支持中文汉字、大小写字母、数字、下划线（_）和中划线（-），最大长度不超过64。
密码算法	通用加密算法（支持RSA，SHA256等国际通用的密码算法），支持SM系列商密算法（支持SM2，SM3，SM4等国密算法）
实例描述	企业版实例的描述，可根据实例用户、实例的用途进行简单的描述。
购买时长	按月购买。 购买时可勾选自动续费，勾选后在实例到期时会自动对实例续费。

步骤3 单击“立即购买”，进入实例规格确认页面。

步骤4 规格确认无误后，单击“去支付”。

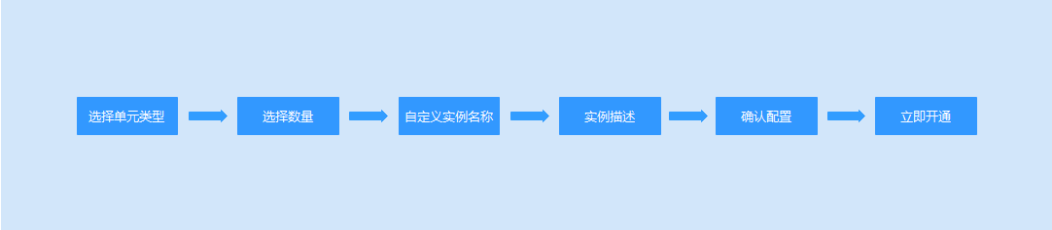
步骤5 在支付页面选择支付方式后，单击“确认付款”完成购买。

----结束

购买标准实例

标准实例提供灵活可配置的实例规格，适用于各类企业客户场景，可根据业务模型选购更经济的平台实例。

开通之前，请浏览整体开通流程，以便提高操作效率。



操作步骤

步骤1 在侧导航栏，选择“IoTDA实例”，单击标准版右侧的“购买实例”。

步骤2 在页面中填写实例配置信息，系统会根据您选择的“实例规格”“按需”计算费用。

参数名称	参数说明
计费模式	选择实例的计费模式，当前仅支持“按需计费”。
区域	设备接入服务部署的区域，当前支持“亚太-曼谷”、“亚太-新加坡”、“非洲-约翰内斯堡”和“中国-香港”。 **说明** 不同区域之间的云服务产品内网互不相通，请就近选择靠近您业务的区域，可降低网络时延，提高访问速度。
规格配置	支持免费规格单元SUF、小规格单元SU1、中规格单元SU2、大规格单元SU3、超大规格SU4。新增规格测算功能，可根据自身业务的上下行TPS获取推荐的实例规格。 **说明** • IoTDA标准版实例的规格=实例包含的单元数量*每个单元的规格； • 一个标准版实例可以包含多个相同类型的单元，单个实例的单元数量上限为100个，单个实例的消息上下行TPS峰值最高到10万TPS（比如实例包含100个S3，但峰值TPS最高只能到10万TPS）； • 支持对实例的单元数进行在线升配，如一个标准实例从3个SU1变更为5个SU2。 • 同一个实例不支持混合叠加多个版本类型的单元，比如M个SU1+N个SU2。
企业项目	该参数仅对开通企业项目的企业客户账号显示。企业项目是一种云资源管理方式，企业项目管理服务提供统一的云资源按项目管理，以及项目内的资源管理、成员管理。了解更多企业项目相关信息，请查看企业管理。

参数名称	参数说明
标签	标签以键值对的形式表示，用于标识实例，便于对实例进行分类和搜索。此处的标签仅用于实例的过滤和管理。详情可参考**实例标签管理**。
实例名称	标准实例的名称，方便根据名称管理实例。支持中文汉字、大小写字母、数字、下划线（_）和中划线（-），最大长度不超过64。
实例描述	标准实例的描述，可根据实例用户、实例的用途进行简单的描述。

步骤3 单击“确认配置”，进入实例规格确认页面。

步骤4 规格确认无误后，单击“立即开通”。

----结束

2.3 实例管理

选择实例

实例创建成功后，需要先切换到该实例，才可以在该实例下创建产品、设备和进行其他功能设置等。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 单击左侧导航栏“IoTDA实例”，会显示实例类型。您可以根据自己需求，单击“切换实例”切换当前实例类型。

图 2-1 实例管理-切换实例

----结束

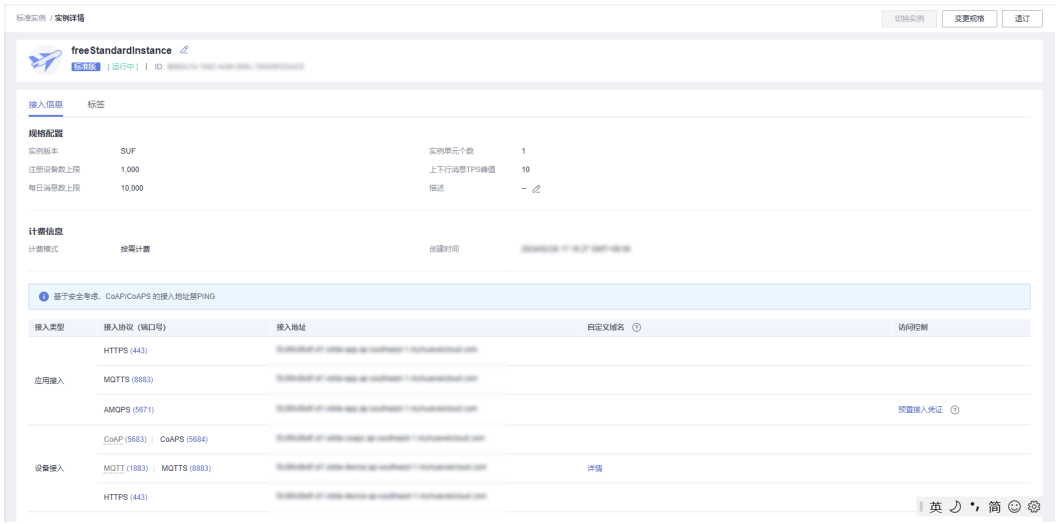
查看实例

IoTDA实例购买成功后，您可以进入实例详情页面查看实例的详情信息，包括实例Id，实例名称，实例规格配置等信息。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 选择左侧导航栏“IoTDA实例”，单击实例对应的“详情”进入实例详情页面。

图 2-2 实例管理-实例详情



----结束

变更实例规格

IoTDA实例创建成功后，您可以根据业务需要，升级实例规格，实例规格变更不会对业务产生影响。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏“IoTDA实例”，单击实例名称进入实例详情页面。

步骤3 单击右上角的“变更配置”，根据您的业务需求，选择变更后的实例规格。

图 2-3 实例管理-变更规格入口

步骤4 可设置延迟生效，选定维护的时间窗口，将会在指定时间窗口内进行变更。

图 2-4 实例管理-变更规格

----结束

退订

针对不再继续使用的实例，您可以单击对应实例的“退订”按钮，释放您的云服务资源。详细退订操作请参考**退订管理**。

图 2-5 实例管理-退订实例

2.4 实例标签管理

2.4.1 标签概述

操作场景

对于拥有大量云资源的用户，可以通过给云资源打标签，快速查找具有某标签的云资源，可对这些资源标签统一进行检视、修改、删除等操作，方便用户对云资源的管理。还可利用标签功能实现基于业务维度的资源成本统计。

标签命名规则

- 每个标签由一对键值对（Key-Value）组成。
- 每个IoTDA实例最多可以添加20个标签。
- 对于每个资源，每个标签键（Key）都必须是唯一的，每个标签键（Key）只能有一个值（Value）。
- 标签共由两部分组成：“标签键”和“标签值”，其中，“标签键”和“标签值”的命名规则如表1所示。

表 2-1 标签命名规则

参数	规则	样例
标签键	不能为空。对于同一个实例，Key值唯一。最大长度36个字符。 字符集：A-Z，a-z ， 0-9， ‘-’， ‘_’， UNICODE字符（\u4E00-\u9FFF）	Organization
标签值	每个值最大长度43个字符，可以为空字符串。 字符集：A-Z，a-z ， 0-9， ‘.’， ‘-’， ‘_’，UNICODE字符（\u4E00-\u9FFF）	Apache

说明

如您的组织已经设定IoTDA的相关标签策略，则需按照标签策略规则为实例添加标签。标签如果不符合标签策略的规则，则可能会导致实例创建失败，请联系组织管理员了解标签策略详情。

2.4.2 添加标签

给IoTDA实例添加标签有以下两种方法。

- **在实例详情页添加标签**
- **在标签管理页面添加标签**

预定义标签的使方法请参考**预定义标签的使用方法**。

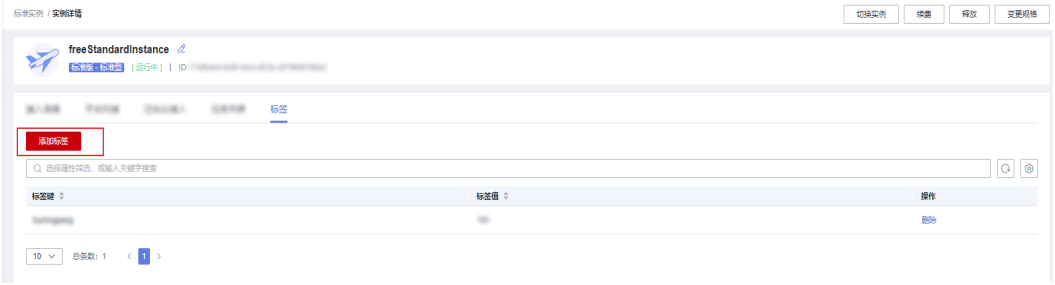
在实例详情页添加标签

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏“IoTDA实例”，单击企业版实例对应的“详情”进入实例详情页面。

步骤3 在“标签”页签下，单击“添加标签”，在弹出的“添加标签”窗口，输入标签的键和值。标签命名规则如**表1**所示。

图 2-6 实例管理-标签添加



----结束

在标签管理页面添加标签

说明

此方法适用于为多个资源统一添加拥有同样标签键的标签。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

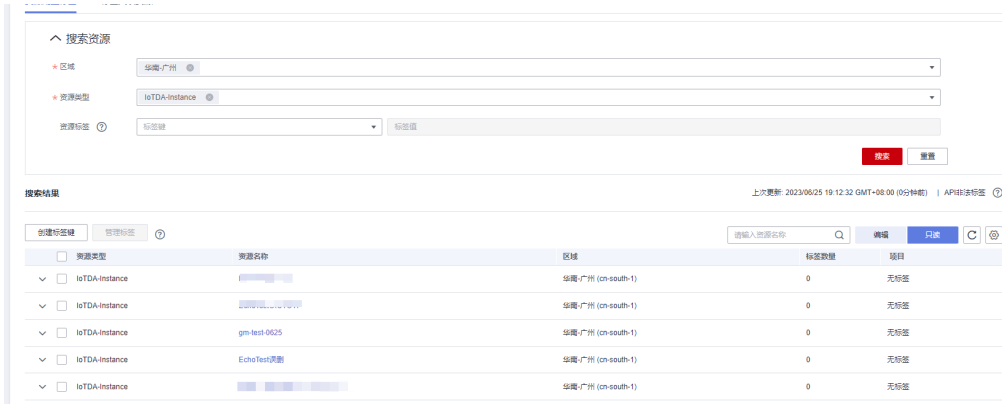
步骤2 在右上角的用户名下选择“标签管理”，进入标签管理服务页面。

图 2-7 标签管理

步骤3 在“资源标签”页面，勾选资源所在的区域，“资源类型”请选择“IoTDA-Instance”，单击“搜索”。系统列出所选区域下的所有IoTDA实例资源。

步骤4 在“搜索结果”区域，单击“创建标签键”，输入键（例如：项目），单击“确定”。创建完成后，该标签键会添加至资源标签列表（如图3）。如果列表中没有显示该标签，单击 ，在下拉列表中勾选创建的标签键。默认该标签键的值为“无标签”，您还需要为每一个资源对应的标签值赋值，完成标签与资源的关联，继续下一步。

图 2-8 资源列表



步骤5 单击“编辑”，切换资源标签列表为可编辑状态。

步骤6 选择设备接入实例资源所在行，输入标签“值”（例如：A）。为标签键赋值后，“标签数量”将加1。按照同样方法依次为其他实例添加标签值。

图 2-9 输入标签值

----结束

预定义标签的使用方法

如果有多种云资源需要添加同一标签，为了避免重复输入标签键和值，您可以在标签管理服务中预定义标签，然后在添加标签时直接选择键和值。具体步骤如下：

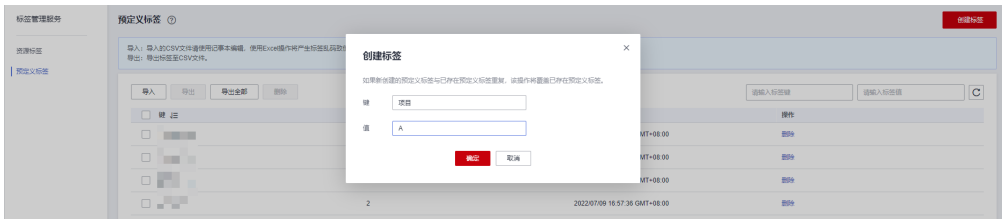
步骤1 登录管理控制台。

步骤2 在右上角的用户名下选择“标签管理”，进入标签管理服务页面。

步骤3 在左侧导航中选择“预定义标签”，单击“创建标签”，输入标签键和值（例如，项目-A）。

步骤4 选择“服务列表 > 设备接入服务”，按照上述添加标签的方法，在标签键和标签值输入框中下拉选择预定义的标签。

图 2-10 预定义标签



----结束

2.4.3 删除标签

如果某个标签已经不再适用于您的资源管理，您可以删除资源标签。有三种途径删除资源标签：

- **在实例详情页删除标签**
- **在标签管理页面单个删除**
- **在标签管理页面批量删除**

在实例详情页删除标签

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏“IoTDA实例”，单击企业版实例对应的“详情”进入实例详情页面。

步骤3 选择“标签”页签，单击标签所在行“操作”列下的“删除”，如果确认删除，在弹出的“删除标签”窗口，单击“确定”。

图 2-11 实例管理-删除标签

----结束

在标签管理页面单个删除

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在右上角的用户名下选择“标签管理”，进入标签管理服务页面。

图 2-12 标签管理

步骤3 在“资源标签”页面，勾选资源所在的区域，“资源类型”请选择“IoTDA-Instance”，单击“搜索”。系统列出所选区域下的所有IoTDA实例资源。

步骤4 页面下方展示的搜索结果包含“编辑”与“只读”两种状态，单击“编辑”，切换资源标签列表为可编辑状态。单击 在下拉列表中勾选需要删除的标签的“键”。勾选需要展示的标签键建议不超过10个。

图 2-13 标签列表

步骤5 单击待删除标签的设备接入实例资源所在行的，资源标签删除完成。

图 2-14 删除标签

----结束

在标签管理页面批量删除

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在右上角的用户名下选择“标签管理”，进入标签管理服务页面。

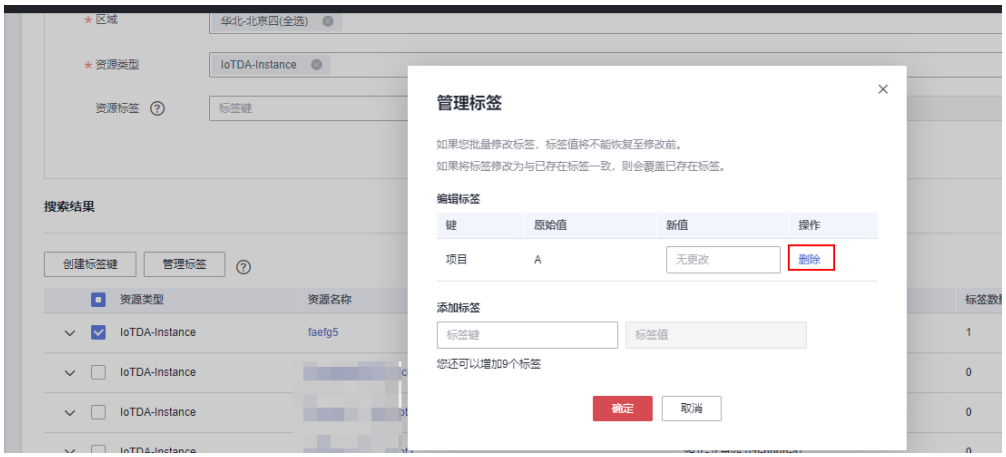
步骤3 在“资源标签”页面，勾选资源所在的区域，“资源类型”请选择“IoTDA-Instance”，单击“搜索”。系统列出所选区域下的所有IoTDA实例资源。

步骤4 勾选待删除标签的IoTDA实例资源。

步骤5 单击列表上方的“管理标签”，进入管理标签页面。

步骤6 单击待删除标签所在行的“删除”。单击“确认”，资源标签删除完成。

图 2-15 批量删除



----结束

2.4.4 使用标签检索资源

为云资源添加标签后，您可以通过本文所述的方法使用标签检索资源。

通过标签管理筛选资源

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

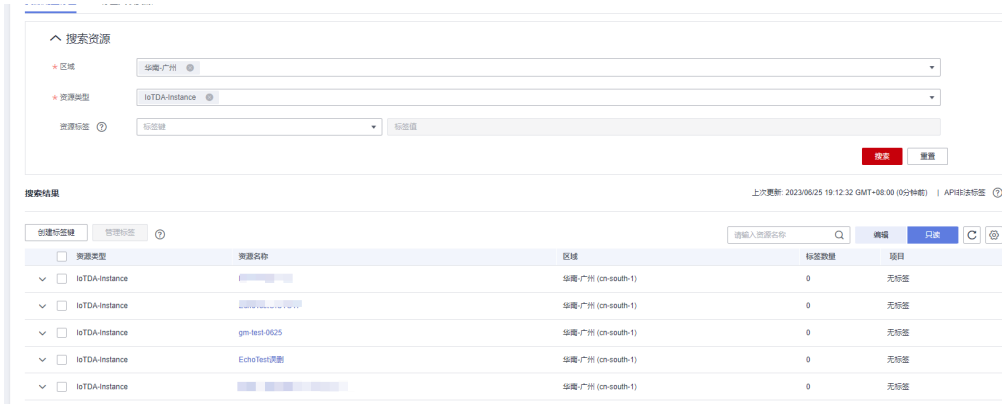
步骤2 在右上角的用户名下选择“标签管理”，进入标签管理服务页面。

图 2-16 标签管理

步骤3 在“资源标签”页面，设置搜索条件（包括区域、资源类型、资源标签）。

步骤4 单击“搜索”。搜索结果区域将列出所有符合搜索条件的资源。

图 2-17 资源列表



----结束

3 资源空间

指在物联网平台中为您的业务划分的一个资源空间，您在平台中创建的资源（如产品、设备等）都需要归属到某个资源空间，您可以基于资源空间实现多业务应用的分域管理。

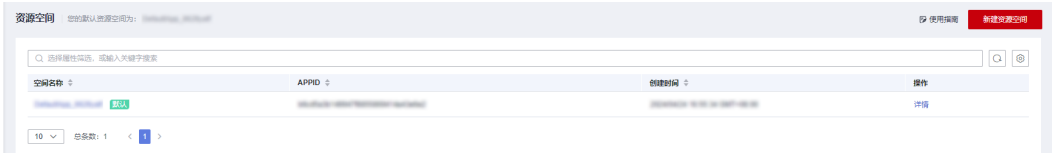
- 物联网平台允许用户最多创建10个资源空间，并默认首次开通服务时平台为用户自动创建的空间为默认资源空间。
- 创建资源空间时，物联网平台会分配一个app_id（接口调用时参数名为app_id）作为资源空间的唯一标识。
- 创建资源空间后，可以在资源空间中查看app_id。
- 默认资源空间不允许删除，其他资源空间删除后项目内的所有资源，如设备、产品、订阅数据在平台中的信息会被全部删除，并且不可恢复，请谨慎操作。

创建资源空间

用户首次开通设备接入服务时，物联网平台自动为用户创建了一个**默认资源空间**，“默认资源空间”每个实例仅有一个，不允许删除。

您可以基于默认资源空间创建产品，注册设备等，也可以参考如下步骤创建新的资源空间。

图 3-1 资源空间-资源空间列表



说明

如果您是老用户，即2020年04月27日00:00前开通的设备接入服务，平台设置默认资源空间的规则请参考**设置默认资源空间的规则**。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏选择“资源空间”，单击右上角的“新建资源空间”，在弹出的页面中，填写参数后，单击“确定”。

资源空间名称必须为账号下唯一。

图 3-2 资源空间-创建资源空间

----结束

查看资源空间

创建资源空间后，您可以在“资源空间列表”单击“详情”，查看该资源空间的APPID（即app_id）、创建时间、以及归属在该资源空间下的产品数、设备总数、群组数及创建的规则总数，若需要在其他资源空间下创建产品、设备、群组及规则，请在相应页面切换资源空间。

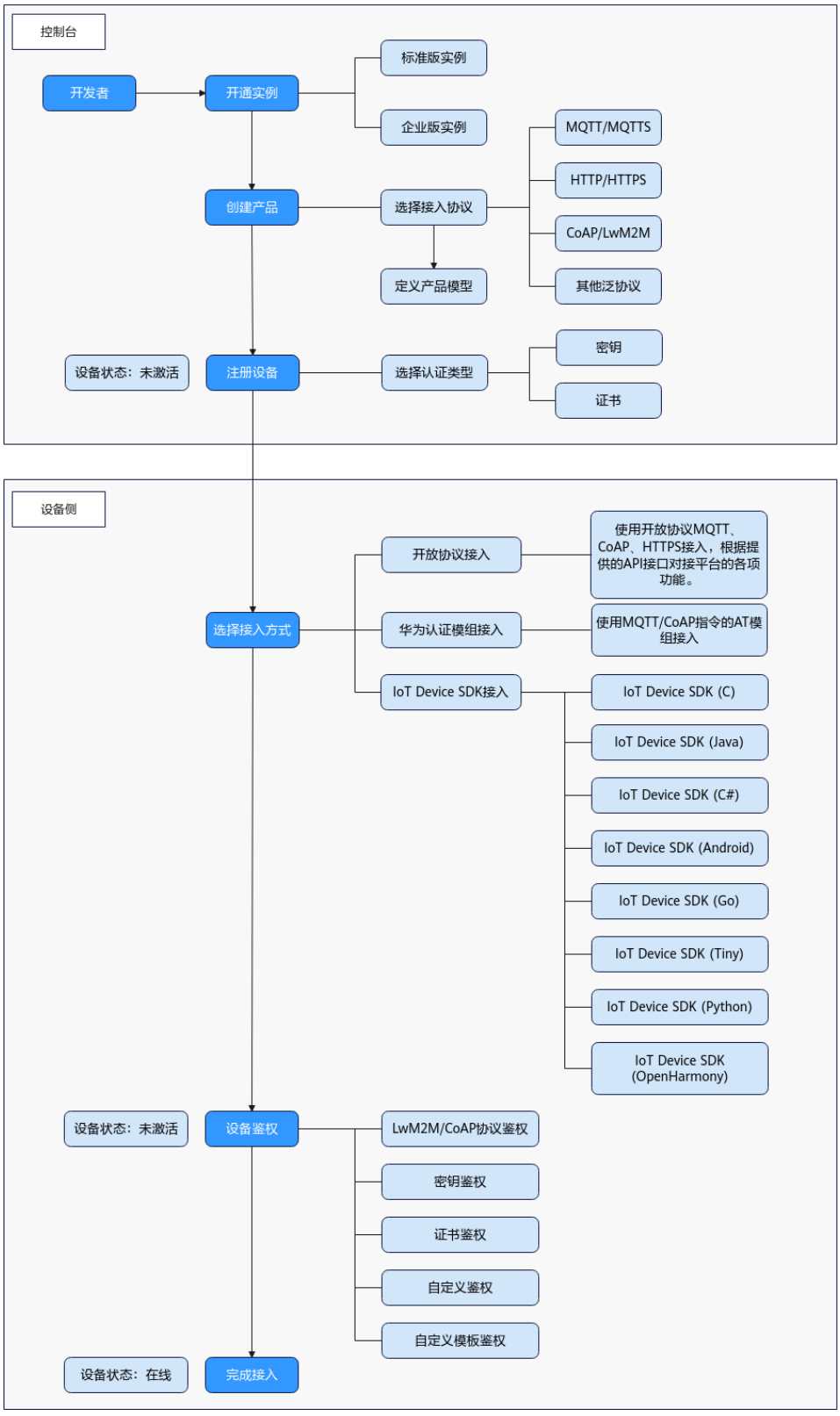
图 3-3 资源空间-查看资源空间

4 设备接入

4.1 概述

设备接入流程图

图 4-1 设备接入流程图



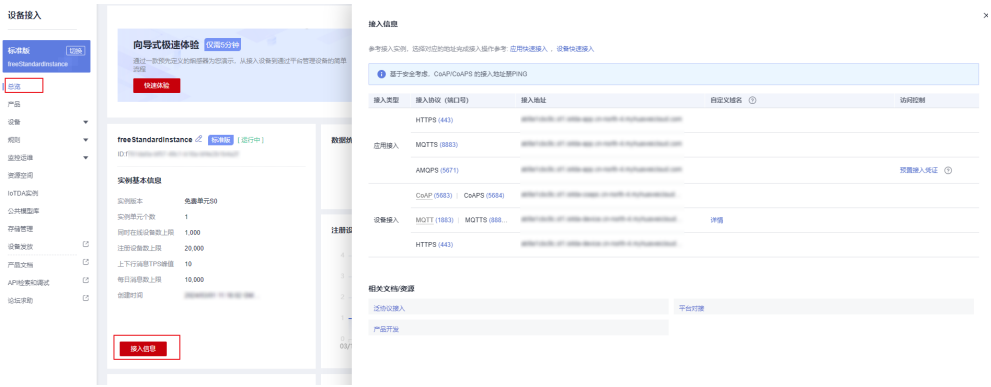
平台对接信息

1. 进入IoTDA的**管理控制台**界面，选择左侧导航栏“IoTDA实例”，切换至需要的版本实例。

图 4-2 实例管理-切换实例

2. 选择左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。

图 4-3 总览-获取接入信息



证书资源

当设备和应用需要对IoT平台进行校验时可使用以下证书。

说明

- 此证书文件只适用于华为云物联网平台，且必须配合对应域名使用。
- CA证书具有一个过期日期，在该日期后，这些证书将无法用于验证服务器的证书；请在 CA 证书的过期日期前替换这些证书，以确保设备可以正常的连接到IoT平台。

表 4-1 证书资源

证书包名称	region &版本	证书类型	证书格式	说明	下载
certificate	中国-香港、亚太-新加坡、亚太-曼谷、非洲-约翰内斯堡	设备侧证书	pem、jks、bks	用于设备校验平台的身份。该证书必须配合当前设备侧接入域名使用。	证书文件

4.2 设备鉴权

4.2.1

设备鉴权是指物联网平台对接入平台的设备进行身份认证。对于不同接入方式的设备，鉴权方式不同。

接入类型	鉴权方式
使用LwM2M/CoAP协议接入的设备	在设备接入物联网平台前，用户通过应用服务器调用创建设备接口或通过控制台在物联网平台注册设备。若为非安全设备，在设备接入物联网平台时携带设备唯一标识，完成设备的接入鉴权；当采用DTLS/DTLS+传输层安全协议接入时，即设备为安全设备时，携带密钥和nodeId完成设备的接入鉴权。

接入类型	鉴权方式
使用MQTT(S)协议接入的设备	• 使用密钥鉴权： 在设备接入物联网平台前，用户通过应用服务调用**创建设备**接口或通过控制台在物联网平台注册设备，获取设备ID和密钥，并把设备ID和密钥烧录到设备中。对于使用MQTTS协议接入的设备，需要在设备侧预置CA证书；对于使用MQTT非安全协议接入的设备，无需在设备侧预置CA证书。在设备接入物联网平台时携带设备ID和密钥，完成设备的接入鉴权。 • 使用证书鉴权： 在设备接入物联网平台前，用户通过控制台上传设备CA证书，然后应用服务调用**创建设备**接口或通过控制台在物联网平台注册设备，获取设备ID，并把设备ID烧录到设备中。在设备接入物联网平台时携带设备侧X.509证书，完成设备的接入鉴权。 • 使用自定义鉴权： 在设备接入物联网平台前，用户可以通过应用服务调用控制台配置自定义鉴权信息，然后通过调用函数服务（**FunctionGraph入门简介**）配置自定义鉴权函数。在设备接入物联网平台时，物联网平台会获取设备ID和自定义鉴权函数名称等参数，并向FunctionGraph发起鉴权请求，由用户实现鉴权逻辑，完成设备的接入鉴权。 • 使用自定义模板鉴权： 用户可以通过配置自定义鉴权模板，对平台提供的内部函数进行编排，灵活自定义MQTT设备鉴权三元组参数ClientId、Username、Password，实现自定义设备鉴权。

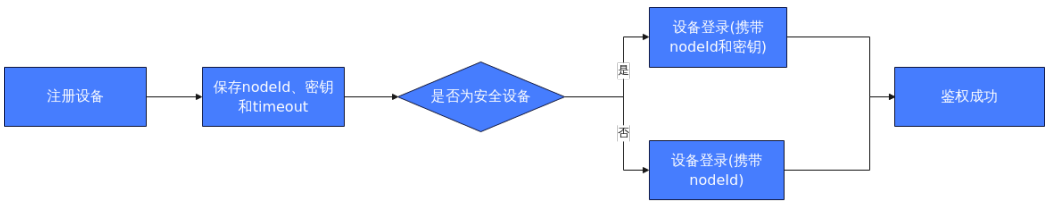
4.2.2 LwM2M/CoAP 协议鉴权

概述

LwM2M/CoAP协议鉴权支持加密与非加密两种接入方式，若设备采用非加密方式接入时，非加密端口为5683，在设备接入物联网平台时携带设备唯一标识nodeId，完成设备的接入鉴权；当设备采用加密方式接入时，加密业务数据交互端口为5684，使用DTLS/DTLS+传输层安全协议通道接入，并携带nodeId和密钥以完成设备的接入鉴权。

使用 LwM2M/CoAP 协议接入的鉴权流程

图 4-4 LwM2M/CoAP 协议接入鉴权流程图



1. 通过调用注册接口向物联网平台发送注册请求或者在控制台上注册设备。
2. 物联网平台向设备分配密钥，返回timeout。

   说明

   - 密钥可以在注册设备时自定义，如果没有定义，平台将自动分配预置密钥。
   - timeout是指超时时间，若设备在有效时间未接入物联网平台，则平台会删除该设备的注册信息。
3. 设备登录时，安全设备携带设备唯一标识码nodeId（如IMEI）和密钥发起接入鉴权请求；非安全设备携带设备唯一标识码nodeId发起接入鉴权请求。
4. 平台验证通过后，返回成功响应，设备连接物联网平台成功。

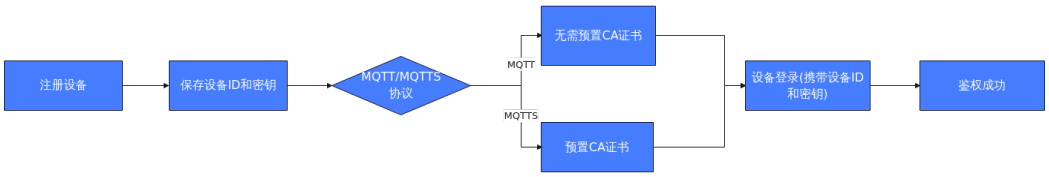
4.2.3 MQTT(S)协议-密钥鉴权

概述

MQTT(S)协议-密钥鉴权是指设备在接入物联网平台时，携带设备ID和密钥以完成设备的接入鉴权。对于使用MQTTS协议接入的设备，需要在设备侧预置CA证书；对于使用MQTT非安全协议接入的设备，无需在设备侧预置CA证书。

使用 MQTT(S)协议-密钥接入的鉴权流程

图 4-5 MQTT(S)协议-密钥接入鉴权流程图



1. 通过调用注册接口向物联网平台发送注册请求或者在控制台上注册设备。

   说明

   注册时需要填写设备标识码，通常使用MAC地址，Serial No或IMEI作为nodeId。
2. 物联网平台向设备分配全局唯一的设备ID（deviceId）和密钥（secret）。

   说明

   密钥可以在注册设备时自定义，如果没有定义，平台将自动分配密钥。
3. 设备侧需集成预置CA证书（仅针对MQTTS协议接入的鉴权流程）。
4. 设备登录时，携带设备ID（deviceId）和密钥（secret）发起接入鉴权请求。
5. 平台验证通过后，返回成功响应，设备连接物联网平台成功。

4.2.4 MQTT(S)协议-证书鉴权

概述

MQTT(S)协议-证书鉴权是指在设备接入物联网平台前，用户通过控制台上传设备CA证书，然后应用服务调用**创建设备**接口或通过控制台在物联网平台注册设备，获取设备ID。在设备接入物联网平台时携带设备侧X.509证书（一种用于通信实体鉴别的数字证书），完成设备的接入鉴权。

约束与限制

- 当前物联网平台只支持基于MQTT协议接入的设备使用X.509证书进行设备身份认证。
- 每个用户最多上传100个设备CA证书。

使用 MQTT(S)协议-证书接入的鉴权流程

图 4-6 MQTT(S)协议-证书接入鉴权流程图

1. 在控制台上传设备CA证书。
2. 通过调用注册接口向物联网平台发送注册请求或者在控制台上注册设备。

   说明

   注册时需要填写设备标识码，通常使用MAC地址，Serial No或IMEI作为nodeId。
3. 物联网平台向设备分配全局唯一的设备ID（deviceId）。
4. 设备登录时，携带设备侧**X.509证书**发起接入鉴权请求。
5. 平台验证通过后，返回成功响应，设备连接物联网平台成功。

相关 API 接口

- **创建设备**
- **重置设备密钥**
- **获取设备CA证书列表**
- **上传设备CA证书**
- **删除设备CA证书**
- **验证设备CA证书**

4.2.5 MQTT(S)协议-自定义鉴权

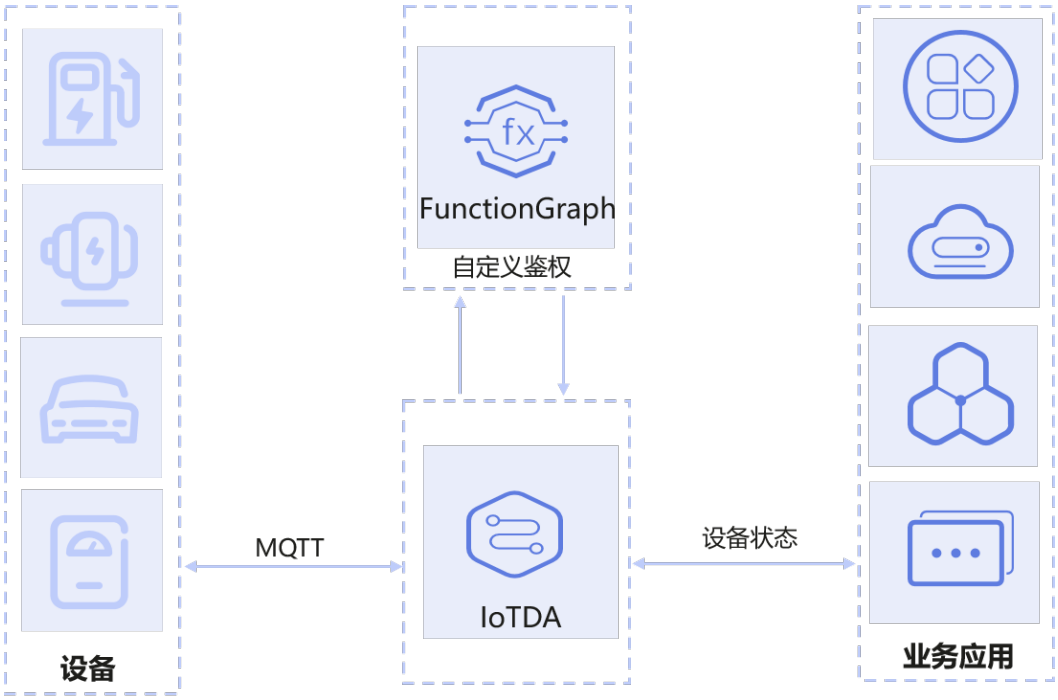
4.2.5.1 自定义鉴权概述

概述

自定义鉴权是指用户可以通过函数服务自定义实现鉴权逻辑，以对接入平台的设备进行身份认证。

在设备接入物联网平台前，用户可以通过应用服务调用控制台配置自定义鉴权信息，然后通过调用函数服务（**FunctionGraph入门简介**）配置自定义鉴权函数。在设备接入物联网平台时，物联网平台会获取设备ID和自定义鉴权函数名称等参数，并向FunctionGraph发起发起鉴权请求，由用户实现鉴权逻辑以完成设备的接入鉴权。

图1 自定义鉴权业务架构图



应用场景

- 迁移场景：当用户的设备从第三方云平台迁移到IoTDA平台时，存量设备可以依据现有的设备鉴权方式，在平台自定义配置设备的鉴权逻辑，以实现设备鉴权方式免改动的能力。
- 原生场景：用户有自定义实现鉴权逻辑的需求，而无需依赖于平台默认的鉴权方式。

约束与限制

- 使用自定义鉴权功能，要求设备必须使用TLS同时支持**SNI(Server Name Indication)**，SNI中需要携带平台分配的域名。
- 每个用户默认最多支持10个自定义鉴权的配置。
- 自定义鉴权的函数最大处理时间为5秒，5秒内函数没返回结果，则认为鉴权失败。
- 每个用户总鉴权请求的TPS限制参考**产品规格说明**，自定义鉴权为总鉴权TPS的50%（不包含设备自注册）。
- 若用户开启了缓存FunctionGraph的鉴权结果，则在相同参数下，函数服务的修改生效时间需在缓存超期后才能生效。
- 在所有的设备接入鉴权方式中，当满足自定义鉴权条件时（匹配到设备携带的自定义鉴权器名称或用户配置了默认自定义鉴权器），优先采用自定义鉴权方式进行设备接入。

4.2.5.2 自定义鉴权使用说明

使用说明

图 4-7 自定义鉴权操作流程

操作步骤

步骤1 配置自定义鉴权函数：用户通过调用函数服务（**FunctionGraph入门简介**）创建自定义鉴权函数。

图 4-8 函数列表-创建函数

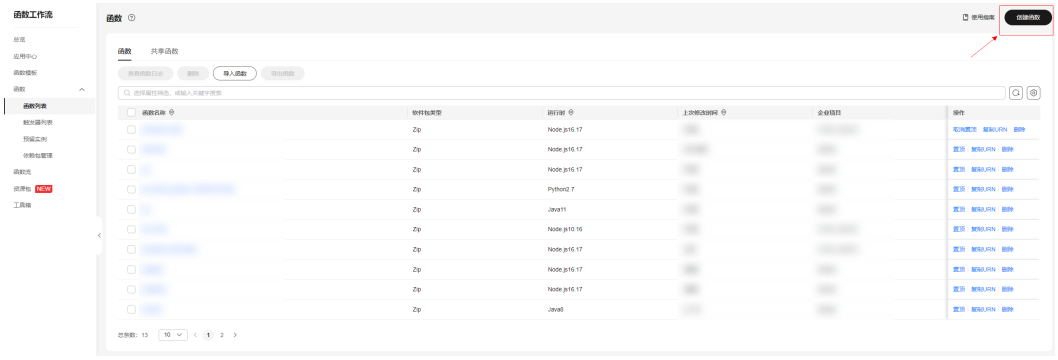
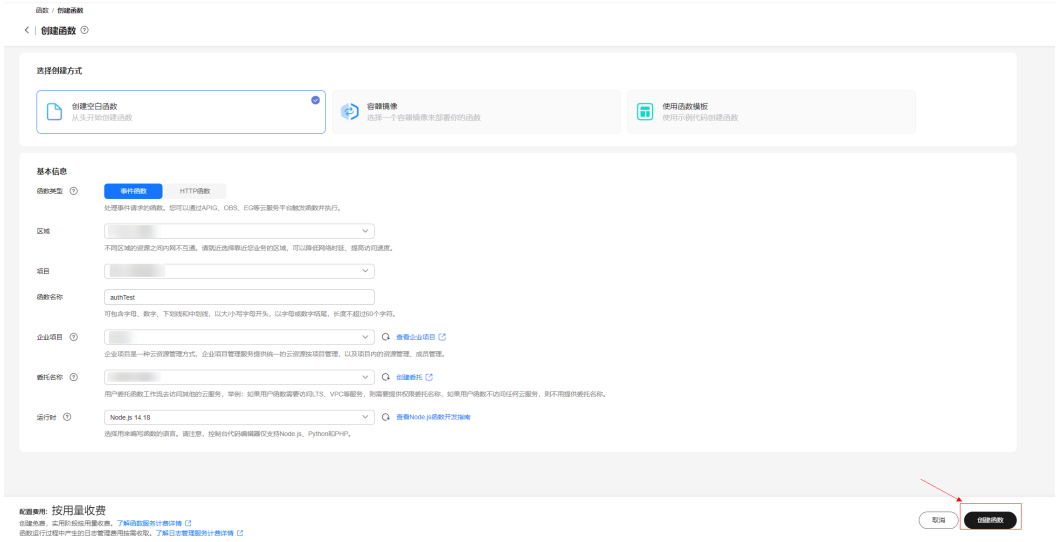


图 4-9 创建函数-参数信息



步骤2 创建自定义鉴权：用户可以通过Console配置自定义鉴权信息，IoTDA负责自定义鉴权信息存库和进行相应的管理维护。用户最多支持配置10个自定义鉴权器，其中最多可以设置1个默认鉴权器。

图 4-10 自定义鉴权-创建鉴权

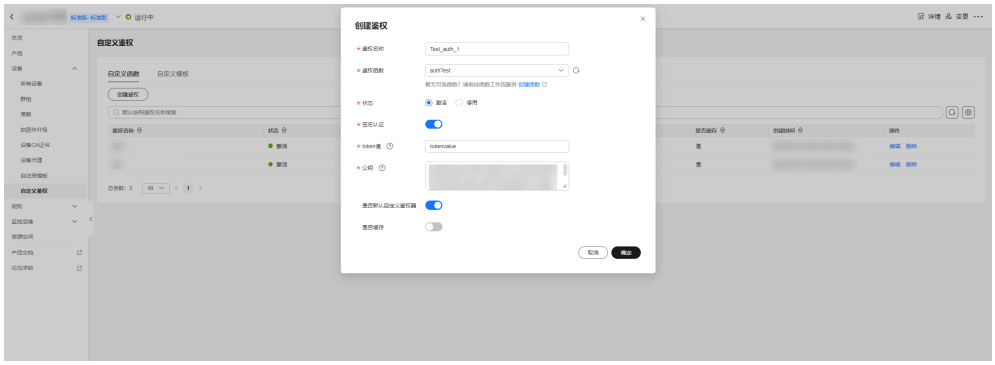


表 4-2 自定义鉴权参数信息

参数名称	是否必选	描述
鉴权名称	是	自定义鉴权器名称。
鉴权函数	是	自定义鉴权器对应的函数名称，从**步骤1**中FunctionGraph已经创建的函数列表中选取。
状态	是	激活停用参数，用于表示该鉴权是否为激活状态，默认为停用状态。若需使用该鉴权器，则应激活后才能生效。
签名认证	是	启动签名认证后，不满足签名要求的鉴权信息将被拒绝，以减少无效的函数调用，默认为开启。
token值	否	签名校验的token值，开启签名校验时使用，用于认证设备携带的签名信息是否正确。
公钥	否	签名校验的公钥，开启签名校验时使用，用于认证设备携带的签名信息是否正确。
是否默认自定义鉴权器	是	开启后，如果设备发起鉴权时的username没有携带authorizer_name参数，则默认使用此鉴权器。此处默认为不开启。
是否缓存	是	缓存开关，用于开启缓存FunctionGraph的鉴权结果，缓存时间为300分钟~1天，默认不开启。

步骤3 设备发起鉴权请求：设备通过MQTT协议发起的CONNECT请求需携带username参数，该参数应包含自定义鉴权的相关可选参数。

- username格式如下。参数传值时需去掉大括号“{}”，username的每个参数需采用分隔符“|”分隔，各参数具体内容不要出现“|”，否则可能导致鉴权失败。

  ```
  {device-identifier}|authorizer-name={authorizer-name}|authorizer-signature={token-signature}|signing-token={token-value}
  示例：
  659b70a0bd3f665a471e5ec9_auth|authorizer-name=Test_auth_1|authorizer-signature=***|signing-token=tokenValue
  ```

表 4-3 username 参数信息

参数名称	是否必选	描述
device-identifier	是	设备标识符，建议设定为设备id。
authorizer-name	否	自定义鉴权器名称，需要同用户配置的鉴权器一致。不携带鉴权器名称时，如用户配置了默认的自定义鉴权器，则使用自定义鉴权，否则使用原有的密钥/证书鉴权方式。
authorizer-signature	否	签名校验功能开启时，需要携带该参数。该参数由私钥和signing-token进行加密后获取。需与**步骤2**中创建自定义鉴权时填写的鉴权名称保持一致。
signing-token	否	签名校验功能开启时，需要携带该参数。该参数为签名校验的token，需与**步骤2**中创建自定义鉴权时填写的token值保持一致。

- authorizer-signature通过如下命令行获取：
  ```
  echo -n {signing-token} | openssl dgst -sha256 -sign {private key} | openssl base64
  ```

表 4-4 命令行参数解析

参数名称	描述
echo -n {signing-token}	使用echo命令将签名令牌（signing-token）的值输出，并使用-n参数去掉末尾的换行符。signing-token需与**步骤2**中创建自定义鉴权时填写的**token值**保持一致。
openssl dgst -sha256 -sign	使用 SHA256 算法对输入的数据进行哈希处理。
{private key}	使用 RSA 算法加密的私钥，可传入.pem/.key格式的私钥文件。
openssl base64	将签名结果进行Base64编码，以便于传输和存储。

步骤4 IoTDA处理鉴权请求：IoTDA收到鉴权请求时，会根据username的参数信息和自定义鉴权的配置来决策是否使用自定义鉴权方式。

1. 判断username是否携带了自定义鉴权名称，如果有携带，则根据平台的自定义鉴权名称，去匹配鉴权器处理函数。如果没有携带，则使用用户配置的默认自定义鉴权器去匹配鉴权处理函数，若无匹配到则使用原有的密钥/证书鉴权方式。
2. 检查用户是否开启了签名校验，如果开启了签名校验，则校验username携带的签名信息是否可以校验成功，校验失败则直接返回鉴权失败。
3. 匹配到处理函数后，携带设备的鉴权信息（即**步骤5**的入参event）并通过函数urn向FunctionGraph发起鉴权请求。

步骤5 用户实现鉴权逻辑：用户在**步骤1**中通过FunctionGraph创建的处理函数中进行开发，函数的返回结果有要求，函数使用示例及返回的JSON格式要求如下：

```
exports.handler = async (event, context) => {
    console.log("username=" + event.username);
    // 此处编写您的校验逻辑


    // 返回的JSON格式（格式固定不变）
    const authRes = {
        "result_code": 200,
        "result_desc": "sucessful",
        "refresh_seconds": 300,
        "device": {
            "device_id": "myDeviceId",
            "provision_enable": true,
            "provisioning_resource": {
                "device_name": "myDeviceName",
                "node_id": "myNodeId",
                "product_id": "myProductId",
                "app_id": "customization0000000000000000000",
                "policy_ids": ["657a4e0c2ea0cb2cd831d12a", "657a4e0c2ea0cb2cd831d12b"]
            }
        }
    }
    return JSON.stringify(authRes);
}
```

其中，函数的请求参数（即event，JSON格式）如下：

```
{
    "username": "myUserName",
    "password": "myPassword",
    "client_id": "myClientId",
    "certificate_info": {
        "common_name": "",
        "fingerprint": "123"
    }
}
```

表 4-5 请求参数信息

参数名称	参数类型	是否必选	描述
username	String	是	MQTT协议中CONNCT消息的username字段，与**步骤3**的username格式相同。
password	String	是	MQTT协议中CONNCT消息的password参数。
client_id	String	是	MQTT协议中CONNCT消息的clientId参数。
certificate_info	JsonObject	否	MQTT协议中CONNCT消息的设备证书信息。

表 4-6 certificate_info 证书信息

参数名称	参数类型	是否必选	描述
common_name	String	是	设备携带设备证书时，从设备证书中解析的commonName信息。

参数名称	参数类型	是否必选	描述
fingerprint	String	是	设备携带设备证书时，从设备证书中解析的指纹信息。

表 4-7 返回参数信息

参数名称	参数类型	是否必选	描述
result_code	Integer	是	鉴权结果码， 返回200标识鉴权成功。
result_desc	String	否	鉴权结果描述信息。
refresh_seconds	Integer	否	鉴权结果的缓存时间，单位(s)。
device	JsonObject	否	认证成功时的设备信息。当设备信息中的设备ID不存在时，如果用户开启自注册，则平台会根据设备信息自动创建对应的设备。

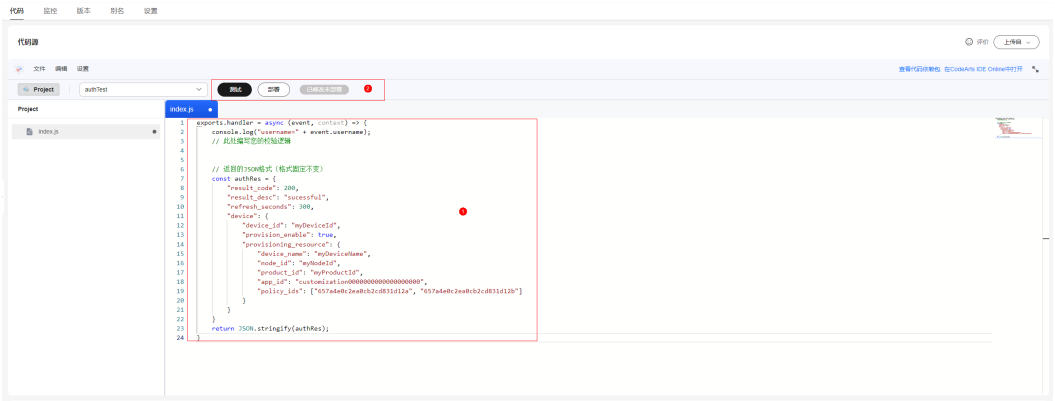
表 4-8 device 设备信息

参数名称	参数类型	是否必选	描述
device_id	String	是	**参数说明：**设备ID，自注册场景、非自注册场景都必选。全局唯一，用于唯一标识一个设备。如果携带该参数，平台将设备ID设置为该参数值；建议使用product_id + _ + node_id拼接而成。 **取值范围：**长度不超过128，只允许字母、数字、下划线（ _ ）、连接符（ - ）的组合，建议不少于4个字符。
provision_enable	Boolean	否	**参数说明：**自注册开关，默认为false。如果设备不存在，平台将触发设备的自动注册。
provisioning_resource	JsonObject	自注册场景必选	**参数说明：**自注册参数信息。

表 4-9 provisioning_resource 自注册参数信息

参数名称	参数类型	是否必选	描述
device_name	String	否	**参数说明：**设备名称，资源空间下唯一，用于资源空间下唯一标识一个设备。 **取值范围：**长度不超过256，只允许中文、字母、数字、以及_?'#().,&%@!-等字符的组合，建议不少于4个字符。 最小长度：**1** 最大长度：**256**
node_id	String	是	**参数说明：**设备标识码，通常使用IMEI、MAC地址或Serial No作为node_id。 设备标识码长度为1到64个字符，包含英文字母、数字、连接号-和下划线_。 注意：NB设备由于模组烧录信息后无法配置，所以NB设备会校验node_id全局唯一。 **取值范围：**长度不超过64，只允许字母、数字、下划线（ _ ）、连接符（ - ）的组合，建议不少于4个字符。
product_id	String	是	**参数说明：**设备关联的产品ID，用于唯一标识一个产品模型，创建产品后获得。取值范围：长度不超过256，只允许中文、字母、数字、以及_?'#().,&%@!-等字符的组合，建议不少于4个字符。 最小长度：**1** 最大长度：**256**
app_id	String	是	**参数说明：**资源空间ID，该参数指定创建的设备归属到哪个资源空间。 **取值范围：**长度不超过36，只允许字母、数字、下划线（ _ ）、连接符（ - ）的组合。
policy_ids	List<String>	否	**参数说明：**Topic策略ID。

图 4-11 函数编写-部署



步骤6 FunctionGraph返回结果后，判断是否需要支持自注册功能，如果需要则触发设备的自动注册。自注册设备全部默认为密钥认证，密钥随机生成。IoTDA收到鉴权结果后，进行接下来的业务实现流程。

----结束

4.2.6 MQTT(S)协议-自定义模板鉴权

4.2.6.1 自定义模板鉴权概述

概述

自定义模板鉴权是指用户可以使用平台提供的**内部函数**实现自定义的鉴权方式，而不需使用**平台默认鉴权方式**，对接入平台的设备进行身份认证，用户可以通过平台预置的函数灵活编排鉴权方式。

应用场景

- 迁移场景：当设备从第三方物联网平台迁移到华为云IoTDA时，通过配置平台提供的自定义鉴权模板可以兼容原来的鉴权方式，设备侧无需改动实现无缝迁移。
- 原生场景：自定义模板鉴权可以让用户扩展自己的设备鉴权方式，而不需要与平台默认鉴权绑定，极大的提高了设备可扩展性。

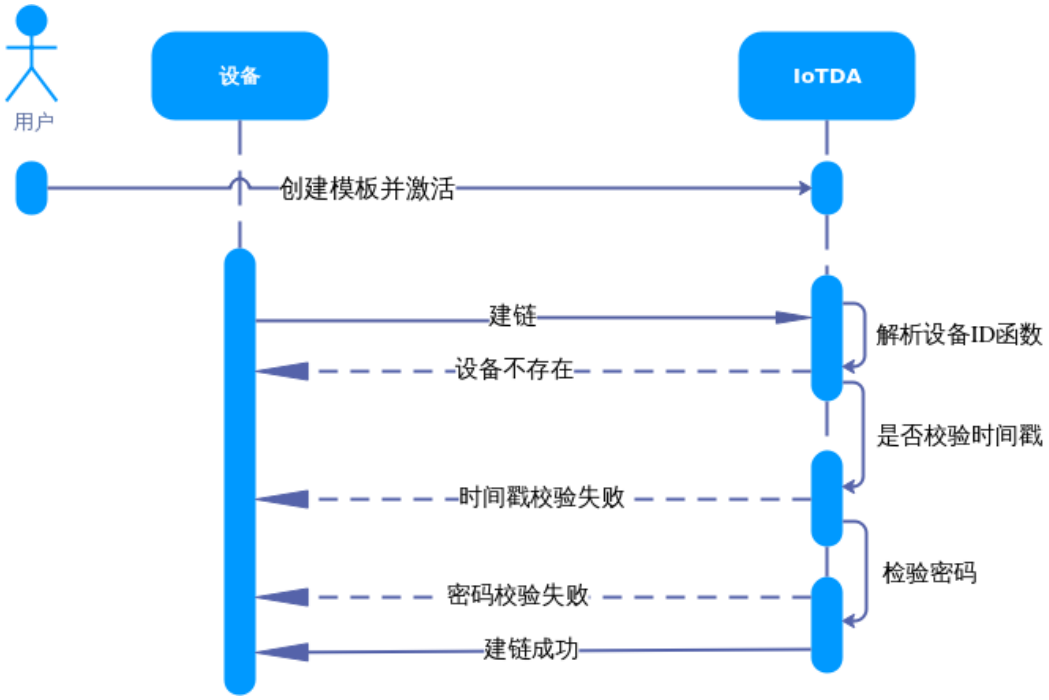
约束与限制

1. 使用自定义鉴权功能，要求设备必须使用TLS同时支持**SNI(Server Name Indication)**，SNI中需要携带平台分配的域名。
2. 默认每个用户最多支持5个自定义鉴权模板，只能启用一个激活状态的模板。
3. 鉴权模板函数嵌套最大深度为5层。
4. 模板内容体最大长度不能超过4000字符，且不能包含中文字符。
5. 设备为密钥认证类型时，模板密码函数必须包含设备原始密钥参数(iotda::device::secret)。
6. 使用模板鉴权时，鉴权参数username不能与自定义函数鉴权username格式重叠，否则会使用自定义函数鉴权，比如：
   ```
   {deviceId}|authorizer-name={authorizer-name}|xxx
   ```
7. 自定义模板鉴权优先级高于平台默认鉴权，即激活自定义鉴权模板后设备就会使用模板鉴权，不会再使用平台默认鉴权方式。

4.2.6.2 自定义模板鉴权使用说明

使用说明

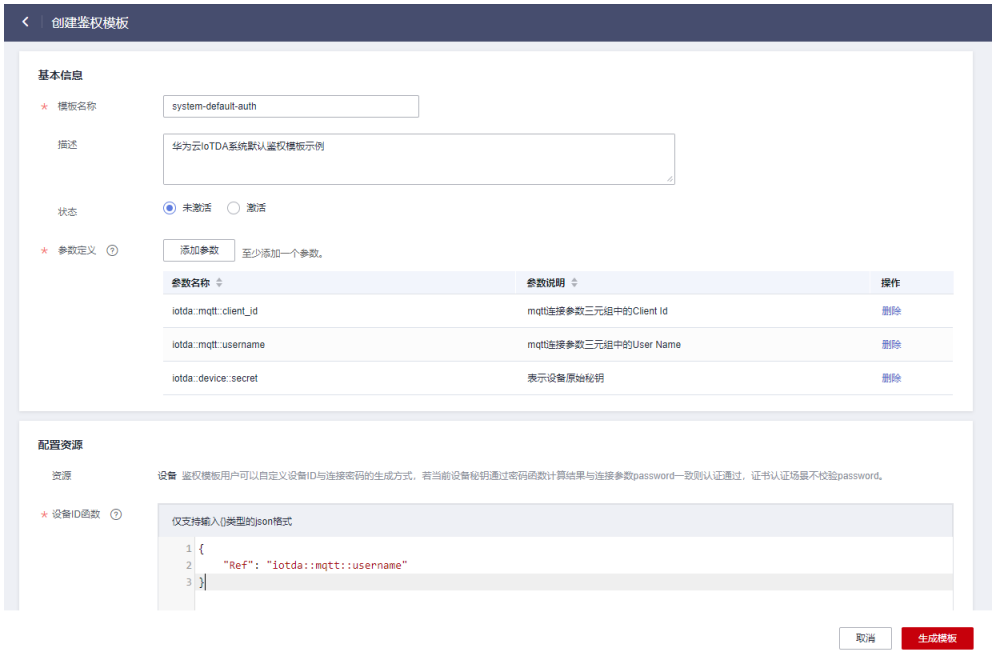
图 4-12 自定义模板鉴权流程图



操作步骤

步骤1 创建鉴权模板：进入设备接入控制台左侧导航栏，选择“设备 > 自定义鉴权”，单击“自定义模板”，单击“创建鉴权模板”。本示例演示使用的鉴权模板与**系统默认鉴权**一致。

图 4-13 自定义鉴权-创建鉴权模板



模板整体内容如下：

```
{
  "template_name": "system-default-auth",
  "description": "华为云IoTDA系统默认鉴权模板示例",
  "status": "ACTIVE",
  "template_body": {
    "parameters": {
      "iotda::mqtt::client_id": {
        "type": "String"
      },
      "iotda::mqtt::username": {
        "type": "String"
      },
      "iotda::device::secret": {
        "type": "String"
      }
    },
    "resources": {
      "device_id": {
        "Ref": "iotda::mqtt::username"
      },
      "timestamp": {
        "type": "FORMAT",
        "pattern": "yyyyMMddHH",
        "value": {
          "Fn::SubStringAfter": [
            "${iotda::mqtt::client_id}",
            "_0_1_"
          ]
        }
      },
      "password": {
        "Fn::HmacSHA256": [
          "${iotda::device::secret}",
          {
            "Fn::SubStringAfter": [
              "${iotda::mqtt::client_id}",
              "_0_1_"
            ]
```

```
      }
    ]
   }
  }
 }
}
```

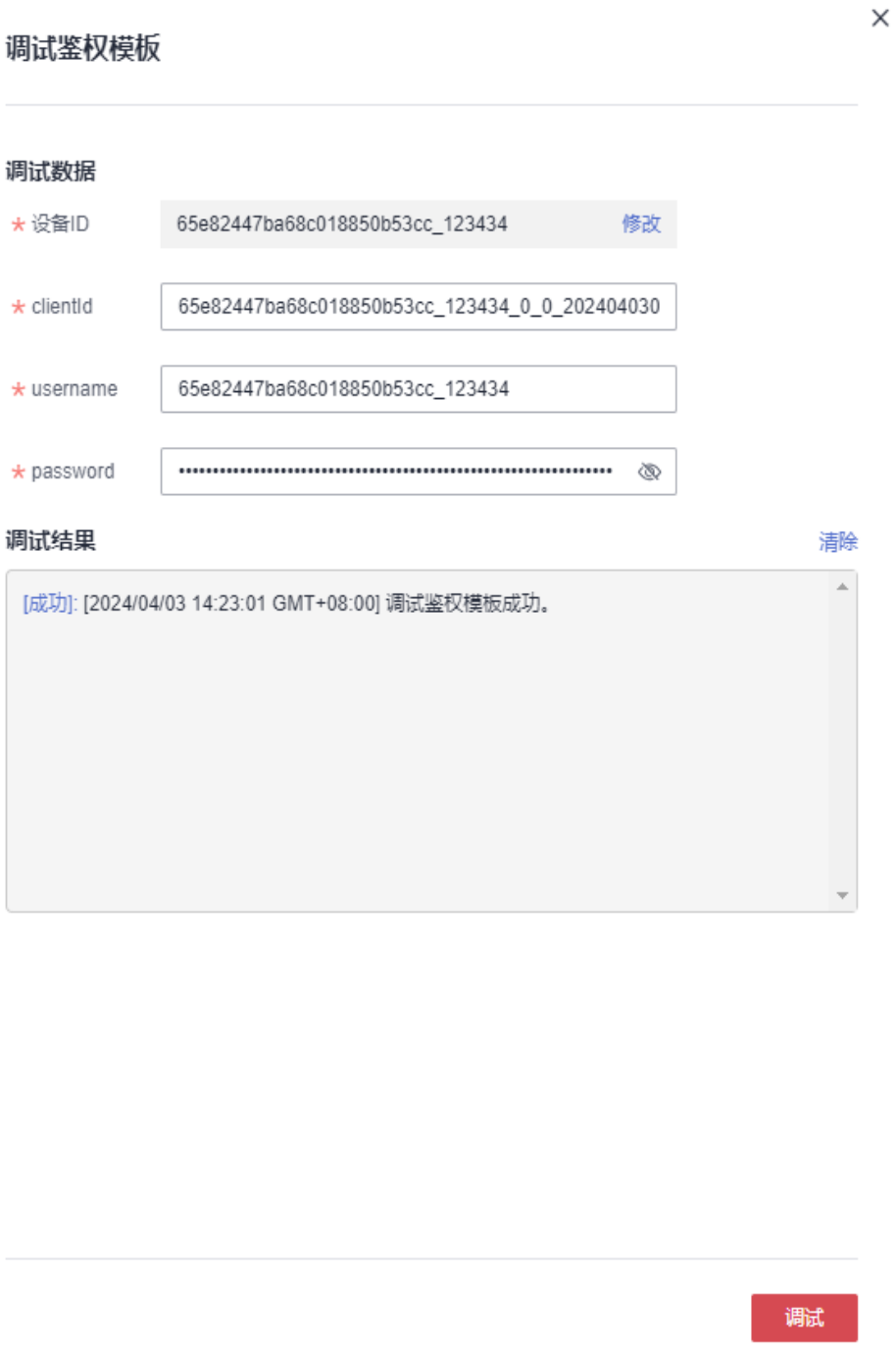
表 4-10 鉴权模板参数信息

参数	参数名称	是否必填	描述
template_name	模板名称	是	鉴权模板名称，单个用户下模板名称不能重复，长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
description	描述	否	鉴权模板的描述信息，长度不超过2048，只允许中文、字母、数字、以及_?'#().,&%@!-等字符的组合。
status	状态	否	是否激活该模板，默认状态为未激活，一个用户下只能有一个已激活状态的模板。
parameters	参数	是	平台预定义的MQTT连接参数列表，当设备使用密码认证时模板必须包含设备原始密钥参数（iotda::device::secret）。 平台预定义了如下参数： iotda::mqtt::client_id： mqtt连接参数三元组中的Client Id iotda::mqtt::username： mqtt连接参数三元组中的User Name iotda::certificate::country : 设备证书（国家/地区,C ） iotda::certificate::organization : 设备证书（组织,O ） iotda::certificate::organizational_unit : 设备证书（组织单位,OU ） iotda::certificate::distinguished_name_qualifier : 设备证书（可辨别名称限定符,dnQualifier ） iotda::certificate::state_name : 设备证书（省市,ST ） iotda::certificate::common_name : 设备证书（公用名,CN ） iotda::certificate::serial_number : 设备证书（序列号,serialNumber ） iotda::device::secret： 表示设备原始密钥
device_id	设备ID函数	是	设备ID取值函数，JSON格式，平台通过解析该函数获取对应设备信息。

参数	参数名称	是否必填	描述
timestamp	是否开启时间戳校验	否	是否校验设备连接信息中的时间戳，如果设备连接参数（clientId、username）中包含时间戳建议开启校验。开启校验平台会对比设备携带时间戳与平台系统时间，若设备时间戳加一小时小于平台系统时间则校验失败。
type	时间戳类型	否	UNIX：表示时间戳格式为Unix时间戳，长整型，单位秒。 FORMAT：格式化时间戳类型，比如：2024-03-28 11:47:39、2024/03/28 03:49:13
pattern	时间戳格式	否	时间格式模板，时间戳类型为FORMAT时必填，具体字符含义如下： y：年 M：月 d：日 H：时 m：分 s：秒 S：毫秒 示例：yyyy-MM-dd HH:mm:ss、yyyy/MM/dd HH:mm:ss
value	时间戳取值函数	否	开启时间戳校验后必填，设备时间戳获取函数，平台通过执行该函数获取设备建链时的时间戳。
password	MQTT密码函数	否	密码函数，当设备认证类型为密钥认证时必填，且模板参数中必须包含设备原始密钥参数（iotda::device::secret），当设备为证书认证是可不填。设备认证类型参考**注册设备**。平台把设备原始密钥等参数填入函数计算结果，若函数结果与设备建链携带的password参数一致则认证通过，否则认证失败。

步骤2 选择设备调试模板：单击“调试”选择一个设备进行调试，输入mqtt连接参数后单击“调试”查看调试结果。注意：使用系统标准格式的clientId平台会校验username参数与clientId前缀一致。

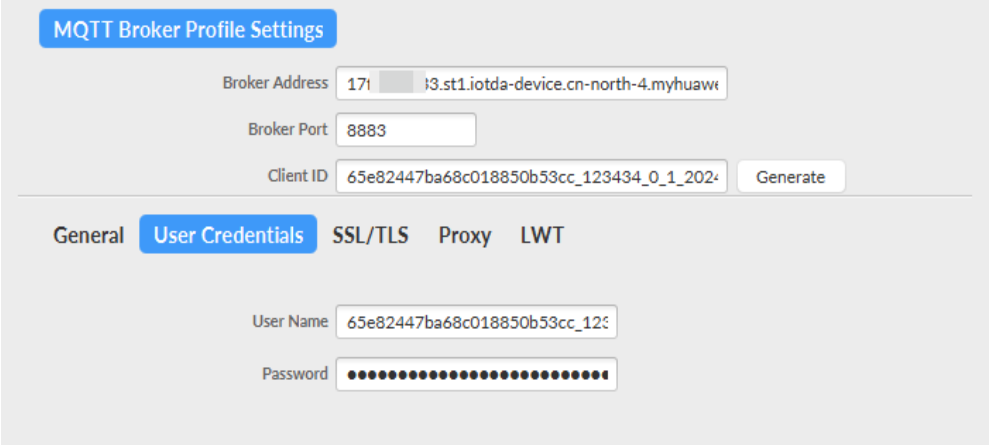
图 4-14 自定义模板-调试



设备调试成功后单击“激活”启用模板，一旦激活模板所有设备鉴权将使用该模板，且已激活状态下的模板不能修改，后续修改模板建议新创建一个副本模板进行调试，确认无误后进行模板切换。

步骤3 使用mqtt.fx工具模拟设备真实建链结果，Broker Address填**平台接入地址**，选择“总览 > 接入信息”，端口使用8883端口。

图 4-15 设备建链



----结束

4.2.6.3 自定义模板示例

示例 1

证书认证设备，不限制UserName与ClientId参数取值，从设备证书通用名称（Common Name）中取值设备ID。

表 4-11 鉴权参数

参数	说明
Client ID	任意值
User Name	任意值
Password	空值

鉴权模板：

```
{
   "template_name": "template1",
   "description": "template1",
   "template_body": {
      "parameters": {
         "iotda::certificate::common_name": {
            "type": "String"
         }
      },
      "resources": {
         "device_id": {
            "Ref": "iotda::certificate::common_name"
         }
      }
   }
}
```

示例 2

设备ID格式为：${ProductId}_${NodeId}。

表 4-12 鉴权参数

参数	说明
Client ID	固定格式： **${ClientId}\|securemode=2,signmethod=hmacsha256\|timestamp=${timestamp}|** ● *${ClientId}：* 固定格式 ${ProductId}.${NodeId}。 – *${NodeId}：* 设备标识码。 – *${ProductId}*：产品ID。 ● *${timestamp}：* Unix时间戳，毫秒。
User Name	由设备标识码和产品ID组成，固定格式： **${NodeId}&${ProductId}**
Password	以设备密码为密钥，将设备参数与参数值拼接后的字符串为加密串进行hmacsha256算法加密。 加密串格式： **clientId${clientId}deviceName${nodeId}productKey${productId}timestamp${timestamp}** ● *${ClientId}：* 固定格式 ${ProductId}.${NodeId}。 ● *${NodeId}：* 设备标识码。 ● *${ProductId}*：产品ID。 ● *${timestamp}：* 时间戳。

鉴权模板：

```
{
   "template_name": "template2",
   "description": "template2",
   "template_body": {
      "parameters": {
         "iotda::mqtt::client_id": {
            "type": "String"
         },
         "iotda::mqtt::username": {
            "type": "String"
         },
         "iotda::device::secret": {
            "type": "String"
         }
      },
      "resources": {
         "device_id": {
            "Fn::Join": [{
               "Fn::SplitSelect": [
                  "${iotda::mqtt::username}",
                  "&",
                  1
               ]
            }, "_", {
               "Fn::SplitSelect": [
                  "${iotda::mqtt::username}",
                  "&",
```

```
                    0
                ]
            }]
        },
        "timestamp": {
            "type": "UNIX",
            "value": {
                "Fn::MathDiv": [{
                    "Fn::ParseLong": {
                        "Fn::SplitSelect": [{
                            "Fn::SplitSelect": ["${iotda::mqtt::client_id}", "|", 2]
                        }, "=", 1]
                    }
                }, 1000]
            }
        },
        "password": {
            "Fn::HmacSHA256": [{
                    "Fn::Sub": [
                        "clientId${clientId}deviceName${deviceName}productKey${productKey}timestamp$
{timestamp}",
                        {
                            "clientId": {
                                "Fn::SplitSelect": [
                                    "${iotda::mqtt::client_id}",
                                    "|",
                                    0
                                ]
                            },
                            "deviceName": {
                                "Fn::SplitSelect": [
                                    "${iotda::mqtt::username}",
                                    "&",
                                    0
                                ]
                            },
                            "productKey": {
                                "Fn::SplitSelect": [
                                    "${iotda::mqtt::username}",
                                    "&",
                                    1
                                ]
                            },
                            "timestamp": {
                                "Fn::SplitSelect": [{
                                    "Fn::SplitSelect": ["${iotda::mqtt::client_id}", "|", 2]
                                }, "=", 1]
                            }
                        }
                    ]
                },
                "${iotda::device::secret}"
            ]
        }
    }
  }
}
```

示例 3

设备ID格式为：${productId}${nodeId}。

表 4-13 鉴权参数

参数	说明
Client ID	固定格式： **${productId}${nodeId}** ● *${productId}*：产品ID。 ● *${nodeId}：* 设备标识码。
User Name	固定格式： **${productId}${nodeId};12010126;${connid};${expiry}** ● *${productId}：* 产品ID。 ● *${nodeId}：* 设备标识码。 ● *${connid}：*一个随机字符串。 ● *${expiry}：Unix时间戳，单位秒。*
Password	固定格式： **${token};hmacsha256** ● *${token}：*以BASE64解码后的设备密码为密钥，对User Name字段进行hmacsha256算法加密后的值。

鉴权模板：

```
{
   "template_name": "template3",
   "description": "template3",
   "template_body": {
      "parameters": {
         "iotda::mqtt::client_id": {
            "type": "String"
         },
         "iotda::mqtt::username": {
            "type": "String"
         },
         "iotda::device::secret": {
            "type": "String"
         }
      },
      "resources": {
         "device_id": {
            "Ref": "iotda::mqtt::client_id"
         },
         "timestamp": {
            "type": "UNIX",
            "value": {
               "Fn::ParseLong": {
                  "Fn::SplitSelect": ["${iotda::mqtt::username}", ";", 3]
               }
            }
         },
         "password": {
            "Fn::Sub": [
               "${token};hmacsha256",
               {
                  "token": {
                     "Fn::HmacSHA256": [
                        "${iotda::mqtt::username}",
                        {
                           "Fn::Base64Decode": "${iotda::device::secret}"
                        }
```

```
                ]
              }
            }
          ]
        }
      }
    }
  }
}
```

4.2.6.4 内部函数

使用说明

华为云IoTDA提供了多个内部函数供用户在模板中使用，使用时请认真阅读每个函数的功能定义，包括入参类型，参数长度，返回值类型等。

说明

- 整个函数必须是合法的Json格式。
- 函数中可使用${}变量占位符或者"Ref"函数引用入参定义的参数值。
- 函数所使用的参数必须在模板参数中声明。
- 单一入参的函数后面直接跟参数，比如："Fn::Base64Decode": "${iotda::mqtt::username}"。
- 多个入参的函数后面接数组格式，比如："Fn::HmacSHA256": ["${iotda::mqtt::username}", "${iotda::device::secret}"]。
- 函数可以嵌套使用，即一个函数的参数可以是另一个函数，注意嵌套函数的返回值必须跟当前函数参数类型一致，比如：{"Fn::HmacSHA256": ["${iotda::mqtt::username}", {"Fn::Base64Encode": "${iotda::device::secret}"}]}

Fn::ArraySelect

内部函数Fn::ArraySelect返回一个字符串数组中索引为index的字符串元素。

JSON

```
{"Fn::ArraySelect": [index, [StringArray]]}
```

表 4-14 参数说明

参数名称	类型	说明
index	int	整型，数组元素索引值，从0开始计算。
StringArray	String[]	字符串数组元素。
返回值	String	索引为index的元素。

示例如下：

```
{
    "Fn::ArraySelect": [1, ["123", "456", "789"]]
}
return: "456"
```

Fn::Base64Decode

内部函数Fn::Base64Decode将一个字符串按BASE64解码成一个字节数组。

JSON

```
{ "Fn::Base64Decode" : "content" }
```

表 4-15 参数说明

参数名称	类型	说明
content	String	待解码的字符串。
返回值	byte[]	base64解码后的字节数组。

示例如下：

```
{
    "Fn::Base64Decode": "123456"]
}
return: d76df8e7 //为了方便展示，此处转化为16进制字符串
```

Fn::Base64Encode

内部函数Fn::Base64Encode将一个字符串按BASE64编码。

JSON

```
{"Fn::Base64Encode": "content"}
```

表 4-16 参数说明

参数名称	类型	说明
content	String	待编码的字符串。
返回值	String	base64编码后的字符串。

示例如下：

```
{
    "Fn::Base64Encode": "testvalue"
}
return: "dGVzdHZhbHVl"
```

Fn::GetBytes

内部函数Fn::GetBytes返回一个字符串UTF-8编码的字节数组。

JSON

```
{"Fn::GetBytes": "content"}
```

表 4-17 参数说明

参数名称	类型	说明
content	String	待编码的字符串。
返回值	byte[]	字符串UTF-8编码后的字节数组。

示例如下：

```
{
    "Fn::GetBytes": "testvalue"
}
return: "7465737476616c7565" //为了方便展示，此处转化为16进制字符串
```

Fn::HmacSHA256

内部函数Fn::HmacSHA256将一个字符串按给定密钥进行HmacSHA256算法加密。

JSON

```
{"Fn::HmacSHA256": ["content", "secret"]}
```

表 4-18 参数说明

参数名称	类型	说明
content	String	待加密的字符串。
secret	String 或 byte[]	加密密钥，可以是字符串或者字节数组类型
返回值	String	使用HmacSHA256算法加密后的值。

示例如下：

```
{
    "Fn::HmacSHA256": ["testvalue", "123456"]
}
return: "0f9fb47bd47449b6ffac1be951a5c18a7eff694940b1a075b973ff9054a08be3"
```

Fn::Join

内部函数Fn::Join可将多个字符串（数量最大值为10）拼接成一个字符串。

JSON

```
{"Fn::Join": ["element", "element"...]}
```

表 4-19 参数说明

参数名称	类型	说明
element	String	需拼接的字符串。

参数名称	类型	说明
返回值	String	子字符串拼接在一起后的字符串。

示例如下：

```
{
    "Fn::Join": ["123", "456", "789"]
}
return: "123456789"
```

Fn::MathAdd

内部函数Fn::MathAdd将两个整数进行数学加法运算。

JSON

```
{"Fn::MathAdd": [X, Y]}
```

表 4-20 参数说明

参数名称	类型	说明
X	long	加数。
Y	long	加数。
返回值	long	和，X+Y后的值。

示例如下：

```
{
    "Fn::MathAdd": [1, 1]
}
return: 2
```

Fn::MathDiv

内部函数Fn::MathDiv将两个整数进行数学除法运算。

JSON

```
{"Fn::MathDiv": [X, Y]}
```

表 4-21 参数说明

参数名称	类型	说明
X	long	被除数。
Y	long	除数。
返回值	long	X 除Y后的值。

示例如下：

```
{
    "Fn::MathDiv": [10, 2]
}
return: 5

{
    "Fn::MathDiv": [10, 3]
}
return: 3
```

Fn::MathMod

内部函数Fn::MathMod将两个整数进行数学取余运算。

JSON

```
{"Fn::MathMod": [X, Y]}
```

表 4-22 参数说明

参数名称	类型	说明
X	long	被取余数。
Y	long	取余数。
返回值	long	X 取余 Y后的值。

示例如下：

```
{
    "Fn::MathMod": [10, 3]
}
return: 1
```

Fn::MathMultiply

内部函数Fn::MathMultiply将两个整数进行数学乘法运算。

JSON

```
{"Fn::MathMultiply": [X, Y]}
```

表 4-23 参数说明

参数名称	类型	说明
X	long	乘数。
Y	long	乘数。
返回值	long	X 乘以 Y后的值。

示例如下：

```
{
    "Fn::MathMultiply": [3, 3]
}
return: 9
```

Fn::MathSub

内部函数Fn::MathSub将两个整数进行数学减法运算。

JSON

```
{"Fn::MathSub": [X, Y]}
```

表 4-24 参数说明

参数名称	类型	说明
X	long	被减数。
Y	long	减数。
返回值	long	X 减 Y后的值。

示例如下：

```
{
    "Fn::MathSub": [9, 3]
}
return: 6
```

Fn::ParseLong

内部函数Fn::ParseLong可将一个数字字符串转化为整数。

JSON

```
{"Fn::ParseLong": "String"}
```

表 4-25 参数说明

参数名称	类型	说明
String	String	待转换的字符串。
返回值	long	字符串转换为数字后的值。

示例如下：

```
{
    "Fn::ParseLong": "123"
}
return: 123
```

Fn::Split

内部函数Fn::Split将一个字符串按指定的分隔符分割成字符串数组。

JSON

```
{ "Fn::Split" : ["String", "Separator"] }
```

表 4-26 参数说明

参数名称	类型	说明
String	String	被分割的字符串。
Separator	String	分隔符。
返回值	String[]	原始参数String被分隔符Separator拆分后的字符串数组。

示例如下：

```
{
    "Fn::Split": ["a|b|c", "|"]
}
return: ["a", "b", "c"]
```

Fn::SplitSelect

内部函数Fn::SplitSelect将一个字符串按指定的分隔符分割成字符串数组，然后返回数组指定索引的元素。

JSON

```
{ "Fn::SplitSelect" : ["String", "Separator", index] }
```

表 4-27 参数说明

参数名称	类型	说明
String	String	被分割的字符串。
Separator	String	分隔符。
index	int	返回元素在数组中的索引值，从0开始。
返回值	String	字符串按特定分隔符分割后指定索引的子字符串。

示例如下：

```
{
    "Fn::SplitSelect": ["a|b|c", "|", 1]
}
return: "b"
```

Fn::Sub

内部函数Fn::Sub将输入字符串中的变量替换为指定的值。在模板中你可以使用此函数来构造一个动态的字符串。

JSON

```
{ "Fn::Sub" : [ "String", { "Var1Name": Var1Value, "Var2Name": Var2Value } ] }
```

表 4-28 参数说明

参数名称	类型	说明
String	String	一个包含变量的字符串，变量使用“${}”占位符定义。
VarName	String	变量名称，必须在参数“String”中定义。
VarValue	String	变量的取值，支持函数嵌套。
返回值	String	返回原始“String”参数字符串变量替换后的值。

示例如下：

```
{
   "Fn::Sub": ["${token};hmacsha256", {
      "token": {
         "Fn::HmacSHA256": ["${iotda::mqtt::username}", {
            "Fn::Base64Decode": "${iotda::mqtt::client_id}"
         }]
      }
   }]
}
当变量
${iotda::mqtt::username}="test_device_username"
${iotda::device::client_id}="OozqTPlCWTTJjEH/5s+T6w=="
return：  "0773c4fd6c92902a1b2f4a45fdcdec416b6fc2bc6585200b496e460e2ef31c3d"
```

Fn::SubStringAfter

内部函数Fn::SubStringAfter截取字符串指定分隔符后的子字符串。

JSON

```
{ "Fn::SubStringAfter" : ["content", "separator"] }
```

表 4-29 参数说明

参数名称	类型	说明
content	String	待截取的字符串。
separator	String	分隔符。
返回值	String	字符串被指定分隔符分割后的子字符串。

示例如下：

```
{
   "Fn::SubStringAfter": ["content:123456", ":"]
```

```
]
return: "123456"
```

Fn::SubStringBefore

内部函数Fn::SubStringBefore截取字符串指定分隔符前的子字符串。

JSON

```
{ "Fn::SubStringBefore" : ["content", "separator"] }
```

表 4-30 参数说明

参数名称	类型	说明
content	String	待截取的字符串。
separator	String	分隔符。
返回值	String	字符串被指定分隔符分割前的子字符串。

示例如下：

```
{
    "Fn::SubStringBefore": ["content:123456", ":"]
]
return: "content"
```

Ref

内部函数Ref将返回指定引用参数的值，引用参数必须在模板中有声明。

JSON

```
{ "Ref" : "paramName" }
```

表 4-31 参数说明

参数名称	类型	说明
paramName	String	引用的参数名称。
返回值	String	引用参数对应的值。

示例如下：

```
{
    "Ref": "iotda::mqtt::username"
}
当参数iotda::mqtt::username="device_123"
return: "device_123"
```

4.3 开放协议接入

4.3.1 LwM2M/CoAP 协议接入

概述

LwM2M（Lightweight M2M，轻量级M2M），由开发移动联盟（OMA）提出，是一种轻量级的、标准通用的物联网设备管理协议，可用于快速部署客户端/服务器模式的物联网业务。LwM2M为物联网设备的管理和应用建立了一套标准，它提供了轻便小巧的安全通信接口及高效的数据模型，以实现M2M设备管理和服务支持。物联网平台支持加密与非加密两种接入设备接入方式，其中加密业务数据交互端口为5684端口，采用DTLS+CoAP协议通道接入，非加密端口为5683，接入协议为CoAP。物联网平台从安全角度考虑，强烈建议采用安全接入方式。

说明

LwM2M的语法和接口细节，请以此**标准规范**为准。

物联网平台支持协议规定的plain text, opaque, Core Link ,TLV , JSON编码格式。在多字段操作时（比如写多个资源），默认用TLV格式。

使用限制

表 4-32 使用限制

描述	限制
支持的LwM2M协议版本	1.1
支持的DTLS版本	DTLS 1.2
支持的加密算法套件	TLS_PSK_WITH_AES_128_CCM_8， TLS_PSK_WITH_AES_128_CBC_SHA256
支持的body体最大长度	1KB
接口规格说明	请参考**产品规格说明**。

调用说明

物联网平台的Endpoint请参见：**地区和终端节点**。

说明

使用“设备接入-> CoAP (5683)| CoAPS (5684)”对应的Endpoint，端口为5683（非加密接入方式）或者5684（加密接入方式）。

4.3.2 HTTPS 协议接入

概述

HTTPS是基于HTTP协议，通过SSL加密的一种安全通信协议。物联网平台支持HTTPS协议通信。

使用限制

描述	限制
支持的HTTP协议版本	支持 Hypertext Transfer Protocol — HTTP/1.0 协议 支持 Hypertext Transfer Protocol — HTTP/1.1 协议
支持HTTPS协议	物联网平台仅支持HTTPS协议，证书下载请参考**证书资源**。
支持的TLS版本	TLS 1.2
支持的body体最大长度	1MB
接口规格说明	请参考**产品规格说明**。
网关上报子设备属性时一次最大可上报子设备数	50

调用说明

物联网平台的Endpoint请参见：**地区和终端节点**。

说明

使用“设备接入-> HTTPS(443)”对应的Endpoint，端口为443。

HTTPS 设备与物联网平台通信

设备使用HTTPS协议接入平台时，平台和设备通过HTTPS接口调用通信。通过这些接口，平台和设备可以实现设备鉴权、消息上报及属性上报。

消息类型	说明
设备鉴权	用于设备获取鉴权信息access_token。
设备属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。
设备消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
网关批量属性上报	用于网关设备将多个子设备的属性数据一次性上报给平台。

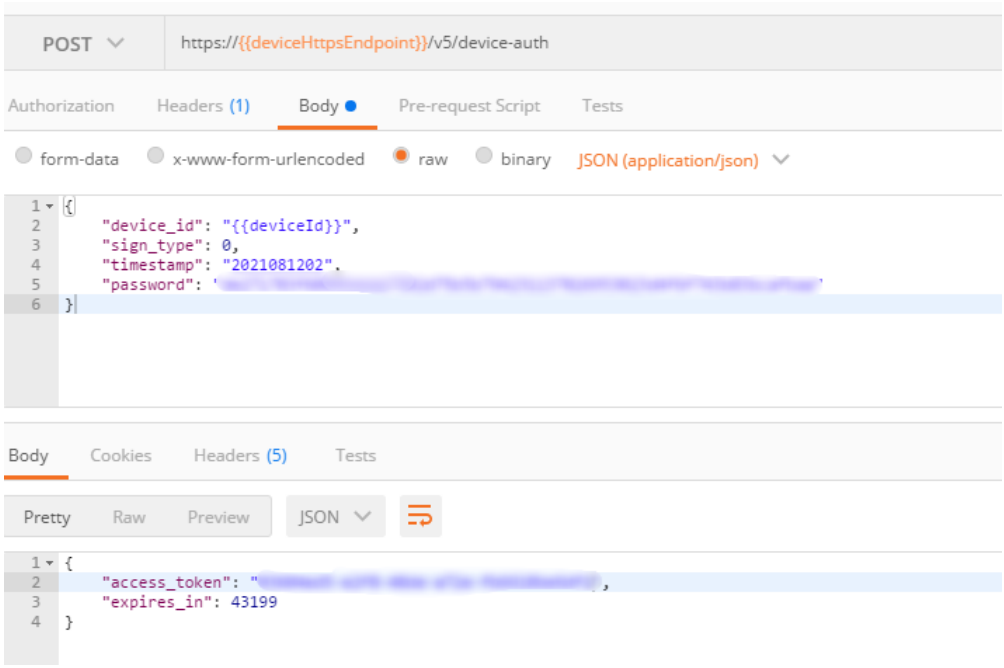
业务流程

步骤1 设备接入前，需创建产品（可通过控制台创建或者使用应用侧API**创建产品**）。

步骤2 产品创建完毕后，需注册设备（可通过控制台**注册单个设备**或者使用应用侧API**注册设备**创建）。

步骤3 设备注册完毕后，通过设备鉴权接口获取设备的access_token。

图 4-16 获取设备 access_token



步骤4 获取到access_token之后，可以使用消息/属性上报等功能。其中access_token放于消息头中，下面示例为上报属性：

图 4-17 上报属性

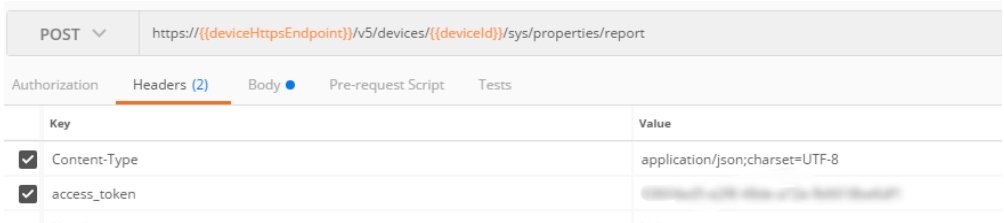
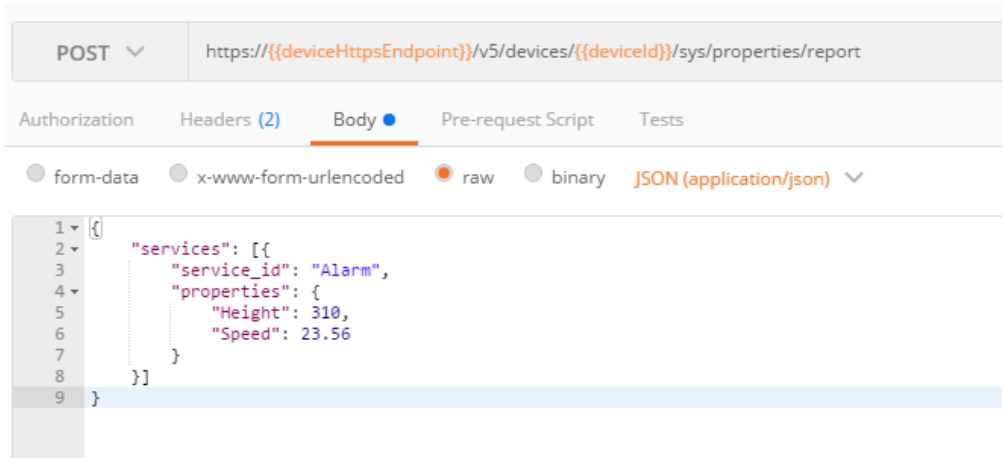


图 4-18 上报属性



----结束

HTTP 接口介绍

物联网平台的接口如下表所示：

接口分类	API	接口	说明
设备鉴权相关接口	设备鉴权接口说明	/v5/device-auth	设备鉴权接口，鉴权通过后才能建立设备与平台间的业务处理连接。鉴权成功后平台返回access_token。调用属性上报、消息上报等其他接口时，都需要携带access_token信息。如果access_token超期，需要重新认证设备获取access_token。如果access_token未超期重复获取access_token，原access_token在未超期前保留30s，30s之后失效。
设备消息相关接口	设备消息上报接口说明	/v5/devices/{device_id}/sys/messages/up	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
设备属性相关接口	设备属性上报接口说明	/v5/devices/{device_id}/sys/properties/report	用于设备按产品模型中定义的格式将属性数据上报给平台。
	网关上报子设备属性接口说明	/v5/devices/{device_id}/sys/gateway/sub-devices/properties/report	用于批量设备上报属性数据给平台。网关设备可以用此接口同时上报最多50个子设备的属性数据。

设备鉴权接口说明

设备鉴权接口鉴权通过后才能建立设备与平台间的业务处理连接。鉴权成功后平台返回access_token，调用属性上报、消息上报等其他接口时，都需要携带access_token信息。如果access_token超期，需要重新认证设备获取access_token。如果access_token未超期重复获取access_token，原access_token在未超期前保留30s，30s之后失效。

请求方法	POST
URI	/v5/device-auth
传输协议	HTTPS

名称	必选	类型	位置	说明
device_id	是	String	Body	**参数说明:** 设备ID，用于唯一标识一个设备。在注册设备时直接指定，或者由物联网平台分配获得。由物联网平台分配时，生成规则为"product_id" + "_" + "node_id"拼接而成。 **取值范围**：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。 **取值范围:** 长度1-128
sign_type	是	Integer	Body	**参数说明:** 密码校验方式: 0 代表HMACSHA256校验时间戳时不会校验消息时间戳与平台时间是否一致，仅判断密码是否正确; 1 代表HMACSHA256校验时间戳时会先校验消息时间戳与平台时间是否一致，再判断密码是否正确。 **取值范围:** 大小0~1
timestamp	是	String	Body	**参数说明:** 时间戳：为设备连接平台时的UTC时间，格式为YYYYMMDDHH，如UTC 时间2018/7/24 17:56:20 则应表示为2018072417。 **取值范围:** 固定长度10
password	是	String	Body	**参数说明:** password的值为使用“HMACSHA256”算法对secret进行签名后的密钥（以时间戳为key，对平台返回的secret进行签名后的值，参考**密钥生成工具**）。secret为注册设备时平台返回的secret。 **取值范围:** 固定长度64

名称	类型	说明
access_token	String	**参数说明:** 设备token，用于设备鉴权。 **取值范围:** 长度32-256
expires_in	Integer	**参数说明:** 鉴权信息的剩余有效时间, 单位：秒。

请求示例如下：

```
POST https://{endpoint}/v5/device-auth
Content-Type: application/json
```

```
{
  "device_id" : "60a87ffebaccd902c2f1abbb_0001",
  "sign_type" : 0,
  "timestamp" : "2019120219",
  "password" : "0a5c5c4adcee661b1e730cbf6c8c343ff5924c2f3100ec2f51cad6b060183ed0"
}
```

响应示例如下：

Status Code: 200 OK

```
Content-Type: application/json

{
  "access_token" : "d144a524-1997-4b99-94bf-f27128da8a34",
  "expires_in" : 86399
}
```

HTTP状态码	HTTP状态码描述	错误码	错误码描述	错误码中文描述
400	Bad Request	IOTDA.000006	Invalid input data.	请求参数不合法
401	Unauthorized	IOTDA.000002	The request is unauthorized.	鉴权失败
403	Forbidden	IOTDA.021101	Request reached the maximum rate limit.	请求已经达到限制速率
		IOTDA.021102	The request rate has reached the upper limit of the tenant, limit %s.	请求已经达到租户的限制速率

设备消息上报接口说明

用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。

请求方法	POST
URI	/v5/devices/{device_id}/sys/messages/up
传输协议	HTTPS

名称	必选	类型	位置	说明
access_token	是	String	Header	**参数说明:** 调用设备鉴权信息返回的access_token。 **取值范围:** 长度1-256

名称	必选	类型	位置	说明
device_id	是	String	Path	**参数说明: 参数说明**：设备ID，用于唯一标识一个设备。在注册设备时直接指定，或者由物联网平台分配获得。由物联网平台分配时，生成规则为"product_id" + "_" + "node_id"拼接而成。 **取值范围**：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。 **取值范围:** 长度1-128

说明

该接口支持设备将自定义数据通过请求中的body体上报给平台，平台收到该请求后会将body内容转发给应用服务器或华为云其他云服务上进行存储和处理。平台对body中的内容无具体格式限制，小于1MB的数据可以通过该接口携带。

请求示例如下：

```
POST https://{endpoint}/v5/devices/{device_id}/sys/messages/up
Content-Type: application/json
access_token: d144a524-1997-4b99-94bf-f27128da8a34
{
  "name" : "name",
  "id" : "id",
  "content" : "messageUp"
}
```

响应示例如下：

Status Code: 200 ok

HTTP状态码	HTTP状态码描述	错误码	错误码描述	错误码中文描述
400	Bad Request	IOTDA.000006	Invalid input data.	请求参数不合法
403	Forbidden	IOTDA.000004	Invalid access token.	非法token
		IOTDA.021101	Request reached the maximum rate limit.	请求已经达到限制速率
		IOTDA.021102	The request rate has reached the upper limit of the tenant, limit %s.	请求已经达到租户的限制速率

设备属性上报接口说明

用于设备按产品模型中定义的格式将属性数据上报给平台。

请求方法	POST
URI	/v5/devices/{device_id}/sys/properties/report
传输协议	HTTPS

名称	必选	类型	位置	说明
access_token	是	String	Header	**参数说明:** 调用设备鉴权信息返回的access_token。 **取值范围:** 长度1-256
device_id	是	String	Path	**参数说明:** **参数说明**：设备ID，用于唯一标识一个设备。在注册设备时直接指定，或者由物联网平台分配获得。由物联网平台分配时，生成规则为"product_id" + "_" + "node_id"拼接而成。 **取值范围**：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。 **取值范围:** 长度1-128
services	是	List<表4-33>	Body	**参数说明:** 设备服务数据列表。

表 4-33 ServiceProperty

名称	必选	类型	说明
service_id	是	String	**参数说明:** 设备服务id。
properties	是	Object	**参数说明:** 设备服务的属性列表，具体字段在设备关联的产品模型中定义。
event_time	否	String	**参数说明:** 设备采集数据UTC时间（格式：yyyy-MM-dd'T'HH:mm:ss.SSS'Z'），设备上报数据不带该参数或参数格式错误时，则数据上报时间以平台时间为准。

请求示例如下：

```
POST https://{endpoint}/v5/devices/{device_id}/sys/properties/report
Content-Type: application/json
access_token: d144a524-1997-4b99-94bf-f27128da8a34
```

```
{
  "services" : [ {
    "service_id" : "serviceId",
    "properties" : {
      "Height" : 124,
      "Speed" : 23.24
    },
    "event_time" : "2021-08-13T10:10:10.555Z"
  } ]
}
```

响应示例如下：

Status Code: 200 上报正常

HTTP状态码	HTTP状态码描述	错误码	错误码描述	错误码中文描述
400	Bad Request	IOTDA.000006	Invalid input data.	请求参数不合法
		IOTDA.021104	Subdevices in the request does not exist or does not belong to the gateway.	请求中有部分子设备不存在或不属于该网关.
403	Forbidden	IOTDA.000004	Invalid access token.	非法token
		IOTDA.021101	Request reached the maximum rate limit.	请求已经达到限制速率
		IOTDA.021102	The request rate has reached the upper limit of the tenant, limit %s.	请求已经达到租户的限制速率
		IOTDA.021105	The content reported in a single request cannot exceed 1 MB.	单次请求上报的内容不能超过1MB

网关上报子设备属性接口说明

用于批量设备上报属性数据给平台。网关设备可以用此接口同时上报最多50个子设备的属性数据。

请求方法	POST
URI	/v5/devices/{device_id}/sys/gateway/sub-devices/properties/report
传输协议	HTTPS

名称	必选	类型	位置	说明
access_token	是	String	Header	**参数说明:** 调用设备鉴权信息返回的access_token。 **取值范围:** 长度1-256
device_id	是	String	Path	**参数说明:** **参数说明**：设备ID，用于唯一标识一个设备。在注册设备时直接指定，或者由物联网平台分配获得。由物联网平台分配时，生成规则为"product_id" + "_" + "node_id"拼接而成。 **取值范围**：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。 **取值范围:** 长度1-128
devices	是	List<表4-34>	Body	**参数说明:** 设备数据列表。 **取值范围:** 长度不超过50

表 4-34 DeviceProperty

名称	必选	类型	说明
device_id	是	String	**参数说明:** **参数说明**：子设备的设备ID，用于唯一标识一个设备，在注册设备时由物联网平台分配获得。 **取值范围**：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
services	是	List<表4-35>	**参数说明:** 设备服务数据列表

表 4-35 ServiceProperty

名称	必选	类型	说明
service_id	是	String	**参数说明:** 设备服务id。
properties	是	Object	**参数说明:** 设备服务的属性列表，具体字段在设备关联的产品模型中定义。
event_time	否	String	**参数说明:** 设备采集数据UTC时间（格式：yyyy-MM-dd'T'HH:mm:ss.SSS'Z'），设备上报数据不带该参数或参数格式错误时，则数据上报时间以平台时间为准。

请求示例如下：

```
POST https://{endpoint}/v5/devices/{device_id}/sys/gateway/sub-devices/properties/report
Content-Type: application/json
access_token: d144a524-1997-4b99-94bf-f27128da8a34

{
  "devices" : [ {
    "device_id" : "deviceId_0001",
    "services" : [ {
      "service_id" : "serviceId",
      "properties" : {
        "Height" : 124,
        "Speed" : 23.24
      },
      "event_time" : "2021-08-13T10:10:10.555Z"
    } ]
  }, {
    "device_id" : "deviceId_0002",
    "services" : [ {
      "service_id" : "serviceId",
      "properties" : {
        "Height" : 124,
        "Speed" : 23.24
      },
      "event_time" : "2021-08-13T10:10:10.555Z"
    } ]
  } ]
}
```

响应示例如下：

Status Code: 200 上报正常

HTTP状态码	HTTP状态码描述	错误码	错误码描述	错误码中文描述
400	Bad Request	IOTDA.000006	Invalid input data.	请求参数不合法
		IOTDA.021104	Subdevices in the request does not exist or does not belong to the gateway.	请求中有部分子设备不存在或不属于该网关.
403	Forbidden	IOTDA.000004	Invalid access token.	非法token
		IOTDA.021101	Request reached the maximum rate limit.	请求已经达到限制速率
		IOTDA.021102	The request rate has reached the upper limit of the tenant, limit %s.	请求已经达到租户的限制速率
		IOTDA.021103	The request batch properties number has reached the upper limit, limit %s.	请求中子设备数量达到上限

HTTP状态码	HTTP状态码描述	错误码	错误码描述	错误码中文描述
		IOTDA.021105	The content reported in a single request cannot exceed 1 MB.	单次请求上报的内容不能超过1MB

4.3.3 MQTT(S)协议接入

概述

MQTT消息由固定报头（Fixed header）、可变报头（Variable header）和有效载荷（Payload）三部分组成。

其中固定报头（Fixed header）和可变报头（Variable header）格式的填写请参考**MQTT标准规范**，有效载荷（Payload）的格式（默认采用UTF-8编码格式）由应用定义，即由设备和物联网平台之间定义。

说明

MQTT的语法和接口细节，请以**MQTT标准规范**为准。

常见MQTT消息类型主要有CONNECT、SUBSCRIBE、PUBLISH。

- CONNECT：指客户端发起与服务端的连接请求。有效载荷（Payload）的主要参数，参考**设备连接鉴权**填写。
- SUBSCRIBE：指客户端发起订阅的请求。有效载荷（Payload）中的主要参数“Topic name”，参考**Topic定义**中订阅者为设备的Topic。
- PUBLISH：平台发布消息。
  - 可变报头（Variable header）中的主要参数“Topic name”，指当设备上报到物联网平台，发布者为设备时所对应的Topic。详细请参考**Topic定义**。
  - 有效载荷（Payload）中的主要参数为完整的数据上报和命令下发的消息内容，目前是一个JSON对象。

Topic 说明

设备使用MQTT协议接入时,可通过Topic实现消息的发送和接收。

- 以$oc开头的topic是IoTDA预置的系统topic。您可以在允许的情况下订阅和发布到这些系统预置的Topic；具体Topic列表和功能说明可参考**Topic定义**。
- 您可以创建非$oc 开头的topic进行自定义消息的发送和接收。

使用限制

描述	限制
单个MQTT直连设备在同一时间的连接数	1
单个MQTT直连设备最大建链每分钟请求次数	5/min

描述	限制
单账户设备侧每秒最大建链请求数量	500/s
单账号设备侧每秒最大上行的请求数量（单消息payload平均为512字节）	20000/s
单个MQTT连接每秒最大上行消息数量	50/s
单个MQTT连接最大带宽（上行消息）	1MB（默认）
MQTT单条发布消息最大长度。超过此大小的发布请求将被直接拒绝。	1MB
MQTT协议规范	MQTT v5.0、MQTT v3.1.1、MQTT v3.1
与标准MQTT协议的区别	• 不支持QoS2 • 不支持will、retain msg
MQTT协议支持的安全等级	采用TCP通道基础 + TLS协议（TLSv1、 TLSv1.1、TLSv1.2和TLSv1.3版本）
MQTT连接心跳时间建议值	心跳时间限定为30至1200秒，推荐设置为120秒
MQTT协议消息发布与订阅	设备只能对自己的Topic进行消息发布与订阅
MQTT协议每个订阅请求的最大订阅数	无限制
MQTT自定义Topic支持的最大长度	64字节
MQTT自定义Topic支持每个产品添加的最大个数	10个/产品
单账号支持上传设备侧CA证书个数	100个

与标准 MQTT 协议的兼容说明

华为云IoTDA服务支持设备基于**MQTT 5.0**、**MQTT 3.1.1**和 **MQTT 3.1**规范的接入，但同这些MQTT协议规范有一些差异， IoTDA服务不是简单的MQTT Broker，而是在支持设备使用MQTT协议接入的基础上集成消息通信、设备管理、规则引擎、数据流转等能力。与MQTT标准规范的的区别如下：

- 支持设备与IoTDA服务之间使用MQTT规范中的CONNECT、CONNACK、PUBLISH、PUBACK、SUBSCRIBE、SUBACK、UNSUBSCRIBE、UNSUBACK、PINGREQ、PINGRESP、DISCONNECT等报文进行通信。
- 支持MQTT的服务质量等级为QoS 0、QoS 1，不支持QoS 2。
- 支持MQTT协议规范中的clean session。
- 不支持MQTT协议规范中的will。 IoTDA提供设备状态推送的能力，设备离线后支持根据流转规则将设备状态推送到客户应用或者云服务。

- 不支持MQTT协议规范中retain msg。IoTDA提供消息缓存的能力消息上报和消息下发时支持对消息进行缓存。

支持的 MQTT 5.0 特性说明

说明

MQTT5.0相关特性仅在企业版支持。

IoTDA服务支持的MQTT 5.0的部分新增特性如下：

- 支持Topic Alias。将消息通信Topic缩小为整型数值，来减小MQTT报文，节约网络带宽资源。
- 支持ResponseTopic 和CorrelationData。消息上报和下发时支持携带这两个参数，实现类似云HTTP的请求和响应。
- 支持设置UserProperty属性列表。每个属性由Key和Value组成，用于在非payload区传输属性数据。
- 支持Content-Type属性。消息上报的报文可以携带Content-Type属性，标识报文类型。
- 支持在CONNACK和PUBACK报文中返回码，便于设备快速定位请求状态及问题。

MQTT 的 TLS 支持

平台推荐使用TLS来保护设备和平台的传输安全。 平台目前支持TLS1.0、1.1 、1.2和1.3版本以及GMTLS。其中TLS 1.0 和1.1 计划后续不再支持，建议使用 TLS 1.3作为首选 TLS 版本。GMTLS版本仅在国密算法的企业版才支持。

基础版，标准版以及支持通用加密算法的企业版使用TLS连接时，平台支持如下加密套件：

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA

支持国密算法的企业版使用TLS连接时平台支持如下加密套件：

- ECC_SM4_GCM_SM3
- ECC_SM4_CBC_SM3
- ECDHE_SM4_GCM_SM3
- ECDHE_SM4_CBC_SM3
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256

- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

说明

带RSA 和CBC的加密套件存在安全风险，请谨慎使用。

4.4 自定义设备侧域名

概述

自定义域名允许设备指定要连接到 IoTDA 的自定义完全限定域名 (FQDN)。使用自定义域名，用户可以管理自己的服务器证书，例如用于签署证书的根证书颁发机构 (CA)、签名算法、以及证书的生命周期。

使用场景

- 自己管理服务器证书的根证书颁发机构 (CA)、签名算法、以及证书的生命周期等。
- 出于品牌推广目的向客户公开公司自己的域名。
- 迁移场景继承自己原有的域名和服务器证书。

使用限制

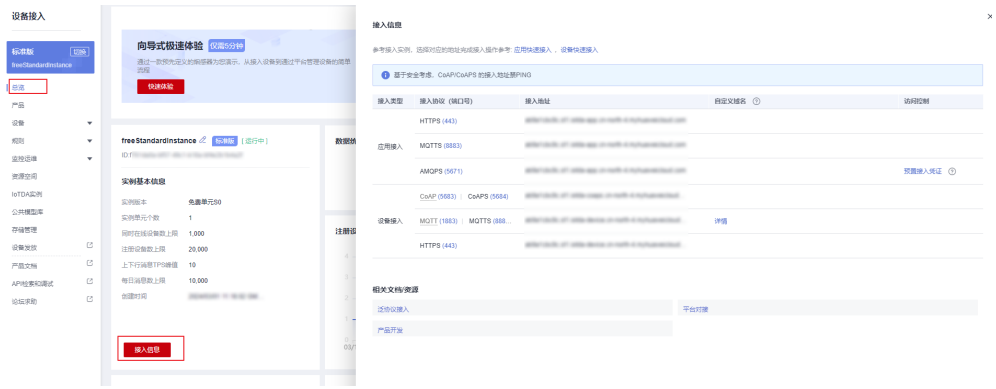
- 仅标准版和企业版支持该功能。
- 仅支持使用MQTT协议接入的8883端口生效。
- 使用自定义域名功能，要求设备必须使用TLS同时支持**SNI(Server Name Indication)**，SNI中需要携带指定的自定义域名。
- 单个IoTDA实例可配置的自定义域名数量为1个。

使用说明

步骤1 配置自定义域名。

1. 选择左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。

图 4-19 总览-获取接入信息



2. 单击“接入信息”页面中的“自定义域名”列中的“详情”，进入配置自定义域名的界面。

图 4-20 总览-自定义域名详情

3. 在自定义域名界面，单击“添加域名”，根据参数说明配置自定义域名后单击“确定”。

图 4-21 配置自定义域名

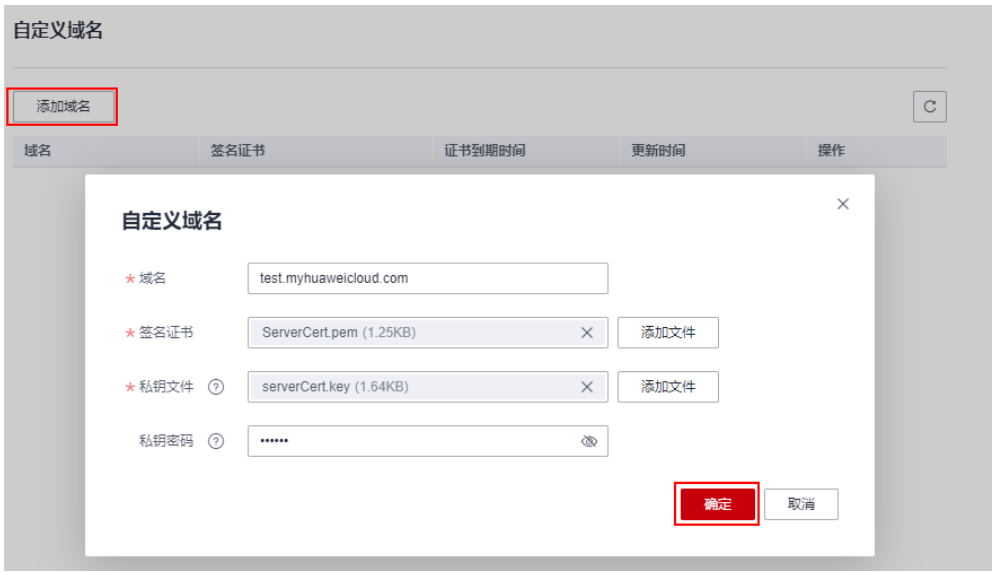


表 4-36 自定义域名参数信息说明

参数名称	是否必选	描述
域名	是	设备用于连接平台的自定义域名。 设备在连接时的域名必须同平台配置的自定义域名一致。
签名证书	是	域名对应的数字签名证书文件，支持pem、jks格式。证书公用名 (CN) 或使用者备用名称 (SAN) 字段中的域名需要同自定义域名匹配。
私钥文件	是	数字签名证书对应的私钥文件。
私钥密码	否	私钥对应的私钥密码。私钥文件使用密码加密时需要携带此参数。

步骤2 创建DNS记录；配置自定义域名后需要联系购买域名的厂商添加域名解析，以便自定义域名指向华为云IoTDA的接入点。您可以参考**平台对接信息**获取设备的接入点。

----结束

5 消息通信

5.1 设备数据上报

5.1.1 概述

概述

当设备和物联网平台完成对接后，设备可通过以下方式发送数据到物联网平台：

表 5-1 数据上报

类型	子类型	描述	适用场景	协议	物模型	大小
消息上报	设备消息上报	是设备直接将数据传到云端的一种方式，实现设备侧到应用侧的数据直接透传，平台对设备上报的消息不进行解析和存储。	常用于高频率数据的传输或需要用户自定义数据格式的场景。例如：短时间发送大量传感器数据到应用侧。	MQTT、HTTP	不依赖	• 单次请求：上报的消息最大为1MB。 • 单用户：标准版最大可使用带宽10Mb/s，企业版最大可使用带宽为50Mb/s。
	自定义Topic通信	根据客户需求，自定义Topic，平台将上报的数据直接透传。应用侧可以通过订阅自定义Topic区分不同的业务。	一般用于设备上报的业务类型有多样，需要根据实际业务自定义Topic的场景，或在数据转发中需要转发到特定的Topic的场景，比如数据迁移。	MQTT		

类型	子类型	描述	适用场景	协议	物模型	大小
属性上报	设备属性上报	属性上报的数据，设备侧到应用侧的数据不直接透传。数据在平台中会通过定义的产品模型来校验、进行过滤，若上报的数据不符合产品模型定义，平台会丢弃该数据。	希望建立统一的模型：规定数据的格式与取值范围、设备的数据需要平台解析、存储的场景。或希望平台能够存储最新的镜像数据的场景。例如：发送路灯的开关数据到应用侧。	MQTT、HTTP、LwM2M/CoAP	依赖	• 单次请求：上报的属性最大为64KB。 • 单用户：标准版最大可使用带宽10Mb/s，企业版最大可使用带宽为50Mb/s。
	网关批量属性上报	一次性上报多个子设备的属性，设备侧到应用侧的数据不直接透传。平台会将网关上报多子设备的数据分发到对应的子设备。	网关下关联多个子设备的场景。并且对于子设备数据上报时间不敏感，可以把多个子设备的数据打包后再一起上报。	MQTT		

说明

- 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时，可将原始二进制数据透传到物联网平台。通过**开发编解码插件**实现二进制数据到平台定义JSON格式的转换。
- 若发送到平台的数据需要发到华为云其他云服务上进行存储和处理，可以通过**数据转发规则**功能进行转发，然后再通过其他云服务的控制台或者API接口进行进一步的数据处理。

图 5-1 消息上报概念图

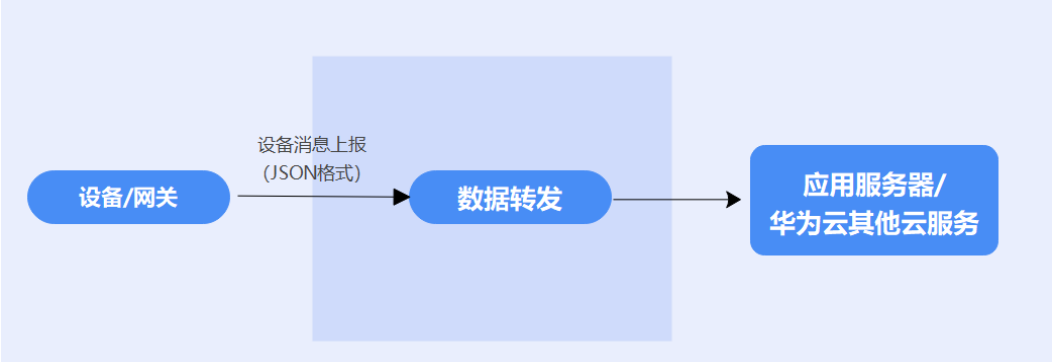


图 5-2 属性上报概念图

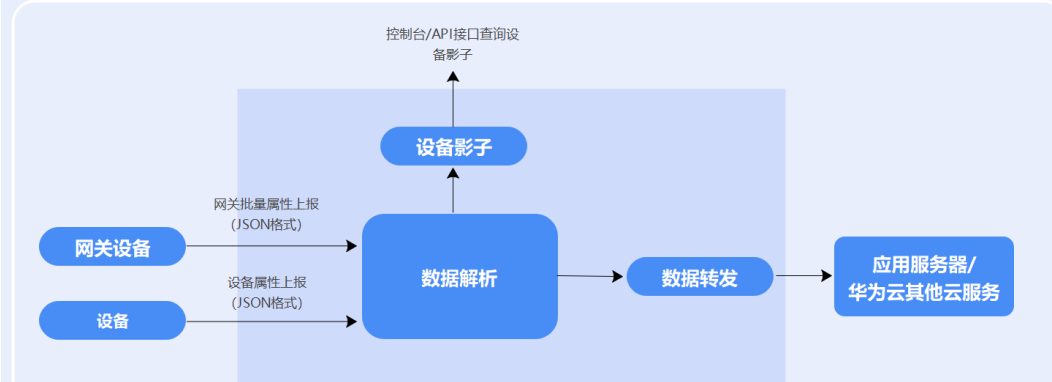
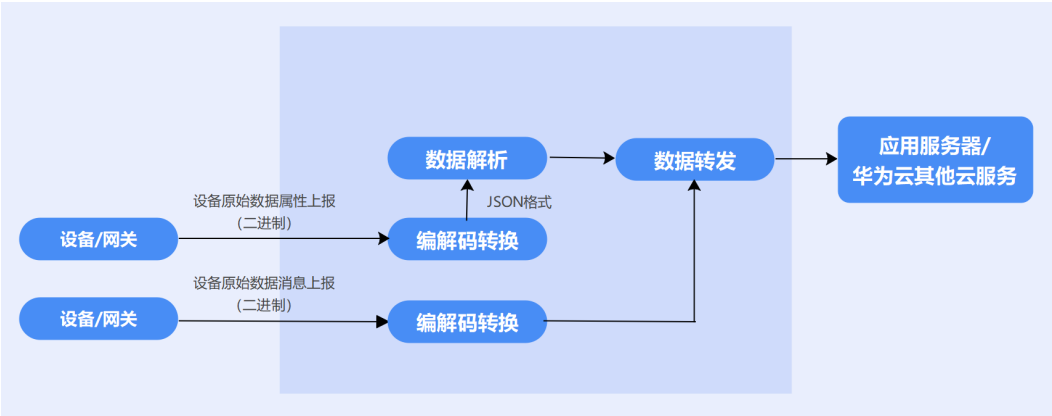


图 5-3 原始二进制数据上报概念图



相关应用侧 API 接口

- 修改设备属性
- 查询设备消息
- 查询设备
- 查询设备影子数据

相关设备侧 MQTT 接口

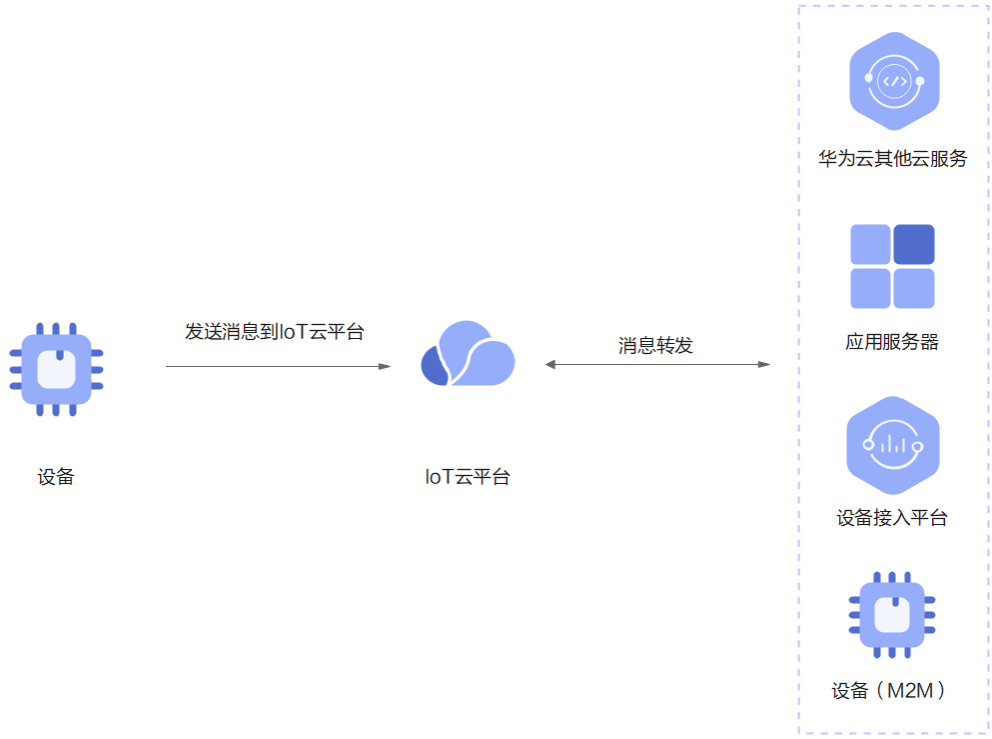
- 设备消息上报
- 设备属性上报
- 网关批量设备属性上报

5.1.2 设备消息上报

概述

消息上报是设备直接将数据传到云端，通过数据流转功能将数据转发到应用侧或华为云其他云服务的一种方式。平台对设备上报的消息不进行解析和存储，不需要建立产品模型便可以使用。

图 5-4 设备消息上报流程



使用场景

平台对设备上报的数据不进行解析和存储，需要通过**数据转发规则**转发到华为云其他云服务上进行存储和处理的场景。

使用限制

- 单个消息内容不大于1M。
- 单个MQTT连接最大带宽1MB/s。
- 单个MQTT连接每秒最大上行消息数量为50条（一次请求为一条）。

使用说明

图 5-5 设备消息上报操作流程

步骤1 创建产品与设备：**创建产品**流程、**创建设备**流程。

步骤2 **设备鉴权**：平台验证设备是否具有接入权限。

步骤3 设备消息上报：设备通过MQTT/HTTPS等协议发送消息数据。

按照不同的设备协议，调用的接口不一样，下面分别介绍MQTT、HTTPS协议消息上报的样例：

MQTT：通过消息上报接口**MQTT协议消息上报**上报数据到物联网平台。

- MQTT消息上报Topic样例如下：
  ```
  $oc/devices/{device_id}/sys/messages/up
  ```
- MQTT消息上报数据格式样例如下：
  ```
  {
     "content": {"hello":"123"}
  }
  ```

HTTPS：通过消息上报接口**HTTP协议消息上报**上报数据到物联网平台，其中access_token获取参考：**HTTPS设备鉴权**。HTTPS消息上报样例如下：

```
POST https://{endpoint}/v5/devices/{device_id}/sys/messages/up
Content-Type: application/json
access_token: d144a524-1997-4b99-94bf-f27128da8a34
{
   "name": "name",
   "id": "id",
   "content": "messageUp"
}
```

说明

协议接口详情请参考：**MQTT协议消息上报**、**HTTP协议消息上报**。

步骤4 数据转发：可以通过**数据流转功能**转发到应用侧或华为云其他云服务上进行进一步处理。

----结束

消息上报 JAVA SDK 使用示例

本部分介绍如何使用JAVA SDK进行消息上报的开发。本示例使用的开发环境为JDK 1.8及以上版本。

配置设备侧SDK步骤如下：

步骤1 SDK代码获取：**SDK下载**。

步骤2 配置设备侧SDK的Maven依赖。

```
<dependency>
   <groupId>com.huaweicloud</groupId>
   <artifactId>iot-device-sdk-java</artifactId>
   <version>1.1.4</version>
</dependency>
```

步骤3 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://接入地址:端口号。
//接入地址获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入地址。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
```

```
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤4 上报设备消息：

```
device.getClient().reportDeviceMessage(new DeviceMessage("hello"), new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        log.info("reportDeviceMessage success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        log.error("reportDeviceMessage fail: "+var2);
    }
});
```

----结束

测试验证步骤如下：

步骤1 在设备接入控制台，进入“设备 > 所有设备”，单击具体设备，启动“消息跟踪”。

步骤2 设备端运行设备侧SDK代码，设备侧消息上报日志格式样例如下：

图 5-6 java SDK 消息上报结果 log

步骤3 “消息跟踪”显示结果如下，平台已经收到设备的消息上报，并且已经触发流转规则:

图 5-7 消息跟踪-消息上报

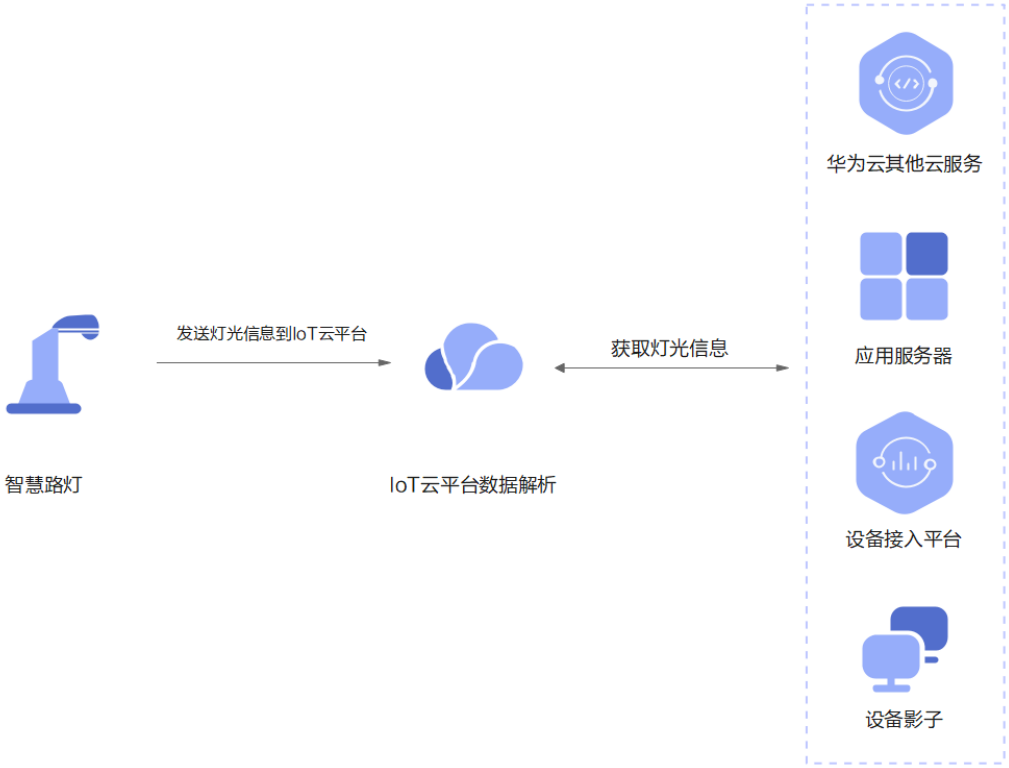
----结束

5.1.3 设备属性上报

概述

属性上报是一种需要平台解析、缓存，通过数据流转功能将数据转发到应用侧或华为云其他云服务的一种方式。需要在平台建立产品模型。对于属性上报的数据，平台会记录属性的最新一次上报值，对符合产品模型定义的属性数据进行存储。设备可以通过**设备侧获取平台的设备影子数据**向平台获取最新的设备属性值。

图 5-8 设备属性上报



使用场景

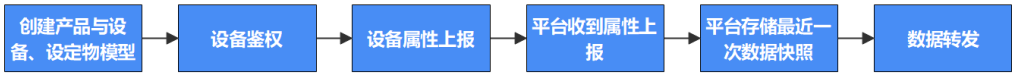
- 设备端及应用端需要平台进行转换、管理、缓存的数据。
- 需要通过**数据转发规则**转发到华为云其他云服务上进行存储和处理的场景。

使用限制

- 单个消息内容不大于64KB。
- 需要设置产品模型，数据内容需要与产品模型中定义的属性相匹配。
- 网关上报子设备属性时一次最大可上报子设备数量为100个。

使用说明

图 5-9 设备属性上报操作流程



步骤1 创建产品与设备、设定物模型：**创建产品**流程、**创建设备**流程、设定**产品模型**。

步骤2 **设备鉴权**：平台验证设备是否具有接入权限。

步骤3 设备属性上报：设备通过MQTT/HTTP/LwM2M等协议上报属性数据。

按照不同的设备协议，调用的接口不一样，下面分别介绍MQTT、HTTPS、LwM2M/CoAP这三种协议中属性上报的样例：

- **MQTT**：通过属性上报接口**MQTT协议属性上报**上报数据到物联网平台，MQTT属性上报样例如下：

  Topic：$oc/devices/{device_id}/sys/properties/report
  数据格式样例：

```
{
    "services": [
        {
            "service_id": "Temperature",
            "properties": {
                "value": 57,
                "value2": 60
            }
        }
    ]
}
```

- **HTTPS**：通过属性上报接口**HTTP协议属性上报**上报数据到物联网平台，其中access_token参考：**HTTPS设备鉴权**。HTTPS属性上报样例如下：

```
POST https://{endpoint}/v5/devices/{device_id}/sys/properties/report
Content-Type: application/json
access_token: d144a524-1997-4b99-94bf-f27128da8a34
{
    "services": [
        {
            "service_id": "serviceId",
            "properties": {
                "Height": 124,
                "Speed": 23.24
            }
        }
    ]
}
```

- **LwM2M/CoAP**：通过属性上报接口**LwM2M/CoAP属性上报**上报数据到物联网平台。LwM2M/CoAP属性上报样例如下：

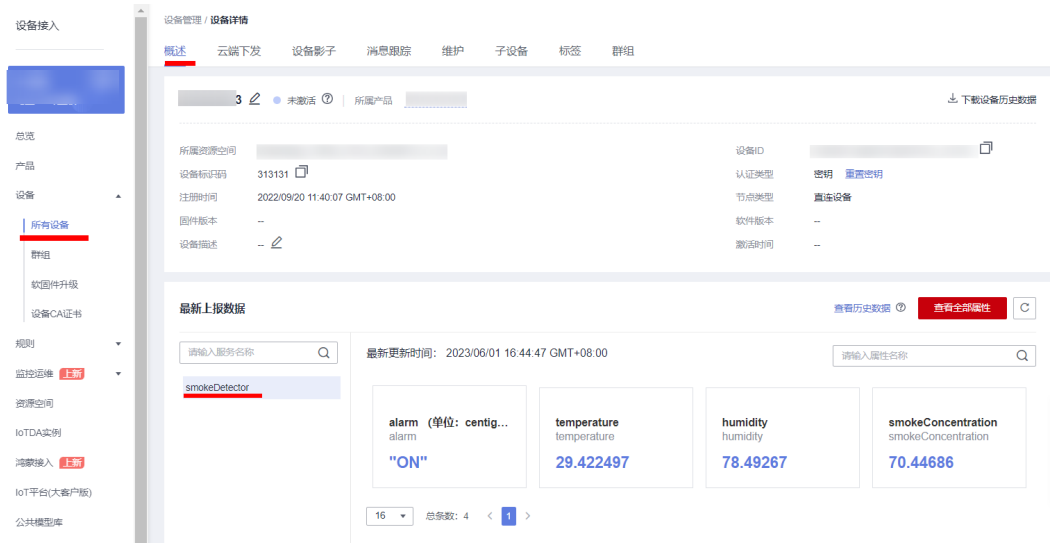
```
//假设设备上报的数据内容(value)为c4 0d 5a 6e 96 0b c3 0e 2b 30 37
 NON-2.05   MID=48590, Token=2cb6a673cba24c04, OptionSet={"Observe":22, "Content-Format":"application/octet-stream"}, c4 0d 5a 6e 96 0b c3 0e 2b 30 37
```

说明

- 设备属性上报内容需要与产品模型中定义的属性相匹配。
- 协议详情请参考：**MQTT协议属性上报**、**HTTP协议属性上报**、**LwM2M/CoAP属性上报**。

步骤4 平台存储最近一次数据快照：当上报的数据符合产品模型定义时，在设备接入控制台，选择“设备 > 所有设备”，单击具体设备，在“设备信息”中可以看到最新的数据快照。如下图：

图 5-10 平台接收到并成功解析设备属性上报



步骤5 数据转发：通过**数据流转功能**可以转发到应用侧，也可以转发到华为云其他云服务上进行存储和处理。

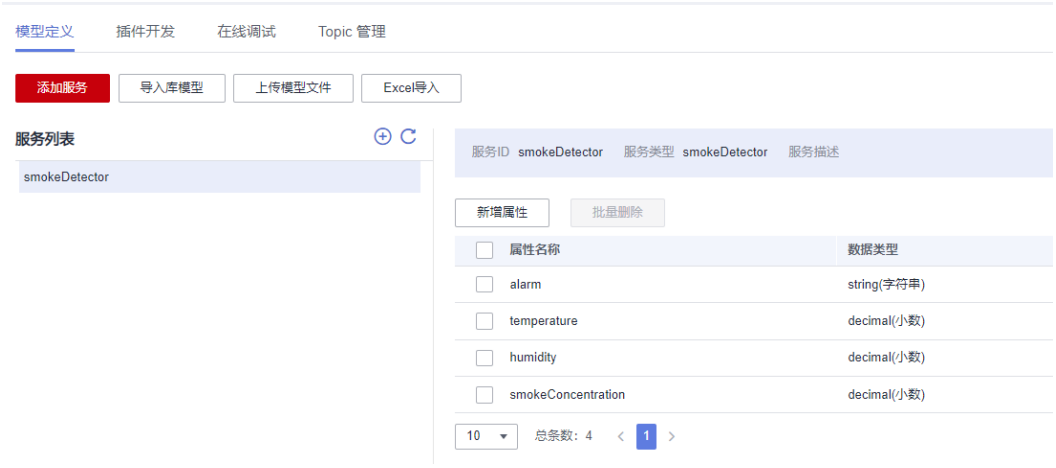
----结束

属性上报 JAVA SDK 使用示例

本部分介绍如何使用JAVA SDK进行属性上报的开发。本示例使用的开发环境为JDK 1.8及以上版本。

上报的属性需要与该设备对应**产品模型**中定义的属性相匹配，以下设备侧SDK代码示例定义的产品模型为：

图 5-11 属性上报产品定义



配置设备侧SDK步骤如下：

步骤1 SDK代码获取：**SDK下载**。

步骤2 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
```

```
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

步骤3 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤4 上报设备属性：

```
Map<String ,Object> json = new HashMap<>();
Random rand = new Random();

//按照物模型设置属性
json.put("alarm", alarm);
json.put("temperature", rand.nextFloat()*100.0f);
json.put("humidity", rand.nextFloat()*100.0f);
json.put("smokeConcentration", rand.nextFloat() * 100.0f);

ServiceProperty serviceProperty = new ServiceProperty();
serviceProperty.setProperties(json);
serviceProperty.setServiceId("smokeDetector");//serviceId要和物模型一致

device.getClient().reportProperties(Arrays.asList(serviceProperty), new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        log.info("reportProperties success" );
    }

    @Override
    public void onFailure(Object context, Throwable var2) {
        log.error("reportProperties failed" + var2.toString());
    }
});
```

----结束

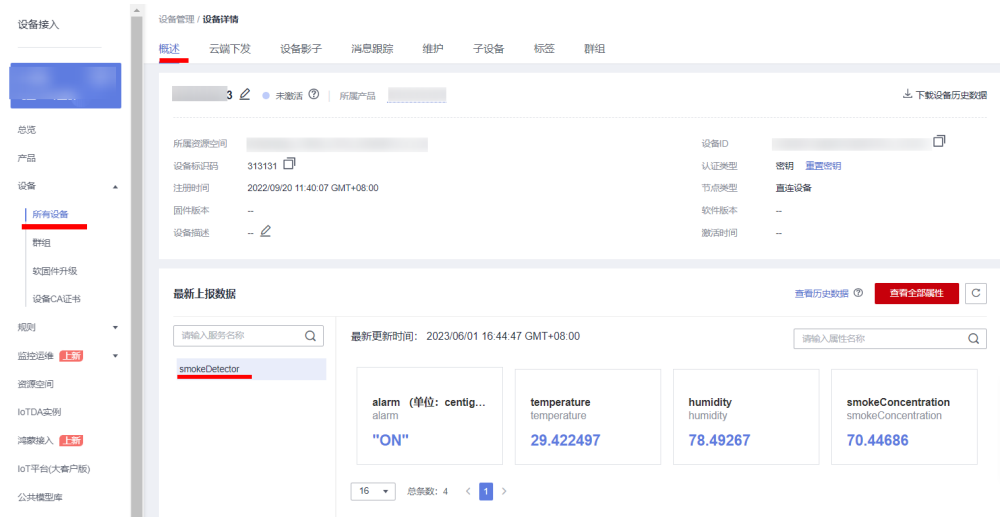
测试验证步骤如下：

步骤1 设备端运行设备侧 SDK代码，设备侧属性上报日志格式样例如下：

图 5-12 java SDK 属性上报结果 log

步骤2 进入设备接入控制台，选择“设备 > 所有设备”，单击具体设备，在“概述”中可以看到最新的上报数据。

图 5-13 平台接收到并成功解析设备属性上报



----结束

5.2 云端数据下发

5.2.1 概述

当设备和物联网平台完成对接后，物联网平台可通过以下方式发送数据到设备：

数据下发	描述	适用场景	设备影子	同步或异步	平台是否缓存	支持协议（设备侧）	物模型
消息下发	平台向设备直接下发消息，不依赖产品模型。提供给设备的单向通知，具有消息缓存功能，若设备不在线，则在设备在线后发送数据（支持配置，最长缓存时间24小时）。	设备无法按照产品模型中定义的格式进行指令下发时，可使用此接口下发自定义格式的数据给设备。例如：发送没有定义产品模型的数据。	不支持	异步	支持	MQTT	不依赖

数据下发	描述	适用场景	设备影子	同步或异步	平台是否缓存	支持协议（设备侧）	物模型
属性下发	用于设置、查询设备的属性值。设备接收到下发的属性后，需要设备及时将命令的执行结果返回给平台，如果设备没回响应，平台会认为命令执行超时。	用于平台主动获取或修改设备的属性值。例如APP每隔一段时间获取设备的地理位置。	支持	同步	不支持	MQTT、LwM2M/CoAP	依赖
命令下发	平台向设备下发设备控制命令，下发命令后，需要设备进行响应，该响应可以携带设备执行操作成功或者失败后的响应参数。 • 平台同步下发命令，需要设备及时将命令的执行结果返回给平台，若20s内无返回，则判定为失败。 • 异步命令下发具有消息缓存功能，若设备不在线，则在设备在线后发送数据（支持配置，最长缓存时间48小时）。	需要立即确认的命令。例如打开风扇、控制路灯开关。	不支持	同步	不支持	MQTT	依赖
				异步	支持	LwM2M/CoAP	

说明

配置且资源受限或者对网络流量有要求的设备，不适合直接接收JSON数据时，通过**开发编解码插件**实现应用侧JSON数据转换为设备侧二进制数据。

应用侧相关 API 接口

- **下发设备消息**
- **查询设备消息**
- **查询设备属性**
- **修改设备属性**
- **下发同步设备命令**
- **下发异步设备命令**
- **查询命令详情**

MQTT 设备相关 API 接口

- 平台命令下发
- 平台消息下发

LwM2M/CoAP 设备相关 API 接口

- 平台命令下发

5.2.2 消息下发

概述

消息下发不依赖产品模型，提供给设备的单向通知，具有消息缓存功能；云端消息下发中，平台会以异步方式（消息下发后无需等待设备侧回复响应）下发消息给设备；若设备不在线，则在设备在线后发送数据（支持配置，最长缓存时间24小时）。平台对每个设备默认只保存20条消息，超过20条后，后续的消息会替换下发最早的消息。同时，消息下发支持使用**自定义topic**的格式进行数据下发。

表 5-2 消息下发 Topic 类别

消息下发Topic类别	描述
系统Topic	平台预先定义了各种设备和平台通信的Topic，具体Topic列表和功能说明可参考**Topic定义**。
自定义Topic	用户可以自定义Topic，设备和平台间可以基于用户自定义的Topic进行通信。 **自定义topic分类：** • 在产品中定义需要使用的Topic，这类Topic有$oc/devices/{device_id}/user/前缀，消息上报或者消息下发时平台会校验Topic是否在产品中定义，未在产品中定义的Topic会被平台拒绝。使用方式可以参考链接**使用自定义Topic进行通信**的最佳实践。 • 使用非$oc开头的自定义Topic，如/aircondition/data/up进行消息通信，这类Topic平台不校验Topic权限，根据MQTT协议定义的规则进行Topic的消息上下行通信。

使用场景

- 数据格式需要自定义，不依赖物模型。
- 大量数据高频率下发。

使用限制

- 单个消息内容不大于256KB。
- 单个设备下发消息缓存数量为20个。
- MQTT自定义Topic支持的最大长度为128字节。

- 缓存时间支持配置，最长不超过24小时。

使用服务质量

- 支持MQTT的服务质量等级为QoS 0、QoS 1，不支持QoS 2。
- Topic的QoS为 0 时，消息下发不需要等设备回ACK，仅只下发一次; Topic的QoS为 1时，消息下发需要等设备回ACK。
- 平台默认该设备订阅了QoS为0的系统Topic, 如果需要QoS为1的下行系统Topic，需要设备设置订阅QoS。
- 设备订阅非$oc开头的自定义Topic, 如果需要平台支持该Topic的QoS为1, 需向平台**提交工单**联系申请。
- 如果设备订阅Topic的QoS为1, 平台下发消息没有收到设备确认ACK时, 平台会重发消息, 默认每隔2s重发1次, 共重发3次;

  重发后设备依旧没有回确认响应且消息还在缓存时间内, 设备再次上线或订阅Topic时, 平台会再重发消息, 默认每隔10s重发1次, 共重发5次;

  同时每次重发也会触发每隔2s重发机制, 故订阅Topic的QoS为1时平台会重发消息, 设备可能会收到重复消息，建议设备要有去重机制。

相关 API 接口

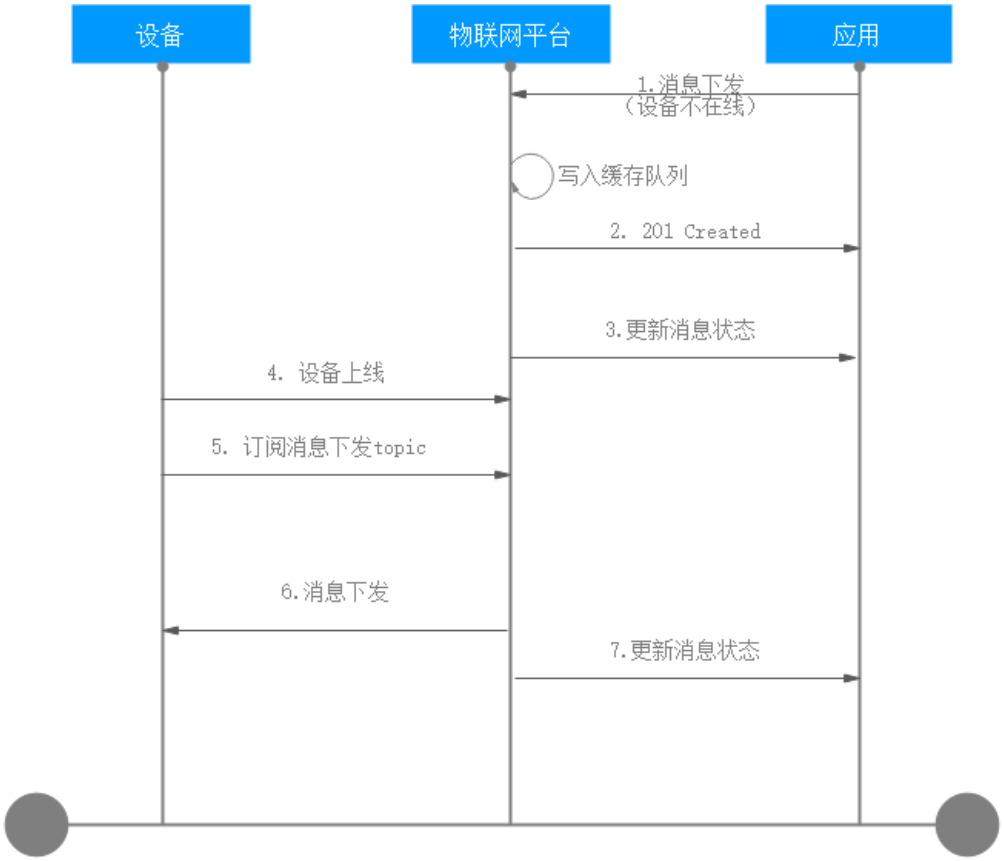
- **下发设备消息**
- **平台消息下发**

消息缓存下发使用说明

消息下发是平台向设备直接下发消息的一种方式。具有缓存特性，当设备不在线时，平台会对下发的消息进行缓存，直到设备上线。

以MQTT设备接入为例，使用系统Topic进行设备消息缓存下发说明：

图 5-14 消息缓存下发流程



1. 应用侧或平台用**下发设备消息**接口，下发请求到物联网平台，下发消息样例如下：

```
POST https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/messages
Content-Type: application/json
X-Auth-Token: ********

{
  "message_id": "99b32da9-cd17-4cdf-a286-f6e849cbc364",
  "name": "messageName",
  "message": "HelloWorld"
}
```

2. 物联网平台向应用返回201 Created，消息状态为PENDING。
3. 物联网平台通过**设备消息状态变更通知接口**推送消息结果给应用，设备未上线时，设备消息状态为等待（PENDING），对应的消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/messages/down
数据格式：
{
    "resource": "device.message.status",
    "event": "update",
    "notify_data": {
      "message_id": "string",
      "name": "string",
      "device_id": "string",
      "status": "PENDING",
      "timestamp": "string"
     }
 }
```

4. 设备上线。

5. 设备订阅消息下发的topic，用于接收消息，订阅的topic见步骤**6**。
6. 物联网平台根据协议规范下发消息给设备。MQTT设备必须先订阅**平台消息下发**下行接口对应的Topic才能收到平台下发的消息，消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/messages/down
数据格式：
{
    "object_device_id": "{object_device_id}",
    "name": "name",
    "id": "id",
    "content": "hello"
}
```

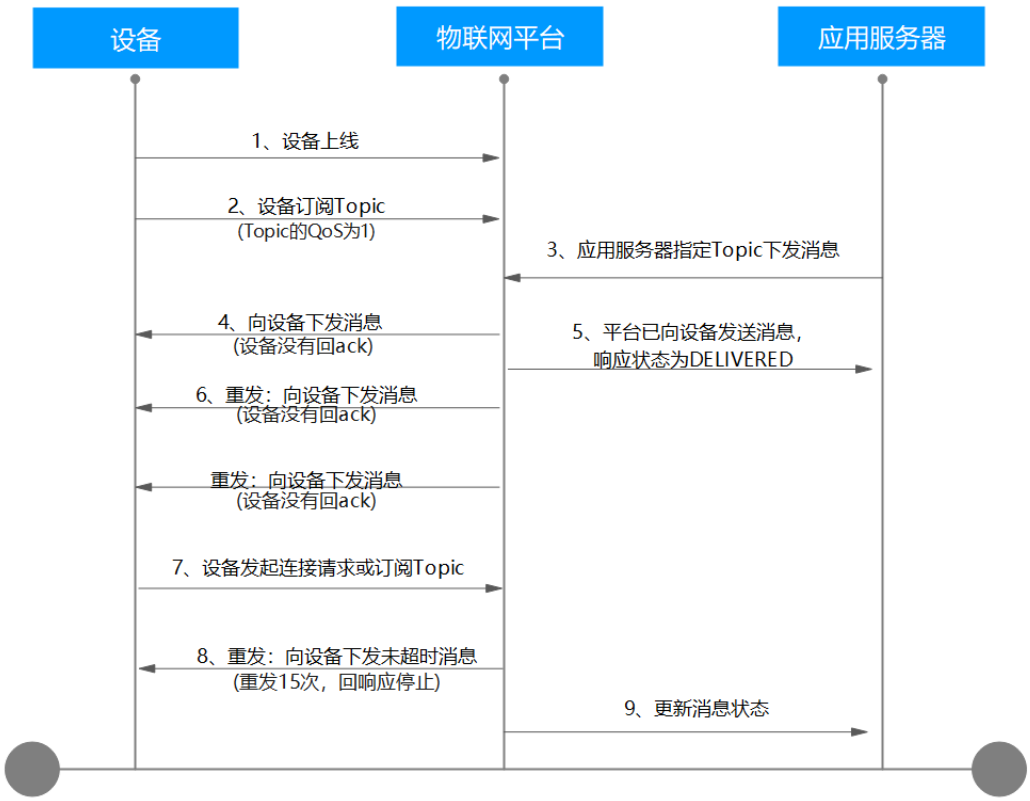
7. 平台将消息的最终结果推送给应用服务器，设备消息状态为已送达（DELIVERED）。使用接口：**设备消息状态变更通知接口**。

```
Topic: $oc/devices/{device_id}/sys/messages/down
数据格式：
{
    "resource": "device.message.status",
    "event": "update",
    "notify_data": {
      "message_id": "string",
      "name": "string",
      "device_id": "string",
      "status": "DELIVERED",
      "timestamp": "string"
     }
}
```

消息下发使用 QoS 1 说明

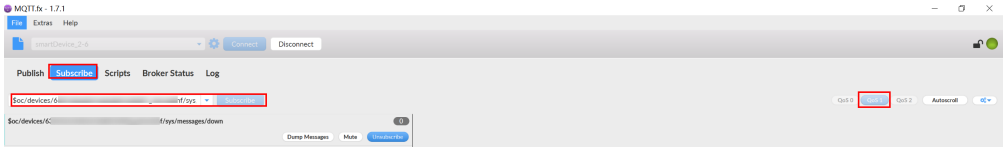
MQTT设备接入，使用QoS 1的系统Topic进行设备消息下发说明：

图 5-15 消息下发使用 Qos 1 流程图



1. 设备上线。
2. 订阅Topic，设置订阅Topic的QoS为1。

图 5-16 订阅 Topic 的 QoS 为 1



3. 应用侧或平台用**下发设备消息**接口，下发请求到物联网平台，下发消息样例如下：

```
POST https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/messages
Content-Type: application/json
X-Auth-Token: ********

{
  "message_id": "99b32da9-cd17-4cdf-a286-f6e849cbc364",
  "name": "messageName",
  "message": "HelloWorld"
}
```

4. 物联网平台根据协议规范下发消息给设备。MQTT设备必须先订阅**平台消息下发**下行接口对应的Topic才能收到平台下发的消息，消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/messages/down
数据格式：
{
    "object_device_id": "{object_device_id}",
    "name": "name",
    "id": "id",
    "content": "hello"
}
```

5. 物联网平台向设备下发消息后，向应用服务器返回201 Created，消息状态为DELIVERED。消息下发是异步操作，不需要等设备ACK就可以回响应。
6. 物联网平台没有收到设备接收消息的ACK响应，重发消息；默认每隔2s重发一次，总下发3次。
7. 设备再次上线或订阅Topic。
8. 物联网平台会重发之前设备未回ACK且未超时的消息，默认每隔10s重发一次，总下发5次；每次重发也会触发每隔2s重发机制。
9. 平台将消息的最终结果推送给应用服务器，设备消息状态为已送达（DELIVERED）或超时（TIMEOUT）。使用接口：**设备消息状态变更通知接口**。

```
Topic: $oc/devices/{device_id}/sys/messages/down
数据格式：
{
    "resource": "device.message.status",
    "event": "update",
    "notify_data": {
      "message_id": "string",
      "name": "string",
      "device_id": "string",
      "status": "DELIVERED",
      "timestamp": "string"
    }
}
```

下发消息状态

MQTT设备消息执行状态以及状态变化机制如下所示。

图 5-17 设备消息状态

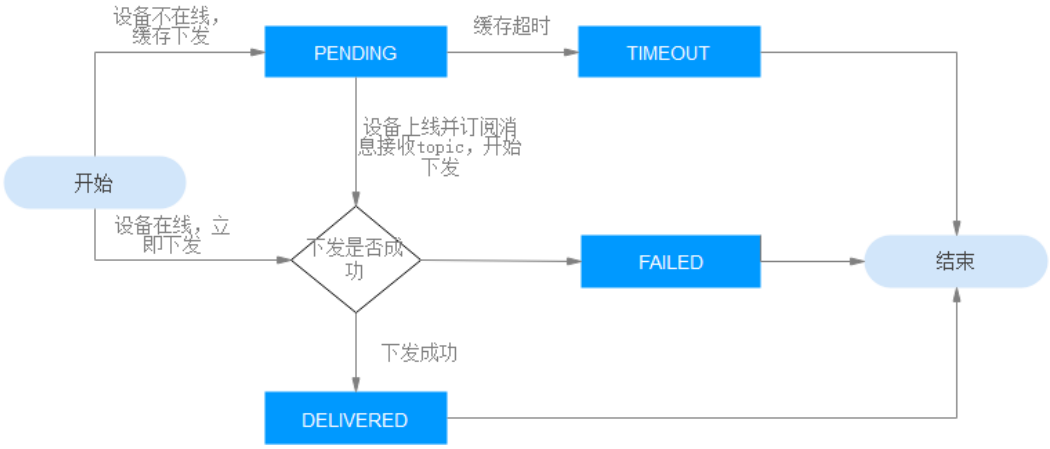


表 5-3 消息执行状态

消息执行状态	说明
等待（PENDING）	MQTT协议设备不在线，物联网平台会将消息进行缓存，此时任务状态为“等待”状态。
超时（TIMEOUT）	物联网平台缓存的PENDING状态的消息，如果1天之内还没有下发下去，物联网平台会将消息状态设置为“超时”。
已送达（DELIVERED）	物联网平台将消息发送给设备后，状态变为“已送达”。
失败（FAILED）	物联网平台发送消息给设备不成功，消息状态变为“失败”。

平台消息下发使用示例

云端消息下发，控制台上创建下发任务，MQTT协议设备接入为例，在设备接入控制台上进行消息缓存下发。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

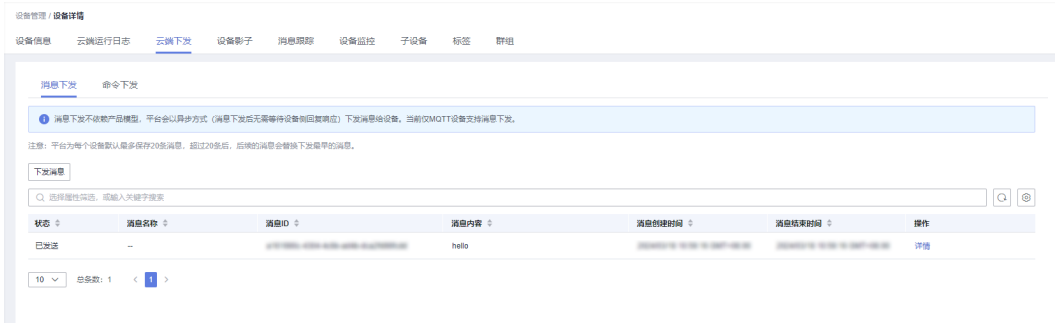
步骤2 单击“所有设备”，在设备列表中，单击具体的设备进入到设备的详情页面。

步骤3 在“云端下发>消息下发”标签页，单击“下发消息”，在弹出的窗口中选择需要下发的命令并设置命令参数。

图 5-18 下发消息-MQTT

步骤4 可以在平台看到下发状态为已发送。

图 5-19 下发消息-查询结果



----结束

配置应用侧Java SDK步骤如下：

步骤1 配置Maven依赖，本示例使用的开发环境为JDK 1.8及以上版本。SDK代码获取：**SDK下载**。

```
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-core</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-iotda</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

步骤2 应用侧向单个设备下发消息样例如下：

```
public class MessageDistributionSolution {
   // REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
   private static final String REGION_ID = "<YOUR REGION ID>";
   // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
   private static final String ENDPOINT = "<YOUR ENDPOINT>";
   // 标准版/企业版：需自行创建Region对象
   public static final Region REGION_CN_NORTH_4 = new Region(REGION_ID, ENDPOINT);
   public static void main(String[] args) {
      String ak = "<YOUR AK>";
```

```
        String sk = "<YOUR SK>";
        String projectId = "<YOUR PROJECTID>";
        // 创建认证
        ICredential auth = new
BasicCredentials().withDerivedPredicate(AbstractCredentials.DEFAULT_DERIVED_PREDICATE)
            .withAk(ak)
            .withSk(sk)
            .withProjectId(projectId);
        // 创建IoTDAClient实例并初始化
        IoTDAClient client = IoTDAClient.newBuilder().withCredential(auth)
            // 基础版：请选择IoTDARegion中的Region对象
            //.withRegion(IoTDARegion.CN_NORTH_4)
            // 标准版/企业版：需自行创建Region对象
            .withRegion(REGION_CN_NORTH_4).build();
        // 实例化请求对象
        CreateMessageRequest request = new CreateMessageRequest();
        request.withDeviceId("<YOUR DEVICE_ID>");
        DeviceMessageRequest body = new DeviceMessageRequest();
        body.withMessage("HelloWorld");
        request.withBody(body);
        try {
            CreateMessageResponse response = client.createMessage(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.printStackTrace();
        } catch (RequestTimeoutException e) {
            e.printStackTrace();
        } catch (ServiceResponseException e) {
            e.printStackTrace();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getRequestId());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

表 5-4 参数说明

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
projectId	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.java**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**平台对接信息**。
REGION_ID	如果是上海一，请填写“cn-east-3”；如果是北京四，请填写“cn-north-4”;如果是华南广州，请填写“cn-south-4”。
ENDPOINT	请在控制台的“总览”界面的“接入信息”中查看“应用接入”的https接入地址。
DEVICE_ID	下发消息的设备ID，用于唯一标识一个设备，在注册设备时由物联网平台分配获得。 取值范围：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。

----结束

配置设备侧Java SDK步骤如下，示例中使用的开发环境为JDK 1.8及以上版本。SDK代码获取：**SDK下载**。

步骤1 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

步骤2 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤3 消息下发回调函数定义。

```
client.setDeviceMessageListener(deviceMessage -> {
    log.info("the onDeviceMessage is {}", deviceMessage.toString());
});
```

----结束

测试验证步骤如下：

步骤1 在设备接入控制台，进入“设备 > 所有设备”，单击具体设备，启动“消息跟踪”。

步骤2 运行应用侧SDK代码，下发消息，应用侧可以看到平台响应样例如下:

图 5-20 应用侧消息下发成功响应

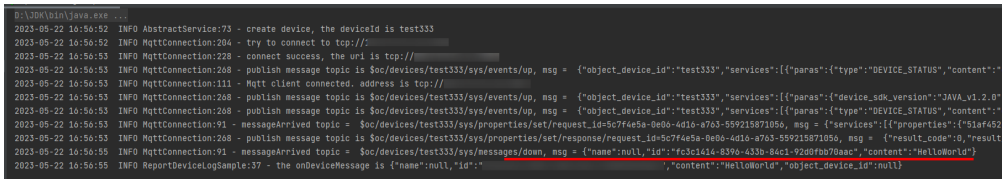
```
class CreateMessageResponse {
    messageId: 8a33c102-3ece-471c-ad87-96313daa0e53
    result: class MessageResult {
        status: PENDING
        createdTime: 20230606T083644Z
        finishedTime: null
    }
}
```

步骤3 在平台的消息跟踪中可以看到：

图 5-21 平台收到应用下发消息并缓存

步骤4 设备端运行设备侧 SDK代码，设备侧收到消息时日志格式样例如下：

图 5-22 设备消息下发成功



----结束

5.2.3 属性下发

概述

属性下发分为查询设备属性和修改属性参数两种，查询设备属性用于应用侧或平台主动获取设备属性数据，修改属性参数用于应用侧或平台设置设备属性值并同步到设备侧。设备接收到属性下发指令后需要立即响应，如果设备没有响应，平台会认为命令执行超时。

使用场景

- 用于平台主动获取或修改设备属性值。
- 进行平台规范、解析、过滤的数据。

使用限制

- 单个消息内容不大于64K。
- 需要定义**产品模型**。

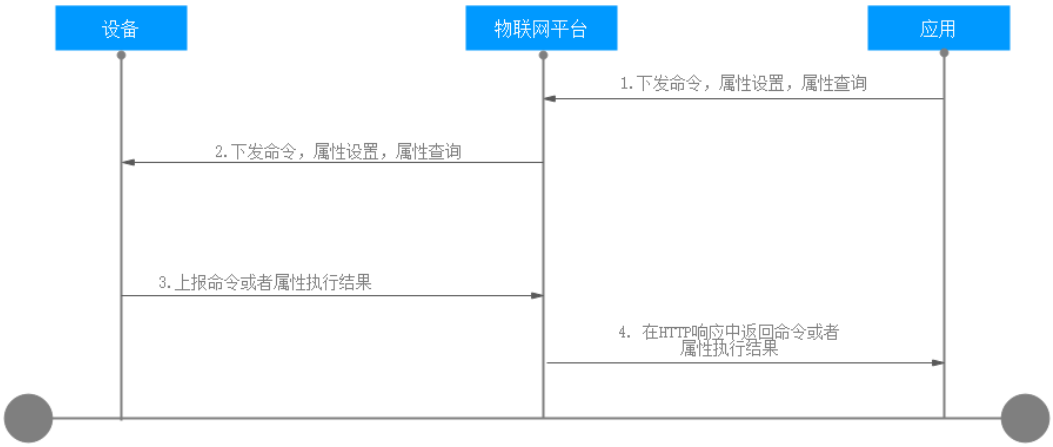
相关 API 接口

- **平台设置设备属性**
- **平台查询设备属性**

属性下发使用说明

属性下发分为修改属性与获取属性值，下列以修改属性为例，介绍属性下发。

图 5-23 属性下发流程图



1. 应用调用**修改设备属性**接口，下发请求到物联网平台，属性下发消息样例如下：

```
PUT https://{endpoint}/v5/iot/{project_id}/devices/{device_id}/properties
{
   "services" : [ {
      "service_id" : "Temperature",
      "properties" : {
         "value" : 57
      }
    }, {
      "service_id" : "Battery",
      "properties" : {
         "level" : 80
      }
   } ]
}
```

2. 物联网平台根据协议规范下发属性给设备。通过MQTT协议中**平台设置设备属性**接口下发属性为样例：

Topic: $oc/devices/{device_id}/sys/properties/set/request_id={request_id}
数据格式：

```
{
   "object_device_id": "{object_device_id} ",
   "services": [
      {
         "service_id": "Temperature",
         "properties": {
            "value": 57,
            "value2": 60
         }
      },
      {
         "service_id": "Battery",
         "properties": {
            "level": 80,
            "level2": 90
         }
      }
   ]
}
```

3. 设备执行属性下发命令后返回命令执行结果，消息样例如下：

Topic：$oc/devices/{device_id}/sys/properties/set/response/request_id={request_id}
数据格式：

```
{
   "result_code": 0,
   "result_desc": "success"
}
```

4. 应用侧收到发送HTTP下发命令的同步响应结果。消息样例如下：

```
Status Code: 200 OK
Content-Type: application/json
{
    "response" : {
        "result_code" : 0,
        "result_desc" : "success"
    }
}
```

属性下发 Java SDK 使用示例

本部分介绍如何使用JAVA SDK进行属性配置的开发。SDK代码获取：**SDK下载**。本示例使用的开发环境为JDK 1.8及以上版本。

配置应用侧SDK步骤如下：

步骤1 配置Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-core</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-iotda</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

步骤2 应用侧设置某个设备的属性值，样例如下：

```
public class AttributeDistributionSolution {
    // REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
    private static final String REGION_ID = "<YOUR REGION ID>";
    // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
    private static final String ENDPOINT = "<YOUR ENDPOINT>";
    // 标准版/企业版：需自行创建Region对象
    public static final Region REGION_CN_NORTH_4 = new Region(REGION_ID, ENDPOINT);
    public static void main(String[] args) {
        String ak = "<YOUR AK>";
        String sk = "<YOUR SK>";
        String projectId = "<YOUR PROJECTID>";
        // 创建认证
        ICredential auth = new BasicCredentials().withDerivedPredicate(AbstractCredentials.DEFAULT_DERIVED_PREDICATE)
            .withAk(ak)
            .withSk(sk)
            .withProjectId(projectId);
        // 创建IoTDAClient实例并初始化
        IoTDAClient client = IoTDAClient.newBuilder().withCredential(auth)
            // 基础版：请选择IoTDARegion中的Region对象
            //.withRegion(IoTDARegion.CN_NORTH_4)
            // 标准版/企业版：需自行创建Region对象
            .withRegion(REGION_CN_NORTH_4).build();
        // 实例化请求对象
        UpdatePropertiesRequest request = new UpdatePropertiesRequest();
        request.withDeviceId("<YOUR DEVICE_ID>");
        DevicePropertiesRequest body = new DevicePropertiesRequest();
        body.withServices("[{\"service_id\":\"smokeDetector\",\"properties\":{\"alarm\":\"hello\"," +
                "\"temperature\":10.323,\"humidity\":654.32,\"smokeConcentration\":342.4}}]");
        request.withBody(body);
        try {
            UpdatePropertiesResponse response = client.updateProperties(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.printStackTrace();
        } catch (RequestTimeoutException e) {
            e.printStackTrace();
```

```
        } catch (ServiceResponseException e) {
            e.printStackTrace();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getRequestId());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

表 5-5 参数说明

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
projectId	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.java**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**平台对接信息**。
REGION_ID	如果是上海一，请填写“cn-east-3”；如果是北京四，请填写“cn-north-4”;如果是华南广州，请填写“cn-south-4”。
ENDPOINT	请在控制台的“总览”界面的“接入信息”中查看“应用接入”的https接入地址。
DEVICE_ID	下发消息的设备ID，用于唯一标识一个设备，在注册设备时由物联网平台分配获得。 取值范围：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。

----结束

配置设备侧SDK步骤如下：

步骤1 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

步骤2 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = AttributeSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
```

```
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤3 定义属性下发回调函数。

```
device.getClient().setPropertyListener(new PropertyListener() {

    //处理写属性
    @Override
    public void onPropertiesSet(String requestId, List<ServiceProperty> services) {

        //遍历service
        for (ServiceProperty serviceProperty: services){

            log.info("OnPropertiesSet, serviceId =  " + serviceProperty.getServiceId());

            //遍历属性
            for (String name :serviceProperty.getProperties().keySet()){
                log.info("property name = "+ name);
                log.info("set property value = "+ serviceProperty.getProperties().get(name));
                if (name.equals("alarm")){
                    //修改本地值
                    alarm = (Integer) serviceProperty.getProperties().get(name);
                }
            }

        }
        //设置属性响应
        device.getClient().respondPropsSet(requestId, IotResult.SUCCESS);
    }

    //处理读属性
    @Override
    public void onPropertiesGet(String requestId, String serviceId) {
        log.info("OnPropertiesGet " + serviceId);
        Map<String ,Object> json = new HashMap<>();
        Random rand = new Random();
        json.put("alarm", alarm);
        json.put("temperature", rand.nextFloat()*100.0f);
        json.put("humidity", rand.nextFloat()*100.0f);
        json.put("smokeConcentration", rand.nextFloat() * 100.0f);

        ServiceProperty serviceProperty = new ServiceProperty();
        serviceProperty.setProperties(json);
        serviceProperty.setServiceId("smokeDetector");

        //上报读属性响应
        device.getClient().respondPropsGet(requestId, Arrays.asList(serviceProperty));
    }
});
```

----结束

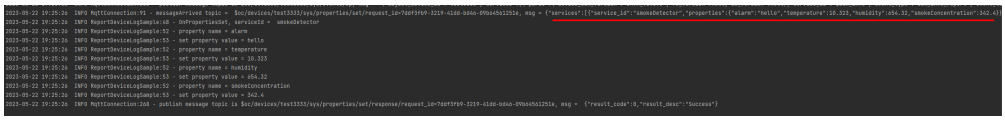
测试验证步骤如下：

步骤1 在设备接入控制台，进入“设备 > 所有设备”，单击具体设备，启动“消息跟踪”。

步骤2 运行设备侧 SDK代码，使设备上线。

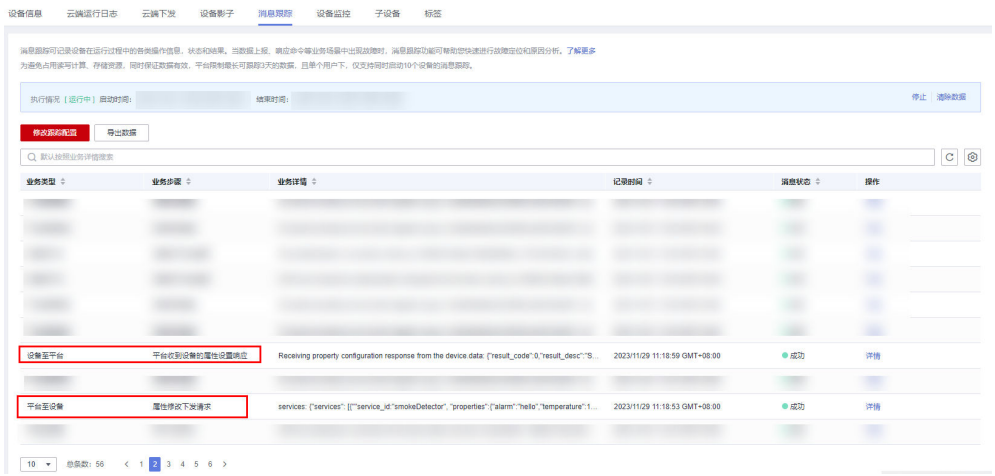
步骤3 运行应用侧SDK代码，调用修改设备属性接口向设备发送请求，设备侧收到的结果如下：

图 5-24 处理修改属性设备侧结果



步骤4 “消息跟踪”显示结果如下:

图 5-25 属性下发在消息跟踪中查看



----结束

5.2.4 命令下发

概述

为能有效地对设备进行管理，设备的产品模型中定义了物联网平台可向设备下发的命令，应用服务器可以调用物联网平台应用侧API接口向设备下发命令，以实现对设备的远程控制。

物联网平台有同步命令下发和异步命令下发两种命令下发机制，如下表所示。

表 5-6 命令下发概述

命令下发机制	定义	适用场景	LwM2M/CoAP协议设备	MQTT协议设备
同步命令下发	应用服务器可调用同步命令下发接口向指定设备下发命令，以实现对设备的同步控制。平台负责将命令以同步方式发送给设备，并将设备执行命令结果在HTTP请求中同步返回, 如果设备没有响应，平台会返回给应用服务器超时。	同步命令下发适合对命令实时性有要求的场景，比如路灯开关灯，燃气表开关阀。使用同步命令下发时，命令下发的时机需要由应用服务器来保证。	不适用	适用

命令下发机制	定义	适用场景	LwM2M/CoAP协议设备	MQTT协议设备
异步命令下发	应用服务器可调用异步命令下发接口向指定设备下发命令，以实现对设备的控制。平台负责将命令发送给设备，并将命令执行结果异步推送给应用。 异步命令下发又分为缓存下发和立即下发。 • 立即下发：不管设备是否在线，平台收到命令后立即下发给设备。如果设备不在线或者设备没收到指令则下发失败。 • 缓存下发：物联网平台在收到命令后先缓存，等设备上线或者设备上报属性时再下发给设备，如果单个设备存在多条缓存命令，则进行排队串行下发。	• 立即命令下发适用于实时性要求高的场景； • 缓存下发适合对命令实时性要求不高的场景，比如配置水表的参数。	适用	不适用

说明

具体使用流程请见：**同步命令下发**、**异步命令下发**。

使用场景

- 同步下发适合对命令实时性有要求的场景。异步用于实现对设备的控制。
- 需要通过**数据转发规则**转发到华为云其他云服务上进行存储和处理的场景。

使用限制

- 单个消息内容不大于256KB。
- 需要定义**产品模型**。
- 同步命令设备响应时间为20秒以内。
- 设备异步命令缓存数量为20个。
- 设备异步命令缓存时间支持配置，最长不超过48小时。

命令下发相关 API 接口

- 平台及应用侧API接口

  - **下发同步设备命令**
  - **下发异步设备命令**
  - **查询命令详情**
- MQTT设备接口
  - **平台命令下发**
  - **平台消息下发**
- LwM2M/CoAP设备接口
  - **平台命令下发**

同步命令下发概述

同步命令下发主要用于MQTT设备，分为单个MQTT设备同步命令下发与批量MQTT设备同步命令下发。

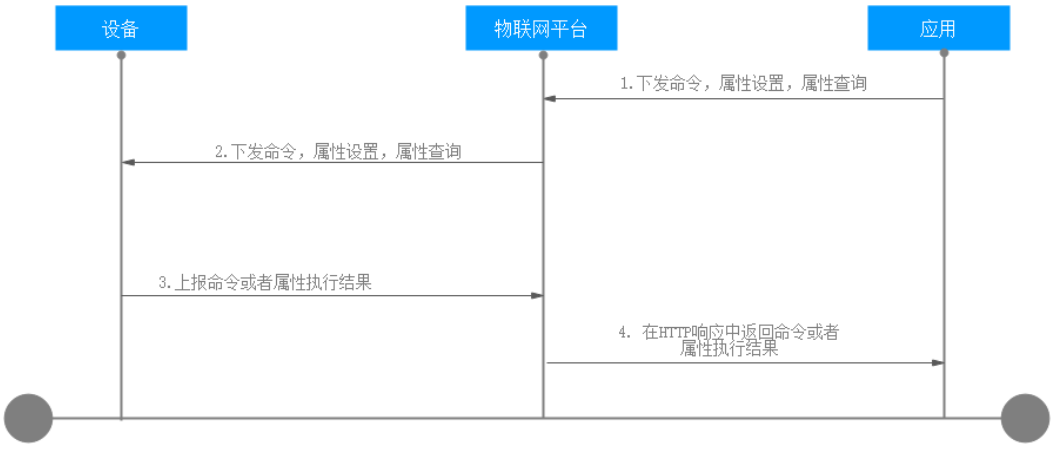
表 5-7 同步命令下发

类型	描述	适用场景	使用示例
单个MQTT设备命令下发	平台向单个设备下发设备控制命令。	对单个设备进行下发设备控制命令。	**单个MQTT设备同步命令下发**
批量MQTT设备命令下发	平台向多个设备下发设备控制命令。可创建批量处理任务，对多个设备进行批量操作	对多个设备进行批量下发设备控制命令。	**批量MQTT设备同步命令下发**

单个 MQTT 设备同步命令下发

属性设置和属性查询分别参考**查询设备属性**和**修改设备属性**接口。

图 5-26 命令下发流程图



1. 应用调用**下发设备命令**接口，下发请求到物联网平台，命令下发消息样例如下：

```
POST https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/commands
Content-Type: application/json
```

```
X-Auth-Token: ********

{
  "service_id" : "WaterMeter",
  "command_name" : "ON_OFF",
  "paras" : {
    "value" : "ON"
  }
}
```

2. 物联网平台根据协议规范下发命令给设备。

   MQTT设备使用**平台命令下发**下行接口对应的Topic用来接收平台下发的命令，消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/commands/request_id={request_id}
数据格式：
{
  "object_device_id": "{object_device_id}",
  "command_name": "ON_OFF",
  "service_id": "WaterMeter",
  "paras": {
    "value": "ON"
  }
}
```

3. 设备执行命令后通过**平台命令下发**上行接口返回命令执行结果，消息样例如下：

```
Topic：$oc/devices/{device_id}/sys/commands/response/request_id={request_id}
数据格式：
{
    "result_code": 0,
    "response_name": "COMMAND_RESPONSE",
    "paras": {
        "result": "success"
    }
}
```

4. 应用侧收到发送HTTP下发命令的同步响应结果。消息样例如下：

```
Status Code: 200 OK
Content-Type: application/json

{
  "command_id" : "b1224afb-e9f0-4916-8220-b6bab568e888",
  "response" : {
    "result_code" : 0,
    "response_name" : "COMMAND_RESPONSE",
    "paras" : {
      "result" : "success"
    }
  }
}
```

批量 MQTT 设备同步命令下发

平台支持通过调用**创建批量任务**接口，对多个MQTT协议设备下发同步命令。下面介绍如何调用**创建批量任务**下发批量命令。

1. 应用调用**创建批量任务**接口，下发请求到物联网平台，下发消息样例如下。

```
POST https://{Endpoint}/v5/iot/{project_id}/batchtasks
Content-Type: application/json
X-Auth-Token: ********
{
  "app_id": "84fb64e43c5f4c6cbec339e52449bcea",
  "task_name": "task123",
  "task_type": "createCommands",
  "targets": [
    "5f2bc9b961e7670469c5ef6d_1997930",
    "5f2bc9b961e7670469c5ef6d_1997931"
  ],
  "document": {
```

```
  "service_id": "water",
  "command_name": "ON_OFF",
  "paras": {
    "value": "ON"
  }
 }
}
```

表 5-8 命令下发创建批量任务参数表

参数	是否必选	描述
app_id	否	资源空间ID。
task_name	是	任务名（自定义）。
task_type	是	批量任务类型。具体可见**创建批量任务**，命令下发中取值有： ● createCommands-批量创建同步命令任务 ● createAsyncCommands-批量创建异步命令任务
targets	否	设备ID数组，执行批量任务的目标。
document	否	命令相关参数，执行任务数据文档，Json格式，Json里面是(K,V)键值对。参考**设备同步命令**

2. 物联网平台向应用返回“201 Created”。
3. 设备订阅下行topic接收命令，并通过上行topic向平台响应命令结果，参考**平台命令下发**。
4. 通过调用**查询批量任务列表**接口查询批量命令下发任务执行情况。

异步命令下发概述

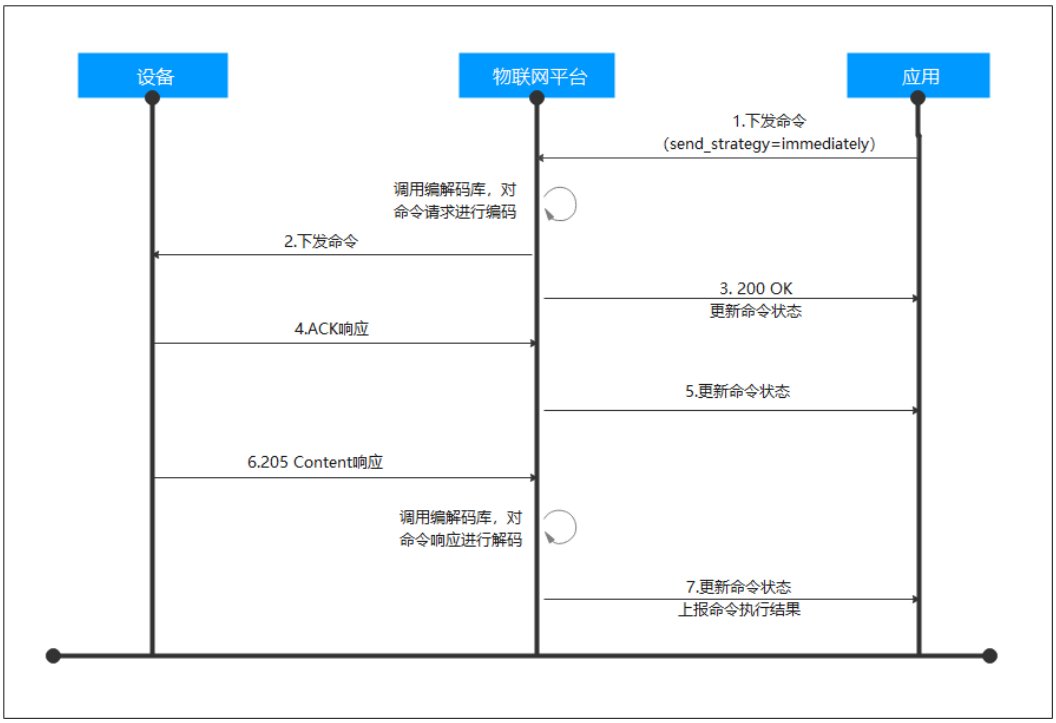
异步命令下发主要用于LwM2M/CoAP设备接入物联网平台，平台或应用侧可通过以下方式进行命令下发：

表 5-9 异步命令下发

类型	描述	适用场景	使用流程
异步命令立即下发	不管设备是否在线，平台收到命令后立即下发给设备。如果设备不在线或者设备没收到指令则下发失败。	适用于实时性要求高的场景。	**异步命令立即下发**
异步命令缓存下发	物联网平台在收到命令后先缓存，等设备上线或者设备上报属性时再下发给设备，如果单个设备存在多条缓存命令，则进行排队串行下发。	适合对命令实时性要求不高的场景，比如配置水表的参数。	**异步命令缓存下发**

异步命令立即下发

图 5-27 LwM2M/CoAP 命令下发流程



按照上述使用流程，进行对应步骤的示例如下：

1. 应用调用**下发异步设备命令**接口，下发请求到物联网平台，携带send_strategy为immediately。消息样例如下：

```
POST https://{endpoint}/v5/iot/{project_id}/devices/{device_id}/async-commands
Content-Type: application/json
X-Auth-Token: ********

{
  "service_id" : "WaterMeter",
  "command_name" : "ON_OFF",
  "paras" : {
    "value" : "ON"
  },
   "expire_time": 0,
   "send_strategy": "immediately"
}
```

2. 物联网平台调用**编解码插件**对命令请求进行编码后，会通过LwM2M协议定义的设备管理和服务实现接口的Execute操作下发命令，消息体为二进制格式。
3. 物联网平台向应用返回200 OK，携带命令状态为**SENT**。（如果设备不在线或者设备没收到指令则下发失败，命令状态为**FAILED**）
4. 设备收到命令后返回ACK响应。
5. 若应用订阅了命令的状态变更通知，物联网平台通过命令状态更新通知接口推送消息给应用，携带命令状态为**DELIVERED**。消息样例如下：

```
Method: POST
request:
Body:
{
    "resource": "device.commmad.status",
    "event": "update",
```

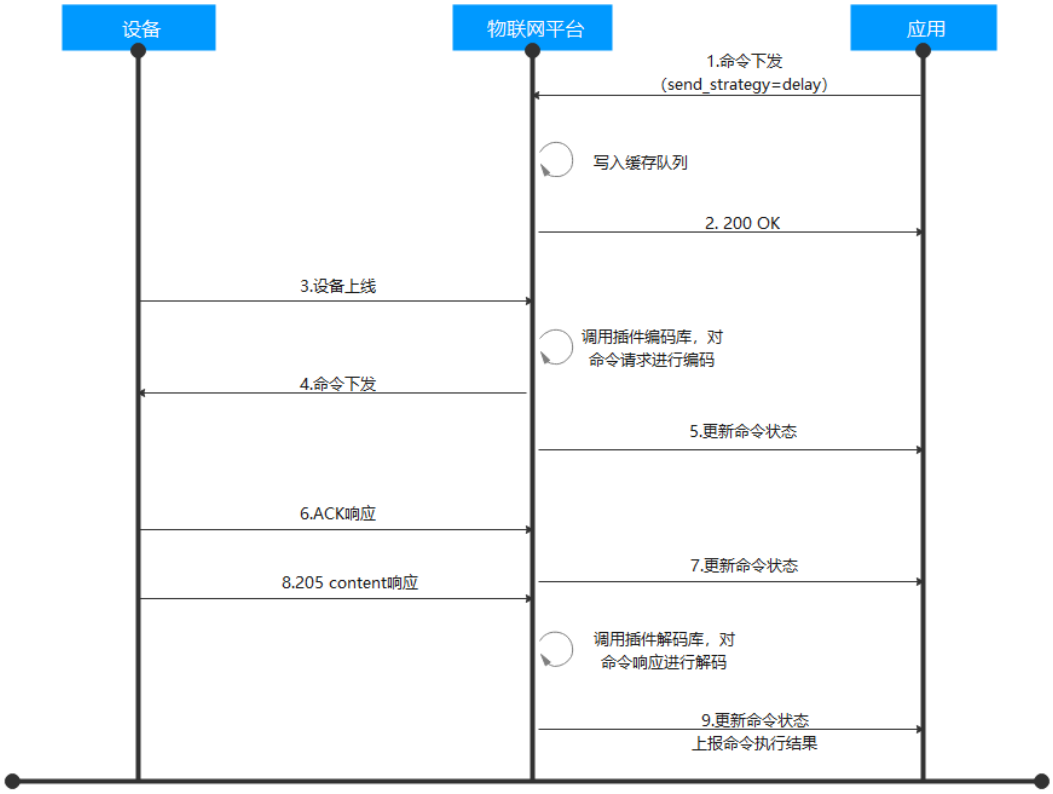
```
    "event_time": "20200811T080745Z",
    "notify_data": {
        "header": {
            "app_id": "8d4a34e5363a49bfa809c6bd788e6ffa",
            "device_id": "5f111a5a29c62ac7edc88828_test0001",
            "node_id": "test0001",
            "product_id": "5f111a5a29c62ac7edc88828",
            "gateway_id": "5f111a5a29c62ac7edc88828_test0001",
            "tags": []
        },
        "body": {
            "command_id": "49ca40af-7e14-4f7b-b97b-78cdd347a6b9",
            "created_time": "20200811T080738Z",
            "sent_time": "20200811T080738Z",
            "delivered_time": "20200811T080745Z",
            "response_time": "",
            "status": "DELIVERED",
            "result": null
        }
    }
}
```


6. 设备执行命令后通过205 Content响应返回命令执行结果。
7. 若应用订阅了命令的状态变更通知，物联网平台会调用编解码插件对设备响应进行解码，然后通过命令状态更新通知接口推送消息给应用，携带命令状态为**SUCCESSFUL**。消息样例如下：


```
Method: POST
request:
Body:
{
    "resource": "device.commmad.status",
    "event": "update",
    "event_time": "20200811T080745Z",
    "notify_data": {
        "header": {
            "app_id": "8d4a34e5363a49bfa809c6bd788e6ffa",
            "device_id": "5f111a5a29c62ac7edc88828_test0001",
            "node_id": "test0001",
            "product_id": "5f111a5a29c62ac7edc88828",
            "gateway_id": "5f111a5a29c62ac7edc88828_test0001",
            "tags": []
        },
        "body": {
            "command_id": "49ca40af-7e14-4f7b-b97b-78cdd347a6b9",
            "created_time": "20200811T080738Z",
            "sent_time": "20200811T080738Z",
            "delivered_time": "20200811T080745Z",
            "response_time": "20200811T081745Z",
            "status": "SUCCESSFUL",
            "result": {
                "resultCode":"SUCCESSFUL",
                "resultDetail": {
                    "value": "ON"
                }
            }
        }
    }
}
```

异步命令缓存下发

图 5-28 LwM2M/CoAP 命令缓存下发流程



1. 应用调用**下发异步设备命令**接口，下发请求到物联网平台，携带send_strategy为delay。
2. 物联网平台将命令写入缓存队列，并上报200 OK，携带命令状态为**PENDING**。
3. 设备上线或设备上报数据到平台。
4. 物联网平台调用**编解码插件**对命令请求进行编码后，根据协议规范下发命令给设备。
5. 若应用订阅了命令的状态变更通知，物联网平台通过命令状态变化通知接口推送消息给应用，携带命令状态为**SENT**。
6. 后续流程请参考“命令立即下发”的步骤**4**到步骤**7**。

LwM2M/CoAP 设备命令执行状态说明

命令执行状态以及状态变化机制如下所示。

图 5-29 LwM2M/CoAP 命令下发状态

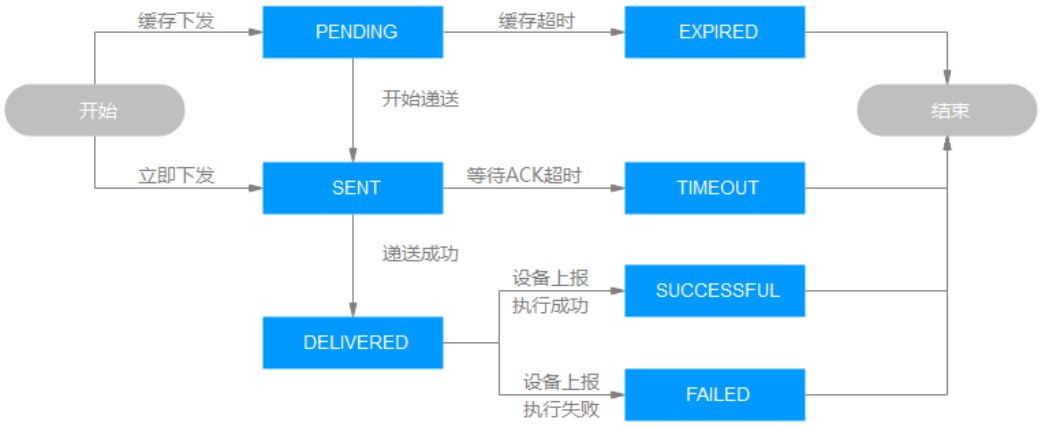


表 5-10 LwM2M/CoAP 命令执行状态

命令执行状态	说明
等待（PENDING）	• LwM2M/CoAP设备采用缓存下发模式下发命令时，如果设备未上报数据，物联网平台会将命令进行缓存，此时任务状态为“等待”状态。 • LwM2M/CoAP设备采用立即下发模式下发命令时，无此状态。
超期（EXPIRED）	• LwM2M/CoAP设备采用缓存下发模式下发命令时，如果在设置的超期时间内，物联网平台未将命令下发给设备，则状态变更为“超期”。超期时间会根据应用侧接口中携带的expireTime为准，如果未携带，默认24h。 • LwM2M/CoAP设备采用立即下发模式下发命令时，无此状态。
已发送（SENT）	• LwM2M/CoAP设备采用缓存下发模式下发命令时，设备上报数据，物联网平台会将缓存的命令发送给设备，此时状态会由“等待”变为“已发送”。 • LWM2M/CoAP设备采用立即下发模式下发命令时，如果设备在线，状态为“已发送”。
超时（TIMEOUT）	LwM2M/CoAP设备收到命令后，物联网平台在180秒内未收到设备反馈的收到命令响应，此时状态会变为“超时”。
已送达（DELIVERED）	物联网平台收到设备反馈的已收到下发命令响应后，状态变为“已送达”。
成功（SUCCESSFUL）	如果设备在执行完命令后，会给物联网平台反馈命令执行成功的结果，将任务状态变更为“成功”。
失败（FAILED）	• 如果设备在执行完命令后，会给物联网平台反馈命令执行失败的结果，将任务状态变更为“失败”。 • LwM2M/CoAP设备采用立即下发模式下发命令时，如果设备离线，状态为“失败”。

平台命令下发使用示例

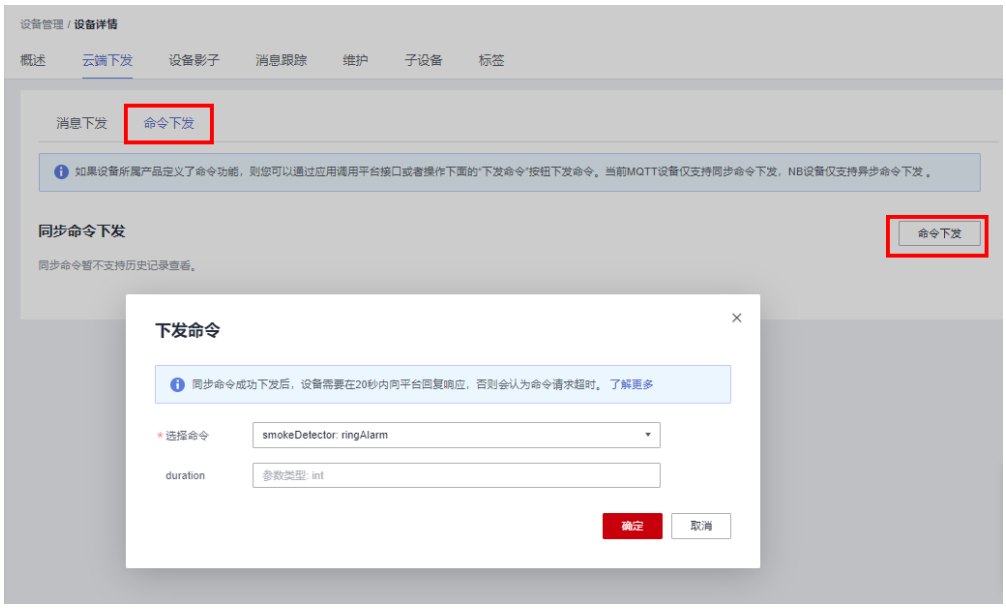
步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 单击“设备”，在设备列表中，单击具体的设备进入到设备的详情页面。

步骤3 在“云端下发>命令下发”页签，根据设备协议的不同，界面也有所不同。

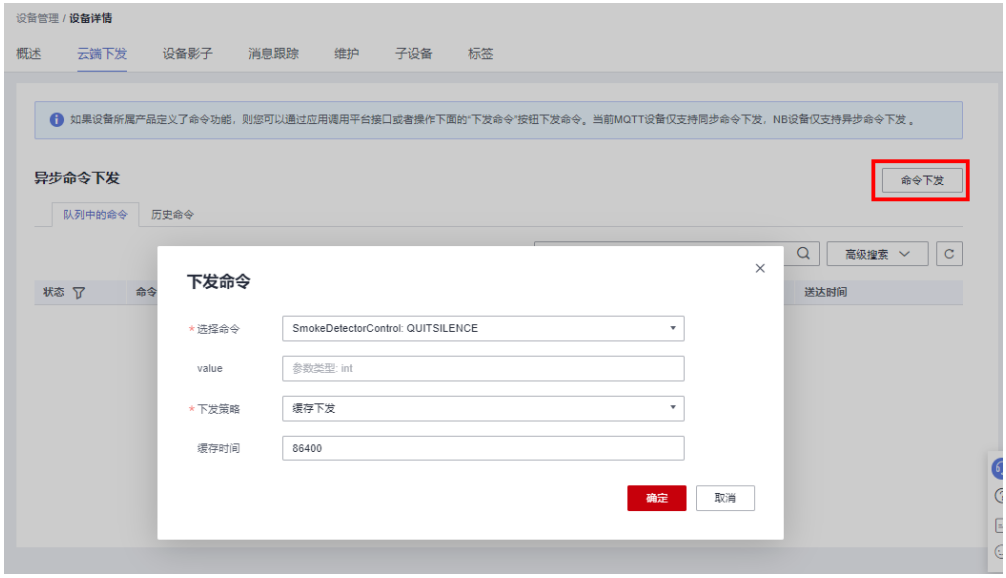
- MQTT设备仅支持同步命令下发，单击右侧的“命令下发”，在弹出的窗口中选择需要下发的命令并设置命令参数。

图 5-30 同步命令下发



- LwM2M/CoAP设备仅支持异步命令下发，单击右侧的“命令下发”，在弹出的窗口中选择需要下发的命令并设置命令参数。您可以选择立即下发或者缓存下发。

图 5-31 异步命令下发



----结束

说明

- 消息跟踪可以查看下发的历史命令列表，通过该功能详细查看命令下发任务的创建时间、平台发送命令的时间、送达的时间、发送的状态等信息，便于用户了解命令的**执行状态**。
- 命令下发支持通过调用**查询设备命令**接口，在物联网平台查询下发命令的状态及内容信息，以了解命令的执行情况。

配置应用侧使用JAVA SDK进行同步命令下发的开发步骤如下，本示例使用的开发环境为JDK 1.8及以上版本。SDK代码获取：**SDK下载**

步骤1 配置Maven依赖。

```
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-core</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-iotda</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

步骤2 以同步命令下发为例，样例如下：

```
public class CommandSolution {
   // REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
   private static final String REGION_ID = "<YOUR REGION ID>";
   // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
   private static final String ENDPOINT = "<YOUR ENDPOINT>";
   // 标准版/企业版：需自行创建Region对象
   public static final Region REGION_CN_NORTH_4 = new Region(REGION_ID, ENDPOINT);
   public static void main(String[] args) {
      String ak = "<YOUR AK>";
      String sk = "<YOUR SK>";
      String projectId = "<YOUR PROJECTID>";
      // 创建认证
      ICredential auth = new BasicCredentials().withDerivedPredicate(AbstractCredentials.DEFAULT_DERIVED_PREDICATE)
            .withAk(ak)
```

```
            .withSk(sk)
            .withProjectId(projectId);
        // 创建IoTDAClient实例并初始化
        IoTDAClient client = IoTDAClient.newBuilder().withCredential(auth)
            // 基础版：请选择IoTDARegion中的Region对象
            //.withRegion(IoTDARegion.CN_NORTH_4)
            // 标准版/企业版：需自行创建Region对象
            .withRegion(REGION_CN_NORTH_4).build();
        // 实例化请求对象
        CreateCommandRequest request = new CreateCommandRequest();
        request.withDeviceId("<YOUR DEVICE_ID>");
        DeviceCommandRequest body = new DeviceCommandRequest();
        body.withParas("{\"value\":\"1\"}");
        request.withBody(body);
        try {
            CreateCommandResponse response = client.createCommand(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.printStackTrace();
        } catch (RequestTimeoutException e) {
            e.printStackTrace();
        } catch (ServiceResponseException e) {
            e.printStackTrace();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getRequestId());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

表 5-11 参数说明

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
projectId	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.java**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**平台对接信息**。
REGION_ID	如果是上海一，请填写“cn-east-3”；如果是北京四，请填写“cn-north-4”;如果是华南广州，请填写“cn-south-4”。
ENDPOINT	请在控制台的“总览”界面的“接入信息”中查看“应用接入”的https接入地址。
DEVICE_ID	下发消息的设备ID，用于唯一标识一个设备，在注册设备时由物联网平台分配获得。 取值范围：长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。

----结束

配置设备侧使用JAVA SDK进行同步命令下发的开发步骤如下，本示例使用的开发环境为JDK 1.8及以上版本。

步骤1 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

步骤2 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤3 设置命令下发回调函数、发送响应。

```
 client.setCommandListener(new CommandListener() {
    @Override
    public void onCommand(String requestId, String serviceId, String commandName, Map<String, Object>
paras) {
        log.info("onCommand, serviceId = " +serviceId);
        log.info("onCommand , name = " + commandName);
        log.info("onCommand, paras =  " + paras.toString());

        //处理命令用户自定义

        //发送命令响应
        device.getClient().respondCommand(requestId, new CommandRsp(0));
    }
});
```

----结束

测试验证步骤如下：

步骤1 在设备接入控制台，进入“设备 > 所有设备”，单击具体设备，启动“消息跟踪”。

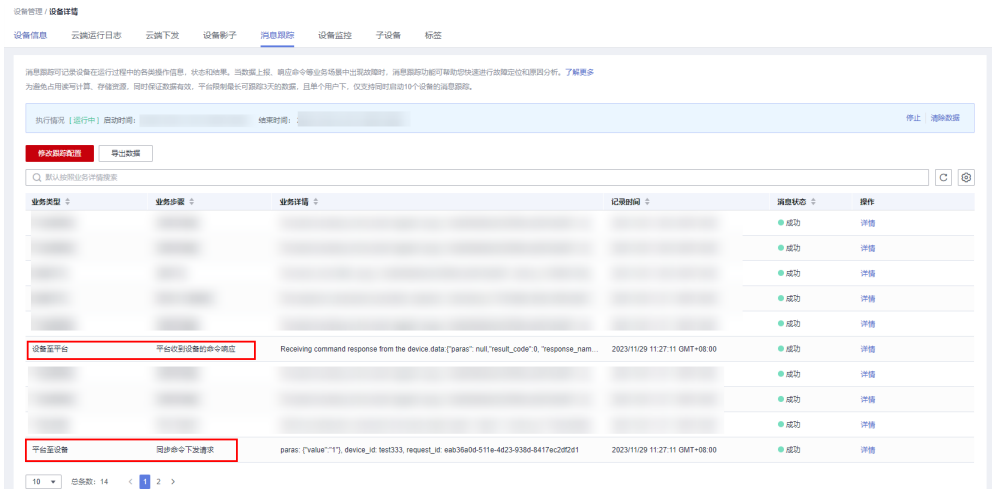
步骤2 先运行设备侧 SDK代码，使设备上线。

步骤3 运行应用侧代码，当设备接收到设备下发的命令后，进行数据处理及响应。设备侧收到的命令下发样例如下：

图 5-32 设备侧命令下发成功结果

步骤4 “消息跟踪”显示结果如下:

图 5-33 平台消息跟踪命令下发结果



----结束

5.3 自定义 Topic 通信

5.3.1 自定义 Topic 通信概述

概述

使用MQTT协议接入的设备，平台和设备之间基于Topic进行通信。Topic分为系统Topic和自定义Topic。系统Topic为平台预置的基本通信Topic，自定义Topic是可以根据实际业务需要用户自行定义的Topic，客户可根据使用场景进行选择使用。值得注意的是，自定义Topic与系统Topic的消息上报一样，在平台都进行透传（平台不主动解析数据具体内容）。

表 5-12 topic 分类

Topic类别	描述	使用场景
系统Topic	平台预先定义了各种设备和平台通信的Topic，具体Topic列表和功能说明可参考**Topic定义**。	消息上报、属性上报、命令下发、事件类主题。

Topic类别	描述	使用场景
自定义Topic	用户可以自定义Topic，设备和平台间可以基于用户自定义的Topic进行通信。 **自定义topic分类：** • **$oc开头的自定义Topic**：在产品中定义需要使用的Topic，这类Topic有$oc/devices/{device_id}/user/前缀，消息上报或者消息下发时平台会校验Topic是否在产品中定义，未在产品中定义的Topic会被平台拒绝。 • **非$oc开头的自定义Topic**：如/aircondition/data/up进行消息通信，平台会通过**Topic策略**校验主题权限，可以用于进行Topic的消息上下行通信。	在业务需要特定Topic的场景。比如说**端到端通信**、**广播通信**、设备迁移等。

使用场景

- 设备端向自定义Topic发布消息；应用端通过**数据转发**功能实现数据平滑流转至消息中间件、存储、数据分析、业务应用。
- 应用端调用**下发设备消息**接口，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。
- **端到端通信**、**广播通信**、设备迁移。

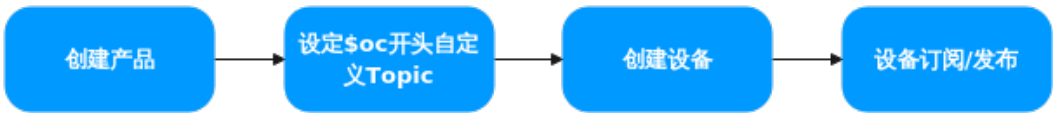
使用限制

- 每个产品模型最多支持50个自定义Topic。
- 自定义Topic只支持消息通信，不支持属性通信。
- MQTT自定义Topic支持的最大长度为128字节。

5.3.2 $oc 开头自定义 Topic 通信使用说明

使用流程&操作步骤

图 5-34 $oc 开头自定义 topic 通信

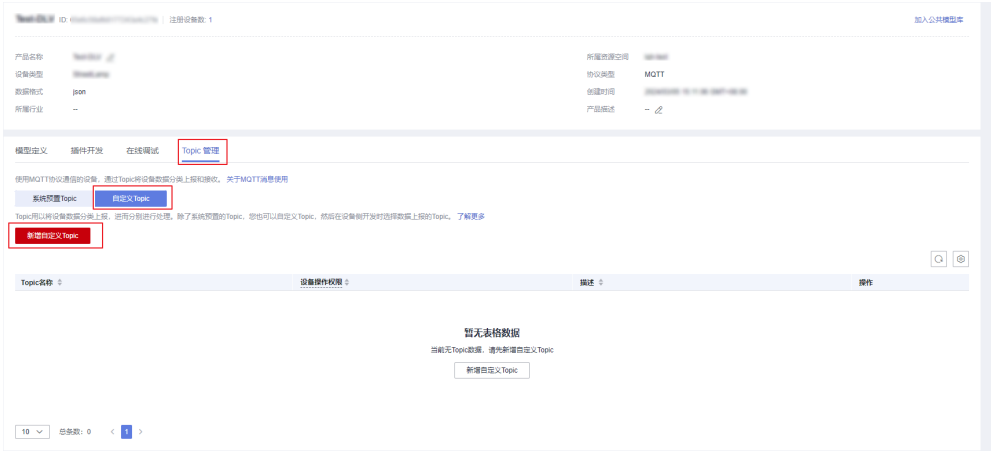


步骤1 创建产品：访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。参考**创建产品**流程。

步骤2 设定$oc开头自定义Topic。在产品详情页中创建一个自定义Topic，Topic前缀固定为：$oc/devices/{device_id}/user/。

1. 选择MQTT协议类产品，在产品详情页中，选择“Topic管理 > 自定义Topic”，单击“新增自定义Topic”。

图 5-35 Topic 管理-自定义 Topic



2. 在弹出的页面中，选择设备操作权限，填写Topic名称。

图 5-36 Topic 管理-新增自定义 Topic

表 5-13 页面参数说明

参数名称	描述
Topic名称	Topic的前缀已经规定好，固定为：$oc/devices/{device_id}/user/，其中{device_id}为标识符变量，实际发布和订阅过程中需要替换为实际的设备ID。用户自定义Topic的格式必须以“/”进行分层。 长度限制为1-64位，只允许输入数字、大小写字母、下划线、斜杠符。其中，斜杠符不能连续。 **说明** 自定义Topic不支持自定义变量，例如$oc/devices/{device_id}/user/setting/{type}，其中的{type}为自定义变量，当前不支持这种使用方式。
设备操作权限	– 发布：设备侧消息上报时，可按配置中自定义的Topic进行消息上报；数据流转时，设备消息中会携带Topic参数标识该消息从哪个Topic上报。 – 订阅：应用侧消息下发时，可在消息内容中指定Topic；消息发往设备时，可以根据指定的Topic下发。 – 发布和订阅：同时具备发布和订阅的权限。
描述	关于该Topic的描述。

3. 单击“确定”，完成新增自定义Topic。自定义Topic添加成功后，您可以在自定义Topic列表执行修改和删除操作。

步骤3 创建设备：在该产品下创建设备。创建的设备将继承产品设定的自定义Topic。详情可见：**创建设备**流程。

步骤4 设备订阅/发布：查看**使用自定义Topic进行通信**的最佳实践，了解自定义Topic的发布与订阅的使用。

----结束

设备侧 JAVA SDK 使用示例

设备端可以通过集成华为云IoT提供的**设备端SDK**快速连接华为云IoTDA，并进行消息上报。以下示例为通过JAVA SDK实现设备连接到华为云IoTDA进行发布、订阅自定义Topic。以订阅"$oc/devices/" + device.getDeviceId() + "/user/wpy"为例。

1. 配置设备侧SDK的Maven依赖。

```
<dependency>
   <groupId>com.huaweicloud</groupId>
   <artifactId>iot-device-sdk-java</artifactId>
   <version>1.1.4</version>
</dependency>
```

2. 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/iot_02_1004.html#section3
URL resource = MessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
```

```
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//初始化设备连接
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

3. 上报设备消息：

```
device.getClient().publishRawMessage(new RawMessage( "$oc/devices/" + device.getDeviceId() + "/
user/wpy", "hello", 1), new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        System.out.println("reportDeviceMessage success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        System.out.println("reportDeviceMessage fail: " + var2);
    }
});
```

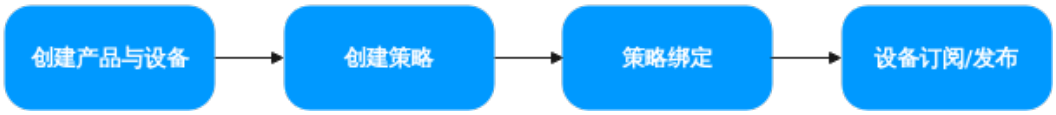
4. 订阅topic：

```
device.getClient().subscribeTopic(new RawMessage("$oc/devices/" + device.getDeviceId() + "/user/
wpy", new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        System.out.println("subscribeTopic success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        System.out.println("subscribeTopic fail: " + var2);
    }
}, 0);
```

5.3.3 非$oc 开头自定义 Topic 通信使用说明

使用流程&操作步骤

图 5-37 非$oc 开头自定义 topic 通信



说明

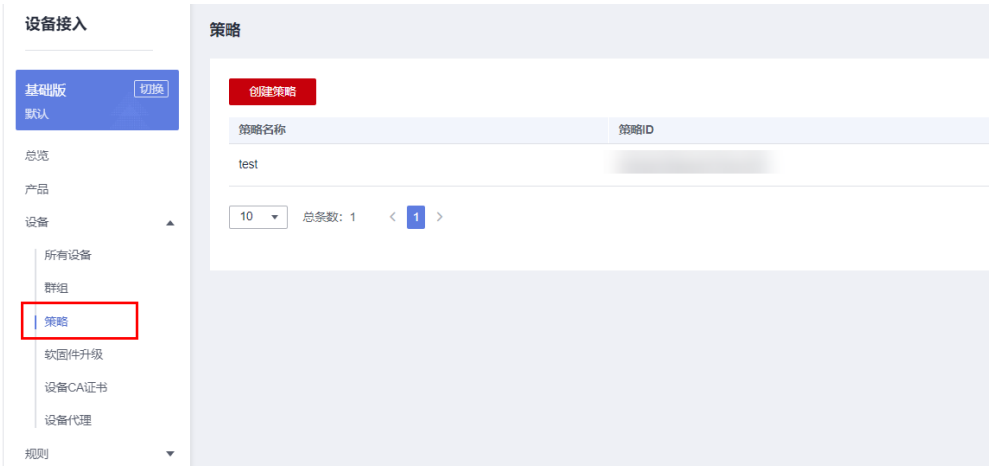
- 为了适配新老客户的使用，策略默认放通所有“非$oc开头的自定义Topic通信”。新增的资源空间会默认加入策略“system_default_policy”，system_default_policy策略会允许所有Topic的订阅与发布。当业务场景不适用时，可以删除该策略。
- 值得注意的是，策略只会限制“非$oc开头的自定义Topic通信”。“$oc开头的自定义Topic”权限由产品下的设定决定。
- 现如今广州、北京四、上海局点不支持策略，可跳过策略（步骤2、3）、直接使用。

步骤1 在平台创建产品与设备。详情可见：**创建产品**流程、**创建设备**流程。

步骤2 创建策略。用于控制放通哪些Topic进行订阅/发布。（可选）

1. 进入策略页面。访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。在左侧导航栏选择“设备 > 策略”。

图 5-38 设备策略-进入界面



2. 创建策略。在策略界面单击“创建策略”，按照业务填写策略参数，填写完成后单击“生成策略”。下图以允许主题“/v1/test/hello”发布及订阅为例。

表 5-14 参数说明

参数说明	
所属资源空间	下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
策略名称	自定义，如PolicyTest。长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
资源	在MQTT主题发布与订阅中，需要以“topic:”作为参数前缀。比如说：禁止订阅/test/v1，则该参数填写 topic:/test/v1。
操作	值为发布或订阅。发布代表MQTT设备端Publish请求，订阅代表MQTT设备端Subscribe请求。
权限	值为允许或拒绝。用于允许或拒绝某topic的发布或订阅。

步骤3 绑定策略目标。用于指定设备/产品/资源空间可以使用该策略。策略可以从“资源空间”、“产品”、“设备”这3个范围进行绑定，被绑定的设备将遵循策略的要求允许或拒绝某Topic的发布或订阅。（可选）

图 5-39 设备策略-绑定设备

表 5-15 参数说明

参数说明	
设备目标类型	下拉选择设备目标类型。类型有“资源空间”、“产品”、“设备”三种。这三种类型并不是互斥的，可以同时存在，比如说：绑定产品A与设备C（C是产品B下的设备）。 • 资源空间：实现多业务应用的分域管理，绑定后所选资源空间下的所有设备都将匹配该策略。可选择多个资源空间绑定。 • 产品：一个产品下一般有多个设备，绑定后所选产品下的所有设备都将匹配该策略，比起资源空间，绑定范围更小。可绑定一个或多个不同资源空间下的产品。 • 设备：绑定策略目标的最小单位，可绑定一个或多个不同资源空间、不同产品的设备。
策略目标	选择对应的“策略目标类型”后，在“策略目标”的参数中会显示可选的数据，勾选需要绑定的即可。

步骤4 设备订阅/发布。定义成功后可以发布、订阅该Topic。没有绑定策略成功的自定义Topic无法订阅/发布。

----结束

设备侧 JAVA SDK 使用示例。

设备端可以通过集成华为云IoT提供的**设备端SDK**快速连接华为云IoTDA，并进行消息上报。以下示例为通过JAVA SDK实现设备连接到华为云IoTDA对自定义Topic“/test/deviceToCloud”进行发布、订阅。

1. 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

2. 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = MessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//初始化设备连接
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

3. 上报设备消息：

```
device.getClient().publishRawMessage(new RawMessage("/test/deviceToCloud", "hello", 1), new
ActionListener() {
    @Override
    public void onSuccess(Object context) {
        System.out.println("reportDeviceMessage success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        System.out.println("reportDeviceMessage fail: " + var2);
    }
});
```

4. 订阅topic：

```
device.getClient().subscribeTopic(new RawMessage("/test/deviceToCloud", new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        System.out.println("subscribeTopic success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        System.out.println("subscribeTopic fail: " + var2);
    }
}, 0);
```

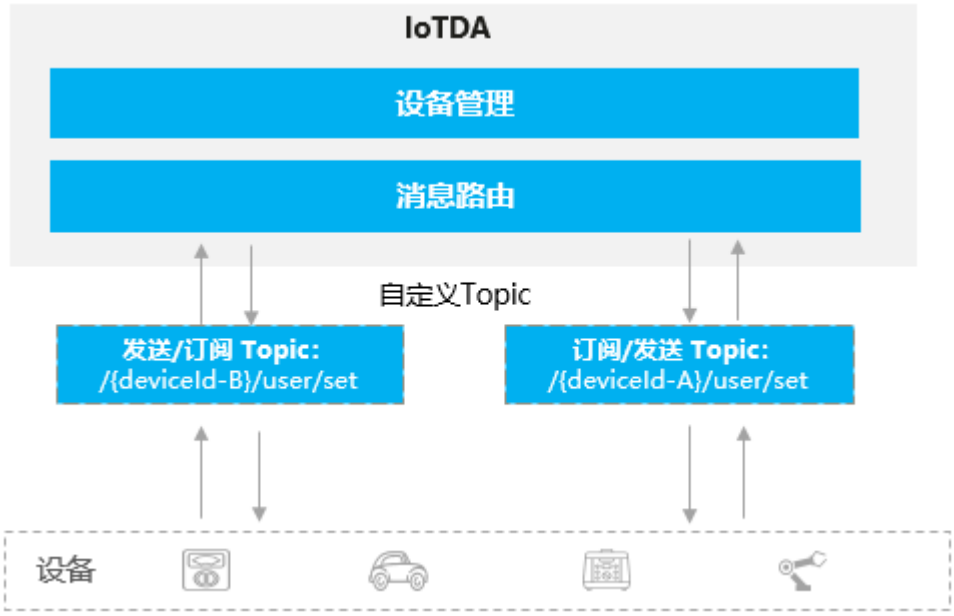
5.4 设备间消息通信（M2M）

5.4.1 设备间消息通信概述

概述

M2M（Machine-to-Machine），物联网平台支持基于MQTT协议实现设备间的消息通信。设备的连接和通信请求都交由平台承担，客户只需要关注自己的业务实现。可以实现设备间1到1、1到N、N到N的使用。

图 5-40 业务流程图



说明

- M2M通信过程中通过Pub接口发的消息和Sub接口接收的消息会算入计费消息数，不产生其他额外费用。

使用场景

- 即时聊天场景，发送方和接收方进行消息通信。
- 智能家居控制场景，手机APP和智能设备之间进行消息通信。
- 设备联动，设备间进行数据传输与消息通信。

使用限制

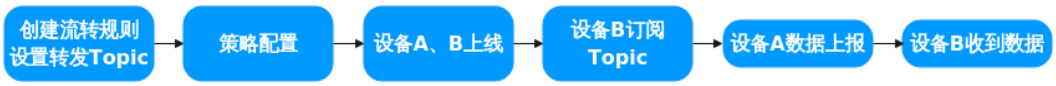
- 基础版不支持。
- 仅支持使用MQTT协议接入的设备。
- Topic长度不大于128个字节。
- MQTT单条发布消息最大长度不超过1MB。
- 同一个Topic最多允许被1,000个设备订阅。
- 单个MQTT连接的最大订阅数为100个，自定义Topic的订阅数不超过50个。
- 单个MQTT连接每秒最大上行消息数为50条。

5.4.2 设备间消息通信使用说明

使用流程

以下流程主要以设备间一对一为例：

图 5-41 M2M 使用流程



步骤1 创建流转规则，设置转发Topic：在控制台界面创建M2M流转规则并设置转发Topic。

步骤2 策略配置：在控制台界面进行策略配置。通过策略配置允许发送、接收数据的设备进行发布及订阅。

步骤3 设备A、B鉴权：设备发起连接鉴权（MQTT设备），鉴权参数填写请参考：**设备连接鉴权**。

步骤4 设备B订阅Topic：设备对云服务端进行主题订阅。设备B订阅的Topic为创建流转规则中设置的Topic。若订阅成功，平台返回订阅成功ACK。

步骤5 设备A数据上报：设备对云服务端进行主题发布。若发布成功，平台返回发布成功ACK。

步骤6 设备B收到数据：若转发成功，设备B将收到设备A发送的数据。

----结束

操作步骤

以下示例主要针对平台中创建流转规则配置。通过修改流转规则可以实现不同场景的应用。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。在左侧导航栏“规则”中单击“数据转发”，可到数据转发界面。

图 5-42 数据转发页面

步骤2 单击“创建规则”按照业务具体填写需要转发的数据参数，填写完成后单击“创建规则”。参数值可参考下图。

图 5-43 创建数据转发规则

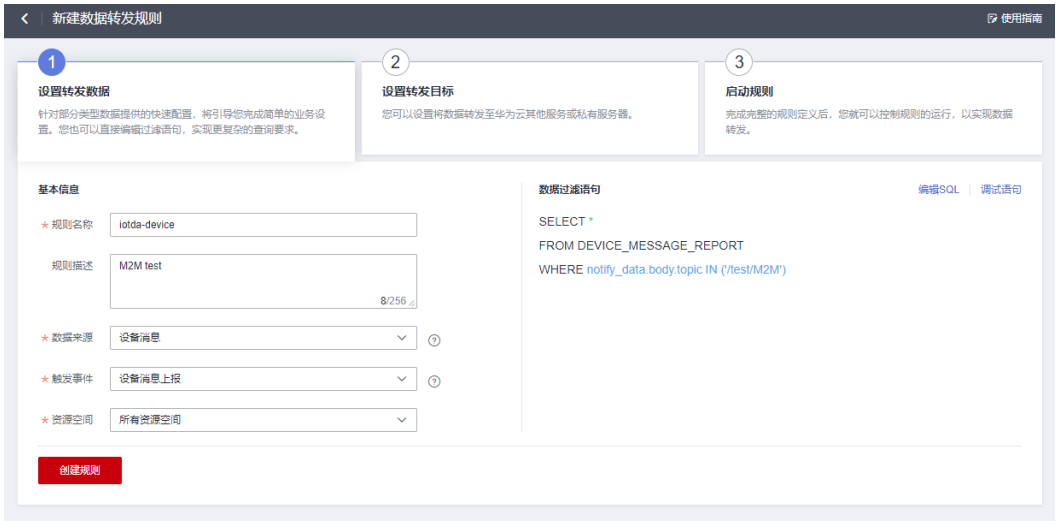


表 5-16 创建数据转发规则-参数说明

参数说明	
规则名称	自定义，如test。长度不超过256，只允许中文、字母、数字、以及_?'#().,&%@!-等字符的组合。
规则描述	自定义，对该规则的描述。
数据来源	转发规则的数据来源，下拉可选择多种数据来源。在使用M2M时，请选择“设备消息”。
触发事件	不同的数据来源有不同的触发事件，若使用M2M，请选择“设备消息上报”。
资源空间	下拉选择所属的资源空间。可以选择所有资源空间，如无对应的资源空间，请先创建**资源空间**。
数据过滤语句	使用SQL语句可以进行数据筛选，详情可见：**SQL语句**。图片中在WHERE中填入notify_data.body.topic IN ('/test/M2M')，代表只有Topic为“/test/M2M”的数据进行数据转发。

步骤3 在创建数据转发规则的第二步，可以添加转发到的目标，包括设置转发后的Topic、缓存时间等。若使用M2M，请将转发目标设置为“设备”，按照业务具体填写参数后单击“确定”。

图 5-44 设置转发目标

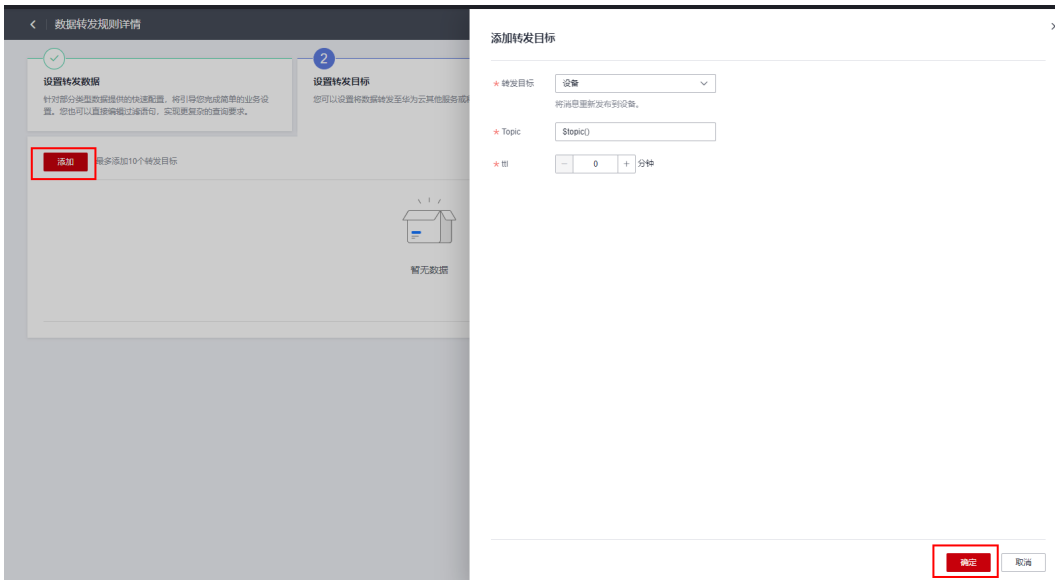


表 5-17 设置转发目标-参数说明

参数说明	
转发目标	下拉选择转发目标，在设备间消息通信中，请选择为“设备”。
Topic	自定义，长度不超过128个字符，可以以$和/开头，不可以$和/符号结尾；不允许有a-zA-Z0-9() ',-.:=@;_!*'%?+\以外的符号。
ttl	数据缓存时间。当设备不在线时，数据会进行缓存（当ttl为0时不缓存），当设备上线时再进行下发。输入值范围在0~1440（一天）分钟，且值为5的倍数。

步骤4 启动规则。在创建数据转发规则的第三步，单击启动，完成规则设置。

图 5-45 启动规则

----结束

5.4.3 设备间消息通信使用示例

前置步骤

1. 创建产品和设备
   - 创建产品。访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。单击左侧导航栏“产品”，在页面中单击红色按钮“创建产品”。根据页面提示填写参数，然后单击“确定”，完成产品的创建。详情可见：**创建产品**。
   - 创建设备。在设备接入控制台，单击左侧导航栏“设备 > 所有设备”，单击页面右上角的“注册设备”。根据页面提示填写参数，然后单击“确定”，完成设备的创建。详情可见：**注册单个设备**。

配置流转规则

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 左侧导航栏单击“规则 > 数据转发”，进入数据转发界面。

图 5-46 数据转发界面

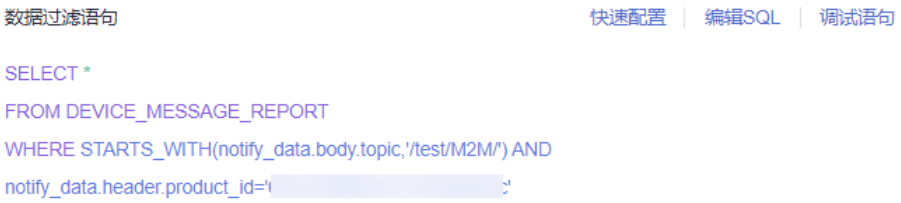
步骤3 在“数据转发>规则列表”界面单击“创建规则”，填写规则参数，设置转发数据，配置数据过滤语句为：STARTS_WITH(notify_data.body.topic,'/test/M2M/')。

图 5-47 创建数据转发规则

说明

- 图2 创建数据转发规则的设置代表：在资源空间XXX中的所有设备，Topic中包含/test/M2M/的消息上报，都会触发该流转规则，转发到设置的转发目标。
- 若希望指定某个设备上报的数据进行流转，可在SQL语句中添加：AND notify_data.header.device_id='${对应的设备ID}'。
- 若希望指定某种产品上报的数据进行流转，可在SQL语句中添加：AND notify_data.header.product_id='${对应的产品ID}'。

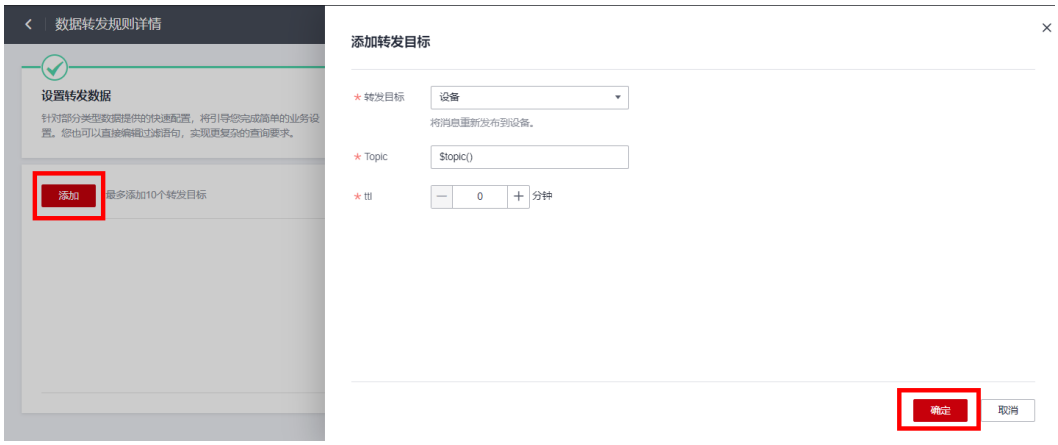
图 5-48 例子



- 关于SQL的设置，详情可见：SQL语句。

步骤4 设置转发目标。单击“添加”，设置转发目标为“设备”，Topic为：“$topic()”（转发后Topic不变），ttl设置为5分钟（数据缓存5分钟）。设置完成后单击“确定”。

图 5-49 添加转发目标



步骤5 启动规则。单击启动，完成规则设置。

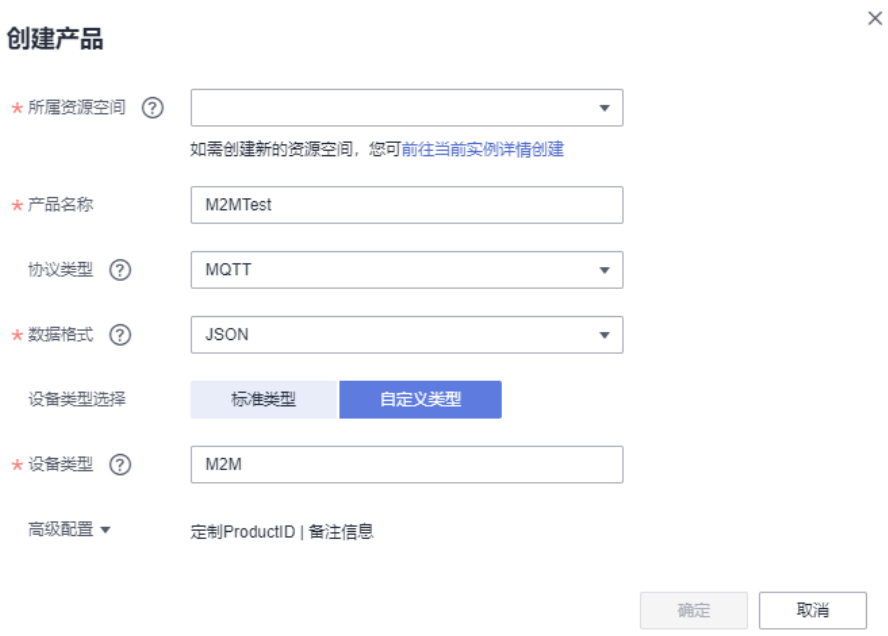
图 5-50 启动规则

----结束

验证

1. 使用设备间消息通信功能时，具体步骤如下：
   a. 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。
   b. **创建产品**，产品的“协议类型”选择“MQTT”。

   图 5-51 创建产品



2. 在步骤1中创建的产品下分别注册设备A（test111）和设备B（test222），详细步骤可参考**注册单个设备**。

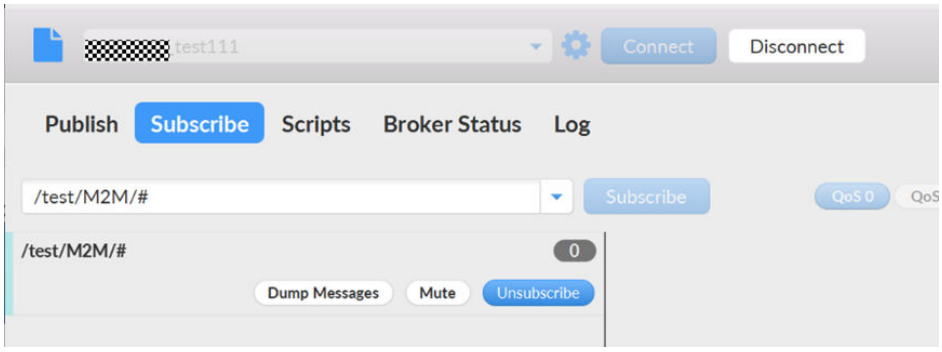
图 5-52 注册设备

3. 您可以根据自己的业务场景来实现设备间消息通信。下面以MQTT.fx为例说明如何进行设备间消息通信：

   a. 打开两个MQTT.fx，分别模拟设备A（test111）、B（test222）。

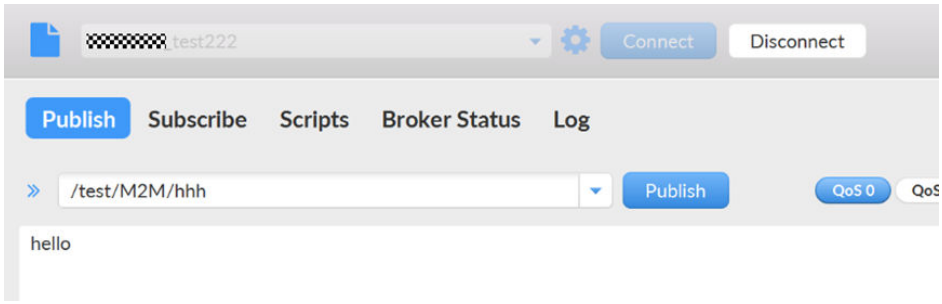
   b. 设备B在Subscribe页面中输入Topic“ /test/M2M/#”后，单击“Subscribe”订阅。

   图 5-53 设备 B（test222）在 Subscribe 页面中输入 topic

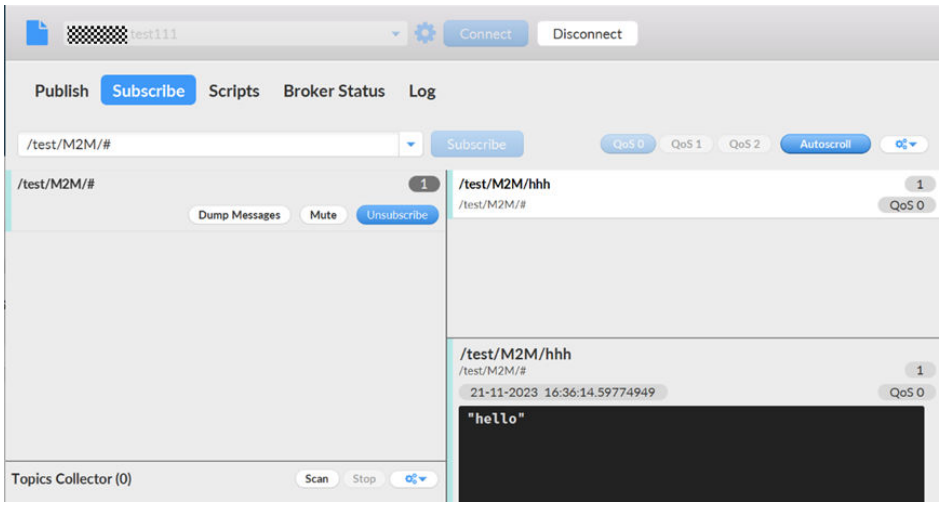
   c. 设备A（test111）向设备B（test222）发送消息，在设备A（test111）的MQTT.fx的“Publish”页面中，输入topic“/test/M2M/${任意单词}”（其中“${任意单词}”替换成任意单词），在内容输入框中输入要发送的消息（如：hello）单击“Publish”即可发送。

图 5-54 设备 A Publish 页面输入消息



在设备B的Subscribe页面可以看到接收的消息如下：

图 5-55 设备 B Subscribe 页面展示



5.5 设备 Topic 策略

5.5.1 设备 Topic 策略概述

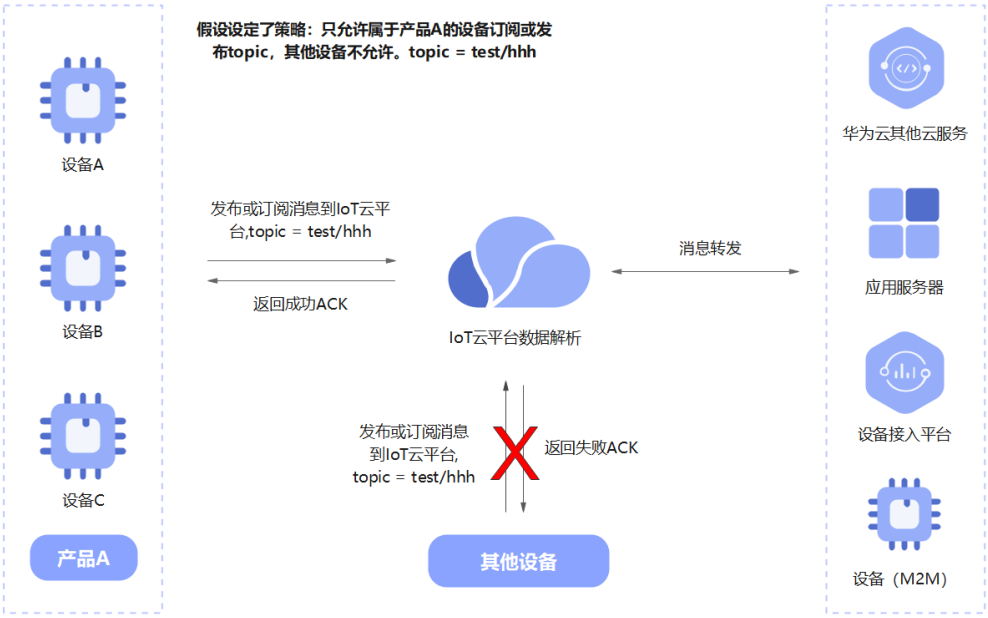
概述

设备策略主要用于对发布/订阅的非$oc开头自定义topic中的数据进行传输限制。通过灵活访问的控制模型，提供了基于用户角色的访问控制，能够管理客户端发布/订阅主题的授权。借助策略功能，可以用于管理一个或多个设备/产品/群组发布、订阅的权限，以保证非$oc开头的自定义Topic的通信安全。设备Topic策略用于发布、订阅机制的协议，比如说设备侧的MQTT、MQTTS协议。现如今华南友好局点、海外局点支持。

须知

对于使用IOTDA的用户，新增的资源空间会默认加入策略“system_default_policy”，system_default_policy策略会允许该资源空间下所有设备的非$oc开头的自定义Topic的订阅与发布。当业务场景不适用时，可以删除该策略。

图 5-56 策略概念图



使用场景

- 群组通信场景，如设备A、设备B、设备C属于一个群组，只允许设备A、设备B、设备C订阅该群组的Topic，其他设备不允许订阅该Topic。
- 用于划分发布/订阅区域。每个区域可以相互通信，其他区域不可访问的情况。

限制

- 一个租户配置的策略数量不超过50个。
- 用于非$oc开头的自定义Topic，对系统主题及$oc开头的自定义Topic无效。
- 一个策略配置的策略文档大小不大于10KB，策略文档数目不大于10条。
- 单个设备或产品最多绑定5个策略。
- 单个设备（客户端）订阅Topic的数量不大于50。
- 设备订阅的Topic的字节长度不超过128字节。
- Topic发布订阅只支持Qos0，Qos1。

5.5.2 设备 Topic 策略使用前必读

设备策略主要用于对发布/订阅的非oc开头自定义topic中的数据进行传输限制。能够管理客户端发布/订阅主题的授权。借助策略功能，可以保证非$oc开头的自定义Topic的

通信安全。设备Topic策略用于发布、订阅机制的协议，比如说设备侧的MQTT、MQTTS协议。现如今华南友好局点支持。

策略通配符

策略中具有不同的通配符，使用前需注意。在策略中，“*”表示字符的任意组合，问号“？”表示任何单个字符，而通配符“+”和“#”被视为没有特殊含义的字符。

表 5-18 策略通配符

通配符	是MQTT通配符	策略配置是否适用	MQTT中主题示例	适用于MQTT主题示例的策略示例
#	是	否	test/#	不适用，“#”被视为没有特殊含义的字符。
+	是	否	test/+/some	不适用，“+”被视为没有特殊含义的字符。
*	否	是	不适用，“*”被视为没有特殊含义的字符。	test/* test/*/some
?	否	是	不适用，“？”被视为没有特殊含义的字符。	test/????/some test/set?????/some

表 5-19 定义策略通配符示例

发布/订阅的Topic	策略Topic定义	解释
假设设备需要订阅/发布以下Topic: “test/topic1/some” “test/topic2/some” “test/topic3/some”	“topic:test/topic?/some”	在发布、订阅的Topic中可以发现有共通点：“test/topic” + 某一字符 + “/some”，而在策略定义中“?”代表某一字符。所以策略Topic可以定义为“topic:test/topic?/some”

发布/订阅的Topic	策略Topic定义	解释
假设设备需要订阅/发布以下Topic: “test/topic1/pub/some” “test/topic2/sub/some” “test/topic3/some”	“topic:test/topic*/some”	在发布、订阅的Topic中可以发现有共通点：“test/topic” + 一个或多个字符 + “/some”，而在策略定义中“*”代表多个或一个字符。所以策略Topic可以定义为“topic:test/topic*/some”

策略变量

在策略中定义resource时，如果不知道对设备资源或条件键的精确值，可以使用策略变量作为占位符，进行发布/订阅主题筛选。策略变量在校验MQTT的主题时，会把变量变为接入设备对应的ID值，再进行匹配。

变量使用前缀“$”标记，后面跟一对大括号“{ }”，其中包含请求中值的变量名称。如下表，假设MQTT设备是在客户端ID为test_clientId，产品ID为test_productId，设备ID为test_deviceId。

表 5-20 策略变量

策略变量	描述	MQTT中主题示例	适用于MQTT主题示例的策略示例
${devices.deviceId}	设备ID	test/test_deviceId/topic	test/${devices.deviceId}/topic
${devices.clientId}	客户端ID	test/test_clientId/topic	test/${devices.clientId}/topic
${devices.productId}	产品ID	test/test_productId/topic	test/${devices.productId}/topic

表 5-21 定义策略变量示例

场景	策略Topic定义示例	描述
想通过topic区分不同设备的自定义上报时。	“test/${devices.deviceId}/topic”	允许Topic为“test/${本设备ID}/topic”的主题订阅或发布。有利于设备间数据隔离。
想通过topic区分不同设备的自定义上报、区分同一设备不同时间段上报的数据时。	“test/${devices.clientId}/topic”	允许Topic为“test/${本设备的clientId}/topic”的主题订阅或发布。与deviceId不同的是，clientId携带时间戳。可用主题来区分时间段。

策略优先级

当某个设备绑定的策略中有多个匹配且Effect不同时，具有优先级：拒绝 > 允许。

例如：某设备同时具有策略一、策略二两个策略，策略一拒绝订阅TopicA，策略二允许订阅TopicA，当该设备订阅TopicA时，平台会拒绝设备的订阅请求。

表 5-22 策略优先级

订阅/发布主题	策略一	策略二	策略匹配结果
test/topic	"effect": "ALLOW", "resources": ["topic:test/topic"]	"effect": "DENY", "resources": ["topic:test/topic"]	拒绝

策略主题基础检查

1. 长度不超过128字节。
2. 发布（publish）不允许使用topic通配符“#”、“+”。
3. 主题中的“/”分隔符不允许连续出现，如（////test/）。
4. 主题中“/”分隔符不超过7个。

注意

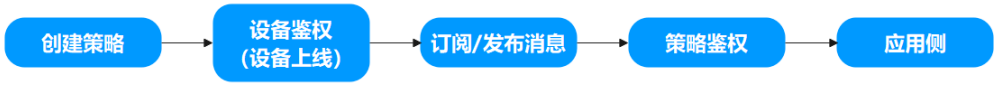
若发布、订阅的主题不符合以上限制，会直接拒绝本次订阅/发布请求。在设备“详情页面>消息跟踪”中可以看到报错。

图 5-57 订阅失败

5.5.3 设备策略使用说明

使用流程

图 5-58 设备策略流程



步骤1 创建策略：在控制台界面创建设备策略，创建设备策略可参考：**设备策略使用说明**。

步骤2 设备鉴权：设备发起连接鉴权（MQTT设备），鉴权参数填写请参考：**设备连接鉴权**。

步骤3 订阅/发布消息：设备对云服务端进行主题发布或订阅。

步骤4 策略鉴权：云服务端根据设备订阅/发布的Topic进行过滤，当该设备订阅的Topic被策略禁止，则响应失败ACK，订阅失败。若允许或未设定，则响应成功ACK，订阅成功。

步骤5 设备侧发布成功的消息数据可通过数据流转推送到应用侧。

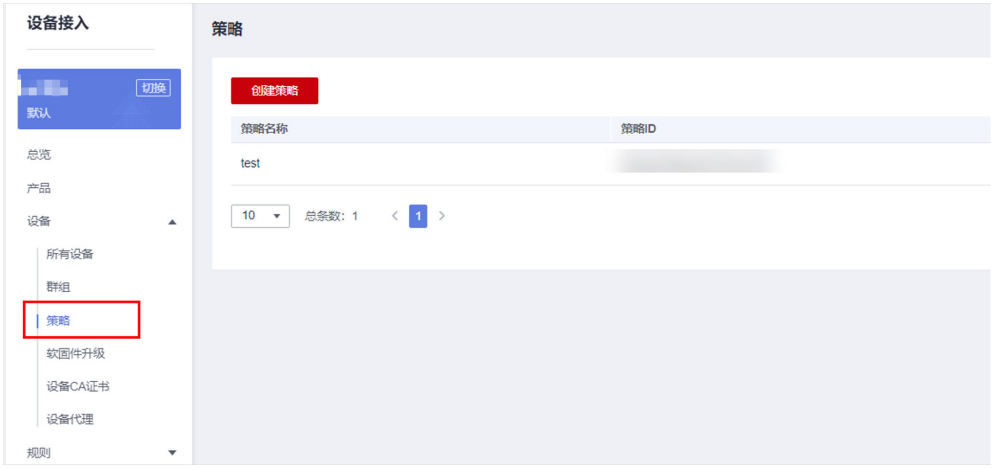
----结束

操作步骤

以下示例主要针对MQTT设备的订阅及发布。在设备接入控制台中可以进行策略的配置及策略目标的绑定。

1. 进入策略页面。访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。在左侧导航栏“设备”中单击“策略”，可到策略界面。

图 5-59 策略界面



2. 创建策略。在策略界面单击“创建策略”，按照业务具体填写策略参数，填写完成后单击生成策略。参数值可参考下图。

图 5-60 创建策略

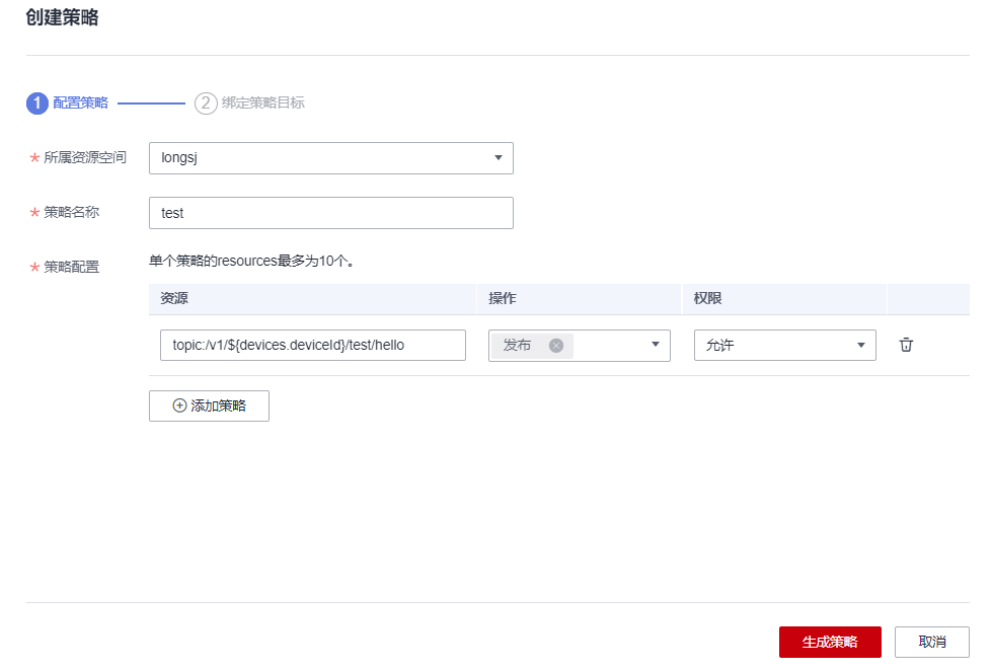


表 5-23 参数说明

参数说明	
所属资源空间	下拉选择所属的资源空间。如无对应的资源空间，请先创建资源空间。
策略名称	自定义，如PolicyTest。长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
资源	在MQTT主题发布与订阅中，需要以“topic:”作为参数前缀。比如说：禁止订阅/test/v1，则该参数填写 topic:/test/v1。

操作	值为发布或订阅。发布代表MQTT设备端Publish请求，订阅代表MQTT设备端Subscribe请求。
权限	值为允许或拒绝。用于允许或拒绝某topic的发布或订阅。

3. 绑定策略目标。策略可以从“资源空间”、“产品”、“设备”这3个范围进行绑定，被绑定的设备将遵循策略的要求允许或拒绝某Topic的发布或订阅。

图 5-61 策略绑定设备

表 5-24 参数说明

参数说明	
设备目标类型	下拉选择设备目标类型。类型有“资源空间”、“产品”、“设备”三种。这三种类型并不是互斥的，可以同时存在，比如说：绑定产品A与设备C（C是产品B下的设备）。 • 资源空间：实现多业务应用的分域管理，绑定后所选资源空间下的所有设备都将匹配该策略。可选择多个资源空间绑定。 • 产品：一个产品下一般有多个设备，绑定后所选产品下的所有设备都将匹配该策略，比起资源空间，绑定范围更小。可绑定一个或多个不同资源空间下的产品。 • 设备：绑定策略目标的最小单位，可绑定一个或多个不同资源空间、不同产品的设备。
策略目标	选择对应的“策略目标类型”后，在“策略目标”的参数中会显示可选的数据，勾选需要绑定的即可。

5.5.4 设备策略使用示例

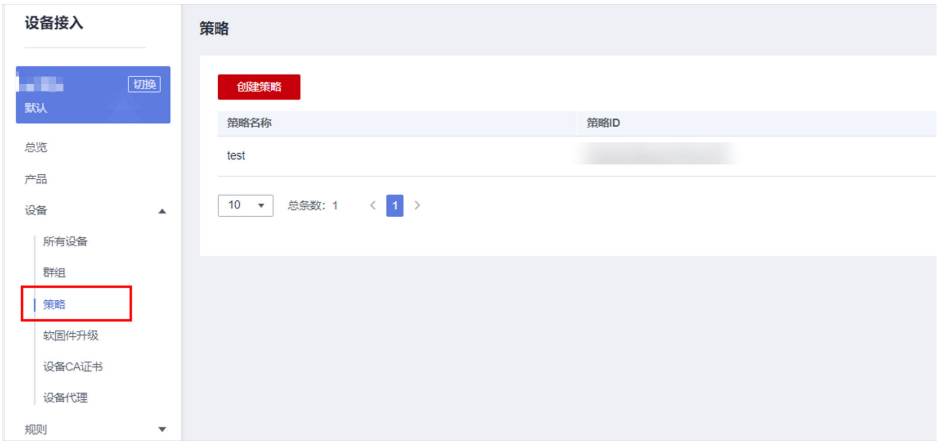
使用场景

- **场景一： 策略允许OR禁止发布某Topic**
- **场景二： 端到端（M2M）+策略**

场景一： 策略允许 OR 禁止发布某 Topic

1. 创建产品和设备
   - 创建产品。访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。单击左侧导航栏“产品”，在页面中单击红色按钮“创建产品”。根据页面提示填写参数，然后单击“确定”，完成产品的创建。详情可见：**创建产品**。
   - 创建设备。在设备接入控制台，单击左侧导航栏“设备 > 所有设备”，单击页面右上角的“注册设备”。根据页面提示填写参数，然后单击“确定”，完成设备的创建。详情可见：**注册单个设备**。
2. 生成策略
   - 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。
   - 左侧导航栏单击“设备 > 策略”，进入策略界面。

     图 5-62 策略界面

   - 在策略界面单击“创建策略”，填写策略参数，单击生成策略。策略的应用范围是资源空间（appId），其中资源“topic:”开头代表MQTT中的Topic，用于发布与订阅。允许发布和订阅主题：/v1/test/hello。

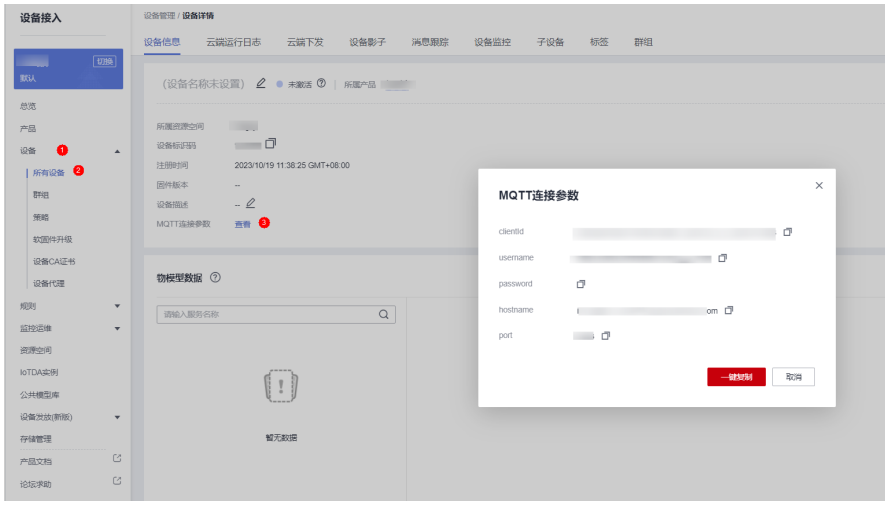
图 5-63 创建策略

- 绑定策略。下面以某个设备为例进行绑定。策略目标类型选择“设备”，单击需要绑定策略的设备。

图 5-64 策略绑定设备

- 验证策略。
  i. 获取连接参数。在“管理控制台”单击左侧导航栏“设备 > 所有设备”，找到上述策略绑定的设备，进入设备详情页面，查看连接参数。

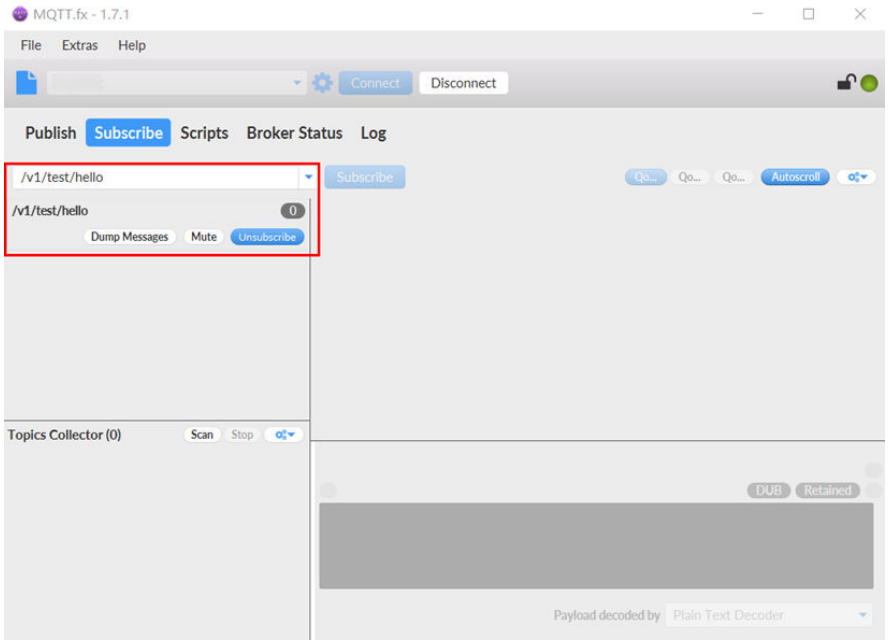
图 5-65 查看设备参数



ii. 使用MQTT.fx连接云平台。打开MQTT.fx软件，配置上述策略绑定的设备的鉴权参数，然后单击“Apply”保存，单击“Connect”进行连接鉴权。

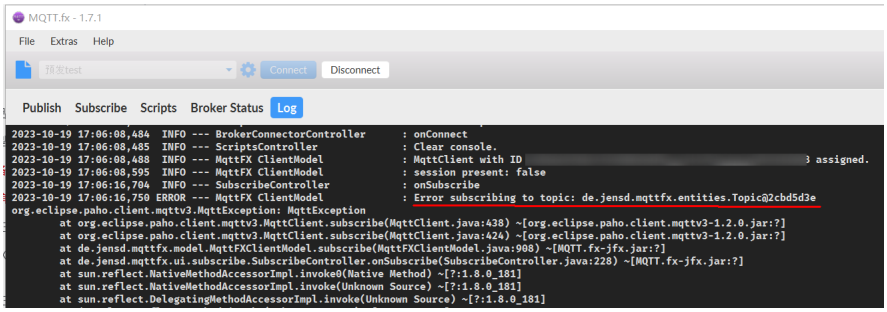
iii. 订阅允许的topic。订阅主题：/v1/test/hello，订阅成功。

图 5-66 订阅成功



iv. 订阅其他Topic。订阅主题：/v2/test/hello，订阅失败。

图 5-67 订阅失败

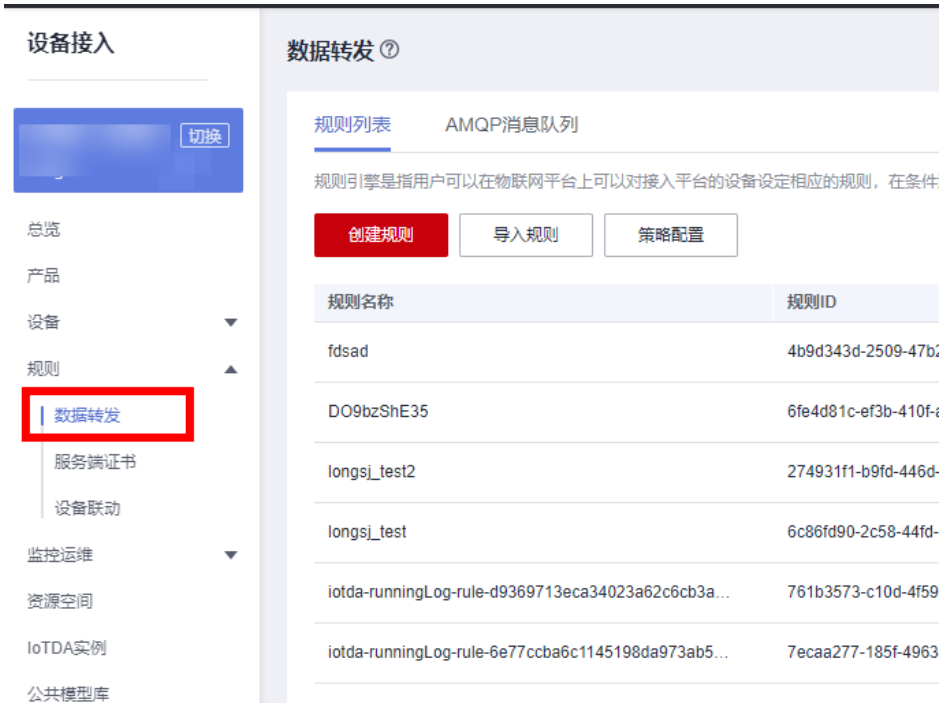


场景二： 端到端（M2M）+策略

接下来讲述产品A下的设备a，对产品B下的所有设备进行1对N通信（假设转发的Topic为以/test/M2M/开头的消息）。同时，只允许产品A下设备a、产品B下的所有设备进行发布、订阅Topic以/test/M2M/开头的消息。

1. 创建产品和设备
   - 创建产品。访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。单击左侧导航栏“产品”，在页面中单击红色按钮“创建产品”。根据页面提示填写参数，然后单击“确定”，完成产品的创建。详情可见：**创建产品**。
   - 创建设备。在设备接入控制台，单击左侧导航栏“设备 > 所有设备”，单击页面右上角的“注册设备”。根据页面提示填写参数，然后单击“确定”，完成设备的创建。详情可见：**注册单个设备**。
2. 设置流转规则
   - 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。在左侧导航栏“规则”中单击“数据转发”，可到数据转发界面。

图 5-68 数据转发页面



- 单击“创建规则”按照业务具体填写需要转发的数据参数，填写完成后单击“创建规则”。配置SQL过滤语句为：
STARTS_WITH(notify_data.body.topic,'/test/M2M/')。

图 5-69 创建数据转发规则

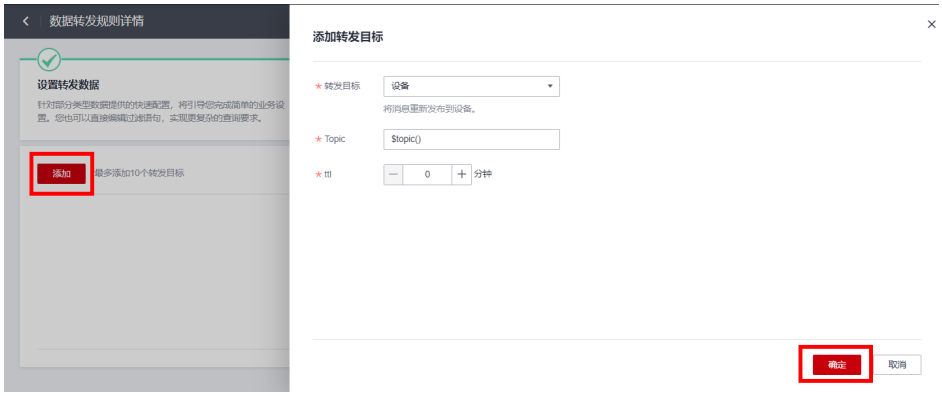
说明

SQL过滤语句的使用详情可见：**SQL语句**。

STARTS_WITH(notify_data.body.topic,'/test/M2M/')代表的是筛选主题（Topic）以“/test/M2M/”开头的数据。

- 设置转发到的目标。请将转发目标设置为“设备”，Topic填写“$topic()”(表示转发的Topic不变，按原本Topic下发)，单击“确定”。

图 5-70 设置转发目标



- 启动规则。单击启动规则，完成规则设置。

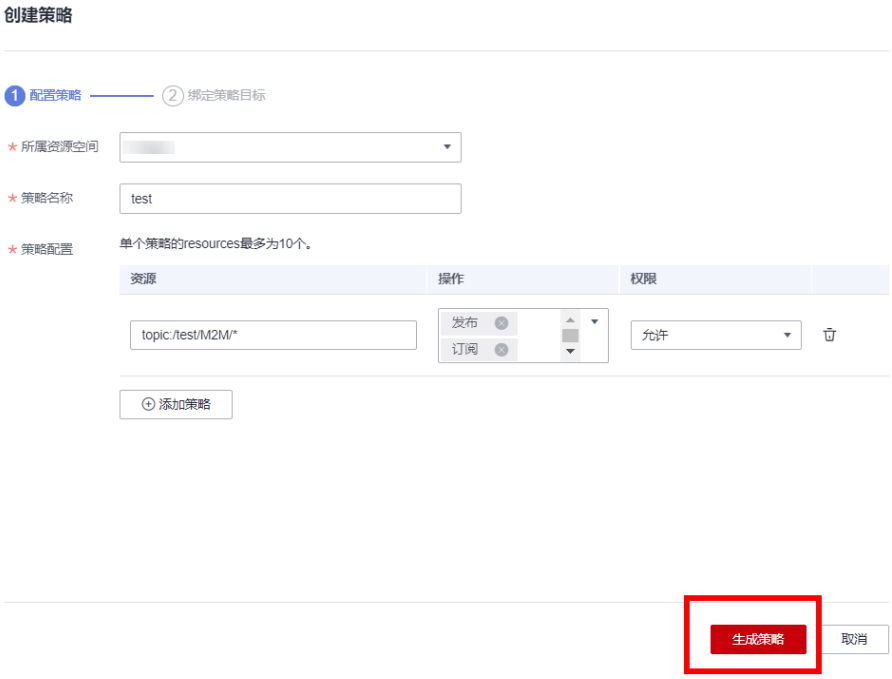
图 5-71 启动规则

3. 设置策略
   - 左侧导航栏单击“设备 > 策略”，进入策略界面。

图 5-72 策略界面

   - 在策略界面单击“创建策略”，填写策略参数，单击生成策略。添加策略如下图所示：

图 5-73 创建策略



- 绑定策略。把需要开放的产品及设备进行绑定。策略目标类型选择“产品”，单击需要绑定策略的产品。确定后可以在“策略详情”页面，进行“编辑”，添加要绑定的设备。

图 5-74 绑定产品

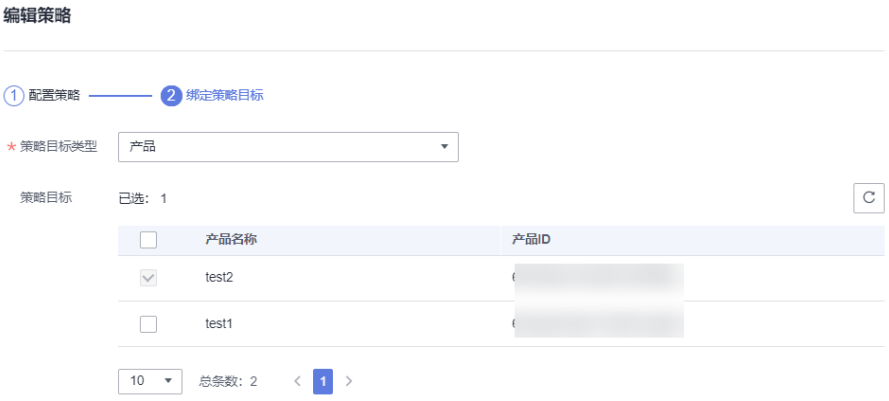
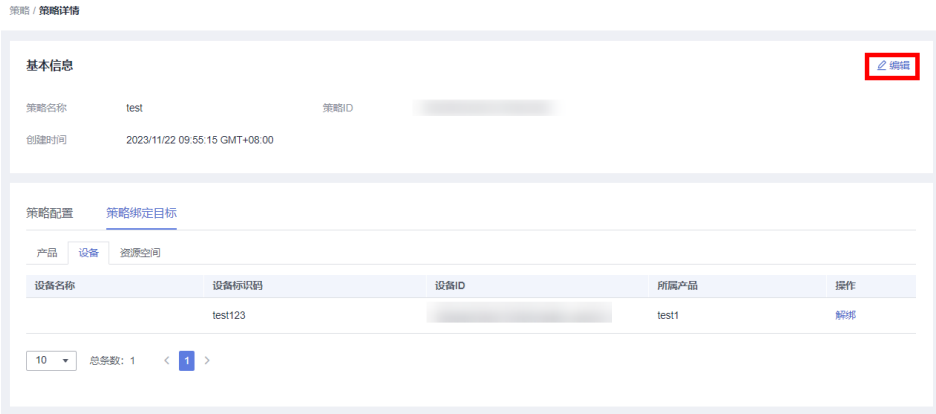


图 5-75 绑定设备

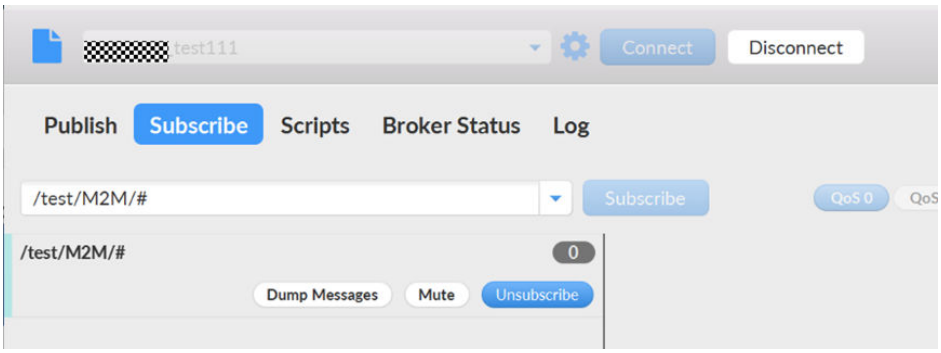


4. 验证策略。

   a. 打开两个MQTT.fx，分别模拟产品A下的设备a（test111）、产品B下的设备b（test222）。

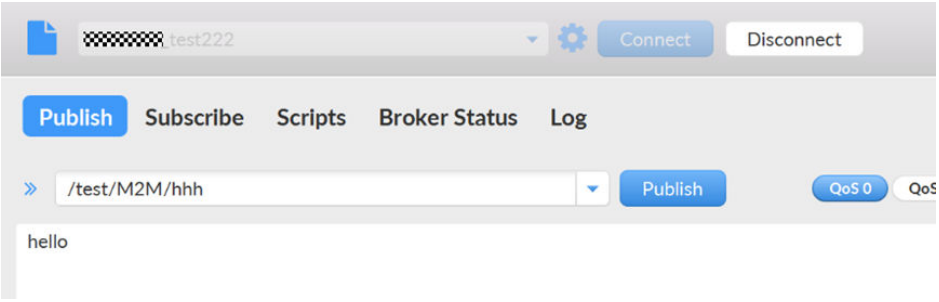
   b. 设备b在Subscribe页面中输入Topic“ /test/M2M/#”后，单击“Subscribe”订阅。

   图 5-76 设备 b（test222）在 Subscribe 页面中输入 topic



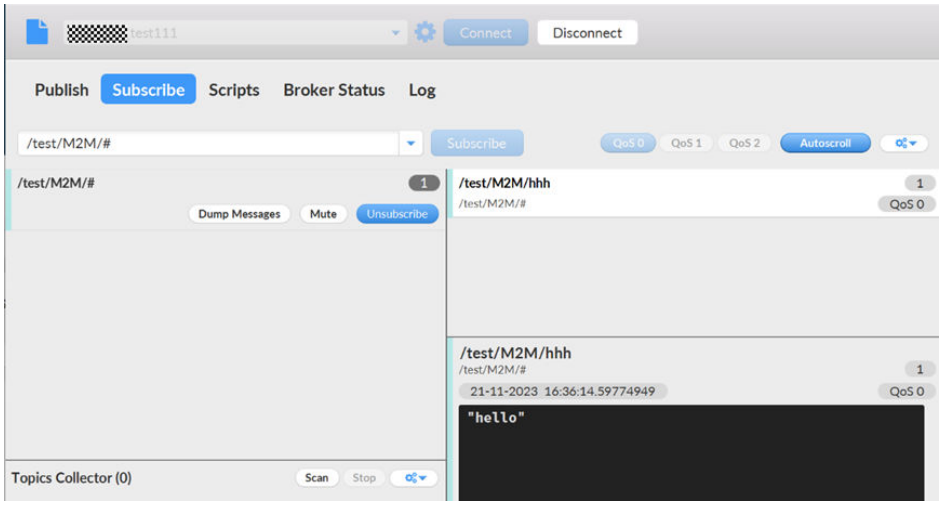
   c. 设备a（test111）向设备b（test222）发送消息，在设备a（test111）的MQTT.fx的“Publish”页面中，输入topic"/test/M2M/${任意单词}”（其中“${任意单词}”替换成任意单词），在内容输入框中输入要发送的消息（如：hello）单击“Publish”即可发送。

   图 5-77 设备 a Publish 页面输入消息



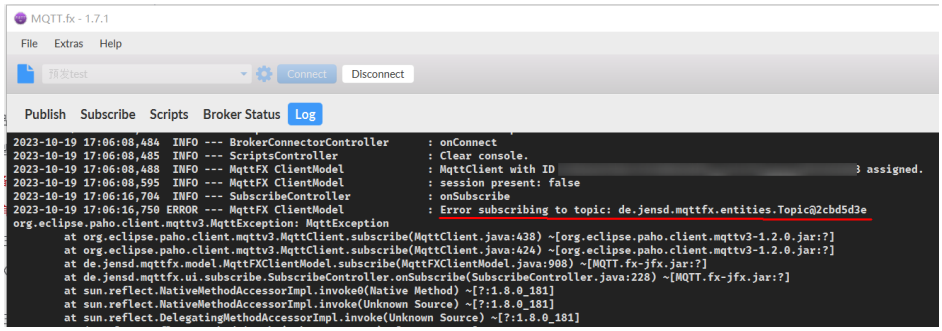
   在设备b的Subscribe页面可以看到接收的消息如下：

图 5-78 设备 b Subscribe 页面展示



d. 非产品B名下设备订阅或发布Topic：“ /test/M2M/#”，订阅、发布失败。

图 5-79 订阅失败



5.6 广播通信

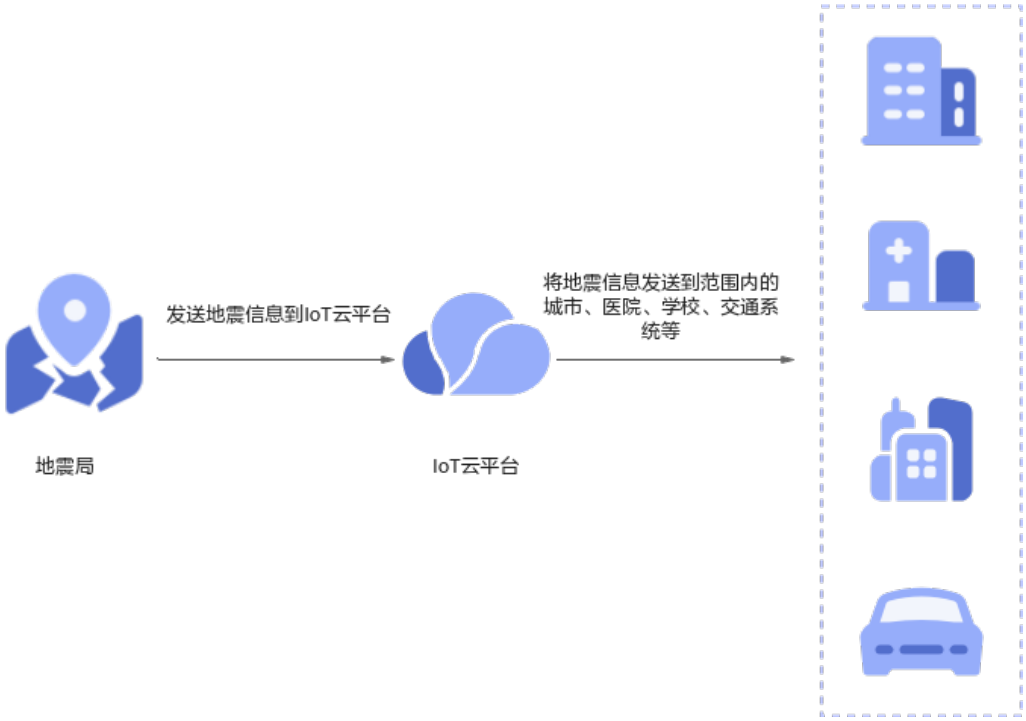
5.6.1 广播通信概述

概述

广播通信常用于一对多的消息通信。多个设备订阅相同的广播Topic，应用服务器调用广播消息下发的接口指定广播Topic的名称，就可以给已订阅该广播Topic的所有在线设备发布消息。广播模式的典型用途是根据设备的类别向设备发送通知。

例如，地震局给指定区域内的所有公民发送地震预警信息。

图 5-80 广播通信场景样例



使用场景

- 群组通信场景给指定群组内的设备下发广播消息。
- 地震预警场景给特定区域的所有在线设备发送广播消息。

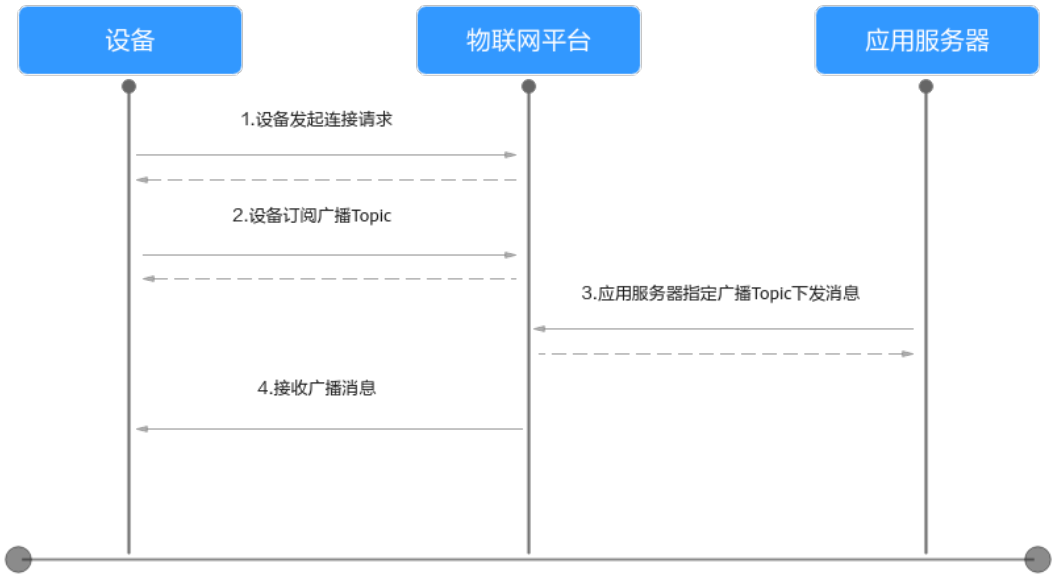
使用限制

- 设备订阅的广播Topic前缀必须为“$oc/broadcast/”。
- 设备订阅的广播Topic的字节长度不超过128字节。
- 设备订阅的广播Topic的“/”分隔符不超过7个。
- 单个设备订阅的广播Topic的数量不超过50。
- 同一个Topic最多允许被1,000个设备订阅。
- 应用侧广播通信接口调用频率限制为1次/每分钟。
- 当前只有北京四标准版支持该功能。

5.6.2 广播通信使用说明

广播通信使用说明

图 5-81 广播通信时序图



操作步骤

步骤1 设备发起连接鉴权，鉴权参数填写规则参考**设备连接鉴权**。

步骤2 设备鉴权成功后，发起广播Topic订阅，广播Topic必须为“$oc/broadcast/”前缀，样例如下：

```
$oc/broadcast/test
```

步骤3 应用服务器下发**广播消息**，指定Topic名称和消息内容。

```
POST https://{Endpoint}/v5/iot/{project_id}/broadcast-messages
Content-Type: application/json
X-Auth-Token: ********

{
  "topic_full_name" : "$oc/broadcast/test",
  "message" : "eyJhIjoxfQ=="
}
```

注意

这里Topic必须为“$oc/broadcast/”前缀，消息内容需要使用BASE64编码。

步骤4 设备接收广播消息，设备接收的广播消息样例如下。

```
Topic: $oc/broadcast/test
数据内容:
{"a":1}
```

----结束

5.6.3 广播通信使用示例

Java SDK 使用示例

本文介绍如何使用JAVA SDK进行广播通信的开发。

开发环境要求

本示例使用的开发环境为JDK 1.8及以上版本。

配置应用侧 SDK

1. 配置Maven依赖。

```
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-core</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
   <groupId>com.huaweicloud.sdk</groupId>
   <artifactId>huaweicloud-sdk-iotda</artifactId>
   <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

2. 下发广播消息完整样例如下，topic必须为“$oc/broadcast/”前缀，消息内容需要使用BASE64编码。

```
public class BroadcastMessageSolution {
   // REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
   private static final String REGION_ID = "<YOUR REGION ID>";
   // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
   private static final String ENDPOINT = "<YOUR ENDPOINT>";
   // 标准版/企业版：需自行创建Region对象
   public static final Region REGION_CN_NORTH_4 = new Region(REGION_ID, ENDPOINT);
   public static void main(String[] args) {
      String ak = "<YOUR AK>";
      String sk = "<YOUR SK>";
      String projectId = "<YOUR PROJECTID>";
      // 创建认证
      ICredential auth = new BasicCredentials().withDerivedPredicate(AbstractCredentials.DEFAULT_DERIVED_PREDICATE)
            .withAk(ak)
            .withSk(sk)
            .withProjectId(projectId);
      // 创建IoTDAClient实例并初始化
      IoTDAClient client = IoTDAClient.newBuilder().withCredential(auth)
            // 基础版：请选择IoTDARegion中的Region对象
            //.withRegion(IoTDARegion.CN_NORTH_4)
            // 标准版/企业版：需自行创建Region对象
            .withRegion(REGION_CN_NORTH_4).build();
      // 实例化请求对象
      BroadcastMessageRequest request = new BroadcastMessageRequest();
      DeviceBroadcastRequest body = new DeviceBroadcastRequest();
      body.withMessage(Base64.getEncoder().encodeToString("hello".getBytes()));
      body.withTopicFullName("$oc/broadcast/test");
      request.withBody(body);
      try {
         BroadcastMessageResponse response = client.broadcastMessage(request);
         System.out.println(response.toString());
      } catch (ConnectionException e) {
         e.printStackTrace();
      } catch (RequestTimeoutException e) {
         e.printStackTrace();
      } catch (ServiceResponseException e) {
         e.printStackTrace();
```

```
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getRequestId());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

表 5-25 参数说明

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
projectId	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.java**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。
REGION_ID	如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
ENDPOINT	请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。

配置设备侧 SDK

1. 配置设备侧SDK的Maven依赖。

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>1.1.4</version>
</dependency>
```

2. 配置设备侧SDK，设备连接参数。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://localhost:8883";
//在IoT平台创建的设备ID。
String deviceId = "deviceId";
//设备ID对应的密钥。
String deviceSecret = "secret";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

3. 订阅广播Topic，广播Topic需使用“$oc/broadcast/”前缀。

```
device.getClient().subscribeTopic("$oc/broadcast/test", null, rawMessage -> {
    log.info(" on receive message topic : {} , payload : {}", rawMessage.getTopic(),
        new String(rawMessage.getPayload()));
    rawMessage.getPayload();
}, 0);
```

测试验证

先运行设备侧 SDK代码，使设备上线，并订阅广播topic。然后运行应用侧SDK代码，调用broadcastMessage接口向设备发送广播消息，设备侧收到的广播消息样例如下：

图 5-82 广播消息样例

5.7 编解码插件

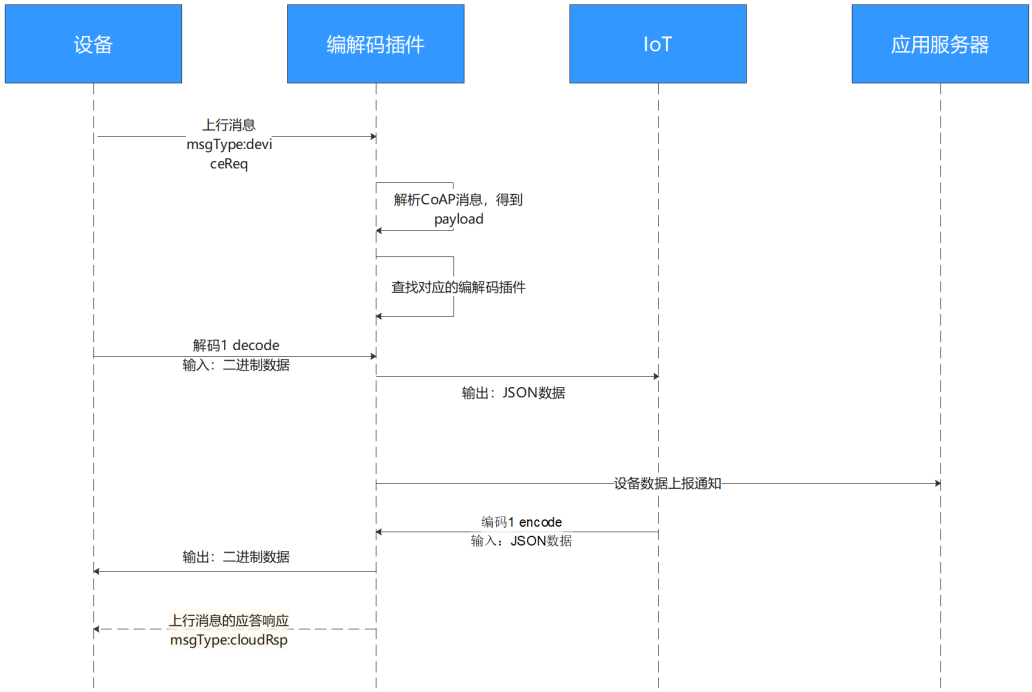
什么是编解码插件

编解码插件是供物联网平台调用，可以完成二进制格式与JSON格式相互转换、也可以完成JSON格式之间的转换。MQTT协议的设备建议使用JS插件、FunctionGraph、LwM2M协议的设备建议使用图形化插件、离线开发插件。

以NB-IoT场景为例，NB-IoT设备和物联网平台之间采用CoAP协议通讯，CoAP消息的payload为应用层数据，应用层数据的格式由设备自行定义。由于NB-IoT设备一般对省电要求较高，所以应用层数据一般不采用流行的JSON格式，而是采用二进制格式。但是，物联网平台与应用侧使用JSON格式进行通信。因此，您需要开发编码插件，供物联网平台调用，以完成二进制格式和JSON格式的转换。

数据上报流程

图 5-83 数据上报流程图

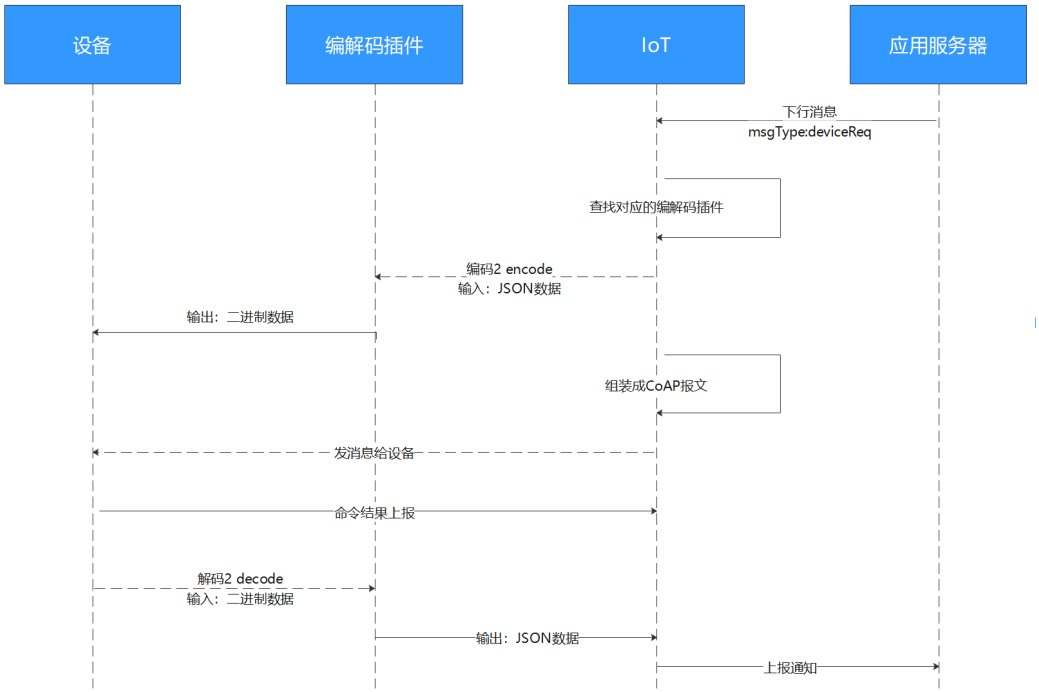


在数据上报流程中，有两处需要用到编解码插件：

- 将设备上报的二进制码流解码成JSON格式的数据，发送给应用服务器。
- 将应用服务器响应的JSON格式数据编码成二进制码流，下发给设备。

命令下发流程

图 5-84 命令下发流程图



在命令下发流程中，有两处需要用到编解码插件：

- 将应用服务器下发的JSON格式数据编码成二进制码流，下发给设备。
- 将设备响应的二进制码流解码成JSON格式的数据，上报给应用服务器。

编解码插件开发方法

物联网平台提供了多种开发编解码插件的方法，您可以根据自己需求，选择对应的方法开发编解码插件。由于离线开发编解码插件的方法较为复杂，且耗时比较长，我们**推荐**使用**图形化开发编解码插件**和**脚本化开发**。

- **图形化开发**：是指在设备接入控制台，通过可视化的方式快速开发一款产品的编解码插件。详细请参考**图形化开发**。
- **脚本化开发**：是指使用JavaScript脚本实现编解码的功能。详细请参考**脚本化开发**。
- **FunctionGraph开发**：是指通过FunctionGraph来实现编解码的功能。详细请参考**FunctionGraph开发**。

6 设备管理

6.1 创建产品

使用物联网平台的第一步就是在控制台创建产品。产品是设备的集合，是指某一类具有相同能力或特征的设备的合集。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。根据页面提示填写参数，然后单击“确定”，完成产品的创建。

基本信息	
所属资源空间	下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	为产品命名。产品名称在相同资源空间有唯一性。长度不超过64，只允许中文、字母、数字、以及_?'#().,&%@!-字符的组合。
协议类型	• MQTT：使用MQTT协议接入平台的设备，数据格式可以是二进制也可以是JSON格式，采用二进制时需要部署编解码插件。 • LwM2M/CoAP：使用在资源受限（包括存储、功耗等）的NB-IoT设备，数据格式是二进制，需要部署编解码插件才能与物联网平台交互。 • HTTPS：HTTPS是基于HTTP协议，通过SSL加密的一种安全通信协议。物联网平台支持HTTPS协议通信。 • Modbus：物联网平台支持使用Modbus协议接入，使用Modbus协议的设备接入IoT边缘节点的方式为非直连。直连设备和非直连设备差异说明，请参考**这里**。 • HTTP(TLS加密)、ONVIF、OPC-UA、OPC-DA、Other：通过边缘接入。

数据格式	• JSON：平台和设备之间的通信协议采用JSON格式。 • 二进制码流：您需在控制台开发**编解码插件**，将设备上报的二进制码流数据转换为JSON格式，将平台下发的JSON格式数据解析为二进制码流格式，设备才能与平台进行通信。
所属行业	请根据实际情况选择。
设备类型	请根据实际情况选择。
高级配置	
产品ID	定制ProductID，用于唯一标识一个产品。如果携带此参数，平台将产品ID设置为该参数值；如果不携带此参数，产品ID在物联网平台创建产品后由平台分配获得。
产品描述	产品描述。请根据实际情况填写。

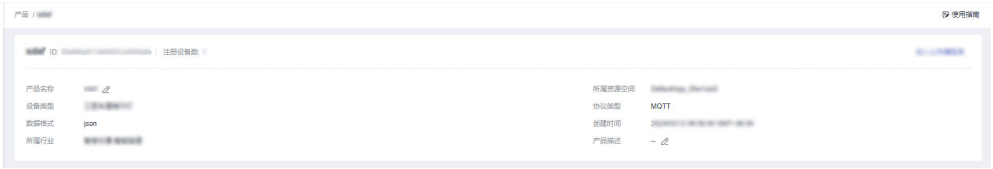
产品创建成功后，您可以单击“删除”删除不再使用的产品。删除产品后，该产品下的产品模型、编解码插件等资源将被清空，请谨慎操作。

----结束

后续步骤

1. 在产品列表中，单击对应的产品，进入产品详情页。您可以查看产品ID、产品名称、设备类型、数据格式、所属资源空间、协议类型等产品基本信息。

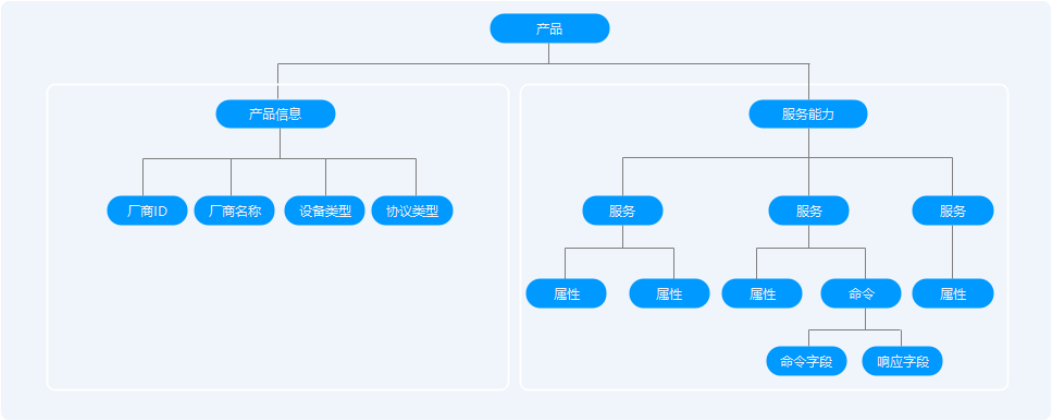
   图 6-1 产品-产品详情

2. 您可以在产品详情页，**开发产品模型**、**开发编解码插件**、**在线调试**、**自定义Topic**。

什么是产品模型

产品模型用于描述设备具备的能力和特性。开发者通过定义产品模型，在物联网平台构建一款设备的抽象模型，使平台理解该款设备支持的服务、属性、命令等信息，如颜色、开关等。当定义完一款产品模型后，再进行**创建设备**时，就可以使用在控制台上定义的产品模型。



产品模型包括产品信息和服务能力：

- **产品信息**

  描述一款设备的基本信息，包括设备类型、协议类型。

  例如：设备类型为“WaterMeter”，协议类型为“CoAP”。

- **服务能力**

  描述设备具备的业务能力。将设备业务能力拆分成若干个服务后，再定义每个服务具备的属性、命令以及命令的参数。

  以水表为例，水表具有多种能力，如上报水流、告警、电量、连接等各种数据，并且能够接受服务器下发的各种命令。产品模型文件在描述水表的能力时，可以将水表的能力划分五个服务，每个服务都需要定义各自的上报属性或命令。说明如下：

服务类型	描述
基础（WaterMeterBasic）	用于定义水表上报的水流量、水温、水压等参数，如果需要命令控制或修改这些参数，还需要定义命令的参数。
告警（WaterMeterAlarm）	用于定义水表需要上报的各种告警场景的数据，必要的话需要定义命令。
电池（Battery）	定义水表的电压、电流强度等数据。
传输规则（DeliverySchedule）	定义水表的一些传输规则，必要的话需要定义命令。
连接（Connectivity）	定义水表连接参数。

说明

具体定义几个服务是非常灵活的，如上面的例子可以将告警服务拆分成水压告警服务和流量告警服务，也可以将告警服务合入到水表基础服务中。

产品模型开发方法

物联网平台提供了多种开发产品模型的方法，您可以根据自己需求，选择对应的方法开发产品模型。

- **自定义模型（在线开发）**：从零自定义构建产品模型。详细参考**在线开发产品模型**。
- **上传模型文件（离线开发）**：将本地写好的产品模型上传到平台。详细请参考**离线开发产品模型**。
- **Excel导入**：通过导入文件的方式快速开发产品模型。详细请参考**Excel导入**。
- **导入库模型（平台预置产品模型）**：您可以使用平台预置的产品模型，快速完成产品开发。当前平台提供了标准模型和厂商模型。标准模型遵循行业标准的产品模型，适用行业内绝大部分厂商设备，而厂商模型针对设备类型发布的产品模型，适用于用行业内少量厂家设备。您可以根据实际需求选择相应的产品模型。

6.2 注册设备

6.2.1 注册单个设备

归属于某个产品下的设备实体，每个设备具有一个唯一的标识码。设备可以是直连物联网平台的设备，也可以是代理子设备连接物联网平台的网关。您可以在物联网平台注册您的实体设备，通过平台分配的设备ID和密钥，在集成了SDK后，您的设备可以接入到物联网平台，实现与平台的通信及交互。

物联网平台支持通过应用服务器调用**创建设备**接口，或者在控制台上注册单个设备。本文介绍如何在控制台上注册单个设备。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“ 设备 > 所有设备”，单击“注册设备”，按照如下表格填写参数后，单击“确定”。

图 6-2 设备-注册密钥设备

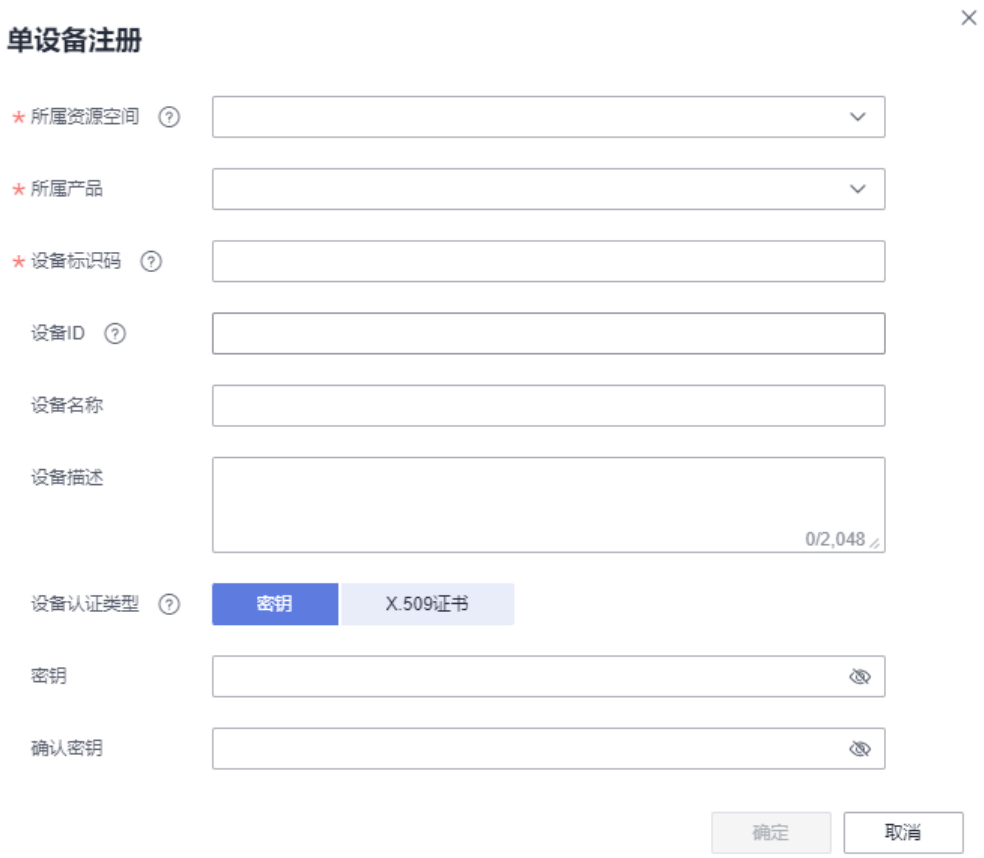


表 6-1 注册密钥设备

参数名称	说明
所属资源空间	选择设备所属的资源空间。
所属产品	选择设备所属的产品。 只有在这里创建了产品，此处才可以选择具体的产品。如没有，请先创建产品。
设备标识码	即node_id，填写为设备的IMEI、MAC地址或Serial No；若没有真实设备，填写自定义字符串，由英文字母、数字、连接号-和下划线_组成。
设备ID	设备ID，用于唯一标识一个设备。如果携带该参数，平台将设备ID设置为该参数值；如果不携带该参数，设备ID由物联网平台分配获得，生成规则为product_id + _ + node_id拼接而成。
设备名称	即device_name，可自定义。
设备描述	设备的描述信息，可自定义。
设备认证类型	• 密钥：设备通过密钥验证身份。 • X.509证书：设备使用X.509证书验证身份。

参数名称	说明
密钥	设备密钥，可自定义，不填写物联网平台会自动生成。
指纹	当“设备认证类型”选择“X.509证书”时填写，导入**设备侧预置的设备证书**对应的指纹，在OpenSSL执行**openssl x509 -fingerprint -sha256 -in deviceCert.pem**命令可查询。 [root@k8s-iot-wl2-2-iump cert1223]# openssl x509 -fingerprint -sha256 -in deviceCert.pem SHA256 Fingerprint=F7:91:90:45:BB:88:37:E6:A7:E7:70:4A:90:75:F3:87:DA:27:5B:7C:49:3E:FF:59:7A:6F:4F:08:4D:F8:54:E8 填写时需要删除冒号。

设备注册成功后，请妥善保管好设备ID和密钥，用于设备接入平台认证。

图 6-3 设备-注册设备成功

说明

若密钥丢失，可以按照**设备认证凭证管理**中的步骤更新设备密钥，无法找回注册设备时生成的密钥。

用户可在**设备列表**删除不再使用的设备。删除设备不支持撤回，请谨慎操作。

----结束

相关 API 参考

- 查询设备列表

- 创建设备
- 查询设备
- 修改设备
- 删除设备
- 重置设备密钥

6.2.2 批量注册设备

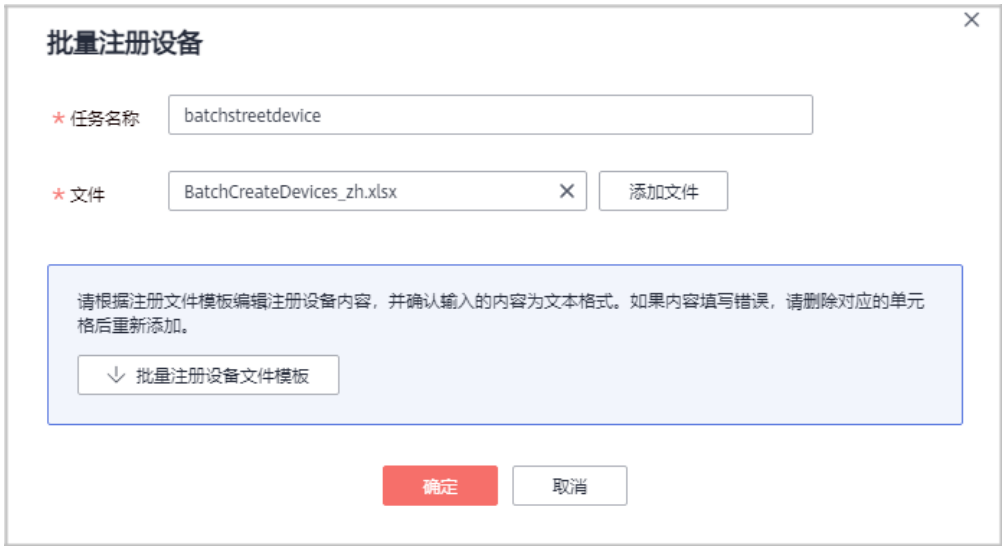
物联网平台支持通过应用服务器调用创建批量任务接口，或者在控制台上批量注册设备。本文介绍如何在控制台上批量注册设备。

操作步骤

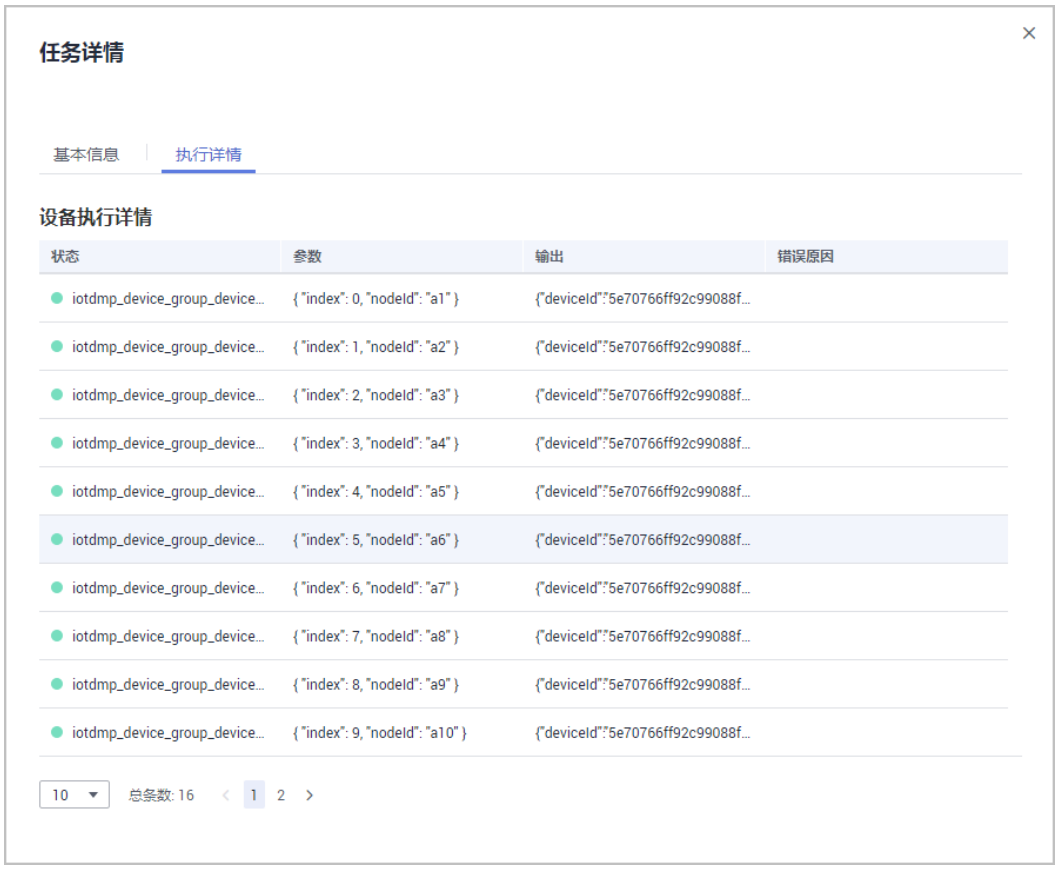
步骤1 访问设备接入服务，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备 > 所有设备”，进入“批量注册”页签，单击“批量注册”。

步骤3 弹出批量注册设备窗口，填写“任务名称”，下载并填写“批量注册设备文件模板”内容并上传文件，单击“确定”创建任务。



步骤4 批量注册执行成功，如果是原生MQTT设备注册，请单击批量任务一行，进入任务的“执行详情”，保存好设备ID和密钥，用于原生MQTT设备接入平台。



----结束

相关 API 参考

- 创建设备
- 查询批量任务列表
- 创建批量任务
- 查询批量任务

6.2.3 注册 X.509 证书认证的设备

X.509是一种用于通信实体鉴别的数字证书，物联网平台支持设备使用自己的X.509证书进行认证鉴权。使用X.509认证技术时，设备无法被仿冒，避免了密钥被泄露的风险。

注册X.509证书认证的设备前，您需要先在物联网平台上传设备的CA证书，然后在注册设备时将设备证书同设备进行绑定。本文介绍如何在物联网平台上传设备CA证书，以及注册X.509证书认证的设备。

限制说明

- 当前只有通过MQTT接入的设备支持使用X.509证书进行设备身份认证。
- 每个用户最多上传100个设备CA证书。

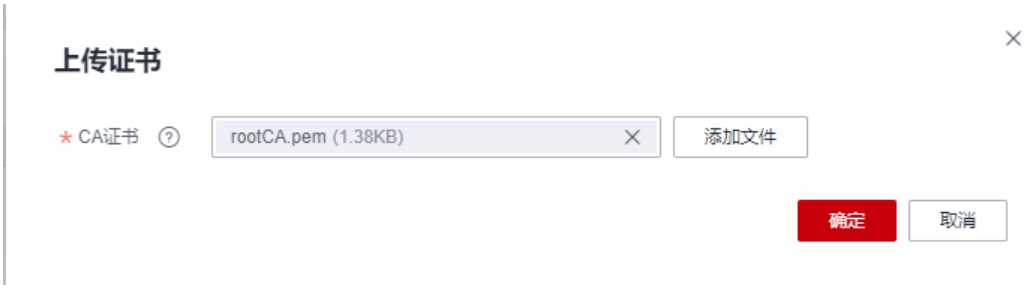
上传设备 CA 证书

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备 > 设备CA证书”，单击右上角的“上传证书”。

步骤3 在弹出的对话框中，单击“添加文件”，然后单击“确定”。

图 6-4 设备 CA 证书-上传证书



说明

设备CA证书由设备厂商提供，调测时可**自行制作调测证书**，商用时建议更换为商用证书，否则会带来安全风险。如果已购买CA证书（ pem、jks等 ），可直接上传到平台。

----结束

制作设备 CA 调测证书

本文以Windows环境为例，介绍通过Openssl工具制作调测证书的方法，生成的证书为PEM编码格式的证书。

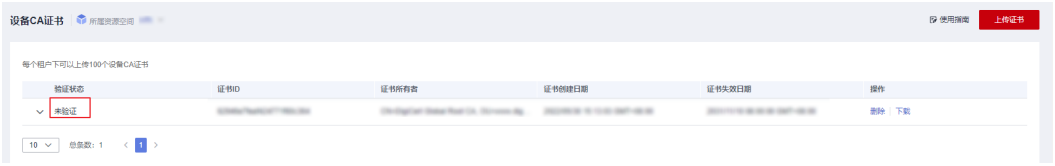
1. 在浏览器中访问**这里**，下载并进行安装OpenSSL工具。
2. 以管理员身份运行cmd命令行窗口。
3. 执行cd c:\openssl\bin（ 请替换为openssl实际安装路径 ），进入openssl命令视图。
4. 执行以下命令生成生成密钥对。
   ```
   openssl genrsa -out rootCA.key 2048
   ```
5. 执行以下命令，使用密钥对中的私有密钥生成 CA 证书。
   ```
   openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
   ```
   系统提示您输入如下信息，所有参数可以自定义。
   - Country Name (2 letter code) [AU]：国家，如CN。
   - State or Province Name (full name) []: 省份，如GD。
   - Locality Name (for example, city) []:城市，如SZ。
   - Organization Name (for example, company) []：组织，如Huawei。
   - Organizational Unit Name (for example, section) []：组织单位，如IoT。
   - Common Name (e.g. server FQDN or YOUR name) []: 名称，如zhangsan。
   - Email Address []:邮箱地址，如1234567@163.com。

   在openssl安装目录的bin文件夹下，获取生成的CA证书（ rootCA.pem ）。

上传验证证书

如果上传的是调测证书，上传后证书状态显示为“未验证”，您需要上传验证证书，来证明您拥有该CA证书。

图 6-5 设备 CA 证书-未验证证书



验证证书是由设备CA证书对应的私钥创建的，请参考如下操作制作验证证书。

步骤1 执行如下命令为私有密钥验证证书生成密钥对。

```
openssl genrsa -out verificationCert.key 2048
```

步骤2 执行如下命令为私有密钥验证证书创建CSR（Certificate Signing Request）。

```
openssl req -new -key verificationCert.key -out verificationCert.csr
```

系统提示您输入如下信息，**Common Name填写为验证证书的验证码**，其他参数自定义。

- Country Name (2 letter code) [AU]：国家，如CN。
- State or Province Name (full name) []: 省份，如GD。
- Locality Name (for example, city) []:城市，如SZ。
- Organization Name (for example, company) []：组织，如Huawei。
- Organizational Unit Name (for example, section) []：组织单位，如IoT。
- Common Name (e.g. server FQDN or YOUR name) []：验证证书的验证码，请参考**步骤5**获取 。
- Email Address []:邮箱地址，如1234567@163.com。
- Password[]：密码，如1234321。
- Optional Company Name[]：公司名称，如Huawei。

步骤3 执行以下命令使用CSR创建私有密钥验证证书。

```
openssl x509 -req -in verificationCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out verificationCert.pem -days 500 -sha256
```

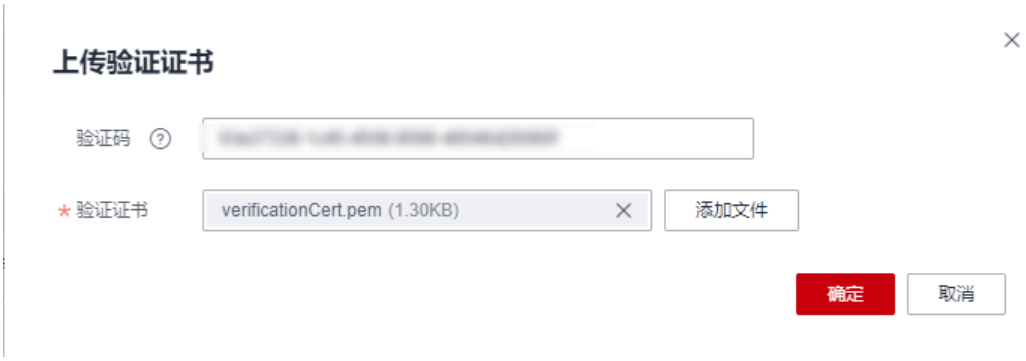
在openssl安装目录的bin文件夹下，获取生成的验证证书（verificationCert.pem）。

步骤4 选择对应证书，单击 ˅ 然后单击“上传验证证书”。

图 6-6 设备 CA 证书-验证证书

步骤5 在弹出的对话框中，单击“添加文件”，然后单击“确定”。

图 6-7 设备 CA 证书-上传验证证书



上传验证证书后，证书状态变为“已验证”，表明您拥有该CA证书。

----结束

预置 X.509 证书

在注册X.509设备之前，您需要在设备侧预置CA机构签发的X.509证书。

说明

X.509证书由CA机构签发，若没有CA机构签发的商用证书，您可以自己**制作X.509调测证书**。如果已购买相关证书，或者权威机构签发的证书（pem、jks等），可直接上传到平台。

制作X.509调测证书

1. 以管理员身份运行cmd命令行窗口，执行cd c:\openssl\bin（请替换为openssl实际安装路径），进入openssl命令视图。
2. 执行如下命令生成密钥对。
   ```
   openssl genrsa -out deviceCert.key 2048
   ```
3. 执行如下命令为设备证书创建CSR（Certificate Signing Request）。
   ```
   openssl req -new -key deviceCert.key -out deviceCert.csr
   ```
   系统提示您输入如下信息，所有参数可以自定义。
   - Country Name (2 letter code) [AU]：国家，如CN。
   - State or Province Name (full name) []: 省份，如GD。
   - Locality Name (for example, city) []:城市，如SZ。
   - Organization Name (for example, company) []：组织，如Huawei。
   - Organizational Unit Name (for example, section) []：组织单位，如IoT。
   - Common Name (e.g. server FQDN or YOUR name) []: 名称，如zhangsan。
   - Email Address []:邮箱地址，如1234567@163.com。
   - Password[]：密码，如1234321。
   - Optional Company Name[]：公司名称，如Huawei。
4. 执行以下命令使用CSR创建设备证书。
   ```
   openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out deviceCert.pem -days 500 -sha256
   ```
   在openssl安装目录的bin文件夹下，获取生成的设备证书（deviceCert.pem）。

注册 X.509 证书认证的设备

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“ 设备 > 所有设备”，单击“注册设备”，按照如下表格填写参数后，单击“确定”。

图 6-8 设备-注册 X.509 设备

表 6-2 注册 X.509 设备

参数名称	说明
所属资源空间	选择设备所属的资源空间。
所属产品	选择设备所属的产品。 只有在**这里**创建了产品，此处才可以选择具体的产品。如没有，请先创建产品。
设备标识码	即node_id，填写为设备的IMEI、MAC地址或Serial No；若没有真实设备，填写自定义字符串，由英文字母、数字、连接号-和下划线_组成。
设备ID	设备ID，用于唯一标识一个设备。如果携带该参数，平台将设备ID设置为该参数值；如果不携带该参数，设备ID由物联网平台分配获得，生成规则为product_id + _ + node_id拼接而成。
设备名称	即device_name，可自定义。

参数名称	说明
设备描述	设备的描述信息，可自定义。
设备认证类型	X.509证书：设备使用X.509证书验证身份。
指纹	当“设备认证类型”选择“X.509证书”时填写，导入**设备侧预置的设备证书**对应的指纹，在OpenSSL执行**openssl x509 -fingerprint -sha256 -in deviceCert.pem**命令可查询。**注：填写时需要删除冒号**。 [root@k8s-iot-wl2-2-jump cert1223]# openssl x509 -fingerprint -sha256 -in deviceCert.pem SHA256 Fingerprint=F7:91:90:45:BB:88:37:E6:A7:E7:70:4A:90:75:F3:87:DA:27:5B:7C:49:3E:FF:59:7A:6F:4F:08:4D:F8:54:E8

----结束

相关 API 参考

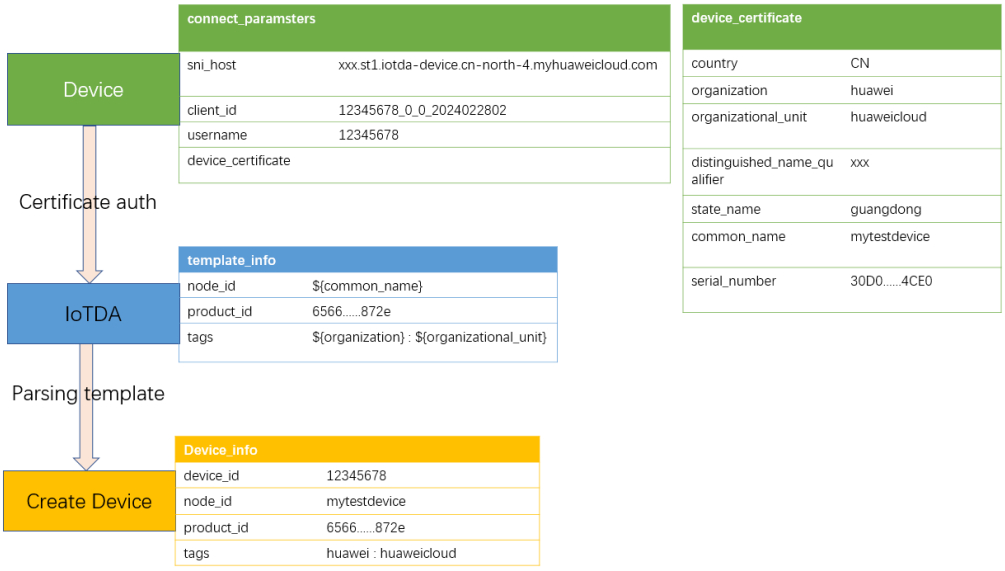
- **获取设备CA证书列表**
- **上传设备CA证书**
- **删除设备CA证书**
- **验证设备CA证书**

6.2.4 设备自注册

概述

自注册模板主要用于设备自动注册的场景，不需要提前在物联网平台进行注册设备。当前物联网平台基于安全考虑，需要将设备的基本信息（例如设备ID、鉴权信息）注册到平台后，设备才能连接上平台。当用户的设备未在平台注册时，可以通过自注册模板在设备首次接入物联网平台时将设备信息自动注册到物联网平台。本文基于设备自注册功能，使用证书+SNI扩展，实现设备自注册。

图 6-9 业务流程图



使用场景

- 通过设备证书触发自动注册，免去走设备发放流程。
- 对于车联网场景，车机启动设备即上线，使用自注册可以减少车端应用侧开发。
- 对于大企业客户购买多个IoTDA实例，使用自注册功能可以免去设备提前在不同实例上发放注册。

使用限制

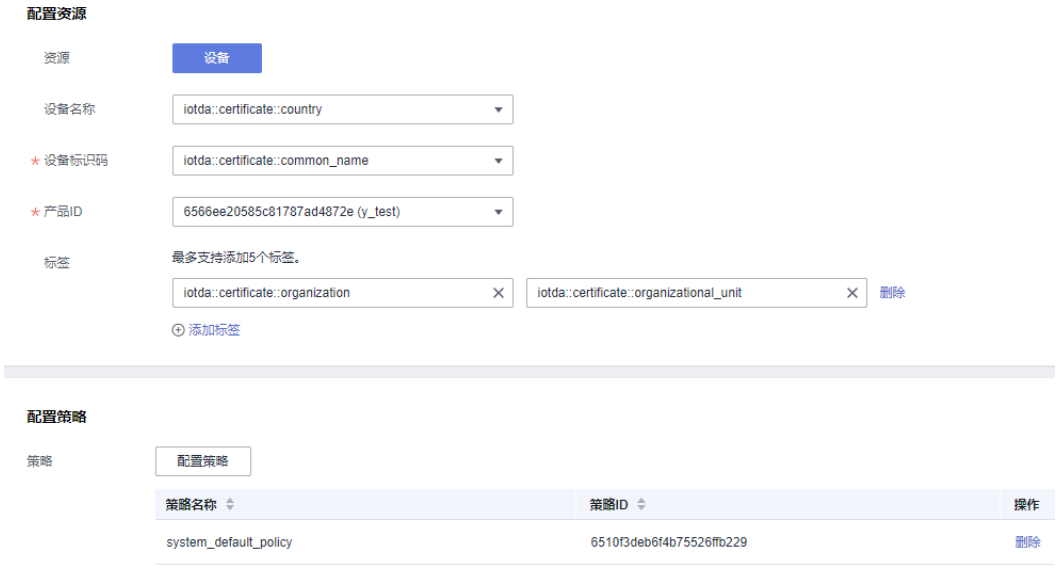
- 单账号下自注册模板最多可以创建10个。
- 使用设备自注册功能，要求设备必须使用TLS同时开启**服务器名称指示（SNI）**扩展，SNI中需要携带平台分配的域名，在“总览 > 接入信息”中查看域名信息。
- 目前该功能仅支持MQTTS证书双向认证的场景。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 创建自注册模板：在设备接入控制台左侧导航栏，选择“设备 > 自注册模板”，单击“创建模板”。模板中可以预先为设备绑定策略，有关设备策略详细使用参考：**设备Topic策略**。其中设备标识码与产品ID为必填配置项，设备ID自定义取值为mqtt连接参数中的Username，产品需提前在平台创建好。

图 6-10 创建模板



说明

平台预定义了模板中可以声明和引用的以下参数，证书必须包含模板中所引用的参数信息：

- iotda::certificate::country：国家
- iotda::certificate::organization：组织
- iotda::certificate::organizational_unit：部门
- iotda::certificate::distinguished_name_qualifier：可分辨名称
- iotda::certificate::state_name：省份
- iotda::certificate::common_name：通用名
- iotda::certificate::serial_number：序列号

步骤3 制作设备证书，参考 **X.509证书认证的设备**，上传CA证书到平台并通过验证，绑定**步骤2**所创建的“自注册模板”并且“开启自注册能力”。

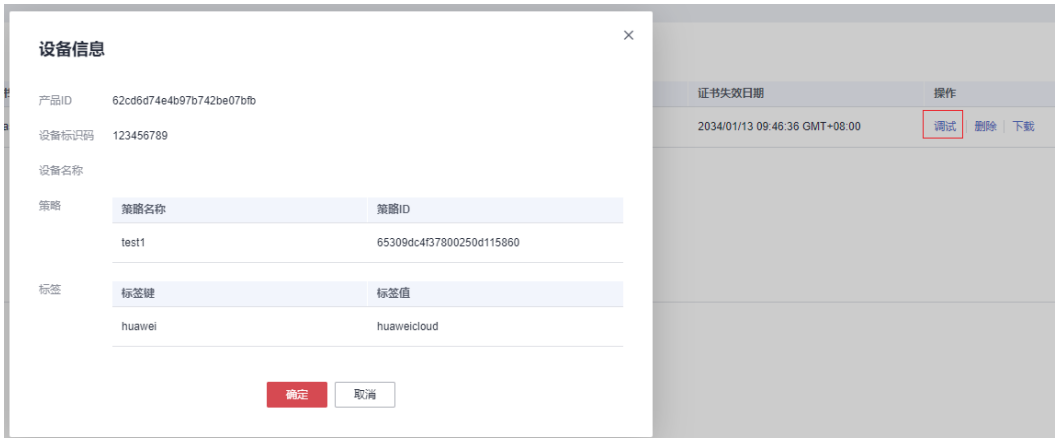
图 6-11 绑定模板

须知

注册设备所在的资源空间即为设备CA证书所在资源空间，请确保CA证书与模板中的产品ID在同一个资源空间下。

步骤4 在设备接入控制台左侧导航栏，选择“设备 > 设备CA证书”，单击“调试”上传**步骤3**制作的设备证书，查看预解析的设备信息是否符合预期。

图 6-12 调试证书



----结束

验证

1. 使用MQTT.fx工具模拟设备首次接入平台并自动注册设备，clientId格式要求参考**连接参数说明**，其中Username即为设备注册在平台上的设备ID的值，Password不用填写，平台CA证书获取地址**证书资源**，连接成功后在平台查看注册上的设备信息。

   图 6-13 连接参数图

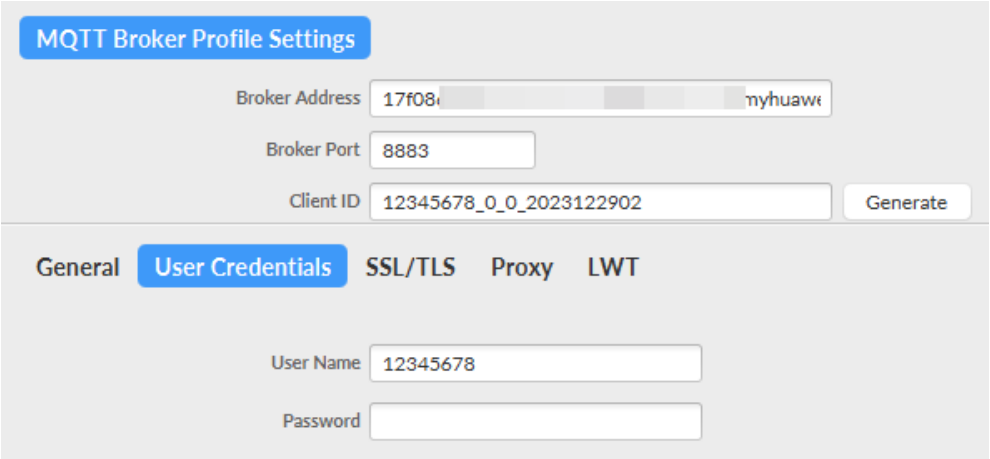
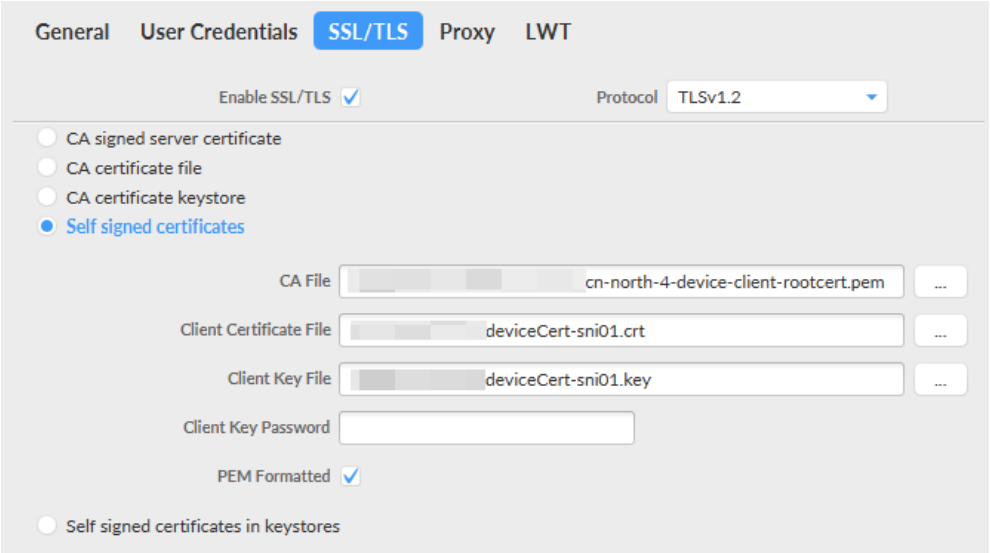
   

   图 6-14 证书信息图

2. 连接成功后，可以在控制台所有设备中找到自注册的设备。

图 6-15 设备自注册信息

6.3 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，也可以冻结设备。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在设备接入控制台左侧导航栏，选择“设备 > 所有设备”，进入设备列表页，设备列表默认显示当前实例下的所有设备。

图 6-16 设备-设备列表

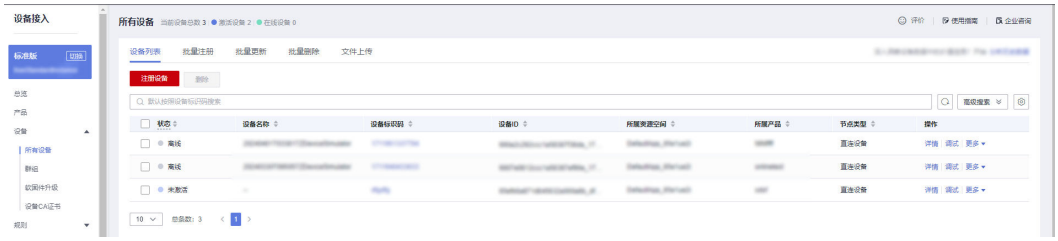


表 6-3 设备列表功能

功能	描述
搜索设备	根据状态、设备名称、设备标识码、设备ID、所属资源空间、所属产品和节点类型搜索具体设备。
查看设备信息	可查看**设备状态**、设备名称、设备标识码等，也可以单击设备对应的“详情”，查看**设备详情**。
删除设备	单击设备对应的“删除”。 **说明** 删除设备后，设备的相关数据也会被删除，请谨慎操作。 如果需要删除的设备数量较多，为方便节省时间，您可以调用**创建批量任务**接口，或者在控制台批量删除设备。详细请参考**批量删除设备**。

功能	描述
冻结设备	单击设备对应的“冻结”。 **说明** 设备冻结后不能再连接上线，当前仅支持冻结与平台直连的设备。 如果需要冻结的设备数量较多，为方便节省时间，您可以调用**创建批量任务**接口实现。
解冻设备	单击设备对应的“解冻”。 如果需要解冻的设备数量较多，为方便节省时间，您可以调用**创建批量任务**接口实现。
调试设备	单击设备对应的“调试”。

----结束

设备状态含义

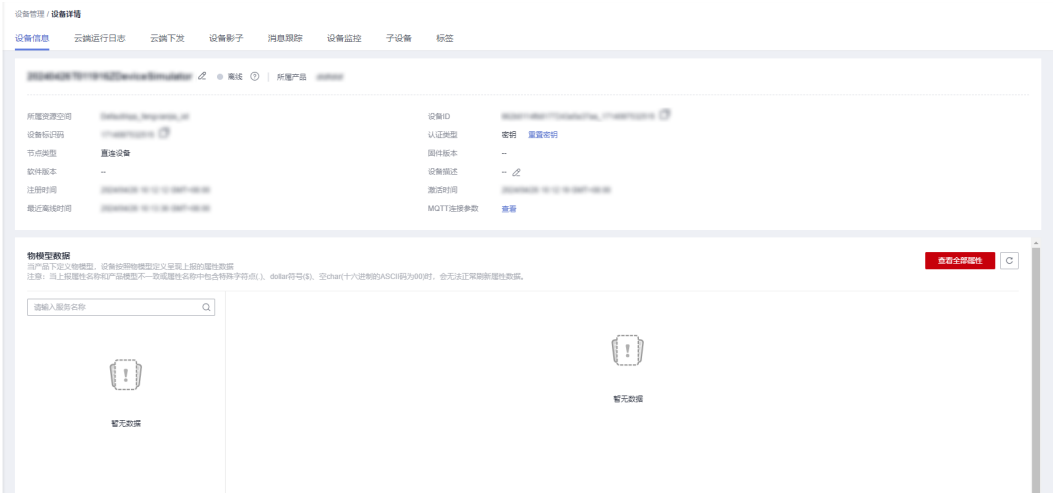
在控制台上可以查看设备当前状态，如在线、离线、未激活、异常、冻结。用户也可以通过**订阅方式**获取设备的状态信息。设备状态定义如下：

状态类型	状态	短连接设备（如NB-IoT设备）	长连接设备（MQTT）
连接状态	在线	如果在25小时内设备有上报过数据，设备的状态为“在线”；如果在25小时内未上报过数据，设备的状态会变为“异常”。	设备与平台之间一直连接，无断开。
	离线	设备接入平台后，设备在超过49小时未上报数据，平台会将设备置为“离线”状态。	设备与平台之间的连接断开1分钟后（数据自动刷新周期为1分钟），置为“离线”状态。 如果在界面上手动刷新状态，则直接显示“离线”。
	异常	设备接入平台后，设备在超过25小时未上报数据，平台会将设备置为“异常”状态。	无此状态。
	未激活	已在平台上完成设备注册但真实设备还未接入平台。请根据**设备初始化**操作完成设备的接入。	已在平台上完成设备注册但真实设备还未接入平台。请根据**设备初始化**操作完成设备的接入。
管理状态	冻结	用户主动将设备状态置于冻结状态，设备冻结后不能再连接上线，当前仅支持冻结与平台直连的设备。	

查看设备详情

在设备列表中，单击具体的设备进入到设备详情页面。

图 6-17 设备-设备详情



页签名	说明
设备信息	• 查看设备信息：查看设备基本信息，包括设备标识码（nodeId）、设备ID（deviceId），节点类型、设备软固件版本信息等。用户也可通过调用**修改设备**接口修改设备的基本信息。 – 设备标识码（nodeId），设备唯一物理标识，如IMEI、MAC地址等，用于设备在接入物联网平台时携带该标识信息完成注册鉴权。 – 设备ID（deviceId），用于唯一标识一个设备，在注册设备时由物联网平台分配获得，是设备在IoT平台上的内部标识，用于设备接入时鉴权，及后续在网络中通过deviceId进行消息传递。 • 重置密钥：密钥用于设备采用原生MQTT、NB-IoT设备、集成SDK的设备接入物联网平台的鉴权认证。重置密钥后，需要将新的密钥信息更新到设备中，设备重新发起注册时，携带新的密钥进行认证。 • 物模型数据：查看最近一次设备上报到平台的数据。
云端运行日志	物联网平台支持记录平台与应用侧及平台与设备侧之间的消息交互情况，您可以在控制台查看这些信息，详细操作请参考**查看运行日志**。
云端下发	您可以在控制台上创建单个设备的命令下发及消息下发（仅MQTT设备支持）任务。详细操作请参考**云端数据下发**。
设备影子	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，可以直接获取下发的命令；设备离线时，上线后可以主动获取下发的命令。详细操作请参考**设备影子**。
消息跟踪	物联网平台支持通过消息跟踪功能进行快速的故障定位和原因分析。详细操作请参考**设备消息跟踪**。

页签名	说明
设备监控	• 设备运行日志：物联网平台支持接收设备上传的日志，您若打开设备日志开关，可将本地日志流转到云日志服务（LTS）（注：此功能仅适用于MQTT设备）。 • 设备异常检测：物联网平台提供设备异常检测功能。详细操作参考**设备异常检测**
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。详细操作参考**网关与子设备**。
标签	物联网平台支持定义不同的标签，并对设备打标签。详细操作请参考**标签**。
群组	物联网平台支持将设备添加到不同群组中，以便处理对海量设备的批量操作。详细操作请参考**群组**。

批量删除设备

设备接入控制台也支持批量删除设备操作。操作方法如下：

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备>所有设备”进入页面，单击“批量删除”页签，再单击“批量删除”。

步骤3 在弹出批量删除设备窗口，先下载“批量删除设备文件模板”，在表格中填写需要删除的设备ID，然后填写“任务名称”，并上传文件，单击“确定”完成设备的批量删除；或者指定产品批量删除设备。

图 6-18 批量删除设备

图 6-19 批量删除设备

界面列表显示任务执行的状态和结果。如果成功率低于100%，则可以单击右侧“详情，进入任务详情，查看执行失败的原因。

----结束

6.4 群组和标签

群组概述

群组是一系列设备的集合，用户可以对资源空间下所有设备，根据区域、类型等不同规则进行分类建立群组，以便处理对海量设备的批量操作。例如，对资源空间下所有水表设备的群组进行固件升级。平台支持群组的增删改查操作，支持给群组绑定和解绑设备，支持一个设备被添加到多个群组中。

表 6-4 群组分类

群组类型	使用说明
静态群组	手动添加设备到群组以及从群组中移除设备；支持群组层级嵌套。 **限制：** • 账号下单实例最多可创建1,000个群组（包含嵌套的子群组）。 • 一个群组内最多添加20,000个设备。 • 一个设备最多可以被添加到10个群组中。 • 群组嵌套关系最大5级。 • 子群组只能归属一个父群组，不支持多父群组。 • 当群组有子群组时不能直接删除，需要先删除子群组才能删除父群组。

群组类型	使用说明
动态群组	按照动态群组规则(设备查询条件，类SQL语句)动态的将符合条件的设备自动添加进群组，不符合条件的自动移除群组；不支持手动管理群组中的设备。 **限制：** ● 账号下单实例最大可创建10个动态群组。 ● 首次创建动态群组，规则最多允许匹配的100,000个设备(增量加入群组的设备无限制)。 ● 动态群组默认为父群组，不支持将动态群组进行嵌套。 ● 动态群组创建成功后，不允许修改动态群组规则。 ● 不允许手动管理动态群组中的设备。 ● 仅标准版实例、企业版实例支持该接口调用，基础版不支持。 ● 单账号创建动态群组的 TPS 限制最大为1/S(每秒1次请求数)。

管理群组

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏，选择“设备 > 群组”。

步骤3 单击相关按钮，进行添加群组、修改群组、删除群组操作。

----结束

静态群组

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏，选择“设备 > 群组”。

步骤3 单击“添加根群组”，进行添加群组，群组类别选择“静态群组”，根据页面提示填写参数，完成后单击“确定”创建群组。

图 6-20 添加静态群组

步骤4 可以进入静态群组详情界面对群组下的设备进行绑定和解绑等操作，具体操作见**表6-5**。

图 6-21 静态群组绑定设备

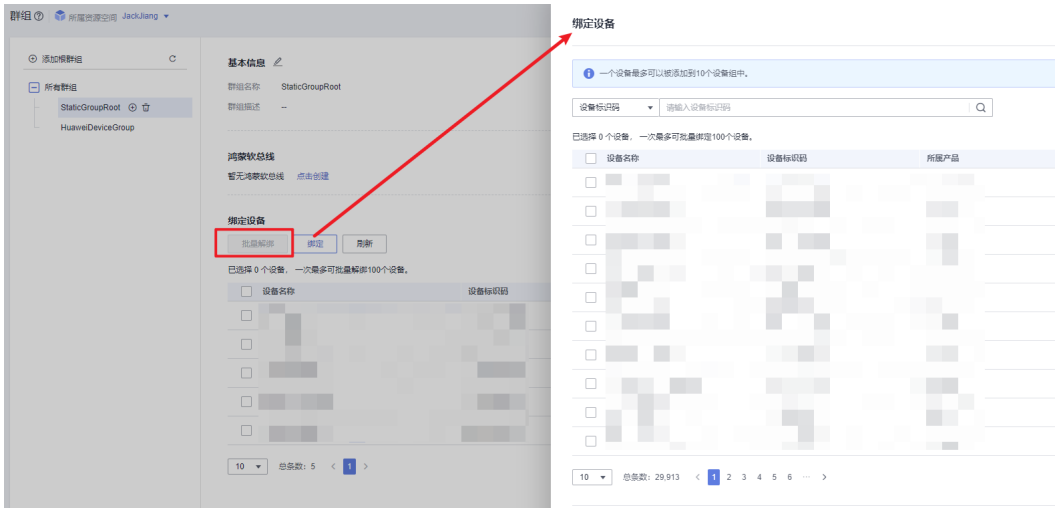


表 6-5 操作说明

操作	操作说明
绑定	单击可绑定设备。
批量解绑	选中多个设备（一次最多选择100个），单击“批量解绑”按钮，可将选择的设备从当前群组中解绑。
解绑	选中设备后，单击“解绑”可将设备从群组中解绑。

步骤5 可以到“设备->设备详情->群组”可以查看当前设备加入的群组信息，并进行相关管理操作，具体操作见**表6-5**。

图 6-22 单设备群组管理

----结束

动态群组

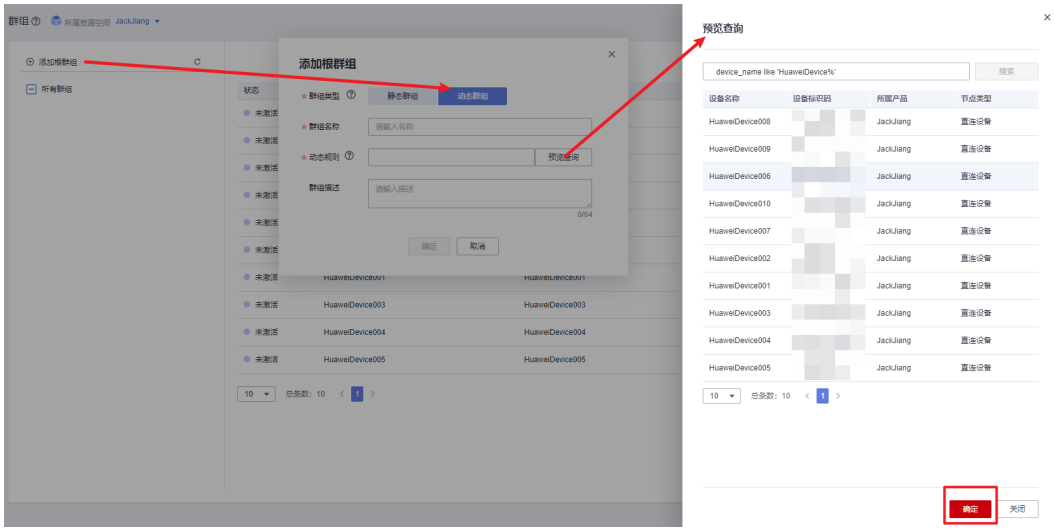
步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏，选择“设备 > 群组”。

步骤3 单击“添加根群组”，进行添加群组，群组类型选择“动态群组”。

步骤4 根据页面提示填写参数，动态规则输入类SQL语句，单击“预览查询”可查看匹配的设备列表，完成后单击“确定”完成动态群组创建。

图 6-23 添加动态群组



说明

- 动态规则语法可参考**高级搜索**。
- 动态群组规则与高级搜索支持字段的不同点：动态群组规则不支持app_id和group_id筛选。
- 动态规则可以直接单击“预览查询”进行填写，填写完成单击“确定”后自动回写该规则。

----结束

动态群组场景示例

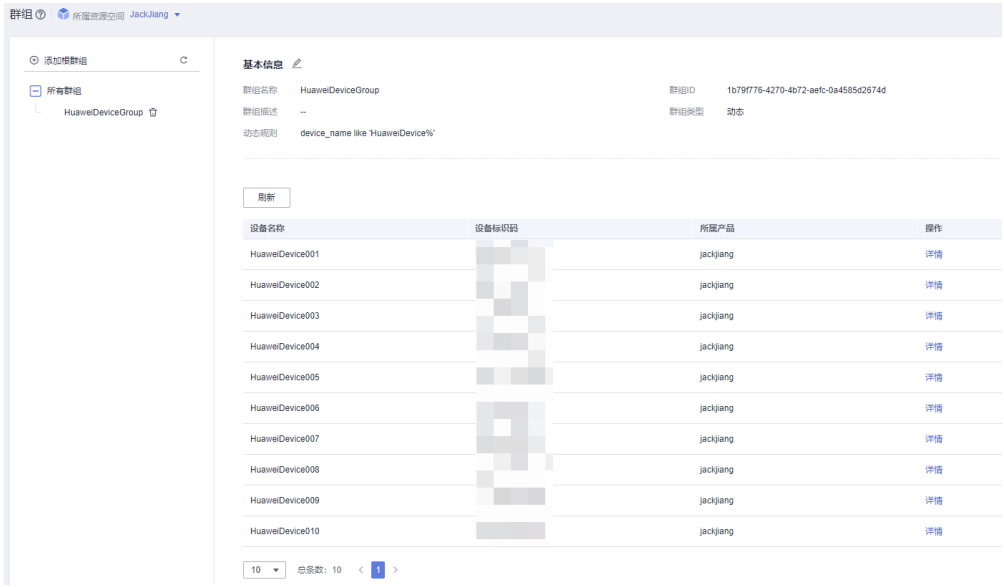
通过设备名称模糊匹配规则（其他条件可以根据实际场景选择）创建动态群组，选择该动态群组执行OTA升级任务。

动态群组中的设备会动态根据设备名称匹配情况进行调整，并且该动态群组关联的OTA升级任务详情状态也会随之动态变化。

具体参考**批量设备固件升级**和**动态群组**的相关步骤进行操作。

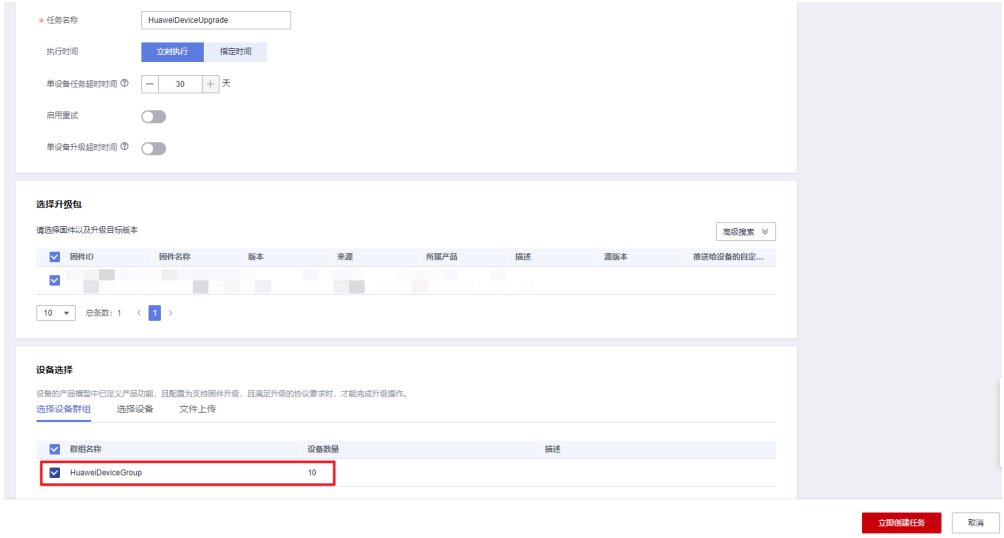
步骤1 创建动态群组，群组名称为“HuaweiDeviceGroup”，群组规则为 “device_name like 'HuaweiDevice%'”。

图 6-24 动态群组详情



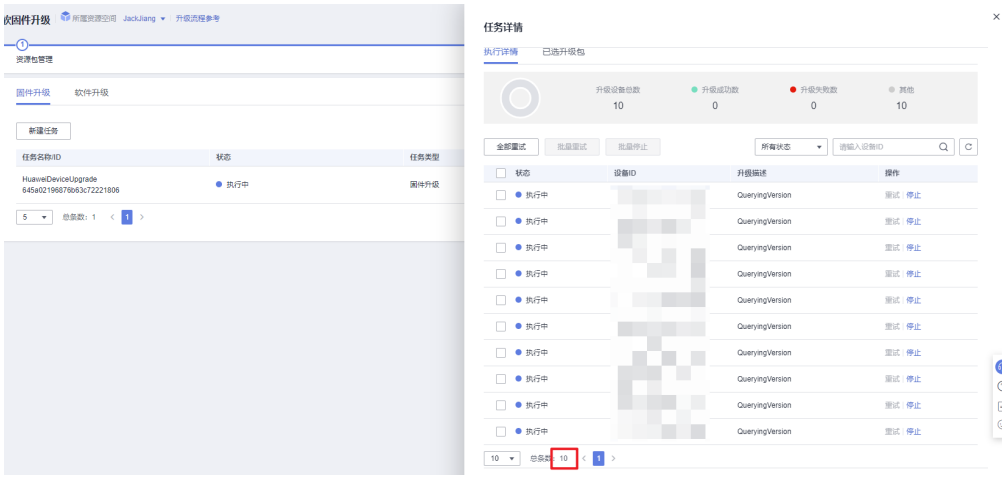
步骤2 创建设备固件升级任务，选择动态群组“HuaweiDeviceGroup”，完成任务创建。

图 6-25 固件升级任务



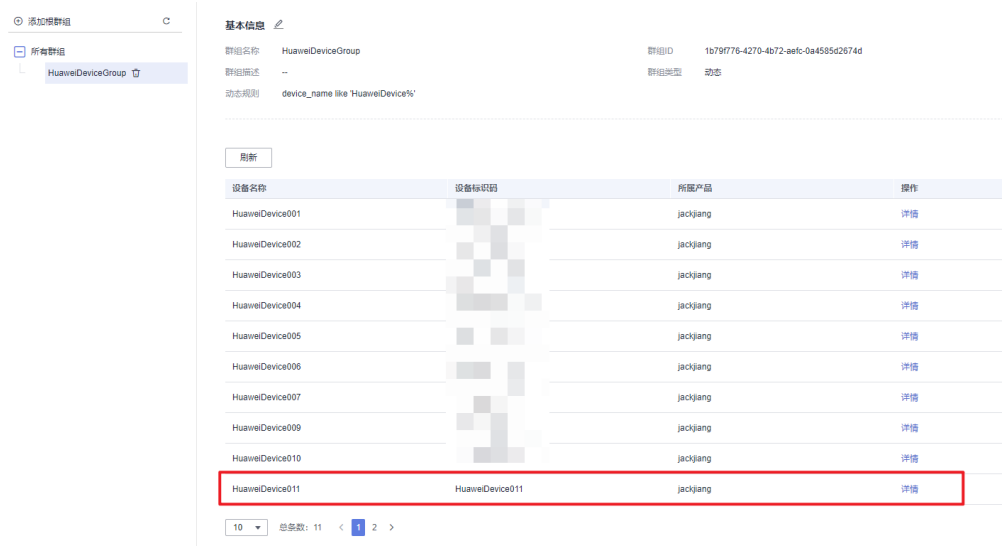
步骤3 创建成功后可以查看动态群组中的设备已加入到该升级任务中。

图 6-26 任务详情



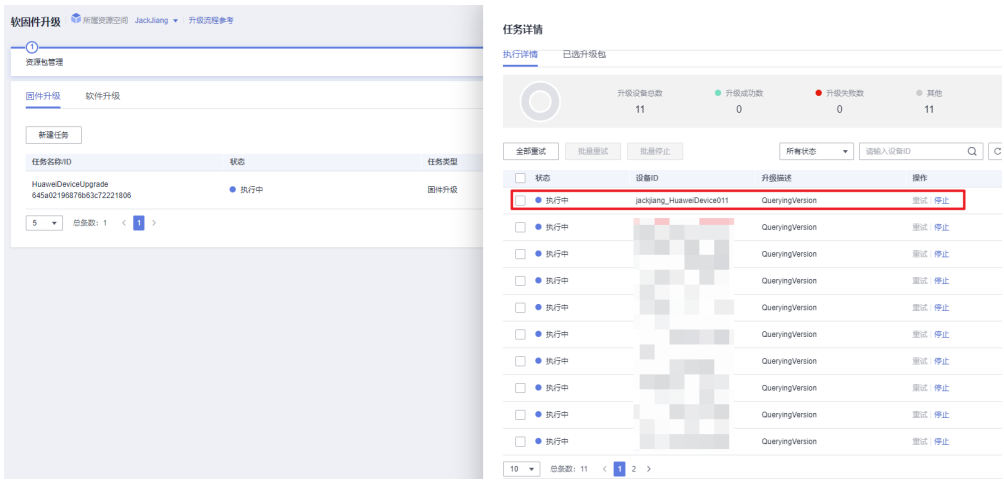
步骤4 参考**注册单个设备**注册设备，设备名称为“HuaweiDevice011”。注册成功后可以查看该设备已自动加入“HuaweiDeviceGroup”动态群组中。

图 6-27 动态群组详情



步骤5 查看软固件升级任务子任务详情，可以看到此设备已自动加入到升级任务中：

图 6-28 固件升级任务

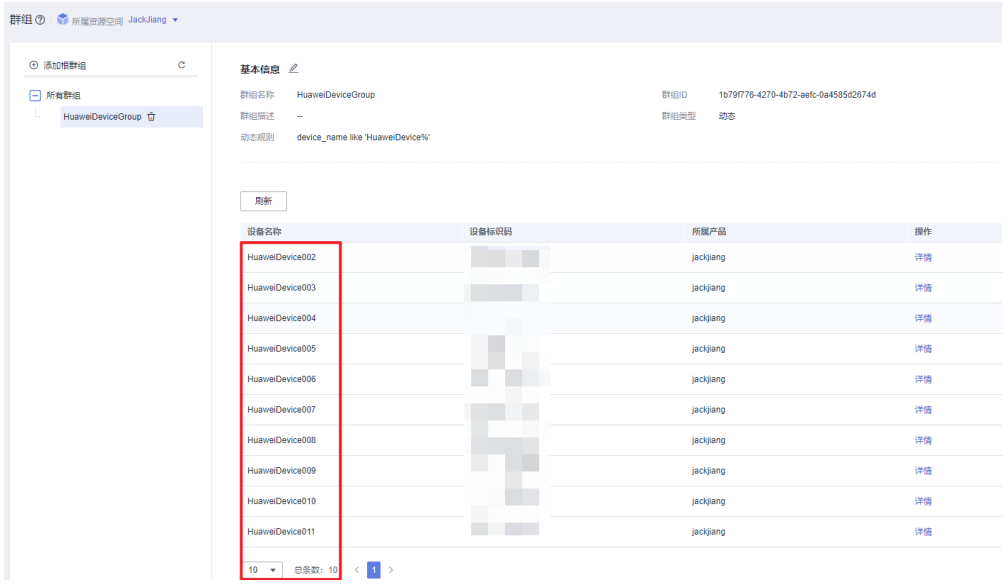


步骤6 进入“HuaweiDevice001”设备详情界面，修改名称为“AbandonedHuaweiDevice001”。

图 6-29 设备详情

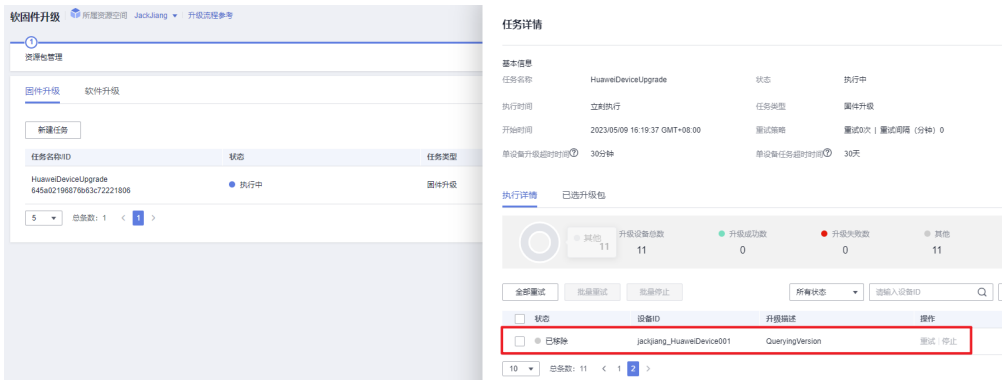
步骤7 修改设备名称成功后，可以查看该设备已自动从“HuaweiDeviceGroup”动态群组中移除。

图 6-30 动态群组详情



步骤8 查看软固件升级任务子任务详情，可以看到此设备的升级状态为“Removed”。

图 6-31 固件升级任务



----结束

标签

标签是一种分类方式，您可以在“设备详情”中为设备绑定标签，以便对设备进行灵活的管理。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备”，单击“查看”进入设备详情。

步骤3 选择“标签”页签，单击“绑定标签”给设备添加标签。

----结束

群组相关 API 接口

查询设备组列表

添加设备组

查询设备组

修改设备组

删除设备组

管理设备组中的设备

查询设备组设备列表

标签相关 API 接口

绑定标签

解绑标签

6.5 设备高级搜索

概述

在海量设备场景下，您可以通过高级搜索功能，通过类SQL语句快速组装灵活的检索条件搜索满足条件的设备，例如：通过前缀模糊搜索设备名称、标签搜索在线设备列表等。本文将介绍高级搜索操作方式以及类SQL语法使用。

使用限制

- 仅标准版实例支持该接口调用，基础版、企业版实例不支持。
- 单账号调用该接口的 TPS 限制最大为1/S(每秒1次请求数)。

使用场景

设备检索：在所有设备-设备列表界面通过类SQL语句方式，检索指定的设备进行后续管理操作。

设备动态分组：通过设备动态分组的类SQL语句规则，过滤出来符合条件的设备自动加入到群组进行管理。

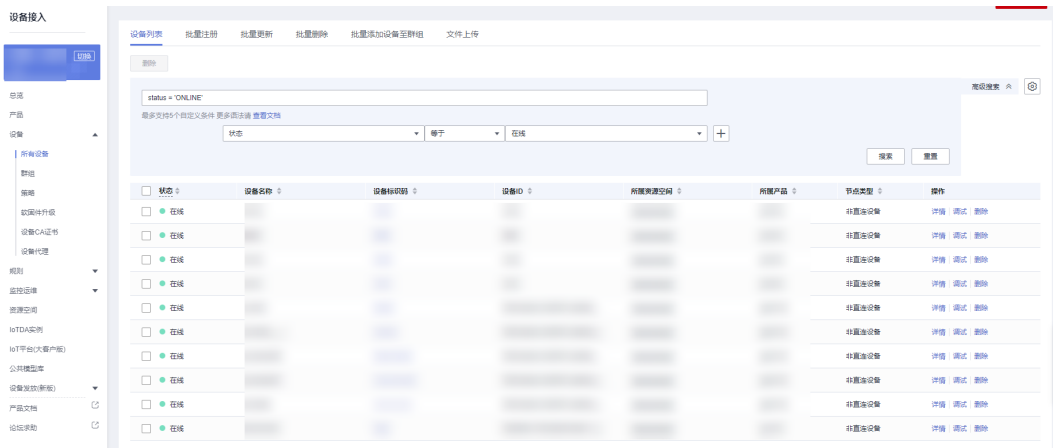
操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台

步骤2 在设备接入控制台左侧导航栏，选择“设备 > 所有设备”，进入设备列表页，设备列表默认显示当前实例下的所有设备。

步骤3 单击“高级搜索”按钮，输入类SQL语句，单击 展示搜索条件下的设备列表。

图 6-32 高级搜索



----结束

类 SQL 语法使用说明

控制台使用类SQL语句时需要省略select、from、order by、limit子句，**仅输入where子句编辑自定义条件即可**，长度限制为400个字符，子句里的内容大小写敏感，SQL语句的关键字大小写不敏感；控制台默认按照marker字段desc方式排序；

where子句格式：

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

示例：

```
product_id = 'testProductId'
```

最多支持5个condition，不支持嵌套；支持的检索字段请参见表6-6和表6-7。

连接词支持AND、OR，优先级参考标准SQL语法，默认AND优先级高于OR。

表 6-6 搜索条件字段说明

字段名	类型	说明	取值范围
app_id	string	资源空间ID	长度不超过36，只允许字母、数字、下划线（_）、连接符（-）的组合。
device_id	string	设备ID	长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
gateway_id	string	网关ID	长度不超过128，只允许字母、数字、下划线（_）、连接符（-）的组合。
product_id	string	设备关联的产品ID	长度不超过36，只允许字母、数字、下划线（_）、连接符（-）的组合。
device_name	string	设备名称	长度不超过256，只允许中文、字母、数字、以及_?'#().,&%@!-等字符的组合。
node_id	string	设备标识码	长度不超过64，只允许字母、数字、下划线（_）、连接符（-）的组合。
status	string	设备的状态	ONLINE(在线)、OFFLINE(离线)、ABNORMAL(异常)、INACTIVE(未激活)、FROZEN(冻结)
node_type	string	设备节点类型	GATEWAY(直连设备或网关)、ENDPOINT(非直连设备)
tag_key	string	标签键	长度不超过64，只允许中文、字母、数字、以及_.-等字符的组合。
tag_value	string	标签值	长度不超过128，只允许中文、字母、数字、以及_.-等字符的组合。
sw_version	string	软件版本	长度不超过64，只允许字母、数字、下划线（_）、连接符（-）、英文点(.)的组合。
fw_version	string	固件版本	长度不超过64，只允许字母、数字、下划线（_）、连接符（-）、英文点(.)的组合。

字段名	类型	说明	取值范围
group_id	string	群组id	长度不超过36，十六进制字符串和连接符（-）的组合。
create_time	string	设备注册时间	格式：yyyy-MM-dd'T'HH:mm:ss.SSS'Z'，如：2015-06-06T12:10:10.000Z。
marker	string	结果记录ID	长度为24的十六进制字符串，如ffffffffffffffffffffffff。

表 6-7 支持的运算符

运算符	支持的字段
=	所有
!=	所有
>	create_time、marker
<	create_time、marker
like	device_name、node_id、tag_key、tag_value
in	除tag_key、tag_value以外字段
not in	除tag_key、tag_value以外字段

SQL 限制

- like：只支持前缀匹配，不支持后缀匹配或者通配符匹配。前缀匹配不得少于4个字符，且不能包含任何特殊字符（只允许中文、字母、数字、下划线（_）、连接符（-））），前缀后必须跟上“%”结尾。
- 不支持其他SQL用法，如嵌套SQL、union、join、别名(Alias)等用法。
- SQL长度限制为400个字符，单个请求条件最大支持5个。
- 不支持“null”和空字符串等条件值匹配。

相关 API 接口：

灵活搜索设备列表

6.6 设备影子

概述

物联网平台支持创建设备的“影子”。设备影子是一个JSON文件，用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。每个设备有

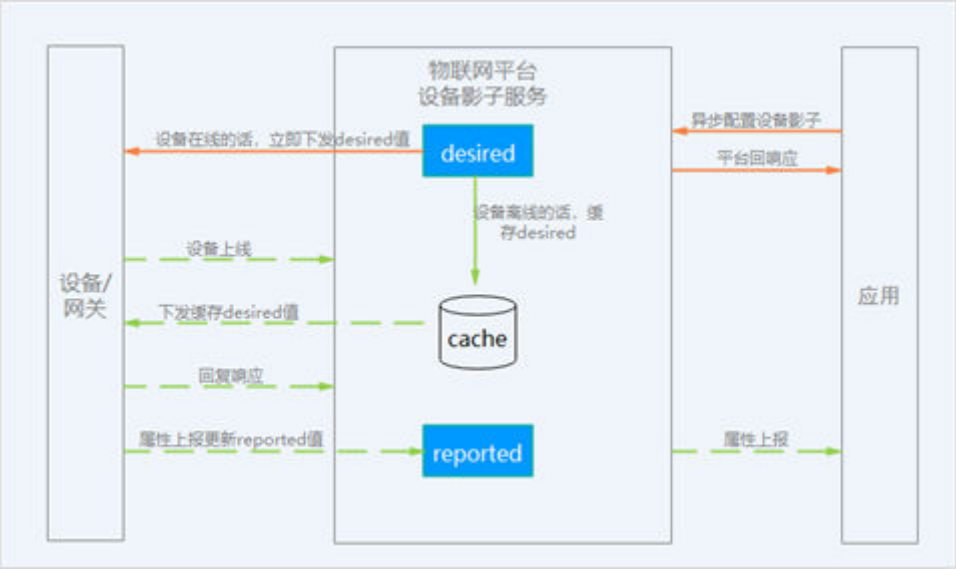
且只有一个设备影子，设备可以获取和设置设备影子以此来同步设备属性值，这个同步可以是影子同步给设备，也可以是设备同步给影子。

设备影子上有desired区和reported区。

- desired区用于存储对设备属性的配置，即期望值。当需要修改设备的服务属性值时，可修改设备影子的desired区的属性值，设备在线时，desired属性值立即同步到设备。如果设备不在线，待设备上线或上报数据时，desired属性值同步到设备。
- reported区用于存储设备最新上报的设备属性值，即上报值。当设备上报数据时，平台刷新reported区属性值为设备上报的设备属性值。

说明

- 设备影子可以通过调用应用侧API接口配置，也可以通过登录控制台，在设备详情->设备影子->属性配置页面配置。（设备影子主要针对设备属性配置，它的配置依赖**产品模型**）。
- 设备影子配置属于异步命令，物联网平台会直接回复配置响应，然后平台通过设备在线状态，决定立即下发还是缓存下发。
- 设备上线后，影子服务会下发desired值给设备，待设备属性上报时，影子服务检查属性值与下发的desired值是否匹配。若匹配，则说明影子数据在设备侧配置成功，缓存清除；若不匹配，则说明影子数据在设备侧未配置成功，在下次设备上线或属性上报时，会继续下发缓存desired值给设备，直到下发配置成功。
- 限制：设备影子JSON文档中的key不允许特殊字符：点(.)、dollar符号($)、空char(十六进制的ASCII码为00)。如果包含了以上特殊字符则无法正常刷新影子文档。
- 设备影子desired配置给设备后，需要设备回响应表示已收到请求。如果设备不回响应(平台则认为设备侧未适配影子设置流程，**平台设置设备属性**)，则平台有个5 分钟的保护期，避免过多流量冲击设备，保护期内设备属性上报时，平台比对reported和desired即使有差异，也不会下发差值。设备正常适配下发流程中的属性设置响应，平台则每次属性上报都会将差值下发给设备。



应用场景

适合资源受限低功耗设备，长期处于休眠状态的场景。

- 查询设备最新上报数据和设备最新在线状态：
  - 当在控制台上查询设备上报数据时，由于设备可能长时间处于离线状态或因网络不稳定掉线，而无法获取到最新数据。通过设备影子机制，设备影子中

始终保持设备最新上报的数据和设备当前状态，控制台上只需要查询设备影子中存储的数据，即可获取设备最新上报的数据和设备状态。

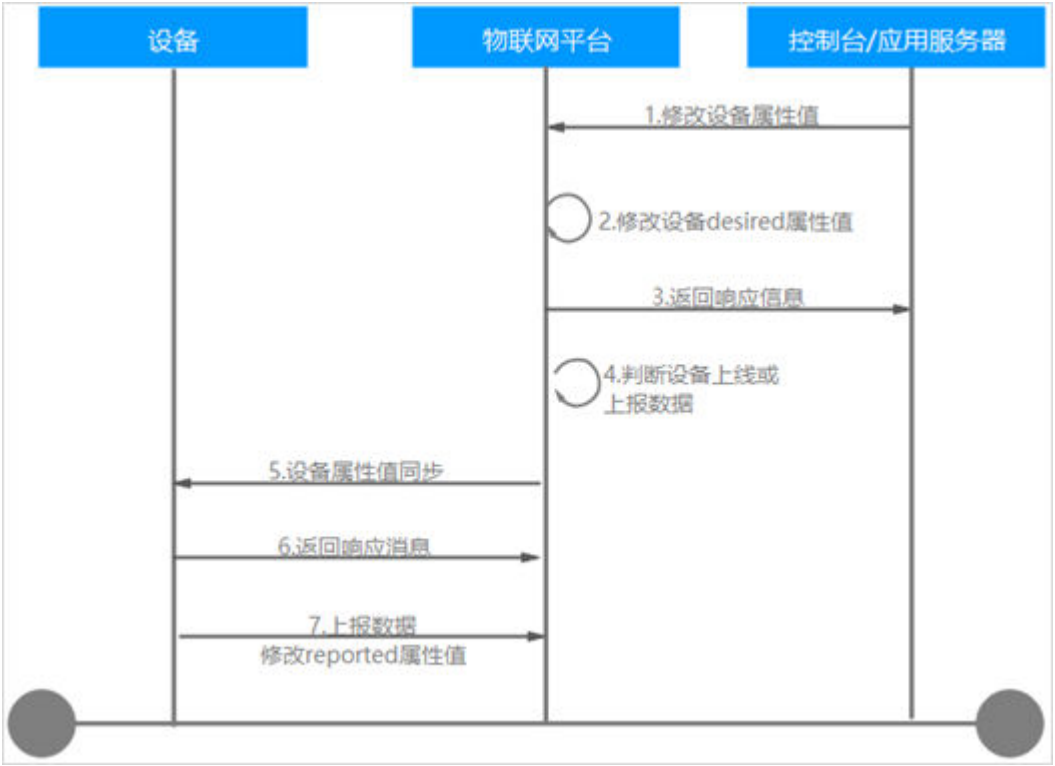
- 很多应用服务器频繁的查询设备在线状态，由于设备处理能力有限，频繁查询会损耗设备性能。使用设备影子机制，设备只需要主动同步状态给设备影子一次，多个应用程序请求设备影子获取设备状态，即可获取设备最新状态，从而将应用程序和设备解耦。

- 修改设备属性值：用户通过“设备 > 设备详情 > 设备影子”修改设备的属性值。由于设备可能长时间处于离线状态，修改设备属性值的操作不能及时下发给设备。在这种情况下，物联网平台可以将修改设备的属性信息存储在设备影子中，待设备上线后，将修改的设备属性值同步给设备，从而完成设备属性值的修改。

业务流程

修改设备属性值

修改desired区属性值，如果设备在线，则设备影子直接同步设备属性值到设备，否则等待设备上线或上报数据时，再同步设备属性值到设备。



1. 用户通过控制台或应用服务器修改设备属性值。消息样例如下：

```
PUT https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/shadow
Content-Type: application/json
X-Auth-Token: ********
Instance-Id: ********

{
  "shadow" : [ {
    "desired" : {
      "temperature" : "60"
    },
    "service_id" : "WaterMeter",
    "version" : 1
  } ]
}
```

2. 物联网平台修改desired区属性值。
3. 物联网平台返回响应消息。
4. 物联网平台判断设备上线或上报数据。
5. 物联网平台将设备属性同步到设备。消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/properties/set/request_id={request_id}
数据格式：
{
    "object_device_id": "{object_device_id} ",
    "services": [
        {
            "service_id": "Temperature",
            "properties": {
                "value": 57,
                "value2": 60
            }
        },
        {
            "service_id": "Battery",
            "properties": {
                "level": 80,
                "level2": 90
            }
        }
    ]
}
```

6. 设备返回响应消息。设备影子desired区的属性值发送给设备后，需要设备回响应表示已收到请求。消息样例如下：

```
Topic：$oc/devices/{device_id}/sys/properties/set/response/request_id={request_id}
数据格式：
{
    "result_code": 0,
    "result_desc": "success"
}
```

7.设备上报数据，当设备进行属性上报时，平台会存储设备最新上报的设备属性值。

- 设备上报属性时，物联网平台修改设备影子reported区属性值为设备上报的设备属性值。消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/properties/report
数据格式：
{
    "services": [
        {
            "service_id": "Temperature",
            "properties": {
                "value": 57,
                "value2": 60
            },
            "event_time": "20151212T121212Z"
        },
        {
            "service_id": "Battery",
            "properties": {
                "level": 80,
                "level2": 90
            },
            "event_time": "20151212T121212Z"
        }
    ]
}
```

- 设备主动删除设备影子的reported区
  - 设备主动删除reported区service下的单个属性

    设备上报属性时，将属性设置为null，平台会将该属性从设备影子reported区删除，消息样例如下：

```
Topic: $oc/devices/{device_id}/sys/properties/report
{
    "services": [
        {
            "service_id": "Temperature",
            "properties": {
                "value": null,
                "value2": 60
            },
            "event_time": "20151212T121212Z"
        }
    ]
}
```

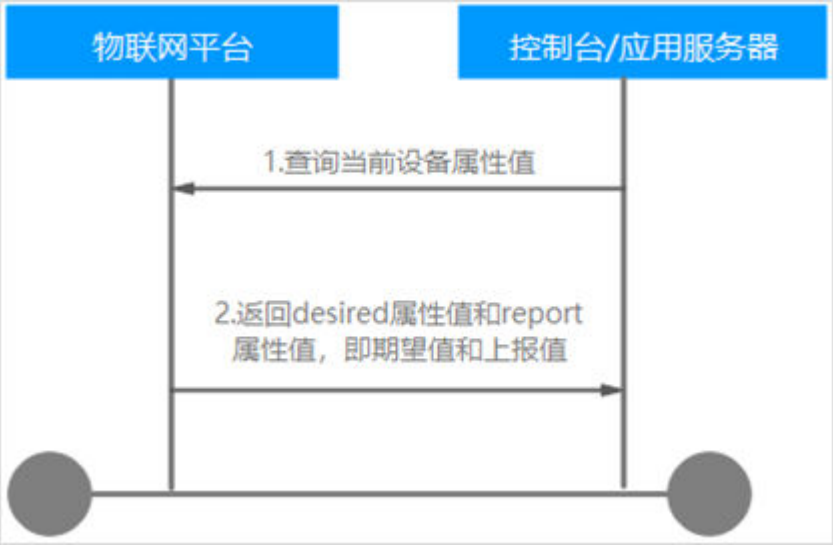
- 设备主动删除影子reported区的service下的全部的属性

  设备上报属性时，将service对应的properties设置为{}时，平台会将reported区该service模块下所有属性从设备影子reported区删除，消息样例如下:

```
Topic: $oc/devices/{device_id}/sys/properties/report
{
    "services": [
        {
            "service_id": "Temperature",
            "properties": {},
            "event_time": "20151212T121212Z"
        }
    ]
}
```

查询设备属性值

设备影子保存的是设备最新的设备属性值，一旦设备属性值产生变化，设备会将设备属性值同步到设备影子。用户便可以及时获取查询结果，无需关注设备是否在线。



1. 用户通过控制台或应用服务器查询设备属性值。消息样例如下:

```
GET https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/shadow
Content-Type: application/json
X-Auth-Token: ********
Instance-Id: ********
```

2. 物联网平台返回desired属性值和report属性值，即期望值和上报值。消息样例如下:

   Status Code: 200 OK

```
Content-Type: application/json

{
```

```
"device_id" : "40fe3542-f4cc-4b6a-98c3-61a49ba1acd4",
"shadow" : [ {
  "desired" : {
    "properties" : {
      "temperature" : "60"
    },
    "event_time" : "20151212T121212Z"
  },
  "service_id" : "WaterMeter",
  "reported" : {
    "properties" : {
      "temperature" : "60"
    },
    "event_time" : "20151212T121212Z"
  },
  "version" : 1
} ]
}
```

查询和修改设备影子

查询设备影子

方法1：应用服务器调用**查询设备影子数据**接口。

方法2：登录**管理控制台**，在左侧导航栏选择“设备”，单击具体的设备进入到设备的详情页面，在“设备影子”页签中，可以查看当前设备属性数据，包括“上报值”和“期望值”。

- 如果当前界面中看到“上报值”与“期望值”不一致，原因可能是设备未在线，暂时存储在设备影子中，待同步给设备，期望值会存在深色底纹。
- 如果当前界面看到的“上报值”与“期望值”一致，则表示设备最近一次上报的属性值与用户期望下发的属性值一致，期望值为白色底纹。

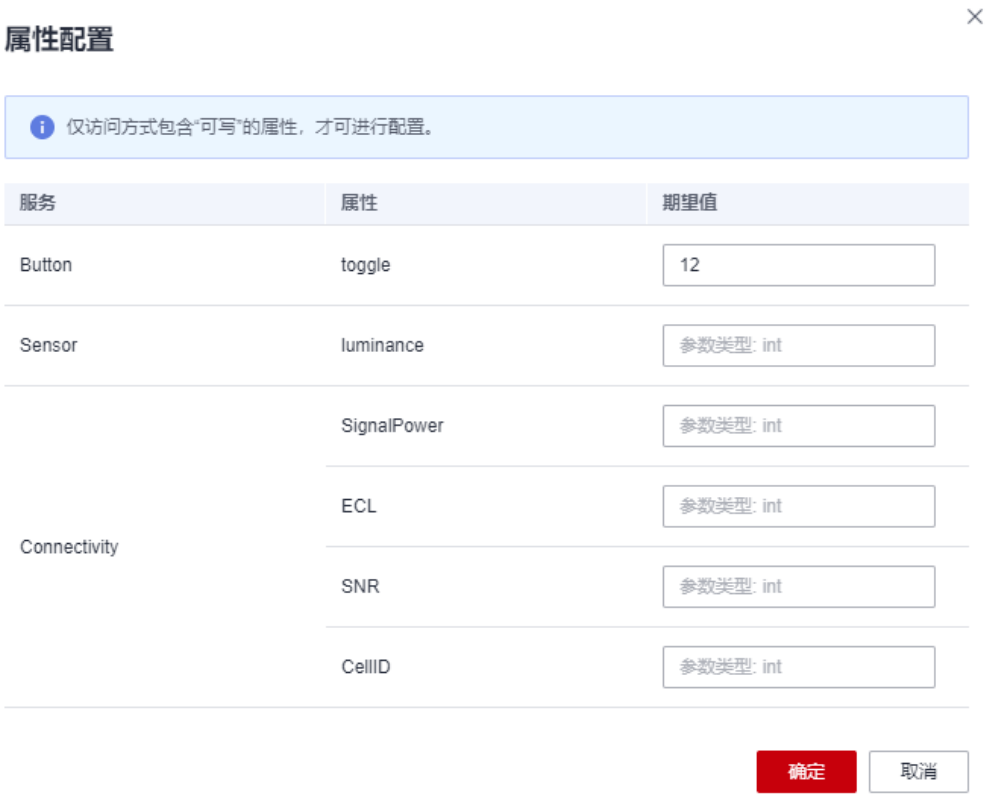
图 6-33 设备影子-查看

修改设备影子

方法1：应用服务器调用**配置设备影子预期数据**接口。

方法2：登录**管理控制台**，在左侧导航栏选择“设备-所有设备”，在设备列表中单击具体的设备进入到设备的详情页面，在“设备影子”页面，单击“属性配置”，在弹出窗口中输入服务属性对应的期望值，单击"确定"完成设备影子的修改。

图 6-34 设备影子-属性配置



相关 API 接口

查询设备影子数据

配置设备影子预期数据

6.7 OTA 升级

6.7.1 软固件包上传

概述

软件（Software）一般分为系统软件和应用软件，系统软件实现设备最基本的功能，比如编译工具、系统文件管理等；应用软件可以根据设备的特点，提供不同的功能，比如采集数据、数据分析处理等。软件升级又称为SOTA（SoftWare Over The Air），是指用户可以通过OTA的方式支持对LwM2M协议和MQTT协议的设备进行软件升级。

- 基于LwM2M协议的产品模型，软件升级遵循的协议为PCP协议（**PCP协议介绍**），设备侧需要遵循PCP协议进行软件升级的适配开发，适配方法请参考**设备侧适配开发指导**。
- 基于MQTT协议的产品模型，不校验软件升级协议类型。

固件（Firmware）一般是指设备硬件的底层“驱动程序”，承担着一个系统最基础最底层工作的软件，比如计算机主板上的基本输入/输出系统BIOS（Basic Input/output

System)。固件升级又称为FOTA（Firmware Over The Air），华为云用户可以通过OTA的方式对支持LwM2M协议和MQTT协议的设备进行固件升级。

操作步骤

华为物联网平台进行软固件升级，需要在console平台添加升级包。平台支持升级包关联OBS服务桶中对象和本地文件上传升级包两种方式。

说明

- 升级包关联OBS服务桶中对象，升级包大小限制最大为1G，由于升级文件保存在OBS服务中，会产生额外的存储和下载费用。

  OBS计费项由存储费用、请求费用、流量费用、数据恢复费用和数据处理费用组成。计费模式分为按需计费和包年包月。具体费用请参考**OBS计费说明**

  例如：在华北-北京四区域下，用户一个月需要升级1万设备，升级包的大小为100M，如果按需计费，则需要存储费用￥0.139，流量费用￥512，请求次数费用￥0.01，总计￥512.15。

  如果包年包月，则需要存储费用￥1.00，流量费用￥505.00（1TB 公网流出流量包1个月价格），请求次数费用￥0.01，总计￥506.01。
- 本地文件上传升级包的方式，不收取费用，但是升级包大小限制最大为20M。
- 升级包只支持.bin、.dav、.tar、.gz、.zip、.gzip、.apk、.tar.gz、.tar.xz、.pack、.exe、.bat、.img格式的文件。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择“设备 > 软固件升级”。

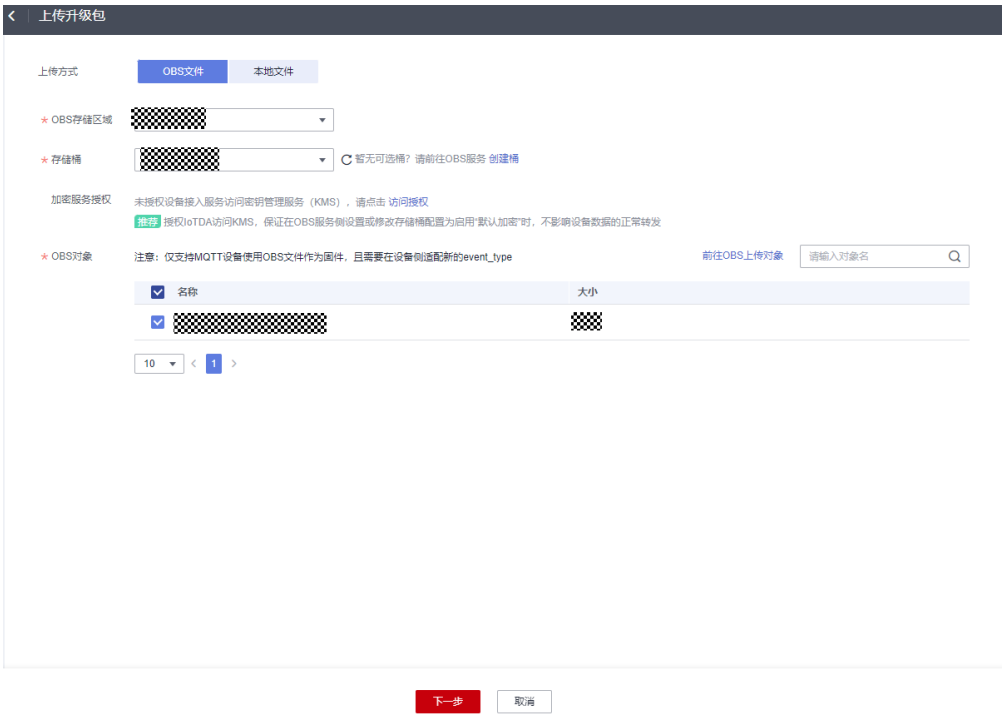
步骤3 上传固件升级包时，选择“固件列表”页签，单击“上传固件”，上传软件升级包时，选择“软件列表”页签，单击“上传软件”。

步骤4 在上传升级包界面，选择上传方式OBS文件或者本地文件

- OBS文件，使用前请单击“访问授权”弹出授权窗口后单击“同意授权”。授权后请选择升级文件所在的桶。如果暂时还没有创建桶，可以单击“创建桶”跳转到OBS页面创建桶。如果没有上传升级文件到对应的桶，可以单击“前往OBS上传对象”跳转到OBS对应桶页面上传升级文件。勾选需要的OBS对象后单击按钮“下一步”。

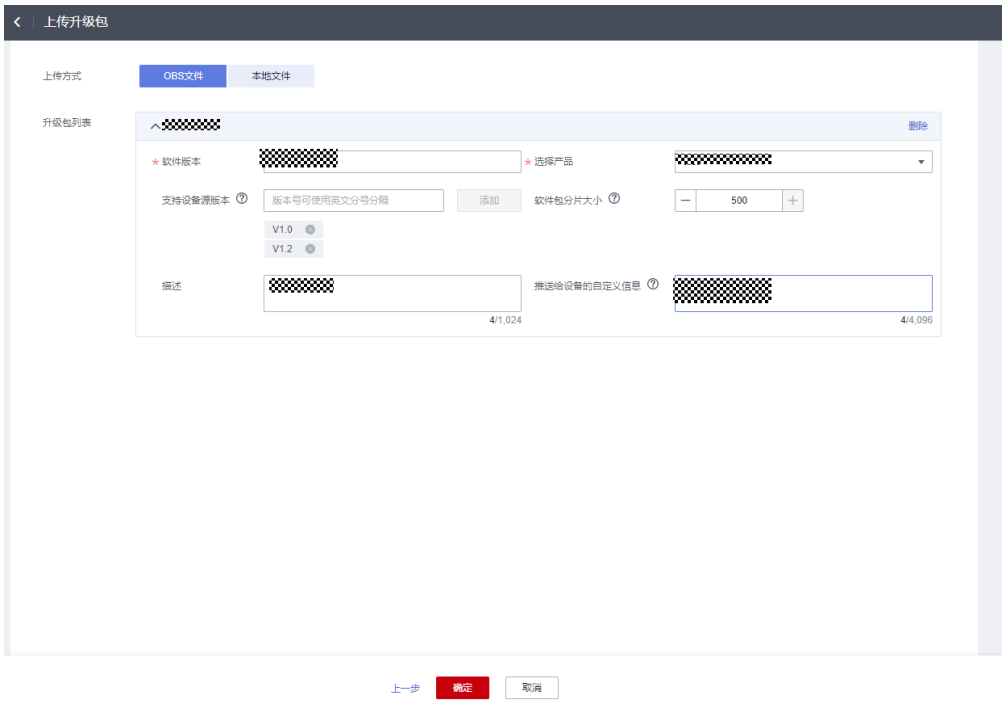
须知

如果设备接入服务未访问授权密钥管理服务（KMS），请单击“加密服务授权”处的“访问授权”进行授权访问，否则在OBS服务侧设置或修改存储桶配置为启用“默认加密”时，会影响升级文件的下载。



- 本地文件，请拖拽或者单击“添加文件”上传软固件升级包。

步骤5 选择OBS对象或上传本地文件后，在升级包列表里面按照如下信息填写参数后，单击“确定”上传软件包。



在上传软固件包时，需要填写如下信息：

参数名称	说明
固件(软件)版本	固件(软件)包的版本。（设备升级完成后需要上报升级版本号，平台检查设备上报的版本号是否与此参数一致，一致为升级成功。）
选择产品	选择对应设备的**产品模型**。
支持设备源版本	支持升级的设备的源版本号。手动输入，如需输入多个，可以在输入完一个版本后，单击“回车”按键，再输入下一个。 **说明** 平台目前暂不支持自动差分升级包的功能，用户可以自己本地差分后上传到平台，并对不同的差分包指定不同的支持升级的设备源版本号。在创建升级任务时可以选择多个差分包创建升级任务。在软固件升级过程中，平台将根据设备上报的源版本号，下发不同的差分包。
软件包分片大小	终端下载软件包的每个分片的大小，单位为byte。取值范围：32~500，默认值500。只有NB协议的设备软件升级支持该功能。
描述	软固件包的描述信息。
推送给设备的自定义信息	平台下发**升级通知**时，会同时下发该自定义信息给设备端。

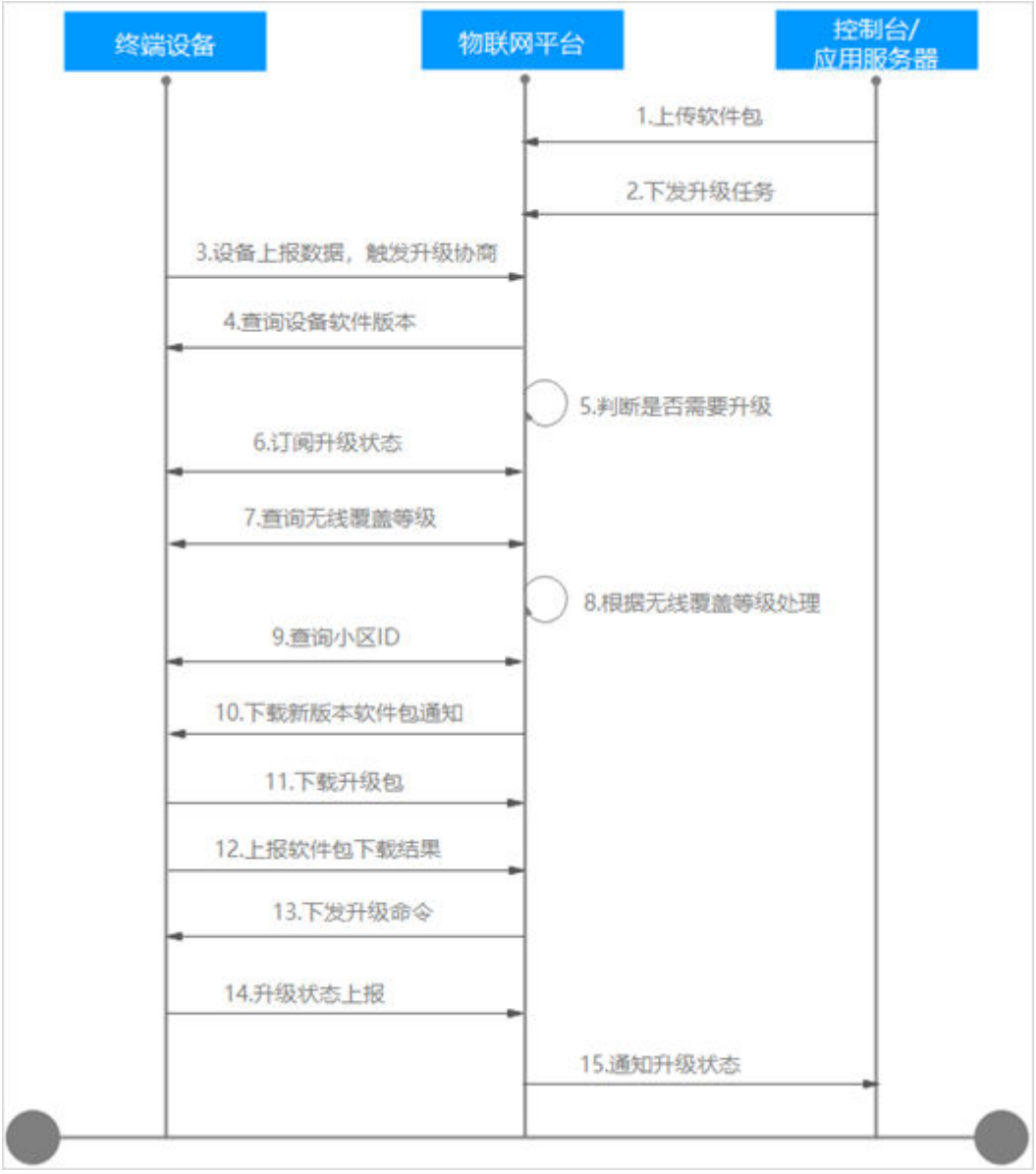
说明

- 平台已经下线上传签名软固件包功能，之前已经上传的签名软固件包能正常升级使用。为了保障您能够正常使用软固件升级功能，请直接上传需要下发给设备的升级文件。
- 只有MQTT协议设备支持使用OBS文件作为软固件升级包，且需要在设备侧适配新的**event_type**。
- 升级包如果不指定支持设备源版本，则软固件升级时将对所有选择的设备进行升级。

----结束

6.7.2 NB-IoT 设备 OTA 升级

LwM2M 协议设备软件升级流程



LwM2M协议SOTA升级流程的详细说明：

1～2. 用户在设备管理服务的控制台上传软件包，并在控制台或者应用服务器上创建软件升级任务。

3. LwM2M设备上报数据，平台感知设备上线，触发升级协商流程。(超时时间为24小时)

4～5. 物联网平台向设备下发查询设备软件版本的命令，查询成功后，物联网平台根据升级的目标版本判断设备是否需要升级。（第4步等待设备上报软件版本，超时时间为3分钟）

- 如果返回的软件版本信息与升级的目标版本信息相同，则升级流程结束，不做升级处理。
- 如果返回的软件版本信息与升级的目标版本信息不同，则继续进行下一步的升级处理。

6. 物联网平台向设备订阅软件升级的状态。

7～8. 物联网平台查询终端设备所在的无线信号覆盖情况，获取小区ID、RSRP（Reference Signal Received Power，参考信号接收功率）和SINR（Signal to Interference Plus Noise Ratio，信号干扰噪声比）信息。（等待上报无线覆盖等级和小区ID，超时时间为3分钟左右）

- 查询成功：则根据如下方式计算可同时升级的并发数计算，并按照**步骤10**进行处理。
  - 如下图所示，如果设备的RSRP强度和SINR强度均落在等级“0”中，则同时可以对该小区的50个相同信号覆盖区间的设备进行同时升级。
  - 如果设备的RSRP强度和SINR强度分别落在等级“0”和“1”中，则以信号较弱的等级“1”为准，则只能同时对该小区的10个设备进行升级。
  - 如果设备的RSRP强度和SINR强度分别落在等级“1”和“2”中，则以信号较弱的等级“2”为准，则只能同时对该小区的1个设备进行升级。
  - 如果设备的RSRP强度和SINR强度不在该3个等级范围内，且均可以查询到，则按照信号最弱覆盖等级“2”处理，则只能同时对1个设备进行升级。

无线覆盖等级	RSRP门限范围 (dBm)	SINR门限范围 (dB)
0	-105 <= RSRP	7 <= SINR
1	-115 <= RSRP < -105	-3 <= SINR < 7
2	-125 <= RSRP < -115	-8 <= SINR < -3

说明

如果用户在软件升级中发现同时进行升级的设备数较少，则可以联系当地运营商检查和优化设备所在小区的无线覆盖情况。

- 查询失败：则按照流程**9**进行处理。

9. 物联网平台继续下发查询小区ID信息的命令，获取终端设备所在的小区ID信息。

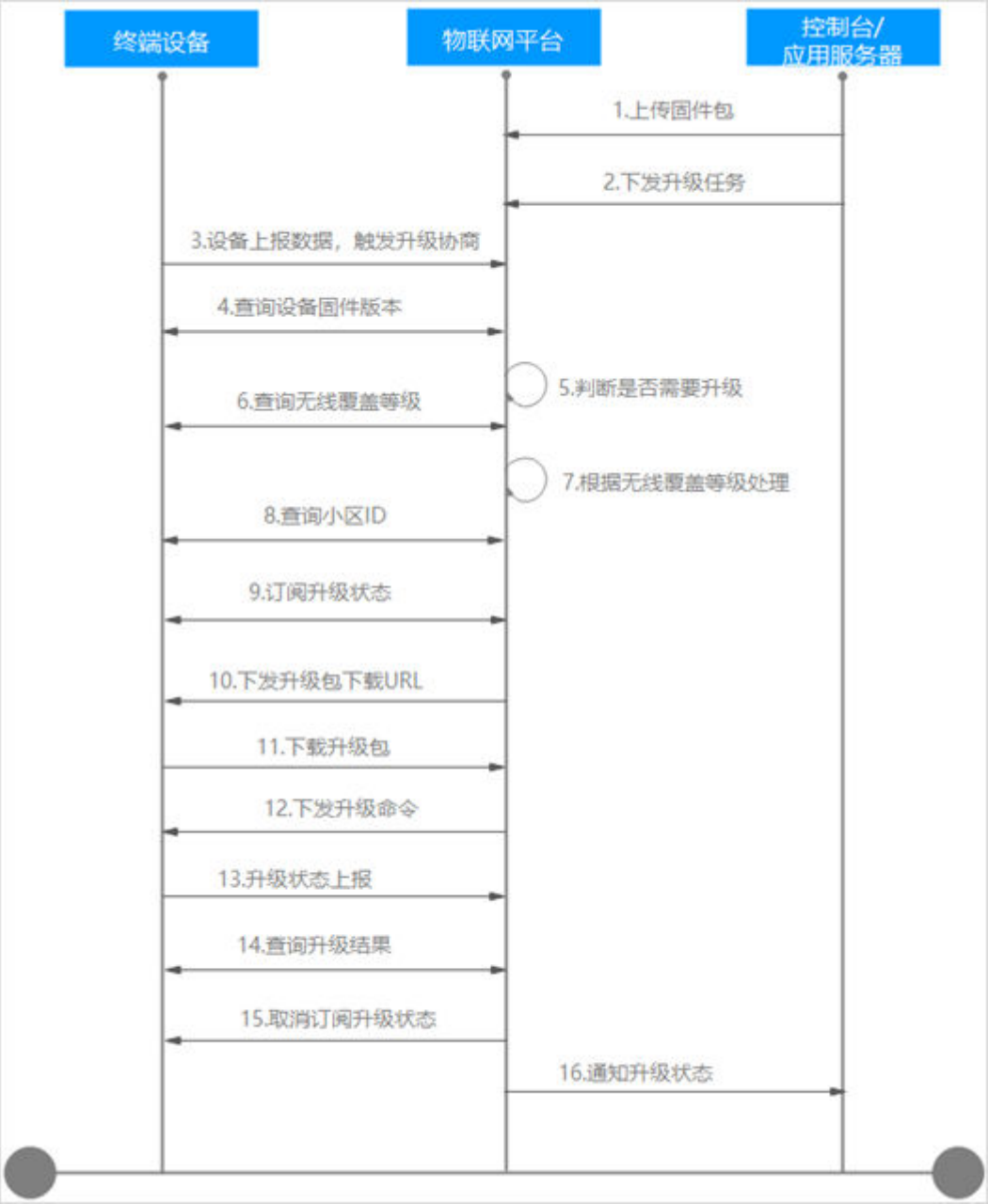
- 如果查询成功：物联网平台支持同时对该小区的10个相同情况的设备进行软件升级。
- 如果查询失败：则升级失败。

10～12. 物联网平台通知设备有新的软件包版本，设备启动软件包的下载。软件包的下载按照分片的方式进行下载，支持断点续传功能，通过软件包分片中携带的“versionCheckCode”确定是否属于同一个软件包。下载完成后，设备知会物联网平台软件包已下载完毕。（第11步超时时间为60分钟）

13～14. 物联网平台向设备下发升级的命令，终端设备进行升级操作，升级完成后终端设备向物联网平台反馈升级的结果。（等待设备上报升级结果和升级状态，超时时间为30分钟）

15. 物联网平台向控制台/应用服务器通知升级的结果。

LwM2M 协议设备固件升级流程



LwM2M协议FOTA升级流程的详细说明：

1～2. 用户在设备接入服务的控制台上传固件包，并在控制台或者应用服务器上创建固件升级任务。

3. LwM2M设备上报数据，平台感知设备上线，触发升级协商流程。(超时时间为24小时)

4～5. 物联网平台向设备下发查询设备固件版本的命令，查询成功后，物联网平台根据升级的目标版本判断设备是否需要升级。（第4步等待设备上报固件版本，超时时间为3分钟）

- 如果返回的固件版本信息与升级的目标版本信息相同，则升级流程结束，不做升级处理。
- 如果返回的固件版本信息与升级的目标版本信息不同，则继续进行下一步的升级处理。

6～7. 物联网平台查询终端设备所在的无线信号覆盖情况，获取小区ID、RSRP（Reference Signal Received Power，参考信号接收功率）和SINR（Signal to Interference Plus Noise Ratio，信号干扰噪声比）信息。（等待上报无线覆盖等级和小区ID，超时时间为3分钟左右）

- 查询成功：则根据如下方式计算可同时升级的并发数计算，并按照**步骤9**进行处理。
  - 如下图所示，如果设备的RSRP强度和SINR强度均落在等级“0”中，则同时可以对该小区的50个相同信号覆盖区间的设备进行同时升级。
  - 如果设备的RSRP强度和SINR强度分别落在等级“0”和“1”中，则以信号较弱的等级“1”为准，则只能同时对该小区的10个设备进行升级。
  - 如果设备的RSRP强度和SINR强度分别落在等级“1”和“2”中，则以信号较弱的等级“2”为准，则只能同时对该小区的1个设备进行升级。
  - 如果设备的RSRP强度和SINR强度不在该3个等级范围内，且均可以查询到，则按照信号最弱覆盖等级“2”处理，则只能同时对1个设备进行升级。

无线覆盖等级	RSRP门限范围 (dBm)	SINR门限范围 (dB)
0	-105 <= RSRP	7 <= SINR
1	-115 <= RSRP < -105	-3 <= SINR < 7
2	-125 <= RSRP < -115	-8 <= SINR < -3

说明

如果用户在固件升级中发现同时进行升级的设备数较少，则可以联系当地运营商检查和优化设备所在小区的无线覆盖情况。

- 查询失败：则按照流程**步骤8**进行处理。

8. 物联网平台继续下发查询小区ID信息的命令，获取终端设备所在的小区ID信息。

- 如果查询成功：物联网平台支持同时对该小区的10个相同情况的设备进行固件升级。
- 如果查询失败：则升级失败。

9. 物联网平台向设备订阅固件升级的状态。

10～11. 物联网平台向设备下发下载固件包的URL地址，通知设备下载固件包。终端设备根据该URL地址下载固件包，固件包的下载支持分片下载，下载完成后，设备知会物联网平台固件包已下载完毕。（第11步超时时间为60分钟）

12～13. 物联网平台向设备下发升级的命令，终端设备进行升级操作，升级完成后终端设备向物联网平台反馈升级结束。（等待设备上报升级结果和升级状态，超时时间为30分钟）

14～16. 物联网平台下发命令查询固件升级的结果，获取升级结果后，向终端设备取消订阅升级状态通知，并向控制台应用服务器通知升级的结果。

说明

在下载包中断的情况下，平台支持断点续传功能。

固件升级失败原因

物联网平台上报的失败原因：

失败原因	原因解释	处理建议
Device Abnormal is not online	设备异常未在线	请检查设备侧。
Task Conflict	任务冲突	请检查当前设备是否有软件升级、固件升级、日志收集或设备重启的任务正在进行。
Waiting for the device online timeout	等待设备上线超时	请检查设备侧。
Wait for the device to report upgrade result timeout	等待设备上报升级结果超时	请检查设备侧。
Waiting for report device firmware version timeout	等待上报设备固件版本超时	请检查设备侧。
Waiting for report cellId timeout	等待上报cellId超时	请检查设备侧。
Updating timeout and query device version for check timeout	等待升级结果超时，且等待设备版本信息超时	请检查设备侧。
Waiting for device downloaded package timeout	等待设备完成下载固件包超时	请检查设备侧。
Waiting for device start to update timeout	等待设备启动更新超时	请检查设备侧。
Waiting for device start download package timeout	等到设备开始下载固件包超时	请检查设备侧。

设备上报的失败原因：

失败原因	原因解释	处理建议
Not enough storage for the new firmware package	下载的固件包存储空间不足	请检查设备存储。
Out of memory during downloading process	下载过程中内存不足	请检查设备内存。
Connection lost during downloading process	下载过程中连接断开	请检查设备连接状态。
Integrity check failure for new downloaded package	下载的固件包完整性校验失败	请检查设备下载的固件包是否完整。
Unsupported package type	固件包类型不支持	请检查设备状态和厂商提供的固件包是否正确。
Invalid URI	URI不可用	检查设备侧的固件包下载地址是否正确。
Firmware update failed	固件更新失败	请检查设备侧。

常见问题

软/固件升级业务热点咨询问题如下，更多咨询问题请访问**查看更多**。

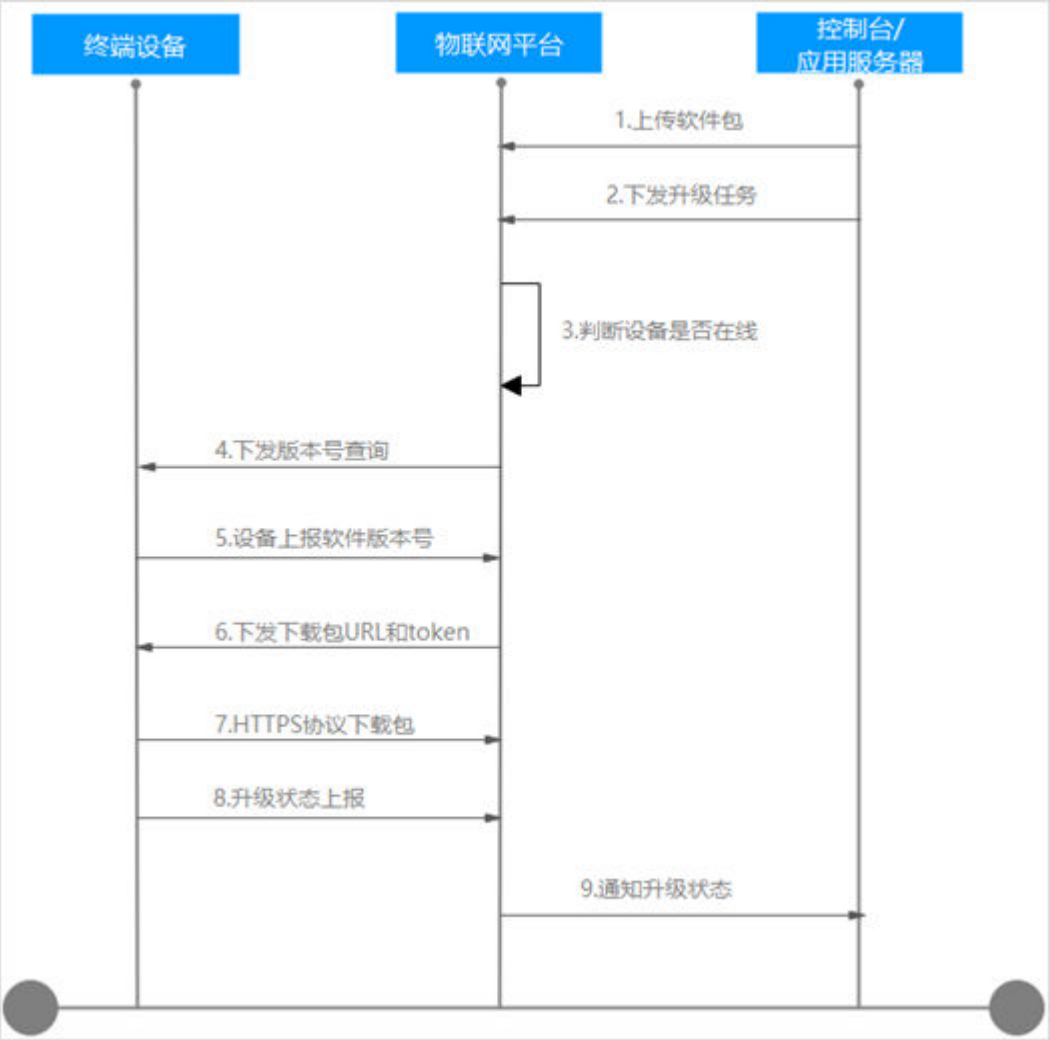
- **目标版本可以比当前版本低吗?**
- **软/固件包及其版本号如何获取?**
- **在软/固件升级任务中，业务处理是否会中断?**
- **常见的软/固件升级错误有哪些?**

相关 API 接口

- **创建批量任务**
- **查询批量任务列表**
- **查询批量任务**

6.7.3 MQTT 设备 OTA 升级

MQTT 协议设备软件升级流程



MQTT协议SOTA升级流程的详细说明：

1～2. 用户在设备管理服务的控制台上传软件包，并在控制台或者应用服务器上创建软件升级任务。

3. 平台感知设备是否在线，当设备在线时立即触发升级协商流程。当设备不在线时，等待设备上线**订阅升级Topic**，平台感知设备上线，触发升级协商流程。（等待设备上线时间25小时以内）

4~5. 平台向设备下发查询设备软件版本号的命令，查询成功后，物联网平台根据升级的目标版本判断设备是否需要升级 。（第5步超时时间3分钟）

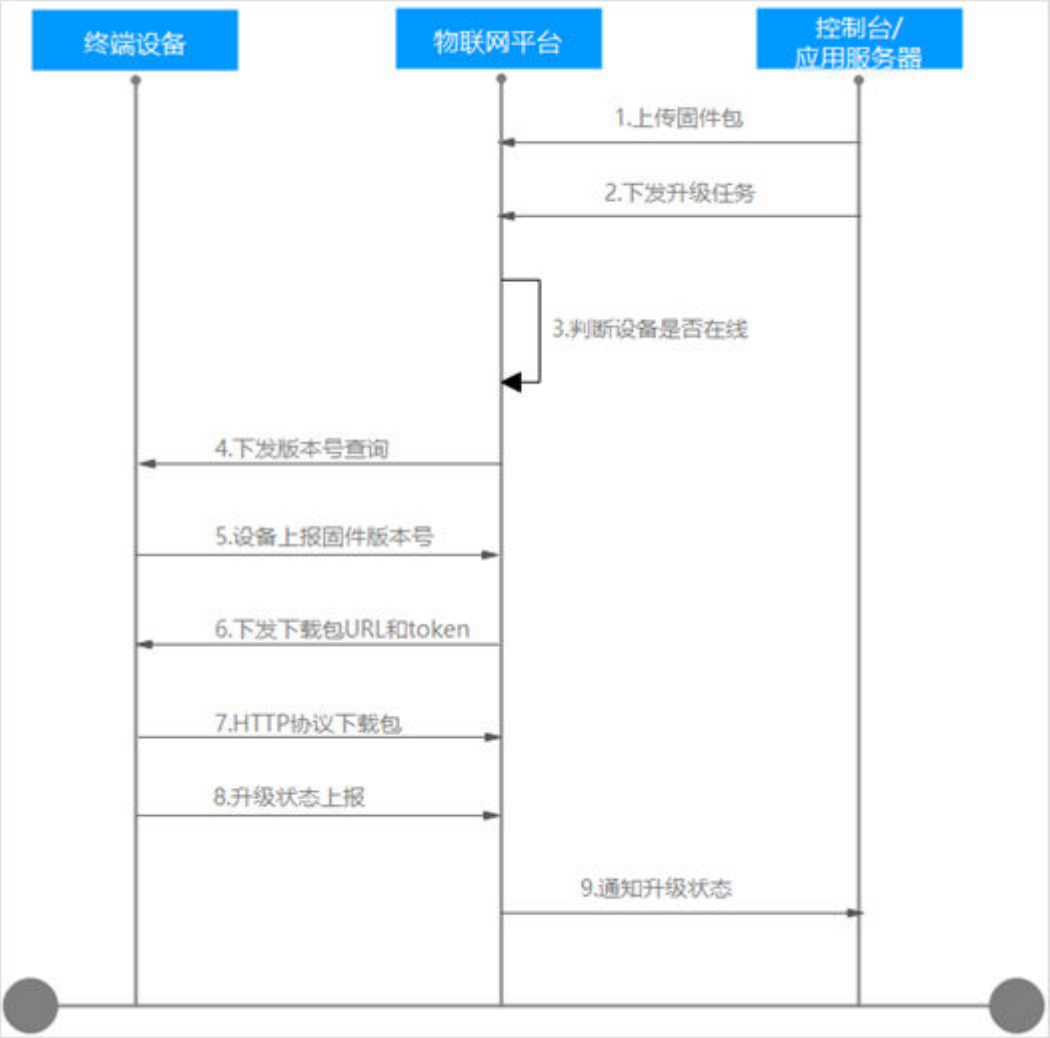
- 如果返回的软件版本信息与升级的目标版本信息相同，则升级流程结束，不做升级处理，升级任务置为成功。
- 如果返回的软件版本信息与升级的目标版本信息不同，且该版本号支持升级，则继续进行下一步的升级处理。

6~7. 物联网平台下发下载包URL访问这里，token及包的相关信息，用户根据下载包URL和token通过HTTPS协议来下载软件包，24小时后token无效。（下载包和升级状态上报超时时间为24小时）

8. 终端设备进行下载包升级操作，升级完成后终端设备向物联网平台反馈升级的结果。(设备升级完成后返回的版本号和设置的版本一致为成功)

9. 物联网平台向控制台/应用服务器通知升级的结果。

MQTT 协议固件升级流程



MQTT协议FOTA升级流程的详细说明：

1～2. 用户在设备接入服务的控制台上传固件包，并在控制台或者应用服务器上创建固件升级任务。

3. 平台感知设备是否在线，当设备在线时立即触发升级协商流程。当设备不在线时，等待设备上线订阅升级Topic，平台感知设备上线，触发升级协商流程。（等待设备上线时间25小时以内）

4~5. 平台向设备下发查询设备固件版本号的命令，查询成功后，物联网平台根据升级的目标版本判断设备是否需要升级 。（第5步超时时间3分钟）

- 如果返回的固件版本信息与升级的目标版本信息相同，则升级流程结束，不做升级处理，升级任务置为成功。
- 如果返回的固件版本信息与升级的目标版本信息不同，且该版本号支持升级，则继续进行下一步的升级处理。

6~7. 物联网平台下发**设备侧升级包下载指导**、token及包的相关信息，用户根据下载包URL和token通过HTTPS协议来下载软件包，24小时后token无效。（下载包和升级状态上报超时时间为24小时）

8. 终端设备进行下载包升级操作，升级完成后终端设备向物联网平台反馈升级的结果。(设备升级完成后返回的版本号和设置的版本一致为成功)

9. 物联网平台向控制台/应用服务器通知升级的结果。

说明

在下载包中断的情况下，平台支持断点续传功能。

固件升级失败原因

物联网平台上报的失败原因：

失败原因	原因解释	处理建议
Device Abnormal is not online	设备异常未在线	请检查设备侧。
Task Conflict	任务冲突	请检查当前设备是否有软件升级、固件升级、日志收集或设备重启的任务正在进行。
Waiting for the device online timeout	等待设备上线超时	请检查设备侧。
Wait for the device to report upgrade result timeout	等待设备上报升级结果超时	请检查设备侧。
Waiting for report device firmware version timeout	等待上报设备固件版本超时	请检查设备侧。
Waiting for report cellId timeout	等待上报cellId超时	请检查设备侧。
Updating timeout and query device version for check timeout	等待升级结果超时，且等待设备版本信息超时	请检查设备侧。
Waiting for device downloaded package timeout	等待设备完成下载固件包超时	请检查设备侧。
Waiting for device start to update timeout	等待设备启动更新超时	请检查设备侧。

失败原因	原因解释	处理建议
Waiting for device start download package timeout	等到设备开始下载固件包超时	请检查设备侧。

设备上报的失败原因：

失败原因	原因解释	处理建议
Not enough storage for the new firmware package	下载的固件包存储空间不足	请检查设备存储。
Out of memory during downloading process	下载过程中内存不足	请检查设备内存。
Connection lost during downloading process	下载过程中连接断开	请检查设备连接状态。
Integrity check failure for new downloaded package	下载的固件包完整性校验失败	请检查设备下载的固件包是否完整。
Unsupported package type	固件包类型不支持	请检查设备状态和厂商提供的固件包是否正确。
Invalid URI	URI不可用	检查设备侧的固件包下载地址是否正确。
Firmware update failed	固件更新失败	请检查设备侧。

常见问题

软/固件升级业务热点咨询问题如下，更多咨询问题请访问**查看更多**。

- **目标版本可以比当前版本低吗?**
- **软/固件包及其版本号如何获取?**
- **在软/固件升级任务中，业务处理是否会中断?**
- **常见的软/固件升级错误有哪些?**

相关 API 接口

- **创建批量任务**
- **查询批量任务列表**
- **查询批量任务**

6.7.4 批量设备 OTA 升级

上传软固件包

创建批量设备软件、固件升级任务前需要上传软件升级包，平台支持两种方式上传软件、固件包：

1. 应用服务器通过调用“创建OTA升级包”API接口，创建OTA升级包，详情请参考**创建OTA升级包**。
2. 通过控制台，在软固件升级页面上传软件、固件升级包，详情请参考**软固件包上传**。

说明

- 通过API接口创建的OTA升级包，只支持MQTT协议设备升级。
- 升级包为OBS对象时，无论OBS桶是否配置了CDN域名加速功能，下发的升级包链接都为OBS链接地址。

批量设备软件升级

用户对批量设备进行软件升级有两种方式：

1. 应用服务器通过调用的“创建软件升级任务”API接口，创建批量设备的升级任务，详情请参考**创建批量任务**。
2. 通过控制台，创建批量设备的软件升级任务。

下面将重点介绍通过控制台创建批量设备的软件升级任务。

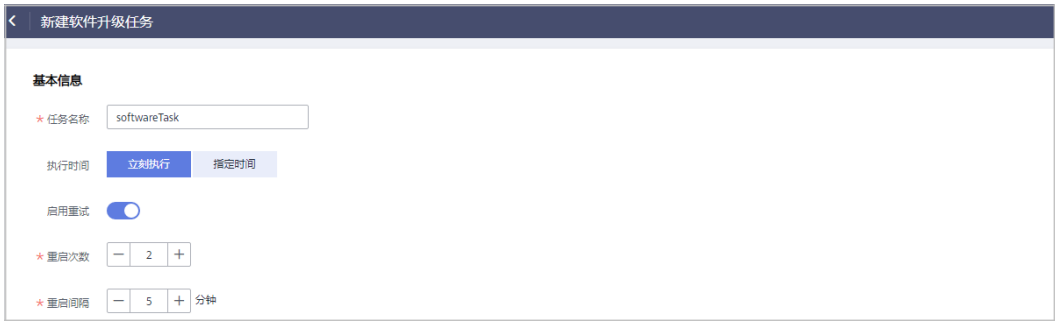
步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择"设备 > 软固件升级"，单击“升级任务”。

步骤3 选择“软件升级”页签，单击“新建任务”按钮，进入新建软件升级任务页面。

步骤4 设置“任务信息”，填写任务名称、执行时间、启用重试。

启用重试后，可以设置重启次数和重启间隔。重启次数建议设置为2次，重启间隔设置为5分钟，即设备升级失败后，隔5分钟后会进行升级重试。



步骤5 选择需要升级的软件包。

步骤6 选择需要升级的设备或者设备群组，然后单击“立即创建任务”。

设备群组可以参考**群组与标签**创建需要升级的设备群组，并绑定对应的设备。

步骤7 创建完批量升级任务后，可以在软件升级任务列表中查看批量任务的执行结果。单击对应任务"查看"按钮，可以在“执行详情”界面查看每个设备的升级结果。

说明

如果升级任务正在执行中，是不允许删除任务的，如需删除，请先在任务列表中，手动停止任务后，再删除升级任务。

----结束

批量设备固件升级

用户对批量设备进行固件升级有两种方式：

1. 应用服务器通过调用的“创建固件升级任务”API接口，创建批量设备的升级任务，详情请参考**创建批量任务**。
2. 通过控制台，创建批量设备的固件升级任务。

下面将重点介绍通过控制台创建批量设备的固件升级任务。

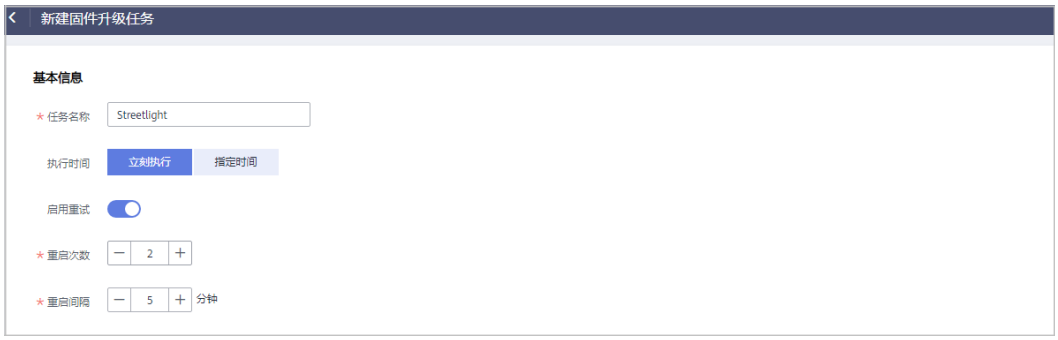
步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择"设备 > 软固件升级"，单击“升级任务”。

步骤3 在“固件升级”页签，单击“新建任务”按钮，进入新建固件任务页面。

步骤4 设置“任务信息”，填写任务名称、执行时间、启用重试。

启用重试后，可以设置重启次数和重启间隔。重启次数建议设置为2次，重启间隔设置为5分钟（最大重启次数为5次，最大重启间隔为1440分钟），即设备升级失败后，隔5分钟后会进行升级重试。



步骤5 选择需要升级的固件包。

步骤6 选择需要升级的设备群组，然后单击“提交”。

设备群组可以参考**群组与标签**创建需要升级的设备群组，并绑定对应的设备。

步骤7 创建完批量升级任务后，可以在固件升级任务列表中查看批量任务的执行结果。单击对应任务"查看"按钮，可以在“执行详情”界面查看每个设备的升级结果。

说明

如果升级任务正在执行中，是不允许删除任务的，如需删除，请先在任务列表中，手动停止任务后，再删除升级任务。

----结束

固件升级失败原因

物联网平台上报的失败原因：

失败原因	原因解释	处理建议
Device Abnormal is not online	设备异常未在线	请检查设备侧。
Task Conflict	任务冲突	请检查当前设备是否有软件升级、固件升级、日志收集或设备重启的任务正在进行。

失败原因	原因解释	处理建议
Waiting for the device online timeout	等待设备上线超时	请检查设备侧。
Wait for the device to report upgrade result timeout	等待设备上报升级结果超时	请检查设备侧。
Waiting for report device firmware version timeout	等待上报设备固件版本超时	请检查设备侧。
Waiting for report cellId timeout	等待上报cellId超时	请检查设备侧。
Updating timeout and query device version for check timeout	等待升级结果超时，且等待设备版本信息超时	请检查设备侧。
Waiting for device downloaded package timeout	等待设备完成下载固件包超时	请检查设备侧。
Waiting for device start to update timeout	等待设备启动更新超时	请检查设备侧。
Waiting for device start download package timeout	等到设备开始下载固件包超时	请检查设备侧。

设备上报的失败原因：

失败原因	原因解释	处理建议
Not enough storage for the new firmware package	下载的固件包存储空间不足	请检查设备存储。
Out of memory during downloading process	下载过程中内存不足	请检查设备内存。
Connection lost during downloading process	下载过程中连接断开	请检查设备连接状态。
Integrity check failure for new downloaded package	下载的固件包完整性校验失败	请检查设备下载的固件包是否完整。
Unsupported package type	固件包类型不支持	请检查设备状态和厂商提供的固件包是否正确。
Invalid URI	URI不可用	检查设备侧的固件包下载地址是否正确。

失败原因	原因解释	处理建议
Firmware update failed	固件更新失败	请检查设备侧。

常见问题

软/固件升级业务热点咨询问题如下，更多咨询问题请访问**查看更多**。

- **目标版本可以比当前版本低吗?**
- **软/固件包及其版本号如何获取?**
- **在软/固件升级任务中，业务处理是否会中断?**
- **常见的软/固件升级错误有哪些?**

相关 API 接口

- **创建批量任务**
- **查询批量任务列表**
- **查询批量任务**

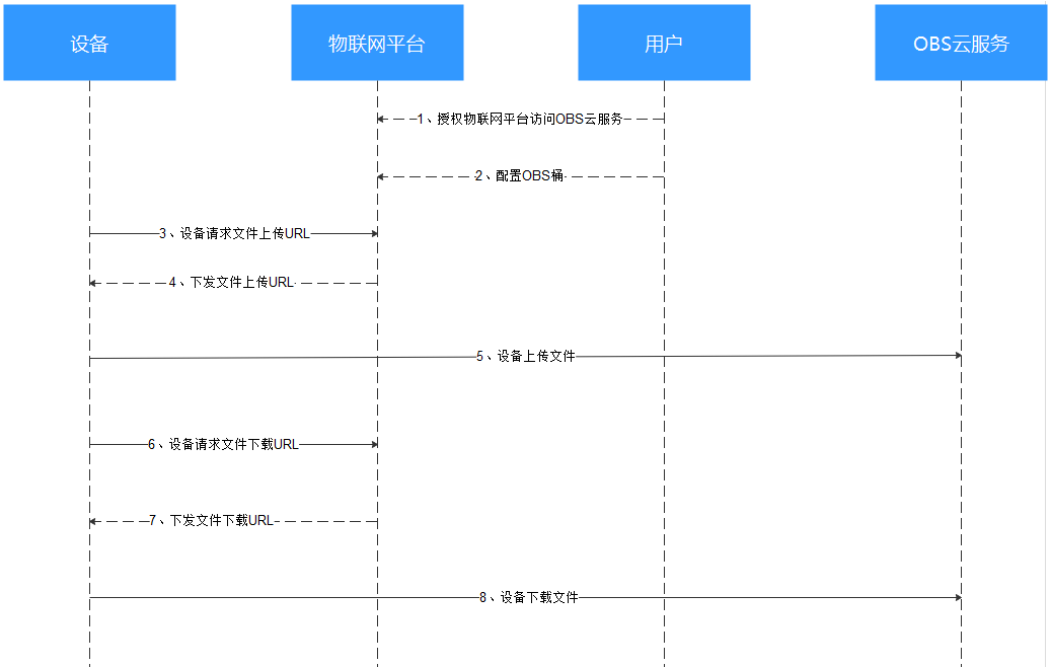
6.8 文件上传

概述

华为物联网平台支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。当设备采用HTTPS方式将文件上传到OBS服务进行备份时，您可以在OBS服务管理已上传的设备文件。

业务流程

图 6-35 文件上传流程图



1. 授权物联网平台访问OBS服务。

2. 配置OBS桶。

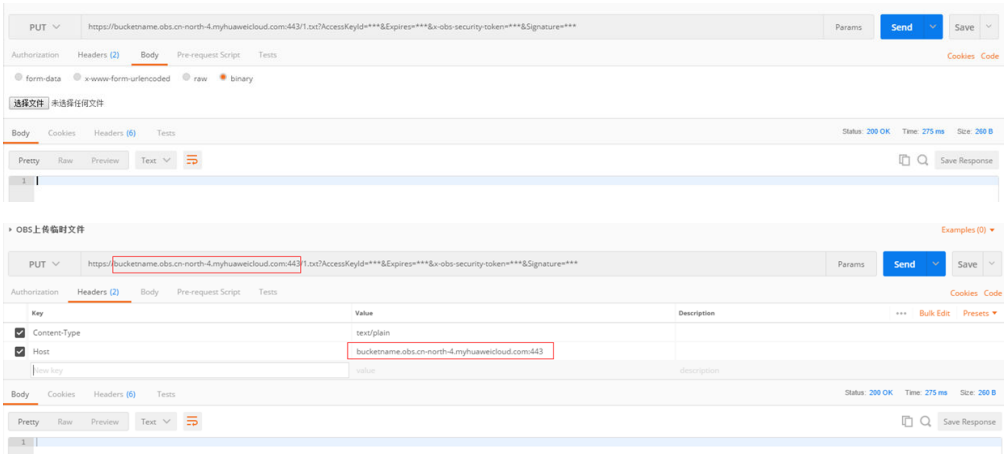
3~4. 设备请求上传文件URL，平台下发文件上传URL，格式请参考**设备上报获取文件上传URL请求**接口文档。

5.调用OBS接口，使用平台下发的URL上传设备文件，URL有效期时间以下发的expire为准，单位为秒，默认是1个小时。

- 方法一：直接使用该URL，以Postman为例。

  使用PUT方法调用URL，body选择binary，选择具体的文件上传，文件名与上报的文件名一样，这样在OBS才可以看到对应的文件。

  调用接口的header可以不带Content-Type或者Host。如果必须要带，请检查Content-Type是否为text/plain，以及Host是否为URL的域名，否则调用接口将返回403状态码SignatureDoesNotMatch错误。



- 方法二：集成OBS的SDK调用接口。

  参考**使用URL进行授权访问**，使用put请求上传对象SDK来上传对象。

6~7. 设备请求下载存储在OBS服务的文件，平台下发文件下载URL，格式请参考**平台下发文件上传临时URL**接口文档。

8. 调用OBS接口，使用平台下发的URL下载设备文件。

- 方法一：使用GET方法调用URL。调用接口的header可以不带Content-Type或者Host。如果必须要带，请检查Content-Type是否为text/plain，以及Host是否为URL的域名，否则调用接口将返回403状态码SignatureDoesNotMatch错误。
- 方法二：集成OBS的SDK调用接口，使用GET请求下载对象SDK来下载文件对象。

配置文件上传功能

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏，选择“设备 > 所有设备”，单击“文件上传”页签。

步骤3 单击“服务授权”，在新弹出的页面中单击“同意授权”。

注：如用户仅授权过设备接入服务访问对象存储服务的权限，可在控制台选择“设备 > 所有设备”，单击“文件上传”页签，然后单击“KMS服务授权”按钮。授权设备接入服务访问密钥管理服务的权限。

步骤4 （可选）若没有桶，请先在OBS服务控制台创建桶，已有OBS桶的话，请跳过此步骤。

1. 访问OBS控制台。
2. 单击右上角的“创建桶”，进入参数配置页面，更新参数说明请参考**创建桶**。

说明

使用OBS云服务管理文件，将由OBS服务进行收费，设备接入服务不再单独对文件存储进行收费。更多OBS文件存储计费详情，请参考**计费说明**。

步骤5 单击“OBS存储配置”，选择正确桶后，该实例下的所有设备文件将上传到配置的OBS桶，您也可以单击“修改配置”更改桶。

图 6-36 OBS 存储配置

说明

调用OBS接口上传设备文件时，每次只能上传一个文件，且文件大小不能超过5GB。

步骤6 如果需要使用自定义域名， 请打开开关“使用自定义域名”，选择OBS桶配置的自定义域名，并选择使用访问方式HTTPS/HTTP，单击“确定”完成配置。

图 6-37 使用自定义域名

说明

使用自定义域名后，平台下发给设备上传或下载的临时URL域名为自定义域名。设备可以通过该URL上传或下载OBS文件。

----结束

6.9 网关与子设备

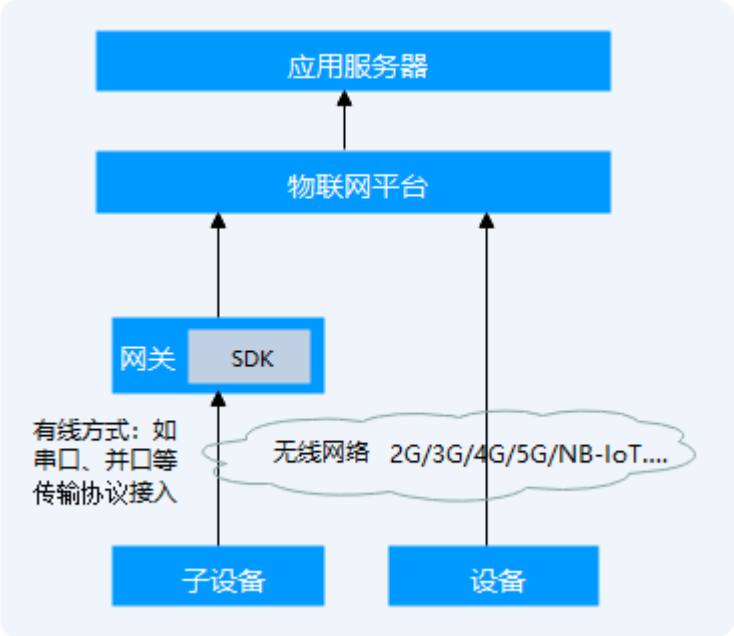
概述

物联网平台设备分为直连设备和非直连设备。

- 直连设备：通过平台支持的协议，直接连接到平台的设备称为直连设备。
- 非直连设备：针对未实现TCP/IP协议栈的设备，由于无法直接同物联网平台通信，它需要通过网关进行数据转发。网关设备为直连设备，当前仅支持通过mqtt协议直连到平台的设备作为网关设备。

直连设备与非直连设备关系如下图：

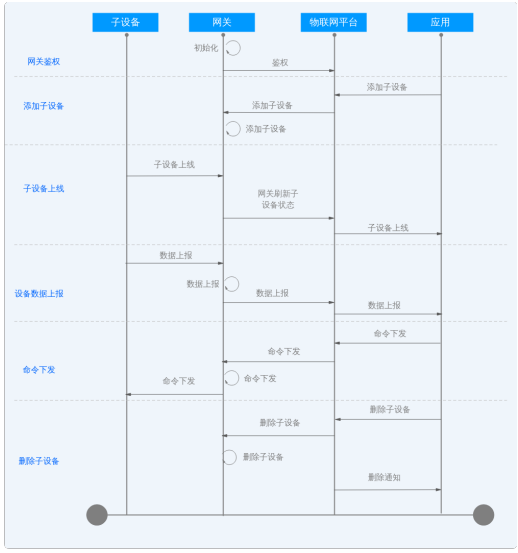
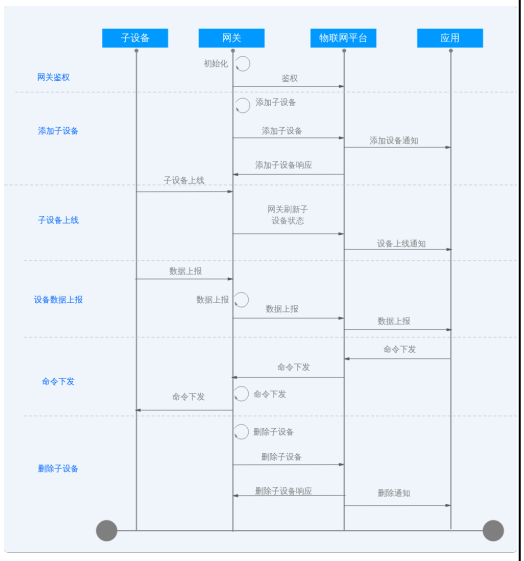
图 6-38 网关与子设备



业务流程

IoT Device SDK提供相关的接口，开发者调用这些接口网关实现与物联网平台的对接。不同语言的SDK的接口名称不一样，每个接口功能请参见**IoT Device SDK使用指南（Java）**、**IoT Device SDK使用指南（C）**、**IoT Device SDK使用指南（C#）**、**IoT Device SDK使用指南（Android）**、**IoT Device SDK Tiny使用指南（C）**。

表 6-8 业务流程

应用侧管理子设备业务流程	网关管理子设备业务流程
图 6-39 应用侧管理子设备流程图 	**图 6-40** 网关侧管理子设备流程图 

应用侧管理子设备业务流程	网关管理子设备业务流程
1.在物联网平台上传网关的产品模型，并注册网关设备。	
2.网关调用鉴权接口上线	
3.在物联网平台上传子设备的产品模型。	
4.网关鉴权成功后，应用调用**添加子设备**接口，填写设备相关信息（与产品模型定义一致）。添加成功后，您可以在物联网平台查看添加的子设备（**如何查看？**）。您也可以通过控制台添加子设备，详细请查看**如何添加?**	4.网关鉴权成功后，网关调用**平台通知网关子设备新增**接口，填写设备相关信息（与产品模型定义一致）。平台在处理完成后，将处理结果通过接口**网关新增子设备请求响应**发送给网关。
5.添加子设备后，子设备状态显示“未激活”。请在子设备添加成功后，或者子设备上报数据前，调用**网关更新子设备状态**接口进行设备状态更新。 **说明** 子设备的状态表示子设备接入网关的状态，由网关上报到物联网平台进行状态的刷新；如果网关不能正常上报子设备的状态信息到物联网平台，则展示的子设备状态不会刷新。例如：某子设备通过网关接入到物联网平台，子设备状态为在线状态，如果此时网关与物联网平台断开连接，则网关不能上报子设备的状态到物联网平台，该子设备的状态会一直显示在线。	
6.网关调用**批量属性上报**接口上报子设备的数据，接口里的参数填写网关和子设备的相关设备信息。	
7.网关订阅命令下发Topic，接收并处理应用服务器或物联网平台下发的命令。	
8.应用服务器调用**删除设备**接口，给网关下发删除子设备命令，网关收到该命令后，可以进行相应的业务处理。	8.网关调用**网关删除子设备请求**接口，平台收到后会进行数据处理，当处理完成后会通过接口**网关删除子设备请求响应**将结果发送给设备。

网关如何接入物联网平台?

通过在网关上集成SDK, 设备将数据上报给网关，通过网关转发到华为物联网平台，网关接入物联网平台的开发指南请参见**设备通过网关接入平台**。

物联网平台上如何添加子设备?

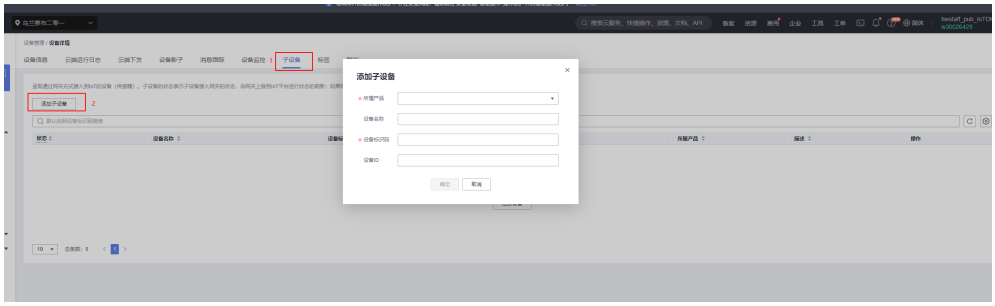
- **方式1**

  当网关接入物联网平台后，调用**创建设备**接口，完成子设备接入到物联网平台。

- **方式2**

  访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择“设备 > 所有设备”，在设备列表中，单击具体的网关设备进入到网关的详情页面，进入“子设备”页签，单击“添加子设备”。

图 6-41 设备-添加子设备



查看子设备

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备 > 所有设备”页面，在设备列表中，单击具体的网关进入到网关的详情页面，选择“子设备”页签。

步骤3 “子设备”页签呈现通过该网关方式接入到物联网平台的设备，可以查看子设备的状态、设备ID、设备标识码等信息。

步骤4 在子设备页签中，单击具体的子设备，可以查看子设备的**设备详情**。

图 6-42 设备-子设备详情

----结束

6.10 设备认证凭证管理

认证凭证介绍

当设备接入物联网平台时，设备需要携带认证凭证在平台鉴权。当前平台支持的认证凭证有：

- 密钥认证：注册设备时在物联网平台提前预置的设备密钥，平台会使用在注册设备时预置的密钥对设备进行认证。认证通过后，设备完成激活，然后与平台进行通信。在平台预置的密钥分为：
  - 主密钥：设备密钥鉴权优先使用的密钥，当设备接入物联网平台时，平台将优先使用主密钥进行校验。
  - 辅密钥：设备的备用密钥，当主密钥校验不通过时，会启用辅密钥校验，辅密钥与主密钥有相同的效力；辅密钥对coap协议接入的设备不生效。
- X.509证书认证：X.509是一种用于通信实体鉴别的数字证书，物联网平台支持设备使用自己的X.509证书进行认证鉴权。详情请参考**基于MQTT.fx的X.509证书接入指导**。使用证书认证时，平台将校验设备证书的指纹，用户可在平台预置设备的证书指纹用于设备建链时的校验，平台预置的指纹分为：

- 主指纹：设备证书鉴权优先使用的指纹，当设备接入物联网平台时，平台将优先使用主指纹进行校验。
- 辅指纹：设备的备用指纹，当主指纹校验不通过时，会启用辅指纹校验，辅指纹与主指纹有相同的效力。

认证凭证更新

设备的接入凭证可能在特定场景需要更新，如设备的X.509证书即将过期，需要将设备证书进行更新。平台提供了密钥/指纹的重置功能，您可以调用接口**重置设备指纹**/**重置设备密钥**进行设备凭证的重置。为避免认证凭据切换过程中，用户未在平台及时重置认证凭据导致设备鉴权失败，出现业务中断，平台推出支持双指纹/双密钥的功能。如在证书认证的场景下，将新证书的指纹设置到设备的备用指纹中，此时如果设备没及时更新证书，仍然可以通过旧的证书接入平台，当设备更新证书后，无需在应用侧进行重置操作，即可立即使用新证书接入平台，实现无损切换认证凭证。

使用场景

1.在高可用场景下，设备持有两个密钥，设备可使用任意一个密钥接入平台。

2.在认证凭证切换的场景下，无损切换设备认证凭证，设备切换认证凭证前后不会出现无法接入平台导致出现短暂呼损。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台

步骤2 在设备接入控制台左侧导航栏，选择“设备 > 所有设备”，进入设备列表页，设备列表默认显示当前实例下的所有设备。

步骤3 单击需要进行操作的设备“详情”按钮，单击“重置密钥/重置指纹”，单击“辅密钥/辅指纹”。

图 6-43 重置设备辅密钥

图 6-44 重置设备辅指纹

----结束

相关 API 接口

- **重置设备指纹**
- **重置设备密钥**

6.11 设备证书

概述

设备证书是指设备使用MQTTS协议X.509证书双向认证时设备侧的证书，设备第一次连接物联网平台时，物联网平台使用用户上传已验证的设备CA证书对设备证书进行认证，认证通过后物联网平台会自动保存设备证书。物联网平台提供了对设备证书查看、停用、过期预警等功能。

约束与限制

1. 物联网平台会对30天内即将过期的设备证书进行告警，请及时更新证书防止接入失败。
2. 物联网平台提供的设备证书配额为设备数配额的1.5倍，请及时清理过期证书，防止新证书存储失败，导致无法在界面上查看该证书，但不影响设备接入平台。
3. 设备与设备证书通过证书指纹关联，停用设备证书后其关联的所有设备将无法接入平台。

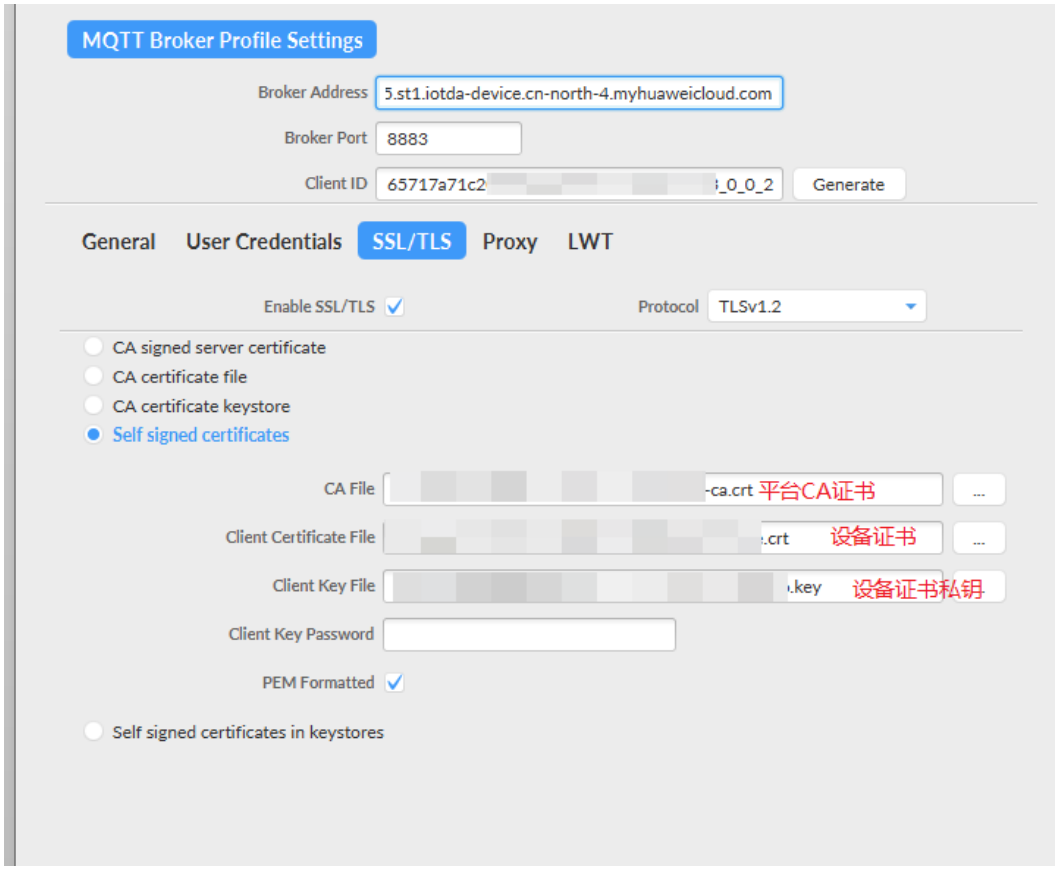
操作步骤

步骤1 上传并验证设备CA证书，设备CA证书制作与验证可参考**注册X.509证书认证的设备**。

步骤2 注册X.509证书认证类型的设备。进入设备接入控制台，左侧导航栏选择“设备 > 所有设备”，单击“注册设备”。认证类型选择X.509证书，指纹填写设备证书SHA256指纹，若不填写则默认记录设备第一次成功接入平台所携带的设备证书指纹。

步骤3 使用设备证书接入平台。

图 6-45 设备连接参数



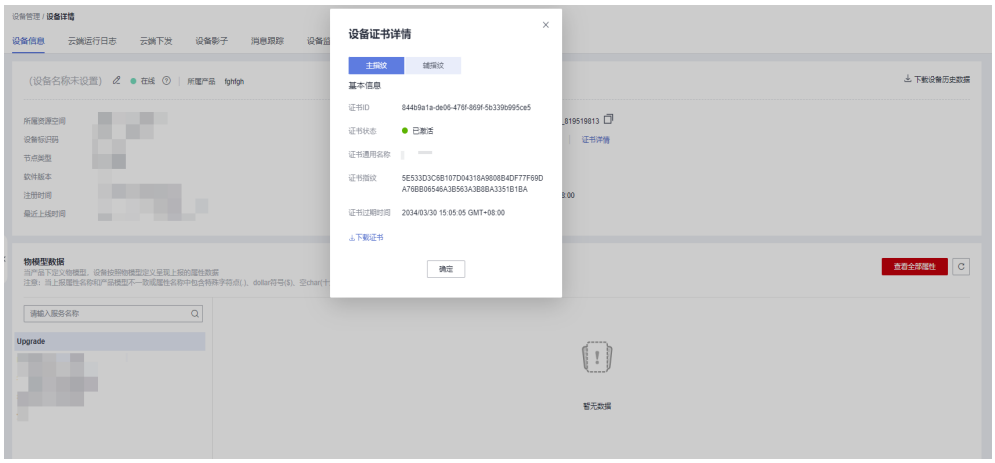
步骤4 进入设备接入控制台，左侧导航栏选择“设备 > 设备证书”，选择“设备证书”查看设备证书列表与详情。

图 6-46 设备证书-证书列表

图 6-47 设备证书-证书详情

步骤5 进入设备接入控制台，左侧导航栏选择“设备 > 所有设备”，选择“详情”进入设备详情页面，单击“证书详情”查看。

图 6-48 设备-设备详情-证书详情



----结束

设备证书告警

- 物联网平台会对设备证书进行过期预警，用户可以在AOM服务告警列表中查看近期一个月内即将过期的证书。

图 6-49 设备证书过期告警-AOM

- 物联网平台会对设备证书数量超过阈值进行预警，请及时清理已过期证书。

图 6-50 设备证书配额不足告警-AOM

AOM服务提供“创建告警行动规则”来发送告警通知，支持通知到短信，邮箱，企业微信等，具体操作可参考：**创建告警行动规则**。

7 规则引擎

7.1 规则引擎介绍

规则引擎是指用户可以在物联网平台上对接入平台的设备设定相应的规则，在条件满足所设定的规则后，平台会触发相应的动作来满足用户需求。包含设备联动和数据转发两种类型。

- **设备联动**

  设备联动指通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前物联网平台支持两种联动规则：云端规则和端侧规则。例如，当用户选择云端规则，执行动作为“发送通知”时，物联网平台对接华为云的**消息通知服务SMN**，进行主题消息的设置和下发。当用户选择端侧规则时，云平台会将规则下发到设备侧，由端侧设备对平台下发的规则进行统一的管理和执行。

- **数据转发**

  数据转发无缝与华为云其他服务、第三方应用对接，实现设备数据的存储、计算、分析的全栈服务。

云服务访问授权

物联网平台支持与华为云其它云服务进行对接。首次创建对接到DIS服务、OBS服务、Kafka服务、ROMA Connect服务、以及SMN服务的规则时，您需要进行云服务访问授权操作。

授权成功后可以通过数据转发功能将物联网平台的数据转发到您购买的华为云其他服务，或通过设备联动规则，下发命令控制设备。

同时设备接入服务将在**统一身份认证服务（IAM）**为您创建名为iotda_admin_trust 的委托，并默认绑定管理员角色。

7.2 数据转发流程

概述

数据转发功能用于提供IoTDA与其他第三方以及华为云服务的连接通道，从而实现将设备数据平滑流转至消息中间件、存储、数据分析、业务应用。当前物联网平台支持如下转发方式：

表 7-1 数据转发概述

分类	转发目标	说明	操作指导
第三方服务	第三方应用服务（HTTP推送）	将数据转发至客户的HTTP服务器，客户可以在数据转发界面创建流转规则，并指定推送的URL，将订阅的数据源信息推送到指定URL的服务器。	**使用HTTP/HTTPS转发**
	AMQP推送消息队列	客户可以通过数据转发界面，订阅指定的AMQP通道，将订阅的数据源信息推送到指定AMQP通道，用户可通过AMQP的客户端与IoT平台建立链接，接收数据。	**使用AMQP转发**
	MQTT推送消息队列	客户可以通过数据转发界面，订阅指定的MQTT Topic，将订阅的数据源信息推送到指定MQTT Topic，用户可通过Mqtt的客户端与IoT平台建立链接，接收数据。	**使用MQTT转发**
	设备间通信	物联网平台支持基于MQTT协议实现设备间的消息通信，客户可以通过数据转发界面，订阅指定的Topic，平台会将设备上报的消息推送到指定的Topic，其他设备可以通过订阅该Topic来接收不同设备的消息。	**设备间通信**
数据存储	**云数据库GeminiDB Influx**	将数据流转到华为云GeminiDB Influx，兼容InfluxDB生态的云原生时序数据库。提供高性能读写、高压缩率、冷热分层存储以及弹性扩容、监控告警等服务能力，可以实现大并发的时序数据读写，压缩存储和类SQL查询等功能，支持多维聚合计算和数据可视化分析能力。 应用场景：广泛应用于资源监控、业务监控分析、物联网设备实时监控、工业生产监控、生产质量评估和故障回溯等。提供了高吞吐量和并发性，可以通过快速的响应时间来支持大量的连接，非常适合要求苛刻的物联网应用。 规格参考：**InfluxDB规格**。	**数据转发至GeminiDB Influx**

分类	转发目标	说明	操作指导
	云数据库 RDS for MySQL	将数据流转到华为云RDS MySQL，相比自建数据库，RDS价格便宜、 即开即用，便捷运维，支持弹性伸缩并具备实例管理、实例监控、备份恢复、日志管理、参数管理等功能，支持单机和主备部署。 应用场景：网站业务、移动应用、游戏业务、电商业务、金融业务以及企业应用等。 规格参考：Mysql性能规格。	数据转发至MySQL
	对象存储服务 OBS	将数据流转至华为云OBS云服务，OBS为客户提供海量、安全、高可靠、低成本的数据存储能力，使用时无需考虑容量限制，并且提供多种存储类型供选择，满足客户各类业务场景诉求，OBS也支持对接实时计算CS云服务，实时分析数据流，分析结果对接到其他云服务或者第三方应用进行数据可视化等。 应用场景：适用于海量大数据存储分析的场景。 规格参考：OBS存储规格。	数据转发至OBS长期存储

须知

通过公网进行数据转发流量限制不超过1M/s，超过后消息会直接丢弃。

操作步骤

步骤1 访问设备接入服务，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则 > 数据转发”，单击页面左上角的“创建规则”。

步骤3 设置转发数据，然后单击“创建规则”。

表 7-2 创建规则参数列表

参数名	参数说明
规则名称	创建的规则名称。
规则描述	对该规则的描述。

参数名	参数说明
数据来源	• 设备：将操作设备的信息，如设备添加、设备删除、设备更新设置为数据来源。当数据来源选择“设备”时，不支持快速配置。 • 设备属性：将归属在某个资源空间下的设备上报给平台的属性值设置为数据来源。单击右侧的“快速配置”勾选需要转发的产品、属性、服务等数据。 • 设备消息：将归属在某个资源空间下的设备上报给平台的消息设置为转发目标。单击右侧的“快速配置”，仅转发指定Topic的数据。选择所属产品，填写Topic名称。您可以使用在产品详情页面**自定义的Topic**，也可以使用**平台预置的Topic**。 • 设备消息状态：将设备和平台之间流转的设备消息状态变更设置为转发目标。设备消息状态详见**这里**。当数据来源选择“设备消息状态”，不支持快速配置。 • 设备状态：将归属在某个资源空间下的直连或非直连设备状态变更转发至其他服务。单击“快速配置”，您可以转发设备状态为“在线”、“离线”和“异常”的设备信息到其他服务。物联网平台直连设备状态详见**这里**。 • 批量任务：将批量任务状态的数据设置为数据来源。当数据来源选择“批量任务”时，不支持快速配置。 • 产品：将操作产品的信息，如产品添加、产品删除、产品更新设置为数据来源。当数据来源选择“产品”时，不支持快速配置。 • 设备异步命令状态：针对LwM2M/CoAP协议的设备，物联网平台支持下发异步命令给设备。将异步命令的状态变更设置为数据来源。物联网平台设备异步命令状态详见**这里**。当数据来源选择“设备异步命令状态”时，不支持快速配置。 • 运行日志：将MQTT设备的业务运行日志设置为数据来源。当数据来源选择“运行日志”时，不支持快速配置。
触发事件	选择数据来源后，对应修改触发事件。
资源空间	您可以选择单个资源空间或所有资源空间。当选择“所有资源空间”时，不支持快速配置。

步骤4 单击“设置转发目标”页签，单击“添加”，设置转发目标。

根据实际情况，选择**数据接入服务DIS**、**分布式消息服务Kafka**、**对象存储服务OBS**、**第三方应用服务（HTTP推送）**、**AMQP推送消息队列**、**函数工作流服务（FunctionGraph）**、**云日志服务（LTS）**、**云数据库 GeminiDB Influx**、**云数据库MySQL**，，第三方应用服务（HTTP推送），AMQP推送消息队列，MQTT推送消息队列，设备。

表 7-3 设置转发目标参数列表

转发目标	参数说明
数据接入服务（DIS）	• 区域：选择转发服务的所在区域，若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。 • 通道归属：可以选择自有通道或他人授权。 – 自有通道：选择需要的通道。若没有通道，请前往DIS服务**创建通道**。 – 他人授权：您也可以使用其他用户授权给您的通道。请前往DIS控制台获取通道ID。
分布式消息服务（Kafka） **说明** 如果选择分布式消息服务，目前只支持转发到Kafka专享版。并且需要开启“kafka自动创建Topic”功能。	• 区域：选择转发服务的所在区域，若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。 • 对接地址：参考**连接已开启SASL的Kafka实例**获取对接地址。设备接入基础版和标准版实例只支持公网接入专享版Kafka，企业版实例支持私网接入专享版Kafka。 • 主题：自定义。 • SASL认证：若开启SASL认认证，则用户名和密码，填写您在**购买kafka实例**中输入的SASL用户名和密码。 • Kafka安全协议：开启SASL认证后，Kafka安全协议选择您购买的Kafka实例中支持的安全协议。 • SASL认证机制：若开启SASL认证后，则SASL认证机制选择您购买的Kafka实例中支持的SASL认证机制。
对象存储服务（OBS）	• 区域：选择转发服务的所在区域，若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。 • 存储桶：选择需要的桶。若没有，请前往OBS服务**创建桶**。 • 参数说明：OBS服务中存储通道文件的自定义目录,多级目录可用(/)进行分隔，不可以斜杠(/)开头或结尾，不能包含两个以上相邻的斜杠(/)。
第三方应用服务（HTTP推送）	您可以使用HTTP或HTTPS协议进行推送。详细参数填写请参考**HTTP/HTTPS服务端订阅**。
AMQP推送消息队列	消息队列：选择需要推送消息的队列。若没有队列，请创建队列。消息队列名限制请参考**配置AMQP服务端**。

转发目标	参数说明
函数工作流服务（FunctionGraph） **说明** 目前仅企业版实例和标准版实例支持转发到函数工作流服务，基础版实例不支持。	• 函数名称：选择需要调用的函数名称（latest版本）。当前不支持跨区域调用函数。若没有函数，请参考**这里**创建函数。
云数据库 GeminiDB Influx **说明** 目前仅企业版实例和标准版实例支持转发到GeminiDB Influx，基础版实例不支持。	• 数据库实例地址：输入GeminiDB Influx实例的连接地址。设备接入企业版实例支持通过内网IP连接GeminiDB Influx，标准版只支持公网连接GeminiDB Influx。详细指导请参考**这里**。 • 数据库名称：选择需要的数据库。若没有，请前往GeminiDB Influx服务**创建数据库**。 • 访问账户/访问密码：访问InfluxDB实例的控制台获取GeminiDB Influx的账户和密码。详细请参考**这里**。 • 表格：输入需要转存的表格（measurement）名称，不存在会自动创建。 • 转存配置： – 转发字段：输入流转数据的属性名。流转数据为json格式，多层级属性名使用“.”分隔，流转数据格式请参考**这里**。 – 目标存储字段：输入数据库的列名。
云数据库 MySQL（RDS） **说明** 目前仅企业版实例和标准版实例支持转发到云数据库 MySQL（RDS），基础版实例不支持。	• 数据库实例地址：输入RDS实例的连接地址。设备接入企业版实例支持通过内网IP连接RDS，标准版只支持公网连接RDS。详细指导请参考**这里**。 • 数据库名称：选择需要的数据库。若没有，请前往RDS服务**创建数据库**。 • 访问账户/访问密码：访问RDS实例的控制台获取RDS的账户和密码。详细请参考**这里**。 • SSL：选择是否通过SSL加密方式连接数据库。建议通过SSL方式连接，不通过SSL方式连接可能存在数据传输安全风险。选择通过SSL方式连接时，需要先在数据库实例中**设置SSL数据加密**。 • 表格：选择需要转存的表格名称。 • 转存配置： – 转发字段：输入流转数据的属性名。流转数据为json格式，多层级属性名使用“.”分隔，流转数据格式请参考**这里**。 – 目标存储字段：选择数据库的列名。
MQTT推送消息队列	推送Topic：选择需要推送消息的Topic。
设备	通过MQTT协议实现设备间消息通信，具体参数请参考：**设备间消息通信使用说明。**

步骤5 启动规则。

完成完整的规则定义后，您可以在“启动规则”页签中，单击右侧开关，启动规则，实现数据转发。

图 7-1 数据转发-启动规则

步骤6 物联网平台提供规则动作转发目标连通性测试功能，具体步骤请参考**连通性测试**。

----结束

7.3 SQL 语句

创建数据转发规则时，需要编写SQL来解析和处理设备上报的JSON数据，JSON数据具体格式参考**流转数据**。本文主要介绍如何编写数据转发规则的SQL表达式。

SQL 语句

SQL语句由SELECT子句和WHERE子句组成，每个子句不能大于500个字符，暂不支持中文等其他字符集。SELECT子句和WHERE子句里的内容大小写敏感，SELECT和WHERE，AS等关键字大小写不敏感。

以设备消息上报为SQL源数据示例:

```
{
  "resource" : "device.message",
  "event" : "report",
  "event_time" : "20151212T121212Z",
  "notify_data" : {
    "header" : {
      "device_id" : "d4922d8a-6c8e-4396-852c-164aefa6638f",
      "product_id" : "ABC123456789",
      "app_id" : "d4922d8a-6c8e-4396-852c-164aefa6638f",
      "gateway_id" : "d4922d8a-6c8e-4396-852c-164aefa6638f",
      "node_id" : "ABC123456789",
      "tags" : [ {
        "tag_value" : "testTagValue",
        "tag_key" : "testTagName"
      } ]
    },
    "body" : {
```

```
    "topic" : "topic",
    "content" : {
       "temperature" : 40,
       "humidity" : 24
    }
  }
 }
}
```

在源数据中，body中的content是设备消息上报的数据，设置当设备上报数据中temperature大于38时触发条件，并筛选出device_id、content，不需要任何其他字段时，SQL语句示例如下：

```
SELECT notify_data.header.device_id AS device_id, notify_data.body.content WHERE
notify_data.body.content.temperature > 38
```

当设备上报消息中temperature大于38度时，会触发转发，转发后的数据格式如下：

```
{
   "device_id": "d4922d8a-6c8e-4396-852c-164aefa6638f",
   "notify_data.body.content" : {
        "temperature" : 40,
        "humidity" : 24
    }
}
```

SELECT 子句

SELECT子句由SELECT后跟多个SELECT子表达式组成，子表达式可以为*，JSON变量，字符串常量或整数常量。JSON变量后跟一个AS关键字和AS变量，长度不超过32个字符。如果使用常量或函数，则必须使用AS指定名称。

- JSON变量

  JSON变量支持大小写字母，数字，下划线和中划线，为了和减号的意思区分，当使用中划线的时候，请将JSON变量使用双引号进行引用，如："msg-type"。

  Json变量抽取嵌套结构体的数据

  ```
  {
   "a":"b",
   "c": {
     "d" : "e"
   }
  }
  ```

  c.d 即可抽取出字符串e，可以多层嵌套。

- AS变量

  AS变量由大小写字母组成，大小写敏感。目前支持[a-zA-Z_-]*的模式，如果使用中划线，需要使用双引号进行引用。

- 常数整数

  正如标准的SQL一样，SELECT支持常数整数，常数后必须跟AS子句，如

  说明

  常数整数的大小范围：-2147483648~2147483647

  ```
  SELECT 5 AS number
  ```

- 常数字符串

  正如标准的SQL一样，SELECT支持常数字符串,目前支持[a-zA-Z_-]*的模式，需要使用单引号进行引用，常数后必须跟AS子句，如

说明

字符串长度范围不能超过50

```
SELECT 'constant_info' AS str
```

WHERE

在WHERE子句中，您可以用JSON变量进行布尔运算，进行一些非空判断，然后使用AND， OR关键字把结果组合起来。

- **为空判断 IS NULL， IS NOT NULL**

  为空判断可以用在WHERE子句中，如果JSON变量抽取不到数据，或者抽取到的数组为空，那么IS NULL成立，反之IS NOT NULL成立。

  ```
  WHERE data IS NULL
  ```

- **IN， NOT IN**

  IN运算符可以用于WHERE子句中，如果目标值在指定值集合中，则IN成立，NOT IN反之。IN运算符支持字符串和数字，IN集合只支持常量且各集合元素值类型必须一致，集合元素值类型与目标值类型必须一致。

  ```
  WHERE notify_data.header.product_id IN ('productId1','productId2')
  ```

- **大于小于运算符 > <**

  大于小于运算符可以用于WHERE子句中，当且仅当JSON变量的值为常量整数时，可以进行两个JSON变量的比较或者JSON变量和常量的比较。大于小于运算符也可以用于常量和常量的比较。也可以通过AND或者OR来连接起来运算

  比如

  ```
  WHERE data.number > 5 可以抽取出json表达式大于5的信息
  WHERE data.tag < 4 可以抽取出json表达式中小于4的信息
  WHERE data.number > 5 AND data.tag < 4 可以抽取出json表达式data.number大于5的信息并且json表达式data.tag中小于4的信息
  ```

- **=**

  =运算符可以用于WHERE子句中，用于JSON变量和JSON变量的比较、JSON变量整数和整数常量的比较、JSON变量字符串和字符串常量的比较。如果两个JSON变量IS NULL成立，那么=比较结果为false。也可以通过AND或者OR来连接起来运算

  ```
  WHERE data.number = 5 可以抽取出json表达式等于5的信息
  WHERE data.tag = 4 可以抽取出json表达式中等于4的信息
  WHERE data.number = 5 OR data.tag = 4 可以抽取出json表达式data.number等于5的信息或者json表达式data.tag中等于4的信息
  ```

使用限制

表 7-4 SQL 语句使用限制

对象	限制
SELECT子句	500个字符
WHERE子句	500个字符
AS子句	10个AS子句
JSON数据最大深度	400层

调试 SQL 语句

物联网平台提供SQL在线调试功能。调试方法如下。

1. 编写SQL后，单击“调试语句”。
2. 在SQL调试对话框的调试参数页签下，输入用于调试数据，然后单击“启动调试”。

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。

表 7-5 函数列表

函数名称	携带参数	用途	返回值类型	限制
GET_TAG	String tagKey	获取指定tag_key对应的tag_value。 `GET_TAG('testTagName')`	字符串	-
CONTAINS_TAG	String tagKey	判断是否包含指定tag_key。 `CONTAINS_TAG('testTagName')`	布尔值	-
GET_SERVICE	String serviceId，boolean fuzzy	获取service，若fuzzy为false或者不填，则获取指定service_id的service，若fuzzy为true，则通过模糊匹配查询service，如果您在一个消息体里有多个service_id相同的service，结果目前不保证。 `GET_SERVICE('Battery',true)`	Json结构体格式	只能在属性上报时使用
GET_SERVICES	String serviceId，boolean fuzzy	获取services，若fuzzy为false或者不填，获取指定service_id的services，若fuzzy为true，则通过模糊匹配查询services。查询结果将汇合为一个数组。 `GET_SERVICES('Battery',true)`	JSON数组格式	只能在属性上报时使用

函数名称	携带参数	用途	返回值类型	限制
CONTAINS_SERVICES	String serviceId，boolean fuzzy	若fuzzy为false或者不填，则判断是否存在指定service_id。若fuzzy为true，则使用模糊匹配的方式判断属性中的service_id是否包含指定参数。 CONTAINS_SERVICES('Battery',true)	布尔值	只能在属性上报时使用
GET_SERVICE_PROPERTIES	String serviceId	获取指定service_id的service中的properties字段。 GET_SERVICE_PROPERTIES('Battery')	Json结构体格式	只能在属性上报时使用
GET_SERVICE_PROPERTY	String serviceId, String propertyKey	获取指定service_id的service中的properties中指定属性的值。 示例： GET_SERVICE_PROPERTY('Battery','batteryLevel')	字符串	限制只能在属性上报时使用
STARTS_WITH	String input, String prefix	判断input的值是否以prefix开头。 STARTS_WITH('abcd','abc') STARTS_WITH(notify_data.header.device_id,'abc') STARTS_WITH(notify_data.header.device_id,notify_data.header.product_id)	布尔值	-
ENDS_WITH	String input, String suffix	判断input的值是否以suffix结尾。 ENDS_WITH('abcd','bcd') ENDS_WITH(notify_data.header.device_id,'abc') ENDS_WITH(notify_data.header.device_id,notify_data.header.node_id)	布尔值	-
CONCAT	String input1, String input2	用于连接字符串，返回连接后的字符串。 CONCAT('ab','cd') CONCAT(notify_data.header.device_id,'abc') CONCAT(notify_data.header.product_id,notify_data.header.node_id)	字符串	-
REPLACE	String input, String target, String replacement	对字符串某部分值进行替换。即用replacement替换input中的target。 REPLACE(notify_data.header.node_id,'nodeId','IMEI')	-	-

函数名称	携带参数	用途	返回值类型	限制
SUBSTRING	String input, int beginIndex, int endIndex(required=false)	获取字符串的子串，即返回input从beginIndex（包含）到endIndex（不包含）的子字符串。 说明：endIndex非必填。 SUBSTRING(notify_data.header.device_id,3) SUBSTRING(notify_data.header.device_id,3,12)	-	-
LOWER	String input	将input中的值全部转换成小写 LOWER(notify_data.header.app_id)	-	-
UPPER	String input	将input中的值全部转换成大写 UPPER(notify_data.header.app_id)	-	-

7.4 连通性测试

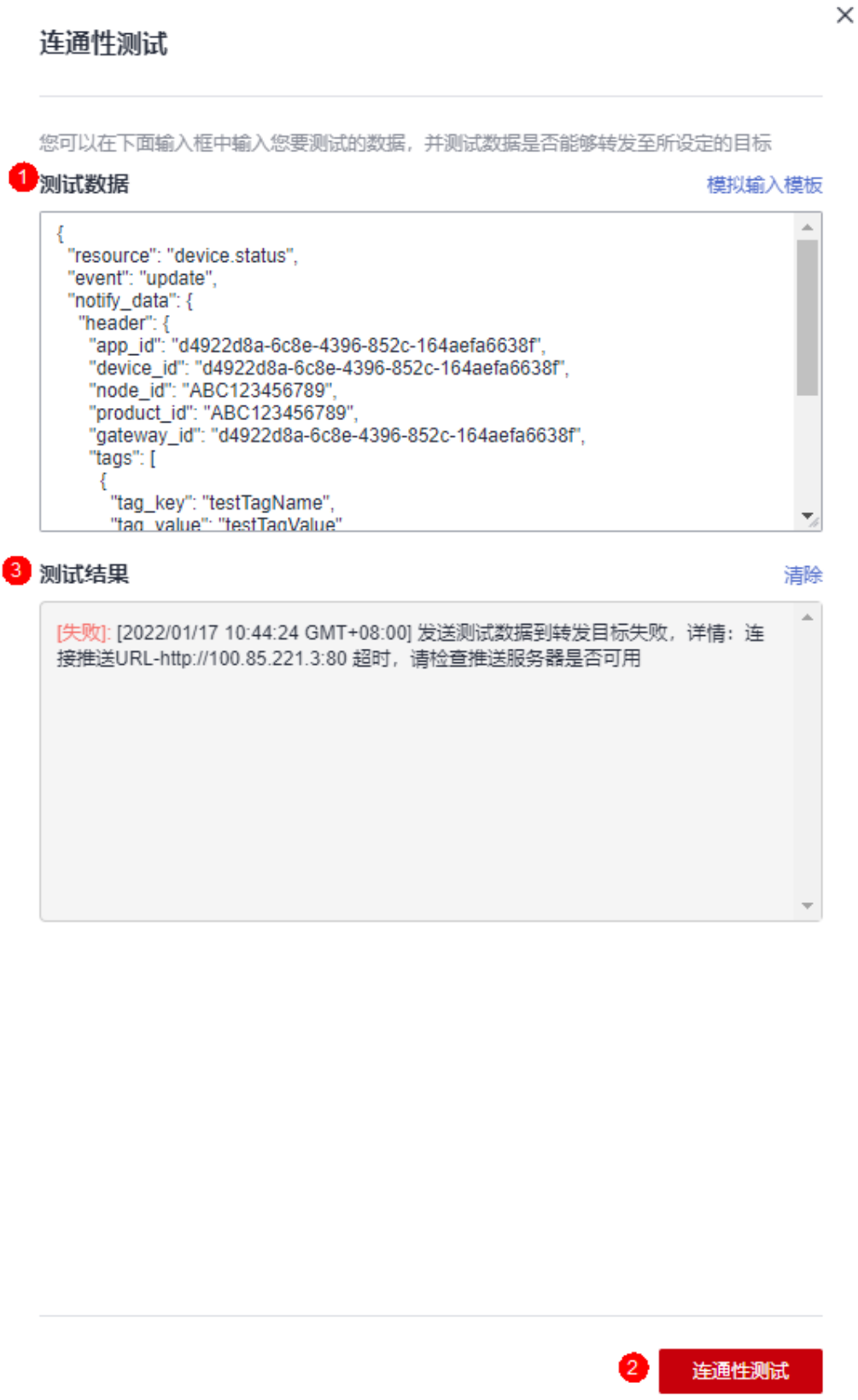
概述

物联网平台提供规则动作转发目标连通性测试功能。在业务对接调测阶段，您可使用连通性测试功能模拟业务数据调测规则动作的可用性及转发数据的业务一致性；在业务运行阶段数据转发出现故障时，您可使用连通性测试功能进行简单的问题复现及定位。

使用步骤

1、创建转发规则及动作后，单击待调测转发目标中的"测试"。

2、在连通性测试对话框的测试数据页签下，输入用于转发的测试数据，或单击右上角"模拟输入模板"，使用模板数据，然后单击"连通性测试"。



7.5 数据转发至华为云服务

7.5.1 数据转发至 DIS

场景说明

将数据流转到DIS云服务，可让您轻松收集、处理和分发实时流数据，以便您对新信息快速做出响应。DIS对接多种第三方数据采集工具，提供丰富的云服务Connector及Agent/SDK。也可以通过转储任务进一步将数据转发到其他云服务进行数据存储、分析，便于客户灵活使用。

购买 DIS 接入通道（以数据流转到 DIS 转存储至 OBS 为例）

步骤1 登录华为云，访问**对象存储服务OBS**，进入对象存储服务管理控制台。

步骤2 单击“创建桶”，按照需求选择桶规格，单击“立即创建”。

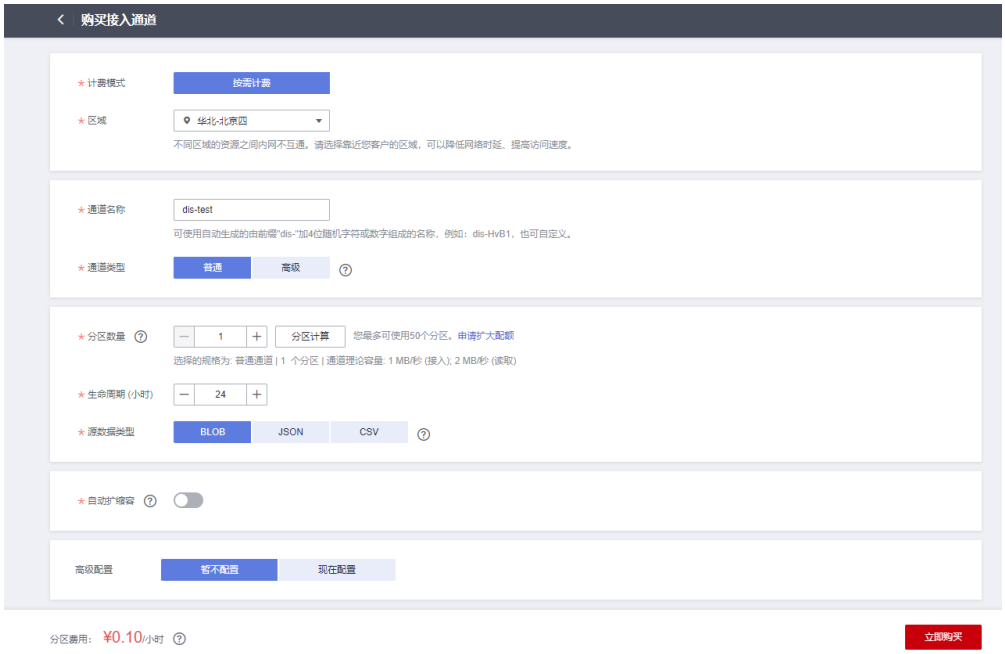
步骤3 单击“桶列表”，单击进入刚刚创建的桶，单击“新建文件夹”，完成OBS桶创建。

步骤4 登录华为云官方网站，访问**数据接入服务DIS**。

步骤5 单击“管理控制台”进入数据接入服务管理控制台。

步骤6 单击右上角“购买接入通道”，根据需求选择通道规格，单击“立即购买”。

图 7-2 购买接入通道



步骤7 单击左侧导航栏“通道管理”按钮，单击选择已购买的通道，选择“转储任务”，单击“添加转储任务”，转储类型选择OBS，数据存储地址选择步骤2中创建的桶，转存储目录选择步骤3中创建的文件夹，单击“立即创建”。

图 7-3 选择通道

图 7-4 选择转存储任务

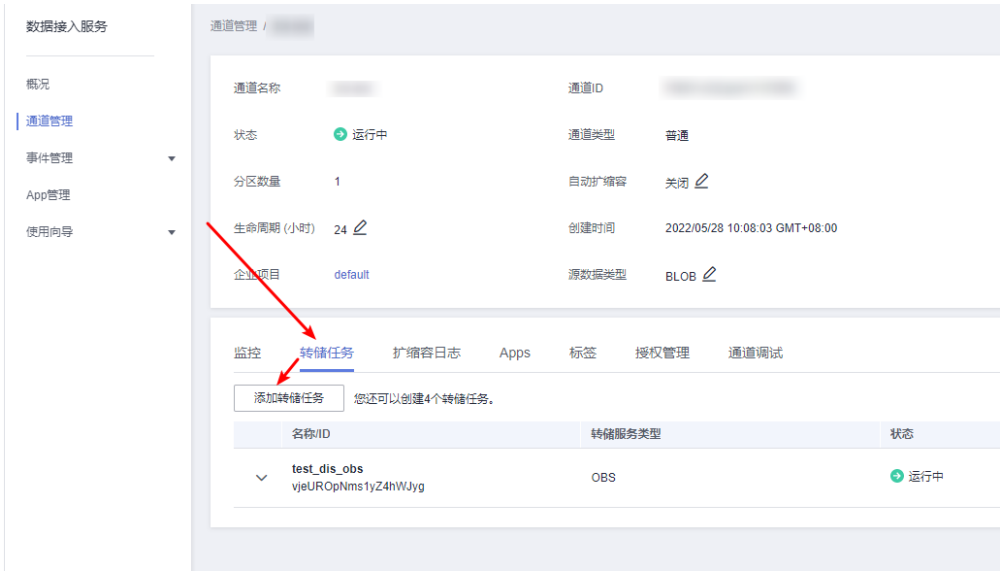
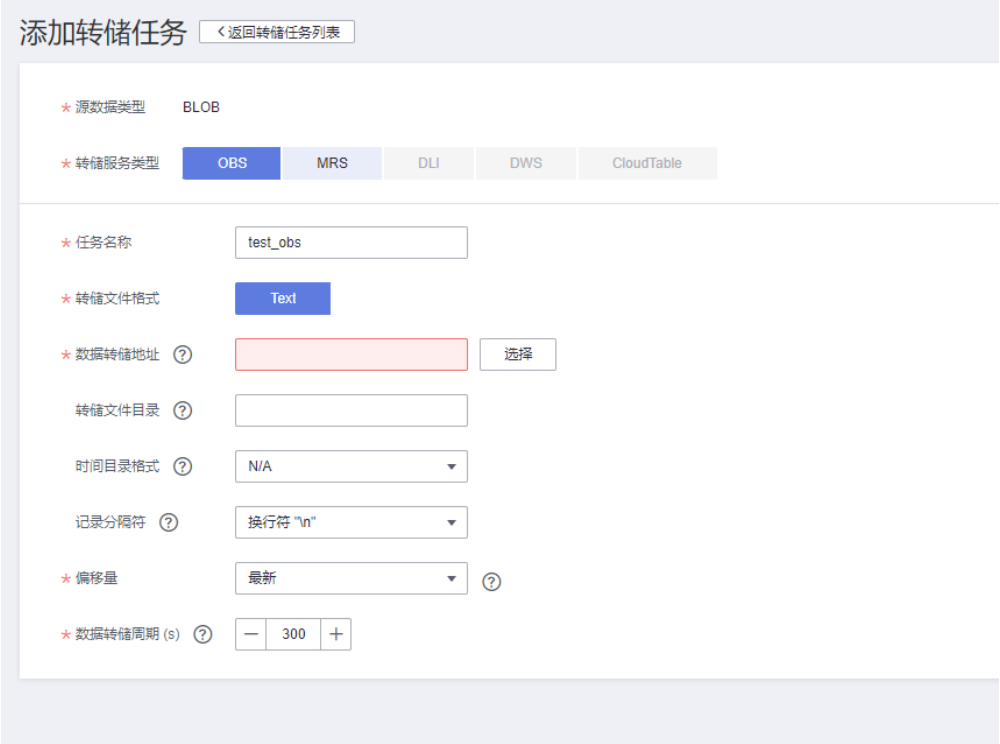


图 7-5 修改转存储任务



----结束

配置设备接入服务

在设备接入服务中设置数据转发规则，实现当设备上报数据时将数据转发至DIS。

步骤1 访问**设备接入服务**，单击“立即使用”进入设备接入控制台。

步骤2 在左侧导航栏选择“规则>数据转发”，单击左上角的“创建规则”。

步骤3 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**创建自己的规则，填写完成后单击“创建规则”。

表 7-6 创建规则参数说明

参数名	参数说明
规则名称	自定义，如“iotda-dis”。
规则描述	自定义，如“数据转发至DIS”。
数据来源	选择“设备属性”。
触发事件	自动匹配“设备属性上报”。
资源空间	选择“所有资源空间”。

步骤4 单击“设置转发目标”页签，单击“添加”，设置转发目标，设置完成后单击“确定”按钮。

表 7-7 转发目标参数说明

参数名	参数说明
转发目标	选择“数据接入服务（DIS）”。
区域	数据接入服务当前仅支持转发至同区域的接入通道。若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。
通道归属	选择添加的通道是自有通道还是他人授权的通道。
通道	选择通道名称。

图 7-6 创建数据转发目标

步骤5 单击“启动规则”，激活配置好的数据转发规则。

----结束

验证操作

- 您可以使用配置设备接入服务时注册的真实设备接入平台，上报任意数据。
- 您也可以使用模拟器模拟设备上报数据，操作方法请参考**在线开发MQTT协议的智慧路灯**。

期望结果：

登录OBS的管理控制台，单击进去**2**中创建的桶，再单击进去**3**中创建的文件夹可以看到最新由DIS转发至OBS的数据。

图 7-7 查看 OBS 数据

7.5.2 数据转发至 GeminiDB Influx

场景说明

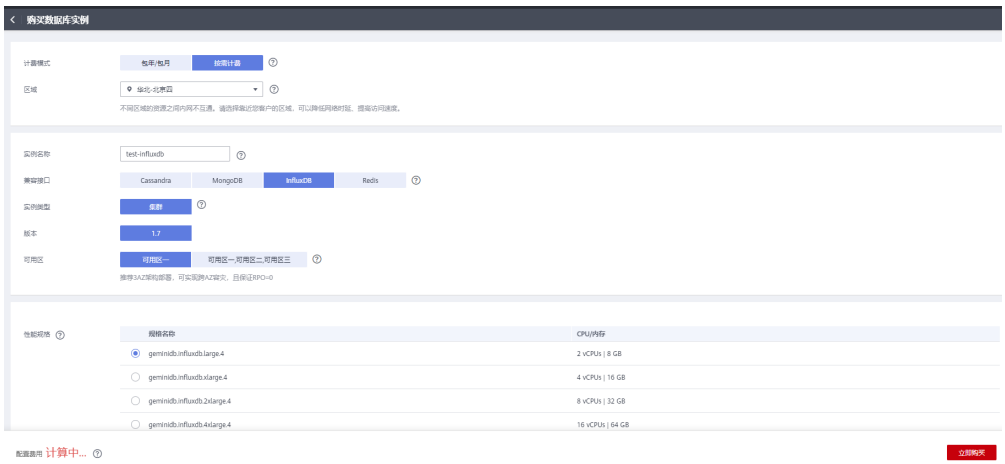
将数据流转到influxDB，兼容InfluxDB生态的云原生时序数据库。提供高性能读写、高压缩率、冷热分层存储以及弹性扩容、监控告警等服务能力，可以实现大并发的时序数据读写，压缩存储和类SQL查询等功能，支持多维聚合计算和数据可视化分析能力。广泛应用于资源监控、业务监控分析、物联网设备实时监控、工业生产监控、生产质量评估和故障回溯等。提供了高吞吐量和并发性，可以通过快速的响应时间来支持大量的连接，非常适合要求苛刻的物联网应用。

购买 GeminiDB Influx

步骤1 登录**云数据库GeminiDB Influx**，单击“立即购买”。

步骤2 根据需求选择按需计费或者包年包月以及性能规格、存储空间等，兼容接口选择InfluxDB。具体可参考**购买集群实例**。

图 7-8 购买 influxDB 实例

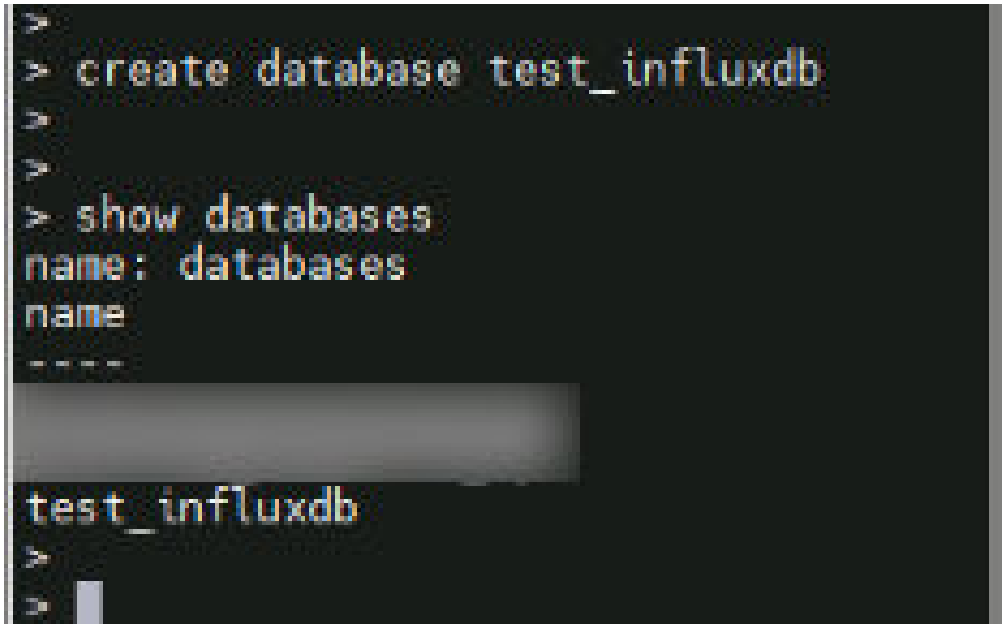


步骤3 下载InfluxDB客户端，通过客户端连接实例，参考**公网连接实例**。

步骤4 使用客户端连接实例后，通过以下命令创建数据库，${databaseName}可以自定义。

```
create database ${databaseName}
```

图 7-9 创建数据库



----结束

配置设备接入服务

在设备接入服务中设置数据转发规则，实现当设备上报数据时将数据转发至InfluxDB。

步骤1 访问**设备接入服务**，单击“立即使用”进入设备接入控制台。

步骤2 在左侧导航栏选择“规则>数据转发”，单击左上角的“创建规则”。

步骤3 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**创建自己的规则，填写完成后单击“创建规则”。

表 7-8 创建规则参数说明

参数名	参数说明
规则名称	自定义，如“iotda-InfluxDB”。
规则描述	自定义，如“数据转发至InfluxDB”。
数据来源	选择“设备”。
触发事件	自动匹配“设备添加”。
资源空间	选择“所有资源空间”。

步骤4 单击“设置转发目标”页签，单击“添加”，设置转发目标，设置完成后单击“下一步”按钮。

表 7-9 创建转发目标参数说明

参数名	参数说明
转发目标	选择“时序数据库（InfluxDB）”。
数据库实例地址	填写购买购买的influx连接地址。
数据库名称	填写在InfluxDB中创建的数据库名。
访问账户	InfluxDB的账户名。
访问密码	InfluxDB的密码。

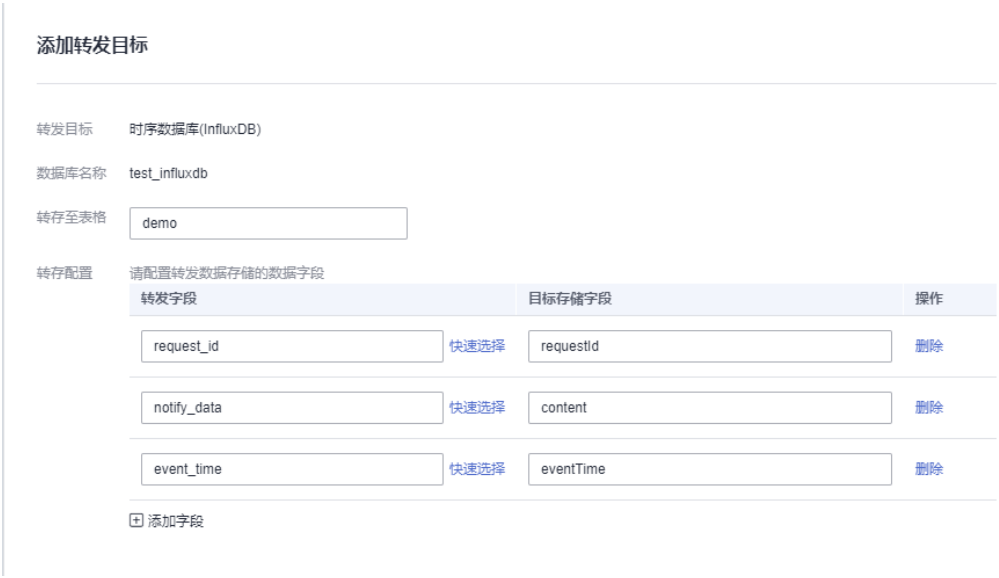
步骤5 填写转存储字段， 填写完成后单击“确定”完成配置。

表 7-10 字段映射参数说明

参数名	参数说明
转存储至表格	填写表格名称（自定义）。
转存配置	填写填写转发字段与转存字段的映射（自定义），转发字段可以参考**设备添加通知**。

图 7-10 创建转发目标

图 7-11 设置转发字段映射



步骤6 单击“启动规则”，激活配置好的数据转发规则。

----结束

验证操作

进入IoTDA管理控制台，创建设备。

期望结果：

使用客户端登录InfluxDB，进入数据库，查询数据成功。

```
show databases     //查询数据库
use test_influxdb  //切换数据库
select * from demo //查询数据
```

图 7-12 验证消息

7.5.3 数据转发至 Kafka 存储

场景说明

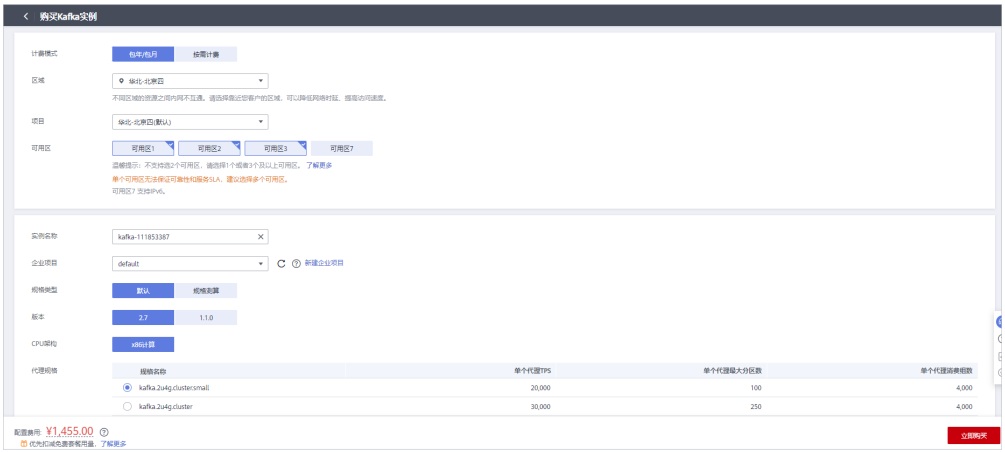
对于设备上报的数据，可以选择让平台将设备上报数据推送给应用服务器，由应用服务器进行保存；还可以选择让平台将设备上报数据转发给分布式消息服务（Kafka），由Kafka进行存储。

本示例为将所有设备上报的数据转发至Kafka存储。

购买 Kafka 实例

1. 登录华为云官方网站，访问**分布式消息服务**。
2. 单击“进入控制台”进入分布式消息服务管理控制台。
3. 单击右上角“购买Kafka实例”，根据需求选择实例规格与**配置安全组**后，单击“立即购买”。

图 7-13 购买 Kafka 实例



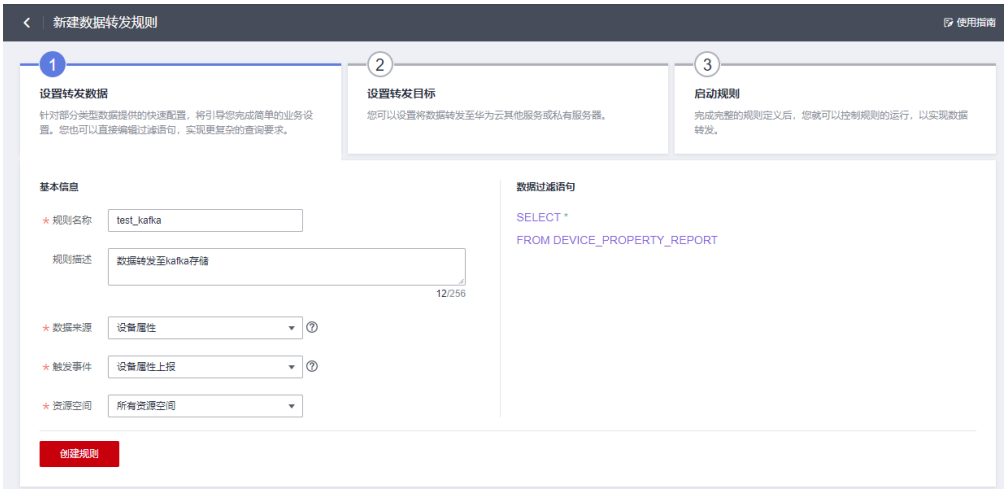
配置设备接入服务

在设备接入服务中创建产品模型、注册设备并设置数据转发规则，实现当设备上报数据时将数据转发至Kafka。

1. 访问**设备接入服务**，单击“立即使用”进入设备接入控制台。
2. 在左侧导航栏选择“规则>数据转发”，单击左上角的“创建规则”。
3. 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**创建自己的规则，填写完成后单击“创建规则”。

参数名	参数说明
规则名称	自定义，如iotda-kafka。
规则描述	自定义，如数据转发至Kafka存储。
数据来源	选择“设备属性”。
触发事件	自动匹配“设备属性上报”。
资源空间	选择“所有资源空间”。

图 7-14 创建数据转发规则



4. 单击“设置转发目标”页签，单击“添加”，设置转发目标。

参数名	参数说明
转发目标	选择“分布式消息服务(kafka)”
区域	选择Kafka服务的所在区域。若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。
对接地址	参考**连接已开启SASL的Kafka专享版实例**获取对接地址。设备接入基础版和标准版实例只支持公网接入专享版Kafka，企业版实例支持私网接入专享版Kafka。
主题	自定义主题，参考**创建Topic**。
SASL认证	若开启SASL认证，请填写您在**购买Kafka实例**中所选安全协议、SASL认证机制以及所填的SASL用户名和密码。
Kafka安全协议	填写您在**购买Kafka实例**中所启用的kafka安全协议。
SASL认证机制	填写您在**购买Kafka实例**中所开启的SASL认证机制。
SASL用户名	填写您在**购买Kafka实例**中输入的SASL用户名。
密码	填写您在**购买Kafka实例**中输入的密码。

图 7-15 创建数据转发目标

5. 单击“启动规则”，激活配置好的数据转发规则。

图 7-16 激活规则

验证操作

- 您可以使用配置设备接入服务时注册的真实设备接入平台，上报任意数据。
- 您也可以使用模拟器模拟设备上报数据，操作方法请参考**在线开发MQTT协议的智慧路灯**。

期望结果：

登录Kafka**管理控制台**，单击Kafka实例名进入实例管理页面后，在“消息查询”页面可以查看到设备上报的数据。

图 7-17 Kafka 服务查询数据上报信息

您也可以使用Kafka的API（**查询消息**）进行文件读取。

7.5.4 数据转发至 FunctionGraph 函数工作流

场景说明

对于设备上报到平台的数据，使用函数工作流FunctionGraph处理实时流数据。通过函数服务，用户只需编写业务函数代码并设置运行的条件，无需配置和管理服务器等基础设施，即可跟踪设备的设备属性、消息上报，状态变更，分析、整理和计量数据流。

本示例为将所有设备上报的属性转发至FunctionGraph函数工作流，根据设备资源空间Id的不同推送到用户Http服务器的不同路径，请您提供自行部署的Http服务器。本示例通过设备接入服务的数据转发能力来驱动事件函数，不需要额外配置触发器。

构建函数工程

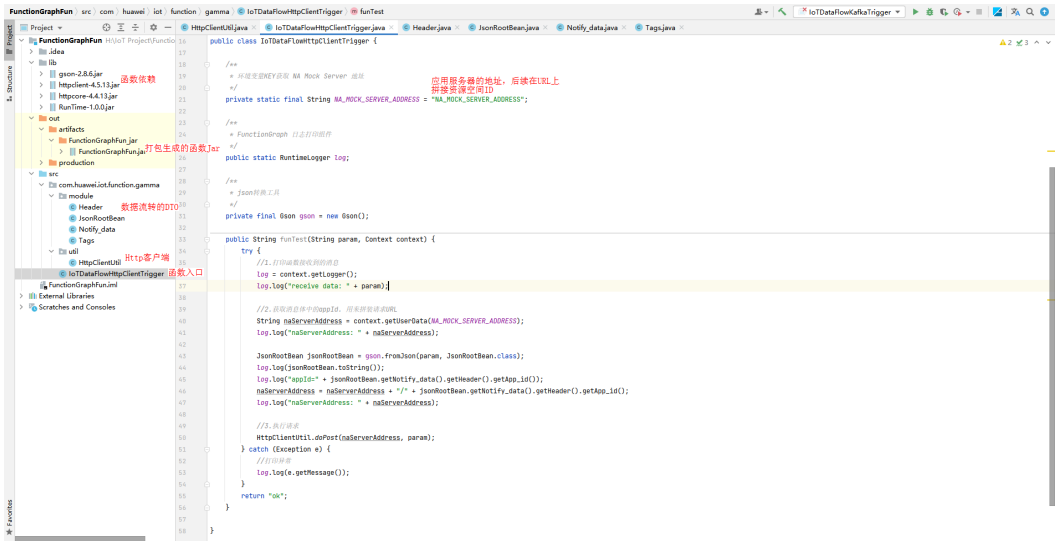
本例提供了设备属性上报格式转换并转发到第三方应用的源码（包含函数依赖），用户可以下载学习使用。

创建工程

本例使用Java语言实现设备接入属性数据流式转换功能与推送功能，有关函数开发的过程请参考FunctionGraph的**Java函数开发指南**，本例不再介绍函数工作流函数实现的代码。

下载样例源码，解压缩并在Idea中导入工程。代码说明可参考：**样例代码说明**，其中用户自己的服务器地址通过函数环境变量NA_MOCK_SERVER_ADDRESS传入。

图 7-18 样例代码说明



打包工程

使用Idea的Build Artifacts打包Jar，Idea配置及打包如下图所示。

图 7-19 工程 Artifacts Output 配置参考

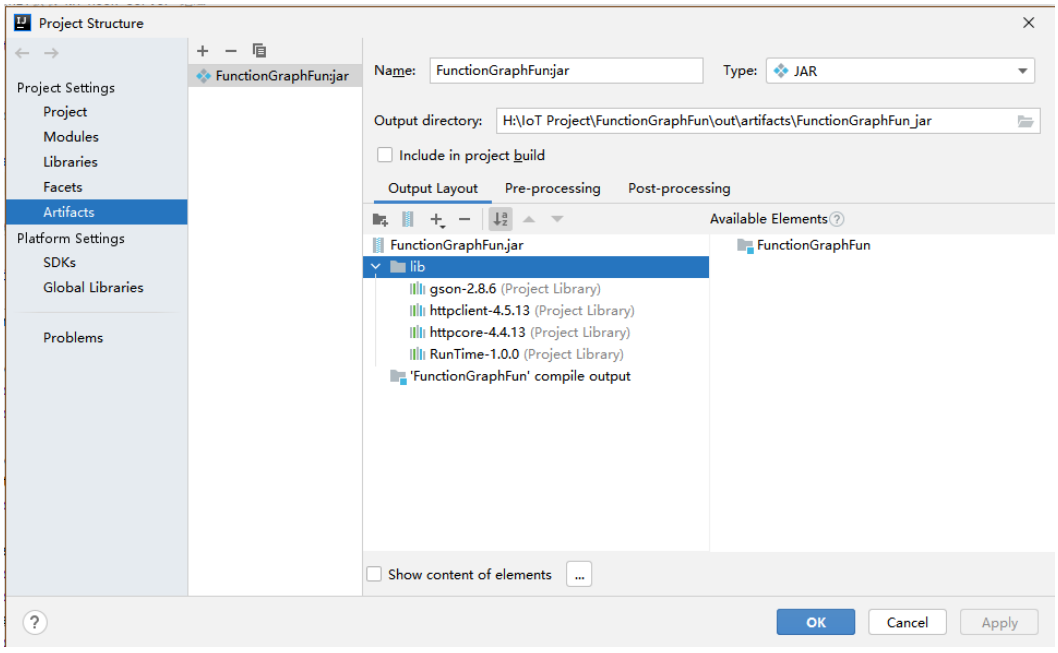
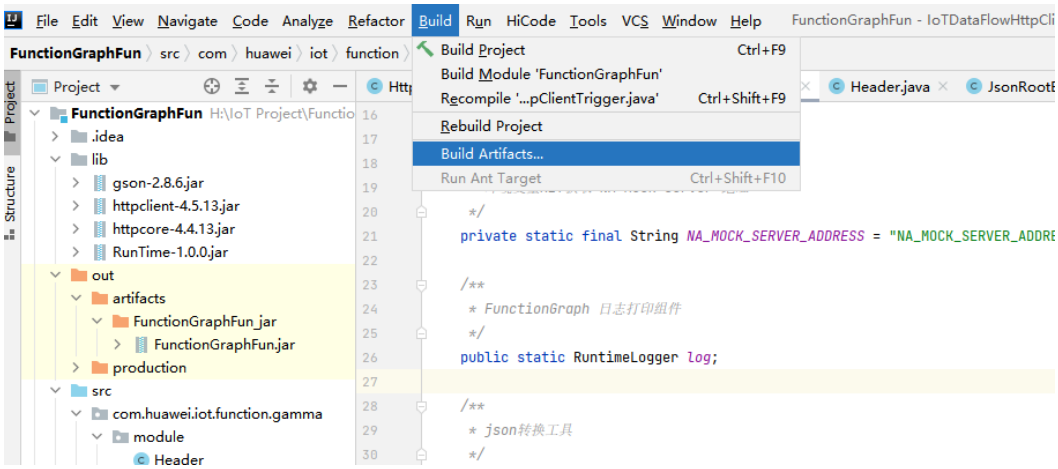


图 7-20 Build Artifacts



上传函数至 FunctionGraph

在函数工作流工作台创建函数

步骤1 登录函数工作流控制台在左侧导航栏选择“函数 > 函数列表”，进入函数列表界面。

步骤2 单击“创建函数”，进入创建函数流程。

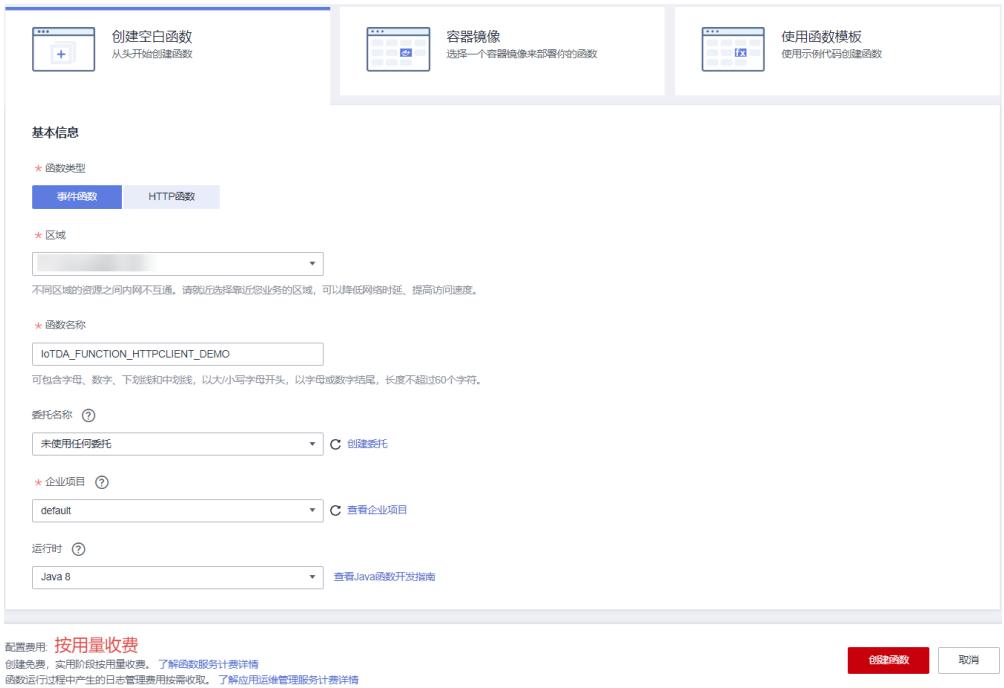
步骤3 填写函数配置信息，如下图所示。

选择：“创建空白函数”。

函数名称输入：“IoTDA_FUNCTION_HTTPCLIENT_DEMO”。

运行时语言选择：“Java 8”。

图 7-21 创建函数界面



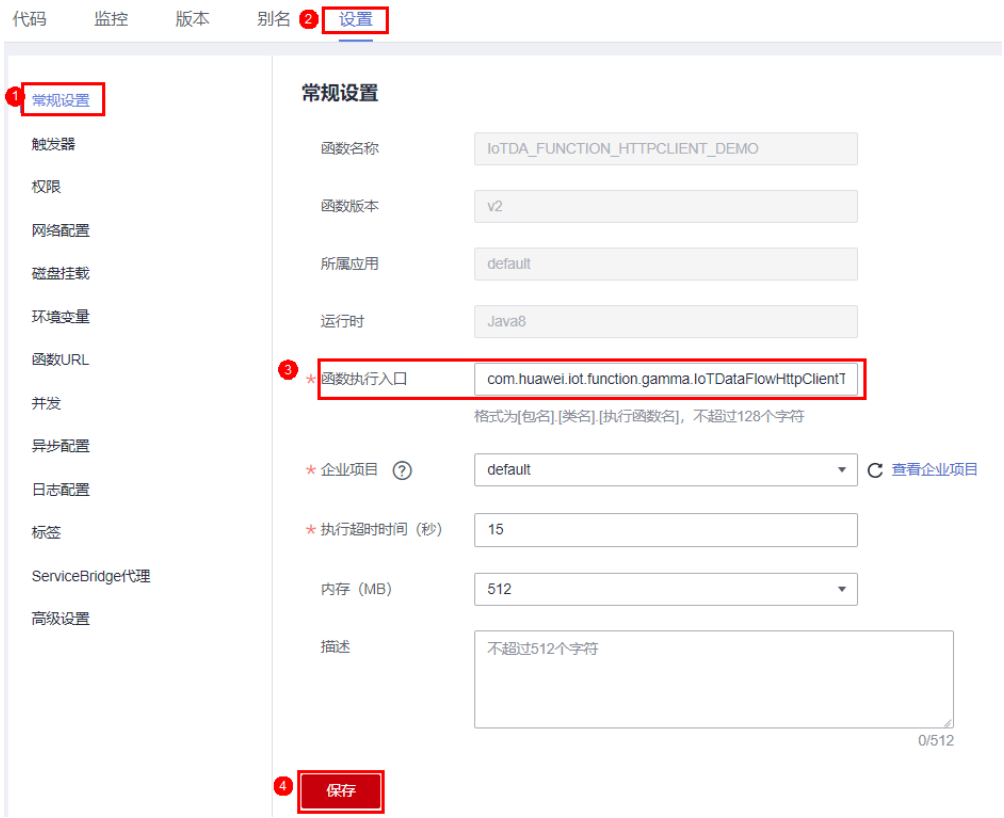
步骤4 单击下方“创建函数”，完成函数创建，完成函数创建后将自动跳转到该函数详情页。

步骤5 在函数详情页中选择“代码”>“上传自”>“JAR文件”，上传程序打包文件夹中的代码包：FunctionGraphFun.jar。

图 7-22 上传代码界面

步骤6 修改函数运行时参数，选择“设置>常规设置”配置函数执行入口：“com.huawei.iot.function.gamma.IoTDataFlowHttpClientTrigger.funTest”，填写完成后，单击打“保存”，保存配置信息。

图 7-23 设置函数执行入口

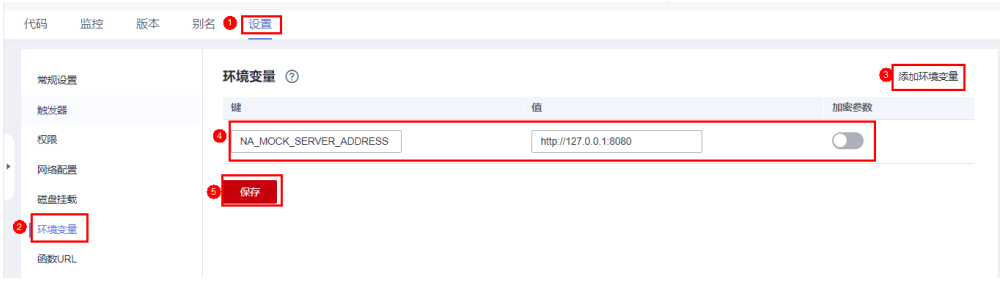


须知

函数默认内存为512MB，超时时间默认为15s，该样例仅演示功能，如需商用，请结合实际使用场景进行参数优化。

步骤7 修改调用函数时传递的环境变量，配置环境变量“NA_MOCK_SERVER_ADDRESS”传入函数要推送的HttpServer地址，注意示例中服务器地址非真实服务器地址，请替换成您真实的http服务器地址，填写完成后，单击“保存”，保存配置信息。

图 7-24 配置函数调用环境变量



----结束

添加事件源

函数创建以后，可以为函数添加事件源，本例通过配置Http推送测试事件，模拟IoT数据转发过来的设备数据，步骤如下。

步骤1 用户进入“IoTDA_FUNCTION_HTTPCLIENT_DEMO”函数详情页，选择“代码”>“配置测试事件”，弹出“配置测试事件”窗口。

图 7-25 配置测试事件

步骤2 在“配置测试事件”窗口中，输入配置信息。

配置测试事件选择：“创建新的测试事件”。

事件模板选择：“空白模板”。

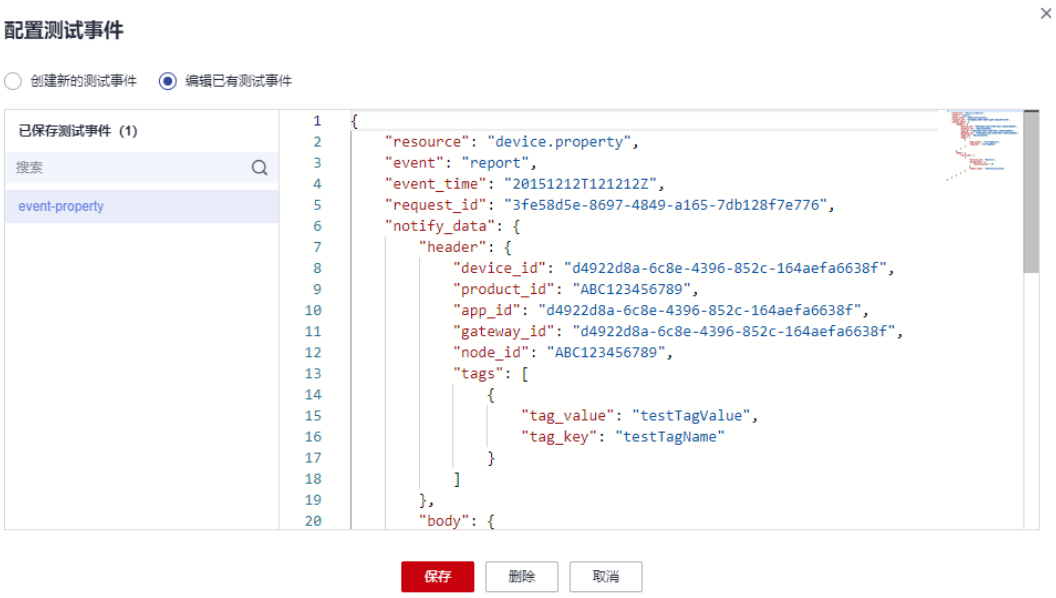
事件名称输入：“event-property”。

设备属性上报测试参数示例如下：

```
{
  "resource":"device.property",
  "event":"report",
  "event_time":"string",
  "notify_data":{
     "header":{
        "app_id":"d4922d8a-6c8e-4396-852c-164aefa6638f",
        "device_id":"d4922d8a-6c8e-4396-852c-164aefa6638f",
```

```
            "node_id":"ABC123456789",
            "product_id":"ABC123456789",
            "gateway_id":"d4922d8a-6c8e-4396-852c-164aefa6638f",
            "tags":[{
                    "tag_key":"testTagName",
                    "tag_value":"testTagValue"
             }]
        },
        "body":{
            "services":[{
                    "service_id":"string",
                    "properties":{
                    },
                    "event_time":"string"
             }]
        }
    }
}
```

图 7-26 配置测试事件



步骤3 单击“保存”，完成测试事件配置。

----结束

测试数据

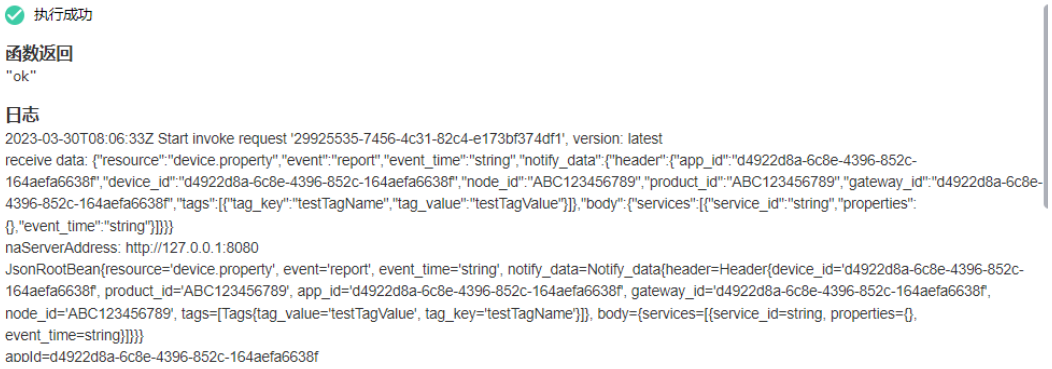
处理模拟数据步骤如下。

步骤1 用户进入函数详情页，选择“event-property”测试事件，单击“测试”，测试函数。

图 7-27 配置测试事件

步骤2 函数执行成功后，可在函数详情页右侧通过日志查看函数执行情况。

图 7-28 函数执行结果



----结束

配置设备接入服务

在设备接入服务中设置数据转发规则，实现当设备上报数据时将数据转发至FunctionGraph。

步骤1 访问**设备接入服务**，单击“立即使用”进入设备接入控制台。

步骤2 在左侧导航栏选择“规则>数据转发”，单击左上角的“创建规则”。

步骤3 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**创建自己的规则，填写完成后单击“创建规则”。

参数名	参数说明
规则名称	自定义，如“iotda-functiongraph”。
规则描述	自定义，如“数据转发至FunctionGraph”。
数据来源	选择“设备属性”。
触发事件	自动匹配“设备属性上报”。
资源空间	选择“所有资源空间”。

步骤4 单击“设置转发目标”页签，单击“添加”，设置转发目标，设置完成后单击“确定”按钮。

参数名	参数说明
转发目标	选择“函数工作流(FunctionGraph)”
区域	函数工作流当前仅支持转发至同区域的函数工作流服务。若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。
目标函数	选择在函数工作流创建的函数名称。

步骤5 单击“启动规则”，激活配置好的数据转发规则。

----结束

验证操作

- 您可以使用设备接入服务中注册的真实设备接入平台，上报对应产品物模型中定义的属性参数。
- 您也可以使用模拟器模拟设备属性上报，操作方法请参考**在线开发MQTT协议的智慧路灯**。

期望结果：能在用户服务端日志中查看到设备上报的数据。

图 7-29 期望结果

```
@RestController
public class MessageReceiveController {
    private static final Logger log = LoggerFactory.getLogger(MessageReceiveController.class);

    @RequestMapping(value = "/{appId}", method = RequestMethod.POST)
    public void messageReceive(@NotNull @PathVariable("appId") String appId,
        @RequestBody JSONObject requestJson) {
        log.info("receive {}, message is {}", appId, requestJson.toJSONString());
    }
}
```

```
NApplication
onsole   Endpoints
 (javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2021-04-16 12:39:49.369  INFO 24304 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2021-04-16 12:39:49.398  INFO 24304 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-04-16 12:39:49.400  INFO 24304 --- [           main] com.huawei.demo.na.NApplication          : Started NApplication in 2.249 seconds (JVM running for 3.378)
2021-04-16 12:39:51.071  INFO 24304 --- [n(2)-172.23.0.1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring FrameworkServlet 'dispatcherServlet'
2021-04-16 12:39:51.071  INFO 24304 --- [n(2)-172.23.0.1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization started
2021-04-16 12:39:51.131  INFO 24304 --- [n(2)-172.23.0.1] o.s.web.servlet.DispatcherServlet        : FrameworkServlet 'dispatcherServlet': initialization completed in 60 ms
2021-04-16 12:44:15.071  INFO 24304 --- [nio-8080-exec-1] c.h.d.n.c.MessageReceiveController       : receive f20f[redacted], message is {"resource":"device.property",
 "notify_data":{"header":{"device_id":"d492[redacted]38f","product_id":"[redacted]","app_id":"f20f[redacted]","gateway_id":"d49[redacted]",
 "node_id":"[redacted]","tags":[{"tag_value":"testTagValue","tag_key":"testTagName"}]},"body":{"services":[{"service_id":"temp","properties":"temp","event_time":"20151212T121212Z"}]}},"event":"report",
 "event_time":"20151212T121212Z"}
```

7.5.5 数据转发至 MySQL 存储

场景说明

对于平台的流转数据可以选择让平台将设备上报数据转发给云数据库（MySQL），由MySQL进行存储，用户无需做额外的数据存储代码开发即可使用设备数据进行业务处理。

本示例为将流转数据转发至MySQL存储。

前提条件

- 已购买设备接入服务的企业版实例或标准版实例。
- 已购买云数据库MySQL实例。

创建 MySQL

步骤1 登录华为云官方网站，访问**云数据库 MySQL**，购买实例。设备接入服务企业版实例支持**通过内网连接MySQL**，标准版实例仅支持**通过公网连接MySQL**。

步骤2 在购买的MySQL实例中，根据**流转数据**格式设计数据库表，可以通过数据过滤语句编辑流转数据。本示例使用**设备属性上报通知的默认格式**，将流转数据中的resource、event、notify_data、event_time分别转存至数据库表中的resource、event、content、event_time字段。

图 7-30 创建数据库表样例

```
CREATE TABLE `notify_all` (
  `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
  `resource` varchar(32) DEFAULT NULL,
  `event` varchar(32) DEFAULT NULL,
  `content` json DEFAULT NULL,
  `event_time` char(16) DEFAULT NULL,
  `receive_time` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4;
```

----结束

配置设备接入服务

在设备接入服务中创建产品模型、注册设备并设置数据转发规则，实现当设备上报数据时将数据转发至MySQL。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 选择左侧导航栏的“产品”，单击右上角创建产品，选择新建产品所属的资源空间。

说明

本文中使用的产品模型和设备仅为示例，您可以使用自己的产品模型和设备进行操作。

步骤3 单击右上角的“创建产品”，创建一个基于MQTT协议的产品，填写参数后，单击“确认”。

基本信息	
产品名称	自定义，如MQTT_Device
协议类型	选择“MQTT”
数据格式	选择“JSON”
所属行业	根据实际情况进行填写。
设备类型	

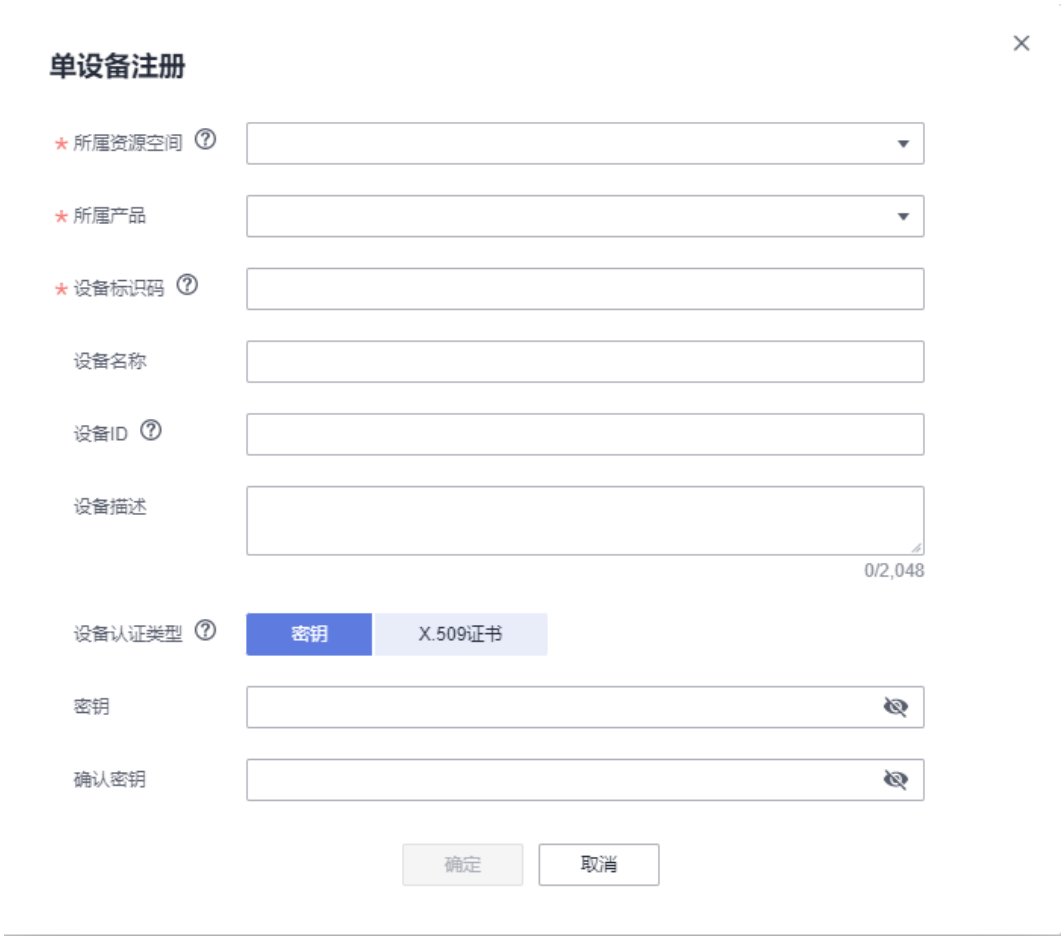
步骤4 单击**文件**，获取产品模型文件样例。

步骤5 在模型定义页面，单击“上传模型文件”，在弹出的页面中加载产品模型文件，然后单击“确认”。

图 7-31 上传模型文件

步骤6 选择左侧导航栏的“设备 > 所有设备”，单击右上角的“注册设备”，填写设备注册参数，完成后单击“确定”，请注意保存注册成功返回的“设备ID”和“设备密钥”。

图 7-32 创建设备



参数名称	说明
所属资源空间	选择步骤3中创建的产品所在的资源空间

参数名称	说明
所属产品	选择步骤3中创建的产品。
设备标识码	即node_id，填写为设备的IMEI、MAC地址或Serial No；若没有真实设备，填写自定义字符串，由英文字母和数字组成。
设备名称	自定义。
设备ID	自定义，可不填，平台会自动生成一个设备ID
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义，不填写物联网平台会自动生成。

步骤7 选择左侧导航栏的“规则>数据转发”，单击左上角的“创建规则”。

说明

您也可以基于已经创建的规则，在规则详情页面，添加新的转发目标MySQL。

步骤8 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**填写具体的流转规则参数，填写完成后单击“创建规则”。

参数名	参数说明
规则名称	自定义，如iotda-mysql。
规则描述	自定义，如数据转发至MySQL存储。
数据来源	选择“设备属性”。
触发事件	自动匹配“设备属性上报”。
资源空间	选择转发的数据来源所属的资源空间，或者所有资源空间。

步骤9 单击“设置转发目标”页签，单击“添加”，设置转发目标。

参数名	参数说明
转发目标	选择“MySQL数据库”
数据库实例地址	填写数据库实例的连接IP（或域名）和端口。
数据库名称	填写数据库实例中转发目标数据库名称。
访问账户	填写数据库实例的账户。
访问密码	填写数据库实例的密码。
SSL	选择是否通过SSL加密方式连接数据库。推荐使用SSL方式连接，不开启SSL方式连接可能存在数据传输安全风险。选择通过SSL方式连接时，需要先在数据库实例中**设置SSL数据加密**。

步骤10 单击“下一步：转发数据配置”，此过程会连接数据库。

步骤11 选择转存表格，配置流转数据和数据库表的映射关系。

- 转发字段：流转数据的json key。
- 目标存储字段：数据库表的字段，选择目标存储字段后自动匹配字段类型。

图 7-33 创建数据转发目标

步骤12 单击“启动规则”，激活配置好的数据转发规则。

----结束

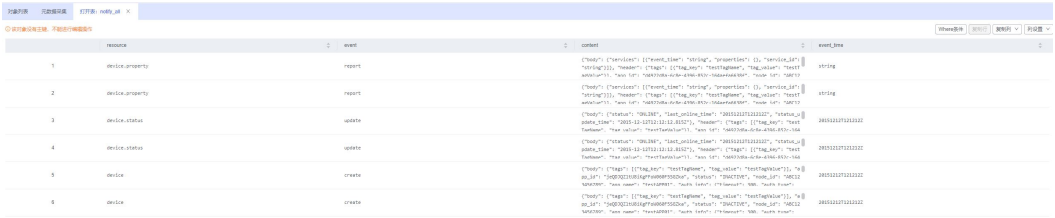
验证操作

触发规则数据来源的事件，比如设备属性上报。

期望结果：

登录MySQL**管理控制台**，打开目标表格，可以查看到表格目标字段转存的数据。

图 7-34 查询数据上报信息



7.5.6 数据转发至 OBS 长期存储

场景说明

对于设备上报的数据，可以选择让平台将设备上报数据推送给应用服务器，由应用服务器进行保存；还可以选择让平台将设备上报数据转发给对象存储服务（OBS），由OBS进行存储。

本示例为将所有设备上报的数据转发至OBS存储。

创建 OBS 桶

步骤1 登录华为云官方网站，访问**对象存储服务**。

步骤2 单击“进入控制台”进入对象存储服务管理控制台。

步骤3 单击页面右上角的“创建桶”，根据需求选择桶规格后，单击“立即创建”。

图 7-35 购买 OBS 服务

----结束

配置设备接入服务

在设备接入服务中创建产品模型、注册设备并设置数据转发规则，实现当设备上报数据时将数据转发至OBS。

创建规则

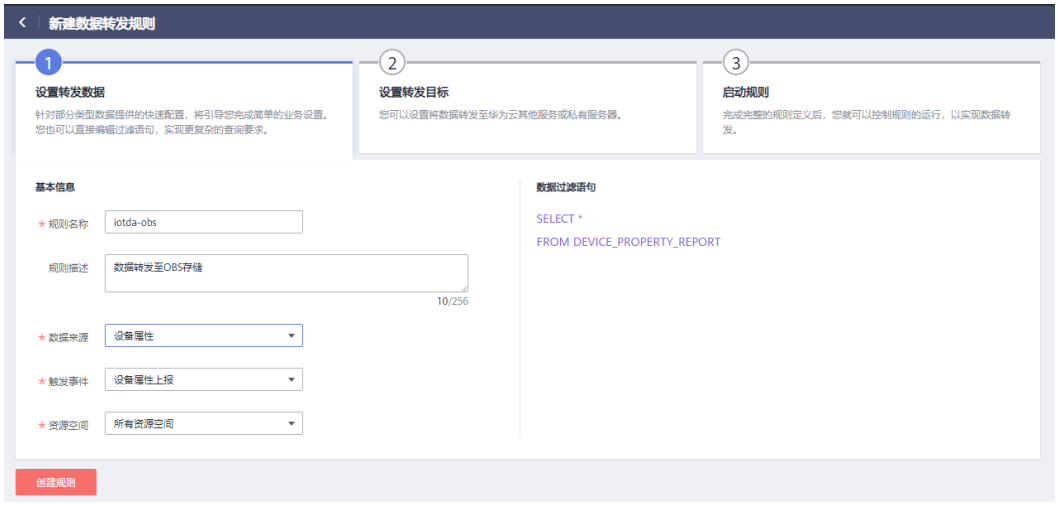
步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则>数据转发”，单击左上角的“创建规则”。

步骤3 参考下表参数说明，填写规则内容。以下参数取值仅为示例，您可参考**数据转发简介**创建自己的规则，填写完成后单击“创建规则”。

参数名	参数说明
规则名称	自定义，如iotda-obs。
规则描述	自定义，如数据转发至OBS存储。
数据来源	选择“设备属性”。
触发事件	自动匹配“设备属性上报”。
资源空间	选择“所有资源空间”。

图 7-36 创建数据转发规则



步骤4 单击“设置转发目标”页签，单击“添加”，设置转发目标。

参数名	参数说明
转发目标	选择“对象存储服务（OBS）”
区域	选择OBS服务的所在区域。若未授权访问此区域的服务，请根据界面提示，配置云服务访问授权。
存储桶	选择要存储数据的桶。若没有，请前往OBS服务**创建桶**。

图 7-37 创建转发目标

步骤5 单击“启动规则”，激活配置好的数据转发规则。

图 7-38 激活规则

----结束

验证操作

- 您可以使用配置设备接入服务时注册的真实设备接入平台，上报任意数据。
- 您也可以使用模拟器模拟设备上报数据，操作方法请参考**在线开发MQTT协议的智慧路灯**。

登录OBS**管理控制台**，单击桶名称进入桶管理页面后，在“对象”页面可以查看到设备上报的数据。

图 7-39 OBS 服务查询上报数据

您也可以使用OBS的API（**获取对象内容**）进行文件读取。

7.6 数据转发至第三方应用

7.6.1 转发方式概述

设备接入到物联网平台后，便可与物联网平台进行通信。设备通过自定义Topic或产品模型方式将数据上报到平台，在控制台设置后，通过订阅推送的方式，将设备生命周

期变更、设备属性上报、设备消息上报、设备消息状态变更、设备状态变更、批量任务状态变更等消息转发到您指定的服务器。

当前华为物联网平台支持HTTP/HTTPS、AMQP、MQTT和设备间通信四种数据转发方式。

- HTTP/HTTPS方式
  - 订阅：应用服务器通过调用物联网平台的**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**接口配置并激活规则，或者在控制台创建订阅任务，向平台获取发生变更的设备业务信息（如设备生命周期管理、设备数据上报、设备消息状态、设备状态等）和管理信息（软固件升级状态和升级结果）。订阅时必须指定应用服务器的URL，也称为回调地址。（**什么是回调地址？**）。
  - 推送：订阅成功后，物联网平台根据应用服务器订阅的数据类型，将对应的变更信息（推送的通知内容可参考**流转数据**）推送给指定的URL地址。如果应用服务器没有订阅该类型的数据通知，即使数据发生了变更也不会进行推送。物联网平台进行数据推送时，数据格式为JSON格式，推送协议可以采用HTTP或HTTPS协议，其中HTTPS协议为加密传输协议，需要进行安全认证，更加安全，推荐使用。

  HTTP/HTTPS方式详细请参考**使用HTTP/HTTPS转发**。
- AMQP方式
  - 订阅：AMQP（Advanced Message Queuing Protocol）即高级队列消息协议。用户通过控制台创建订阅任务，也可以通过调用物联网平台的**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**接口配置并激活规则，向平台获取发生变更的设备业务信息（如设备生命周期管理、设备数据上报、设备消息状态、设备状态等）和管理信息（软固件升级状态和升级结果）。订阅时必须指定具体的AMQP消息通道。
  - 推送：订阅成功后，物联网平台根据用户订阅的数据类型，将对应的变更信息推送给指定的AMQP消息队列。如果用户没有订阅该类型的数据通知，即使数据发生了变更也不会进行推送。用户可通过AMQP的客户端与IoT平台建立链接，来接收数据。

  AMQP详细请参考**使用AMQP转发**。
- MQTT方式
  - 订阅：用户可以通过调用物联网平台的**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**接口配置并激活规则，向平台获取发生变更的设备业务信息（如设备生命周期管理、设备数据上报、设备消息上报、设备状态等）和管理信息（软固件升级状态和升级结果）。订阅时必须指定接收推送消息的Topic。
  - 推送：订阅成功后，物联网平台根据用户订阅的数据类型，将对应的变更信息推送给指定的Topic。如果用户没有订阅该类型的数据通知，即使数据发生了变更也不会进行推送。用户可通过MQTT的客户端与IoT平台建立连接，来接收数据。

  MQTT详细请参考**使用MQTT转发**。
- 设备间通信
  - 订阅：物联网平台支持基于MQTT协议实现设备间的消息通信，用户可通过控制台创建规则，也可以通过调用物联网平台的**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**接口配置并激活规则，向平台获取设备上报的消息。设备订阅只支持消息上报。

- 推送：订阅成功后，物联网平台会将设备上报的消息推送到指定的MQTT Topic，当设备接入平台后，可以通过订阅该Topic来接收数据，从而实现设备间的消息通信。

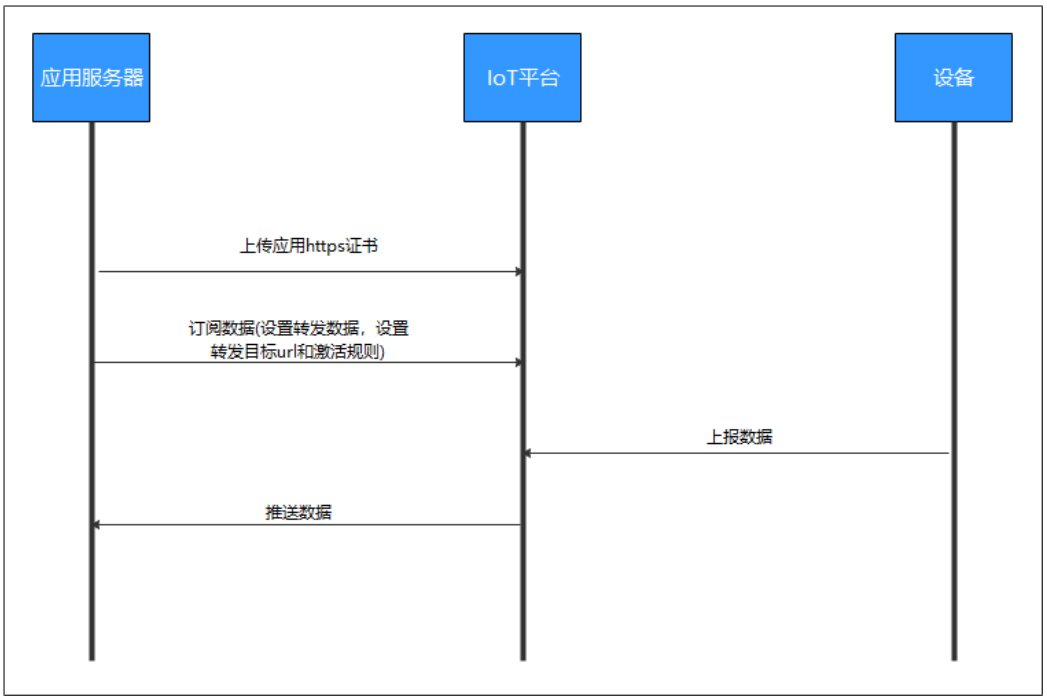
设备间通信详细请参考**设备间通信**。

数据转发方式	适用场景	优点	限制
HTTP/HTTPS订阅推送	应用作为服务端被动接收IoT云服务的消息。	-	流控限制800TPS，不建议大流量推送使用HTTP/HTTPS方式。
AMQP订阅推送	应用作为客户端，可主动拉取IoT云服务的消息，也可以通过监听被动接收 IoT云服务的消息。	能主动拉取数据	请参考**连接规格**。
MQTT订阅推送	应用作为客户端，可以通过订阅接收 IoT云服务的消息。	-	请参考**使用限制**。
设备间通信	• 智能家居控制场景，手机APP和智能设备之间进行消息通信。 • 设备联动，设备间进行数据传输与消息通信。	实现设备间通信	请参考**设备间消息通信概述**。

7.6.2 使用 HTTP/HTTPS 转发

概述

订阅推送的示意图如下图所示：



物联网平台采用HTTPS协议向应用服务器进行消息推送时，物联网平台需要校验应用服务器的真实性，需要在物联网平台上加载CA证书，该证书由应用服务器侧提供（调测时可自行**制作调测证书**，商用时建议更换为商用证书，否则会带来安全风险）。

推送机制：物联网平台向应用服务器推送消息后，如果应用服务器接收消息成功，会向物联网平台返回200 OK响应码。如果应用服务器无响应（或响应时间超过15秒），或者应用服务器向物联网平台返回非200响应码(如500、501、502、503、504等)，表示消息推送失败，消息推送失败后该消息将被丢弃。推送失败连续累计达到10次，物联网平台会将该订阅URL的主机地址加入黑名单，在黑名单期间消息将会积压在平台（默认积压最近24小时或1GB数据，若只想保留最新数据可参考**数据转发积压策略配置**进行配置）。此后每3分钟尝试对黑名单中的订阅URL主机地址进行消息推送，如果推送失败，则继续保持黑名单；如果推送成功，则解除黑名单。解除黑名单后消息将会以最大流控值推送完积压的消息后才会正常推送最新消息（默认流控为800TPS，自定义配置参考**数据转发流控策略配置**）。

如何进行数据订阅

应用服务器接入到“设备接入服务”后，在控制台创建订阅任务，也可以通过调用API接口进行数据订阅。

- 在控制台配置HTTP/HTTPS订阅请参考**配置HTTP/HTTPS服务端订阅**、**加载推送证书**。
- 通过API接口进行数据订阅请参考**如何调用API**和**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**。

推送数据格式

数据订阅成功后，物联网平台推送到应用侧的数据格式样例请参考**流转数据**。

说明

http协议消息头中，媒体类型信息Content-Type为application/json;字符集为utf-8。

加载推送证书

使用HTTPS协议推送，需要参考本小节完成推送证书的加载，然后参考**HTTP/HTTPS服务端订阅**在控制台创建订阅任务。

- 如果应用服务器取消了订阅后再重新订阅（URL不变），需要在物联网平台上重新上传CA证书。
- 如果应用服务器新增了订阅类型（新增URL），需要在物联网平台上加载与该URL对应的CA证书。即使新增URL使用的CA证书与原来推送的URL使用相同的证书，也需要重新上传CA证书。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 在左侧导航栏选择“规则 > 服务端证书”，单击右上角的“上传证书”，按照下表填写相关信息后，单击“确定”完成证书的加载。

参数名称	参数说明
证书名称	可自定义，用于区分不同证书

参数名称	参数说明
CA证书	需要提前申请和购买CA证书文件，CA证书由应用服务器侧提供。 **说明** 调测时可自行**制作调测证书**，商用时建议更换为商用证书，否则会带来安全风险。
域名/IP与端口	物联网平台推送消息到应用服务器的域名或IP地址与端口信息。填写为**创建规则动作**时URL中对应的域名或IP地址与端口信息。例如：推送URL为“https://www.example.com:8443/example/”，域名/IP与端口则为“www.example.com:8443”。

步骤3 单击左侧导航栏规则>服务端证书，选择对应证书，单击 可获取证书ID，用于后续**创建规则动作**时，作为参数使用。

----结束

制作调测证书

调测证书，又叫做自签名证书，用于客户端通过HTTPS访问服务端时进行安全认证。在物联网平台的使用中，可用于物联网平台向应用服务器采用HTTPS协议推送数据时，物联网平台认证应用服务器的合法性。本文以Windows环境为例，介绍通过Openssl工具制作调测证书的方法，生成的证书为PEM编码格式的证书，后缀为.cer。

常见的证书存储格式如下表所示。

存储格式	说明
DER	二进制编码，后缀名**.der/.cer/.crt**
PEM	BASE 64编码，后缀名**.pem/.cer/.crt**
JKS	Java的证书存储格式，后缀名**.jks**

说明

自签名证书仅用于调测阶段，在商用时，您需要向知名CA机构申请证书，否则可能会带来安全风险。

步骤1 在浏览器中访问**这里**，下载并安装OpenSSL工具。

步骤2 以管理员身份运行cmd命令行窗口。

步骤3 执行cd c:\openssl\bin（请替换为openssl实际安装路径），进入openssl命令视图。

步骤4 执行如下命令生成CA根证书私钥文件ca_private.key。

```
openssl genrsa -passout pass:123456 -aes256  -out ca_private.key 2048
```

- aes256：代表加密算法。
- passout pass：代表私钥密码。
- 2048：代表密钥长度。

步骤5 执行如下命令使用CA根证书私钥文件生成csr文件ca.csr，用于6生成CA根证书。

```
openssl req -passin pass:123456 -new -key ca_private.key -out ca.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=IoT/CN=CA"
```

如下信息您可以根据实际情况进行修改。

- C：代表国家，填写CN。
- ST：地区，如GD。
- L：城市，如SZ。
- O：组织，如Huawei。
- OU：组织单位，如IoT。
- CN：Common Name，填写为CA的组织名，如CA。

步骤6 执行如下命令生成CA根证书ca.cer。

```
openssl x509 -req -passin pass:123456 -in ca.csr -out ca.cer -signkey ca_private.key -CAcreateserial -days 3650
```

如下信息您可以根据实际情况进行修改。

- passin pass：必须与4中设置的私钥密码保持一致。
- days：代表证书有效期。

步骤7 执行如下命令生成应用服务器端私钥文件。

```
openssl genrsa -passout pass:123456 -aes256 -out server_private.key 2048
```

步骤8 执行如下命令生成应用服务器端csr文件，用于生成服务端证书。

```
openssl req -passin pass:123456 -new -key server_private.key -out server.csr -subj "/C=CN/ST=GD/L=SZ/O=Huawei/OU=IoT/CN=appserver.iot.com"
```

如下信息您可以根据实际情况进行修改。

- C：代表国家，填写CN。
- ST：地区，如GD。
- L：城市，如SZ。
- O：组织，如Huawei。
- OU：组织单位，如IoT。
- CN：Common Name，一般填写为应用服务器的域名或IP。

步骤9 通过CA私钥文件ca_private.key对服务端csr文件server.csr进行签名，生成服务端证书文件server.cer。

```
openssl x509 -req -passin pass:123456 -in server.csr -out server.cer -sha256 -CA ca.cer -CAkey ca_private.key -CAserial ca.srl -CAcreateserial -days 3650
```

步骤10 （可选）如果您需要.crt/.pem后缀的证书，可以根据如下命令进行转换。下面将以server.cer转为为server.crt为例进行说明，需要转换ca.cer证书时，请将命令中的server替换为ca。

```
openssl x509 -inform PEM -in server.cer -out server.crt
```

步骤11 在openssl安装目录的bin文件夹下，获取生成的CA证书（ca.cer/ca.crt/ca.pem）、应用服务器证书（server.cer/server.crt/server.pem）和私钥文件（server_private.key）。其中CA证书用于加载到物联网平台，应用服务器证书和私钥文件用于加载到应用服务器。

----结束

配置 HTTP/HTTPS 服务端订阅

本小节介绍如何在物联网平台配置HTTP/HTTPS服务端订阅。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则 > 数据转发”，单击页面左上角的“创建规则”。

步骤3 参考下表填写参数后，单击“创建规则”。

表 7-11 创建规则参数列表

参数名	参数说明
规则名称	创建的规则名称。
规则描述	对该规则的描述。
数据来源	• 设备：将操作设备的信息，如设备添加、设备删除、设备更新设置为数据来源。当数据来源选择“设备”时，不支持快速配置。 • 设备属性：将归属在某个资源空间下的设备上报给平台的属性值设置为数据来源。单击右侧的“快速配置”勾选需要转发的产品、属性、服务等数据。 • 设备消息：将归属在某个资源空间下的设备上报给平台的消息设置为转发目标。单击右侧的“快速配置”，仅转发指定Topic的数据。选择所属产品，填写Topic名称。您可以使用在产品详情页面**自定义的Topic**，也可以使用**平台预置的Topic**。 • 设备消息状态：将设备和平台之间流转的设备消息状态变更设置为转发目标。设备消息状态详见**这里**。当数据来源选择“设备消息状态”，不支持快速配置。 • 设备状态：将归属在某个资源空间下的直连或非直连设备状态变更转发至其他服务。单击“快速配置”，您可以转发设备状态为“在线”、“离线”和“异常”的设备信息到其他服务。物联网平台直连设备状态详见**这里**。 • 批量任务：将批量任务状态的数据设置为数据来源。当数据来源选择“批量任务”时，不支持快速配置。 • 产品：将操作产品的信息，如产品添加、产品删除、产品更新设置为数据来源。当数据来源选择“产品”时，不支持快速配置。 • 设备异步命令状态：针对LwM2M/CoAP协议的设备，物联网平台支持下发异步命令给设备。将异步命令的状态变更设置为数据来源。物联网平台设备异步命令状态详见**这里**。当数据来源选择“设备异步命令状态”时，不支持快速配置。 • 运行日志：将MQTT设备的业务运行日志设置为数据来源。当数据来源选择“运行日志”时，不支持快速配置。

参数名	参数说明
触发事件	选择数据来源后，对应修改触发事件。
资源空间	您可以选择单个资源空间或所有资源空间。当选择“所有资源空间”时，不支持快速配置。

步骤4 在设置转发目标页面，单击“添加”，在弹出的页面中参考下表配置完参数后，单击“确认”。

参数名	参数说明
转发目标	选择“第三方应用服务（HTTP推送）”。
推送URL	物联网平台推送消息到应用服务器的URL。例如，推送URL为“https://www.example.com:8443/example/”，则**加载推送证书**时“域名/IP与端口”为“www.example.com:8443”。 • 如果“推送URL”使用HTTP协议，不需要使用CA证书； • 如果“推送URL”使用HTTPS协议，需要上传CA证书，证书的上传可参考**加载推送证书**。
Token	3-32位长度英文或数字， 用于认证签名，平台推送数据到客户服务器时，将会使用Token进行签名并将签名信息组装到头域中进行推送。

步骤5 完成完整的规则定义后，单击“启动规则”，实现数据转发至HTTP/HTTPS消息队列。

----结束

HTTP/HTTPS 推送基于 Token 认证物联网平台

用户如果在添加转发目标到第三方应用服务（HTTP推送）时，已勾选“鉴权”并填写Token，物联网开发平台将在HTTP或HTTPS请求中头部增加如下字段：

参数	描述
timestamp	平台推送时的时间戳
nonce	平台生成的随机数
signature	结合token、timestamp、nonce组成的签名

签名规则：

1. 将token、timestamp、nonce进行字典排序。
2. 将排序后的字符串进行sha256加密。
3. 客户收到推送信息后，可以根据token以及头域中的timestamp和nonce按照该规则进行加密后与头域中的signature进行比较来确认是否是平台的消息。

校验signature的java示例如下：

1. 引入依赖，具体版本请根据实际业务需求来决定。

```
<dependency>
    <groupId>commons-codec</groupId>
    <artifactId>commons-codec</artifactId>
    <version>${commons.version}</version>
</dependency>
```

2. 从请求头中获取签名信息并使用commons-codec依赖包进行签名。

```
public boolean checkSignature(String nonce, String timestamp, String signature, String token) {
    List<String> list = new ArrayList<>();
    list.add(token);
    if (StringUtil.isNotEmpty(nonce)) {
        list.add(nonce);
    }
    if (StringUtil.isNotEmpty(timestamp)) {
        list.add(timestamp);
    }
    Collections.sort(list);
    StringBuilder signatureBuilder = new StringBuilder();
    for (String s : list) {
        signatureBuilder.append(s);
    }
    String serverSignature = DigestUtils.sha256Hex(signatureBuilder.toString());
    if (StringUtil.isNotEmpty(serverSignature) && serverSignature.equals(signature)) {
        return true;
    }
    return false;
}
```

3. 例如某次请求，用户设置的Token为aaaaaa，头域中收到如下参数：

```
nonce：8b9b796d388d49bba43adaa53aaf5bc4
timestamp: 1675654743514
signature： 2ff821fb8a976ede7d06434395ec8c25e4100bff8b3d12d8099ef7e30b58bd4c
```

排序后的字符串为：
16756547435148b9b796d388d49bba43adaa53aaf5bc4aaaaaa，sha256加密后：
2ff821fb8a976ede7d06434395ec8c25e4100bff8b3d12d8099ef7e30b58bd4c

⚠ 注意

token创建后， 每次修改转发目标时都需要重新填写token，否则token将不生效。

平台认证

作为服务端，应用侧如果需要认证IoT平台的身份，需要加载IoT平台的CA证书。请参考**资源**页面获取。

常见问题

订阅推送业务热点咨询问题如下，更多咨询问题请访问**查看更多**。

- **如何获取证书?**
- **调用订阅接口时，回调地址如何获取?**
- **回调地址可以使用域名吗?**
- **订阅后消息推送失败，例如提示503如何处理?**

- 为什么设备上报一条数据后应用服务器会收到多条推送?
- 调用订阅接口时，提示回调地址不合法
- 调用删除单个订阅接口时，subscriptionId如何获取
- 应用服务器只有内网地址，能够订阅成功?

相关 API 接口

创建规则动作

创建规则触发条件

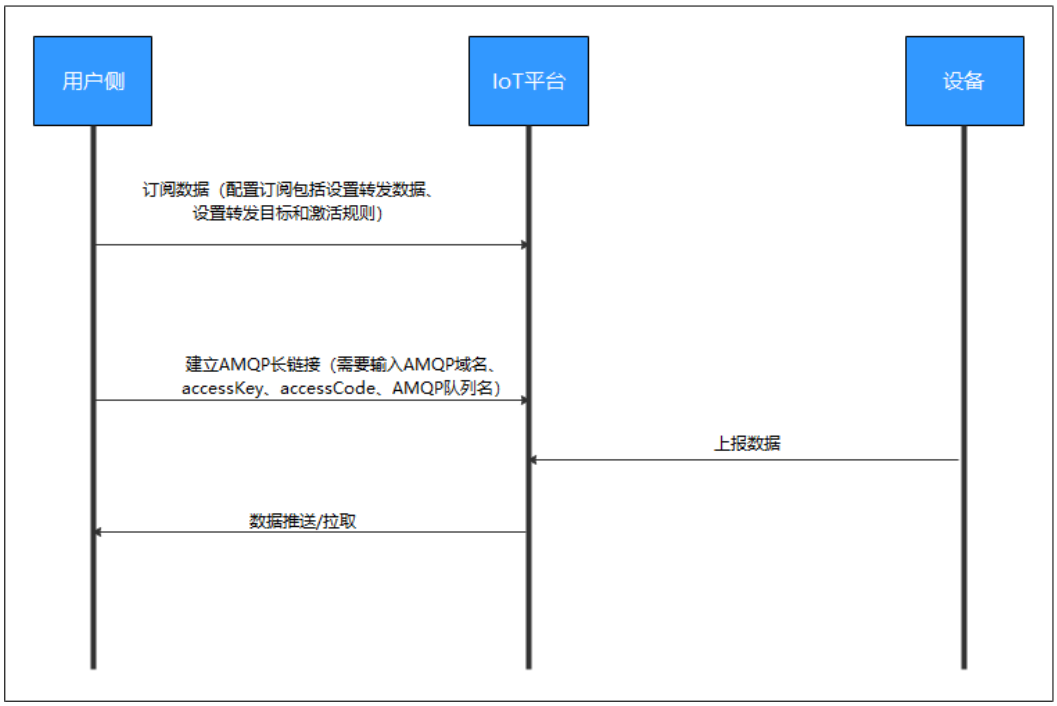
修改规则触发条件

流转数据

7.6.3 使用 AMQP 转发

7.6.3.1 AMQP 转发

订阅推送的示意图如下图所示：



推送机制：物联网平台向用户推送消息后，如果用户接收成功，会向物联网平台回复响应（推荐自动响应方式）。如果用户建立链接后不拉取数据，则会导致数据积压在服务端。服务端仅保存最近24小时，且占用磁盘容量小于1GB的数据，如果用户不及时拉取数据，物联网平台会滚动清除超期和超出容量限制的数据。若用户收到消息后来不及响应，长链接中断，则未响应的数据会在下次链接后重新推送。

如何进行数据订阅

应用服务器接入到“设备接入服务”后，在控制台创建订阅任务，也可以通过调用API接口进行数据订阅。

- 在控制台配置AMQP订阅请参考**配置AMQP服务端**。
- 通过API接口进行数据订阅请参考**如何调用API**和**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**。

推送数据格式

数据订阅成功后，物联网平台推送到应用侧的数据格式样例请参考**流转数据**。

说明

http协议消息头中，媒体类型信息Content-Type为application/json;字符集为utf-8。

相关 API 参考

创建规则动作

创建规则触发条件

修改规则触发条件

流转数据

创建AMQP队列

查询AMQP列表

查询单个AMQP队列

生成接入凭证

7.6.3.2 配置 AMQP 服务端

本文介绍如何在物联网平台设置和管理AMQP服务端订阅。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则 > 数据转发”，单击页面左侧的“创建规则”。

步骤3 参考下表填写参数后，单击“创建规则”。

表 7-12 创建规则参数列表

参数名	参数说明
规则名称	创建的规则名称。
规则描述	对该规则的描述。

参数名	参数说明
数据来源	• 设备：将操作设备的信息，如设备添加、设备删除、设备更新设置为数据来源。当数据来源选择“设备”时，不支持快速配置。 • 设备属性：将归属在某个资源空间下的设备上报给平台的属性值设置为数据来源。单击右侧的“快速配置”勾选需要转发的产品、属性、服务等数据。 • 设备消息：将归属在某个资源空间下的设备上报给平台的消息设置为转发目标。单击右侧的“快速配置”，仅转发指定Topic的数据。选择所属产品，填写Topic名称。您可以使用在产品详情页面**自定义的Topic**，也可以使用**平台预置的Topic**。 • 设备消息状态：将设备和平台之间流转的设备消息状态变更设置为转发目标。设备消息状态详见**这里**。当数据来源选择“设备消息状态”，不支持快速配置。 • 设备状态：将归属在某个资源空间下的直连或非直连设备状态变更转发至其他服务。单击“快速配置”，您可以转发设备状态为“在线”、“离线”和“异常”的设备信息到其他服务。物联网平台直连设备状态详见**这里**。 • 批量任务：将批量任务状态的数据设置为数据来源。当数据来源选择“批量任务”时，不支持快速配置。 • 产品：将操作产品的信息，如产品添加、产品删除、产品更新设置为数据来源。当数据来源选择“产品”时，不支持快速配置。 • 设备异步命令状态：针对LwM2M/CoAP协议的设备，物联网平台支持下发异步命令给设备。将异步命令的状态变更设置为数据来源。物联网平台设备异步命令状态详见**这里**。当数据来源选择“设备异步命令状态”时，不支持快速配置。 • 运行日志：将MQTT设备的业务运行日志设置为数据来源。当数据来源选择“运行日志”时，不支持快速配置。
触发事件	选择数据来源后，对应修改触发事件。
资源空间	您可以选择单个资源空间或所有资源空间。当选择“所有资源空间”时，不支持快速配置。

步骤4 在设置转发目标页面，单击“添加”，在弹出的页面中参考下表配置完参数后，单击“确定”。

参数名	参数说明
转发目标	选择“AMQP推送消息队列”
消息队列	单击“选择”，选择消息队列。 • 若没有消息队列，请新建消息队列，队列名称自定义且单个租户名下唯一，长度8-128，只能包含大写字母、小写字母、数字和指定特殊字符（如_-.:）。 • 若需要删除消息队列，单击消息队列右侧的“删除”即可。 **说明** 已经订阅的队列不允许删除。

步骤5 完成完整的规则定义后，单击“启动规则”，实现数据转发至AMQP消息队列。

----结束

7.6.3.3 AMQP 队列告警配置

用户在订阅AMQP队列进行消费时，由于网络通信问题、未及时对已经收到的消息进行确认等可能导致消费端离线、消息消费速度变慢，造成消息积压，影响消息实时性。

华为云物联网平台支持AMQP队列告警配置，用户可以通过设置告警规则来监控AMQP队列消息的积压以及队列消费速度的情况，规则触发后立即把告警信息推送给用户，方便用户及时发现故障恢复业务。本文介绍如何配置AMQP队列的告警规则。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“规则 > 数据转发”界面。

步骤3 选择"AMQP消息队列"页签，选择需要配置告警的队列名称，单击"详情"按钮，进入"AMQP队列详情"页面。

图 7-40 AMQP 消息队列

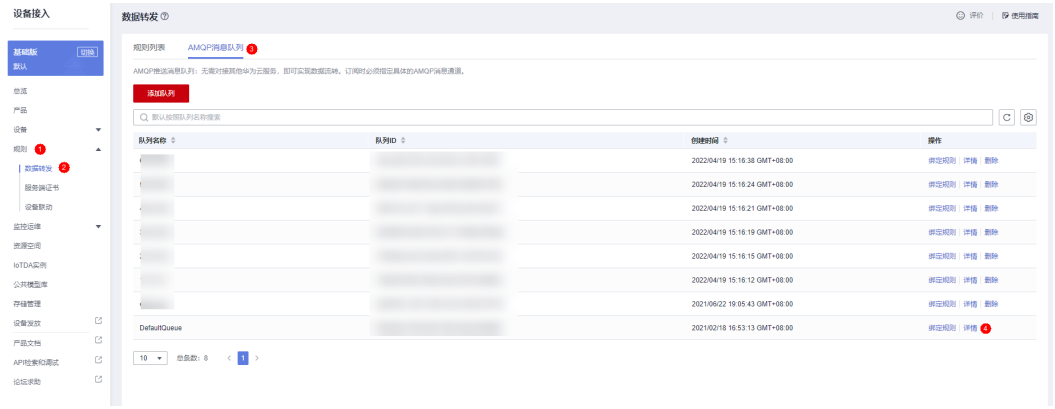
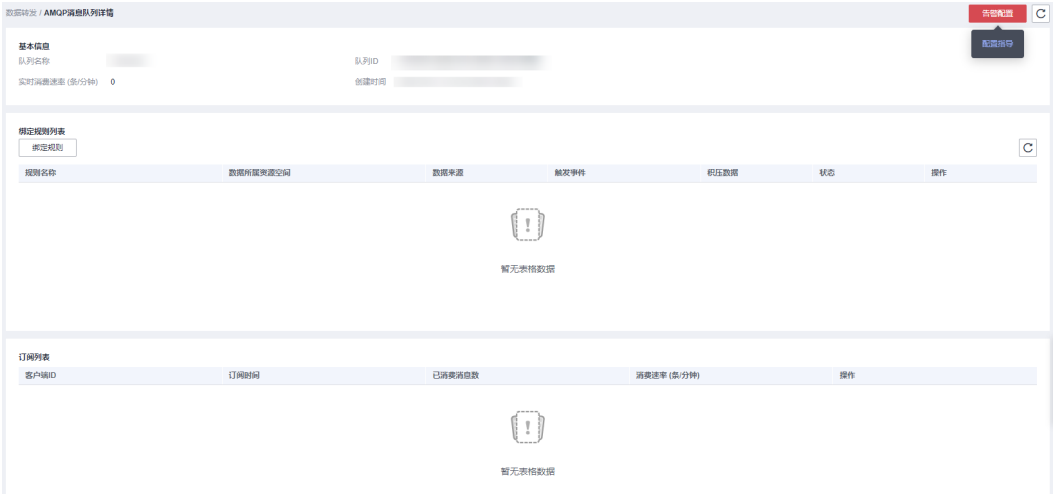
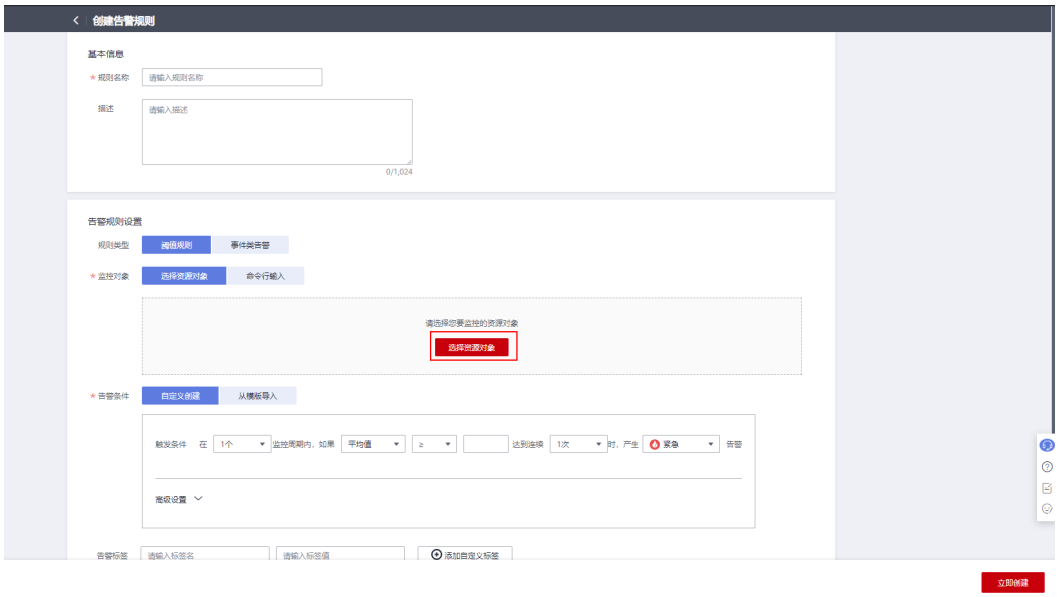


图 7-41 AMQP 队列详情



步骤4 单击"告警配置",跳转到创建告警规则页面。

图 7-42 创建告警规则



步骤5 单击"选择资源对象"按钮，弹出"选择监控对象"页面。在该页面选中"按指标纬度添加"后，根据表1和表2 说明选择合适的指标和指标纬度。

注意

"iotda_amqp_forwaring_backlog_message_count"和"iotda_amqp_forwaring_consume_rate"指标请在"全量指标"下查询。

图 7-43 选择监控对象

表 7-13 指标说明

指标名称	说明
iotda_amqp_forwaring_backlog_message_count	队列中消息堆积数

指标名称	说明
iotda_amqp_forwaring_consume_rate	队列消息消费速度

表 7-14 指标纬度说明

指标纬度	说明
clusterId	集群ID
namespace	命名空间，固定为：AOM.IoTDA
queueName	AMQP队列名称
userName	用户名称

步骤6 根据实际需要设置相应告警条件。

图 7-44 告警条件

步骤7 告警标签选择。如果您需要在"IoTDA>监控运维>当前告警"页面查看到该告警，您需要设置如下自定义标签。

表 7-15 自定义标签

标签名称	标签值
resource_provider	IoTDA

图 7-45 当前告警

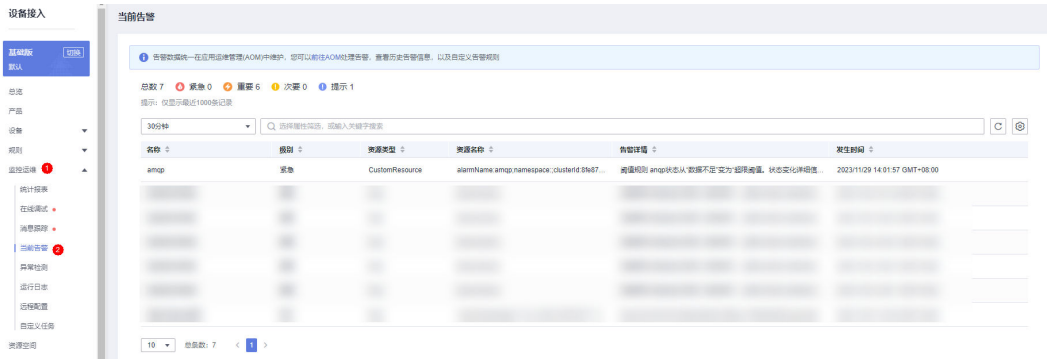


图 7-46 添加自定义标签

步骤8 告警通知可以指定相应的行动规则。当告警触发时，行动规则可通过主题将告警通过不同的通道(例如：邮件或短信)通知给主题的订阅者，详情请参考**创建告警行动规则**。

图 7-47 告警通知

步骤9 单击"立即创建"后完成告警规则配置。

----结束

7.6.3.4 AMQP 客户端接入说明

在调用**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**配置并激活规则后，您需要参考本文将AMQP客户端接入物联网平台，成功接入后，在您的服务端运行AMQP客户端，即可接收订阅的消息。

协议版本说明

AMQP协议标准的详细介绍，请参见**AMQP协议标准**。

华为云物联网平台服务端订阅仅支持AMQP 1.0版的协议标准。

建链认证过程

1. AMQP客户端与物联网平台建立TCP连接，然后进行TLS握手校验。

   说明

   为了保障安全，接收方必须使用TLS加密，且使用TLS1.2版本，不支持非加密的TCP传输。客户端的时间不能与标准时间差5min及以上，否则接入不进来。

2. 客户端请求建立连接。
3. 客户端向物联网平台发起请求，建立Receiver Link（即平台向客户端推送数据的单向通道）。客户端建立Connection成功后，需在15秒内完成Receiver Link的建立，否则物联网平台会关闭连接。建立Receiver Link后，客户端成功接入物联网平台。

说明

一个Connection上最多能够创建十个Receiver Link，不支持创建Sender Link，即只能由平台向客户端推送消息，客户端不能向平台发送消息。

连接配置说明

AMQP客户端接入物联网平台的连接地址和连接认证参数说明如下：

- AMQP接入域名：amqps://${UUCID}.iot-amqps.cn-north-4.myhuaweicloud.com。
- 连接字符串：amqps://${UUCID}.iot-amqps.cn-north-4.myhuaweicloud.com :5671?amqp.vhost=default&amqp.idleTimeout=8000&amqp.saslMechanisms=PLAIN

参数	说明
UUCID	独立域名ID（Unique User Connect ID），每个账号会自动生成，请前往**管理控制台**-总览页面-实例基本信息-接入信息获取。 
amqp.vhost	当前amqp使用是默认的host，只支持default。
amqp.saslMechanisms	连接认证方式当前支持PLAIN-SASL。
amqp.idleTimeout	心跳时间单位为毫秒。如果超过心跳时间，Connection上没有任何帧通信，物联网平台将关闭连接。

- 端口：5671
- 客户端身份认证参数

  username =“accessKey=${accessKey}|timestamp=${timestamp}|instanceId=${instanceId}”

  password =“${accessCode}”

参数	是否必须	说明
accessKey	是	接入凭证键值，当前单个键值最多能和32个客户端同时进行建链。 首次建链时候，请参考**这里**进行预置。
timestamp	是	表示当前时间，13位毫秒值时间戳。 服务端校验客户端的时间戳，且时间戳相差5分钟。

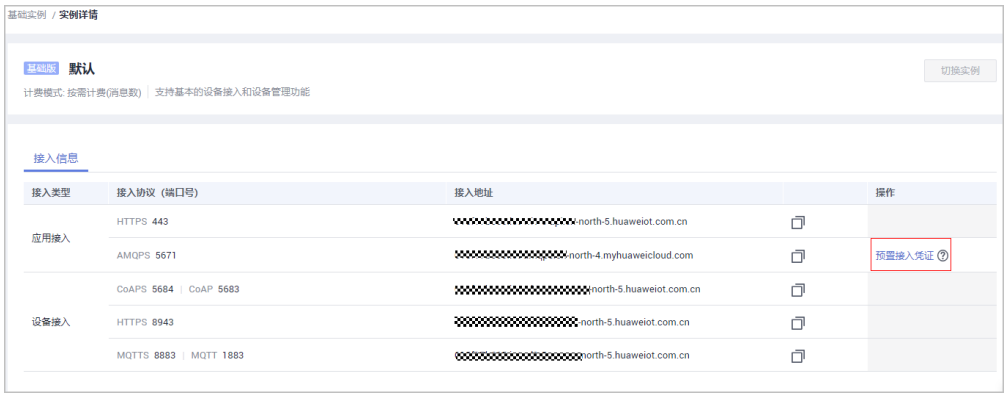
参数	是否必须	说明
instanceId	否	实例Id，同一Region购买多个标准版实例时需要填设置该参数，实例Id参考这里这里获取。
accessCode	是	接入凭证密钥，长度不超过256个。 首次建链时候，请参考这里进行预置。若密钥丢失，可通过调用接入凭证接口进行重置，也可以参考这里进行重置。

获取 AMQP 接入凭证

若应用使用AMQP协议接入物联网平台进行数据流转，请先预置接入凭证。您可以通过调用生成接入凭证接口预置，也可以前往控制台页面进行预置，详细方法请参考如下操作：

步骤1 选择“IoTDA实例 ”，进入您所选择的实例版本，单击“详情”进入实例详情页面。

步骤2 单击“预置接入凭证”预置接入凭证密钥（accessCode）和接入凭证键值（accessKey）。



说明

如果您之前预置过接入凭证，重新预置之后，之前的接入凭证密钥将不能再使用。

----结束

连接规格

Key	Documentation
一个连接能够订阅queue数量	10
单个用户最大队列数	100
单个租户最大连接数	32
单个消息缓存时间	1天

接收平台推送的消息

客户端和平台之间的Receiver Link建链成功后，基于这个link，支持客户端主动拉取数据（推荐使用，客户端可根据自身能力来拉取数据）和注册监听由服务端推送两种方式。

7.6.3.5 Java SDK 接入示例

本文介绍使用AMQP协议的JMS客户端接入华为云物联网平台，接收服务端订阅消息的示例。

开发环境要求

本示例使用的开发环境为JDK 1.8及以上版本。

获取 SDK

AMQP SDK为开源SDK。如果您使用Java开发语言，推荐使用Apache Qpid JMS客户端。请访问**Qpid JMS**下载客户端和查看使用说明。

添加 Maven 依赖

```
<!-- amqp 1.0 qpid client -->
 <dependency>
   <groupId>org.apache.qpid</groupId>
   <artifactId>qpid-jms-client</artifactId>
   <version>0.61.0</version>
 </dependency>
```

代码示例

您可以单击**这里**获取Java SDK接入示例，Demo中涉及的参数说明，请参考**AMQP客户端接入说明**。

⚠ 注意

所有示例代码已经包含与服务端断线重连的逻辑。

示例代码中用到的AmqpClient.java、AmqpClientOptions.java、AmqpConstants.java可以从**这里**获取。

1、创建AmqpClient。

```
// 以下参数请修改为自己的参数值
AmqpClientOptions options = AmqpClientOptions.builder()
                .host(AmqpConstants.HOST)
                .port(AmqpConstants.PORT)
                .accessKey(AmqpConstants.ACCESS_KEY)
                .accessCode(AmqpConstants.ACCESS_CODE)
                .queuePrefetch(1000) // sdk会在内存中分配该参数大小的队列，用来接收消息，客户端内存较小的情况可以调小该参数。
                .build();
        AmqpClient amqpClient = new AmqpClient(options);
        amqpClient.initialize();
```

2、通过设置listener消费amqp消息。

```
 try {
     MessageConsumer consumer = amqpClient.newConsumer(AmqpConstants.DEFAULT_QUEUE);
```

```
    consumer.setMessageListener(message -> {
        try {
            // 此处进行消息处理。如果处理比较耗时，最好进行开启新的线程处理，否则可能造成心跳超时链接断开。
            processMessage(message.getBody(String.class));
            // 如果options.isAutoAcknowledge==false,此处应该调用message.acknowledge();
        } catch (Exception e) {
            log.warn("message.getBody error,exception is ", e);
        }
    });
 } catch (Exception e) {
    log.warn("Consumer initialize error,", e);
 }
```

3、主动拉取amqp消息

```
 // 创建一个线程池用来拉取消息
 ExecutorService executorService = new ThreadPoolExecutor(1, 1, 60, TimeUnit.SECONDS,new LinkedBlockingQueue<>(1));

     try {
         MessageConsumer consumer = amqpClient.newConsumer(AmqpConstants.DEFAULT_QUEUE);
         executorService.execute(() -> {
             while (!isClose.get()) {
                 try {
                     Message message = consumer.receive();
                     // 此处进行消息处理。如果处理比较耗时，最好进行开启新的线程处理，否则可能造成心跳超时链接断开。
                     processMessage(message.getBody(String.class));
                     // 如果options.isAutoAcknowledge==false,此处应该调用message.acknowledge();
                 } catch (JMSException e) {
                     log.warn("receive message error,", e);
                 }
             }
         });
      } catch (Exception e) {
         log.warn("Consumer initialize error,", e);
      }
```

4、更多消费amqp消息的demo，请参考Java SDK接入示例工程。

资源

AmqpClient.java

```
package com.iot.amqp;

import lombok.extern.slf4j.Slf4j;
import org.apache.commons.lang3.StringUtils;
import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionExtensions;
import org.apache.qpid.jms.JmsConnectionFactory;
import org.apache.qpid.jms.JmsQueue;
import org.apache.qpid.jms.transports.TransportOptions;
import org.apache.qpid.jms.transports.TransportSupport;

import javax.jms.Connection;
import javax.jms.JMSException;
import javax.jms.MessageConsumer;
import javax.jms.Session;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

@Slf4j
public class AmqpClient {
    private final AmqpClientOptions options;
    private Connection connection;
    private Session session;
```

```
    private final Set<MessageConsumer> consumerSet = Collections.synchronizedSet(new HashSet<>());

    public AmqpClient(AmqpClientOptions options) {
        this.options = options;
    }

    public String getId() {
        return options.getClientId();
    }

    public void initialize() throws Exception {
        String connectionUrl = options.generateConnectUrl();
        log.info("connectionUrl={}", connectionUrl);
        JmsConnectionFactory cf = new JmsConnectionFactory(connectionUrl);
        // 信任服务端
        TransportOptions to = new TransportOptions();
        to.setTrustAll(true);
        cf.setSslContext(TransportSupport.createJdkSslContext(to));
        String userName = "accessKey=" + options.getAccessKey();
        cf.setExtension(JmsConnectionExtensions.USERNAME_OVERRIDE.toString(), (connection, uri) -> {
            String newUserName = userName;
            if (connection instanceof JmsConnection) {
                newUserName = ((JmsConnection) connection).getUsername();
            }
            if (StringUtils.isEmpty(options.getInstanceId())) {
            // IoTDA的userName组成格式如下：“accessKey=${accessKey}|timestamp=${timestamp}”
                return newUserName + "|timestamp=" + System.currentTimeMillis();
            } else {
            // 同一region购买多个标准版时userName组成格式为“accessKey=${accessKey}|timestamp=$
{timestamp}|instanceId=${instanceId}”
                return newUserName + "|timestamp=" + System.currentTimeMillis() + "|instanceId=" +
options.getInstanceId();
            }
        });
        // 创建连接
        connection = cf.createConnection(userName, options.getAccessCode());
        // 创建 Session, Session.CLIENT_ACKNOWLEDGE: 收到消息后，需要手动调用message.acknowledge()。
Session.AUTO_ACKNOWLEDGE: SDK自动ACK（推荐）。
        session = connection.createSession(false, options.isAutoAcknowledge() ?
Session.AUTO_ACKNOWLEDGE : Session.CLIENT_ACKNOWLEDGE);
        connection.start();
    }

    public MessageConsumer newConsumer(String queueName) throws Exception {
        if (connection == null || !(connection instanceof JmsConnection) || ((JmsConnection)
connection).isClosed()) {
            throw new Exception("create consumer failed,the connection is disconnected.");
        }
        MessageConsumer consumer;

        consumer = session.createConsumer(new JmsQueue(queueName));
        if (consumer != null) {
            consumerSet.add(consumer);
        }
        return consumer;
    }

    public void close() {
        consumerSet.forEach(consumer -> {
            try {
                consumer.close();
            } catch (JMSException e) {
                log.warn("consumer close error,exception is ", e);
            }
        });

        if (session != null) {
            try {
                session.close();
```

```
            } catch (JMSException e) {
                log.warn("session close error,exception is ", e);
            }
        }

        if (connection != null) {
            try {
                connection.close();
            } catch (JMSException e) {
                log.warn("connection close error,exception is", e);
            }
        }
    }
}
```

AmqpClientOptions.java

```
package com.iot.amqp;

import lombok.Builder;
import lombok.Data;
import org.apache.commons.lang3.StringUtils;

import java.text.MessageFormat;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
import java.util.stream.Collectors;

@Data
@Builder
public class AmqpClientOptions {
    private String host;
    @Builder.Default
    private int port = 5671;
    private String accessKey;
    private String accessCode;
    private String clientId;

    /**
     * 实例Id信息，同一个Region购买多个标准版实例时需设置
     */
    private String instanceId;

    /**
     * 仅支持true
     */
    @Builder.Default
    private boolean useSsl = true;

    /**
     * IoTDA仅支持default
     */
    @Builder.Default
    private String vhost = "default";

    /**
     * IoTDA仅支持PLAIN
     */
    @Builder.Default
    private String saslMechanisms = "PLAIN";

    /**
     * true: SDK自动ACK（默认）
     * false:收到消息后，需要手动调用message.acknowledge()
     */
    @Builder.Default
    private boolean isAutoAcknowledge = true;
```

```
    /**
     * 重连时延（ms）
     */
    @Builder.Default
    private long reconnectDelay = 3000L;

    /**
     * 最大重连时延（ms）,随着重连次数增加重连时延逐渐增加
     */
    @Builder.Default
    private long maxReconnectDelay = 30 * 1000L;

    /**
     * 最大重连次数,默认值-1，代表没有限制
     */
    @Builder.Default
    private long maxReconnectAttempts = -1;

    /**
     * 空闲超时，对端在这个时间段内没有发送AMQP帧则会导致连接断开。默认值为30000。单位：毫秒。
     */
    @Builder.Default
    private long idleTimeout = 30 * 1000L;

    /**
     * The values below control how many messages the remote peer can send to the client and be held in a
pre-fetch buffer for each consumer instance.
     */
    @Builder.Default
    private int queuePrefetch = 1000;

    /**
     * 扩展参数
     */
    private Map<String, String> extendedOptions;

    public String generateConnectUrl() {
        String uri = MessageFormat.format("{0}://{1}:{2}", (useSsl ? "amqps" : "amqp"), host,
String.valueOf(port));
        Map<String, String> uriOptions = new HashMap<>();
        uriOptions.put("amqp.vhost", vhost);
        uriOptions.put("amqp.idleTimeout", String.valueOf(idleTimeout));
        uriOptions.put("amqp.saslMechanisms", saslMechanisms);

        Map<String, String> jmsOptions = new HashMap<>();
        jmsOptions.put("jms.prefetchPolicy.queuePrefetch", String.valueOf(queuePrefetch));
        if (StringUtils.isNotEmpty(clientId)) {
            jmsOptions.put("jms.clientID", clientId);
        } else {
            jmsOptions.put("jms.clientID", UUID.randomUUID().toString());
        }
        jmsOptions.put("failover.reconnectDelay", String.valueOf(reconnectDelay));
        jmsOptions.put("failover.maxReconnectDelay", String.valueOf(maxReconnectDelay));
        if (maxReconnectAttempts > 0) {
            jmsOptions.put("failover.maxReconnectAttempts", String.valueOf(maxReconnectAttempts));
        }
        if (extendedOptions != null) {
            for (Map.Entry<String, String> option : extendedOptions.entrySet()) {
                if (option.getKey().startsWith("amqp.") || option.getKey().startsWith("transport.")) {
                    uriOptions.put(option.getKey(), option.getValue());
                } else {
                    jmsOptions.put(option.getKey(), option.getValue());
                }
            }
        }
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.append(uriOptions.entrySet().stream()
                .map(option -> MessageFormat.format("{0}={1}", option.getKey(), option.getValue()))
                .collect(Collectors.joining("&", "failover:(" + uri + "?", ")")));
```

```
        stringBuilder.append(jmsOptions.entrySet().stream()
                .map(option -> MessageFormat.format("{0}={1}", option.getKey(), option.getValue()))
                .collect(Collectors.joining("&", "?", "")));
        return stringBuilder.toString();
    }
}
```

AmqpConstants.java

```
package com.iot.amqp;


public interface AmqpConstants {
    /**
     * AMQP接入域名
     * eg: "****.iot-amqps.cn-north-4.myhuaweicloud.com";
     */
    String HOST = "127.0.0.1";

    /**
     * AMQP接入端口
     */
    int PORT = 5671;

    /**
     * 接入凭证键值
     * 不需要拼接时间戳timestamp
     */
    String ACCESS_KEY = "accessKey";

    /**
     * 接入凭证密钥
     */
    String ACCESS_CODE = "accessCode";

    /**
     * 默认队列
     */
    String DEFAULT_QUEUE = "DefaultQueue";
}
```

7.6.3.6 Node.js SDK 接入示例

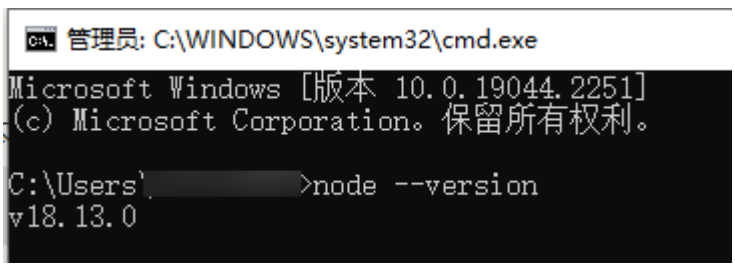
本文介绍使用Node.js语言的AMQP SDK接入华为云物联网平台，接收服务端订阅消息的示例。

开发环境

本示例所使用的开发环境为Node.js 8.0.0及以上版本。请前往Node.js官网**下载**。安装成功之后可以通过以下命令查看node版本。

```
node --version
```

如果能够查询到node版本，且版本高于8.0.0则代表安装成功。



代码示例

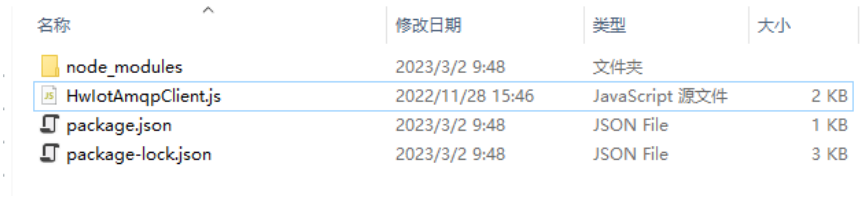
1. 在本地计算机创建一个JavaScript文件(例：HwIotAmqpClient.js)，保存以下示例代码到文件中。参考**AMQP客户端接入说明**修改相关连接参数。

```
const container = require('rhea');
//获取当前时间戳
var timestamp = Math.round(new Date());

//建立连接。
var connection = container.connect({
    //接入域名，请参考这里。
    'host': '${UUCID}.iot-amqps.cn-north-4.myhuaweicloud.com',
    'port': 5671,
    'transport': 'tls',
    'reconnect': true,
    'idle_time_out': 8000,
    //userName组装方法，请参考这里。
     'username': 'accessKey=${yourAccessKey}|timestamp=' + timestamp + '|instanceId=${instanceId}',
    //accessCode，请参考这里。
    'password': '${yourAccessCode}',
    'saslMechannisms': 'PLAIN',
    'rejectUnauthorized': false,
    'hostname': 'default',
});

//创建Receiver连接。 队列名，可以使用默认队列DefaultQueue
var receiver = connection.open_receiver('${yourQueue}');

//接收云端推送消息的回调函数。
container.on('message', function (context) {
    var msg = context.message;
    var content = msg.body;
    console.log(content);
    //发送ACK，注意不要在回调函数有耗时逻辑。
    context.delivery.accept();
 });
```

2. 进入上一步创建的文件（HwIotAmqpClient.js）所在目录，执行以下命令安装rhea库。

```
npm install rhea
```

安装完成后，项目文件如下图所示。



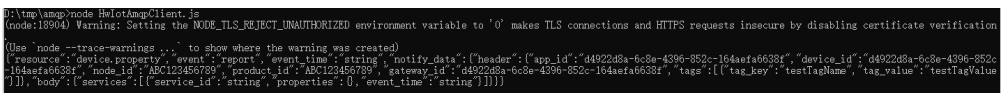
3. 通过命令启动AMQP客户端，命令如下。

```
node HwIotAmqpClient.js
```

4. 运行结果示例

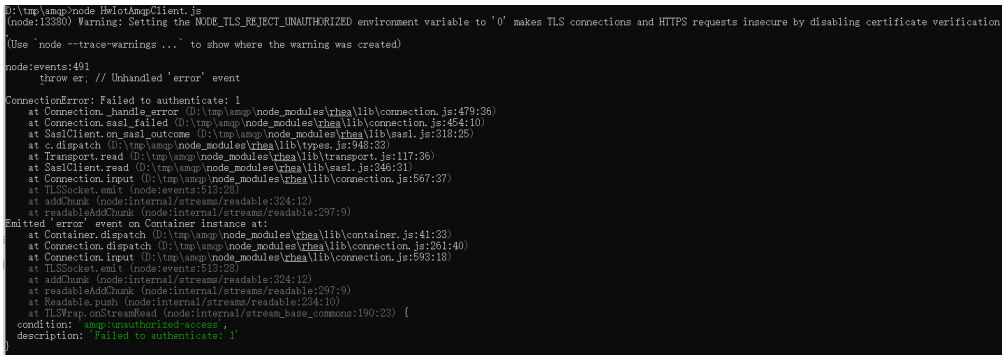
- 订阅成功：

  显示如下日志代表AMQP客户端订阅成功，并成功获取到物联网平台数据。



- 订阅失败

  以下日志代表AMQP客户端在物联网平台鉴权失败，请确认接入码是否正确，时间戳与标准时间的差异是否大于5min。



7.6.3.7 C# SDK 接入示例

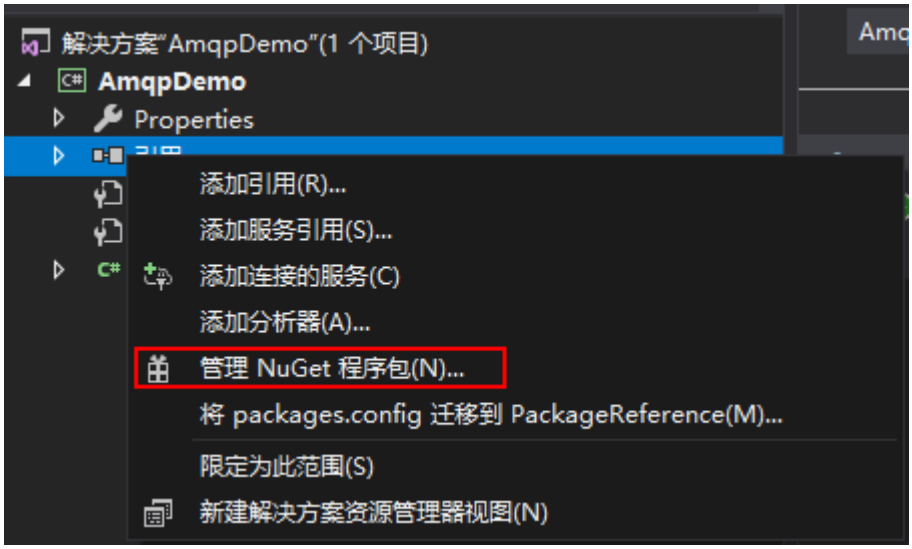
本文介绍使AMQPNetLite客户端接入华为云物联网平台，接收服务端订阅消息的示例。

开发环境要求

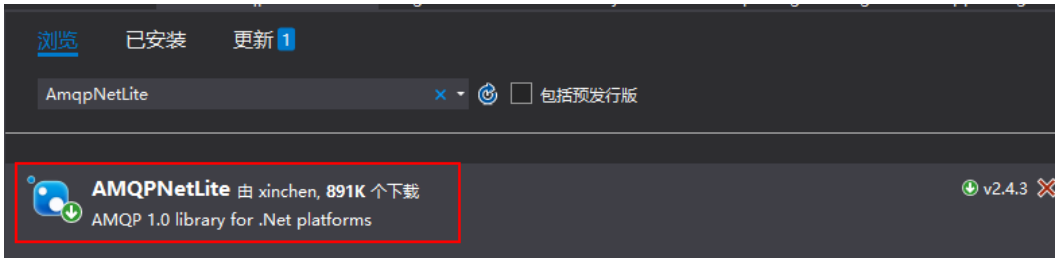
本示例使用的开发环境为.NETFramework V4.6及以上版本。

获取 SDK

1、在工程目录上单击鼠标右键打开"管理NuGet程序包"



2、在"NuGet管理器"中搜索到"AmqpNetLite"后安装所需版本



代码示例

Demo中涉及的参数说明，请参考**AMQP客户端接入说明**。

```
using Amqp;
using Amqp.Framing;
using Amqp.Sasl;
using System;
using System.Threading;

namespace AmqpDemo
{
    class Program
    {
        /// <summary>
        /// 接入域名，请参见AMQP客户端接入说明文档
        /// 请参考连接配置说明
        /// </summary>
        static string Host = "${Host}";

        /// <summary>
        /// 端口
        /// </summary>
        static int Port = 5671;

        /// <summary>
        /// 接入凭证键值
        /// </summary>
        static string AccessKey = "${YourAccessKey}";

        /// <summary>
        /// 接入凭证密钥
        /// </summary>
        static string AccessCode = "${yourAccessCode}";

        /// <summary>
        /// 实例Id信息，同一个Region购买多个标准版实例时需设置
        /// </summary>
        static string InstanceId = "${instanceId}";

        /// <summary>
        /// 队列名
        /// </summary>
        static string QueueName = "${yourQueue}";

        static Connection connection;

        static Session session;

        static ReceiverLink receiverLink;

        static DateTime lastConnectTime = DateTime.Now;

        static void Main(string[] args)
        {
            try
            {
                connection = CreateConnection();
                // 添加Connection Exception回调
                connection.AddClosedCallback(ConnectionClosed);

                // 创建Session。
                var session = new Session(connection);

                // 创建ReceiverLink
                receiverLink = new ReceiverLink(session, "receiverName", QueueName);

                //接收消息。
                ReceiveMessage(receiverLink);
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine(e);
            }

            // 按下"Enter"键后程序退出
            Console.ReadLine();

            ShutDown();
        }

        /// <summary>
        /// 创建Connection
        /// </summary>
        /// <returns>Connection</returns>
        static Connection CreateConnection()
        {
            lastConnectTime = DateTime.Now;
            long timestamp = new DateTimeOffset(DateTime.UtcNow).ToUnixTimeMilliseconds();
            string userName = "accessKey=" + AccessKey + "|timestamp=" + timestamp + "|instanceId=" +
InstanceId;
            Address address = new Address(Host, Port, userName, AccessCode);
            ConnectionFactory factory = new ConnectionFactory();
            factory.SASL.Profile = SaslProfile.External;
            // 信任服务端,跳过证书校验
            factory.SSL.RemoteCertificateValidationCallback = (sender, certificate, chain, sslPolicyError) =>
{ return true; };
            factory.AMQP.IdleTimeout = 8000;
            factory.AMQP.MaxFrameSize = 8 * 1024;
            factory.AMQP.HostName = "default";
            var connection = factory.CreateAsync(address).Result;
            return connection;
        }

        static void ReceiveMessage(ReceiverLink receiver)
        {
            receiver.Start(20, (link, message) =>
            {
                // 在线程池中处理消息，防止阻塞拉取消息的线程
                ThreadPool.QueueUserWorkItem((obj) => ProcessMessage(obj), message);
                // 回ACK
                link.Accept(message);
            });
        }


        static void ProcessMessage(Object obj)
        {
            if (obj is Message message)
            {
                string body = message.Body.ToString();
                Console.WriteLine("receive message, body=" + body);
            }
        }


        static void ConnectionClosed(IAmqpObject amqpObject, Error e)
        {
            // 断线重连
            ThreadPool.QueueUserWorkItem((obj) =>
            {
                ShutDown();
                int times = 0;
                while (times++ < 5)
                {
                    try
                    {
                        Thread.Sleep(1000);
                        connection = CreateConnection();
```

```
                // 添加Connection Exception回调
                connection.AddClosedCallback(ConnectionClosed);

                // 创建Session。
                session = new Session(connection);

                // 创建ReceiverLink
                receiverLink = new ReceiverLink(session, "receiverName", QueueName);

                //接收消息。
                ReceiveMessage(receiverLink);
                break;
            }
            catch (Exception exception)
            {
                Console.WriteLine("reconnect error, exception =" + exception);
            }
        }
    });
}

static void ShutDown()
{
    if (receiverLink != null)
    {
        try
        {
            receiverLink.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("close receiverLink error, exception =" + e);
        }

    }
    if (session != null)
    {
        try
        {
            session.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("close session error, exception =" + e);
        }

    }

    if (connection != null)
    {
        try
        {
            connection.Close();
        }
        catch (Exception e)
        {
            Console.WriteLine("close connection error, exception =" + e);
        }

    }
  }
}
```

7.6.3.8 Python SDK 接入示例

本文介绍使用Python3 SDK通过AMQP接入华为云物联网平台，接收服务端订阅消息的示例。

开发环境

Python 3.0及更高版本。本示例使用了Python 3.9版本。

下载 SDK

本示例使用的Python语言的AMQP SDK为**python-qpid-proton**(本示例使用版本为0.37.0),可以通过以下命令安装最新版本SDK。

```
pip install python-qpid-proton
```

也可以参考（**Installing Qpid Proton**）手动安装。

代码示例

```
import threading
import time

from proton import SSLDomain
from proton.handlers import MessagingHandler
from proton.reactor import Container

# 重连次数
reconnectTimes = 0


def current_time_millis():
    return str(int(round(time.time() * 1000)))


class AmqpClient(MessagingHandler):
    def __init__(self, host, port, accessKey, accessCode, queueName, instanceId):
        super(AmqpClient, self).__init__()
        self.host = host
        self.port = port
        self.accessKey = accessKey
        self.accessCode = accessCode
        self.queueName = queueName
        self.instanceId = instanceId

    def on_start(self, event):
        # 接入域名，请参见AMQP客户端接入说明文档。
        url = "amqps://%s:%s" % (self.host, self.port)

        timestamp = current_time_millis()
        userName = "accessKey=" + self.accessKey + "|timestamp=" + timestamp + "|instanceId=" +
self.instanceId
        passWord = self.accessCode
        # 默认不校验服务端证书
        sslDomain = SSLDomain(SSLDomain.MODE_CLIENT)
        sslDomain.set_peer_authentication(SSLDomain.ANONYMOUS_PEER)
        self.conn = event.container.connect(url, user=userName, password=passWord, heartbeat=60,
ssl_domain=sslDomain,
                                            reconnect=False)
        event.container.create_receiver(self.conn, source=self.queueName)

    # 当连接成功建立时被调用。
    def on_connection_opened(self, event):
        global reconnectTimes
        reconnectTimes = 0
        print("Connection established, remoteUrl: %s", event.connection.hostname)

    # 当连接关闭时被调用。
    def on_connection_closed(self, event):
        print("Connection closed: %s", self)
        ReconnectThread("reconnectThread").start()
```

```
    # 当远端因错误而关闭连接时被调用。
    def on_connection_error(self, event):
        print("Connection error:%s", self)
        ReconnectThread("reconnectThread").start()

    # 当建立AMQP连接错误时被调用，包括身份验证错误和套接字错误。
    def on_transport_error(self, event):
        if event.transport.condition:
            if event.transport.condition.info:
                print("%s: %s: %s" % (event.transport.condition.name, event.transport.condition.description,
                                      event.transport.condition.info))
            else:
                print("%s: %s" % (event.transport.condition.name, event.transport.condition.description))
        else:
            print("Unspecified transport error")
        ReconnectThread("reconnectThread").start()

    # 当收到消息时被调用。
    def on_message(self, event):
        message = event.message
        content = message.body
        print("receive message: content=%s" % content)


class ReconnectThread(threading.Thread):
    def __init__(self, name):
        threading.Thread.__init__(self)
        self.name = name

    def run(self):
        global reconnectTimes
        reconnectTimes = reconnectTimes + 1
        time.sleep(15 if reconnectTimes > 15 else reconnectTimes)
        Container(AmqpClient(amqpHost, amqpPort, amqpAccessKey, amqpAccessCode, amqpQueueName,
instanceId)).run()


# 以下参数配置请参考连接配置说明
# AMQP接入域名
amqpHost = "127.0.0.1"

# AMQP接入端口
amqpPort = 5671

# 接入凭证键值
amqpAccessKey = 'your AccessKey'

# 接入凭证密钥
amqpAccessCode = 'your AccessCode'

# 订阅队列名称
amqpQueueName = 'DefaultQueue'

# 实例Id，同一Region购买多个标准版实例时需要填设置该参数。
instanceId = ''

Container(AmqpClient(amqpHost, amqpPort, amqpAccessKey, amqpAccessCode, amqpQueueName,
instanceId)).run()
```

7.6.3.9 GO SDK 接入示例

本文介绍使用GO SDK通过AMQP接入华为云物联网平台，接收服务端订阅消息的示例。

开发环境要求

本示例使用的开发环境为Go 1.16及以上版本。

添加依赖

在go.mod中添加以下依赖。

```
require (
    pack.ag/amqp v0.12.5 // 本示例使用v0.12.5,根据实际需要选择版本
)
```

代码示例

```
package main

import (
    "context"
    "crypto/tls"
    "fmt"
    "pack.ag/amqp"
    "time"
)

type AmqpClient struct {
    Title       string
    Host        string
    AccessKey   string
    AccessCode string
        InstanceId string
    QueueName   string

    address  string
    userName string
    password string

    client   *amqp.Client
    session  *amqp.Session
    receiver *amqp.Receiver
}

type MessageHandler interface {
    Handle(message *amqp.Message)
}

func (ac *AmqpClient) InitConnect() {
    if ac.QueueName == "" {
        ac.QueueName = "DefaultQueue"
    }
    ac.address = "amqps://" + ac.Host + ":5671"
    ac.userName = fmt.Sprintf("accessKey=%s|timestamp=%d|instanceId=%s", ac.AccessKey,
time.Now().UnixNano()/1000000, ac.InstanceId)
    ac.password = ac.AccessCode
}

func (ac *AmqpClient) StartReceiveMessage(ctx context.Context, handler MessageHandler) {
    childCtx, _ := context.WithCancel(ctx)
    err := ac.generateReceiverWithRetry(childCtx)
    if nil != err {
        return
    }
    defer func() {
        _ = ac.receiver.Close(childCtx)
        _ = ac.session.Close(childCtx)
        _ = ac.client.Close()
    }()

    for {
        // 阻塞接受消息，如果ctx是background则不会被打断。
        message, err := ac.receiver.Receive(ctx)
        if nil == err {
            go handler.Handle(message)
            _ = message.Accept()
```

```
        } else {
            fmt.Println("amqp receive data error: ", err)

            //如果是主动取消，则退出程序。
            select {
            case <-childCtx.Done():
                return
            default:
            }

            //非主动取消，则重新建立连接。
            err := ac.generateReceiverWithRetry(childCtx)
            if nil != err {
                return
            }
        }
    }
}

func (ac *AmqpClient) generateReceiverWithRetry(ctx context.Context) error {
    // 退避重试，从10ms依次x2，直到20s。
    duration := 10 * time.Millisecond
    maxDuration := 20000 * time.Millisecond
    times := 1

    // 异常情况，退避重连。
    for {
        select {
        case <-ctx.Done():
            return amqp.ErrConnClosed
        default:
        }

        err := ac.generateReceiver()
        if nil != err {
            fmt.Println("amqp ac.generateReceiver error ", err)
            time.Sleep(duration)
            if duration < maxDuration {
                duration *= 2
            }
            fmt.Println("amqp connect retry,times:", times, ",duration:", duration)
            times++
            return nil
        } else {
            fmt.Println("amqp connect init success")
            return nil
        }
    }
}

// 由于包不可见，无法判断conn和session状态，重启连接获取。
func (ac *AmqpClient) generateReceiver() error {

    if ac.session != nil {
        receiver, err := ac.session.NewReceiver(
            amqp.LinkSourceAddress(ac.QueueName),
            amqp.LinkCredit(20),
        )
        // 如果断网等行为发生，conn会关闭导致session建立失败，未关闭连接则建立成功。
        if err == nil {
            ac.receiver = receiver
            return nil
        }
    }

    // 清理上一个连接。
    if ac.client != nil {
        _ = ac.client.Close()
    }
```

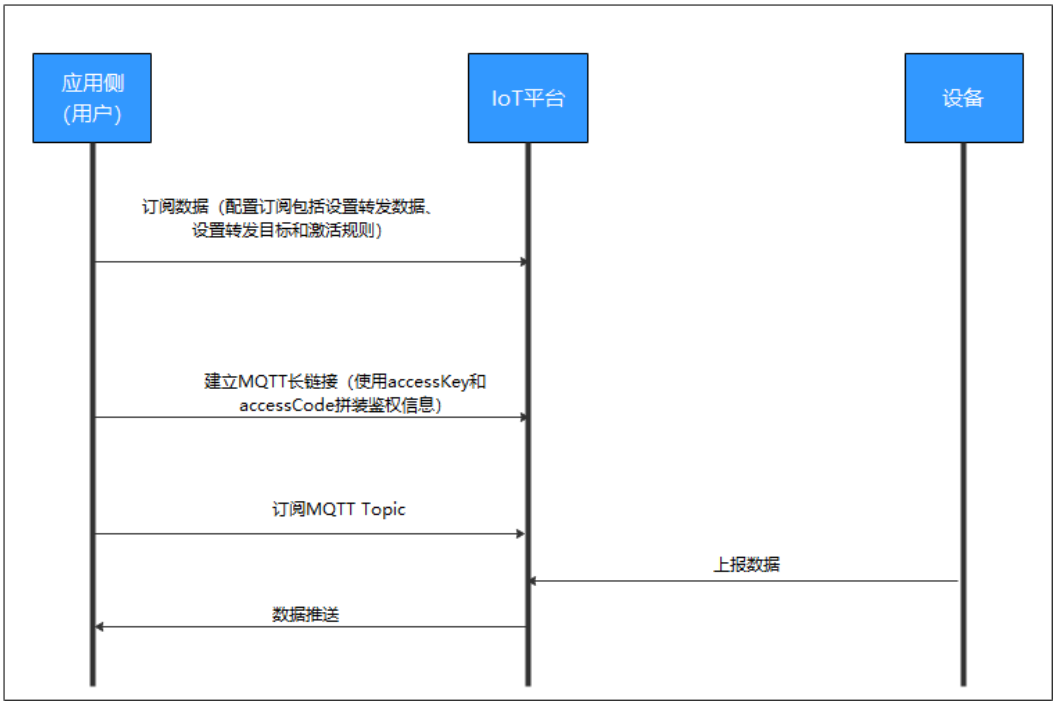
```
        ac.userName = fmt.Sprintf("accessKey=%s|timestamp=%d|instanceId=%s", ac.AccessKey,
time.Now().UnixNano()/1000000, ac.InstanceId)
    fmt.Println("[" + ac.Title + "] Dial... addr=[" + ac.address + "], username=[" + ac.userName + "],
password=[" + ac.password + "]")
    client, err := amqp.Dial(ac.address,
        amqp.ConnSASLPlain(ac.userName, ac.password),
        amqp.ConnProperty("vhost", "default"),
        amqp.ConnServerHostname("default"),
        amqp.ConnTLSConfig(&tls.Config{InsecureSkipVerify: true,
            MaxVersion: tls.VersionTLS12,
        }),
        amqp.ConnConnectTimeout(8*time.Second))
    if err != nil {
        fmt.Println("Dial", err)
        return err
    }
    ac.client = client
    session, err := client.NewSession()
    if err != nil {
        XDebug("Error:  NewSession", err)
        return err
    }
    ac.session = session

    receiver, err := ac.session.NewReceiver(
        amqp.LinkTargetDurability(amqp.DurabilityUnsettledState),
        amqp.LinkSourceAddress(ac.QueueName),
        amqp.LinkCredit(100),
    )
    if err != nil {
        XDebug("Error:  NewReceiver", err)
        return err
    }
    ac.receiver = receiver

    return nil
}

func XDebug(s string, err error) {
    fmt.Println(s, err)
}

type CustomerMessageHandler struct {
}

func (c *CustomerMessageHandler) Handle(message *amqp.Message) {
    fmt.Println("AMQP收到消息:  ", message.Value)
}

func main() {
    // 以下参数配置请参考连接配置说明
        // AMQP接入域名
    amqpHost := "127.0.0.1"

    //接入凭证键值
    amqpAccessKey := "your accessKey"

    // 接入凭证密钥
    amqpAccessCode := "your accessCode"

        // 实例Id
    instanceId:= "your intanceId"

    // 订阅队列名称
    amqpQueueName := "DefaultQueue"

    amqpClient := &AmqpClient{
        Title:      "test",
        Host:       amqpHost,
```

```
    AccessKey:  amqpAccessKey,
    AccessCode: amqpAccessCode,
          InstanceId: instanceId,
    QueueName:  amqpQueueName,
  }

  handle := CustomerMessageHandler{}
  amqpClient.InitConnect()
  ctx := context.Background()
  amqpClient.StartReceiveMessage(ctx, &handle)
}
```

7.6.4 使用 MQTT 转发

7.6.4.1 MQTT 转发

订阅推送的示意图如下图所示：



推送机制：物联网平台向用户推送Qos0的消息，如果用户未建链或者建链后未订阅Topic，等达到或超过最大缓存时长（最近24H）或最大缓存大小（1GB），物联网平台会滚动清除超期和超出容量限制的数据。

如何进行数据订阅

1. 在物联网平台创建规则、添加转发目标为MQTT消息队列后实现数据订阅，详情请参考**配置MQTT服务端订阅**。
2. 通过调用API接口进行数据订阅。通过API接口进行数据订阅请参考**如何调用API**、**创建规则触发条件**、**创建规则动作**和**修改规则触发条件**。

推送数据格式

数据订阅成功后，物联网平台推送到应用侧的数据格式样例请参考**流转数据**。

使用限制

描述	限制
支持的MQTT协议版本	3.1.1
与标准MQTT协议的区别	• 支持Qos 0 • 支持Topic自定义 • 支持共享订阅 • 不支持QoS1，QoS2 • 不支持will、retain msg • 不支持客户端Publish
MQTTS支持的安全等级	采用TCP通道基础 + TLS协议（TLSV1.2） 支持的加密套件列表： • TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 • TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
单账号每秒最大MQTT连接请求数	10个
单个账号支持的最大MQTT连接数	10个/接入凭证
单个MQTT连接每秒最大推送速率	1000TPS
消息最大缓存时长及大小	最大时长1天，最大消息量1GB，以最先到达的限制为准。例如，缓存时长超过1天即使没达到1GB也不会缓存。
MQTT连接心跳时间建议值	心跳时间限定为30秒至1200秒，推荐设置为120秒。
消息发布与订阅	• 支持共享订阅，订阅同一Topic的客户端轮询消费推送数据，客户端只能订阅流转规则中创建的Topic。 • 不支持消息发布。
每个订阅请求的最大订阅数	同账号的最大Topic数一致。
每个账号可订阅的Topic数（在创建规则动作时创建）	100

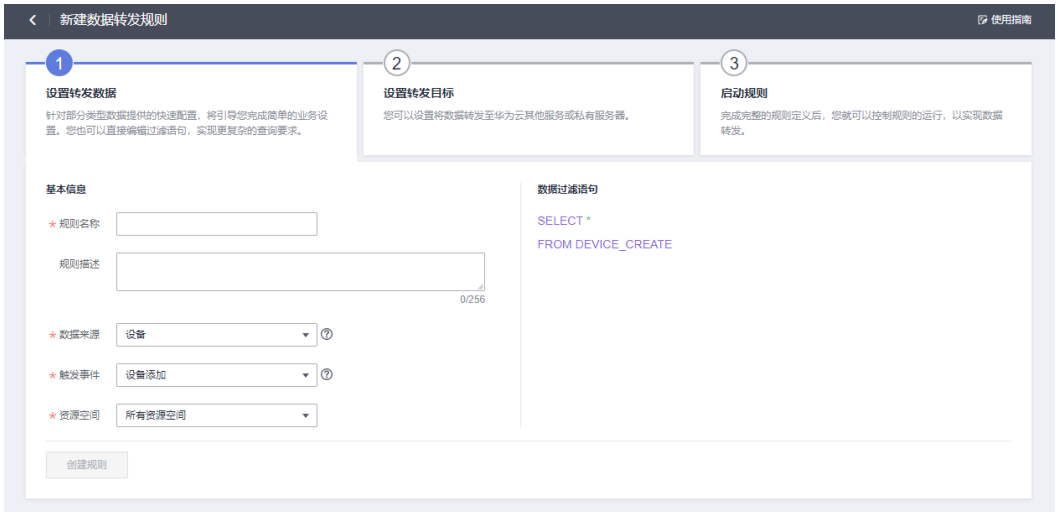
7.6.4.2 配置 MQTT 服务端

本文介绍如何在物联网平台设置和管理MQTT服务端订阅。

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则 > 数据转发”，单击页面左侧的“创建规则”。

图 7-48 创建规则



步骤3 参考下表填写参数后，单击“创建规则”。

表 7-16 创建规则参数列表

参数名	参数说明
规则名称	创建的规则名称。
规则描述	对该规则的描述。

参数名	参数说明
数据来源	• 设备：将操作设备的信息，如设备添加、设备删除、设备更新设置为数据来源。当数据来源选择“设备”时，不支持快速配置。 • 设备属性：将归属在某个资源空间下的设备上报给平台的属性值设置为数据来源。单击右侧的“快速配置”勾选需要转发的产品、属性、服务等数据。 • 设备消息：将归属在某个资源空间下的设备上报给平台的消息设置为转发目标。单击右侧的“快速配置”，仅转发指定Topic的数据。选择所属产品，填写Topic名称。您可以使用在产品详情页面**自定义的Topic**，也可以使用**平台预置的Topic**。 • 设备消息状态：将设备和平台之间流转的设备消息状态变更设置为转发目标。设备消息状态详见**这里**。当数据来源选择“设备消息状态”，不支持快速配置。 • 设备状态：将归属在某个资源空间下的直连或非直连设备状态变更转发至其他服务。单击“快速配置”，您可以转发设备状态为“在线”、“离线”和“异常”的设备信息到其他服务。物联网平台直连设备状态详见**这里**。 • 批量任务：将批量任务状态的数据设置为数据来源。当数据来源选择“批量任务”时，不支持快速配置。 • 产品：将操作产品的信息，如产品添加、产品删除、产品更新设置为数据来源。当数据来源选择“产品”时，不支持快速配置。 • 设备异步命令状态：针对LwM2M/CoAP协议的设备，物联网平台支持下发异步命令给设备。将异步命令的状态变更设置为数据来源。物联网平台设备异步命令状态详见**这里**。当数据来源选择“设备异步命令状态”时，不支持快速配置。 • 运行日志：将MQTT设备的业务运行日志设置为数据来源。当数据来源选择“运行日志”时，不支持快速配置。
触发事件	选择数据来源后，对应修改触发事件。
资源空间	您可以选择单个资源空间或所有资源空间。当选择“所有资源空间”时，不支持快速配置。

步骤4 在设置转发目标页面，单击“添加”，在弹出的页面中参考下表配置完参数后，单击“确认”。

参数名	参数说明
转发目标	选择“MQTT推送消息队列”
推送Topic	输入要转发的MQTT Topic。 • Topic队列名称自定义且单个租户名下唯一，最大长度 128 位，支持大小写英文字符串、数字、下划线（_）、中划线（-）和斜杠（/），不支持除此之外的其他字符。 • 第一次使用的Topic会归属于该规则创建选择的资源空间，后续该Topic只能在该资源空间下使用，如果创建规则时选择的资源空间为"所有资源空间"，则该Topic在所有资源空间下都可以使用。

图 7-49 添加转发目标

步骤5 完成完整的规则定义后，单击“启动规则”，实现数据转发至MQTT消息队列。

----结束

7.6.4.3 MQTT 客户端接入说明

在调用**创建规则触发条件**、**创建规则动作**和**修改规则触发条件**配置并激活规则后，您需要参考本文将MQTT客户端接入物联网平台，成功接入后，在您的服务端运行MQTT客户端，即可接收订阅的消息。

连接配置说明

MQTT客户端接入物联网平台的连接地址和连接认证参数说明如下：

- MQTT接入域名
  每个账号会自动生成，请前往**控制台**-接入信息页面获取。

- 端口：8883
- 客户端身份认证参数

  clientId：全局唯一即可，建议使用“username”。

  username =“accessKey=${accessKey}|timestamp=${timestamp}|instanceId=${instanceId}”

  password =“${accessCode}”

参数	是否必须	说明
${accessKey}	是	接入凭证键值，单个键值最多允许10个客户端同时进行建链。 首次建链时候，请参考这里进行预置。
${timestamp}	是	表示当前时间，13位毫秒值时间戳。服务端校验客户端的时间戳，且时间戳相差5分钟。
instanceId	否	实例Id，同一Region购买多个标准版实例时需要填设置该参数，实例Id参考这里这里获取。
${accessCode}	是	接入凭证密钥，长度不超过256个。

获取 MQTT 接入凭证

若应用使用MQTT协议接入物联网平台进行数据流转需要使用接入凭证，首次使用或者忘记接入凭证请先预置接入凭证。您可以通过调用生成接入凭证接口预置，也可以前往控制台页面进行预置，详细方法请参考如下操作：

步骤1 访问设备接入服务，单击“管理控制台”进入设备接入控制台。

步骤2 选择“规则>数据转发”进入“规则列表”页面。

图 7-50 规则列表

步骤3 单击"详情"(如果没有规则请先创建规则)进入规则详情页面后切换到"设置转发目标"页签。

图 7-51 设置转发目标

步骤4 单击"添加"进入"添加转发目标"页面，设置转发目标为"MQTT推送消息队列",单击“预置接入凭证”预置接入凭证密钥（accessCode）和接入凭证键值（accessKey）。

图 7-52 预置接入凭证

说明

如果您之前预置过接入凭证，重新预置之后，之前的接入凭证密钥将不能再使用。

----结束

接收平台推送的消息

客户端和平台之间建链成功后，订阅数据流转规则中MQTT通道中的Topic，设备上报数据后触发流转规则，平台就会把流转数据推送至MQTT客户端。

7.6.4.4 Java Demo 使用说明

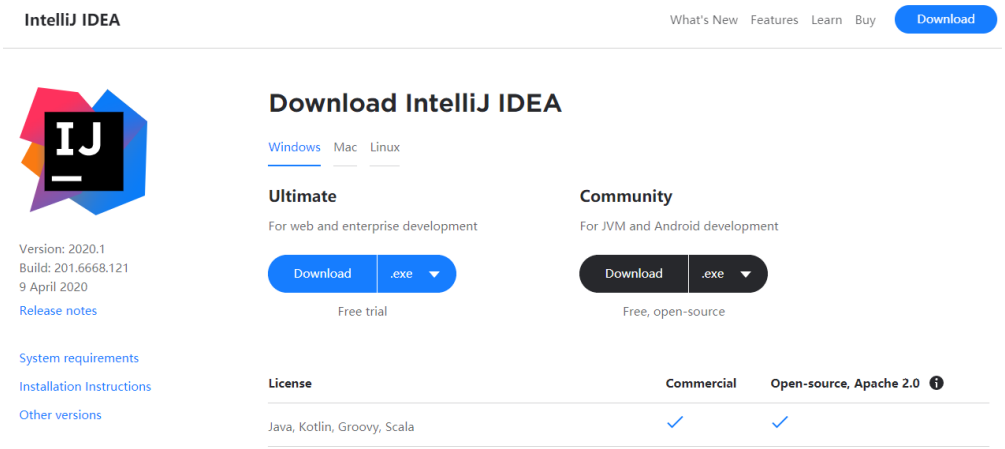
本文以Java语言为例，介绍应用通过MQTTS协议接入平台，接收服务端订阅消息的示例。

前提条件

已安装IntelliJ IDEA开发工具。若未安装请参考安装IntelliJ IDEA。

安装 IntelliJ IDEA

1. 访问IntelliJ IDEA官网，选择合适系统的版本下载。（本文以windows 64-bit系统IntelliJ IDEA 2019.2.3 Ultimate为例）。

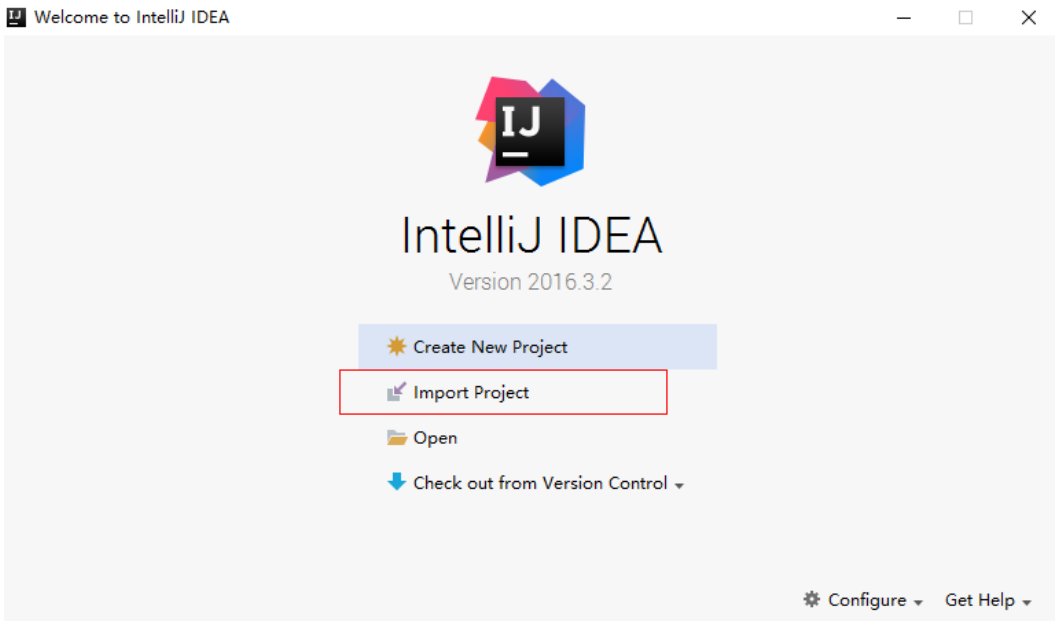


2. 下载完成后，运行安装文件，根据界面提示安装。

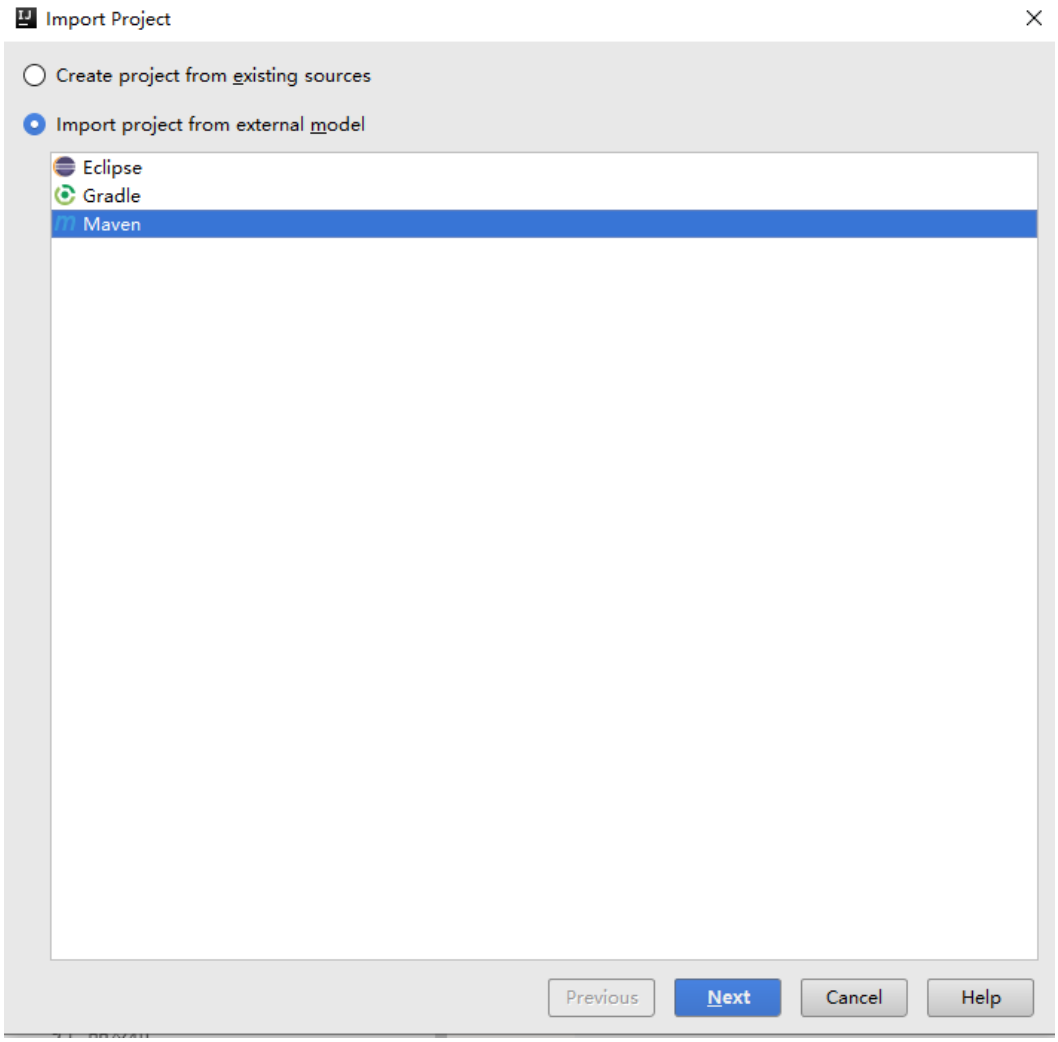
导入代码样例

步骤1 下载JAVA样例。

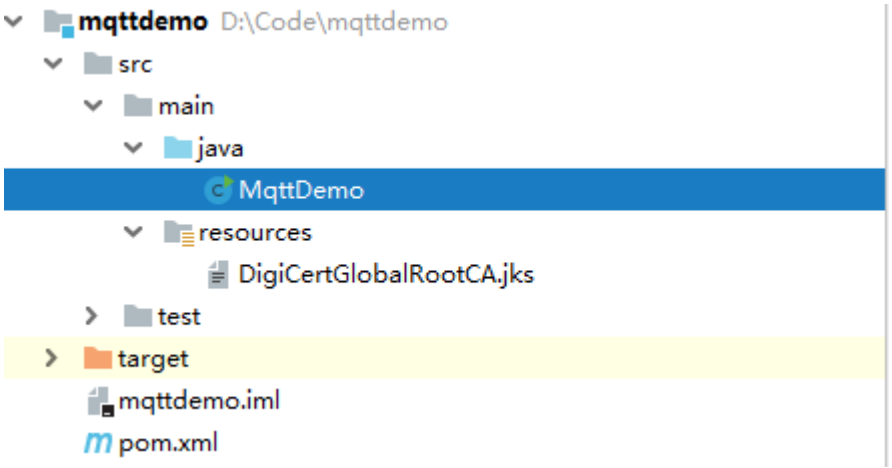
步骤2 打开IDEA开发者工具，单击“ Import Project”。



步骤3 选择步骤1中下载的样例，然后根据界面提示，单击“next”。



步骤4 完成代码导入。



----结束

建立连接

步骤1 设置接入地址及鉴权参数的值：

```
// IoT平台mqtt接入地址，替换成"连接配置说明中"的"MQTT接入域名"
private String serverAddress = "${UUCID}.st1.iotda-app.cn-north-4.myhuaweicloud.com";
// 接入凭证，替换成"获取MQTT接入凭证"中获取的接入凭证
private static String accessKey = "accessKey";
private static String accessCode = "accessCode";
// 接收数据的Topic，替换成"创建规则动作"中的Topic
private static String subscribeTopic = "userTopic";
```

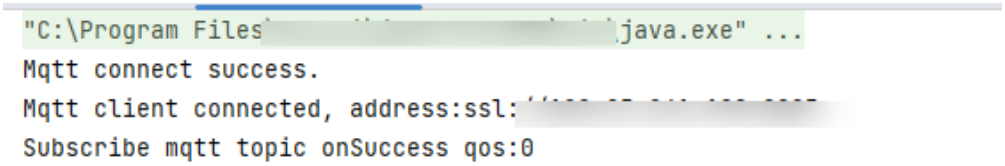
说明

Demo中涉及的参数说明，请参考**连接配置说明**。

步骤2 运行样例代码，根据以下日志信息判断是否订阅成功。

- 订阅成功。

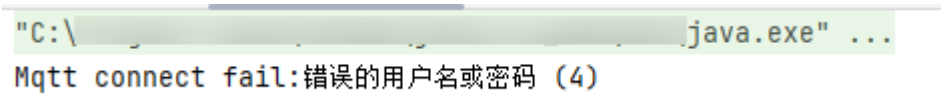
  图 7-53 订阅成功

- 订阅失败。

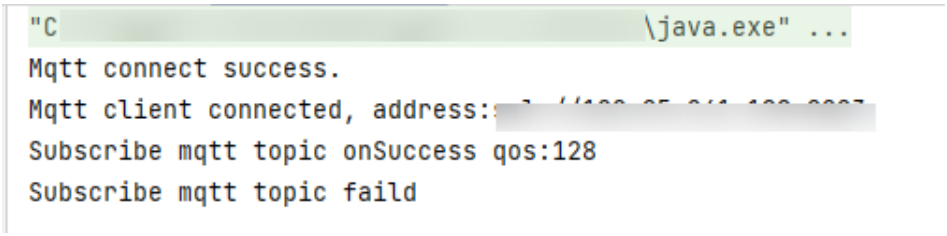
  a. 用户名或密码错误。

  图 7-54 用户或密码错误

  b. 订阅的Topic不存在。

  图 7-55 订阅 topic 不存在

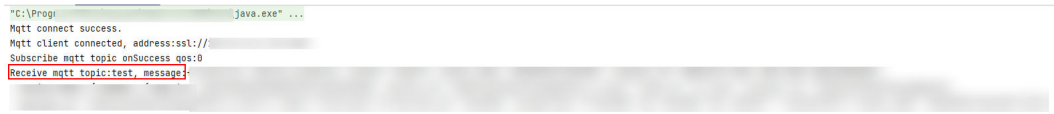
  

----结束

接收数据

Topic订阅后设备上报数据并触发规则后，MQTT客户端就可以收到流转数据。样例代码收取到流转数据的日志如下图所示：

图 7-56 接收到流转数据



7.6.4.5 Python Demo 使用说明

本文以Python语言为例，介绍应用通过MQTTS协议接入平台，接收服务端订阅消息的示例。

前提条件

熟悉Python语言开发环境配置，熟悉Python语言基本语法。

开发环境

本示例使用了Python 3.8.8版本。

添加依赖

本示例使用的Python语言的Mqtt依赖为**paho-mqtt**(本示例使用版本为2.0.0),可以通过以下命令下载依赖。

```
pip install paho-mqtt==2.0.0
```

代码示例

ClientConf代码如下：

```
from typing import Optional
class ClientConf:
    def __init__(self):
        # mqtt订阅地址
        self.__host: Optional[str] = None
        # mqtt订阅端口号
        self.__port: Optional[int] = None
        # mqtt接入凭据access_key
        self.__access_key: Optional[str] = None
        # mqtt接入凭据access_code
        self.__access_code: Optional[str] = None
        # mqtt订阅topic
        self.__topic: Optional[str] = None
        # 实例Id，同一Region购买多个标准版实例时需要填写该参数
        self.__instance_id: Optional[str] = None
        # mqtt qos
        self.__qos = 1

    @property
    def host(self):
        return self.__host
    @host.setter
    def host(self, host):
        self.__host = host
    @property
    def port(self):
        return self.__port
    @port.setter
    def port(self, port):
        self.__port = port
    @property
```

```
    def access_key(self):
        return self.__access_key
    @access_key.setter
    def access_key(self, access_key):
        self.__access_key = access_key
    @property
    def access_code(self):
        return self.__access_code
    @access_code.setter
    def access_code(self, access_code):
        self.__access_code = access_code
    @property
    def topic(self):
        return self.__topic
    @topic.setter
    def topic(self, topic):
        self.__topic = topic
    @property
    def instance_id(self):
        return self.__instance_id
    @instance_id.setter
    def instance_id(self, instance_id):
        self.__instance_id = instance_id
    @property
    def qos(self):
        return self.__qos
    @qos.setter
    def qos(self, qos):
        self.__qos = qos
```

MqttClient代码如下：

```
import os
import ssl
import threading
import time
import traceback
import secrets
from client_conf import ClientConf
import paho.mqtt.client as mqtt
class MqttClient:
    def __init__(self, client_conf: ClientConf):
        self.__host = client_conf.host
        self.__port = client_conf.port
        self.__access_key = client_conf.access_key
        self.__access_code = client_conf.access_code
        self.__topic = client_conf.topic
        self.__instance_id = client_conf.instance_id
        self.__qos = client_conf.qos
        self.__paho_client: Optional[mqtt.Client] = None
        self.__connect_result_code = -1
        self.__default_backoff = 1000
        self.__retry_times = 0
        self.__min_backoff = 1 * 1000  # 1s
        self.__max_backoff = 30 * 1000  # 30s

    def connect(self):
        self.__valid_params()
        rc = self.__connect()
        while rc != 0:
            # 退避重连
            low_bound = int(self.__default_backoff * 0.8)
            high_bound = int(self.__default_backoff * 1.0)
            random_backoff = secrets.randbelow(high_bound - low_bound)
            backoff_with_jitter = int(pow(2, self.__retry_times)) * (random_backoff + low_bound)
            wait_time_ms = self.__max_backoff if (self.__min_backoff + backoff_with_jitter) > self.__max_backoff
else (
                    self.__min_backoff + backoff_with_jitter)
            wait_time_s = round(wait_time_ms / 1000, 2)
            print("client will try to reconnect after " + str(wait_time_s) + " s")
```

```
            time.sleep(wait_time_s)
            self.__retry_times += 1
            self.close()  # 释放之前的connection
            rc = self.__connect()
            # rc为0表示建链成功，其它表示连接不成功
            if rc != 0:
                print("connect with result code: " + str(rc))
                if rc == 134:
                    print("connect failed with bad username or password, "
                          "reconnection will not be performed")
                    pass
        return rc
    def __connect(self):
        try:
            timestamp = self.current_time_millis()
            user_name = "accessKey=" + self.__access_key + "|timestamp=" + timestamp
            if self.__instance_id:
                user_name = user_name + "|instanceId=" + self.__instance_id
            pass_word = self.__access_code
            self.__paho_client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2, "mqttClient")
            # 关闭自动重试， 采用手动重试的方式刷新时间戳
            self.__paho_client._reconnect_on_failure = False
            # 设置回调函数
            self._set_callback()
            # topic放在userdata中，回调函数直接拿topic订阅
            self.__paho_client.user_data_set(self.__topic)
            self.__paho_client.username_pw_set(user_name, pass_word)
            # 当前mqtt broker仅支持TLS1.2
            context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
            # 不校验服务端证书
            context.verify_mode = ssl.CERT_NONE
            context.check_hostname = False
            self.__paho_client.tls_set_context(context)
            rc = self.__paho_client.connect(self.__host, self.__port)
            self.__connect_result_code = rc
            if rc == 0:
                threading.Thread(target=self.__paho_client.loop_forever, args=(1, False),
name="MqttThread").start()
            # 等待建链
            time.sleep(1)
        except Exception as e:
            self.__connect_result_code = -1
            print("Mqtt connection error. traceback: " + traceback.format_exc())
        if self.__paho_client.is_connected():
            return 0
        else:
            return self.__connect_result_code
    def __valid_params(self):
        assert self.__access_key is not None
        assert self.__access_code is not None
        assert self.__topic is not None

    @staticmethod
    def current_time_millis():
        return str(int(round(time.time() * 1000)))
    def _set_callback(self):
        # 当平台响应连接请求时，执行self._on_connect()
        self.__paho_client.on_connect = self._on_connect
        # 当与平台断开连接时，执行self._on_disconnect()
        self.__paho_client.on_disconnect = self._on_disconnect
        # 当订阅topic时，执行self._on_subscribe
        self.__paho_client.on_subscribe = self._on_subscribe
        # 当接收到一个原始消息时，执行self._on_message()
        self.__paho_client.on_message = self._on_message
    def _on_connect(self, client, userdata, flags, rc: mqtt.ReasonCode, properties):
        if rc == 0:
            print("Connected to Mqtt Broker! topic " + self.__topic)
            client.subscribe(userdata, 1)
        else:
```

```
        # 只有当用户名或密码错误，才不进行自动重连。
        # 如果这里不使用disconnect()方法，那么loop_forever会一直进行重连。
        if rc == 134:
            self.__paho_client.disconnect()
        print("Failed to connect. return code :" + str(rc.value) + ", reason" + rc.getName())
    def _on_subscribe(self, client, userdata, mid, granted_qos, properties):
        print("Subscribed: " + str(mid) + " " + str(granted_qos) + " topic: " + self.__topic)
    def _on_message(self, client, userdata, message: mqtt.MQTTMessage):
        print("topic " + self.__topic + " Received message: " + message.payload.decode())
    def _on_disconnect(self, client, userdata, flags, rc, properties):
        print("Disconnect to Mqtt Broker. topic: " + self.__topic)
        # 断链后将客户端主动关闭，手动重连刷新时间戳
        try:
            self.__paho_client.disconnect()
        except Exception as e:
            print("Mqtt connection error. traceback: " + traceback.format_exc())
        self.connect()
    def close(self):
        if self.__paho_client is not None and self.__paho_client.is_connected():
            try:
                self.__paho_client.disconnect()
                print("Mqtt connection close")
            except Exception as e:
                print("paho client disconnect failed. exception: " + str(e))
        else:
            pass
```

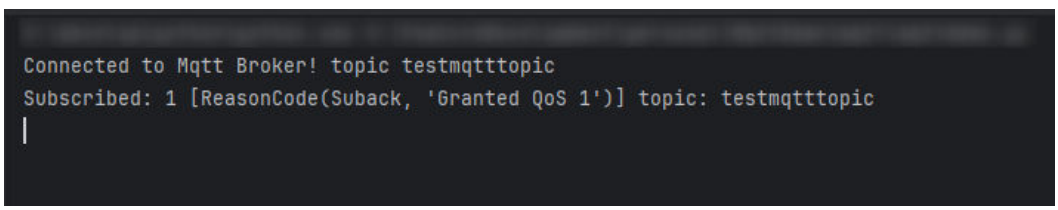
MqttDemo代码如下：

```
from client_conf import ClientConf
from mqtt_client import MqttClient
import os
from typing import Optional
def main():
    client_conf = ClientConf()
    client_conf.host = "your ip host"
    client_conf.port = 8883
    client_conf.topic = "your mqtt topic"
    # mqtt接入凭据access_key可使用环境变量的方式注入
    client_conf.access_key = os.environ.get("MQTT_ACCESS_KEY")
    # mqtt接入凭据access_code可使用环境变量的方式注入
    client_conf.access_code = os.environ.get("MQTT_ACCESS_CODE")
    client_conf.instance_id = "your instance id"
    mqtt_client = MqttClient(client_conf)
    if mqtt_client.connect() != 0:
        print("init failed")
        return
if __name__ == "__main__":
    main()
```

成功示例

接入成功后，客戶端打印信息如下：

图 7-57 python mqtt 订阅接入成功示例



7.6.4.6 GO Demo 使用说明

本文以Go语言为例，介绍应用通过MQTTS协议接入平台，接收服务端订阅消息的示例。

前提条件

熟悉Go语言开发环境配置，熟悉Go语言基本语法。

开发环境

本示例使用了Go 1.18版本。

添加依赖

本示例使用的Go语言的Mqtt依赖为paho.mqtt.golang（本示例使用版本为v1.4.3），在go.mod中添加依赖的代码如下：

```
require (
    github.com/eclipse/paho.mqtt.golang v1.4.3
)
```

代码示例

```
package main

import (
  "crypto/tls"
  "fmt"
  mqtt "github.com/eclipse/paho.mqtt.golang"
  "os"
  "os/signal"
  "time"
)
type MessageHandler func(message string)
type MqttClient struct {
  Host            string
  Port            int
  ClientId        string
  AccessKey       string
  AccessCode      string
  Topic           string
  InstanceId      string
  Qos             int
  Client          mqtt.Client
  messageHandlers []MessageHandler
}
func (mqttClient *MqttClient) Connect() bool {
  return mqttClient.connectWithRetry()
}
func (mqttClient *MqttClient) connectWithRetry() bool {
  // 退避重试，从10ms依次x2，直到20s。
  duration := 10 * time.Millisecond
  maxDuration := 20000 * time.Millisecond
  // 建链失败进行重试
  internal := mqttClient.connectInternal()
  times := 0
  for !internal {
    time.Sleep(duration)
    if duration < maxDuration {
      duration *= 2
    }
    times++
    fmt.Println("connect mqttgo broker retry. times: ", times)
```

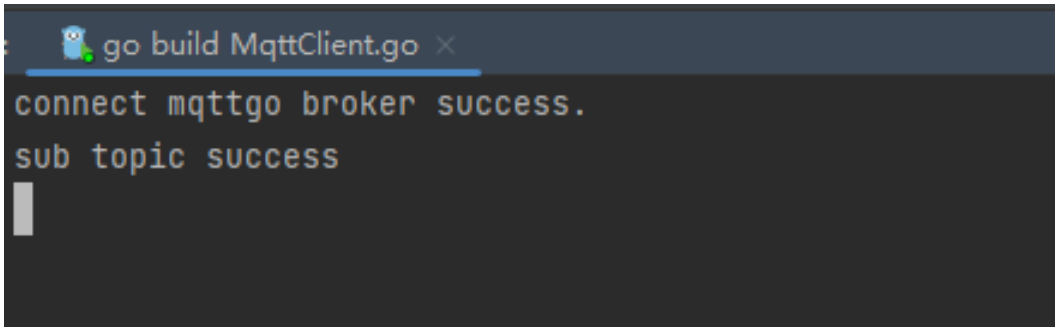
```
    internal = mqttClient.connectInternal()
  }
  return internal
}
func (mqttClient *MqttClient) connectInternal() bool {
  // 建链前先关闭已有连接
  mqttClient.Close()
  options := mqtt.NewClientOptions()
  options.AddBroker(fmt.Sprintf("mqtts://%s:%d", mqttClient.Host, mqttClient.Port))
  options.SetClientID(mqttClient.ClientId)
  userName := fmt.Sprintf("accessKey=%s|timestamp=%d", mqttClient.AccessKey, time.Now().UnixNano()/
1000000)
  if len(mqttClient.InstanceId) != 0 {
    userName = userName + fmt.Sprintf("|instanceId=%s", mqttClient.InstanceId)
  }
  options.SetUsername(userName)
  options.SetPassword(mqttClient.AccessCode)
  options.SetConnectTimeout(10 * time.Second)
  options.SetKeepAlive(120 * time.Second)
  // 关闭sdk内部重连，使用自定义重连刷新时间戳
  options.SetAutoReconnect(false)
  options.SetConnectRetry(false)
  tlsConfig := &tls.Config{
    InsecureSkipVerify: true,
    MaxVersion:         tls.VersionTLS12,
    MinVersion:         tls.VersionTLS12,
  }
  options.SetTLSConfig(tlsConfig)
  options.OnConnectionLost = mqttClient.createConnectionLostHandler()
  client := mqtt.NewClient(options)
  if token := client.Connect(); token.Wait() && token.Error() != nil {
    fmt.Println("device create bootstrap client failed,error = ", token.Error().Error())
    return false
  }
  mqttClient.Client = client
  fmt.Println("connect mqttgo broker success.")
  mqttClient.subscribeTopic()
  return true
}
func (mqttClient *MqttClient) subscribeTopic() {
  subRes := mqttClient.Client.Subscribe(mqttClient.Topic, 0, mqttClient.createMessageHandler())
  if subRes.Wait() && subRes.Error() != nil {
    fmt.Printf("sub topic failed,error is %s\n", subRes.Error())
    panic("subscribe topic failed.")
  } else {
    fmt.Printf("sub topic success\n")
  }
}
func (mqttClient *MqttClient) createMessageHandler() func(client mqtt.Client, message mqtt.Message) {
  messageHandler := func(client mqtt.Client, message mqtt.Message) {
    fmt.Println("receive message from server.")
    go func() {
      for _, handler := range mqttClient.messageHandlers {
        handler(string(message.Payload()))
      }
    }()
  }
  return messageHandler
}
func (mqttClient *MqttClient) createConnectionLostHandler() func(client mqtt.Client, reason error) {
  // 断链后进行自定义重连
  connectionLostHandler := func(client mqtt.Client, reason error) {
    fmt.Printf("connection lost from server. begin to reconnect broker. reason: %s\n", reason.Error())
    connected := mqttClient.connectWithRetry()
    if connected {
      fmt.Println("reconnect mqttgo broker success.")
    }
  }
  return connectionLostHandler
```

```
}
func (mqttClient *MqttClient) Close() {
  if mqttClient.Client != nil {
    mqttClient.Client.Disconnect(1000)
  }
}
func main() {
  // 以下参数配置请参考连接配置说明
  // mqtt接入域名
  mqttHost := "your mqtt host"
  // mqtt接入端口
  mqttPort := 8883
  //接入凭证键值
  mqttAccessKey := os.Getenv("MQTT_ACCESS_KEY")
  //接入凭证密钥
  mqttAccessCode := os.Getenv("MQTT_ACCESS_CODE")
  //订阅topic名称
  mqttTopic := "your mqtt topic"
  //实例Id
  instanceId := "your instance Id"
  //mqttgo client id
  clientId := "your mqtt client id"

  mqttClient := MqttClient{
    Host:       mqttHost,
    Port:       mqttPort,
    Topic:      mqttTopic,
    ClientId:   clientId,
    AccessKey:  mqttAccessKey,
    AccessCode: mqttAccessCode,
    InstanceId: instanceId,
  }
  //自定义消息处理handler
  mqttClient.messageHandlers = []MessageHandler{func(message string) {
    fmt.Println(message)
  }}
  connect := mqttClient.Connect()
  if !connect {
    fmt.Println("init mqttgo client failed.")
    return
  }
  //阻塞方法，保持mqtt客户端一直拉消息
  interrupt := make(chan os.Signal, 1)
  signal.Notify(interrupt, os.Interrupt)
  for {
    <-interrupt
    break
  }
}
```

成功示例

接入成功后，客户端打印信息如下：

图 7-58 go mqtt 客户端接入成功示例



7.6.4.7 Node.js Demo 使用说明

本文以Node.js语言为例，介绍应用通过MQTTS协议接入平台，接收服务端订阅消息的示例。

前提条件

熟悉Node.js语言开发环境配置，熟悉Node.js语言基本语法。

开发环境

本示例所使用的开发环境为Node.js v13.14.0版本。请前往Node.js官网**下载**。安装成功之后可以通过以下命令查看node版本。

```
node --version
```

添加依赖

本示例使用的Node.js语言的Mqtt依赖为mqtt（本示例使用版本为4.0.0），可以通过以下命令下载依赖。

```
npm install mqtt@4.0.0
```

代码示例

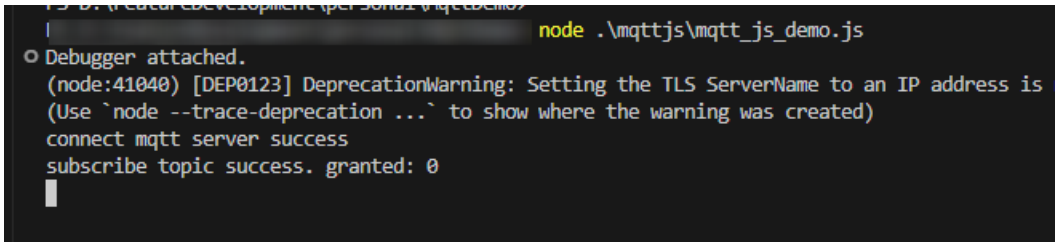
```
const mqtt = require('mqtt');
// 订阅topic名称
var topic = "your mqtt topic";
// 接入凭证键值，可通过环境变量预制
var accessKey = process.env.MQTT_ACCESS_KEY;
// 接入凭证密钥，可通过环境变量预制
var accessCode = process.env.MQTT_ACCESS_CODE;
// mqtt接入地址
var mqttHost = "your mqtt host";
// mqtt接入端口
var mqttPort = 8883;
// 实例Id
var instanceId = "your instanceId";
// mqtt client id
var clientId = "your clientId";
// mqtt 客户端
var client = null;
connectWithRetry();
async function connectWithRetry() {
    // 退避重试，从1s依次x2，直到20s。
    var duration = 1000;
    var maxDuration = 20000;
    var success = connect(topic);
    var times = 0;
    while (!success) {
      await sleep(duration)
      if (duration < maxDuration) {
        duration *= 2
      }
      times++
      console.log('connect mqtt broker retry. times: ' + times)
       if (client == null) {
           connect(topic)
           continue
       }
      client.end(true, function() {
           connect(topic)
       });
    }
```

```
}
function sleep(ms) {
    return new Promise(resolve => setTimeout(() => resolve(), ms))
}
function connect(topic) {
    try {
        client = mqtt.connect(getClientOptions())
        if (client == null) {
            return false
        }
        client.on('connect', connectCallBack)
        client.subscribe(topic, subscribeCallBack)
        client.on('message', messageCallBack)
        client.on('error', clientErrorCallBack)
        client.on('close', closeCallBack)
        return true
    } catch (error) {
        console.log('connect to mqtt broker failed. err ' + error)
    }
    return false
}
function getClientOptions() {
    var timestamp = Math.round(new Date);
    const username = 'accessKey=' + accessKey + '|timestamp=' + timestamp + '|instanceId=' + instanceId;
    var options = {
        host: mqttHost,
        port: mqttPort,
        connectTimeout: 4000,
        clientId: clientId,
        protocol: 'mqtts',
        keepalive: 120,
        username: username,
        password: accessCode,
        rejectUnauthorized: false,
        secureProtocol: 'TLSv1_2_method'
    };
    return options;
};
function connectCallBack() {
    console.log('connect mqtt server success');
};
function subscribeCallBack(err, granted) {
    if (err != null || granted[0].qos === 128) {
        console.log('subscribe topic failed. granted: ' + granted[0].qos)
        return
    }
    console.log('subscribe topic success. granted: ' + granted[0].qos);
};
function clientErrorCallBack(err) {
    console.log('mqtt client error ' + err);
};
function messageCallBack(topic, message) {
    console.log('receive message ' + message);
};
function closeCallBack() {
    console.log('Disconnected from mqtt broker')
    client.end(true, function() {
        console.log('close connection');
        connectWithRetry();
     });
}
```

成功示例

接入成功后，客户端打印信息如下：

图 7-59 node.js mqtt 客户端接入成功示例



7.6.4.8 C# Demo 使用说明

本文以C#语言为例，介绍应用通过MQTTS协议接入平台，接收服务端订阅消息的示例。

前提条件

熟悉.NETFramework开发环境配置，熟悉C#语言基本语法。

开发环境

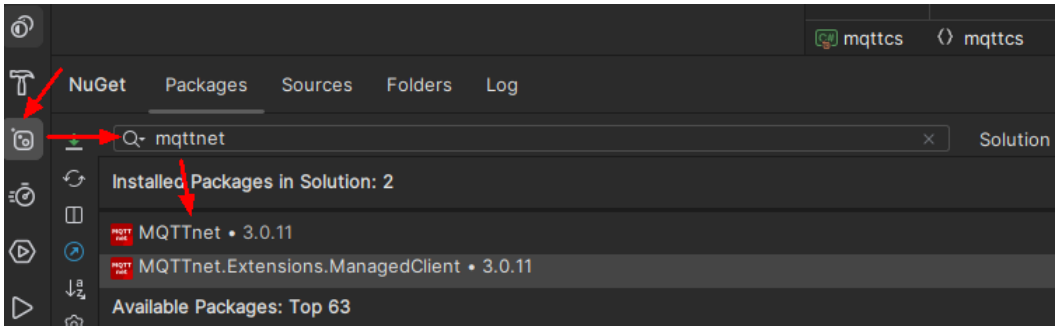
本示例所使用的开发环境为.NETFramework 4.6.2版本，.Net SDK 6.0.421版本。请前往.NET官网下载。安装成功之后可以通过以下命令查看.Net SDK版本。

```
dotnet -v
```

添加依赖

本示例使用C#语言的Mqtt依赖为MQTTnet和MQTTnet.Extension.ManagedClient（使用版本为3.0.11），可以在NuGet管理器中搜索到"MQTTnet"后安装所需版本。

图 7-60 nuget 安装依赖



代码示例

ClientConf.cs代码如下：

```
using MQTTnet.Protocol;

namespace mqttcs
{
    public class ClientConf
    {
        // mqtt订阅地址
```

```
        public string ServerUri { get; set; }

        // mqtt订阅端口号
        public int Port { get; set; }

        // mqtt接入凭据access_key
        public string AccessKey { get; set; }

        // mqtt接入凭据access_code
        public string AccessCode { get; set; }

        // mqtt 客户端ID
        public string ClientId { get; set; }

        // 实例Id，同一Region购买多个标准版实例时需要填写该参数
        public string InstanceId { get; set; }

        // mqtt订阅topic
        public string Topic { get; set; }

        // mqtt qos
        public MqttQualityOfServiceLevel Qos { get; set; }

    }
}
```

MqttListener代码如下：

```
using System;
using MQTTnet.Client.Connecting;
using MQTTnet.Client.Disconnecting;
using MQTTnet.Extensions.ManagedClient;

namespace mqttcs
{
    public interface MqttListener
    {
        // mqtt客户端与服务端断链回调函数
        void ConnectionLost(MqttClientDisconnectedEventArgs e);

        // mqtt客户端与服务端建链成功回调函数
        void ConnectComplete(MqttClientConnectResultCode resultCode, String reason);

        // mqtt客户端消费消息回调函数
        void OnMessageReceived(String message);

        // mqtt客户端与服务端建链失败回调函数
        void ConnectFail(ManagedProcessFailedEventArgs e);
    }
}
```

MqttConnection.cs代码示例如下：

```
using System;
using System.Text;
using System.Threading;
using MQTTnet;
using MQTTnet.Client.Connecting;
using MQTTnet.Client.Disconnecting;
using MQTTnet.Client.Options;
using MQTTnet.Client.Receiving;
using MQTTnet.Extensions.ManagedClient;
using MQTTnet.Formatter;

namespace mqttcs
{
    public class MqttConnection
    {
        private static IManagedMqttClient client = null;
```

```
    private static ManualResetEvent mre = new ManualResetEvent(false);

    private static readonly ushort DefaultKeepLive = 120;

    private static int _retryTimes = 0;

    private readonly int _retryTimeWait = 1000;

    private readonly ClientConf _clientConf;

    private MqttListener _listener;

    public MqttConnection(ClientConf clientConf, MqttListener listener)
    {
        _clientConf = clientConf;
        _listener = listener;
    }

    public int Connect()
    {
        client?.StopAsync();
        // 退避重试，从1s直到20s
        var duration = 1000;
        var maxDuration = 20 * 1000;
        var rc = InternalConnect();
        while (rc != 0)
        {
            Thread.Sleep((int)duration);
            if (duration < maxDuration)
            {
                duration *= 2;
            }
            client?.StopAsync();
            _retryTimes++;
            Console.WriteLine("connect mqtt broker retry. times: " + _retryTimes);
            rc = InternalConnect();
        }

        return rc;
    }

    private int InternalConnect()
    {
        try
        {
            client = new MqttFactory().CreateManagedMqttClient();
            client.ApplicationMessageReceivedHandler =
                new
MqttApplicationMessageReceivedHandlerDelegate(ApplicationMessageReceiveHandlerMethod);
            client.ConnectedHandler = new MqttClientConnectedHandlerDelegate(OnMqttClientConnected);
            client.DisconnectedHandler = new
MqttClientDisconnectedHandlerDelegate(OnMqttClientDisconnected);
            client.ConnectingFailedHandler = new
ConnectingFailedHandlerDelegate(OnMqttClientConnectingFailed);
            IManagedMqttClientOptions options = GetOptions();
            // Connects to the platform.
            client.StartAsync(options);
            mre.Reset();

            mre.WaitOne();
            if (!client.IsConnected)
            {
                return -1;
            }

            var mqttTopicFilter = new
MqttTopicFilterBuilder().WithTopic(_clientConf.Topic).WithQualityOfServiceLevel(_clientConf.Qos).Build();

            client.SubscribeAsync(mqttTopicFilter).Wait();
```

```
            Console.WriteLine("subscribe topic success.");
            return 0;
        }
        catch (Exception e)
        {
            Console.WriteLine("Connect to mqtt server failed. err: " + e);
            return -1;
        }
    }

    private void ApplicationMessageReceiveHandlerMethod(MqttApplicationMessageReceivedEventArgs e)
    {
        string payload = null;
        if (e.ApplicationMessage.Payload != null)
        {
            payload = Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
        }
        try
        {
            _listener?.OnMessageReceived(payload);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Message received error, the message is " + payload);
        }

    }

    private void OnMqttClientConnected(MqttClientConnectedEventArgs e)
    {
        try
        {
            _retryTimes = 0;
            _listener?.ConnectComplete(e.AuthenticateResult.ResultCode, e.AuthenticateResult.ReasonString);
            mre.Set();
        }
        catch (Exception exception)
        {
            Console.WriteLine("handle connect callback failed. e: " + exception.Message);
        }
    }

    private void OnMqttClientDisconnected(MqttClientDisconnectedEventArgs e)
    {
        try
        {
            _listener?.ConnectionLost(e);
        }
        catch (Exception exception)
        {
            Console.WriteLine("handle disConnect callback failed. e: " + exception.Message);
        }

    }

    private void OnMqttClientConnectingFailed(ManagedProcessFailedEventArgs e)
    {
        try
        {
            if (_listener != null)
            {
                _listener.ConnectFail(e);
            }
            Thread.Sleep(_retryTimeWait);
            Connect();
        }
        catch (Exception exception)
        {
            Console.WriteLine("handle connect failed callback failed. e: " + exception.Message);
```

```
        }
    }

    private IManagedMqttClientOptions GetOptions()
    {
        IManagedMqttClientOptions options = null;
        long timestamp = new DateTimeOffset(DateTime.UtcNow).ToUnixTimeMilliseconds();
        string userName = "accessKey=" + _clientConf.AccessKey + "|timestamp=" + timestamp + "|
instanceId=" + _clientConf.InstanceId;

        options = new ManagedMqttClientOptionsBuilder()
            .WithClientOptions(new MqttClientOptionsBuilder()
                .WithTcpServer(_clientConf.ServerUri, _clientConf.Port)
                .WithCredentials(userName, _clientConf.AccessCode)
                .WithClientId(_clientConf.ClientId)
                .WithKeepAlivePeriod(TimeSpan.FromSeconds(DefaultKeepLive))
                .WithTls(new MqttClientOptionsBuilderTlsParameters()
                {
                    AllowUntrustedCertificates = true,
                    UseTls = true,
                    CertificateValidationHandler = delegate { return true; },
                    IgnoreCertificateChainErrors = false,
                    IgnoreCertificateRevocationErrors = false,
                    SslProtocol = System.Security.Authentication.SslProtocols.Tls12,
                })
                .WithProtocolVersion(MqttProtocolVersion.V500)
                .Build())
            .Build();
        return options;
    }
  }
}
```

MqttClient.cs代码示例如下：

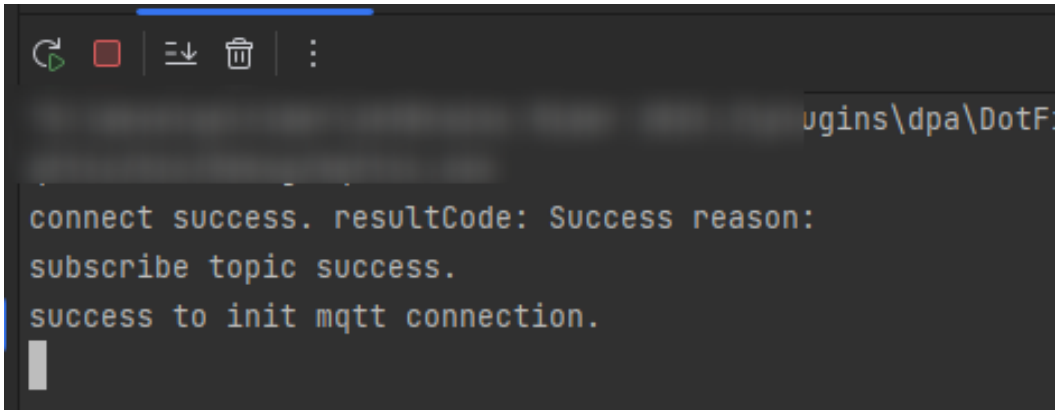
```
using System;
using System.Threading;
using System.Threading.Tasks;
using MQTTnet.Client.Connecting;
using MQTTnet.Client.Disconnecting;
using MQTTnet.Extensions.ManagedClient;
using MQTTnet.Protocol;

namespace mqttcs
{
  class MqttClient: MqttListener
  {
    private static ManualResetEvent mre = new ManualResetEvent(false);

    public static async Task Main(string[] args)
    {
        ClientConf clientConf = new ClientConf();
        clientConf.ClientId = "your mqtt clientId";
        clientConf.ServerUri = "your mqtt host";
        clientConf.Port = 8883;
        clientConf.AccessKey = Environment.GetEnvironmentVariable("MQTT_ACCESS_KEY");
        clientConf.AccessCode = Environment.GetEnvironmentVariable("MQTT_ACCESS_CODE");
        clientConf.InstanceId = "your instanceId";
        clientConf.Topic = "your mqtt topic";
        clientConf.Qos = MqttQualityOfServiceLevel.AtMostOnce;

        MqttConnection connection = new MqttConnection(clientConf, new MqttClient());
        var connect = connection.Connect();
        if (connect == 0)
        {
            Console.WriteLine("success to init mqtt connection.");
            mre.WaitOne();
        }
    }
```

```
    public void ConnectionLost(MqttClientDisconnectedEventArgs e)
    {
        if (e?.Exception != null)
        {
            Console.WriteLine("connect was lost. exception: " + e.Exception.Message);
            return;
        }
        Console.WriteLine("connect was lost");

    }

    public void ConnectComplete(MqttClientConnectResultCode resultCode, String reason)
    {
        Console.WriteLine("connect success. resultCode: " + resultCode + " reason: " + reason);
    }

    public void OnMessageReceived(string message)
    {
        Console.WriteLine("receive msg: " + message);
    }

    public void ConnectFail(ManagedProcessFailedEventArgs e)
    {
        Console.WriteLine("connect mqtt broker failed. e: " + e.Exception.Message);
    }
  }
}
```

成功示例

接入成功后，客户端打印如下：

图 7-61 c#客户端接入成功示例



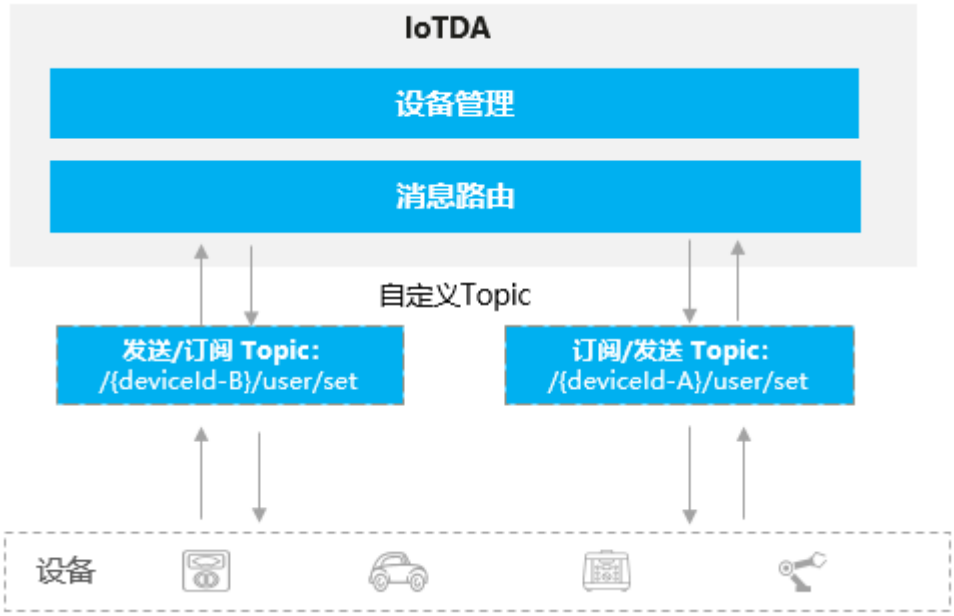
7.6.5 设备间通信

概述

订阅：物联网平台支持基于MQTT协议实现设备间的消息通信，用户可通过控制台创建规则，也可以通过调用物联网平台的**创建规则触发条件**、**创建规则动作**、**修改规则触发条件**接口配置并激活规则，向平台获取设备上报的消息。设备订阅只支持消息上报。

推送：订阅成功后，物联网平台会将设备上报的消息推送到指定的MQTT Topic，当设备接入平台后，可以通过订阅该Topic来接收数据，从而实现设备间的消息通信。设备间消息通信示意图如下：

图 7-62 设备间消息通信



如何进行设备订阅

设备订阅方式请参考**设备间消息通信使用说明**。

7.7 查看数据转发通道详情

概述

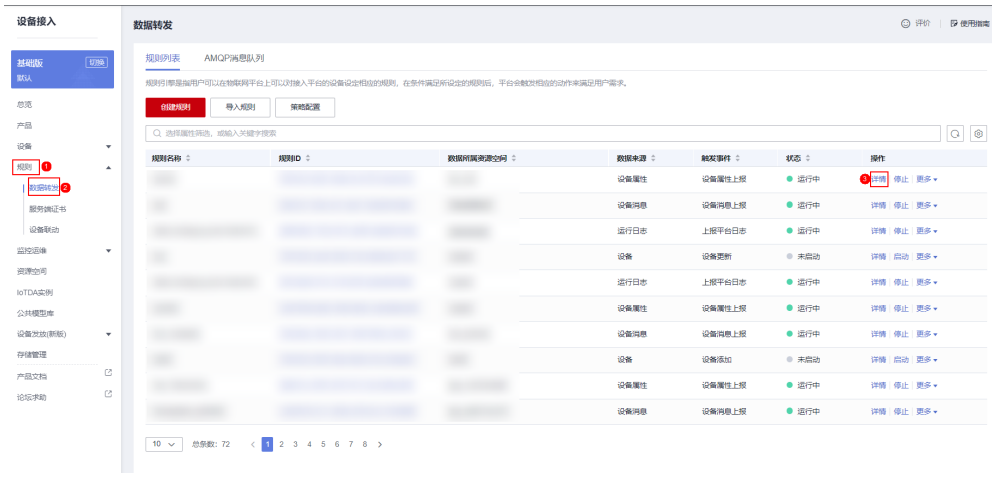
在使用数据转发能力时，可以根据数据转发通道的使用情况，推断出转发目标(如第三方应用服务器等)的性能是否满足业务需求。例如，物联网平台的流转数据因为转发目标（第三方应用服务器）性能等问题无法快速处理时，这些数据会被积压（缓存）在IoT物联网平台中。此时，通过观察数据转发通道的详细信息，我们可以发现消息生产速率一直高于消息推送速率，同时消息积压量不断增加。这表明转发目标（第三方应用服务器）的性能无法满足当前业务需求，需要进行扩容处理。此外，IoT物联网平台还提供了**数据转发通道积压数据清理**的能力。

转发通道详情查看

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台。

步骤2 单击“规则”，选择“数据转发”，找到对应规则单击“详情”进入数据转发规则详情界面。

图 7-63 规则详情-数据转发规则



步骤3 选择“设置转发目标”，找到数据转发通道单击“详情”，在弹窗中可查看推送详情。

图 7-64 数据转发规则详情-数据转发规则

图 7-65 推送详情-数据转发规则

表 7-17 参数说明

参数名称	描述
消息推送速率	物联网平台每秒往转发目标推送消息数量。
消息生产速率	设备侧每秒往物联网平台生产的消息数量。
消息积压量	当生产速率大于推送速率时，积压在物联网的消息数量。物联网平台默认配置**单个转发规则**流转数据的最大积压（缓存）大小为1GB，最大积压（缓存）时间为24小时，如需修改可参考**数据转发积压策略配置**。

----结束

清空积压消息

当规则引擎将消息转发至第三方应用服务器时，若该服务器故障导致消息无法及时处理，IoT物联网平台将会积压数据。考虑到用户对数据实时性的要求，我们支持清理积压在转发通道中的数据，以确保数据的及时处理和传输。

例如：当定时上报用户使用量的水表将数据发送至目标服务器时，由于服务器出现故障导致数据积压，为保证数据的实时性，我们可以使用"清空积压消息"能力丢弃积压数据，优先处理新上报的数据。

须知

在特定的转发目标详情中单击“清空积压消息”，对应的规则转发目标还没完成流转的数据将被全部清空，请评估好业务影响后谨慎操作。

7.8 数据转发积压策略配置

概述

当转发目标（如第三方应用服务器等）由于性能等原因无法快速处理IoT物联网平台流转数据时，未能及时处理的流转数据会积压（缓存）在IoT物联网平台。目前IoT物联网平台默认配置**单个转发规则**流转数据的最大积压（缓存）大小为1GB，最大积压（缓存）时间为24小时，超过最大积压（缓存）大小或积压（缓存）时间时，最早未被处理的流转数据会被**丢弃**直至满足积压（缓存）大小和时间限制。

客户可以根据自身的业务场景以及转发目标（如第三方应用服务器等）的性能等情况，在IoT物联网平台上创建合适的积压策略，控制流转数据在IoT物联网平台的积压情况。

例如：当业务对数据实时性的要求高于完整性，而转发目标（如第三方应用服务器等）长时间性能不足或业务中断一段时间，未能及时处理IoT物联网平台流转数据，导致流转数据大量积压在IoT物联网平台，转发目标（如第三方应用服务器等）接收到的

数据一直是延迟、滞后的数据， 此时可以考虑使用积压策略配置较小的积压大小和积压时间，**丢弃**过时的数据，接收、处理较为实时的流转数据。

使用限制

单个租户在单个IoT物联网平台实例下最多可以创建1个数据流转积压策略。

须知

1. 积压策略创建成功后对所有转发规则生效，即覆盖所有转发规则的默认积压大小（1GB）和默认积压时间（24小时）生效。
2. 超过最大积压（缓存）大小或积压（缓存）时间时，最早未被处理的流转数据会被**丢弃**直至满足积压（缓存）大小和时间限制，请慎重考虑后使用积压策略并配置合理积压大小和时间。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 依次单击“规则 - 数据转发 - 策略配置 - 数据流转积压策略”进入积压策略配置界面。

图 7-66 策略配置界面

步骤3 在弹出的界面中填写名称、描述、积压大小、积压时间等信息后，单击确定即可完成积压策略创建。

图 7-67 创建积压策略

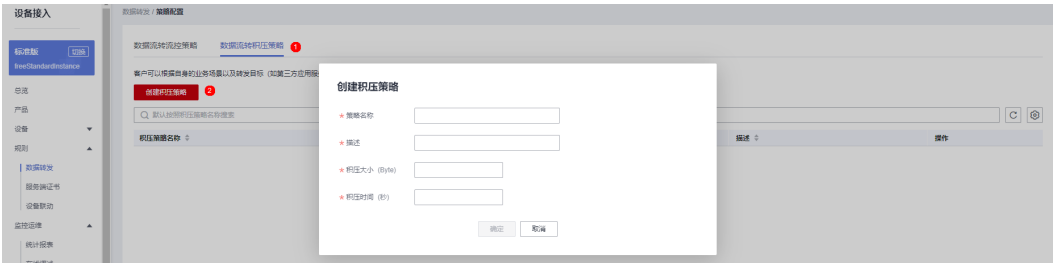


表 7-18 参数说明

参数名称	描述
策略名称	长度为4-256，只允许中文、字母、数字以及_?'#().,&%@!-等字符的组合。

参数名称	描述
描述	关于该策略的描述，长度为4-256，只允许中文、字母、数字以及_?'#().,&%@!-等字符的组合。
积压大小	单个转发规则流转数据在IoT物联网平台最大积压（缓存）大小，单位为B，最大可配置为1073741823 B，即1GB。
积压时间	单个转发规则流转数据在IoT物联网平台最大积压（缓存）时间，单位为秒，最大可配置为86399秒，即24小时。

----结束

7.9 数据转发流控策略配置

概述

客户可以根据自身的业务场景以及转发目标（如第三方应用服务器等）的性能等情况，在IoT物联网平台上创建不同维度的流控策略，控制IoT物联网平台数据转发到转发目标（如第三方应用服务器等）的流量限制。

流控策略维度

表 7-19 流控策略类型

流控策略类型	描述
实例级别流控	使用此策略类型时，流控的范围为当前实例中所有数据转发的流量，超出流控的数据会被丢弃。
转发通道流控	使用此策略类型时，流控的范围为当前实例中所有转发至指定目标的流量。
转发规则流控	使用此策略类型时，流控的范围为当前实例中所有触发指定转发规则的流量，超出流控的数据会被丢弃。
转发动作流控	使用此策略类型时，流控的范围为当前实例中所有转发至指定动作的流量。

须知

1. 流控策略创建成功后，不支持修改流控策略类型。
2. 超出实例和转发规则级别流控的数据会被**丢弃**，请慎重考虑后再使用。
3. 同时创建多个不同类型的流控策略，以最先达到流控阈值的策略为准。例如配置了转发规则A的流控大小为50tps，转发规则A下转发动作B的流控大小为100tps，触发转发规则A的流转数据流量为80tps，则触发了转发规则A的流控。

使用限制

租户在单个IoT物联网平台实例下最多可以创建4个数据流转流控策略。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 单击“规则” ，选择“数据转发” ，选择 “策略配置” ，进入流控策略配置界面。

图 7-68 策略配置界面

步骤3 在弹出的界面中填写相关信息后，单击“确定”即可完成流控策略创建。

图 7-69 创建流控策略

表 7-20 参数说明

参数名称	描述
策略名称	长度为4-256，只允许中文、字母、数字以及_?'#().,&%@!-等字符的组合。
描述	关于该策略的描述，长度为4-256，只允许中文、字母、数字以及_?'#().,&%@!-等字符的组合。
流控大小	取值范围为1-1000。

参数名称	描述
流控策略类型	包括实例级别流控、转发通道流控、转发规则流控、转发动作流控四种流控策略类型。
转发目标	当流控策略类型为转发通道流控时生效，对应当前实例支持的转发通道类型。
绑定规则	当流控策略类型为转发规则流控时生效，对应IoT物联网平台上的数据转发规则。
绑定动作	当流控策略类型为转发动作流控时生效，对应IoT物联网平台上的数据转发动作。

----结束

7.10 设备联动

概述

设备联动指通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。例如：设置水表的电池电量阈值为小于等于20%时，上报电池电量过低的告警，用户就能及时了解设备的供电情况，以便及时更换电池。

图 7-70 设备联动架构图

若您想要进一步体验设备联动功能，可参考**设备触发告警并邮件或短信通知**。

7.10.1 云端规则

概述

当用户设置云端规则时，物联网平台会判断是否满足规则触发条件，在条件满足时，平台会执行用户预设的动作。比如：事件告警、主题通知、设备命令下发等。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 选择左侧导航栏的“规则 > 设备联动”，单击页面右上角的“创建规则”。

步骤3 参考下表参数说明，创建设备联动规则。

参数名称	说明	最佳实践
规则名称	创建的规则名称。	• **温度过高时自动关闭设备** • **设备触发告警并邮件或短信通知** • **煤气浓度过高自动打开推窗器** • **监测设备状态变化情况并发送通知**
激活规则	• 勾选：创建规则后，规则处于激活状态。 • 不勾选：创建规则后，规则处于未激活状态。	
规则类型	• 云端规则：创建的规则在云端平台执行。 • 端侧规则：创建的规则下发到端侧执行，需要端侧设备搭载有端侧规则引擎能力的SDK，详见**端侧规则**。	
生效时间	• 一直生效：没有时间限制，持续检查当前规则条件是否满足。 • 指定时间：可以选择时间段，在特定的时间检查规则条件是否满足。	
描述	对该规则的描述。	

参数名称	说明	最佳实践
触发条件	满足条件：可设置满足全部条件，或者任意一个条件，触发规则。 ● **设备属性触发**：可以将设备上报的属性作为触发条件，如：温度过高达到80℃，关闭设备。 – 选择产品：选择特定产品。 – 选择设备范围： ■ 全部设备：对选择产品下的全部设备进行条件设置。 ■ 指定设备：对选择产品下的指定设备进行条件设置。 – 选择服务：选择产品后选择对应的服务类型。 – 选择属性：选择所需的设备属性作为条件。 **说明** ■ 属性的数据类型为int、long和decimal时，支持选择多种判断符号。 ■ 属性的数据类型为string，date time，jsonObject时，判断条件仅支持相等。 – 触发机制：选择触发策略类型，建议选择重复抑制。 – 数据时效（秒）：数据有效时间。例如：设备产生数据时间为19:00，时效设为30分钟，平台收到数据时间为20:00，该情况下即使满足触发条件也不触发动作。 ● **定时触发**：可以设置规则触发的时间点，该条件一般用于周期性的触发条件，如每天7:00，关闭路灯。 **说明** 当触发条件选择“定时触发”，则动作中不能设置“发送通知”、“上报告警”、“恢复告警”的执行动作。 – 每日定时触发：可选择每日指定时间点触发。 – 按策略定时触发： ■ 时间点：可以选择规则触发的起始时间点。 ■ 重复次数：规则重复触发的次数（1~1440次）。 ■ 间隔（分钟）：在起始时间点后，重复触发规则的时间间隔（1~1440分钟）。 ● **设备状态触发**：可以将设备的上下线状态作为触发条件，如：设备离线持续时长达到5分钟，上报告警。 – 选择产品：选择特定产品。 – 选择设备范围： ■ 全部设备：对选择产品下的全部设备进行条件设置。	

参数名称	说明	最佳实践
	▪ 指定设备：对选择产品下的指定设备进行条件设置。 – 选择触发设备状态： ▪ 上线：设备由离线状态转变为在线状态。 ▪ 下线：设备由在线状态转变为离线状态。 ▪ 上下线：设备状态发生变更。 – 状态持续时长：设备状态发生变更后，新状态的持续时长，单位：分钟，可选0-60分钟。	

参数名称	说明	最佳实践
执行动作	单击“添加动作”，设置在规则触发后，需要执行的动作。 • **下发命令**：依次选择需要执行下发命令的设备、服务、命令，然后配置下发命令的参数。 • **发送通知**：选择SMN云服务对应所在区域。如果未授权，根据界面提示配置云服务访问授权，单击相应链接跳转到SMN云服务页面设置主题。 – 消息标题：给邮箱订阅者发送邮件时作为邮件主题。 – 消息类型：可从自定义和使用模板中选择。 – 消息内容：发送的自定义的消息内容。 – 模板选择：使用SMN云服务定义的模板，发送消息时将模板中变量替换成对应的参数值。设备接入定义了一些通用的模板变量，在规则触发后，以下模板变量将会替换成相应的具体值。 {ruleName}：触发的规则名称 {ruleId}：触发的规则ID {deviceId}：触发规则的设备ID {deviceName}：触发规则的设备名称 {productId}：触发规则的设备所属产品的产品ID {productName}：触发规则的设备所属产品的产品名称 {YYYY}：规则触发的年(UTC时间） {MM}：规则触发的月(UTC时间） {DD}：规则触发的日(UTC时间） {HH}：规则触发的时(UTC时间） {mm}：规则触发的分钟(UTC时间） {ss}：规则触发的秒(UTC时间） **说明** 例如，在SMN云服务创建消息模板为以下内容： 时间：{YYYY}-{MM}-{DD} {HH}:{mm}:{ss} 规则名称：{ruleName} 规则Id：{ruleId} 产品Id：{productId} 产品名称：{productName} 设备Id：{deviceId} 设备名称：{deviceName} 事件：设备上下线 则设备上线并触发规则后，收到的消息内容为以下图片所示： 	

参数名称	说明	最佳实践
	• **上报告警**：定义告警级别、告警名称、告警隔离级别和告警内容等。当触发设置的条件后，在**应用运维管理**界面产生一条对应的设备告警。 – 告警级别：可从提示、次要、重要和紧急中选择。 – 告警隔离维度：可从用户、资源空间和设备三个维度中选择，选择不同维度，上报的告警会带上不同的维度标识，如选择设备维度，上报的告警会携带设备ID作为隔离标识。 – 告警名称：上报的告警名称。 – 告警内容：上报的告警需要携带的内容。 • **恢复告警**：定义告警级别、告警名称、告警隔离级别和告警内容等，参数描述同上报告警参数描述，表示清除设备上报到平台的告警。 **说明** 在应用运维管理中，告警级别、告警名称、告警隔离级别组合起来共同标识一条告警，恢复告警时，这三个属性需要和上报告警时保持一致。	

步骤4 单击右下角“创建规则”，完成设备联动规则的创建。新创建的规则默认状态为“激活”，您可以在规则列表“状态”列，禁用规则。

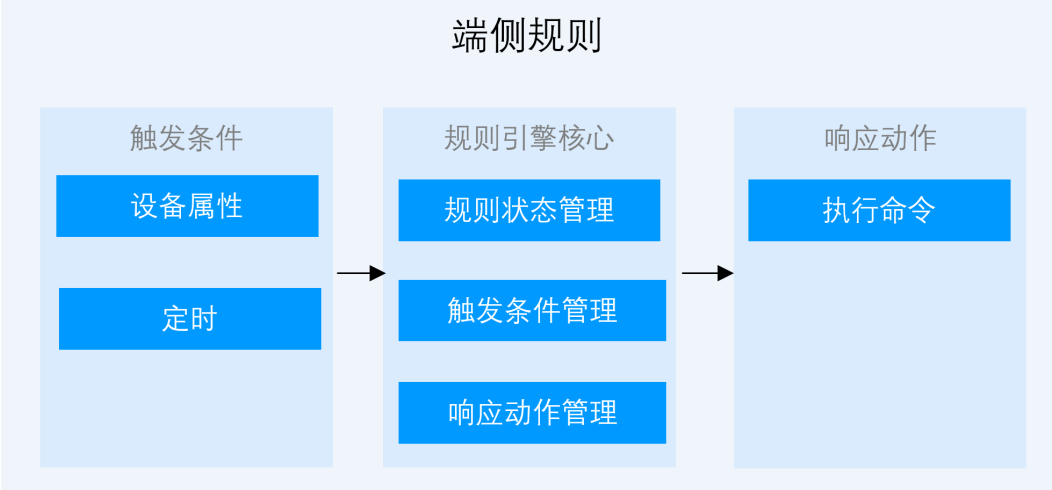
----结束

7.10.2 端侧规则

概述

在**云端规则**中，用户创建的规则的解析及执行均在云端完成，云平台需要判断条件是否满足并触发相应的设备联动操作。端侧规则是指用户在云平台创建的设备联动规则，可以下发到端侧设备，该设备上会运行端侧规则引擎，对云端下发的规则进行解析并执行。端侧规则可以在网络中断或设备无法与云端交互情况下，继续在端侧执行指定规则。端侧规则可以扩展用户应用场景，提升端侧设备运行的稳定性及执行效率等。例如：设置室内光照强度低于20时，打开灯控总开关，自动照明，实现不依赖网络设备的智能控制。

图 7-71 端侧规则架构图



相关背景概念说明可参考**基础概念**。

使用场景

公路隧道中的监控设备种类较多，数量较大，并且隧道内网络环境复杂，网络质量也不稳定。然而，在应急处理时对网络实时性要求较高，因此无法将应急设备间的联动完全依赖于云端规则处理，需要借助端侧规则引擎实现预案联动。在实施时，可以预先针对火灾、交通事故等不同情况制定相应的设备联动预案。监控人员可以根据隧道内发生的情况，一键启动设备预案，通过端侧规则引擎实现多种相关设备同步进行状态变化，从而降低对网络质量的依赖，提高整体设备联动效率。例如，当烟道温度过高时，可以联动排水阀控制器打开排水阀实现降温；当一氧化碳浓度过高时，可以联动covi设备控制风机来通风。

使用限制

- 相对云端规则，目前端侧规则的执行动作只支持命令下发。
- 设备需适配指定的SDK：Device SDK C_v1.1.2及以上版本。
- 设备需通过上述SDK提供的API，主动上报SDK版本号至华为云IoT平台。

使用说明与实例

下面以智能路灯系统为例，介绍如何使用端侧规则。

步骤1 访问**设备接入服务**。

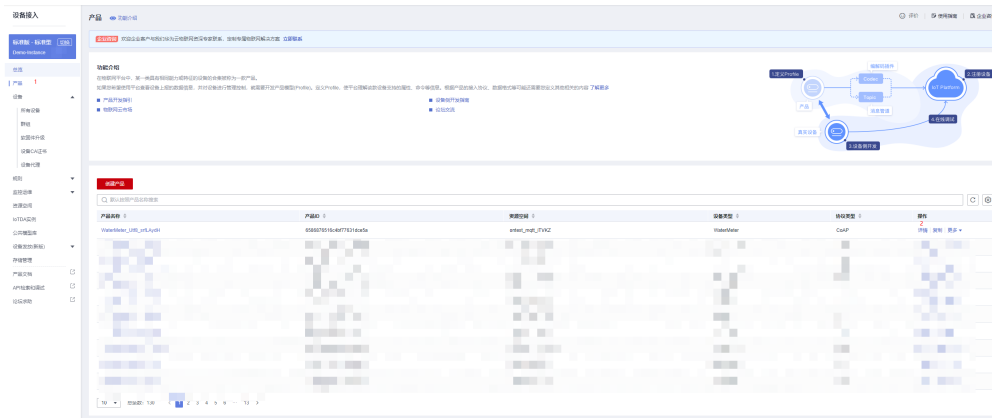
步骤2 创建产品以及新增物模型。

1. 创建产品，进入**管理控制台**，选择“产品-创建产品”，单击“设备类型”，在弹出的对话框内选择“智能路灯”，填写好产品名称后单击确定。

图 7-72 创建产品

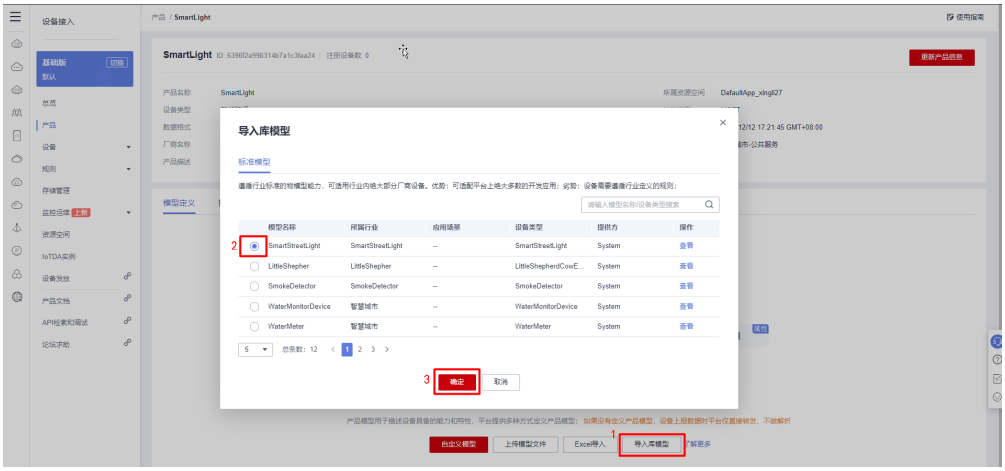
2. 创建模型，右侧选择“产品”，找到“SmartLight”，然后单击“查看”。

图 7-73 查看产品



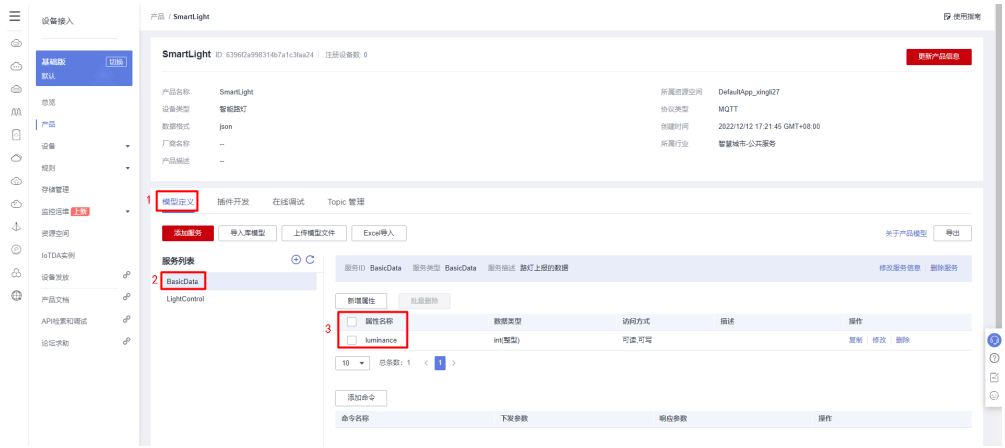
3. 进入产品信息页面后，单击“导入库模型”，选择“SmartStreetLight”，然后单击确定。

图 7-74 导入库模型



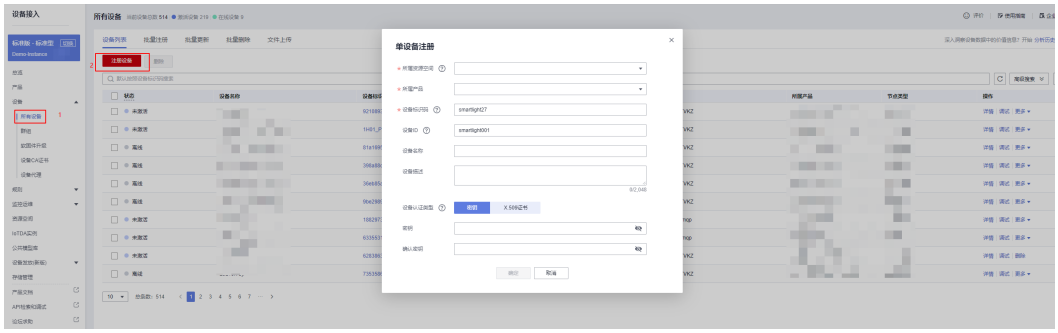
4. 单击“模型定义”，可见模型定义中包含两个服务，分别为“BasicData”和“LightControl”，单击“BasicData”可见服务详情中包含一个名为“luminance”的属性。单击“LightControl”可见其包含一个名为“switch”的命令。

图 7-75 模型定义



步骤3 在左侧选择“设备-所有设备”，单击右上角“注册设备”。选择**步骤2**创建产品选择的资源空间以及所属产品，填写设备标识码，单击“确定”。

图 7-76 创建设备



步骤4 设备创建成功，复制设备密钥，此处需要保存好设备密钥，后面需要用到。

图 7-77 设备创建成功

步骤5 创建规则。

1. 在左侧选择“规则-设备联动”，单击上方"所属资源空间"，选择设备所在产品的所属资源空间，然后单击右上角“创建规则”。

图 7-78 创建规则

2. 进入创建规则的页面，填写规则名称，选中“端侧执行”，然后选择规则执行设备，即选择将此条规则下发给哪个设备去解析执行。

图 7-79 选择端侧执行

3. 选择刚刚创建的设备“smartlight001”，选好后，单击右下角按钮“确定”。

图 7-80 选择规则执行设备

说明

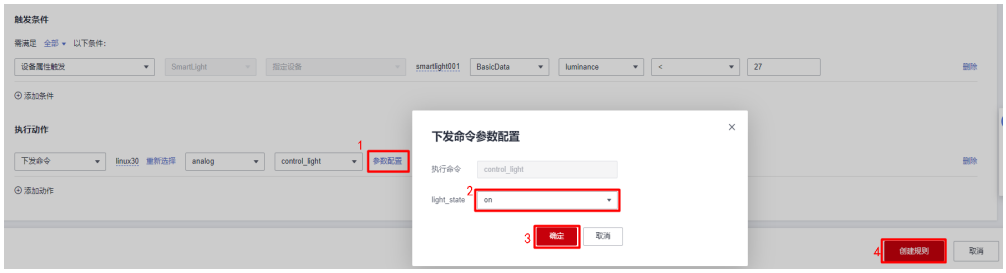
只有适配了端侧SDK的设备（目前只支持Device SDK C_v1.1.2版本），才支持创建端侧规则。

4. 单击“添加条件”，可见设备为本设备不可选择；单击“添加动作”，用户可根据需求选择指定设备，该设备可以为本设备，也可以为其他设备。

图 7-81 添加触发条件及执行动作

5. 在触发条件中配置属性触发为“luminance<= 27”，在执行动作中配置control_light指令并填写参数值，light_state为“on”。

图 7-82 设置触发条件及动作



步骤6 参考下表参数说明，创建设备联动规则。

表 7-21 参数说明

参数名称	说明
规则名称	创建的规则名称。
激活规则	勾选：创建规则后，规则处于激活状态。 不勾选：创建规则后，规则处于未激活状态。
生效时间	• 一直生效：没有时间限制，持续检查是否满足当前规则条件。 • 指定时间：可以选择时间段，在特定的时间内检查是否满足规则条件。 **说明** 由于端侧规则存储在内存当中，当设备下电时，端侧存储的规则将清空。当设备重新开机或上电时，端侧设备会从云端更新该设备的历史全量规则。
描述	对该规则的描述。

参数名称	说明
触发条件	满足条件：可设置满足全部条件，或者任意一个条件。 **说明** 当设置满足全部条件时，不能同时设置设备属性触发和定时触发，只支持同时设置多种设备属性触发。 触发类型：目前只支持设备属性触发和定时触发。 ● 设备属性触发：设备上报属性时触发。 – 选择服务：选择对应的服务类型。 – 选择属性：选择满足条件后上报数据的设备属性。 **说明** – 属性的数据类型为int和decimal时，支持选择多种判断符号。 – 属性的数据类型为string时，判断条件仅支持相等。 ● 定时触发：设置个性化的定时条件，如每日定时触发或按策略定时触发。 – 每日定时触发：可以设置规则触发的时间点，该条件一般用于周期性的触发条件，如每天7:00，关闭路灯。 – 按策略定时触发 ■ 时间点：可以选择规则触发的起始时间点。 ■ 重复次数：规则重复触发的次数（1~1440次）。 ■ 间隔（分钟）：在起始时间点后，重复触发规则的时间间隔（1~1440分钟）。
执行动作	单击“添加动作”，设置在规则触发后，需要执行的动作。 下发命令：依次选择需要执行下发命令的设备、服务、命令，然后配置下发命令的参数。

步骤7 单击右下角“创建规则”，完成端侧规则的创建。新创建的规则默认激活，用户可以在规则列表“状态”列禁用规则。

步骤8 编写设备侧代码。在支持端侧规则引擎的SDK中（目前仅支持Device C SDK），只需要用户实现属性上报及命令处理的回调函数即可。单击**这里**获取iot-device-sdk-c，按说明文档步骤操作，在**准备工作**完成之后进行以下修改：

1. 打开文件src/device_demo/device_demo.c，找到函数HandleCommandRequest。

图 7-83 命令处理

```
void HandleCommandRequest(EN_IOTA_COMMAND *command)
{
    if (command == NULL) {
        return;
    }

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), messageId %d\n",
        command->mqtt_msg_info->messageId);

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), object_device_id %s\n",
        command->object_device_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), service_id %s\n", command->service_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), command_name %s\n", command->command_name);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), paras %s\n", command->paras);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), request_id %s\n", command->request_id);
                                                                    用户可在此处实现具体命令处理
    Test_CommandResponse(command->request_id); // response command
}
```

此处为方便示例，将上述命令处理实现为：

```
printf("----- execute command----- \n");
printf("service_id: %s\n", command->service_id);
printf("command_name: %s\n", command->command_name);
printf("paras: %s\n", command->paras);
```

2. 打开文件src/device_demo/device_demo.c，找到函数TestPropertiesReport。

图 7-84 代码替换

```
void Test_PropertiesReport()
{
    const int serviceNum = 2; // reported services' totol count
    ST_IOTA_SERVICE_DATA_INFO services[serviceNum];

    // ---------------the data of service1-------------------------------
    char *service1 = "{\"Load\":\"6\",\"ImbA_strVal\":\"7\"}";

    services[0].event_time =
        GetEventTimesStamp(); // if event_time is set to NULL, the time will be the iot-platform's time.
    services[0].service_id = "parameter";
    services[0].properties = service1;

    // ---------------the data of service2-------------------------------
    char *service2 = "{\"PhV_phsA\":\"9\",\"PhV_phsB\":\"8\"}";                    替换此处代码

    services[1].event_time = NULL;
    services[1].service_id = "analog";
    services[1].properties = service2;

    int messageId = IOTA_PropertiesReport(services, serviceNum, 0, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_PropertiesReport() failed, messageId %d\n", messageId);
    }

    MemFree(&services[0].event_time);
}
```

替换代码为：

```
const int serviceNum = 1; // reported services' total count
ST_IOTA_SERVICE_DATA_INFO services[serviceNum];

#define MAX_BUFFER_LEN 70
char propsBuffer[MAX_BUFFER_LEN];
//此处为获取温度值的示例，从真实传感器获取数值需要由用户实现
if(sprintf_s(propsBuffer, sizeof(propsBuffer), "{\"luminance\": %d}", 20) == -1){
printf("can't create string of properties\n");
return;
}
```

```
services[0].event_time = GetEventTimesStamp(); // if event_time is set to NULL, the time will be the
iot-platform's time.
services[0].service_id = "BasicData";
services[0].properties = propsBuffer;

int messageId = IOTA_PropertiesReport(services, serviceNum, 0, NULL);
if (messageId != 0) {
printf("report properties failed, messageId %d\n", messageId);
}
free(services[0].event_time);
```

3. 编译sdk并执行。观察输出，可以看到对应的command。

```
----- execute command-----
service_id: BasicData
command_name: control_light
paras: {
        "light_state":       "on"
}
```

以上日志仅为一个示例。用户需要实现上述1中具体的命令处理代码。

跨设备执行命令时，需要设备间具备相互通信的能力。同时由于用户使用的通信协议可能不同，如WiFi，BLE，ZigBee等，所以要调用函数IOTA_SetDeviceRuleSendMsgCallback注册自定义发送函数。

Demo中已默认注册HandleDeviceRuleSendMsg，用户需要自行在预置函数HandleDeviceRuleSendMsg实现消息发送。执行命令的设备接收消息后，则需要自行实现命令解析和执行。

图 7-85 解析命令并执行

```
huaweicloud-iot-device-sdk-c > src > device_demo > C device_demo.c > SetMyCallbacks()
1052
1053  int HandleDeviceRuleSendMsg(char *deviceId, char *message)
1054  {
1055      PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceRuleSendMsg(), deviceId is %s, the message is %s",
1056          deviceId, message);
1057      return 0;
1058  }
1059
1060  // ----------------------------------------------------------------------
1061
1062  > void SetAuthConfig() ···
1082
1083
1084  void SetMyCallbacks()
1085  {
1086      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_SUCCESS, HandleConnectSuccess);
1087      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_FAILURE, HandleConnectFailure);
1088
1089      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_SUCCESS, HandleDisConnectSuccess);
1090      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_FAILURE, HandleDisConnectFailure);
1091      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECTION_LOST, HandleConnectionLost);
1092
1093      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS, HandleSubscribesuccess);
1094      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE, HandleSubscribeFailure);
1095
1096      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_SUCCESS, HandlePublishSuccess);
1097      IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_FAILURE, HandlePublishFailure);
1098
1099      IOTA_SetMessageCallback(HandleMessageDown);
1100      IOTA_SetUserTopicMsgCallback(HandleUserTopicMessageDown);
1101      IOTA_SetCmdCallback(HandleCommandRequest);
1102      IOTA_SetPropSetCallback(HandlePropertiesSet);
1103      IOTA_SetPropGetCallback(HandlePropertiesGet);
1104      IOTA_SetEventCallback(HandleEventsDown);
1105      IOTA_SetShadowGetCallback(HandleDeviceShadowRsp);
1106      IOTA_SetDeviceRuleSendMsgCallback(HandleDeviceRuleSendMsg);
1107  }
```

----结束

8 监控运维

8.1 设备消息跟踪

在设备鉴权、命令下发、数据上报、平台数据转发等业务场景中出现故障时，物联网平台可以通过消息跟踪功能进行快速的故障定位和原因分析。目前物联网平台支持NB-IoT设备和MQTT设备的消息跟踪，单个用户下，最多支持同时进行跟踪的设备数上限为10。

操作步骤

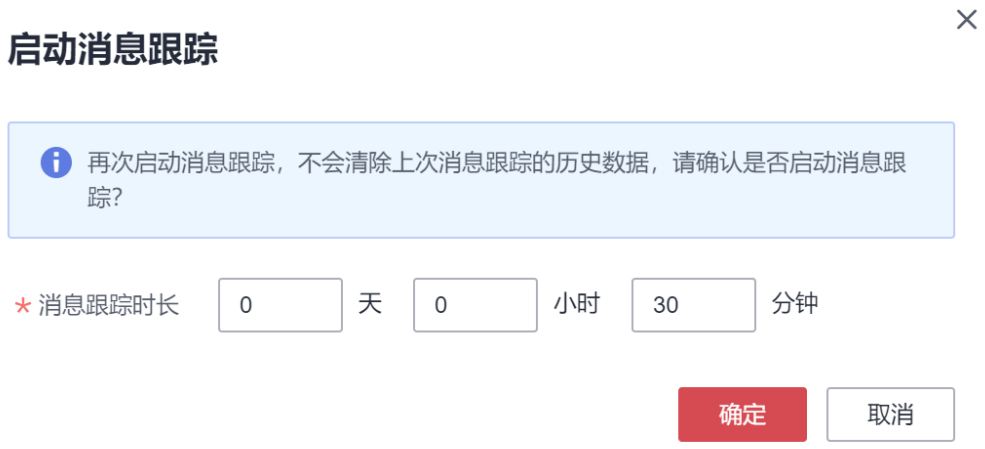
步骤1 访问**设备接入服务**，单击“管理控制台 ”进入设备接入控制台。

步骤2 在左侧导航栏选择“设备 > 所有设备”界面。

步骤3 通过检索条件，快速找到需要跟踪的设备，并单击"查看"按钮，进入设备详情。

步骤4 在“设备详情”页，单击“消息跟踪”页签，单击“启动消息跟踪”按钮，并设置设备的消息跟踪时间，如下图所示，消息跟踪时长表示从启动消息跟踪功能开始到结束消息跟踪的总时长，在设置的时间段内进行消息跟踪，修改跟踪配置后，会以修改后时间为准。

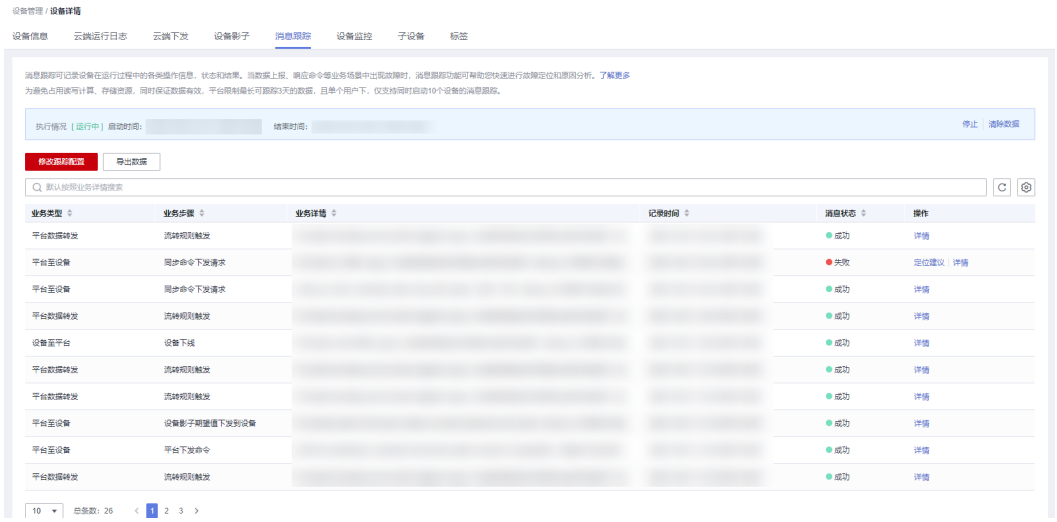
图 8-1 启动消息跟踪



步骤5 您可以在“设备详情 > 消息跟踪”页面，查看正在进行消息跟踪的业务（如果单击“停止”，则停止跟踪设备）。

如果跟踪的结果中数据量较多时，您可以根据“消息状态”、“消息类型”和“记录时间”进行过滤；如果对跟踪结果数据需要做进一步的分析，用户可以通过“导出数据”功能，将跟踪的结果数据导出，以便物联网平台运维人员做进一步的分析。

图 8-2 查看消息跟踪数据



若消息状态显示“失败”，可以通过单击“定位建议”按钮，查看失败场景的定位建议并进行问题的定位分析，通过单击“查看”按钮查看跟踪结果的详情信息。

----结束

说明

数据上报过程中，可以通过在topic后面携带?request_id={request_id}来指定request_id，比如属性上报:$oc/devices/{device_id}/sys/properties/report?request_id={request_id}，如果不指定，则平台自动生成request_id，标识此次请求。

8.2 查看报表

物联网平台为用户提供了丰富的报表功能，能够将数据直观地呈现出来。

总览页面的报表统计实例维度的数据，报表数据有效时间为一个月。在控制台左侧导航栏，选择"总览"进入总览页面，报表名称及功能详细说明如下表所示。

表 8-1 总览页报表

报表名称	报表说明	数据刷新频率	可选择时间范围
注册设备数	统计当前实例的总注册设备数。	每小时	小时、日、月

报表名称	报表说明	数据刷新频率	可选择时间范围
同时在线设备数	统计当前实例的同时在线设备数。小时维度：统计每小时的最大在线设备数，天维度：统计每天的最大在线设备数。	每小时	小时、日、月
设备消息数	统计当前实例的设备上下行消息数。设备上行消息数包含消息、属性、事件上报。设备下行消息数包含消息下发、属性设置、属性查询、命令下发。	每小时	小时、日、月
用户消息数	统计当前实例的用户消息数量。 包含设备发送到云端的数据+通过云端下发给设备的数据+数据转发的数据(其中低于设备发送到云端的消息量部分不计算，超过部分参与计算)。	每小时	小时、日、月

监控运维下统计报表中默认统计实例维度的数据，同时支持查看不同资源空间下的统计数据，单击报表右上角"+"可选择查看不同资源空间的数据，报表数据有效时间为一个月。在控制台左侧导航栏，选择“监控运维 > 统计报表”进入报表页面，报表名称及功能详细说明如下表所示。

表 8-2 监控运维页报表

报表名称	报表说明	数据刷新频率	可选择时间范围
设备连接状态	统计当前实例/资源空间下，设备的总数、在线数、在线率、未激活数、未激活率、离线数、离线率、异常数和异常率。	每小时	-
设备消息	设备上报消息数：统计设备上报到平台的消息数。 下行消息数：统计平台下发到设备的消息数。	每小时	小时、日
设备总趋势	统计当前实例/资源空间下设备总数和设备在线数的趋势。	每小时	小时、日
设备在线趋势	统计当前实例/资源空间下设备在线和离线情况，以百分比形式呈现。	每小时	小时、日
设备数量统计	统计当前实例/资源空间下未激活设备数、异常设备数和离线设备数。	每小时	小时、日
软件升级状态	统计当前实例/资源空间下，历史上所有设备的软件升级任务，执行成功和失败的设备数量。	每小时	-
固件升级状态	统计当前实例/资源空间下，历史上所有设备的固件升级任务，执行成功和失败的设备数量。	每小时	-
设备配置状态	统计当前实例/资源空间下，历史上所有设备的设备配置更新任务，执行成功和失败的设备数量。	每小时	-

更多报表，可登录**应用运维管理**控制台，选择监控 >云服务监控 >物联网>IoTDA，查看报表信息。目前，应用运维管理支持在实例或资源空间维度下查看设备接入服务的监控信息：

表 8-3 仪表盘

报表名称	报表说明	数据刷新频率	可选择时间范围
设备状态	统计当前实例/资源空间下设备在线、离线、异常及未激活设备数。	每10分钟	一小时、六小时、十二小时、一天、七天
设备总趋势	统计当前实例/资源空间下设备总数、在线设备数及离线设备数趋势。	每10分钟	一小时、六小时、十二小时、一天、七天
数据流转趋势	统计当前实例/资源空间下AMQP流转次数及HTTP推送流转次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天
数据上报趋势	统计当前实例/资源空间下NB数据上报总数、MQTT事件上报总数、MQTT属性上报总数及MQTT消息上报总数趋势。	每分钟	一小时、六小时、十二小时、一天、七天

表 8-4 指标

报表名称	报表说明	数据刷新频率	可选择时间范围
设备总数	统计当前实例/资源空间下设备总数、在线设备数、离线设备数、异常设备数及未激活设备数趋势。	每10分钟	一小时、六小时、十二小时、一天、七天
NB数据上报数	统计当前实例/资源空间下NB数据上报总数、NB数据上报成功次数及NB数据上报失败次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天
MQTT事件上报数	统计当前实例/资源空间下MQTT事件上报总数、MQTT事件上报成功次数及MQTT事件上报失败次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天

报表名称	报表说明	数据刷新频率	可选择时间范围
MQTT属性上报数	统计当前实例/资源空间下MQTT属性上报总数、MQTT属性上报成功次数及MQTT属性上报失败次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天
MQTT消息上报数	统计当前实例/资源空间下MQTT消息上报总数、MQTT消息上报成功次数及MQTT消息上报失败次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天
AMQP流转次数	统计当前实例/资源空间下AMQP流转次数、AMQP流转成功次数及AMQP流转失败次数趋势。	每分钟	一小时、六小时、十二小时、一天、七天
FunctionGraph流转次数	统计当前实例/资源空间下FunctionGraph流转次数、FunctionGraph流转成功次数及FunctionGraph流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
MRS Kafka流转次数	统计当前实例/资源空间下MRS Kafka流转次数、MRS Kafka流转成功次数及MRS Kafka流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
MQTT流转次数	统计当前实例/资源空间下MQTT流转次数、MQTT流转成功次数及MQTT流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
MySql流转次数	统计当前实例/资源空间下MySql流转次数、MySql流转成功次数及MySql流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
InfluxDB流转次数	统计当前实例/资源空间下InfluxDB流转次数、InfluxDB流转成功次数及InfluxDB流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
HTTP推送流转次数	统计当前实例/资源空间下HTTP推送流转次数、HTTP推送流转成功次数及HTTP推送流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
OBS流转次数	统计当前实例/资源空间下OBS流转次数、OBS流转成功次数及OBS流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天

报表名称	报表说明	数据刷新频率	可选择时间范围
DMS Kafka流转次数	统计当前实例/资源空间下DMS Kafka流转次数、DMS Kafka流转成功次数及DMS Kafka流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
DIS流转次数	统计当前实例/资源空间下DIS流转次数、DIS流转成功次数及DIS流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
ROMA流转次数	统计当前实例/资源空间下ROMA流转次数、ROMA流转成功次数及ROMA流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
LTS流转次数	统计当前实例/资源空间下LTS流转次数、LTS流转成功次数及LTS流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
BCS华为云区块链流转次数	统计当前实例/资源空间下BCS华为云区块链流转次数、BCS华为云区块链流转成功次数及BCS华为云区块链流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
BCS-Hyperledger Fabric增强版流转次数	统计当前实例/资源空间下BCS-Hyperledger Fabric增强版流转次数、BCS-Hyperledger Fabric增强版流转成功次数及BCS-Hyperledger Fabric增强版流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天
MongoDB流转次数	统计当前实例/资源空间下MongoDB流转次数、MongoDB流转成功次数及MongoDB流转失败次数。	每分钟	一小时、六小时、十二小时、一天、七天

如果需要通过API获取报表数据，可以参考AOM的**查询监控数据**接口。其中关于IoTDA指标自定义参数，请参考表5和表6，表5中的”指标维度名称”对应该接口参数：metrics[].metric.dimensions[].name，”指标维度取值”对应该接口参数：metrics[].metric.dimensions[].value。表6中的namespace对应该接口参数：metrics[].metric.namespace，metricName对应该接口参数：metrics[].metric.metricName。

表 8-5 维度定义

指标维度名称	指标维度取值
app	资源空间ID

指标维度名称	指标维度取值
instance	实例ID
taskType	任务类型：软件升级状态（softwareUpgrade）、固件升级状态（firmwareUpgrade）、设备配置状态（deviceConfig）

表 8-6 指标名称和指标命名空间定义

报表名称	namespace	metricName
设备总数	IoTDA.DEVICE_STATUS	设备总数：iotda_device_status_totalCount
		在线设备数：iotda_device_status_onlineCount
		离线设备数：iotda_device_status_offlineCount
		异常设备数：iotda_device_status_abnormalCount
		未激活设备数：iotda_device_status_inactiveCount
NB数据上报数	IoTDA.NB_DATA_REPORT	NB数据上报总数：iotda_south_dataReport_totalCount
		NB数据上报成功次数：iotda_south_dataReport_successCount
		NB数据上报失败次数：iotda_south_dataReport_failedCount
MQTT事件上报数	IoTDA.EVENT_UP	MQTT事件上报总数：iotda_south_eventUp_totalCount
		MQTT事件上报成功次数：iotda_south_eventUp_successCount
		MQTT事件上报失败次数：iotda_south_eventUp_failedCount
MQTT属性上报数	IoTDA.PROPERTIES_REPORT	MQTT属性上报总数：iotda_south_propertiesReport_totalCount
		MQTT属性上报成功次数：iotda_south_propertiesReport_successCount

报表名称	namespace	metricName
		MQTT属性上报失败次数：iotda_south_propertiesReport_failedCount
MQTT消息上报数	IoTDA.MESSAGE_UP	MQTT消息上报总数：iotda_south_messageUp_totalCount
		MQTT消息上报成功次数：iotda_south_messageUp_successCount
		MQTT消息上报失败次数：iotda_south_messageUp_failedCount
AMQP流转次数	IoTDA.AMQP_FORWARDING	AMQP流转次数：iotda_amqp_forwarding_totalCount
		AMQP流转成功次数：iotda_amqp_forwarding_successCount
		AMQP流转失败次数：iotda_amqp_forwarding_failedCount
FunctionGraph流转次数	IoTDA.FUNCTIONGRAPH_FORWARDING	FunctionGraph流转次数：iotda_functionGraph_forwarding_totalCount
		FunctionGraph流转成功次数：iotda_functionGraph_forwarding_successCount
		FunctionGraph流转失败次数：iotda_functionGraph_forwarding_failedCount
MRS Kafka流转次数	IoTDA.MRS_KAFKA_FORWARDING	MRS Kafka流转次数：iotda_mrsKafka_forwarding_totalCount
		MRS Kafka流转成功次数：iotda_mrsKafka_forwarding_successCount
		MRS Kafka流转失败次数：iotda_mrsKafka_forwarding_failedCount
MQTT流转次数	IoTDA.MQTT_FORWARDING	Mqtt流转次数：iotda_mqtt_forwarding_totalCount

报表名称	namespace	metricName
		Mqtt流转成功次数：iotda_mqtt_forwarding_successCount
		Mqtt流转失败次数：iotda_mqtt_forwarding_failedCount
MySql流转次数	IoTDA.MYSQL_FORWARDING	MySql流转次数：iotda_mysql_forwarding_totalCount
		MySql流转成功次数：iotda_mysql_forwarding_successCount
		MySql流转失败次数：iotda_mysql_forwarding_failedCount
InfluxDB流转次数	IoTDA.INFLUXDB_FORWARDING	InfluxDB流转次数：iotda_influxDB_forwarding_totalCount
		InfluxDB流转成功次数：iotda_influxDB_forwarding_successCount
		InfluxDB流转失败次数：iotda_influxDB_forwarding_failedCount
HTTP推送流转次数	IoTDA.HTTP_FORWARDING	HTTP推送流转次数：iotda_http_forwarding_totalCount
		HTTP推送流转成功次数：iotda_http_forwarding_successCount
		HTTP推送流转失败次数：iotda_http_forwarding_failedCount
OBS流转次数	IoTDA.OBS_FORWARDING	OBS流转次数：iotda_obs_forwarding_totalCount
		OBS流转成功次数：iotda_obs_forwarding_successCount
		OBS流转失败次数：iotda_obs_forwarding_failedCount
DMS Kafka流转次数	IoTDA.DMS_KAFKA_FORWARDING	DMS Kafka流转次数：iotda_dmsKafka_forwarding_totalCount

报表名称	namespace	metricName
		DMS Kafka流转成功次数：iotda_dmsKafka_forwarding_successCount
		DMS Kafka流转失败次数：iotda_dmsKafka_forwarding_failedCount
DIS流转次数	IoTDA.DIS_FORWARDING	DIS流转次数：iotda_dis_forwarding_totalCount
		DIS流转成功次数：iotda_dis_forwarding_successCount
		DIS流转失败次数：iotda_dis_forwarding_failedCount
ROMA流转次数	IoTDA.ROMA_FORWARDING	ROMA流转次数：iotda_roma_forwarding_totalCount
		ROMA流转成功次数：iotda_roma_forwarding_successCount
		ROMA流转失败次数：iotda_roma_forwarding_failedCount
LTS流转次数	IoTDA.LTS_FORWARDING	LTS流转次数：iotda_lts_forwarding_totalCount
		LTS流转成功次数：iotda_lts_forwarding_successCount
		LTS流转失败次数：iotda_lts_forwarding_failedCount
BCS华为云区块链流转次数	IoTDA.BCS_HW_FORWARDING	BCS华为云区块链流转次数：iotda_bcshw_forwarding_totalCount
		BCS华为云区块链流转成功次数：iotda_bcshw_forwarding_successCount
		BCS华为云区块链流转失败次数：iotda_bcshw_forwarding_failedCount
BCS-Hyperledger Fabric增强版流转次数	IoTDA.BCS_FABRIC_FORWARDING	BCS-Hyperledger Fabric增强版流转次数：iotda_bcsfabric_forwarding_totalCount

报表名称	namespace	metricName
		BCS-Hyperledger Fabric增强版流转成功次数：iotda_bcsfabric_forwarding_successCount
		BCS-Hyperledger Fabric增强版流转失败次数：iotda_bcsfabric_forwarding_failedCount
MongoDB流转次数	IoTDA.MONGODB_FORWARDING	MongoDB流转次数：iotda_mongodb_forwarding_totalCount
		MongoDB流转成功次数：iotda_mongodb_forwarding_successCount
		MongoDB流转失败次数：iotda_mongodb_forwarding_failedCount
软固件升级/远程配置	AOM.IoTDA	成功数量：iotda_batchtask_success_count
		失败数量：iotda_batchtask_failure_count

8.3 告警管理

当物联网平台监控到满足用户通过规则设置的告警触发条件时或设备消息上行的速度超过平台预设的阈值，平台就会上报告警。用户需要密切关注告警并及时进行处理，确保设备的正常运行。

告警分为规则类告警、设备流控类告警和自定义指标告警。

- 规则类告警：如果用户在控制台上设置设备联动类的**规则引擎**时，定义了响应动作为上报告警，且定义了告警属性、告警级别等，则当满足触发条件时，平台就会上报告警。例如：智能水表设备3天未上报数据，可能存在水表设备发生故障导致，平台会产生对应的告警，维护人员可通过告警信息找到对应告警的水表设备，进行快速定位维修。
- 系统告警：用户的某些资源达到用户配额的上限，如当设备数达到用户的配额上限，IoTDA平台就会上报系统告警至AOM，这类告警无需用户配置，平台自动触发，但需要配置通知规则。具体系统告警见表1：

表 8-7 系统告警

告警名称	告警解释
单设备MQTT消息流控	单个MQTT设备连接每秒上行数据的流量大小超过设定的阈值（默认值为3K/秒）时，会进入设备连接流控状态，平台会上报告警。
设备上行消息超租户流控阈值	用户的设备上行消息/建链(根据告警的资源的API名称区别，PUBLISH为消息上行，CONNECT为建链，BANDWIDTH为带宽)的速率之和超过用户的阈值。上行消息基础版默认为500/秒，建链基础版默认为100/秒，标准版和企业版请参考**产品规格说明**，超出部分将会被流控，同时触发告警。
用户设备数达到阈值	用户注册设备数达到阈值，当用户注册设备数达到实例阈值（基础版50000，标准版/企业版请查看**产品规格说明**，一般为在线设备数的20倍）的80%和100%时会触发告警。
用户在线设备数达到阈值	用户在线设备数达到阈值，当用户同时在线设备数达到阈值（标准版/企业版请查看**产品规格说明**，与购买单元数量相关）的80%和100%时会触发告警，超过阈值后会拒绝设备接入，告警一小时触发一次。
网关下子设备数达到阈值	用户网关下子设备数达到阈值，当用户单个网关下子设备数达到阈值的80%和100%时会触发告警。
联动规则触发并发限制阈值	联动规则触发并发限制阈值，联动规则每秒触发的规则数超过用户阈值（基础版/标准版为10/秒，企业版为100/秒），超出部分将会被流控，同时触发告警，该告警一天仅会触发一次。
租户调用接口达到流控阈值	租户调用接口达到流控阈值，租户调用接口TPS超过阈值（具体API无特殊说明的默认限制50/s。单个账号调用API的每秒最大次数：基础版/标准版为100/s），超出部分将会被流控，同时触发告警，该告警一天仅会触发一次。
数据转发目标被列入黑名单	数据转发失败次数达到一定数量（默认10次），当前转发目标被拉入黑名单后触发告警。

- 自定义指标告警：用户可以登录**应用运维管理**控制台配置自定义指标阈值告警。当前支持的指标如下：

表 8-8 自定义告警指标

统计指标	指标名称
设备总数	iotda_device_status_totalCount
在线设备数	iotda_device_status_onlineCount
离线设备数	iotda_device_status_offlineCount

统计指标	指标名称
异常设备数	iotda_device_status_abnormalCount
未激活设备数	iotda_device_status_inactiveCount
激活设备数	iotda_device_status_activeCount
累计在线设备数	iotda_device_status_dailyOnlineCount
NB数据上报总数	iotda_south_dataReport_totalCount
NB数据上报失败次数	iotda_south_dataReport_failedCount
MQTT事件上报总数	iotda_south_eventUp_totalCount
MQTT事件上报成功次数	iotda_south_eventUp_successCount
MQTT事件上报失败次数	iotda_south_eventUp_failedCount
MQTT属性上报总数	iotda_south_propertiesReport_totalCount
MQTT属性上报成功次数	iotda_south_propertiesReport_successCount
MQTT属性上报失败次数	iotda_south_propertiesReport_failedCount
MQTT消息上报总数	iotda_south_messageUp_totalCount
MQTT消息上报成功次数	iotda_south_messageUp_successCount
MQTT消息上报失败次数	iotda_south_messageUp_failedCount
AMQP流转次数	iotda_amqp_forwarding_totalCount
AMQP流转成功次数	iotda_amqp_forwarding_successCount
AMQP流转失败次数	iotda_amqp_forwarding_failedCount
FunctionGraph流转次数	iotda_functionGraph_forwarding_totalCount
FunctionGraph流转成功次数	iotda_functionGraph_forwarding_successCount
FunctionGraph流转失败次数	iotda_functionGraph_forwarding_failedCount
MRS Kafka流转次数	iotda_mrsKafka_forwarding_totalCount
MRS Kafka流转成功次数	iotda_mrsKafka_forwarding_successCount
MRS Kafka流转失败次数	iotda_mrsKafka_forwarding_failedCount
Mqtt流转次数	iotda_mqtt_forwarding_totalCount
Mqtt流转成功次数	iotda_mqtt_forwarding_successCount
Mqtt流转失败次数	iotda_mqtt_forwarding_failedCount
MySql流转次数	iotda_mysql_forwarding_totalCount
MySql流转成功次数	iotda_mysql_forwarding_successCount

统计指标	指标名称
MySql流转失败次数	iotda_mysql_forwarding_failedCount
InfluxDB流转次数	iotda_influxDB_forwarding_totalCount
InfluxDB流转成功次数	iotda_influxDB_forwarding_successCount
InfluxDB流转失败次数	iotda_influxDB_forwarding_failedCount
HTTP推送流转次数	iotda_http_forwarding_totalCount
HTTP推送流转成功次数	iotda_http_forwarding_successCount
HTTP推送流转失败次数	iotda_http_forwarding_failedCount
OBS流转次数	iotda_obs_forwarding_totalCount
OBS流转成功次数	iotda_obs_forwarding_successCount
OBS流转失败次数	iotda_obs_forwarding_failedCount
DMS Kafka流转次数	iotda_dmsKafka_forwarding_totalCount
DMS Kafka流转成功次数	iotda_dmsKafka_forwarding_successCount
DMS Kafka流转失败次数	iotda_dmsKafka_forwarding_failedCount
DIS流转次数	iotda_dis_forwarding_totalCount
DIS流转成功次数	iotda_dis_forwarding_successCount
DIS流转失败次数	iotda_dis_forwarding_failedCount
ROMA流转次数	iotda_roma_forwarding_totalCount
ROMA流转成功次数	iotda_roma_forwarding_successCount
ROMA流转失败次数	iotda_roma_forwarding_failedCount
LTS流转次数	iotda_lts_forwarding_totalCount
LTS流转成功次数	iotda_lts_forwarding_successCount
LTS流转失败次数	iotda_lts_forwarding_failedCount

配置步骤:

a. 进入登录**应用运维管理**控制台，在左侧导航栏中选择“告警 > 告警行动规则”，单击“创建告警行动规则”按钮，填写对应的参数后，完成告警行动规则的创建。

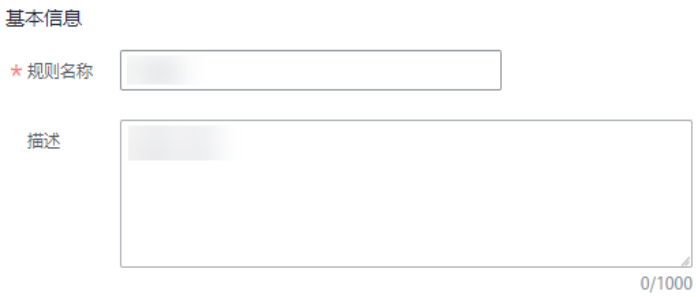
图 8-3 创建告警行动规则

b. 在左侧导航栏中选择“告警 > 告警规则”，单击右上角的“添加告警”，进入“创建告警规则”页面。

c. 设置阈值规则

i. 设置告警基本信息：在“规则名称”文本框中输入阈值规则名称，并根据需要填写规则的描述信息。

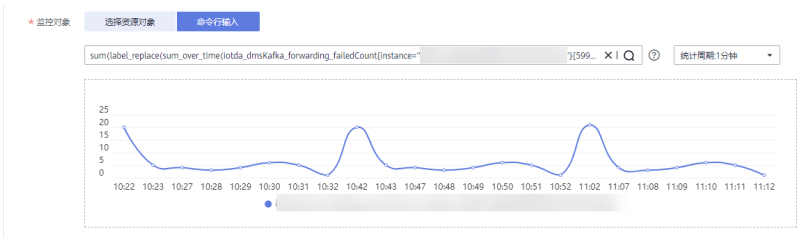
图 8-4 设置告警基本信息



ii. 设置告警规则的详细信息。

1) 设置“规则类型”为“阈值规则”。
2) 设置“监控对象”为“命令行输入”，并输入相应的命令。

图 8-5 设置监控对象



说明

命令行输入使用Prometheus格式命令，如需查看Prometheus格式命令行的详细说明，请将光标移至搜索框后的 处，单击“**了解更多**”。

例如查询实例A下的“DMS Kafka流转失败次数”，可输入如下命令：
sum(label_replace(sum_over_time(**iotda_dmsKafka_forwarding_failedCount**{instance="**实例A的InstanceId**"}[59999ms]),"__name__","**iotda_dmsKafka_forwarding_failedCount**","","")by(__name__,instance)

其中**iotda_dmsKafka_forwarding_failedCount**为对应的指标名称，可以从**表2**中获取。

3) 设置“告警条件”为“自定义创建”，在条件中可配置统计周期、连续周期、阈值条件等触发条件参数，具体参数说明如**表3**。

图 8-6 设置告警条件

以上图为例，即为在3个统计周期内，总数均大于10，则产生一个次要告警。

表 8-9 告警条件参数说明

参数类别	参数名称	参数说明
触发条件	统计周期	指标数据按照所设置的统计周期进行聚合。默认只统计一个周期，最多可统计5个周期指标数据。
	连续周期	连续多少个周期满足阈值条件后，发送阈值告警。
	统计方式	指标数据按照所设置的统计方式进行聚合，包括：平均值、最小值、最大值、总计、样本个数。
	阈值条件	阈值告警的触发条件，由判断条件（>=、<=、>、<）和阈值组成。例如，阈值条件设置为“>85”，表示指标的实际值大于已设置的阈值85时，生成阈值告警。
	告警级别	阈值告警的级别，包括：紧急、重要、次要、提示。

参数类别	参数名称	参数说明
高级设置	告警恢复	监控周期内监控对象不满足触发条件时，则恢复告警。默认只监控一个周期，最多可监控5个周期指标数据。
	无数据处理	监控周期内无指标数据产生或指标数据不足时系统的处理方式，根据业务需要启动或者关闭。 默认只监控一个周期，最多可监控5个周期指标数据。 系统处理方式包括：告警、数据不足并发送事件、保持上一个状态、正常。

iii. 设置告警通知

1) 设置告警方式为“直接告警”。
2) 行动规则选择**步骤1**创建的行动规则。
3) 开启“告警恢复通知”开关。

图 8-7 设置告警通知

说明

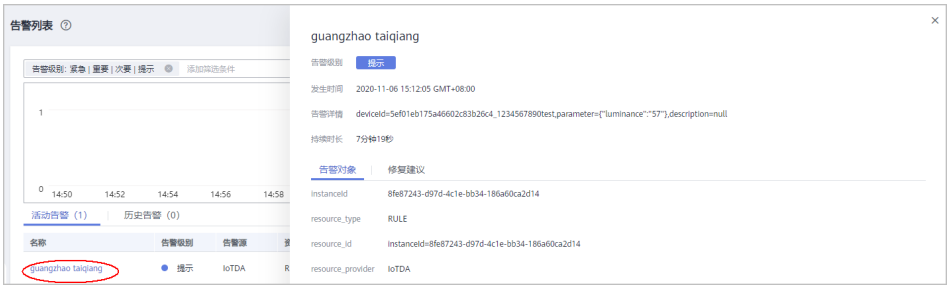
如果需要使用“告警降噪”功能，可参考**告警降噪说明**。

- 查看告警信息

  用户可以使用应用运维管理服务，查看告警。该服务支持查看最近15天的告警信息。

  a. 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。
  b. 在左侧导航栏选择“监控运维>当前告警”，单击“前往AOM”，跳转至应用运维管理服务，查看设备接入服务的告警信息。
  c. 单击目标告警，查看告警详情。

图 8-8 查看告警详情



d. 清除告警。当设备故障解除时，可在告警列表中单击目标告警所在“操作”列中的  。

更多详细内容，请参考**查看告警**。

8.4 查看审计日志

操作场景

用户进入云审计服务创建管理类追踪器后，系统开始记录云服务资源的操作。在创建数据类追踪器后，系统开始记录用户对OBS桶中数据的操作。云审计服务管理控制台会保存最近7天的操作记录。

本节介绍如何在云审计服务管理控制台查看或导出最近7天的操作记录：

- **在新版事件列表查看审计事件**
- **在旧版事件列表查看审计事件**

使用限制

- 单账号跟踪的事件可以通过云审计控制台查询。多账号的事件只能在账号自己的事件列表页面去查看，或者到组织追踪器配置的OBS桶中查看，也可以到组织追踪器配置的CTS/system日志流下面去查看。
- 用户通过云审计控制台只能查询最近7天的操作记录。如果需要查询超过7天的操作记录，您必须配置转储到对象存储服务(OBS)，才可在OBS桶里面查看历史文件。否则，您将无法追溯7天以前的操作记录。
- 云上操作后，1分钟内可以通过云审计控制台查询管理类事件操作记录，5分钟后才可通过云审计控制台查询数据类事件操作记录。

在新版事件列表查看审计事件

步骤1 登录管理控制台。

步骤2 单击左上角  ，选择“管理与监管 > 云审计服务 CTS”，进入云审计服务页面。

步骤3 单击左侧导航树的“事件列表”，进入事件列表信息页面。

步骤4 事件列表支持通过高级搜索来查询对应的操作事件，您可以在筛选器组合一个或多个筛选条件：

- 事件名称：输入事件的名称。
- 事件ID：输入事件ID
- 资源名称：输入资源的名称，当该事件所涉及的云资源无资源名称或对应的API接口操作不涉及资源名称参数时，该字段为空。
- 资源ID：输入资源ID，当该资源类型无资源ID或资源创建失败时，该字段为空。
- 云服务：在下拉框中选择对应的云服务名称。
- 资源类型：在下拉框中选择对应的资源类型。
- 操作用户：在下拉框中选择一个或多个具体的操作用户。
- 事件级别：可选项为“normal”、“warning”、“incident”，只可选择其中一项。

  - normal：表示操作成功。
  - warning：表示操作失败。
  - incident：表示比操作失败更严重的情况，例如引起其他故障等。
- 时间范围：可选择查询最近1小时、最近1天、最近1周的操作事件，也可以自定义最近1周内任意时间段的操作事件。

步骤5 在事件列表页面，您还可以导出操作记录文件、刷新列表、设置列表展示信息等。

1. 在搜索框中输入任意关键字，单击 按钮，可以在事件列表搜索符合条件的数据。
2. 单击“导出”按钮，云审计服务会将查询结果以.xlsx格式的表格文件导出，该.xlsx文件包含了本次查询结果的所有事件，且最多导出5000条信息。
3. 单击按钮，可以获取到事件操作记录的最新信息。
4. 单击按钮，可以自定义事件列表的展示信息。启用表格内容折行开关，可让表格内容自动折行，禁用此功能将会截断文本，默认停用此开关。

步骤6 关于事件结构的关键字段详解，请参见**事件结构**和**事件样例**。

步骤7 （可选）在新版事件列表页面，单击右上方的“返回旧版”按钮，可切换至旧版事件列表页面。

----结束

在旧版事件列表查看审计事件

步骤1 登录管理控制台。

步骤2 单击左上角 ，选择“管理与监管 > 云审计服务 CTS”，进入云审计服务页面。

步骤3 单击左侧导航树的“事件列表”，进入事件列表信息页面。

步骤4 用户每次登录云审计控制台时，控制台默认显示新版事件列表，单击页面右上方的“返回旧版”按钮，切换至旧版事件列表页面。

步骤5 事件列表支持通过筛选来查询对应的操作事件。当前事件列表支持四个维度的组合查询，详细信息如下：

- 事件类型、事件来源、资源类型和筛选类型，在下拉框中选择查询条件。
  - 筛选类型按资源ID筛选时，还需手动输入某个具体的资源ID。
  - 筛选类型按事件名称筛选时，还需选择某个具体的事件名称。
  - 筛选类型按资源名称筛选时，还需选择或手动输入某个具体的资源名称。
- 操作用户：在下拉框中选择某一具体的操作用户，此操作用户指用户级别，而非租户级别。
- 事件级别：可选项为“所有事件级别”、“Normal”、“Warning”、“Incident”，只可选择其中一项。
- 时间范围：可选择查询最近7天内任意时间段的操作事件。
- 单击“导出”按钮，云审计服务会将查询结果以CSV格式的表格文件导出，该CSV文件包含了本次查询结果的所有事件，且最多导出5000条信息。

步骤6 选择完查询条件后，单击“查询”。

步骤7 在事件列表页面，您还可以导出操作记录文件和刷新列表。

- 单击“导出”按钮，云审计服务会将查询结果以CSV格式的表格文件导出，该CSV文件包含了本次查询结果的所有事件，且最多导出5000条信息。
- 单击按钮，可以获取到事件操作记录的最新信息。

步骤8 在需要查看的事件左侧，单击 展开该记录的详细信息。

步骤9 在需要查看的记录右侧，单击“查看事件”，会弹出一个窗口显示该操作事件结构的详细信息。

步骤10 关于事件结构的关键字段详解，请参见**事件结构**和**事件样例**。

步骤11 （可选）在旧版事件列表页面，单击右上方的“体验新版”按钮，可切换至新版事件列表页面。

----结束

云审计服务支持的 IoTDA 操作列表

用户在使用物联网平台IoTDA的过程中，通过云审计服务，可查看用户及平台的操作及结果，当某项功能发生异常时，用户可以根据日志的记录信息定位并处理故障信息。目前支持查看以下IoTDA的操作记录：

表 8-10 云审计服务支持的 IoTDA 操作列表

功能类别	操作名称	资源类型	事件名称
联动规则管理	创建规则	rules	createRules

功能类别	操作名称	资源类型	事件名称
	删除规则	rules	deleteRules
	更新规则	rules	updateRules
	修改规则状态	rules	changeRuleStatus
JS脚本管理	上传js插件脚本	scripts	createScript
	删除js插件脚本	scripts	deleteScript
	调试js插件脚本	scripts	runScript
Function插件管理	function插件上传	functions	createProductFunctions
	function插件删除	functions	deleteProductFunctions
	function插件下载	functions	getProductFunctions
批量任务管理	创建批量任务	batchtasks	createBatchtasks
	批量任务重试	batchtasks	retryBatchtasks
	批量任务停止	batchtasks	stopBatchtasks
	删除批量任务	batchtasks	deleteBatchtasks
批量任务文件管理	上传批量任务文件	batchtask-files	uploadBatchTaskFile
	删除批量任务文件	batchtask-files	deleteBatchTaskFile
导出任务	创建导出任务	export-tasks	createExportTasks
	删除导出任务	export-tasks	deleteExportTask
	下载导出文件	export-tasks	createTaskreport
应用侧证书管理	上传推送CA证书	Certificate	createCertificate
	更新推送CA证书	Certificate	updateCertificate
	删除推送CA证书	Certificate	deleteCertificate
证书管理	上传设备CA证书	certificate	addCertificate
	删除设备CA证书	certificate	deleteCertificate
	调试设备CA证书	certificate	debugCertificate
	验证设备CA证书	certificate	verifyCertificate
	下载设备CA证书	certificate	downloadCertificate

功能类别	操作名称	资源类型	事件名称
服务端证书管理	企业版创建证书	ServerCertificate	addServerCertificate
	企业版替换证书	ServerCertificate	updateServerCertificate
	企业版删除证书	ServerCertificate	deleteServerCertificate
资源空间管理	创建资源空间	application	addApplication
	删除资源空间	application	deleteApplication
	修改资源空间	application	updateApplication
接入码管理	创建接入码	accessCode	createAccessCode
	校验接入码	accessCode	verifyAccessCode
软固件升级包管理	创建OTA升级包	upgradeTask	uploadOtaPackages
	删除OTA升级包	upgradeTask	deleteOtaPackages
文件存储管理	配置文件上传obs桶	upgradeTask	createBucket
流转规则管理	创建规则触发条件	routing-rule	addRule
	修改规则触发条件	routing-rule	modifyRule
	删除规则触发条件	routing-rule	deleteRule
	测试sql连通性	rule-sql	checkSql
流转规则动作管理	创建规则动作	rule-action	addAction
	修改规则动作	rule-action	modifyAction
	删除规则动作	rule-action	deleteAction
	测试连通性接口	rule-action	sendMessage
外出流控策略管理	创建外出流控策略	create-flow-control-policy	createRoutingFlowControlPolicy
	更新外出流控策略	update-flow-control-policy	updateRoutingFlowControlPolicy
	删除外出流控策略	delete-flow-control-policy	deleteRoutingFlowControlPolicy
外出推送积压策略管理	创建外出推送积压策略	create-routing-backlog-policy	createRoutingBacklogPolicy
	修改外出推送积压策略	update-routing-backlog-policy	updateRoutingBacklogPolicy

功能类别	操作名称	资源类型	事件名称
	删除外出推送积压策略	delete-routing-backlog-policy	deleteRoutingBacklogPolicy
设备影子	配置设备影子预期数据	deviceShadow	updateDeviceShadow
插件映射管理	修改映射关系	plugin	addMapping
插件消息管理	修改消息信息	plugin	addMessage
插件管理	部署在线插件	plugin	deployPlugin
	保存插件信息	plugin	savePluginMessage
	更新插件信息	plugin	modifyPluginMessage
	部署离线插件	plugin	bundlePackages
模拟器管理	注册调试设备模拟器	plugin	registerEmulatedDevice
设备调试消息	上行发送码流	plugin	simulateReport
隧道管理	创建隧道	tunnels	createTunnel
	删除隧道	tunnels	deleteTunnel
	修改隧道	tunnels	updateTunnel
产品管理	创建产品	product	addProduct
	修改产品	product	updateProduct
	删除产品	product	deleteProduct
自定义Topic管理	修改产品自定义topic	topic	updateTopic
	删除产品自定义topic	topic	deleteTopic
	创建产品自定义topic	topic	addTopic
安全异常检测配置	安全异常检测配置	productConfig	addProductConfig
AMQP队列管理	创建AMQP队列	amqp	addQueue
	删除AMQP队列	amqp	deleteQueue
	中止receive-link消费能力	amqp	hangUpConnection
云云对接配置管理	创建云云对接配置	service-integration	addServiceIntegrationConfig

功能类别	操作名称	资源类型	事件名称
	删除云云对接配置	service-integration	deleteServiceIntegrationConfig
	修改云云对接配置	service-integration	modifyServiceIntegrationConfig
群组管理	添加设备组	device-group	addDeviceGroup
	修改设备组	device-group	updateDeviceGroup
	删除设备组	device-group	deleteDeviceGroup
	管理设备组中的设备	device-group	manageDevicesInGroup
设备标签管理	绑定标签	tag	bindTagsToResource
	解绑标签	tag	unbindTagsToResource
设备管理	创建设备	device	addDevice
	修改设备	device	updateDevice
	删除设备	device	deleteDevice
	重置设备密钥	device	resetDeviceSecret
	冻结设备	device	freeze-device
	解冻设备	device	unfreeze-device
鸿蒙软总线	创建鸿蒙软总线	harmony-soft-bus	create-harmony-soft-bus
	删除鸿蒙软总线	harmony-soft-bus	delete-harmony-soft-bus
	重置鸿蒙软总线key	harmony-soft-bus	reset-harmony-soft-bus-key
	同步鸿蒙软总线	harmony-soft-bus	sync-harmony-soft-bus
设备代理管理	删除设备代理	device-proxy	deleteDeviceProxy
	创建设备代理	device-proxy	addDeviceProxy
	修改设备代理	device-proxy	updateDeviceProxy
设备策略管理	创建设备策略	device-policy	addDevicePolicy
	删除设备策略	device-policy	deleteDevicePolicy
	更新设备策略	device-policy	updateDevicePolicy
	绑定设备策略	device-policy	bindDevicePolicy

功能类别	操作名称	资源类型	事件名称
	解绑设备策略	device-policy	unbindDevicePolicy
消息跟踪管理	修改消息跟踪配置	message-trace	updateMessageTraceConfig
	删除消息跟踪配置	message-trace	deleteMessageTraceConfig
	删除消息跟踪数据	message-trace	deleteMessageTraceData
设备运维配置管理	修改设备运维配置	device-config	updateDeviceConfig
设备命令管理	下发设备命令	command	sendCommand
	下发异步设备命令	asyncCommand	sendAsyncCommand
远程登录	建立ssh通道	SshConnect	SshConnect
	下发ssh命令	SshComand	SshComand
	关闭ssh通道	SshDisconnect	SshDisconnect

8.5 查看运行日志（旧版）

华为云物联网平台支持记录平台与设备端，周边应用系统之间的对接情况，并以日志的形式上报到云日志服务（LTS），由LTS提供实时查询、海量存储、结构化处理和可视化图表分析能力，LTS每月免费赠送500M额度，超过后按需收费。LTS服务介绍和计费说明参见**云日志服务LTS**。

目前仅支持记录MQTT设备的业务运行日志，详细请参考下表。

表 8-11 业务类型

业务类型	业务流程
设备状态	设备上线
	设备下线
设备消息	应用侧下发设备消息API
	物联网平台给设备下发消息
	设备给物联网平台上报消息
设备命令	应用侧下发设备命令API
	物联网平台给设备下发命令
设备属性	应用侧修改设备属性API
	设备属性上报

业务类型	业务流程
	网关批量上报设备属性
	平台设置设备属性

运行日志格式如下表，各字段之间以"|"分割。

表 8-12 格式说明

字段	说明
时间	日志采集时间，时间格式为yyyy-MM-dd'T'HH:mm:ss,SSS'Z'，例如：2020-06-16T09:24:45,708Z
设备Id	设备的deviceId
业务类型	对应业务类型： 设备状态：device.status 设备消息：device.message 设备命令：device.command 设备属性：device.property
操作	对应的操作名称，例如：API url，MQTT消息的Topic。
请求参数	操作的请求参数，例如：API的请求体。
结果信息	操作的结果，例如：API的响应体，错误信息等。
执行状态	操作状态码。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。

步骤2 选择左侧导航栏的“监控运维 > 运行日志”，单击“配置日志开关”。

图 8-9 配置日志开关

步骤3 在弹出的页面，勾选需要启动采集的业务类型，然后单击“确认”。

图 8-10 配置日志开关

步骤4 创建运行日志转发规则，将采集的日志数据转发到其他云服务，以便您查看和处理，建议把日志数据转发至云日志服务LTS。以下以转发到云日志服务为例。

1. 选择左侧导航栏的“规则 > 数据转发”，单击“创建规则”。
2. 参考下表参数说明，填写规则内容，填写完成后单击“创建规则”。

表 8-13 创建规则

参数名	参数说明
规则名称	创建的规则名称。
规则描述	对该规则的描述。
数据来源	选择“运行日志”
触发事件	选择数据来源后，自动匹配触发事件。
资源空间	您可以选择单个资源空间或所有资源空间。

3. 单击“设置转发目标”页签，单击“添加”，设置转发目标。

表 8-14 设置转发目标

参数名	参数说明
转发目标	选择“云日志服务（LTS）”
区域	当前仅支持转发至同区域的云日志服务。
日志组/日志流	选择LTS的日志组和日志流，未创建则访问**创建日志组**和**创建日志流**进行创建。

说明

云日志服务创建的日志组默认存储时间为7天，超出存储时间的日志将被自动删除，您可以按需将日志数据转储至OBS桶中长期存储，具体步骤参考**日志转储至OBS**。

云日志服务每月免费赠送500M额度，默认超额继续采集，超过的部分按需收费，如果您需要关闭超额继续采集开关，登录云日志服务管理控制台，单击"配置中心"进行修改。

4. 单击“设置完成”，然后单击“启动规则”，实现运行日志转发至云日志服务。

步骤5 登录**云日志服务**管理控制台，单击“日志管理”。

步骤6 选择**3**创建的日志组和日志流，查看IoTDA上报的日志信息。可参考**搜索日志**，对原始日志进行搜索，搜索目标日志。例如可以按照设备Id和业务类型搜索。

图 8-11 查看日志

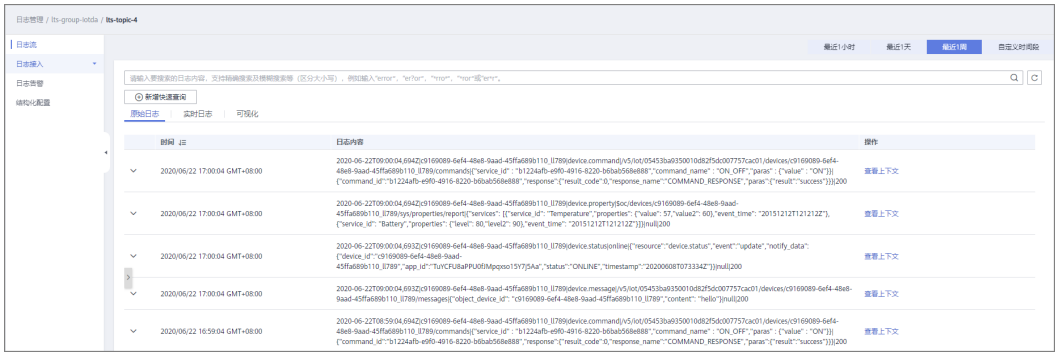
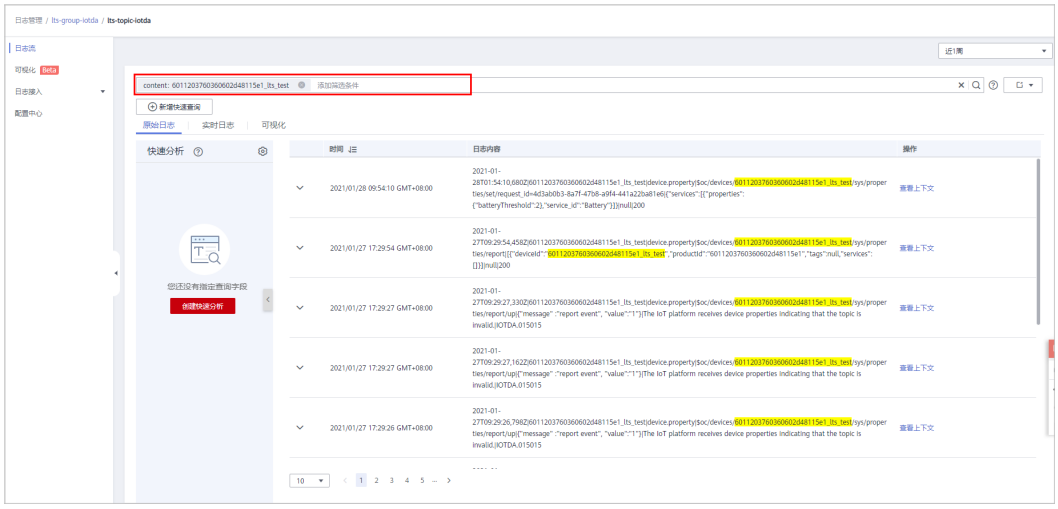


图 8-12 查看日志



步骤7 在日志流详情页面，单击左侧导航栏的“可视化”，进入日志结构化配置页面，单击“分隔符”，推荐使用“分隔符 | ”作为日志提取方法。

图 8-13 日志可视化

步骤8 在选择示例日志页面，选择一条示例日志，日志结构依次为时间、设备ID、业务类型、操作、请求参数、结果信息、执行状态。选择指定分隔符"|",单击“智能提取”，修改智能提取的字段名称，如：accessTime、deviceId、serviceType、operate、request、response、status，修改字段类型为string。单击“保存”，完成日志的结构化配置，具体操作方法可参考**结构化日志**。

图 8-14 结构化日志

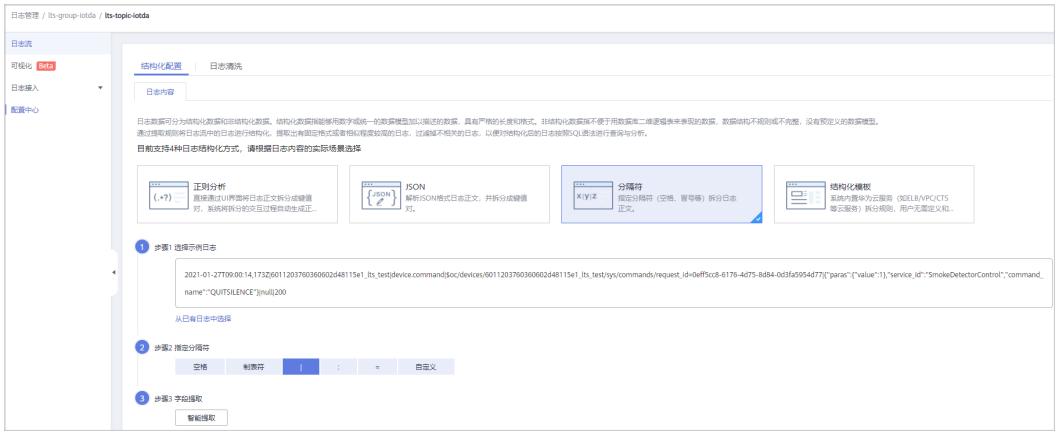
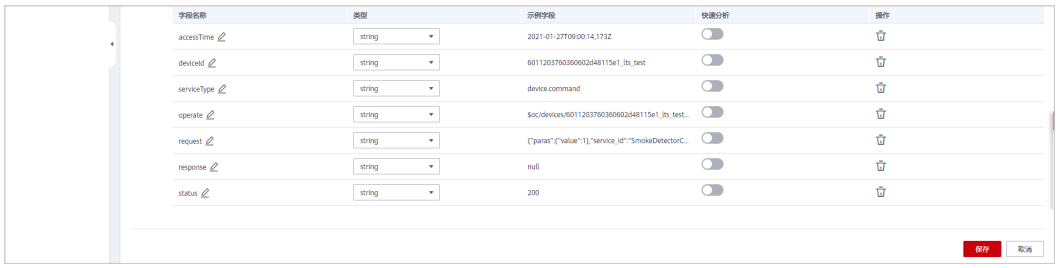
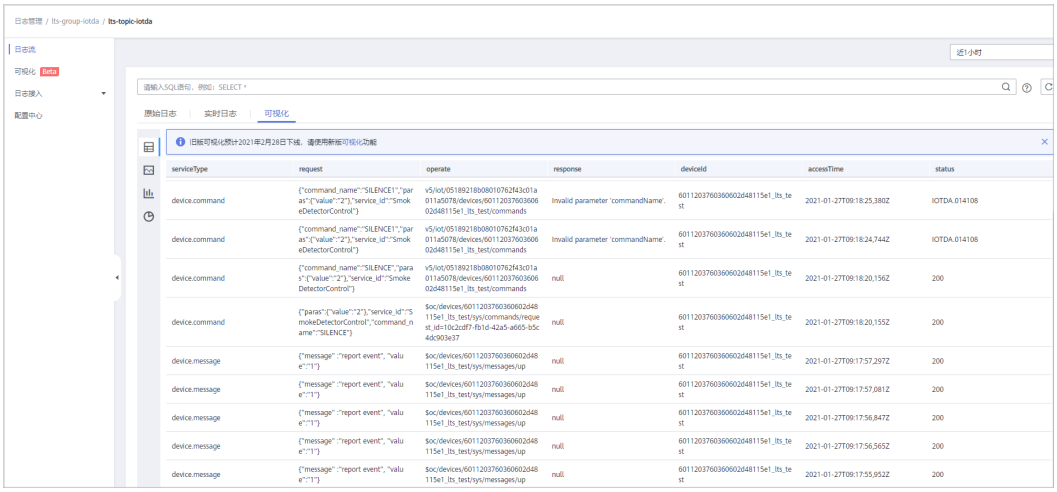


图 8-15 结构化日志



步骤9 对原始日志结构化后，触发响应业务。在“日志管理”页面，选择3创建的日志组和日志流，单击“可视化”，即可查看可视化之后的日志。可对日志进行SQL查询。例如按照设备Id和业务类型查询：select * where deviceId = '5ebac4b54d9b0202c5d8ef0c_test_log' and serviceType = 'device.property'

图 8-16 日志可视化



----结束

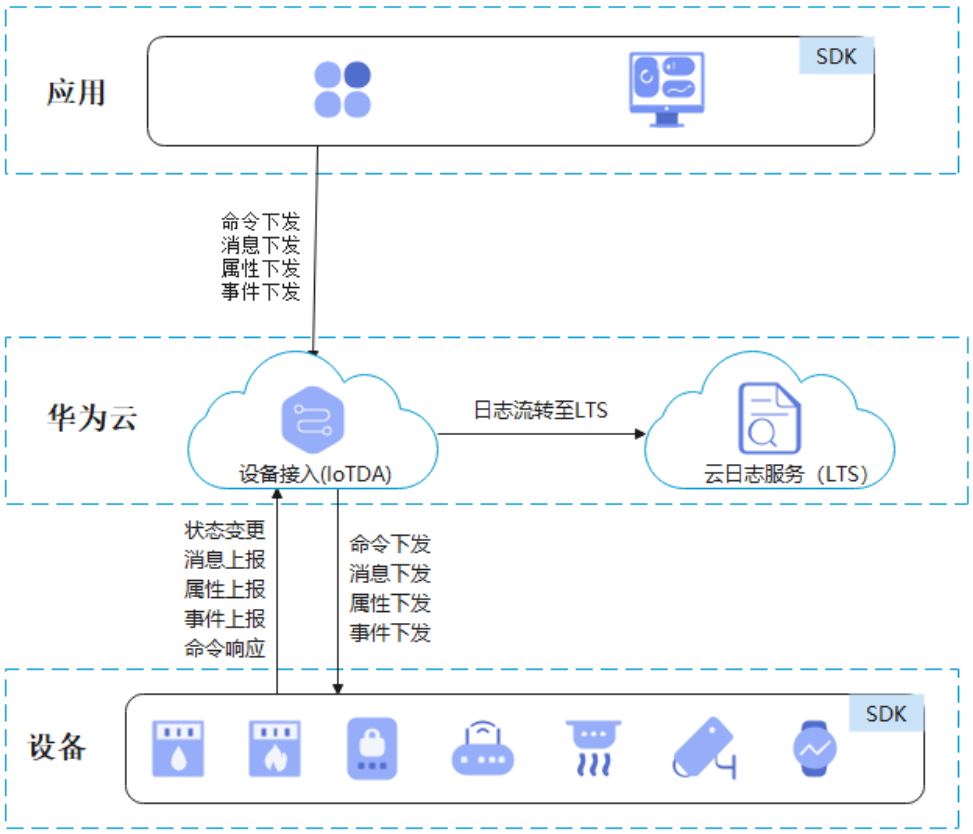
8.6 查看运行日志（新版）

概述

华为云物联网平台支持记录平台与应用侧及平台与设备侧之间的消息交互情况，并以日志的形式上报到云日志服务（LTS），由LTS提供实时查询、海量存储、结构化处理和可视化图表分析能力。

与旧版运行日志相比，使用新版运行日志时，不需要您手动创建日志组、日志流、流转规则及动作，平台将自动为您完成相关配置，并且您可以直接在华为云物联网平台中查看运行日志的内容。

图 8-17 运行日志流程图



使用限制

目前运行日志仅支持记录MQTT设备的业务运行日志，详细请参考**表1 业务类型**。

表 8-15 业务类型

业务类型	业务流程
设备状态	**设备上线**
	设备下线
设备消息	**物联网平台接收应用侧下发消息的请求**
	物联网平台向设备下发消息
	物联网平台接收设备上报的消息
设备命令	**物联网平台接收应用侧下发设备命令的请求**
	物联网平台向设备下发命令
	物联网平台接收设备回复的命令响应
设备属性	**物联网平台接收应用侧修改设备属性的请求**
	物联网平台向设备修改属性

业务类型	业务流程
	物联网平台接收设备上报的属性
	物联网平台接收网关批量上报的设备属性
设备事件	物联网平台通知网关设备新增的事件
	物联网平台通知网关设备删除的事件
	物联网平台接收网关同步子设备列表的事件
	物联网平台接收网关更新子设备状态的事件
	物联网平台接收网关新增子设备请求的事件
	物联网平台下发新增子设备请求响应的事件
	物联网平台接收网关删除子设备请求的事件
	物联网平台下发删除子设备请求响应的事件
	物联网平台接收网关更新子设备请求的事件
	物联网平台下发更新子设备请求响应的事件
	物联网平台下发获取版本信息的事件
	物联网平台接收设备上报软固件版本的事件
	物联网平台下发升级通知的事件
	物联网平台接收设备上报升级状态的事件
	物联网平台接收设备上报获取文件上传URL请求的事件
	物联网平台下发文件上传临时URL的事件
	物联网平台接收设备上报文件上传结果的事件
	物联网平台接收设备上报获取文件下载URL请求的事件
	物联网平台下发文件下载临时URL的事件
	物联网平台接收设备上报文件下载结果的事件
	物联网平台接收设备时间同步请求的事件
	物联网平台下发时间同步响应的事件
	物联网平台接收设备信息上报的事件
	物联网平台下发日志收集通知的事件
	物联网平台接收设备上报日志内容的事件
	物联网平台下发配置的通知的事件
	物联网平台接收设备上报配置响应的事件
	物联网平台接收设备下载升级包的事件

业务类型	业务流程
批量任务	批量任务的子任务执行结果，详细请参考**表2 批量任务运行日志格式**

表 8-16 批量任务运行日志格式

字段	说明
appId	应用ID
deviceId	设备ID
categoryName	日志类型：batch.task
operation	动作，这里为批量任务的task_id，详情参考**创建批量任务**

request	请求内容，json格式，
	{
	"task_type" : "createDevices"，// 任务类型
	"package_id" : "f2303267a6e8f0053037c2a9"，//软固件升级包
	"package_ids" : ["65f3ebe2682b9f4bcc38baad"] //软固件升级包
	}
	说明
	当批量任务任务类型为softwareUpgrade或firmwareUpgrade，支持参数package_id和package_ids

response	响应内容，json格式，
	成功的场景内容为
	{
	"output" : "xxxxxxxxxxxx"
	}
	失败的场景为
	{
	"error" : {
	"error_code" : "IOTDA.XXXXX",
	"error_msg" : "XXXXX."
	}
	}

status	执行结果，这里为子任务的状态，支持Success、Fail、Stopped和Removed

运行日志使用说明

步骤1 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。

步骤2 选择左侧导航栏的“监控运维 > 运行日志”。

步骤3 若您当前在使用旧版运行日志，那么您可以单击右上角“前往新版”按钮，进入新版运行日志页面。若您已使用新版运行日志，则自动进入新版页面。

图 8-18 前往新版

步骤4 首次使用，需要单击“启动并配置”按钮，此时您可以选择两种方式进行运行日志的配置

1. 一键配置：

   采用一键配置后，平台将会自动创建日志组及日志流，并配置结构化，同时创建对应的转发规则及动作，无需用户手动操作。

   图 8-19 一键配置

2. 自定义配置：

   采用自定义配置后，可以选择自己创建的规则及日志组和日志流来创建运行日志，该方式更加灵活。

   若您在日志流中配置了结构化，平台也将会对结构化进行修改。

图 8-20 自定义配置

说明

请谨慎删除数据来源为“运行日志”的转发规则，否则会影响功能的正常使用。

步骤5 启动并配置成功后，您可以在物联网平台中查看或搜索运行日志（支持按时间，日志类型，设备ID，动作以及请求内容进行搜索），对业务进行分析。运行日志保存在云日志服务（LTS）中，默认保存30天，如果当前存储时长不能满足业务要求，您可以在界面上进行修改，最长可保存365天。

图 8-21 日志信息

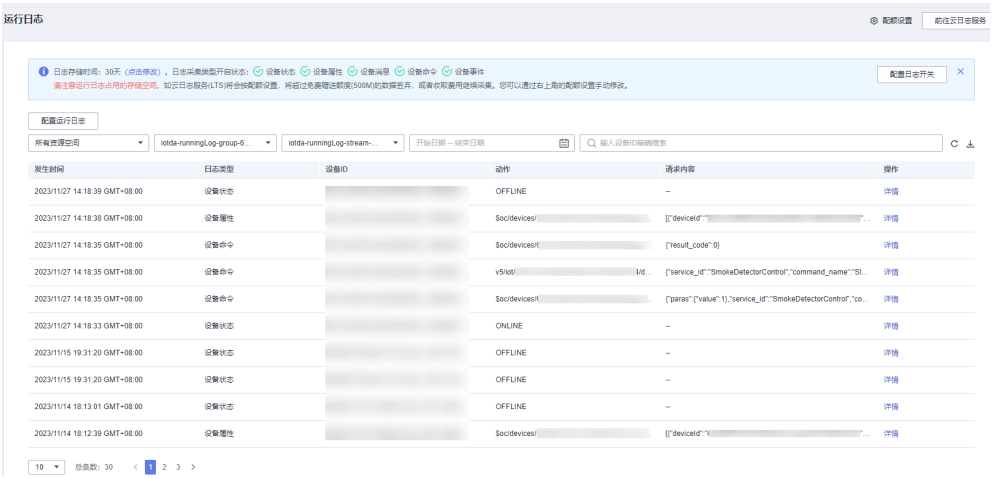
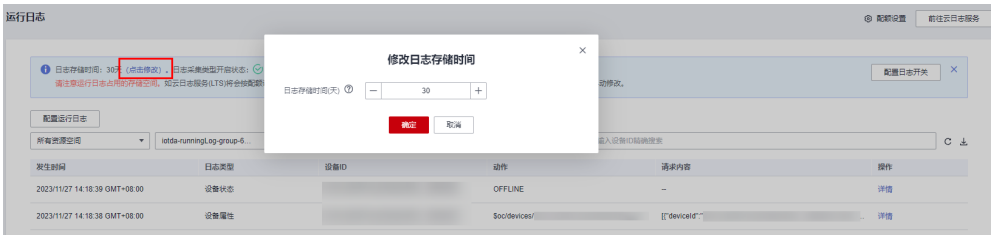


图 8-22 修改日志存储时间



说明

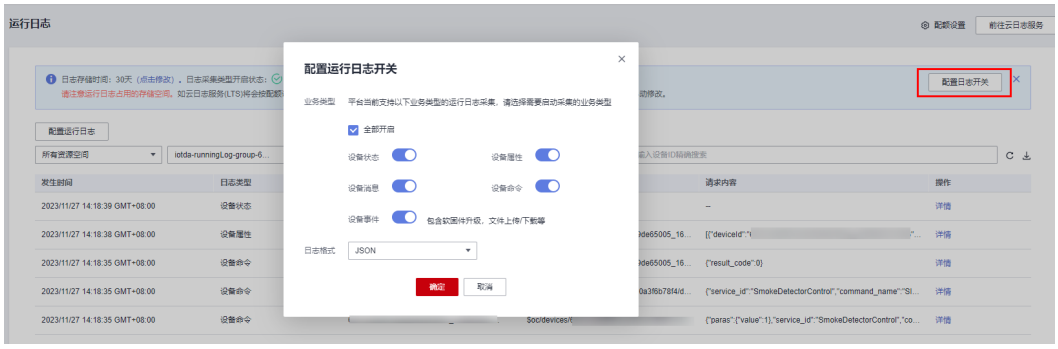
请注意运行日志均存储在云日志服务（LTS）中。LTS每月免费赠送500M额度，您可通过右上角“配额设置”设置超过免费部分的日志的处理策略，同时您也可以在**LTS配置中心**开启日志资源使用量预警。LTS服务介绍参见**云日志服务LTS**。

- 开启开关，则表示日志超过每月免费赠送的额度（500M）时，将继续采集日志，超过部分按需收费，**了解计费详情**；
- 关闭开关，则表示日志超过每月免费赠送的额度（500M）时，将直接丢弃新上报的日志。

图 8-23 配额设置

步骤6 运行日志支持开启一个或多个业务类型，您可单击“配置日志开关”，在弹出框中修改日志收集的类型；所有类型均不选择时，即为关闭运行日志收集功能。

图 8-24 配置日志开关



----结束

运行日志使用示例

本部分介绍如何使用**JAVA SDK**上报消息，触发运行日志流转至LTS，并在物联网平台页面查看消息上报的日志。本示例使用的开发环境为JDK 1.8及以上版本。

前提条件如下：

1. 已在物联网平台注册设备。
2. 已启用并配置新版运行日志，并开启了设备消息日志开关。

配置设备侧SDK步骤如下：

步骤1 配置设备侧SDK的Maven依赖。

```
<dependency>
   <groupId>com.huaweicloud</groupId>
   <artifactId>iot-device-sdk-java</artifactId>
   <version>1.1.4</version>
</dependency>
```

步骤2 配置设备侧SDK，设备连接参数。注意：实际代码中请替换域名（domain），设备ID（deviceId）以及设备密钥（secret）。

```
//加载iot平台的ca证书，获取连接参考：https://support.huaweicloud.com/devg-iothub/
iot_02_1004.html#section3
URL resource = BroadcastMessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());

//注意格式为：ssl://域名信息:端口号。
//域名获取方式：登录华为云IoTDA控制台左侧导航栏“总览”页签，在选择的实例基本信息中，单击“接入信息”。选择8883端口对应的接入域名。
String serverUrl = "ssl://{domain}:8883";
//在IoT平台创建的设备ID。
String deviceId = "{deviceId}";
//设备ID对应的密钥。
String deviceSecret = "{secret}";
//创建设备
IoTDevice device = new IoTDevice(serverUrl, deviceId, deviceSecret, file);
if (device.init() != 0) {
    return;
}
```

步骤3 上报消息

```
device.getClient().reportDeviceMessage(new DeviceMessage("hello"), new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        log.info("reportDeviceMessage ok");
    }

    @Override
    public void onFailure(Object context, Throwable var2) {
        log.error("reportDeviceMessagefail: "+ var2);
    }
});
```

----结束

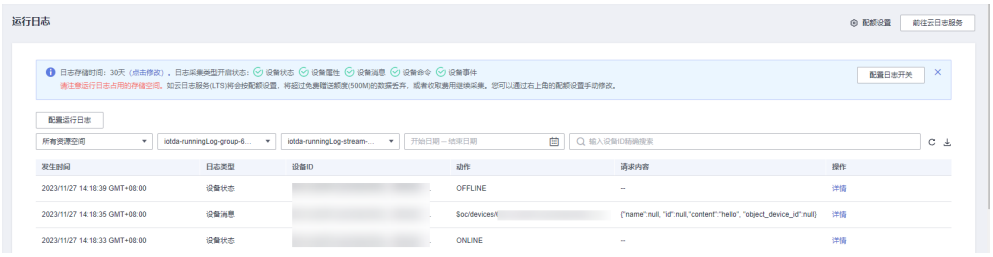
测试验证步骤如下：

步骤1 运行设备测SDK代码，控制台打印如下信息说明设备上线并上报消息成功

```
2023-04-27 17:05:26  INFO MqttConnection:88 - Mqtt client connected. address :ssl://{domain}:8883
2023-04-27 17:05:26  INFO MqttConnection:214 - publish message topic =  $oc/devices/{deviceId}/sys/
messages/up, msg = {"name":null,"id":null,"content":"hello","object_device_id":null}
2023-04-27 17:05:26  INFO MessageSample:43 - reportDeviceMessage ok
```

步骤2 在控制台中查看运行日志，运行日志界面可查询设备上下线以及设备上报消息的记录。

图 8-25 控制台查看运行日志



----结束

8.7 设备异常检测

物联网平台提供设备异常检测功能，当前主要有安全检测和离线分析功能。

安全检测

物联网平台提供安全检测能力，可持续检测设备的安全威胁。本文介绍具体的安全检测项，及如何查看并处理检测出的安全风险。

检测项说明

检测项	说明
设备侧使用非加密方式接入	设备与物联网平台之间，未使用加密协议建立安全连接，可能导致中间人劫持、重放攻击，会对业务造成影响。
使用不安全的TLS版本协议	不安全的TLS协议版本（TLS v1.0、v1.1）存在可被利用的安全漏洞，可能会造成设备数据泄露等安全风险。
使用不安全的加密算法套件	当前主要检测包含以下几种不安全的加密算法套件： TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA, TLS_ECDHE_PSK_WITH_AES_256_CBC_SHA, TLS_PSK_WITH_AES_128_CBC_SHA, TLS_PSK_WITH_AES_256_CBC_SHA 不安全的加密算法套件存在可被利用的安全漏洞，可能会造成设备数据泄露等安全风险。
设备侧单位时间内多次建链	设备侧在1秒内与物联网平台进行多次建链，存在设备被暴力破解，导致身份信息泄露的可能，会造成正常设备被迫下线、业务数据被窃取等安全风险。
设备鉴权失败	设备身份认证信息错误，导致设备无法上线，可能会对业务造成影响。

上述通用异常检测功能检测项开关默认开启，同时设备异常检测包括一些非公共检测项，用户可以根据需求进行检测项的开关等配置。

表 8-17 检测项说明

检测项	说明
内存泄漏检测	检测端侧设备是否存在内存泄漏。
异常端口检测	检测端侧设备是否开启了异常端口。
CPU使用率检测	检测端侧设备CPU使用率是否过高。
磁盘空间检测	检测端侧设备磁盘空间是否不足。
电池电量检测	检测端侧设备电池电量是否过低。
恶意IP检测	检测与设备通信的IP地址是否为恶意IP地址。
本地登录检测	检测设备是否被通过非SSH等网络方式登录。

检测项	说明
暴力破解登录检测	检测设备是否被尝试通过暴力破解账号密码进行登录。
文件篡改检测	检测设备指定目录下的文件是否被篡改。

离线分析

当设备发生离线事件时，需要对离线原因进行分析。根据离线发生的时间、设备的离线原因来统计离线设备的特征，帮助您全面了解、分析设备离线的原因。

离线原因	说明
设备侧主动离线	设备主动向物联网平台发送MQTT协议的DISCONNECT报文，进行离线。
设备侧长时间不发送心跳导致设备离线	设备侧未按照MQTT协议规定，在设置的心跳周期 * 1.5 时间范围内物联网平台发送MQTT协议层的心跳报文，导致物联网平台认为该设备链路已失效，按照协议要求，断开设备链接。 （注：心跳周期是设备侧在与物联网平台进行建链时指定的）
设备侧跟云端之间TCP链路断开，导致设备离线	物联网平台收到设备侧发送的TCP拆链报文，导致设备侧与物联网平台之间的TCP链路断开。
删除设备导致链路断开，设备离线	租户在物联网平台上对该设备进行删除，物联网平台对该设备进行断链。
冻结设备导致链路断开，设备离线	租户在物联网平台上对该设备进行冻结，物联网平台对该设备进行断链。
平台主动断开设备链路，导致设备离线	物联网平台升级期间，会主动断开设备链路。
设备与平台建立多条链路，导致老链路被断链	设备侧与物联网平台主动建立了多条链路，物联网平台将该设备侧的老链路断开，保留新建立的链路。
重置设备密钥，导致设备离线	租户在物联网平台上对该设备进行密钥重置并设置强制断链时，物联网平台对该设备进行断链。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。

步骤2 选择左侧导航栏的“监控运维 > 异常检测”，单击“立即授权”。使用异常检测功能需要授权IoTDA服务对LTS服务进行操作。

步骤3 授权后，可以看到安全检测和离线分析两个页面，两个功能需要手动开启才能使用，单击开启功能。

图 8-26 安全检测

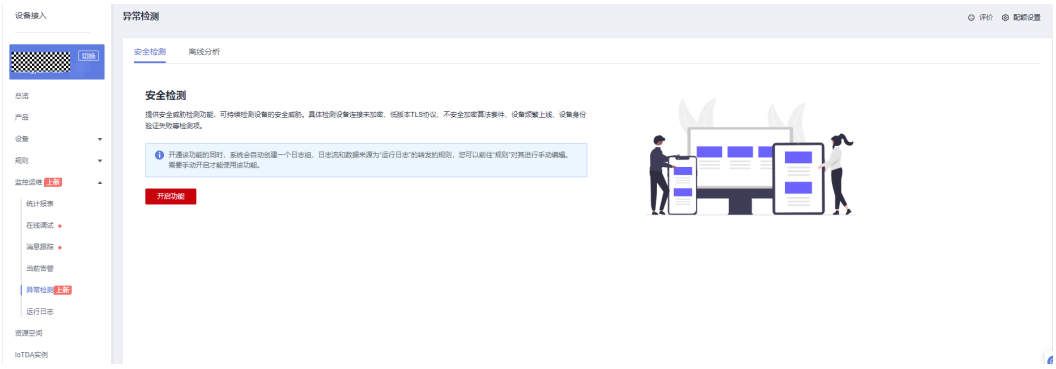
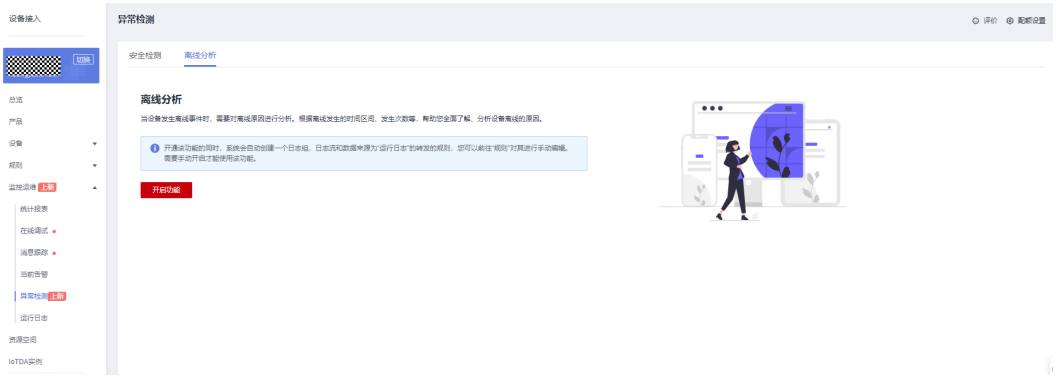


图 8-27 离线分析



说明

1、开通该功能的同时，平台会自动创建一个日志组，日志流和数据来源为所属所有资源空间的“运行日志”转发规则。

其中，日志组名称为：{domainName}-device-exception-group；日志流名称为：{domainName}-device-exception-stream；转发规则名称为：{domainName}-device-exception-rule。

请谨慎删除数据来源为“运行日志”的转发规则，否则会影响功能的正常使用，关闭功能不会删除规则，且再次开启功能后会延用此规则，无需重新创建。

2、请注意异常检测日志占用的存储空间。如云日志服务(LTS)将会按配额设置，将超过免费赠送额度(500M)的数据丢弃，或者收取费用继续采集。您可以通过右上角的“配额设置”手动修改

步骤4 异常检测页面如需开启安全检测配置，请进行如下操作，否则跳过该步骤。

1. 选择异常检测中的“安全检测”，单击“安全检测配置”，弹出对话框单击“添加安全产品配置”。

图 8-28 添加安全配置

2. 在配置页面中，选择产品所在资源空间，以及要配置的产品名称，根据需求开启相应的检测配置项。

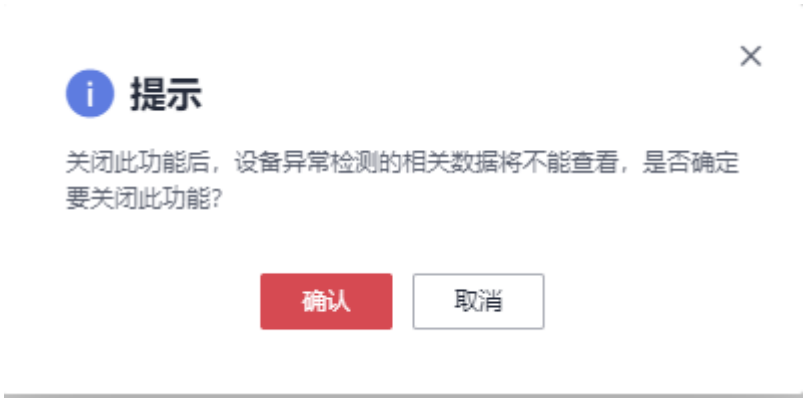
图 8-29 配置检测项

说明

其中内存、CPU使用率、磁盘空间、电池电量检测，通过将设备采集后上报的数值与检测项中配置的对应阈值进行对比，判断是否产生相应告警。异常端口以及恶意IP检测，以设备上报参数值与检测项配置的相应白名单的匹配结果，判断是否产生告警，其中白名单IP支持IP段配置如192.168.1.10/24。异常端口检测以及恶意IP检测项打开时，需要在检测项输入框中配置相应的白名单端口和IP才能实现下发安全配置。

步骤5 物联网平台安全检测与离线分析功能支持开启后手动关闭，页面中单击“关闭功能”，关闭后不再提供该能力，如想再使用，可单击“开启功能”。

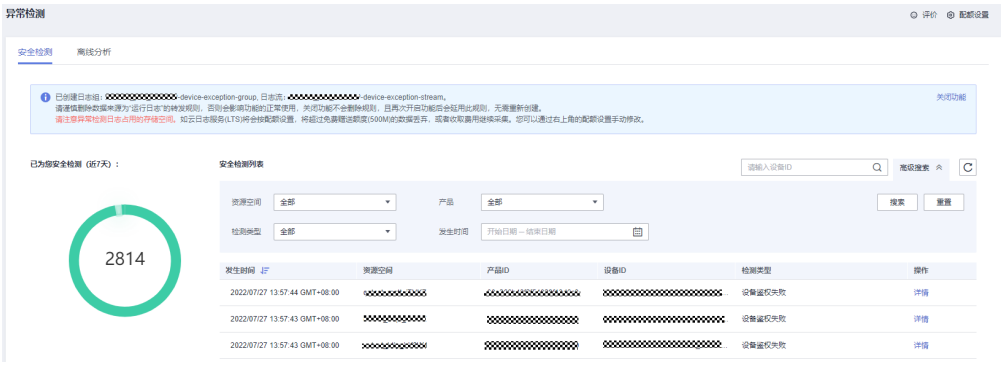
图 8-30 关闭功能提示



说明

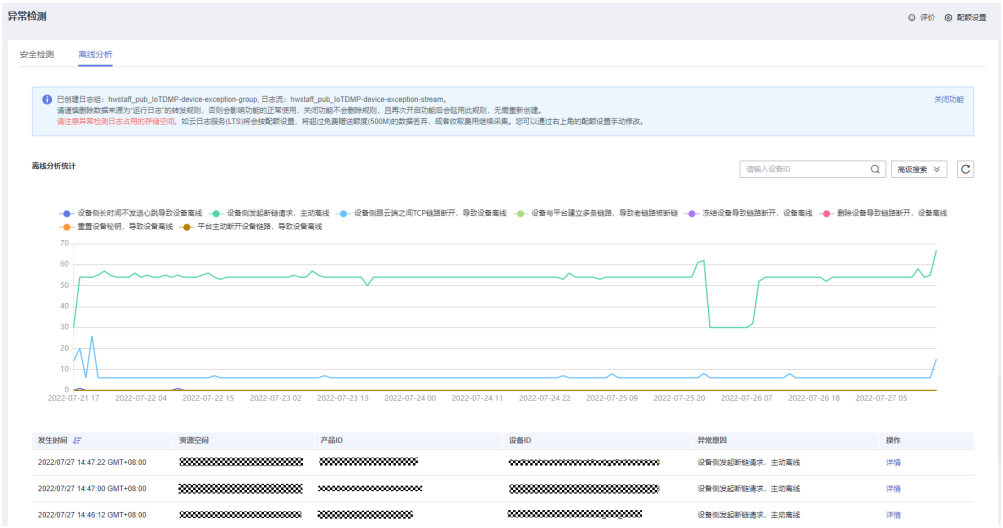
- 开启安全检测功能后，物联网平台即会开始对设备进行安全检测，安全检测最多保持近7天的数据，支持通过设备ID、资源空间、产品、检测类型和发生时间范围进行过滤搜索，单击“详情”可查看检查项内容。

图 8-31 安全检测概览



- 打开离线分析后，物联网平台对设备离线原因自动进行分析，离线分析最多保持近7天的数据，支持通过设备ID、资源空间、产品、异常原因和发生时间范围进行过滤搜索，单击“详情”可查看具体内容。

图 8-32 离线分析概览



----结束

8.8 设备远程登录

物联网平台提供远程登录功能，支持通过控制台远程SSH登录设备，可在控制台输入设备支持的命令，进行功能调试及问题定位，从而方便地实现设备管理及远程运维。下面介绍远程登录的具体使用方法。

前提条件

1. 设备使用的是linux操作系统；

2. 设备上已安装SSH Server 应用；
3. 设备已集成物联网平台官方SDK(IoT Device SDK C v1.1.1及以后版本），请参考（**IoT Device SDK（C）使用指南**）；
4. 设备已在线。

使用限制

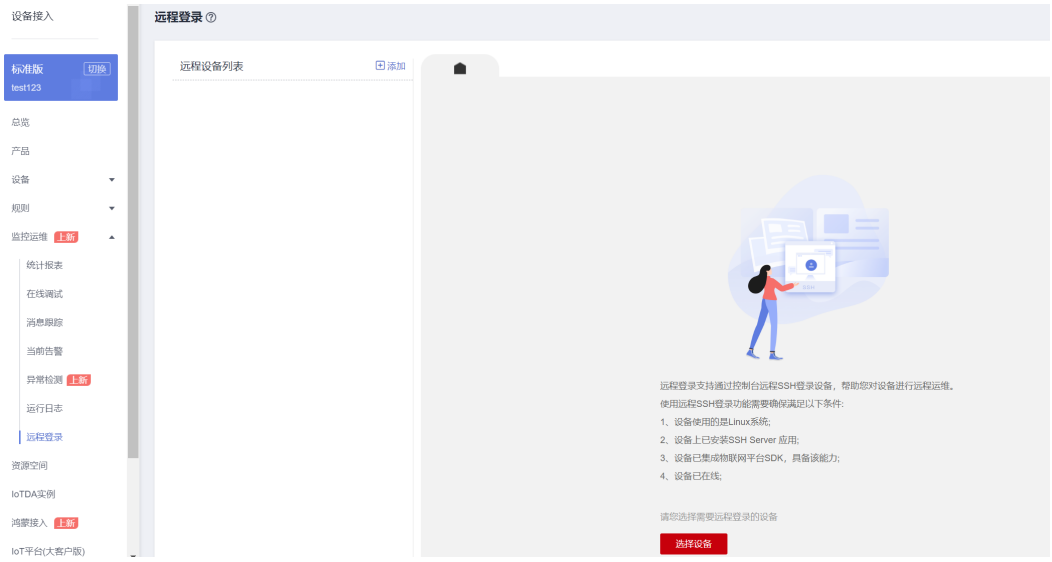
1. 远程连接基于SSH技术实现，物联网平台仅为设备建立SSH服务通道，远程控制台具体支持的管理能力，需您自己在设备端开发。
2. 仅标准版和企业版支持远程登录功能，企业版的应用接入需要提供域名接入方式。
3. 每个设备仅支持同时开启一条远程登录连接，单个租户每个实例最多支持同时开启100个设备远程登录功能。

操作步骤

步骤1 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。

步骤2 选择左侧导航栏的“监控运维 > 远程登录”。

图 8-33 远程登录界面



步骤3 单击选择设备，具体选择需要进行远程登录的设备，输入SSH登录的用户名和密码。

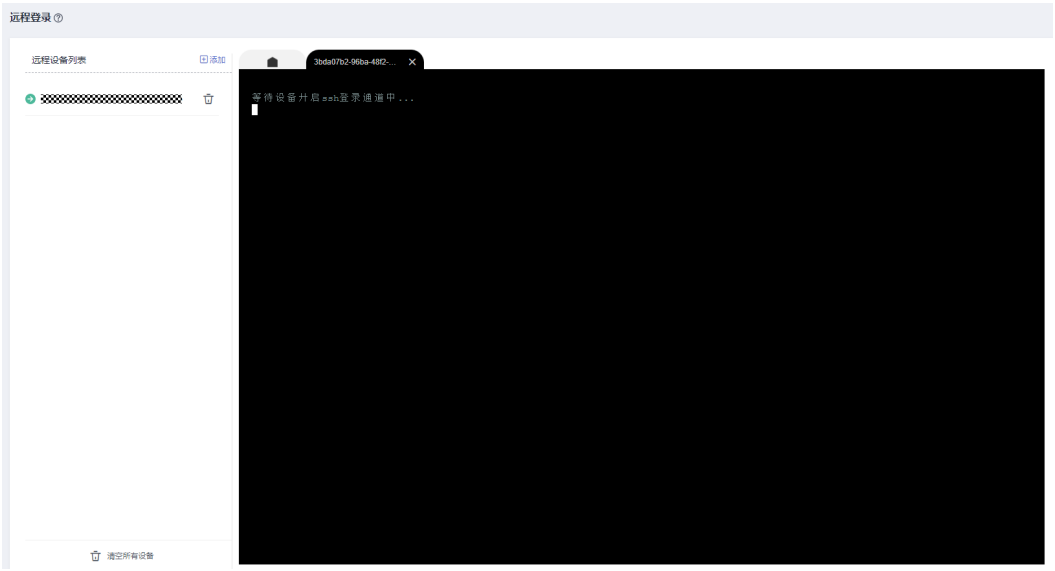
说明

物联网平台不会保存用户输入的用户名和密码，仅透传给设备

图 8-34 选择设备界面

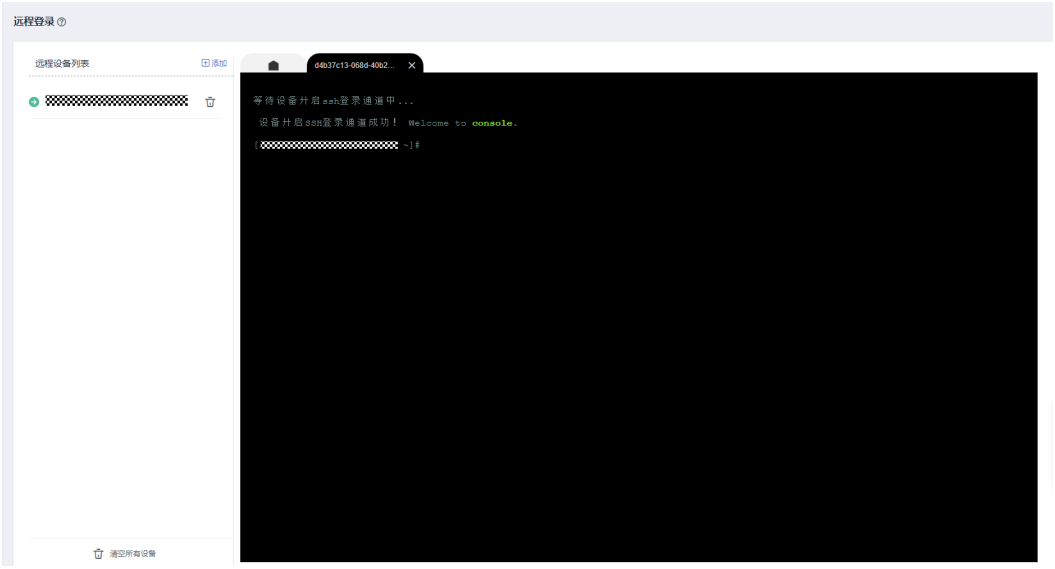
步骤4 单击确定后，您将看到如下远程控制台页面，界面提示等待设备开启SSH功能。

图 8-35 等待设备开启 SSH 功能



步骤5 登录成功后，您将看到如下远程控制台页面，您可根据设备本身功能对设备进行管理。

图 8-36 远程登录成功界面



----结束

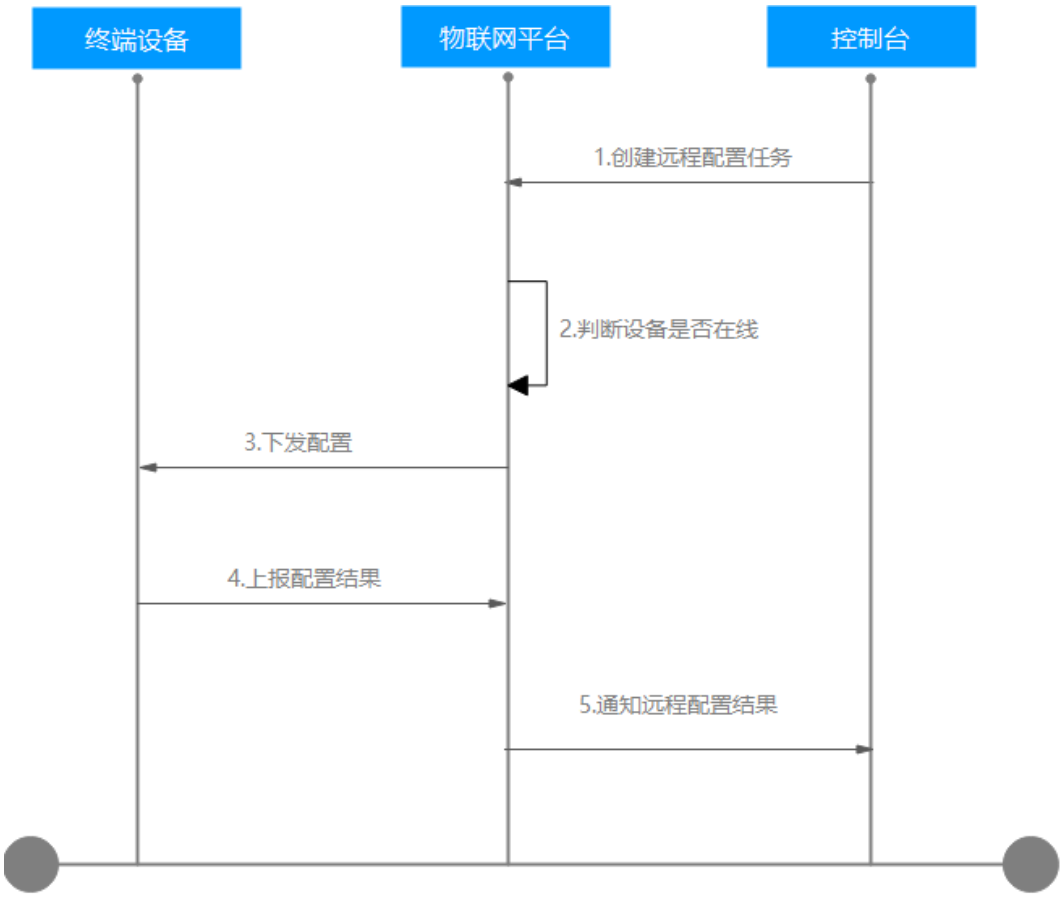
8.9 设备远程配置

概述

物联网平台为用户提供远程配置功能，用户可以在不中断设备运行的情况下，远程更新设备的系统参数、运行参数等配置信息。

例如：Windows的收银台机器，需要通过远程配置修改一些系统参数；车联网中的T-BOX，需要通过远程配置修改设备侧的一些数据上报频率等。

业务流程



设备远程配置流程详细说明：

1. 用户在设备接入服务的控制台上创建远程配置任务。一个应用下最多同时运行10个远程配置任务，每个任务最多支持对10万个设备下发配置。如果一个设备已经在一个已有的远程配置任务中，并且该设备远程配置还未完成，新建的远程配置任务如果包含该设备，则该设备的新远程配置将直接失败。
2. 平台感知设备是否在线，当设备在线时立即下发配置给设备。当设备不在线时，等待设备上线订阅**远程配置Topic**，平台感知设备上线后下发配置。在创建远程配置任务时可以选择配置超时时间（1-30天），默认30天。
3. 设备完成配置更新后，调用**远程配置响应**接口向物联网平台反馈配置更新结果。

操作步骤

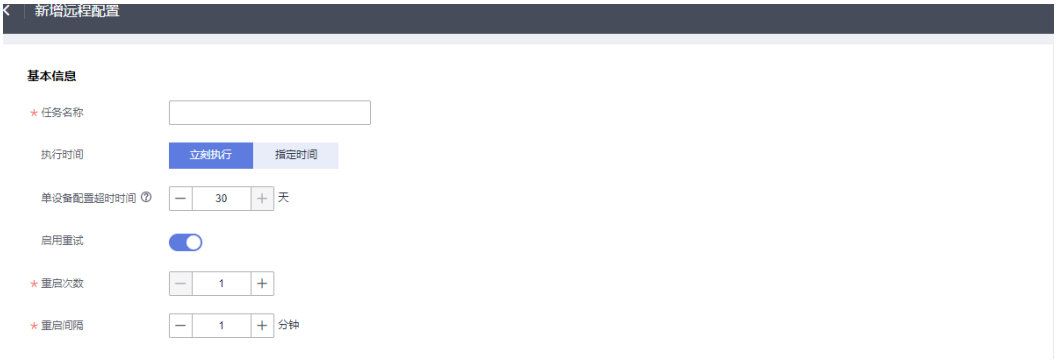
步骤1 访问**设备接入服务**，单击“管理控制台”，进入设备接入控制台。

步骤2 选择左侧导航栏的“监控运维 > 远程配置”。

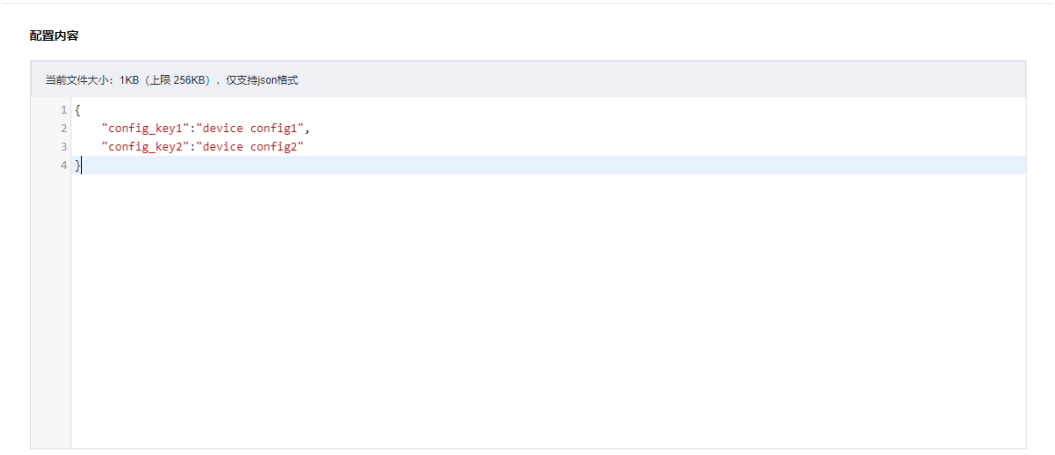
步骤3 单击“新增远程配置”按钮，进入创建远程配置任务界面。

步骤4 在远程配置任务界面，输入任务名称，选择执行时间，配置超时时间和重试策略。

启用重试后，可以设置重启次数和重启间隔。重启次数建议设置为2次，重启间隔设置为5分钟（最大重启次数为5次，最大重启间隔为1440分钟），即设备远程配置失败后，隔5分钟后会进行远程配置重试。

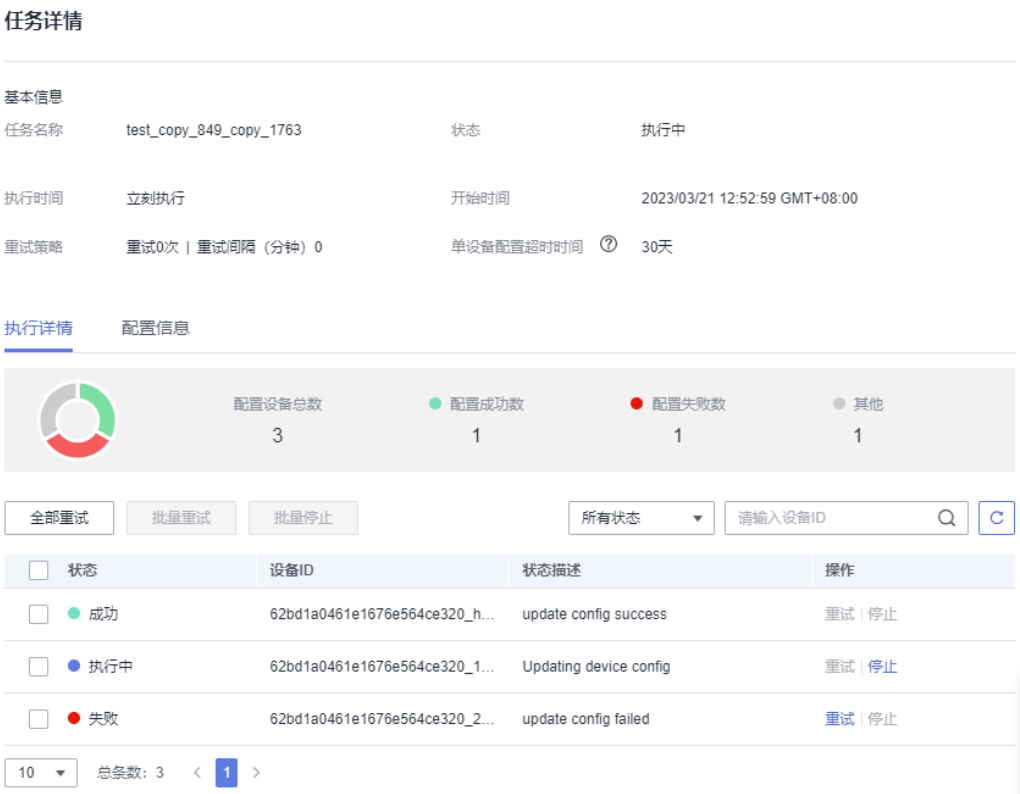


步骤5 输入配置内容，仅支持json格式。



步骤6 选择需要下发配置的设备，支持群组（一个群组最多包含2万设备），文件上传（最大上传10万个设备）以及设备选择（手动筛选需要下发配置的多个设备,最多支持3万个设备。如果需要配置的设备比较多，建议选择群组或者文件上传的方式）三种方式。

步骤7 创建远程配置任务后，设备上线，就能收到平台下发的配置通知。设备更新配置并上报结果后，用户在任务详情页面可以查看设备远程配置的结果。用户可以停止正在执行中的单个设备远程配置或者批量停止多个设备（一次最多支持100个设备）的远程配置任务，也可以单个重试或者批量重试（一次最多支持100个设备）多个设备的配置任务以及全部重试所有失败的远程配置任务。



----结束