ALIBABA CLOUD

阿里云

物联网平台

消息通信

文档版本：20220712

阿里云

法律声明

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

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

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

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

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

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

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.通信方式概述

您的设备和服务器接入阿里云物联网平台，通过物联网平台进行通信。

背景信息

阿里云物联网平台提供的设备端Link SDK，可实现设备快速接入物联网平台，并进行通信。例如使用C语言的Link SDK，完成设备端通信能力开发，请参见MQTT接入示例、自定义Topic通信示例和物模型Topic通信示例。

设备端Link SDK还支持Java、Python、Node.js、Android、iOS语言或平台的开发，更多功能，请参见功能特性。

设备发送数据到物联网平台

设备接入物联网平台后，便可与物联网平台进行通信。设备可通过以下方式发送数据到物联网平台：

- 使用自定义Topic发送自定义格式的数据。
  i. 在物联网平台上，为产品自定义**操作权限**为**发布**的Topic类。产品的Topic类会自动映射到产品下的设备中。

     自定义Topic类的两种方式：

     - 参见自定义Topic，在物联网平台控制台上自定义Topic类。
     - 使用云端SDK调用云端API CreateProductTopic，自定义Topic类。

  ii. 开发设备端时，配置设备将消息发送到自定义Topic中。

     需在设备端上，配置发送消息的自定义Topic和消息格式。使用阿里云提供的Link SDK配置示例，请参见设备发送消息给服务器。

- 使用物模型功能相关Topic，发送标准化的物模型数据。

  物模型功能说明，请参见什么是物模型。

  设备可主动上报属性和事件。

  实现过程：

  i. 参见单个添加物模型，根据您的设备业务情况，在物联网平台控制台定义物模型。

  ii. 开发设备端时，根据已定义的物模型，配置设备上报属性和事件数据。

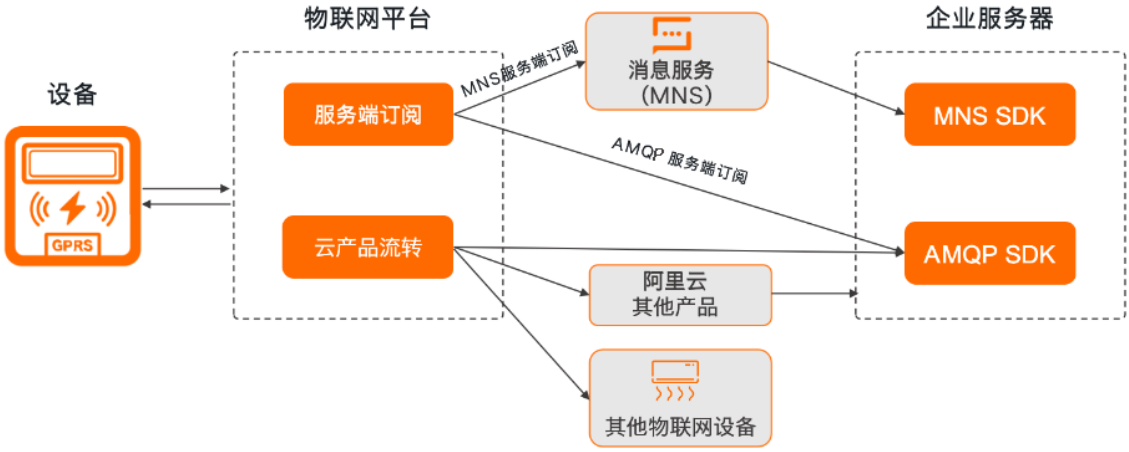
     设备上报属性和事件的标准数据格式，请参见设备上报属性和设备上报事件。

     > **说明** 物联网平台已定义了物模型功能相关的Topic，直接使用即可。

     使用阿里云提供的Link SDK开发示例，请参见设备端SDK上报属性和事件。

物联网平台流转数据到服务器

可设置物联网平台，通过以下方式将设备上报消息、设备状态变化通知、设备生命周期变更、物模型历史数据上报、OTA升级状态通知、网关发现子设备上报、设备拓扑关系变更等消息流转到您的服务器。消息基于Topic传递数据，Topic中的数据格式请参见数据格式。



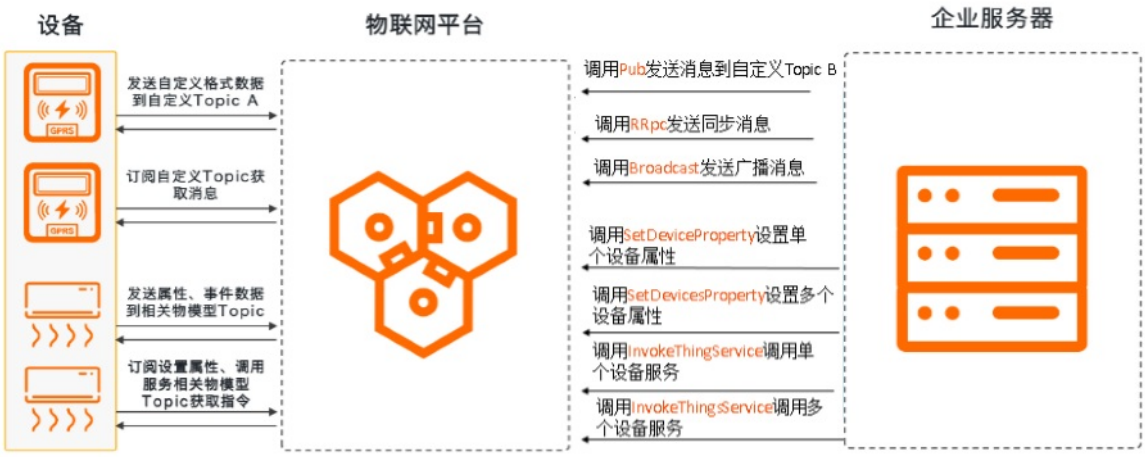
- 服务端订阅：使用物联网平台的服务端订阅功能，订阅一个或多个消息类型。物联网平台根据您设置的订阅，将产品下所有设备的该类型消息流转至您的服务器。支持以下两种方式的服务端订阅：
  - 参见AMQP服务端订阅及相关文档，设置使用AMQP SDK接收物联网平台流转的设备消息。
  - 参见MNS服务端订阅，设置使用MNS SDK接收物联网平台流转到MNS队列的设备消息。
- 云产品流转：使用规则引擎的云产品流转功能，通过数据流转规则将指定设备数据流转到消息服务（MNS）的主题或消息队列（RocketMQ）的队列中。服务器通过MNS或RocketMQ SDK接收消息。
  i. 参见设置数据流转规则，创建规则，将指定设备的数据流转到MNS主题或RocketMQ队列中。
  ii. 参见MNS SDK主题使用手册或RocketMQ使用手册中订阅消息部分，配置MNS SDK或RocketMQ SDK接收消息。

  使用规则引擎流转数据的示例，请参见设备消息通过RocketMQ流转到服务器。

云产品流转和服务端订阅的区别，请参见数据流转方案对比。

服务器远程控制设备

在服务器上，使用物联网平台提供的云端SDK，调用云端API发送指令到设备，实现远程控制设备。服务器可通过以下方式发送指令到设备。



- 使用自定义Topic远程控制设备。

- 异步控制：调用Pub接口向设备操作权限为**订阅**的自定义Topic发送自定义格式的消息。设备通过订阅该Topic获取消息。

  > **说明** 不可以使用Pub接口发送物模型相关指令。

  使用自定义Topic远程控制设备实践案例，请参见服务器发送消息给设备。
- 同步控制：调用RRpc接口向指定设备发送消息，并同步返回响应结果。

  MQTT同步通信相关说明，请参见什么是RRPC。

  调用RRpc接口同步控制设备实践案例，请参见远程控制树莓派服务器。
- 批量控制：调用PubBroadcast接口向产品下的全量在线设备发布广播消息，实现批量控制设备。

  批量控制设备的详细说明，请参见广播通信。

- 使用物模型远程控制设备。

  您可以在云端通过发送物模型数据的特有接口，发布设置属性值和调用服务指令。
  - 控制单个设备。
    - 调用SetDeviceProperty向单个设备发送设置属性值的指令。

      云端下发属性设置命令和设备收到并执行该命令是异步的。设备是否成功设置属性值，以设备上报属性为准。
    - 调用InvokeThingService向单个设备发送调用服务的指令。

      服务是同步调用还是异步调用，取决于您自定义服务时选择的调用方式。

      如果该服务的调用方式是**同步**，调用InvokeThingService后，会同步返回结果。

      如果是**异步**，则InvokeThingService不会同步返回结果。设备响应结果，可以通过规则引擎获取设备的响应消息。需设置规则SQL的数据来源Topic为**物模型数据上报**，具体Topic为`thing/downlink/reply/message`。数据流转规则设置详情，请参见设置数据流转规则。
  - 批量控制设备。
    - 调用SetDevicesProperty向多个设备发送设置属性值的指令。
    - 调用InvokeThingsService向多个设备发送调用服务的指令。

  使用物模型远程控制设备的实践案例，请参见物模型通信。

设备与设备之间通信

将两端设备接入物联网平台，设备间的连接和通信请求都由物联网平台承担。您可以通过以下两种方式实现设备与设备间通信：

- 基于规则引擎的M2M设备间通信
- 基于Topic消息路由的M2M设备间通信

设备数据使用场景示例

通过大数据平台搭建设备监控大屏

推送设备上报数据到钉钉群

使用IoT Studio搭建监控大屏

设备消息通过RocketMQ流转到服务器

服务端订阅（MNS）

2.管理消费组

消费组是消息消费端的身份标识。消息消费端以消费组身份接入物联网平台，并接收物联网平台转发到消费组的消息。本文介绍如何在物联网平台添加、查看和删除消费组。

背景信息

- 功能说明：

  您可通过以下方法监听消费组，获取转发的消息。

  - 配置AMQP服务端订阅：使用AMQP服务端订阅功能，订阅某产品下全部设备的指定类型消息，并将消息流转到指定消费组。
  - 设置数据流转规则：使用云产品流转功能，将指定Topic中的消息流转到AMQP服务端订阅消费组。

  服务端订阅和云产品流转的区别，请参见数据流转方案对比。

- 使用说明：

  将消费组ID配置到AMQP客户端上，AMQP客户端以消费组身份接入物联网平台，接收消息。具体说明，请参见AMQP客户端接入说明。

  多个AMQP客户端（最多64个）可以使用同一个消费组ID，组成消息消费集群。当设备消息到达时，物联网平台将消息随机发往消费组ID中的任一个客户端。

创建消费组

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 服务端订阅**，单击**消费组列表**页签。
4. 单击**创建消费组**。
5. 在**创建消费组**对话框中，输入组名，单击**确认**。

   消费组名称支持中文、英文字母、日文、数字和下划线（_），长度范围为4~30个字符。一个中文或日文占2个字符。

查看和监控消费组

您可以查看消费组内的消息消费速率、消息堆积量，也可以设置云监控报警规则来监控消费组。

1. 在**消费组列表**中，找到要查看的消费组，并单击对应的**查看**。
2. 在**消费组状态**页签，查看订阅消息的**实时消息消费速率**、**堆积消息消费速率**、**消息堆积量**、**最近消费时间**和在线客户端列表。

   当消息堆积量大于等于（≥）1时，**消息堆积量**右侧会显示**清空**按钮，您可清除堆积消息。

> **说明** 消息组创建后：
>
> - 若未上线过，此时消费组处于离线状态，向消费组发送消息时，不会有消息堆积。
> - 若已上线，然后离线，此时消费组处于离线状态，向消费组发送消息时，会有消息堆积。

3. 在**消费组状态**页签，单击**报警配置**，然后在**创建报警规则**页面，配置云监控阈值报警规则，监控消费组消息堆积数、消费组消息消费速率，并接收报警消息。

   您需选择**产品**为**物联网平台-服务端订阅**，其他参数根据您的需求设置。具体操作，请参见创建阈值报警规则。

4. 在**消费组详情**页面，单击**消费日志**页签，可查看具体的消费记录。

删除消费组

您创建的消费组可以删除，物联网平台的默认消费组不可删除。删除消费组后，该组内的所有消费端会停止接收消息。

1. 解除订阅。如果消费组已关联订阅关系，则需先解除订阅；如果消费组无订阅关系，请忽略此步骤。
   i. 在**消费组列表**中，单击消费组对应的**查看**。
   ii. 在**消费组详情**页面的**订阅产品**页签下，单击产品名对应的**解除订阅**，再单击**确认**。

   > **说明** 如果该产品的服务端订阅只有一个消费组，则不能在**消费组详情**页面解除订阅。您可返回**服务端订阅**页面的**订阅列表**页签，编辑或删除订阅。

2. 在**服务端订阅**页面的**消费组列表**页签下，单击消费组对应的**删除**，然后单击**确认**。

相关文档

将消费组ID配置到AMQP客户端上，以接收消息，请参见：

- AMQP客户端接入说明
- Java SDK接入示例
- Node.js SDK接入示例
- .NET SDK接入示例
- Python 2.7 SDK接入示例
- Python3 SDK接入示例
- PHP SDK接入示例
- Go SDK接入示例

相关API

API	描述
CreateConsumerGroup	创建一个消费组，用于创建AMQP服务端订阅。
UpdateConsumerGroup	修改消费组名称。
QueryConsumerGroupByGroupId	根据消费组ID查询消费组详情。

API	描述
QueryConsumerGroupList	查询用户所有消费组列表，或按消费组名称进行模糊查询。
QueryConsumerGroupStatus	使用AMQP服务端订阅时，查询某个消费组的状态，包括在线客户端信息、消息消费速率、消息堆积数、最近消息消费时间。
ResetConsumerGroupPosition	使用AMQP服务端订阅时，清空某个消费组的堆积消息。
DeleteConsumerGroup	删除消费组。

3.服务端订阅

3.1. 什么是服务端订阅

服务端可以直接订阅产品下多种类型的消息，例如设备上报消息、设备状态变化通知、设备生命周期变更、网关发现子设备上报、设备拓扑关系变更等。配置服务端订阅后，物联网平台会将产品下所有设备中已订阅类型的消息，转发至您的服务器。

使用场景

服务端订阅可应用于以下场景：

- 服务端单纯的接收设备数据的场景，并且适用于高并发场景。
- 服务端接收产品下全部设备的订阅数据。

> **说明** 如果您有多个服务器消费同一个产品的订阅消息，消息会随机转发至某个服务器。

服务端订阅与规则引擎数据流转的使用场景和能力对比，请参见数据流转方案对比。

使用AMQP服务端订阅消息

AMQP（Advanced Message Queuing Protocol）即高级消息队列协议。您配置AMQP服务端订阅后，物联网平台会将产品下所有已订阅类型的消息，通过AMQP通道推送至您的服务器。

AMQP服务端订阅消息流转流程如下图所示。

AMQP服务端订阅优势：

- 支持多消费组。同一个账号，可以在开发环境下使消费组A订阅产品A，同时在正式环境下使消费组B订阅产品B。

  > **说明** 如果多个不同消费组同时订阅产品B，则不同消费组可同时收到来自设备B的相同信息。

- 方便排查问题。支持查看客户端状态、查看堆积和消费速率。
- 线性扩展。在消费者能力足够，即客户端机器足够的情况下，可轻松线性扩展推送能力。
- 实时消息优先推送，消息堆积不会影响服务。

  设备实时消息直接推送，推送限流或失败时进入堆积队列，堆积态消息采用降级模式，不会影响实时推送能力。

  即使消费者的客户端宕机，或因消费能力不足堆积了消息，消费端恢复后，设备生成的消息也可以和堆积消息并行发送，使设备优先恢复实时推送消息状态。

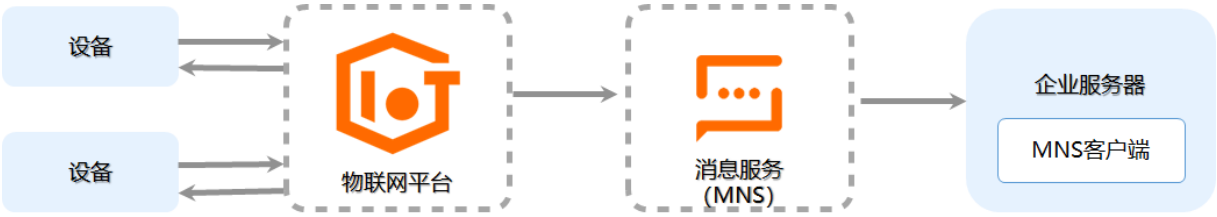
使用AMQP服务端订阅，需先在控制台配置AMQP服务端订阅，请参见配置AMQP服务端订阅。然后开发AMQP客户端，接入物联网平台，接收消息，请参见AMQP客户端接入说明。

> **说明** 使用AMQP服务端订阅，物联网平台会根据推送的消息数量进行计费。消息计费方式，请参见消息通信计费和规则引擎TPS。

使用MNS服务端订阅消息

物联网平台将订阅的消息推送到消息服务（MNS）的队列中，您的服务器MNS客户端通过监听MNS队列接收设备消息。

MNS服务端订阅消息流转流程如下图所示。



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

> **说明** 消息服务会对创建的队列和接收的消息收取费用。消息服务计费和使用，请参见消息服务文档。

3.2. 服务端订阅使用限制

本文介绍服务端订阅的使用限制。

AMQP订阅使用限制

限制项	描述
认证超时	建立连接之后，需要立刻发送认证请求。如果15秒内没有认证成功，服务器将主动关闭连接。
数据超时	服务端与物联网平台建立连接时，需传入心跳时间（AMQP协议参数idle-timeout），取值范围为30~300秒。 如果超过心跳时间，连接上无任何帧通信，物联网平台将关闭连接。 建立连接后，用户服务端需在心跳时间内发送PING包来维持连接。若没有在心跳时间内发送PING包，物联网平台将断开连接。 **说明** 使用阿里云提供的SDK，建立连接后，无需发送PING包维持连接。SDK存在保活心跳，只需保证主进程不退出即可。
失败推送重试策略	由于消费客户端离线、消息消费慢等原因，消息不能实时消费，而进入堆积队列。 • 消费客户端重新上线并恢复稳定消费能力后，物联网平台重试推送堆积消息。 • 如果客户端对重试推送的消息消费失败，可能导致堆积队列阻塞。按大约一分钟间隔，物联网平台向客户端再次重试推送。
消息保存条数	一个消费组最多可堆积1亿条消息。

限制项	描述
消息保存时长	1天。
实时消息推送限流	一个连接限流1,000 TPS。您可通过增加连接数扩容，连接数最大为64个。
堆积消息推送限流	一个消费组限流200 TPS。 **说明** 为避免大量消息堆积，请确保消费客户端在线，并对平台推送的消息及时ACK回复。
一个产品可关联的消费组数量	最多10个。
一个消费组可关联的产品数量	最多1,000个。
一个消费组可订阅的Topic数量	最多100个。
消费组个数限制	一个阿里云账号最多创建1,000个消费组。
消费端个数限制	一个消费组最多支持64个消费端。
连接次数限制	一个消费组1分钟内，消费端请求连接不超过100次。 **说明** 消费端是指接收物联网平台消息的AMQP客户端，而不是设备端。

MNS订阅使用限制

MNS服务端订阅使用限制，请参见MNS使用限制中，队列相关限制说明。

说明

- 创建MNS服务端订阅后，无论是否有消息从物联网平台流转到队列中，MNS都会收取队列资源占用费。
- MNS队列接收的每条消息不能超过64 KB，否则消息会被丢弃。

3.3. 使用AMQP服务端订阅

3.3.1. 配置AMQP服务端订阅

物联网平台支持通过消费组订阅需要的Topic消息到AMQP服务端。本文介绍在物联网平台控制台设置和管理AMQP服务端订阅的操作步骤。

前提条件

已创建待订阅消息的消费组。您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。具体操作，请参见管理消费组。

设置订阅

在物联网平台控制台设置服务端订阅的消息类型。

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 服务端订阅**。
4. 在**服务端订阅**页，单击**创建订阅**。
5. 在**创建订阅**对话框中，完成配置，单击**确认**。

参数	说明
产品	选择订阅消息源设备所属的产品。
订阅类型	选择AMQP。
消费组	选择消息的消费组。 一个产品可选择多个消费组，且一个消费组下可创建多个产品的订阅。 物联网平台已提供一个默认消费组，用于消费消息。如果您需要对消费端进行分组，可单击**选择目标消费组**对话框右下角的**创建消费组**，新建消费组。消费组相关说明，请参见管理消费组。

参数	说明
推送消息类型	服务端要订阅的消息类型。目前，服务端可订阅的设备消息类型包括： **注意** 对于基础型（MQTT）实例下产品和设备，及尊享型实例下MQTT云网关产品和设备，仅支持推送**设备上报消息**、**设备状态变化通知**和**设备生命周期变更**的数据。更多说明，请参见消息通信说明。 ○ **设备上报消息**：产品下所有设备Topic列表中，**操作权限**为**发布**的Topic中的消息。更多信息，请参见什么是Topic。 设备上报消息，包括设备上报的自定义数据和物模型数据（属性上报、事件上报、属性设置响应和服务调用响应）。推送到服务端的物模型数据是经物联网平台系统处理过后的数据，数据格式请参见数据格式。 例如，一个产品有3个Topic类，分别是： ■ `/${YourProductKey}/${YourDeviceName}/user/get`，具有订阅权限。 ■ `/${YourProductKey}/${YourDeviceName}/user/update`，具有发布权限。 ■ `/${YourProductKey}/${YourDeviceName}/thing/event/property/post`，具有发布权限。 那么，服务端订阅会推送具有发布权限的Topic类中的消息，即 `/${YourProductKey}/${YourDeviceName}/user/update` 和 `/${YourProductKey}/${YourDeviceName}/thing/event/property/post` 中的消息。 **注意** 若订阅异步服务调用响应数据，设备端返回的响应消息**Id**必须与物联网平台下发消息的**Id**相同，才可实现数据正常订阅。 ○ **设备状态变化通知**：该产品下的设备上下线状态变化时通知的消息。 ○ **网关子设备发现上报**：网关将发现的子设备信息上报给物联网平台。需要网关上的应用程序支持。网关产品特有消息类型。 ○ **设备拓扑关系变更**：子设备和网关之间的拓扑关系建立和解除消息。网关产品特有消息类型。 ○ **设备生命周期变更**：设备创建、删除、禁用、启用等消息。 ○ **物模型历史数据上报**：设备上报的属性和事件历史数据。 ○ **OTA升级设备状态通知**：包括升级包验证和批量升级时，设备升级成功、失败、取消和进度的事件通知。 ○ **设备标签变更**：设备上报的标签变更消息。 ○ **OTA模块版本号上报**：设备上报的OTA模块版本号变更消息。 ○ **OTA升级批次状态通知**：设备OTA升级批次状态变化通知。 **说明** 如果您需要对监听的设备消息进行过滤或处理，可先通过云产品流转规则将数据转发到**AMQP服务端订阅消费组**，再通过AMQP客户端，监听相关消息。更多信息，请参见云产品流转。 传递消息至AMQP服务端与对应订阅的设备Topic说明，请参见规则引擎流转与设备原始通信的Topic说明。

后续步骤

配置AMQP客户端：建议您使用阿里云物联网平台提供的AMQP SDK接入示例。对于您自研的AMQP SDK，阿里云不提供后续技术支持服务。

> **说明** 所有配置完成，且设备上报订阅数据，并被AMQP客户端接收后，您可以登录物联网平台控制台，在对应实例的**监控运维 > 日志服务 > 云端运行日志**页签，查看设备上报数据、物联网平台转发数据到AMQP客户端和AMQP客户端返回ACK的日志记录。

3.3.2. AMQP客户端接入说明

控制台配置完AMQP服务端订阅后，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，即可接收设备消息。

协议版本说明

AMQP协议标准的详细介绍，请参见AMQP协议标准。

阿里云物联网平台服务端订阅仅支持AMQP 1.0版的协议标准。

连接认证过程

1. 首先，AMQP客户端与物联网平台经过三次握手建立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，不支持非加密的TCP传输。

2. 客户端请求建立Connection。

   连接认证方式为PLAIN-SASL，可以理解为用户名（userName）和密码（password）认证。物联网平台的云端认证userName和password通过后，建立Connection。

   此外，根据AMQP协议，客户端建连时，还需在Open帧中携带心跳时间，即AMQP协议的idle-time-out参数。心跳时间单位为毫秒，取值范围为30,000~300,000。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。SDK不同，idle-time-out参数设置方法不同。具体设置方法，请参见各语言SDK示例文档。

3. 客户端向物联网平台的云端发起请求，建立Receiver Link（即云端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上只能创建一个Receiver Link，不支持创建Sender Link，即只能由物联网平台的云端向客户端推送消息，客户端不能向云端发送消息。
   > - Receiver Link在有的SDK中名称不同，例如在有的SDK上称为MessageConsumer，请根据具体SDK设置。

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：

- 接入域名：公共实例和企业版实例中，AMQP的接入域名，请参见查看实例终端节点。

> **说明**
> - SDK中的*${YourHost}*即为接入域名。
> - 接入前，请确保已在对应实例下创建产品和设备。

- 端口：
  - 对于Java、.NET、Python 2.7、Node.js、Go客户端：端口号为5671。
  - 对于Python3、PHP客户端：端口号为61614。
- 客户端身份认证参数：

```
userName = clientId|iotInstanceId=${iotInstanceId},authMode=aksign,signMethod=hmacsha1,co
nsumerGroupId=${consumerGroupId},authId=${accessKey},timestamp=1573489088171|
password = signMethod(stringToSign, accessSecret)
```

userName参数说明

参数	是否必需	说明
clientId	是	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。
iotInstanceId	否	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 ○ 若有ID值，必须传入该ID值。 ○ 若无ID值，则无需传入。
authMode	是	鉴权模式。目前仅支持aksign模式。
signMethod	是	签名算法。可选：hmacmd5、hmacsha1和hmacsha256。
consumerGroupId	是	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
authId	是	认证信息。在aksign鉴权模式下，authId取值为您的阿里云账号AccessKey ID。 登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey。 **说明** 若使用RAM用户的AccessKey ID，则需要给该RAM用户授予配置AMQP服务端订阅的权限（iot:sub），否则将会连接失败。授权方法，请参见授权RAM用户访问物联网平台。
timestamp	是	当前时间。Long类型的毫秒值时间戳。

password参数说明

参数	是否必需	说明
signMethod	是	签名算法。请使用userName中指定的签名算法计算签名值，并转为base64字符串。
stringToSign	是	待签名的字符串。 将需要签名的参数的键值对按照首字母字典排序，并在键值间添加等号（=）；参数间添加与号（&），拼接成待签名的字符串。 需要签名的参数为：authId和timestamp。 待签名的字符串为 `stringToSign = authId=${accessKey}&timestamp=1573489088171` 。
accessSecret	是	您的阿里云账号的AccessKey Secret。 登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey。 **说明** 若使用RAM用户的AccessKey ID，则需要给该RAM用户授予配置AMQP服务端订阅的权限（iot:sub），否则将会连接失败。授权方法，请参见授权RAM用户访问物联网平台。

接入示例

阿里云提供以下语言的AMQP客户端接入示例。示例中的参数配置，请参见连接配置说明。

注意 建议您使用阿里云物联网平台提供的AMQP SDK接入示例。对于您自研的AMQP SDK，阿里云不提供后续技术支持服务。

- Java SDK接入示例
- .NET SDK接入示例
- Node.js SDK接入示例
- Python 2.7 SDK接入示例
- Python3 SDK接入示例
- PHP SDK接入示例
- Go SDK接入示例

接入过程中，您可能会遇到消息相关的错误码。更多信息，请参见消息相关错误码。

接收物联网平台推送的消息

客户端和物联网平台云端之间的Receiver Link建连成功后，云端就可以在这条Link上向AMQP客户端推送消息。

说明 客户端仅支持接收物联网平台订阅了的消息，要向设备发送消息或指令，可根据需要，调用对应的API。更多信息，请参见API列表。

物联网平台推送的消息：

- 消息体：消息的payload为二进制格式。
- 消息的业务属性，如消息Topic和Message ID等，需要从AMQP协议的Application Properties中获取。格式为 `key:value` 。

Key	含义
topic	消息Topic。
messageId	消息ID。
generateTime	消息生成时间。 **说明** 消息生成时间generateTime不能作为判断消息顺序的依据。

消息回执：

按照AMQP协议的定义，客户端需要给物联网平台的云端回执（AMQP协议上一般称为settle），通知云端消息已经被成功接收。AMQP客户端通常会提供自动回执模式（推荐）和手动回执模式。具体请参考相应的客户端的使用说明。

消息策略：

- 实时消息直接推送。
- 进入堆积列队的消息

  由于消费客户端离线、消息消费慢等原因，消息不能实时消费，而进入堆积队列。
  - 消费客户端重新上线并恢复稳定消费能力后，物联网平台重试推送堆积消息。
  - 如果客户端对重试推送的消息消费失败，可能导致堆积队列阻塞。按大约一分钟间隔，物联网平台向客户端再次重试推送。

说明
- 消费端存在短暂的流量不均衡，属于正常现象。一般能在10分钟内恢复。如果您的消息QPS较高或消息处理较耗费资源，建议增加消费端的数量，保持消费能力冗余。
- 数据流转时，为确保消息送达，同一条消息可能重复发送，直到客户端返回ACK或消息过期。同一条消息的消息ID相同，您可根据消息ID去重。
- 关于消息限制的更多信息，请参见服务端订阅使用限制。
- 您可以在物联网平台控制台清除堆积消息。具体操作，请参见查看和监控消费组。

消息时序：

说明 消息不保序。

- 设备上下线消息：

  收到消息的时间不是实际设备上下线时间。设备上下线顺序可按照Time排序。

  例如，您依次收到3条消息：

  i. 上线：`2018-08-31 10:02:28.195` 。

  ii. 下线：`2018-08-31 10:01:28.195` 。

iii. 下线：`2018-08-31 10:03:28.195`。

这3条消息展示了，设备先下线，再上线，最后下线的过程。

关于消息中参数的更多信息，请参见数据格式。

- 其他类型的消息：

  您需要在业务层，给消息增加序列号。根据接收到消息中的序列号，幂等判断消息是否需要处理。

3.3.3. Java SDK接入示例

本文介绍使用AMQP协议的JMS客户端接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

准备开发环境

示例使用的开发环境如下：

- 操作系统：Windows10
- JDK版本：JDK8
- 集成开发环境：IntelliJ IDEA社区版

下载Apache Qpid JMS客户端

您可访问Qpid JMS 0.57.0，查看Qpid JMS使用说明。

本文示例中，通过在Maven工程中添加如下依赖，下载Qpid JMS客户端。

```
<!-- amqp 1.0 qpid client -->
 <dependency>
   <groupId>org.apache.qpid</groupId>
   <artifactId>qpid-jms-client</artifactId>
   <version>0.57.0</version>
 </dependency>
 <!-- util for base64-->
 <dependency>
   <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.10</version>
</dependency>
```

示例Demo

1. 下载Demo代码包，并解压。
2. 打开IntelliJ IDEA，导入Demo包中的示例工程*amqp-demo*。

   在pom.xml文件中，已添加Maven依赖，下载Qpid JMS客户端。
3. 在*src/main/java/com.aliyun.iotx.demo*目录下*AmqpClient.java*文件中，参照下表，修改AMQP的接入信息。

```
private final static Logger logger = LoggerFactory.getLogger(AmqpClient.class);
private static String accessKey = "${YourAccessKey}";
private static String accessSecret = "${YourAccessSecret}";
private static String consumerGroupId = "${YourConsumerGroupId}";

//iotInstanceId: 实例ID。若是2021年07月30日之前（不含当日）开通的公共实例，请填空字符串。
private static String iotInstanceId = "${YourIotInstanceId}";

//控制台服务端订阅中消费组状态页客户端ID一栏将显示clientId参数。
//建议使用机器UUID、MAC地址、IP等唯一标识等作为clientId。便于您区分识别不同的客户端。
private static String clientId = "${YourClientId}";

//${YourHost}为接入域名，请参见AMQP客户端接入说明文档。
private static String host = "${YourHost}";

// 指定单个进程启动的连接数
// 单个连接消费速率有限，请参考使用限制，最大64个连接
// 连接数和消费速率及rebalance相关，建议每500QPS增加一个连接
private static int connectionCount = 4;
```

参数	示例	说明
accessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
accessSecret	iMS8ZhCDdfJbCMeA005sieKe******	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt*******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 - 若有ID值，必须传入该ID值。 - 若无ID值，传入空值，即 `iotInstanceId = ""` 。
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。

参数	示例	说明
connectionCount	4	启动AMQP客户端的连接数，最大不超过64个。用于实时消息推送的扩容。 **消费组详情**页面会以 `${clientId}+"-"+数字` 形式，显示连接的客户端。其中数字最小值为0。
host	iot-cn-***.amqp.iothub.aliyuncs.com	AMQP接入域名。 `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。

更多参数说明，请参见AMQP客户端接入说明。

4. 运行*AmqpClient.java*程序。

> **说明** 本示例Demo代码中，添加了结束程序的代码（ `Thread.sleep(60 * 1000);` ），即程序启动成功，运行一分钟后会结束。实际场景中，您可根据需要自行设置运行时间。

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。

```
18:07:26.465 [pool-1-thread-3] INFO com.aliyun.iotx.demo.AmqpClient - receive message,
 topic = /[redacted]event/property/post,
 messageId = [redacted],
 content = {"temperature":29,"humidity":29,"time":1617357297270,"deviceName":"[redacted]"}
```

参数	示例	说明
topic	/**********/******/thing/event/property/post	设备属性上报的Topic。
messageId	1324198300680719360	消息的ID。
content	{"temperature":23,"humidity":21,"time":1604548451951}	消息的内容。

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

  您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。

```
16:52:28.567 [FailoverProvider: async work thread] DEBUG org.apache.qpid.jms.provider.failover.FailoverProvider - Connection attempt:[10] to: amqps://[redacted] in-progress
16:52:28.574 [FailoverProvider: async work thread] INFO org.apache.qpid.jms.provider.failover.FailoverProvider - Connection attempt:[10] to: amqps://[redacted] failed
16:52:28.576 [FailoverProvider: async work thread] WARN org.apache.qpid.jms.provider.failover.FailoverProvider - Failed to connect after: 10 attempt(s) continuing to retry.
```

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.4. .NET SDK接入示例

本文介绍使用.NET语言的AMQP SDK接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。

- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

开发环境

本示例使用的开发环境要求如下表。

Framework	支持版本
.Net Framework	3.5、4.0、4.5及以上版本
.NET Micro Framework	4.2及以上版本
.NET nanoFramework	1.0及以上版本
.NET Compact Framework	3.9及以上版本
.Net Core on Windows 10 and Ubuntu 14.04	1.0及以上版本
Mono	4.2.1及以上版本

下载SDK

.NET版本AMQP SDK，推荐使用AMQP.Net Lite库。请访问AMQP.Net Lite下载库和查看使用说明。

添加依赖

在*packages.config*中添加以下依赖。

```
<packages>
  <package id="AMQPNetLite" version="2.2.0" targetFramework="net47" />
</packages>
```

代码示例

```
using System;
using System.Text;
using Amqp;
using Amqp.Sasl;
using Amqp.Framing;
using System.Threading;
using System.Security.Cryptography.X509Certificates;
using System.Net.Security;
using System.Security.Cryptography;
namespace amqp
{
    class MainClass
    {
        //接入域名，请参见AMQP客户端接入说明文档。
        static string Host = "${YourHost}";
        static int Port = 5671;
        static string AccessKey = "${YourAccessKey}";
        static string AccessSecret = "${YourAccessSecret}";
        static string consumerGroupId = "${YourConsumerGroupId}";
        static string clientId = "${YourClientId}";
        //iotInstanceId: 实例ID。
```

```
        //iotInstanceId：实例ID
        static string iotInstanceId = "${YourIotInstanceId}";
        static int Count = 0;
        static int IntervalTime = 10000;
        static Address address;
        public static void Main(string[] args)
        {
            long timestamp = GetCurrentMilliseconds();
            string param = "authId=" + AccessKey + "&timestamp=" + timestamp;
            //userName组装方法，请参见AMQP客户端接入说明文档。
            string userName = clientId + "|authMode=aksign,signMethod=hmacmd5,consumerGroupId=" + consumerGroupId
                + ",iotInstanceId=" + iotInstanceId + ",authId=" + AccessKey + ",timestamp=" + timestamp + "|";
            //计算签名，password组装方法，请参见AMQP客户端接入说明文档。
            string password = doSign(param, AccessSecret, "HmacMD5");
            DoConnectAmqp(userName, password);
            ManualResetEvent resetEvent = new ManualResetEvent(false);
            resetEvent.WaitOne();
        }
        static void DoConnectAmqp(string userName, string password)
        {
            address = new Address(Host, Port, userName, password);
            //创建Connection。
            ConnectionFactory cf = new ConnectionFactory();
            //如果需要，使用本地TLS。
            //cf.SSL.ClientCertificates.Add(GetCert());
            //cf.SSL.RemoteCertificateValidationCallback = ValidateServerCertificate;
            cf.SASL.Profile = SaslProfile.External;
            cf.AMQP.IdleTimeout = 120000;
            //cf.AMQP.ContainerId、cf.AMQP.HostName请自定义。
            cf.AMQP.ContainerId = "client.1.2";
            cf.AMQP.HostName = "contoso.com";
            cf.AMQP.MaxFrameSize = 8 * 1024;
            var connection = cf.CreateAsync(address).Result;
            //Connection Exception已关闭。
            connection.AddClosedCallback(ConnClosed);
            //接收消息。
            DoReceive(connection);
        }
        static void DoReceive(Connection connection)
        {
            //创建Session。
            var session = new Session(connection);
            //创建Receiver Link并接收消息。
            var receiver = new ReceiverLink(session, "queueName", null);
            receiver.Start(20, (link, message) =>
            {
                object messageId = message.ApplicationProperties["messageId"];
                object topic = message.ApplicationProperties["topic"];
                string body = Encoding.UTF8.GetString((Byte[])message.Body);
                //注意：此处不要有耗时的逻辑，如果这里要进行业务处理，请另开线程，否则会堵塞消费。如果消费一直延时，会增加消息重发的概率。
                Console.WriteLine("receive message, topic=" + topic + ", messageId=" + messageId + ", body=" + body);
```

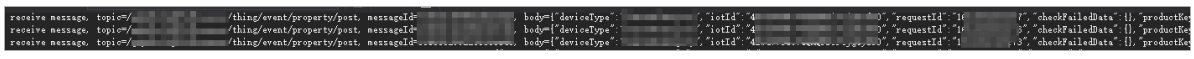
```
                //ACK消息。
                link.Accept(message);
            });
        }
        //连接发生异常后，进入重连模式。
        //这里只是一个简单重试的示例，您可以采用指数退避方式，来完善异常场景，重连策略。
        static void ConnClosed(IAmqpObject _, Error e)
        {
            Console.WriteLine("ocurr error: " + e);
            if(Count < 3)
            {
                Count += 1;
                Thread.Sleep(IntervalTime * Count);
            }
            else
            {
                Thread.Sleep(120000);
            }
            //重连。
            DoConnectAmqp(address.User, address.Password);
        }
        static X509Certificate GetCert()
        {
            string certPath = Environment.CurrentDirectory + "/root.crt";
            X509Certificate crt = new X509Certificate(certPath);
            return crt;
        }
        static bool ValidateServerCertificate(object sender, X509Certificate certificate, X5
09Chain chain, SslPolicyErrors sslPolicyErrors)
        {
            return true;
        }
        static long GetCurrentMilliseconds()
        {
            DateTime dt1970 = new DateTime(1970, 1, 1);
            DateTime current = DateTime.Now;
            return (long)(current - dt1970).TotalMilliseconds;
        }
        //签名方法：支持hmacmd5，hmacsha1和hmacsha256。
        static string doSign(string param, string accessSecret, string signMethod)
        {
            //signMethod = HmacMD5
            byte[] key = Encoding.UTF8.GetBytes(accessSecret);
            byte[] signContent = Encoding.UTF8.GetBytes(param);
            var hmac = new HMACMD5(key);
            byte[] hashBytes = hmac.ComputeHash(signContent);
            return Convert.ToBase64String(hashBytes);
        }
    }
}
```

您需按照如下表格中的参数说明，修改代码中的参数值。更多参数说明，请参见AMQP客户端接入说明。

参数	示例	说明
Host	iot-cn-***.amqp.iothub.aliyuncs.com	AMQP接入域名。 `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。
AccessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
AccessSecret	iMS8ZhCDdfJbCMeA005sieKe********	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 • 若有ID值，必须传入该ID值。 • 若无ID值，传入空值，即 `iotInstanceId = ""` 。
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。

运行结果示例

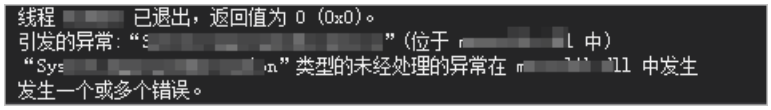
- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。



参数	示例	说明
topic	/**********/******/thing/event/property/post	设备属性上报的Topic。
messageId	1324198300680719360	消息的ID。

参数	示例	说明
body	{"deviceType":"CustomCategory","iotId":"4EwuVV***","requestId":"161268***","checkFailedData":{},"productKey":"g4***S","gmtCreate":1612682173249,"deviceName":"Esensor","items":{"temperature":{"value":-1,"time":1612682173247},"humidity":{"value":74,"time":1612682173247}}}	消息的内容。

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。



相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.5. Python 2.7 SDK接入示例

本文介绍使用Python 2.7 SDK接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

准备开发环境

本示例所使用的开发环境为Python 2.7版。

下载SDK

Python语言的AMQP SDK，推荐使用Apache Qpid Proton 0.29.0，该库中已封装了Python API。请访问Qpid Proton 0.29.0下载库和查看使用说明。

安装Proton。安装操作指导，请参见Installing Qpid Proton。

安装完成后，通过以下Python命令查看SSL库是否安装成功。

```
import proton;print('%s' % 'SSL present' if proton.SSL.present() else 'SSL NOT AVAILABLE')
```

代码示例

```
# encoding=utf-8
import sys
import logging
import time
from proton.handlers import MessagingHandler
from proton.reactor import Container
import hashlib
import hmac
```

```
import base64
reload(sys)
sys.setdefaultencoding('utf-8')
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(levelname)s - %(m
essage)s')
logger = logging.getLogger(__name__)
console_handler = logging.StreamHandler(sys.stdout)
def current_time_millis():
    return str(int(round(time.time() * 1000)))
def do_sign(secret, sign_content):
    m = hmac.new(secret, sign_content, digestmod=hashlib.sha1)
    return base64.b64encode(m.digest())
class AmqpClient(MessagingHandler):
    def __init__(self):
        super(AmqpClient, self).__init__()
    def on_start(self, event):
        # 接入域名，请参见AMQP客户端接入说明文档。
        url = "amqps://${YourHost}:5671"
        accessKey = "${YourAccessKeyID}"
        accessSecret = "${YourAccessKeySecret}"
        consumerGroupId = "${YourConsumerGroupId}"
        clientId = "${YourClientId}"
        # iotInstanceId：实例ID。
        iotInstanceId = "${YourIotInstanceId}"
        # 签名方法：支持hmacmd5，hmacsha1和hmacsha256。
        signMethod = "hmacsha1"
        timestamp = current_time_millis()
        # userName组装方法，请参见AMQP客户端接入说明文档。
        userName = clientId + "|authMode=aksign" + ",signMethod=" + signMethod \
                        + ",timestamp=" + timestamp + ",authId=" + accessKey \
                        + ",iotInstanceId=" + iotInstanceId + ",consumerGroupId=" + consumer
GroupId + "|"
        signContent = "authId=" + accessKey + "&timestamp=" + timestamp
        # 计算签名，password组装方法，请参见AMQP客户端接入说明文档。
        passWord = do_sign(accessSecret.encode("utf-8"), signContent.encode("utf-8"))
        conn = event.container.connect(url, user=userName, password=passWord, heartbeat=60)
        self.receiver = event.container.create_receiver(conn)
    # 当连接成功建立时被调用。
    def on_connection_opened(self, event):
        logger.info("Connection established, remoteUrl: %s", event.connection.hostname)
    # 当连接关闭时被调用。
    def on_connection_closed(self, event):
        logger.info("Connection closed: %s", self)
    # 当远端因错误而关闭连接时被调用。
    def on_connection_error(self, event):
        logger.info("Connection error")
    # 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:
                logger.error("%s: %s: %s" % (
                    event.transport.condition.name, event.transport.condition.description,
                    event.transport.condition.info))
            else:
```

```
                logger.error("%s: %s" % (event.transport.condition.name, event.transport.con
dition.description))
        else:
            logging.error("Unspecified transport error")
    # 当收到消息时被调用。
    def on_message(self, event):
        message = event.message
        content = message.body.decode('utf-8')
        topic = message.properties.get("topic")
        message_id = message.properties.get("messageId")
        print("receive message: message_id=%s, topic=%s, content=%s" % (message_id, topic, c
ontent))
        event.receiver.flow(1)
Container(AmqpClient()).run()
```

您需按照如下表格中的参数说明，修改代码中的参数值。更多参数说明，请参见AMQP客户端接入说明。

参数	示例	说明
url	amqps://iot-cn-***.amqp.iothub.aliyuncs.com:5671	AMQP客户端接入物联网平台的连接地址。格式：`"amqps://${YourHost}:5671"`。 `${YourHost}`对应的AMQP接入域名信息，请参见查看实例终端节点。
accessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
accessSecret	iMS8ZhCDdfJbCMeA005sieKe********	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 • 若有ID值，必须传入该ID值。 • 若无ID值，传入空值，即`iotInstanceId = ""`。

参数	示例	说明
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。

运行结果示例

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。

参数	示例	说明
message_id	1324198300680719360	消息的ID。
topic	/**********/******/thing/event/property/post	设备属性上报的Topic。
content	{"deviceType":"CustomCategory","iotId":"qPi***","requestId":"161***","checkFailedData":{},"productKey":"g4***","gmtCreate":1613635594038,"deviceName":"de***","items":{"Temperature":{"value":24,"time":1613635594036},"Humidity":{"value":26,"time":1613635594036}}}	消息的内容。

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

  您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。

```
2021-02-18 16:19:40,993 - proton - ERROR - Couldn't connect: 10053
2021-02-18 16:19:40,996 - __main__ - ERROR - proton.pythonio: Connection error: 10053
2021-02-18 16:19:40,996 - proton - INFO - Disconnected, reconnecting...
Traceback (most recent call last):
  File "                                    y", line 87, in <module>
    Container(AmqpClient()).run()
  File "                                                                ", line 181, in run
    while self.process(): pass
  File "                                                                ", line 240, in process
    event.dispatch(self._global_handler)
  File "                                                                ", line 135, in dispatch
    _dispatch(handler, type.method, self)
  File "                                                                ', line 117, in _dispatch
    handler.on_unhandled(method, *args)
  File "                                                                , line 665, in on_unhandled
    event.dispatch(self.base)
  File "                                                                ", line 135, in dispatch
    _dispatch(handler, type.method, self)
  File "                                                                , line 115, in _dispatch
    m(*args)
  File "                                                                ", line 892, in on_connection_bound
    addrs = socket.getaddrinfo(host, port, socket.AF_UNSPEC, socket.SOCK_STREAM)
socket.gaierror: [Errno 11001] getaddrinfo failed
```

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.6. Python3 SDK接入示例

本文介绍使用Python3 SDK接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

准备开发环境

可使用Python 3.0及更高版本。本示例使用了Python 3.8版本。

下载SDK

本示例使用stomp.py和schedule，您可访问stomp.py和schedule查看使用说明。

安装stomp.py和schedule的操作指导，请参见Installing Packages。

代码示例

本文提供基于stomp.py的7.0.0版本示例代码。

```
# encoding=utf-8
import time
import sys
import hashlib
import hmac
import base64
import stomp
import ssl
import schedule
import threading
def connect_and_subscribe(conn):
    accessKey = "${YourAccessKeyId}"
    accessSecret = "${YourAccessKeySecret}"
    consumerGroupId = "${YourConsumerGroupId}"
    # iotInstanceId：实例ID。
    iotInstanceId = "${YourIotInstanceId}"
    clientId = "${YourClientId}"
    # 签名方法：支持hmacmd5，hmacsha1和hmacsha256。
    signMethod = "hmacsha1"
    timestamp = current_time_millis()
    # userName组装方法，请参见AMQP客户端接入说明文档。
    # 若使用二进制传输，则userName需要添加encode=base64参数，服务端会将消息体base64编码后再推送。具体添加方法请参见下一章节“二进制消息体说明”。
    username = clientId + "|authMode=aksign" + ",signMethod=" + signMethod \
                   + ",timestamp=" + timestamp + ",authId=" + accessKey \
                   + ",iotInstanceId=" + iotInstanceId \
                   + ",consumerGroupId=" + consumerGroupId + "|"
```

```
    signContent = "authId=" + accessKey + "&timestamp=" + timestamp
    # 计算签名，password组装方法，请参见AMQP客户端接入说明文档。
    password = do_sign(accessSecret.encode("utf-8"), signContent.encode("utf-8"))
    conn.set_listener('', MyListener(conn))
    conn.connect(username, password, wait=True)
    # 清除历史连接检查任务，新建连接检查任务
    schedule.clear('conn-check')
    schedule.every(1).seconds.do(do_check,conn).tag('conn-check')
class MyListener(stomp.ConnectionListener):
    def __init__(self, conn):
        self.conn = conn
    def on_error(self, frame):
        print('received an error "%s"' % frame.body)
    def on_message(self, frame):
        print('received a message "%s"' % frame.body)
    def on_heartbeat_timeout(self):
        print('on_heartbeat_timeout')
    def on_connected(self, headers):
        print("successfully connected")
        conn.subscribe(destination='/topic/#', id=1, ack='auto')
        print("successfully subscribe")
    def on_disconnected(self):
        print('disconnected')
        connect_and_subscribe(self.conn)
def current_time_millis():
    return str(int(round(time.time() * 1000)))
def do_sign(secret, sign_content):
    m = hmac.new(secret, sign_content, digestmod=hashlib.sha1)
    return base64.b64encode(m.digest()).decode("utf-8")
# 检查连接，如果未连接则重新建连
def do_check(conn):
    print('check connection, is_connected: %s', conn.is_connected())
    if (not conn.is_connected()):
        try:
            connect_and_subscribe(conn)
        except Exception as e:
            print('disconnected, ', e)
# 定时任务方法，检查连接状态
def connection_check_timer():
    while 1:
        schedule.run_pending()
        time.sleep(10)
#  接入域名，请参见AMQP客户端接入说明文档。这里直接填入域名，不需要带amqps://前缀
conn = stomp.Connection([('${YourHost}', 61614)], heartbeats=(0,300))
conn.set_ssl(for_hosts=[('${YourHost}', 61614)], ssl_version=ssl.PROTOCOL_TLS)
try:
    connect_and_subscribe(conn)
except Exception as e:
    print('connecting failed')
    raise e
# 异步线程运行定时任务，检查连接状态
thread = threading.Thread(target=connection_check_timer)
thread.start()
```

您需按照如下表格中的参数说明，修改代码中的参数值。更多参数说明，请参见AMQP客户端接入说明。

参数	示例	说明
accessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
accessSecret	iMS8ZhCDdfJbCMeA005sieKe********	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 • 若有ID值，必须传入该ID值。 • 若无ID值，传入空值，即 `iotInstanceId = ""` 。
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。
conn	stomp.Connection([('iot-cn-***.amqp.iothub.aliyuncs.com', 61614)])	创建AMQP客户端与物联网平台的TLS连接。 `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。
conn.set_ssl	for_hosts=[('iot-cn-***.amqp.iothub.aliyuncs.com', 61614)], ssl_version=ssl.PROTOCOL_TLS	

运行结果示例

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。

```
successfully connected
successfully subscribe
check connection, is_connected: %s True
check connection, is_connected: %s True
```

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。

二进制消息体说明

当您需要传输二进制数据时，由于STOMP协议为文本协议，需要使用base64编码参数，否则消息体可能会被截断。

本示例中，userName需要按以下方法添加encode=base64参数，使服务端将消息体base64编码后再推送。

```
username = clientId + "|authMode=aksign" + ",signMethod=" + signMethod \
                    + ",timestamp=" + timestamp + ",authId=" + accessKey \
                    + ",iotInstanceId=" + iotInstanceId \
                    + ",consumerGroupId=" + consumerGroupId \
                    + ",encode=base64"+"|"
```

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.7. Node.js SDK接入示例

本文介绍使用Node.js语言的AMQP SDK接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

准备开发环境

支持使用的开发环境为Node.js 8.0.0及以上版本。

下载安装SDK

Node.js版本AMQP SDK，推荐使用rhea。请访问rhea下载库和查看使用说明。

本文示例使用命令 `npm install rhea` ，下载rhea库。

代码示例

1. 在Windows系统或Linux系统下载并安装Node.js。本文以Windows 10（64位）系统为例，下载安装包node-v14.15.1-x64.msi。
2. 安装成功后，打开CMD窗口，通过以下命令查看node版本。

   ```
   node --version
   ```

显示如下版本号，表示安装成功。

```
v14.15.1
```

3. 在本地计算机创建一个JavaScript文件（例如*amqp.js*），用来存放Node.js示例代码。

   示例代码如下，按照以下表格中的参数说明，修改代码中的参数值。

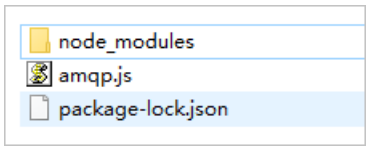
```
const container = require('rhea');
const crypto = require('crypto');
//创建Connection。
var connection = container.connect({
    //接入域名，请参见AMQP客户端接入说明文档。
    'host': '${YourHost}',
    'port': 5671,
    'transport':'tls',
    'reconnect':true,
    'idle_time_out':60000,
    //userName组装方法，请参见AMQP客户端接入说明文档。
    'username':'${YourClientId}|authMode=aksign,signMethod=hmacsha1,timestamp=1573489088
171,authId=${YourAccessKeyId},iotInstanceId=${YourIotInstanceId},consumerGroupId=${YourC
onsumerGroupId}|',
    //计算签名，password组装方法，请参见AMQP客户端接入说明文档。
    'password': hmacSha1('${YourAccessKeySecret}', 'authId=${YourAccessKeyId}&timestamp=
1573489088171'),
});
//创建Receiver Link。
var receiver = connection.open_receiver();
//接收云端推送消息的回调函数。
container.on('message', function (context) {
    var msg = context.message;
    var messageId = msg.message_id;
    var topic = msg.application_properties.topic;
    var content = Buffer.from(msg.body.content).toString();
    // 输出内容。
    console.log(content);
    //发送ACK，注意不要在回调函数有耗时逻辑。
    context.delivery.accept();
});
//计算password签名。
function hmacSha1(key, context) {
    return Buffer.from(crypto.createHmac('sha1', key).update(context).digest())
        .toString('base64');
}
```

参数	示例	说明
host	iot-cn-***.amqp.iothub.aliyuncs.com	AMQP接入域名。 `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。

参数	示例	说明
username	'test|authMode=aksign,signMethod=hmacsha1,timestamp=1573489088171,authId=LTAI****vKu****,iotInstanceId=iot-060a02ne,consumerGroupId=DEFAULT_GROUP|'	接入物联网平台的身份认证信息。其中： ○ `${YourClientId}`：替换为客户端ID，可自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。 ○ `${YourAccessKeyId}`、`${YourAccessKeySecret}`：替换为您物联网平台的AccessKey ID和AccessKey Secret。 登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。 ○ `${YourIotInstanceId}`：替换为实例ID。 您可登录物联网平台控制台，在**实例概览**找到对应的实例，查看实例ID。 **注意** 若没有ID，直接删除${YourIotInstanceId}即可。 ○ `${YourConsumerGroupId}`：替换为消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
password	hmacSha1('iMS8ZhCDd****', 'authId=LTAI4GFGQ****&timestamp=1573489088171')	

4. 打开CMD窗口，使用cd命令找到*amqp.js*文件所在路径，在该路径下使用npm命令下载rhea库。下载后，项目文件如下图所示。

```
npm install rhea
```



5. 在CMD窗口输入如下命令，运行*amqp.js*代码，启动AMQP客户端服务器。

```
node amqp.js
```

运行结果示例

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

  您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.8. Go SDK接入示例

本文介绍使用AMQP协议的Go客户端接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

准备开发环境

本示例的测试环境为Go 1.12.7。

下载SDK

可使用以下命令导入Go语言AMQP SDK。

```
import "pack.ag/amqp"
```

SDK使用说明，请参见package amqp。

代码示例

```
package main
import (
    "context"
    "crypto/hmac"
    "crypto/sha1"
    "encoding/base64"
    "fmt"
    "pack.ag/amqp"
    "time"
)
```

```
//参数说明，请参见AMQP客户端接入说明文档。
const accessKey = "${YourAccessKey}"
const accessSecret = "${YourAccessSecret}"
const consumerGroupId = "${YourConsumerGroupId}"
const clientId = "${YourClientId}"
//iotInstanceId：实例ID。
const iotInstanceId = "${YourIotInstanceId}"
//接入域名，请参见AMQP客户端接入说明文档。
const host = "${YourHost}"
func main() {
    address := "amqps://" + host + ":5671"
    timestamp := time.Now().Nanosecond() / 1000000
    //userName组装方法，请参见AMQP客户端接入说明文档。
    userName := fmt.Sprintf("%s|authMode=aksign,signMethod=Hmacsha1,consumerGroupId=%s,authI
d=%s,iotInstanceId=%s,timestamp=%d|",
        clientId, consumerGroupId, accessKey, iotInstanceId, timestamp)
    stringToSign := fmt.Sprintf("authId=%s&timestamp=%d", accessKey, timestamp)
    hmacKey := hmac.New(sha1.New, []byte(accessSecret))
    hmacKey.Write([]byte(stringToSign))
    //计算签名，password组装方法，请参见AMQP客户端接入说明文档。
    password := base64.StdEncoding.EncodeToString(hmacKey.Sum(nil))
    amqpManager := &AmqpManager{
        address:address,
        userName:userName,
        password:password,
    }
    //如果需要做接受消息通信或者取消操作，从Background衍生context。
    ctx := context.Background()
    amqpManager.startReceiveMessage(ctx)
}
//业务函数。用户自定义实现，该函数被异步执行，请考虑系统资源消耗情况。
func (am *AmqpManager) processMessage(message *amqp.Message) {
    fmt.Println("data received:", string(message.GetData()), " properties:", message.Applica
tionProperties)
}
type AmqpManager struct {
    address     string
    userName     string
    password     string
    client       *amqp.Client
    session     *amqp.Session
    receiver     *amqp.Receiver
}
func (am *AmqpManager) startReceiveMessage(ctx context.Context)  {
    childCtx, _ := context.WithCancel(ctx)
    err := am.generateReceiverWithRetry(childCtx)
    if nil != err {
        return
    }
    defer func() {
        am.receiver.Close(childCtx)
        am.session.Close(childCtx)
        am.client.Close()
    }()
    for {
```

```
        //阻塞接受消息，如果ctx是background则不会被打断。
        message, err := am.receiver.Receive(ctx)
        if nil == err {
            go am.processMessage(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 := am.generateReceiverWithRetry(childCtx)
            if nil != err {
                return
            }
        }
    }
}
func (am *AmqpManager) generateReceiverWithRetry(ctx context.Context) error {
    //退避重连，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 := am.generateReceiver()
        if nil != err {
            time.Sleep(duration)
            if duration < maxDuration {
                duration *= 2
            }
            fmt.Println("amqp connect retry,times:", times, ",duration:", duration)
            times ++
        } else {
            fmt.Println("amqp connect init success")
            return nil
        }
    }
}
//由于包不可见，无法判断Connection和Session状态，重启连接获取。
func (am *AmqpManager) generateReceiver() error {
    if am.session != nil {
        receiver, err := am.session.NewReceiver(
            amqp.LinkSourceAddress("/queue-name"),
            amqp.LinkCredit(20),
        )
        //如果断网等行为发生，Connection会关闭导致Session建立失败，未关闭连接则建立成功。
        if err == nil {
```

```
            am.receiver = receiver
            return nil
        }
    }
    //清理上一个连接。
    if am.client != nil {
        am.client.Close()
    }
    client, err := amqp.Dial(am.address, amqp.ConnSASLPlain(am.userName, am.password), )
    if err != nil {
        return err
    }
    am.client = client
    session, err := client.NewSession()
    if err != nil {
        return err
    }
    am.session = session
    receiver, err := am.session.NewReceiver(
        amqp.LinkSourceAddress("/queue-name"),
        amqp.LinkCredit(20),
    )
    if err != nil {
        return err
    }
    am.receiver = receiver
    return nil
}
```

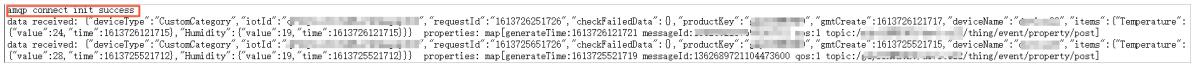
您需按照如下表格中的参数说明，修改代码中的参数值。更多参数说明，请参见AMQP客户端接入说明。

参数	示例	说明
accessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
accessSecret	iMS8ZhCDdfJbCMeA005sieKe**********	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。

参数	示例	说明
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 • 若有ID值，必须传入该ID值。 • 若无ID值，传入空值，即 `iotInstanceId = ""` 。
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。
host	iot-cn-***.amqp.iothub.aliyuncs.com	AMQP接入域名。 `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。

运行结果示例

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。



- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

  您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。

```
amqp connect retry, times: 1 , duration: 20ms
amqp connect retry, times: 2 , duration: 40ms
amqp connect retry, times: 3 , duration: 80ms
amqp connect retry, times: 4 , duration: 160ms
amqp connect retry, times: 5 , duration: 320ms
amqp connect retry, times: 6 , duration: 640ms
amqp connect retry, times: 7 , duration: 1.28s
amqp connect retry, times: 8 , duration: 2.56s
amqp connect retry, times: 9 , duration: 5.12s
amqp connect retry, times: 10 , duration: 10.24s
amqp connect retry, times: 11 , duration: 20.48s
```

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.3.9. PHP SDK接入示例

本文介绍使用PHP SDK接入阿里云物联网平台，接收服务端订阅消息的示例。

前提条件

已获取消费组ID，并订阅Topic消息。

- 管理消费组：您可使用物联网平台默认消费组（DEFAULT_GROUP）或创建消费组。
- 配置AMQP服务端订阅：您可通过消费组订阅需要的Topic消息。

下载SDK

本示例提供基于Stomp PHP库的代码示例，使用STOMP协议和物联网平台云端通信。请访问Stomp PHP下载客户端和查看使用说明。

因Stomp PHP 5.0.0以下版本存在SDK断开后可能无法重连问题，建议您下载Stomp PHP 5.0.0或其以上版本SDK。详细说明，请参见Issues。

您可在PHP项目文件目录下，执行以下命令，下载Stomp PHP 5.0.0版本的SDK。

```
composer require stomp-php/stomp-php 5.0.0
```

代码示例

```
<?php
require __DIR__ . '/vendor/autoload.php';
use Stomp\Client;
use Stomp\Network\Observer\Exception\HeartbeatException;
use Stomp\Network\Observer\ServerAliveObserver;
use Stomp\StatefulStomp;
//参数说明，请参见AMQP客户端接入说明文档。
$accessKey = "${YourAccessKeyId}";
$accessSecret = "${YourAccessKeySecret}";
$consumerGroupId = "${YourConsumerGroupId}";
$clientId = "${YourClientId}";
//iotInstanceId：实例ID。
$iotInstanceId = "${YourIotInstanceId}";
$timeStamp = round(microtime(true) * 1000);
//签名方法：支持hmacmd5，hmacsha1和hmacsha256。
$signMethod = "hmacsha1";
//userName组装方法，请参见AMQP客户端接入说明文档。
//若使用二进制传输，则userName需要添加encode=base64参数，服务端会将消息体base64编码后再推送。具体添加方法请参见下一章节“二进制消息体说明”。
$userName = $clientId . "|authMode=aksign"
            . ",signMethod=" . $signMethod
            . ",timestamp=" . $timeStamp
            . ",authId=" . $accessKey
            . ",iotInstanceId=" . $iotInstanceId
            . ",consumerGroupId=" . $consumerGroupId
            . "|";
$signContent = "authId=" . $accessKey . "&timestamp=" . $timeStamp;
//计算签名，password组装方法，请参见AMQP客户端接入说明文档。
$password = base64_encode(hash_hmac("sha1", $signContent, $accessSecret, $raw_output = TRUE)
);
//接入域名，请参见AMQP客户端接入说明文档。
$client = new Client('ssl://${YourHost}:61614');
$sslContext = ['ssl' => ['verify_peer' => true, 'verify_peer_name' => false], ];
$client->getConnection()->setContext($sslContext);
//服务端心跳监听。
$observer = new ServerAliveObserver();
$client->getConnection()->getObservers()->addObserver($observer);
//心跳设置，需要云端每30s发送一次心跳包。
```

```
//心跳设置，需要云端与30S发送一次心跳包。
$client->setHeartbeat(0, 30000);
$client->setLogin($userName, $password);
try {
    $client->connect();
}
catch(StompException $e) {
    echo "failed to connect to server, msg:" . $e->getMessage() , PHP_EOL;
}
//无异常时继续执行。
$stomp = new StatefulStomp($client);
$stomp->subscribe('/topic/#');
echo "connect success";
while (true) {
    try {
        // 检查连接状态
        if (!$client->isConnected()) {
            echo "connection not exists, will reconnect after 10s.", PHP_EOL;
            sleep(10);
            $client->connect();
            $stomp->subscribe('/topic/#');
            echo "connect success", PHP_EOL;
        }
        //处理消息业务逻辑。
        echo $stomp->read();
    }
    catch(HeartbeatException $e) {
        echo 'The server failed to send us heartbeats within the defined interval.', PHP_EOL
;
        $stomp->getClient()->disconnect();
    } catch(Exception $e) {
        echo 'process message occurs error: '. $e->getMessage() , PHP_EOL;
        $stomp->getClient()->disconnect();
    }
}
```

您需按照如下表格中的参数说明，修改代码中的参数值。更多参数说明，请参见AMQP客户端接入说明。

参数	示例	说明
accessKey	LTAI4GFGQvKuqHJhFa******	登录物联网平台控制台，将鼠标移至账号头像上，然后单击**AccessKey管理**，获取AccessKey ID和AccessKey Secret。 **说明** 如果使用RAM用户，您需授予该RAM用户管理物联网平台的权限（AliyunIOTFullAccess），否则将连接失败。授权方法请参见授权RAM用户访问物联网平台。
accessSecret	iMS8ZhCDdfJbCMeA005sieKe**********	

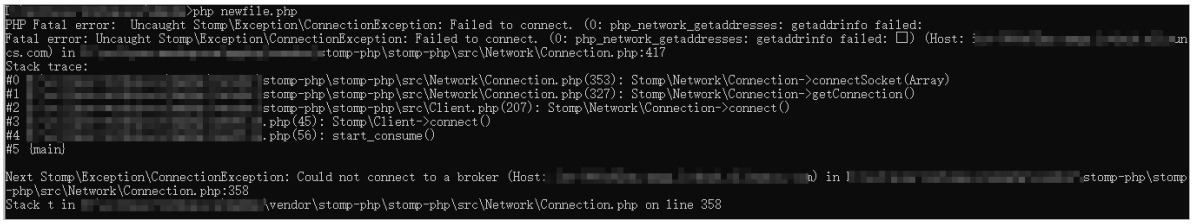
参数	示例	说明
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	消费组ID。 登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**查看您的消费组ID。
iotInstanceId	iot-***j	实例ID。您可在物联网平台控制台的**实例概览**页面，查看当前实例的ID。 • 若有ID值，必须传入该ID值。 • 若无ID值，传入空值，即 `iotInstanceId = ""` 。
clientId	12345	表示客户端ID，需您自定义，长度不可超过64个字符。建议使用您的AMQP客户端所在服务器UUID、MAC地址、IP等唯一标识。 AMQP客户端接入并启动成功后，登录物联网平台控制台，在对应实例的**规则引擎 > 服务端订阅 > 消费组列表**页签，单击消费组对应的**查看**，**消费组详情**页面将显示该参数，方便您识别区分不同的客户端。
client	new Client('ssl://iot-cn-***.amqp.iothub.aliyuncs.com:61614')	创建AMQP客户端与物联网平台的连接： `$client = new Client('ssl://${YourHost}:61614');` `${YourHost}` 对应的AMQP接入域名信息，请参见查看实例终端节点。

运行结果示例

- 成功：返回类似如下日志信息，表示AMQP客户端已接入物联网平台并成功接收消息。

- 失败：返回类似如下日志信息，表示AMQP客户端连接物联网平台失败。

  您可根据日志提示，检查代码或网络环境，然后修正问题，重新运行代码。



二进制消息体说明

当您需要传输二进制数据时，由于STOMP协议为文本协议，需要使用base64编码参数，否则消息体可能会被截断。

本示例中，userName需要按以下方法添加encode=base64参数，使服务端将消息体base64编码后再推送。

```
$userName = $clientId . "|authMode=aksign"
                . ",signMethod=" . $signMethod
                . ",timestamp=" . $timeStamp
                . ",authId=" . $accessKey
                . ",iotInstanceId=" . $iotInstanceId
                . ",consumerGroupId=" . $consumerGroupId
                . ",encode=base64" . "|";
```

相关文档

服务端订阅消息相关错误码，请参见消息相关错误码。

3.4. 使用MNS服务端订阅

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

前提条件

如果使用子账号，子账号需拥有 `AliyunIOTAccessingMNSRole` 角色权限。

> **说明** 华北2（北京）、华南1（深圳）地域不支持MNS服务端订阅。

操作步骤

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

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

iii. 在左侧导航栏，选择**规则引擎 > 服务端订阅**。

iv. 在**服务端订阅**页的**订阅列表**页签下，单击**创建订阅**。

v. 在**创建订阅**对话框中，完成配置，单击**确认**。

参数	说明
产品	选择订阅消息源设备所属的产品。
订阅类型	选择为**MNS**。
推送消息类型	服务端要订阅的消息类型。目前，服务端可订阅的设备消息类型包括： ■ **设备上报消息**：产品下所有设备Topic列表中，**操作权限**为**发布**的Topic中的消息。 设备上报消息，包括设备上报的自定义数据和物模型数据（属性上报、事件上报、属性设置响应和服务调用响应）。推送到服务端的物模型数据是经物联网平台系统处理过后的数据，数据格式请参见数据格式。 例如，一个产品有3个Topic类，分别是： ■ `/${productKey}/${deviceName}/user/get`，具有订阅权限。 ■ `/${productKey}/${deviceName}/user/update`，具有发布权限。 ■ `/${productKey}/${deviceName}/thing/event/property/post`，具有发布权限。 那么，服务端订阅会推送具有发布权限的Topic类中的消息，即 `/${productKey}/${deviceName}/user/update` 和 `/${productKey}/${deviceName}/thing/event/property/post` 中的消息。 ■ **设备状态变化通知**：该产品下的设备上下线状态变化时通知的消息。 ■ **网关子设备发现上报**：网关将发现的子设备信息上报给物联网平台。需要网关上的应用程序支持。网关产品特有消息类型。 ■ **设备拓扑关系变更**：子设备和网关之间的拓扑关系建立和解除消息。网关产品特有消息类型。 ■ **设备生命周期变更**：设备创建、删除、禁用、启用等消息。 ■ **物模型历史数据上报**：设备上报的属性和事件历史数据。 ■ **OTA升级状态通知**：验证升级包和批量升级时，设备升级成功或失败的事件通知。

vi. 在弹出的确认对话框中，单击**确认**。

物联网平台将自动创建MNS消息队列，名称格式为 `aliyun-iot-${productKey}` 。您在配置监听MNS队列时，需配置为该队列。

MNS会收取费用，具体计费方式，请参见MNS计费。

> **说明** 如果删除已创建的MNS服务端订阅，对应的MNS队列也会自动删除。

2. 配置MNS客户端，监听MNS队列，以接收设备消息。

以下示例中，使用MNS Java SDK监听消息。

下载MNS SDK Demo，请访问MNS文档。

i. 在pom.xml文件中，添加如下依赖安装MNS Java SDK。

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

ii. 配置接收消息时，需填入以下信息。

```
CloudAccount account = new CloudAccount( $AccessKeyId, $AccessKeySecret, $AccountEnd
point);
```

- $AccessKeyId和$AccessKeySecret需替换为您的阿里云账号访问API的基本信息。在物联网平台控制台，鼠标移动到您的账号头像上，然后单击**AccessKey管理**，创建或查看AccessKey。
- $AccountEndpoint需填写实际的Endpoint值。在MNS控制台，单击**获取Endpoint**获取。

iii. 填写接收设备消息的逻辑。

```
MNSClient client = account.getMNSClient();
CloudQueue queue = client.getQueueRef("aliyun-iot-a1xxxxxx8o9"); //请输入物联网平台自
动创建的队列名称。
    while (true) {
    // 获取消息。
    Message popMsg = queue.popMessage(10); //长轮询等待时间为10秒。
    if (popMsg != null) {
        System.out.println("PopMessage Body: "+ popMsg.getMessageBodyAsRawString());
//获取原始消息。
        queue.deleteMessage(popMsg.getReceiptHandle()); //从队列中删除消息。
    } else {
        System.out.println("Continuing"); } }
```

iv. 运行程序，完成对MNS队列的监听。

3. 启动设备，上报消息。

   设备端SDK开发，请参见Link SDK文档。

4. 检查云端应用是否监听到设备消息。若成功监听，将获得如下所示消息代码。

```
{
"messageid":" ",
"messagetype":"upload",
"topic":"/a112345****/device123/user/update",
"payload":" ",
"timestamp": " "
}
```

参数	说明
messageid	物联网平台生成的消息ID。

参数	说明
messagetype	消息类型。 取值： ○ upload：设备上报消息 ○ status：设备状态变化通知 ○ topo_listfound：网关子设备发现上报 ○ topo_lifecycle：设备拓扑关系变更 ○ device_lifecycle：设备生命周期变更 ○ thing_history：物模型历史数据上报 ○ ota_event：OTA升级状态通知
topic	服务端监听到的信息来源的物联网平台Topic。
payload	Base64编码的消息数据。 payload数据格式，请参见数据格式。
timestamp	时间戳，以Epoch时间表示。

相关文档

- 数据格式
- 服务端订阅（MNS）

4.云产品流转

4.1. 云产品流转概述

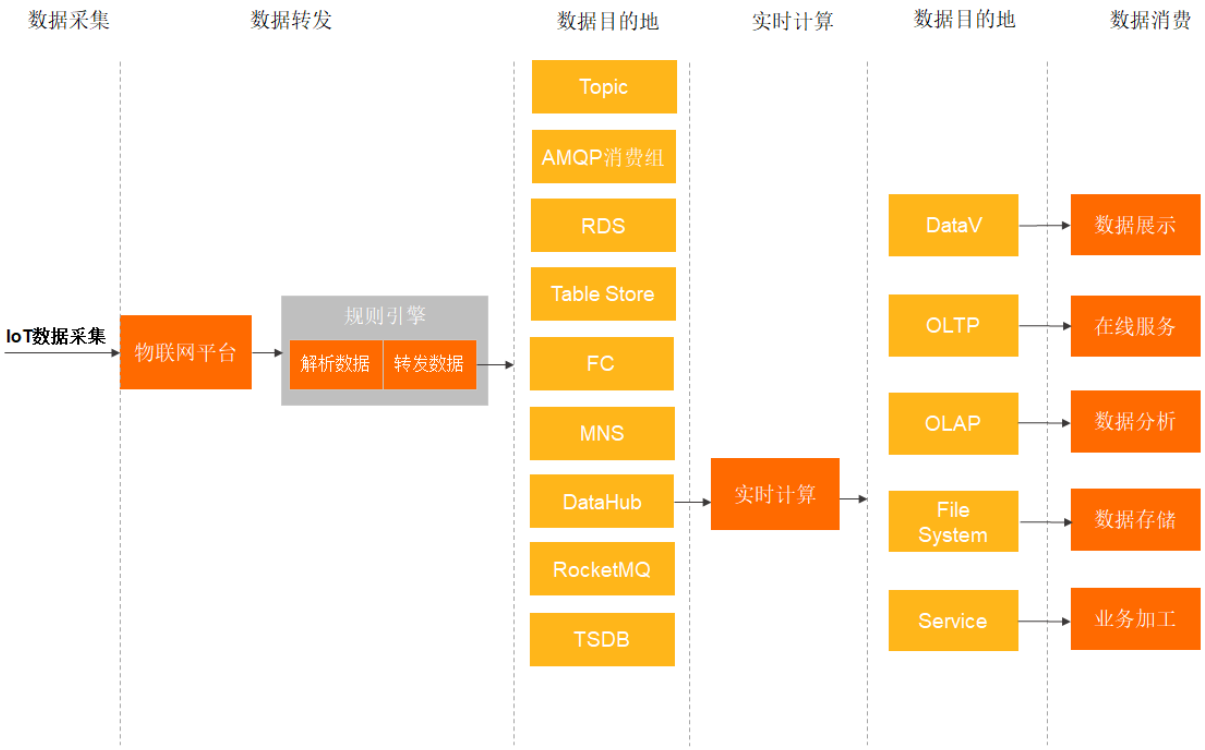
使用物联网平台的数据流转功能，可将Topic中的数据消息转发至其他Topic或其他阿里云产品进行存储或处理。

什么是云产品流转

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

- 将数据转发到另一个设备的Topic中，可以实现设备间的M2M通信。
- 将数据转发到AMQP服务端订阅消费组，您的服务器通过AMQP客户端监听消费组中的消息。AMQP客户端配置操作指导，请参见AMQP客户端接入说明。
- 将数据转发到RDS、表格存储、TSDB中进行存储。
- 将数据转发到DataHub中，然后使用实时计算进行流计算，使用MaxCompute进行大规模离线计算。
- 将数据转发到函数计算进行事件计算。
- 可以转发到消息队列RocketMQ、消息服务实现高可靠消费数据。

使用数据流转功能后，您无需购买服务器部署分布式架构，即可实现采集—计算—存储的全栈服务。



数据流转使用指南

- 设置数据流转规则：如何设置一条数据转发规则。
- SQL表达式：规则中SQL表达式的写法详解。
- 函数列表：规则中SQL表达式支持的函数列表。
- 数据流转过程：进行数据流转的过程和各阶段的数据格式。

- 数据格式：基础通信Topic、物模型通信Topic消息经物模型解析后的数据格式。数据流转规则中，SQL字段需按照解析后的数据格式编写。
- 地域和可用区：查看不同地域和可用区所支持的目的云产品。

相关文档

云产品流转（新版）

4.2. 数据流转方案对比

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

目前，数据流转方式有两类：

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

云产品流转与服务端订阅对比

目前，数据流转方式有两类：

- 云产品流转：对设备数据进行过滤并转换，然后再流转到其他阿里云云产品实例。
- 服务端订阅：通过AMQP或消息服务（MNS）客户端直接获取设备消息。

两者的对比，如下表所示。

流转方式	使用场景	优缺点	使用限制
云产品流转	• 需将设备数据消息转发至其他Topic或其他阿里云产品进行存储或处理的场景。 • 海量吞吐量场景。	优点： • 功能相对完备。 • 支持在规则运行时，调整流转规则。 • 支持对数据进行简单过滤处理。 • 支持将数据流转至其他阿里云云产品。 数据流转至其他阿里云云产品简要对比，请参见下表云产品流转各方案对比。 缺点： 需编写SQL和配置规则，使用相对复杂。	请参见云产品流转使用限制。
服务端订阅	• 仅需接收设备数据的场景。 • 服务端接收产品下已订阅的全部设备数据。	优点：相对简单易用且高效。 缺点：缺少过滤和转换能力。	请参见服务端订阅使用限制。

云产品流转各方案对比

流转目标	使用场景	优点	缺点
消息队列RocketMQ	要对设备数据进行复杂或精细化处理的海量设备场景。 设备消息量>1,000 QPS的场景，推荐使用消息队列 RocketMQ。	• 稳定可靠。 • 支持海量数据。	公网支持略差（铂金版公网性能较好）。
消息服务（MNS）	公网环境场景下，对设备数据进行复杂或精细化处理。 设备消息量<1,000 QPS的场景，推荐使用MNS。	• 采用HTTPS协议。 • 公网支持较好。	性能略低于消息队列RocketMQ。
云数据库RDS版	适合单纯的数据保存场景。	数据直接写入数据库。	不涉及
时序数据库（TSDB）	适合根据设备数据进行业务分析和监控的场景。	数据直接写入时序数据库。	不涉及
DataHub	适合需对数据进行分析处理的场景。	数据直接写入DataHub。	不涉及
表格存储（Tablestore）	适合单纯的数据存储场景。	数据直接写入表格存储实例。	不涉及
函数计算（Function Compute）	需要简化设备开发过程，且对设备数据进行一定自由度的处理的场景。	• 数据处理自由度高。 • 功能多。 • 无需部署。	费用略高。

服务端订阅

服务端可以通过AMQP SDK或消息服务（MNS）SDK接收已订阅的产品下所有类型的消息，包含设备上报消息、设备状态变化通知、设备生命周期变更、物模型历史数据上报、OTA升级状态通知、网关发现子设备上报、设备拓扑关系变更等。

使用限制	使用注意	相关文档
• 不支持细粒度的过滤及订阅功能，只能接收租户下全部的消息。 • 目前，单消费组的QPS限制为1,000 QPS。 详细的服务端订阅使用限制，请参见服务端订阅使用限制。	• 服务端订阅的特性是保障实时消息优先，会对堆积消息降级处理。请确保消费端能及时消费。 • 服务端订阅不能满足需对数据进行高级过滤以及精细化处理的场景，这类场景推荐使用规则引擎。	• 服务端订阅概述 • 配置AMQP服务端订阅 • AMQP客户端接入说明 • 使用MNS服务端订阅

云产品流转到消息队列RocketMQ

通过云产品流转功能，将物联网平台中指定Topic的消息流转到RocketMQ中的Topic，然后通过RocketMQ的SDK接收相应的消息。

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

使用限制	使用注意	相关文档
• RocketMQ的公网测试集群不能用于生产环境。 • RocketMQ的非公网Endpoint无法用于本地调试，需要在该地域的ECS中运行调试。	• RocketMQ的公网测试集群可以支持开发者本地接收消息，但是稳定性很差，请勿用于线上生产环境。 • 对于较大的设备消息量场景（>5,000 QPS）或者稳定性要求特别高的场景，推荐使用RocketMQ铂金版。 • 规则引擎在数据流转失败，再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的影响防护。	• 设置数据流转规则 • 数据转发到消息队列RocketMQ • 设备消息通过RocketMQ流转到服务器 • RocketMQ使用指南

云产品流转到消息服务（MNS）

可以通过云产品流转功能，将指定Topic的消息流转到MNS的主题中，然后通过MNS的SDK接收相应的消息。MNS对公网环境支持友好。设备消息量不是特别大（<1,000QPS），推荐使用MNS。

使用限制	使用注意	相关文档
请参见MNS使用限制中，主题相关使用限制。	规则引擎在数据流转失败，再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的防护。	• 设置数据流转规则 • 数据转发到消息服务 • MNS使用指南

云产品流转到函数计算

通过云产品流转，将指定Topic的消息转入到函数计算中，开发者可以进一步对消息进行处理。函数计算免部署，可以简化业务的开发。

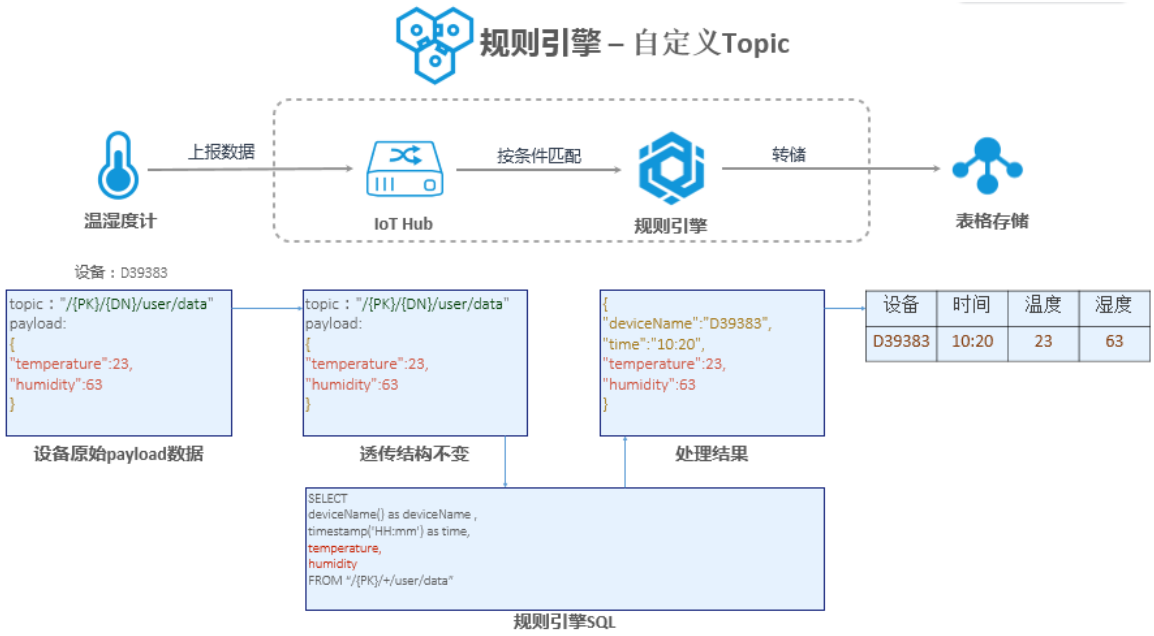
使用限制	使用注意	相关文档
请参见函数计算使用限制。	• 适用于对于设备消息有定制化的处理需求或者需简化开发运维的场景。 • 规则引擎在数据流转失败，再重试失败数次后，会丢弃消息。业务场景一定要做好消息丢失或者延迟送达的影响防护。	• 设置数据流转规则 • 数据转发到函数计算 • 实践案例（温湿度计上报数据到钉钉群机器人） • 函数计算使用指南

4.3. 数据流转过程

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

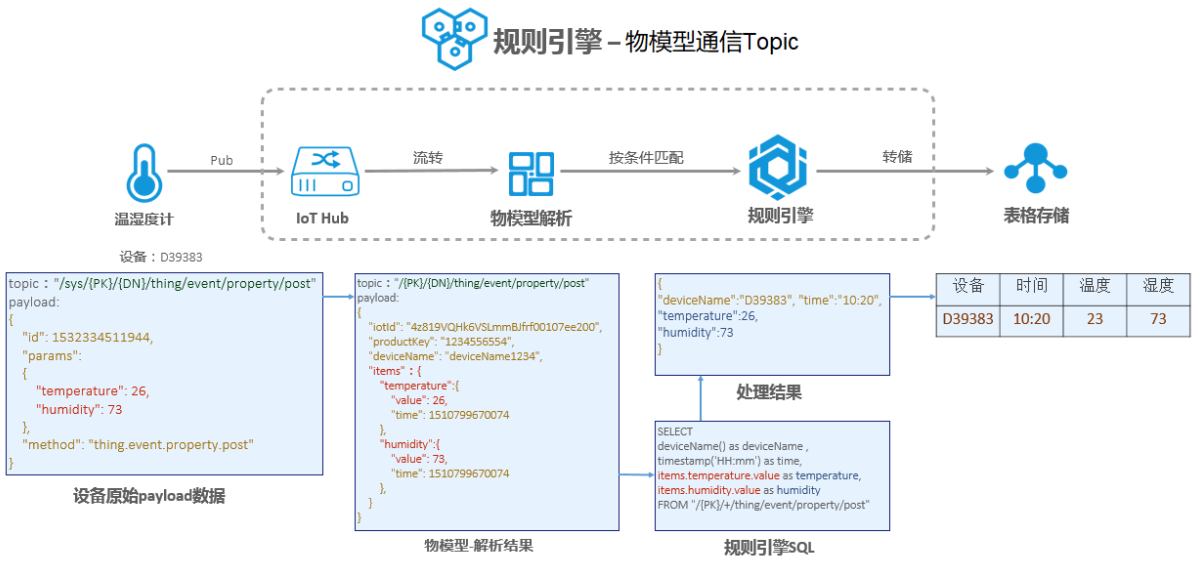
自定义Topic

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



物模型通信Topic

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



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

4.4. 设置数据流转规则

通过规则引擎的云产品流转功能，物联网平台可将指定Topic的数据，流转至其他的Topic和阿里云产品中。本文将为您介绍设置数据流转规则的完整操作步骤，依次是创建规则、编写处理数据的SQL、设置数据流转目的地、设置流转失败的数据转发目的地。

操作步骤

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

   > 注意　目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 在**云产品流转**页面，单击**创建规则**。

   > 注意　若当前页面显示新版功能，先单击右上角**返回旧版**，进入旧版功能页面，再单击**创建规则**。

5. 填写参数后，单击**确认**。

参数	描述
规则名称	输入规则名称。支持中文、英文字母、日文、数字、下划线（_）和短划线（-），长度为1~30个字符，一个中文或日文占2个字符。

参数	描述
数据格式	选择该规则处理数据的格式。可选：**JSON**和**二进制**。 **说明** ○ 因数据流转基于Topic处理数据，此处的数据格式需与被处理Topic中的数据格式保持一致。 ○ 若选择为二进制，该规则不能处理基础通信Topic、物模型通信Topic的消息，且不能将数据转发至实例内的时序数据存储、时序数据库、表格存储和云数据库RDS版。
规则描述	规则描述信息。

6. 规则创建成功后，将跳转到**数据流转规则**页面。您需编辑处理消息数据的SQL、设置数据转发目的地、流转失败数据转发目的地。
   i. 单击**编写SQL**，编写处理消息字段的SQL。

   SQL编写方法，可参见SQL表达式和函数列表。

参数	描述
规则查询语句	系统会在这里，根据您设置的字段、Topic和条件自动补充完整规则查询语句。
字段	指定要处理的消息内容字段，即SQL中SELECT后的内容。 例如，填入*deviceName() as deviceName*，则表示需筛选出消息中的deviceName字段内容。字段中可使用的函数，请参见函数列表。 **说明** 基础通信Topic、物模型通信Topic中的数据为Alink JSON格式，流转到规则引擎前，会经过物模型解析，具体说明，请参见数据流转过程。解析后的数据格式，请参见数据格式。编写SQL字段时，需按照解析后的数据格式来编写。
Topic	选择需要处理的消息Topic，即SQL中FROM后的内容。可选Topic，请参见下表*Topic说明*。 **注意** 当规则的**数据格式**是**二进制**时，仅支持选择**自定义**。
条件	设置规则触发条件，即SQL中WHERE后的内容。

Topic说明

Topic	说明	相关文档

Topic	说明	相关文档
自定义	流转自定义数据格式消息的Topic，与自定义Topic的格式相同。格式为：`/${productKey}/${deviceName}/user/${TopicShortName}`。 其中`${TopicShortName}`为自定义的Topic类，即自定义Topic的后缀。 支持使用通配符（+）和（#）： ■ 全部设备（`+`）：指定产品下所有设备。 ■ `/user/#`：指定设备的所有自定义Topic。	自定义Topic
设备状态变化通知	流转设备上下线状态变更消息的Topic：`/as/mqtt/status/${productKey}/${deviceName}`。	设备上下线状态
物模型数据上报	包含： ■ 流转设备上报属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/post`。 ■ 流转设备上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`。 ■ 流转设备批量上报属性数据的Topic：`/${productKey}/${deviceName}/thing/property/batch/post`。 ■ 流转设备批量上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/batch/post`。 ■ 流转设备响应云端命令返回消息的Topic：`/${productKey}/${deviceName}/thing/downlink/reply/message`。	■ 设备上报属性 ■ 设备上报事件 ■ 设备属性批量上报 ■ 设备事件批量上报 ■ 设备下行指令结果
	对应设备Topic如下： ■ 设备上报属性的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/post`。 ■ 设备上报事件的Topic：`/sys/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`、`/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:{tsl.event.identifier}/post`。 ■ 设备批量上报属性、事件数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/batch/post`。	■ 设备上报属性 ■ 设备上报事件 ■ 设备批量上报属性、事件
设备生命周期变更	流转设备创建、删除、禁用、启用等消息的Topic：`/${productKey}/${deviceName}/thing/lifecycle`。	设备生命周期变更
网关发现子设备上报	网关设备特有的Topic：`/${productKey}/${deviceName}/thing/list/found`，将发现的子设备信息上报给物联网平台，然后进行流转。	网关发现子设备

Topic	说明	相关文档
设备拓扑关系变更	网关设备特有Topic：`/${productKey}/${deviceName}/thing/topo/lifecycle`，流转子设备和网关之间的拓扑关系建立和解除消息的Topic。	设备拓扑关系变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/topo/change`。	通知网关拓扑关系变化
设备标签变更	流转设备标签信息变更的Topic：`/${productKey}/${deviceName}/thing/deviceinfo/update`。	设备标签变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/update`。	标签信息上报
物模型历史数据上报	包含： ■ 流转设备上报历史属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/history/post`。 ■ 流转设备上报历史事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/history/post`。	■ 历史属性上报 ■ 历史事件上报
	对应设备上报物模型历史数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/history/post`。	物模型历史数据上报
OTA升级设备状态通知	包含： ■ 流转设备上报OTA升级结果的Topic：`/${productKey}/${deviceName}/ota/upgrade`。 ■ 流转设备上报OTA升级进度的Topic：`/${productKey}/${deviceName}/ota/progress/post`。	■ OTA升级状态通知 ■ OTA升级进度通知
	对应设备上报升级进度的Topic：`/ota/device/progress/${productKey}/${deviceName}`。	设备上报升级进度
OTA模块版本号上报	流转设备上报OTA模块版本号变更的Topic：`/${productKey}/${deviceName}/ota/version/post`。	OTA模块版本号变更通知
	对应设备上报OTA模块版本的Topic：`/ota/device/inform/${productKey}/${deviceName}`。	设备上报OTA模块版本
OTA升级批次状态通知	物联网平台通知OTA升级批次状态变化的Topic：`/${productKey}/${packageId}/${jobId}/ota/job/status`。	OTA升级批次状态通知

Topic	说明	相关文档
任务事件	包含： ■ 流转设备任务状态通知的Topic：`/sys/uid/${uid}/job/${jobId}/lifecycle`。 ■ 流转实例迁移任务状态通知的Topic：`/sys/uid/${uid}/distribution/${jobId}/lifecycle`。 **说明** 迁移产品的名称为实例迁移的任务名称。	■ 设备任务的状态通知 ■ 实例迁移任务的状态通知
孪生节点属性变更	流转数字孪生节点属性数据的Topic：`/sys/uid/${uid}/digitaltwin/${dtInstanceId}/${nodeId}/property/update`。	数字孪生节点属性变更

ii. 单击**转发数据**一栏的**添加操作**，设置数据转发目的地。数据转发配置的具体示例，请参见数据流转使用示例目录下的具体文档。

说明 最多可为一个规则创建10个数据转发操作。

数据转发时，因选择的数据目的地（云产品）出现异常情况导致转发失败时：

- 转发到RocketMQ、RDS、TSDB等类型的云产品时，若云产品资源变化导致云产品无法访问，系统将停止执行转发，并在规则列表中显示状态为**规则异常**。您需要重新设置数据转发目的地。
- 其他异常情况，系统将间隔1秒、3秒、10秒进行3次重试（重试策略可能会调整）。3次重试均失败后，消息会被丢弃。如果您对消息可靠性要求比较高，可以进行：添加错误操作，将重试失败的消息，转发到其他云产品中。

iii. 单击**转发错误操作数据**一栏的**添加错误操作**，设置参数，将重试失败的错误消息转发至指定位置。

注意

- 最多支持添加一个错误操作。
- 正常操作和错误操作的转发目的地不能是相同的云产品。例如，不能同时转发到表格存储。
- 错误消息转发失败后，不会再进行重试。
- 这里的错误消息仅针对因其他云产品实例问题导致的规则引擎转发失败错误。

消息转发至云产品失败后，会进行重试。若重试失败，将根据错误操作数据转发的设置转发错误消息。

错误消息格式：

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

错误消息参数说明如下：

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

7. 所有设置完成后，返回至**云产品流转**页面，单击规则对应的**启动**。规则启动后，数据即可按照规则进行转发。

   您也可以单击：

   - **查看**：在**数据流转规则**页面，修改规则的具体设置。

- **删除**：删除对应规则。

  > **说明** 运行中的规则不可删除。

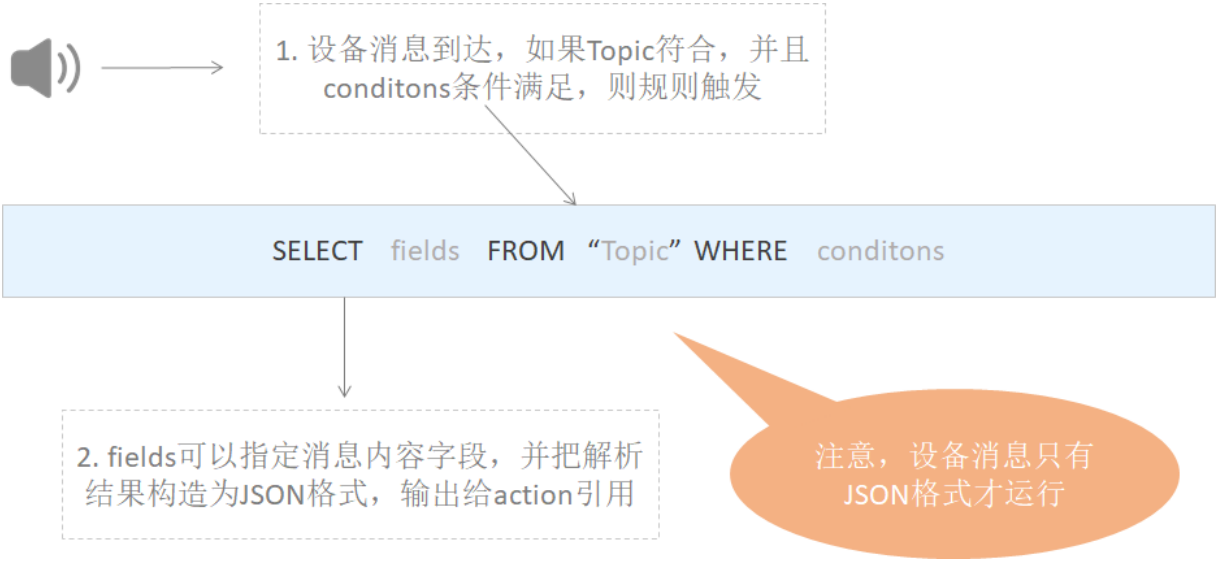
- **停止**：停止对应规则转发数据。

4.5. SQL表达式

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

SQL表达式

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



数据流转规则SQL示例：

- 处理自定义Topic数据的SQL示例。

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

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

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

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

- 处理基础通信Topic、物模型通信Topic数据的SQL示例。基础通信Topic、物模型通信Topic中的数据流转到规则引擎后，规则引擎会对数据进行解析，解析后的数据格式，请参见数据格式。

某温湿度传感器的属性如下：

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

温湿度传感器上报的温度和湿度属性数据经规则引擎解析后，数据格式如下：

```
{
    "deviceType": "TemperatureHumidityDetector",
    "iotId": "N5KURkKdibnZvSls****000100",
    "productKey": "a15NNfl****",
    "gmtCreate": 1564569974224,
    "deviceName": "N5KURkKdibnZvSls3Yfx",
    "items": {
        "CurrentHumidity": {
            "value": 70,
            "time": 1564569974229
        },
        "CurrentTemperature": {
            "value": 23.5,
            "time": 1564569974229
        }
    }
}
```

说明 SQL查询时，必须通过 `items.${属性标识符}.value` 来访问指定属性的数据。

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


```
SELECT deviceName() as deviceName, items.CurrentHumidity.value as Humidity, items.Current
Temperature.value as Temperature
FROM "/sysa15NNfl****/N5KUR***/thing/event/property/post"
WHERE items.CurrentTemperature.value > 38
```

如果是自定义模块（testFB）下属性，属性标识符格式为 **模块标识符:属性标识符** ，访问指定属性数据时，必须使用英文双引号（“”），SQL语句如下：

```
SELECT deviceName() as deviceName, "items.testFB:CurrentHumidity.value" as Humidity, "ite
ms.testFB:CurrentTemperature.value" as Temperature
FROM "/sysa15NNfl****/N5KUR***/thing/event/property/post"
WHERE "items.testFB:CurrentTemperature.value" > 38
```

SELECT

- JSON数据格式

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

  支持 `*` 和函数的组合。不支持子SQL查询。

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

  本文SQL示例中：

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

  > **说明** `items.testFB:CurrentHumidity.value` 和 `items.testFB:CurrentTemperature.value` 来自于上报的自定义模块属性数据中的字段。

- 二进制数据格式
  - 可填 `*` 直接透传数据。 `*` 后不能再使用函数。
  - 可使用内置函数，如 `to_base64(*)` 函数，将原始payload二进制数据转成base64的String提取出来； `deviceName()` 函数，将设备名称信息提取出来。

> **说明** SELECT语句中最多可包含50个字段。

FROM

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

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

以上SQL示例中：

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

WHERE

- JSON数据格式

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

  上文两个示例中，条件语句 `WHERE temperature > 38` 表示温度大于38℃时，才会触发该规则。

- 二进制数据格式

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

SQL结果

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

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

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

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

数组使用说明

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

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

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

条件表达式支持列表

操作符	描述	举例
=	相等	color = 'red'
<>	不等于	color <> 'red'
AND	逻辑与	color = 'red' AND siren = 'on'
OR	逻辑或	color = 'red' OR siren = 'on'
+	算术加法	4 + 5
-	算术减	5 - 4
/	除	20 / 4
*	乘	5 * 4
%	取余数	20 % 6
<	小于	5 < 6
<=	小于或等于	5 <= 6
>	大于	6 > 5
>=	大于或等于	6 >= 5
函数调用	支持函数，更多信息，请参见函数列表。	deviceId()

操作符	描述	举例
JSON属性表达式	可以从消息payload以JSON表达式提取属性。	state.desired.color,a.b.c[0].d
CASE ... WHEN ... THEN ... ELSE ...END	Case表达式（不支持嵌套）。	CASE col WHEN 1 THEN 'Y' WHEN 0 THEN 'N' ELSE '' END as flag
IN	仅支持枚举，不支持子查询。	例如：where a in(1,2,3)。不支持以下形式：where a in(select xxx)。
like	匹配某个字符，仅支持 % 通配符，代表匹配任意字符串。	where c1 like '%abc', where c1 not like '%def%'

调试SQL

若创建数据流转规则时，**数据格式**选择了**JSON**，可在物联网平台进行SQL在线调试。调试方法如下。

1. 编写SQL后，单击**SQL调试**。
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**调试**。

   请根据Topic上报的数据格式，输入调试用的payload数据。数据格式说明：

   - 若Topic为自定义Topic，输入的payload数据格式需与Topic上报的数据格式一致。
   - 若Topic为基础通信Topic或物模型通信Topic，请参见数据格式。

3. 单击**调试结果**，查看结果。

4.6. 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。

数据流转支持的函数

函数名	函数说明
abs(number)	返回绝对值。
asin(number)	返回number值的反正弦。
attribute(key)	返回key所对应的设备标签。如果设备没有该key对应的标签，则返回值为空。使用SQL调试时，因为没有真实设备及对应的标签，返回值为空。
concat(string1, string2)	用于连接字符串。返回连接后的字符串。 示例：`concat(field,'a')`。
cos(number)	返回number值的余弦。
cosh(number)	返回number值的双曲余弦（hyperbolic cosine）。
crypto(field,String)	对field的值进行加密。 第二个参数String为算法字符串。可选：MD2、MD5、SHA1、SHA-256、SHA-384、SHA-512。
deviceName()	返回当前设备名称。使用SQL调试时，因为没有真实设备，返回值为空。
endswith(input, suffix)	判断input的值是否以suffix结尾。
exp(number)	返回以自然常数e为底的指定次幂。
floor(number)	返回一个最接近它的整数，它的值小于或等于这个浮点数。
log(n, m)	返回自然对数。 如果不传m值，则返回`log(n)`。

函数名	函数说明
lower(string)	返回小写字符串。
mod(n, m)	n%m余数。
nanvl(value, default)	返回属性值。 若属性值为null，则返回default。
newuuid()	返回一个随机UUID字符串。
payload(textEncoding)	返回设备发布消息的payload转义字符串。 字符编码默认UTF-8，即 `payload()` 默认等价于 `payload('utf-8')` 。
power(n,m)	返回n的m次幂。
rand()	返回 `[0~1)` 之间的一个随机数。
replace(source, substring, replacement)	对某个目标列值进行替换，即用replacement替换source中的substring。 示例： `replace(field,'a','1')` 。
sin(n)	返回n值的正弦。
sinh(n)	返回n值的双曲正弦（hyperbolic sine）。
tan(n)	返回n值的正切。
tanh(n)	返回n值的双曲正切（hyperbolic tangent）。
thingPropertyFlatMap(property)	获取物模型属性对应数值，并去掉item层级，value有多个时，使用“_”拼接。当物模型属性多于50个时，云产品流转无法流转全量物模型属性，使用该函数可以拉平物模型属性结构，实现全量物模型属性流转。 property为需要获取的属性，可以传入多个property，为空则表示提取所有属性。 示例：使用 `thingPropertyFlatMap('Power', 'Position')` ，将在消息体中添加 `"Power": "On", "Position_latitude": 39.9, "Position_longitude": 116.38` 。
timestamp(format)	返回当前系统时间，格式化后返回当前地域的时间。 format为可选。如果为空，则返回当前系统时间戳毫秒值，例如，使用 `timestamp()` ，返回的时间戳为 `1543373798943` ；如果指定format，则根据指定格式，返回当前系统时间，如 `timestamp('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'')` ，返回的时间戳为 `2018-11-28T10:56:38Z` 。

函数名	函数说明
timestamp_utc(format)	返回当前系统时间，格式化后返回UTC时间。 format为可选。如果format为空，则返回当前系统时间戳毫秒值，例如，使用 `timestamp_utc()` ，返回的时间戳为 `1543373798943` ；使用 `timestamp_utc('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'')` ，返回的时间戳为 `2018-11-28T02:56:38Z` 。
topic(number)	返回Topic分段信息。 如，有一个Topic： `/alDbcLe****/TestDevice/user/set` 。使用函数 `topic()` ，则返回整个Topic；使用 `topic(1)` ，则返回Topic的第一级类目 `alDbcLe****` ；使用 `topic(2)` ，则返回第二级类目 `TestDevice` ，以此类推。
upper(string)	将字符串中的小写字母转为大写字母。 示例：函数 `upper(alibaba)` 的返回结果是 `ALIBABA`
to_base64(*)	当原始Payload数据为二进制数据时，可使用该函数，将所有二进制数据转换成base64String。
messageId()	返回物联网平台生成的消息ID。

函数名	函数说明
substring(target, start, end)	返回从start（包括）到end（不包括）的字符串。 参数说明： • target：要操作的字符串。必填。 • start：起始下标。必填。 • end：结束下标。非必填。 说明 • 目前，仅支持String和Int类型数据。Int类型数据会被转换成String类型后再处理。 • 常量字符串，请使用单引号。双引号中的数据，将视为Int类型数据进行解析。 • 如果传入的参数错误，例如参数类型不支持，会导致SQL解析失败，而放弃执行规则。 字符串截取示例： • substring('012345', 0) = "012345" • substring('012345', 2) = "2345" • substring('012345', 2.745) = "2345" • substring(123, 2) = "3" • substring('012345', -1) = "012345" • substring(true, 1.2) error • substring('012345', 1, 3) = "12" • substring('012345', -50, 50) = "012345" • substring('012345', 3, 1) = ""
to_hex(*)	当原始Payload数据为二进制数据时，可使用该函数，将二进制数据转换成十六进制字符串。
user_property()	设备使用MQTT 5.0协议通信时，获取UserProperty列表中的属性数据。 • `user_property()` 获取所有属性数据。 • `user_property('${Key}')` 获取指定Key的数据。 例如设备上报的UserProperty为 `{"a": "1", "b": "2"}` ： • `user_property()` 返回 `"{\"a\": \"1\", \"b\": \"2\"}"` 。 • `user_property('a')` 返回 `"{\"a\": \"1\"}"` 。
things_function_type()	获取上报的物模型功能类型。仅对物模型消息生效。各功能返回值如下： • 属性：返回 `property` 。 • 事件：返回事件的标识符。 • 服务：返回服务的标识符。 例如Topic： `/SdfgeW***/device1/thing/event/BrokenInfo/post` ，流转事件BrokenInfo的数据， `things_function_type()` 返回上报事件标识符 `BrokenInfo` 。

函数名	函数说明
things_property('${参数名称}')	获取物模型属性、服务、事件中参数对应的值。仅对物模型消息生效。 例如Topic：`/${productKey}/${deviceName}/thing/event/property/post`，流转上报属性中CurrentTemperature（当前温度）数据，`things_property('CurrentTemperature')`返回CurrentTemperature上报的值。

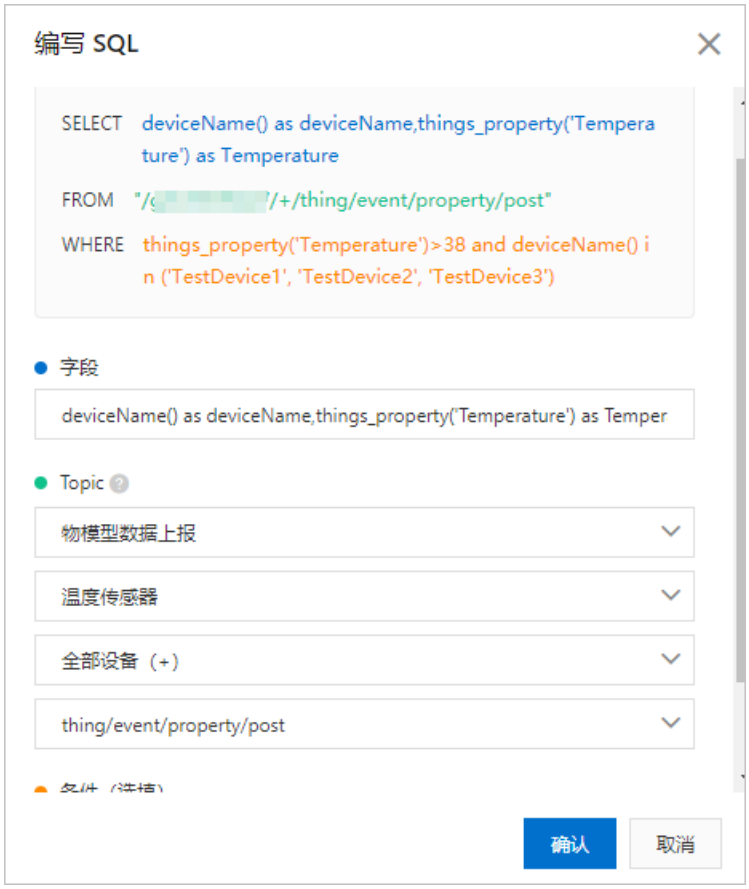
使用示例

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

例如，某温度传感器产品功能定义的默认模块有一个温度属性（Temperature），设备上报的温度属性数据经物模型转化后的数据格式如下：

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

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



SQL语句：

```
SELECT deviceName() as deviceName,things_property('Temperature') as Temperature
FROM "/g5or0***/+thing/event/property/post"
WHERE things_property('Temperature')>38 and deviceName() in ('TestDevice1', 'TestDevice2', '
TestDevice3')
```

以上示例中，使用了函数deviceName()、things_property('Temperature')：

- 在SELECT字段使用以上函数，表示从数据中筛选出设备名称，和属性Temperature的值。
- 在条件字段使用以上函数，并指定为 `>38` 、 `in ('TestDevice1', 'TestDevice2', 'TestDevice3')` ，表示仅当Temperature值大于38，且设备名称的值为TestDevice1、TestDevice2或TestDevice3中的其中一个时，才会进行数据流转。

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

4.7. 地域和可用区

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

背景信息

目前物联网平台的华东2（上海）、新加坡、日本（东京）、美国（硅谷、弗吉尼亚）、德国（法兰克福）地域默认开通公共实例服务，仅华东2（上海）、华南1（深圳）、华北2（北京）地域开通企业版实例服务。

- 对于已开通企业版实例服务的地域，在物联网平台控制台提供**实例概览**页面，展示公共实例和已购买的企业版实例列表。您可单击目标实例，进入功能界面。
- 对于未开通企业版实例服务的地域，在物联网平台控制台不提供**实例概览**页面，默认直接展示公共实例的功能界面。

公共实例

各地域和可用区支持转发的目的云产品，及对应支持的数据格式如下表所示。

- 华东2（上海）

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	支持

- 新加坡

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	支持

- 日本（东京）

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持

- 美国（硅谷、弗吉尼亚）

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持

- 德国（法兰克福）

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	不支持

企业版实例

目前仅华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。各地域和可用区支持转发的目的云产品，及对应支持的数据格式如下所示。

> **注意** 企业版实例不支持跨地域转发数据。例如华东2（上海）的物联网平台实例数据只能转发到华东2（上海）的RDS数据表中。

- 华东2（上海）

数据转发目的云产品	数据格式	
	JSON	二进制
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	支持

- 华南1（深圳）

数据转发目的云产品	数据格式	
	JSON	二进制
实例内的时序数据存储	支持	不支持
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
时序数据库（TSDB）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	支持

- 华北2（北京）

数据转发目的云产品	数据格式	
	JSON	二进制
实例内的时序数据存储	支持	不支持
消息服务（MNS）	支持	支持
消息队列RocketMQ	支持	支持
函数计算（Function Compute）	支持	支持
表格存储（Tablestore）	支持	不支持
云数据库RDS版	支持	不支持
DataHub	支持	支持

4.8. 数据流转使用示例

4.8.1. 数据转发到另一Topic

您可以设置将规则SQL处理完的数据，转发到另一个Topic中，实现M2M通信或者更多其他通信场景。

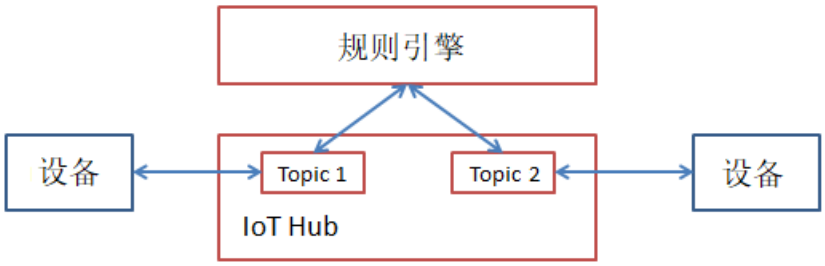
前提条件

已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

背景信息

规则引擎数据转发功能将Topic 1中的数据转发到Topic 2内。

数据流转示意图如下。



操作步骤

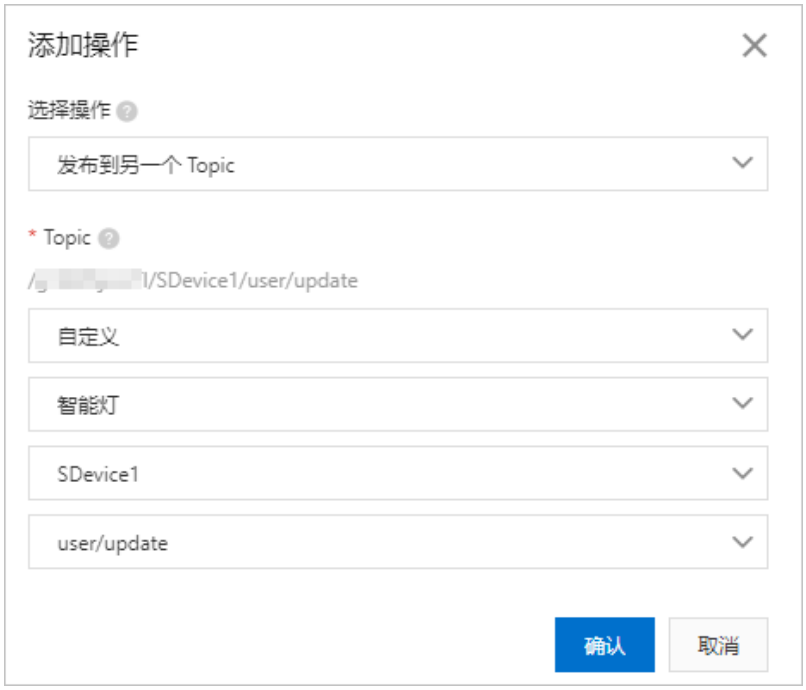
1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> 注意 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**发布到另一个Topic**。按照界面提示，设置其他信息，单击**确认**。



参数	说明
选择操作	选择**发布到另一个Topic**。
Topic	选择数据转发目的地Topic。 可选的Topic类型： ○ 自定义：目的地Topic为一个自定义Topic。该自定义Topic的设备操作权限需为**订阅**，即所属设备可订阅这个Topic，获取转发的消息。 ○ 物模型数据下发：目的地Topic为设备接收设置属性值指令的Topic：`thing/service/property/set`。设备从该Topic接收转发数据，并根据数据内容，设置属性值。用于目的地Topic所属设备根据转发的数据更改属性值的场景。 选择Topic类型后，您还需选择产品、设备和Topic名称，即指定具体Topic。 **注意** 目的地Topic所属设备若未订阅该Topic，则收不到转发的消息。 例如使用MQTT.fx工具接入物联网平台的设备A，不会自动订阅设备Topic，若设备B向设备A的Topic：`thing/service/property/set` 转发数据，而设备A未手动订阅该Topic，则设备A收不到设备B转发的消息。 设备Topic自动订阅的详细说明，请参见自动订阅Topic说明。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

基于规则引擎的M2M设备间通信

4.8.2. 数据转发到AMQP服务端订阅消费组

您可以使用规则引擎将设备发送到物联网平台的消息，经过规则SQL处理后，再转发到AMQP服务端订阅消费组，并通过AMQP客户端消费消息。

前提条件

- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。
- 已创建AMQP服务端订阅消费组，作为数据转发目的地。消费组的创建和管理，请参见管理消费组。

操作步骤

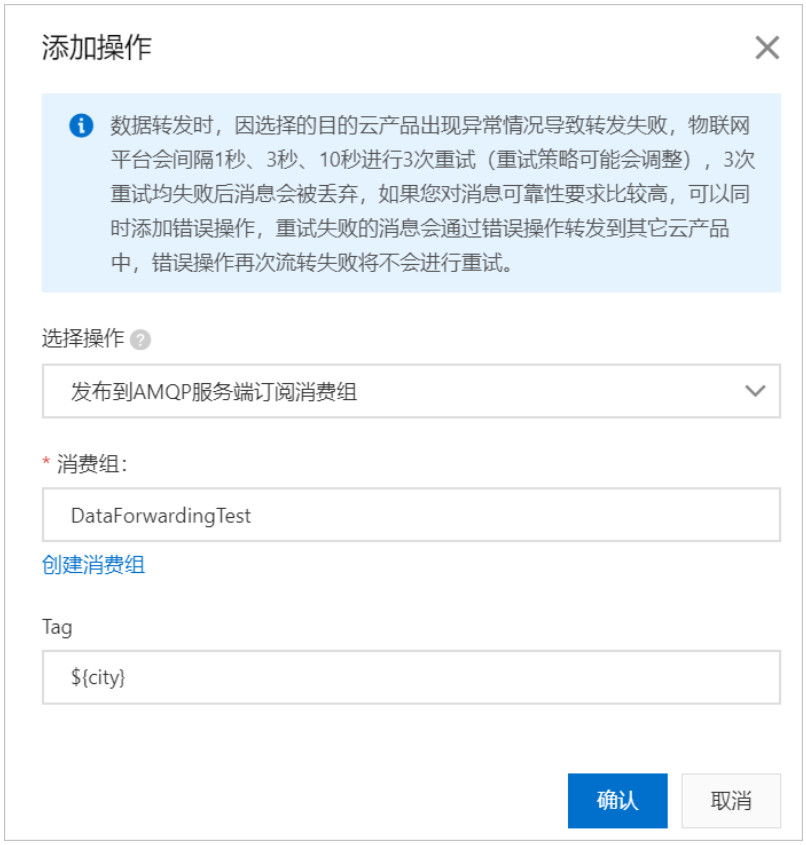
1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**发布到AMQP服务端订阅消费组**。按照界面提示，设置其他信息，单击**确认**。



参数	描述
选择操作	选择**发布到AMQP服务端订阅消费组**。
消费组	选择一个已创建的消费组作为数据转发目标。单击**创建消费组**可以进行消费组创建。
Tag	设置tag后，所有通过该操作流转到AMQP服务端订阅消费组里的消息都会携带该tag。 tag长度为1~128个字符，可以输入常量或变量。 ○ 常量支持输入中文汉字、英文字母、数字。 ○ 变量格式为${key}，代表SQL处理后的JSON数据中key对应的value值。如果取不到value值，则消息不携带tag。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

配置AMQP客户端消费消息

数据转发到AMQP消费组后，您的服务器需通过AMQP客户端消费消息。AMQP客户端开发说明，请参见AMQP客户端接入说明。

AMQP客户端开发示例，请参见：

- Java SDK接入示例
- .NET SDK接入示例
- Node.js SDK接入示例
- Python 2.7 SDK接入示例

- Python3 SDK接入示例
- PHP SDK接入示例
- Go SDK接入示例

4.8.3. 数据转发到消息队列RocketMQ

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

前提条件

- 已创建消息队列（RocketMQ）实例和用于接收数据的Topic。RocketMQ使用方法，请参见RocketMQ快速入门。

  > **注意** 目前，RocketMQ实例所在地域必须与当前物联网平台实例所在地域保持一致。存量的跨地域实例下数据流转配置不受影响，仍可继续正常流转数据。

- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

操作步骤

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**发送数据到消息队列（RocketMQ）中**。按照界面提示，设置其他信息，单击**确认**。

参数	说明
选择操作	选择**发送数据到消息队列（RocketMQ）中**。
地域	选择RocketMQ所在地域。
实例	选择RocketMQ实例。 您可以单击**创建实例**，跳转到消息队列控制台，创建RocketMQ实例，请参见消息队列文档。
Topic	选择用于接收物联网平台数据的RocketMQ Topic。 您可以单击**创建Topic**，跳转到消息队列控制台，创建RocketMQ Topic。
Tag	（可选）设置标签。 设置标签后，所有通过该操作流转到RocketMQ对应Topic里的消息都会携带该标签。您可以在RocketMQ消息消费端，通过标签进行消息过滤。 标签长度限制为128字节，可以输入常量或变量。变量格式为${key}，代表SQL处理后的JSON数据中key对应的value值。
授权	授权物联网平台将数据写入RocketMQ。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。
8. 测试。

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

设备消息通过RocketMQ流转到服务器

4.8.4. 数据转发到表格存储

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

前提条件

- 已创建表格存储实例和用于接收数据的数据表。表格存储使用方法，请参见表格存储文档。

  > 注意 企业版实例中，表格存储实例所在地域，必须与企业版实例所在地域一致。

- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

  本文示例的规则，定义了如下SQL语句：

```
SELECT items.temperature.value as temperature, items.humidity.value as humidity,deviceNam
e() as deviceName
FROM "/sys/a1ktuxe****/BZoyHO***/thing/event/property/post"
```

操作步骤

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

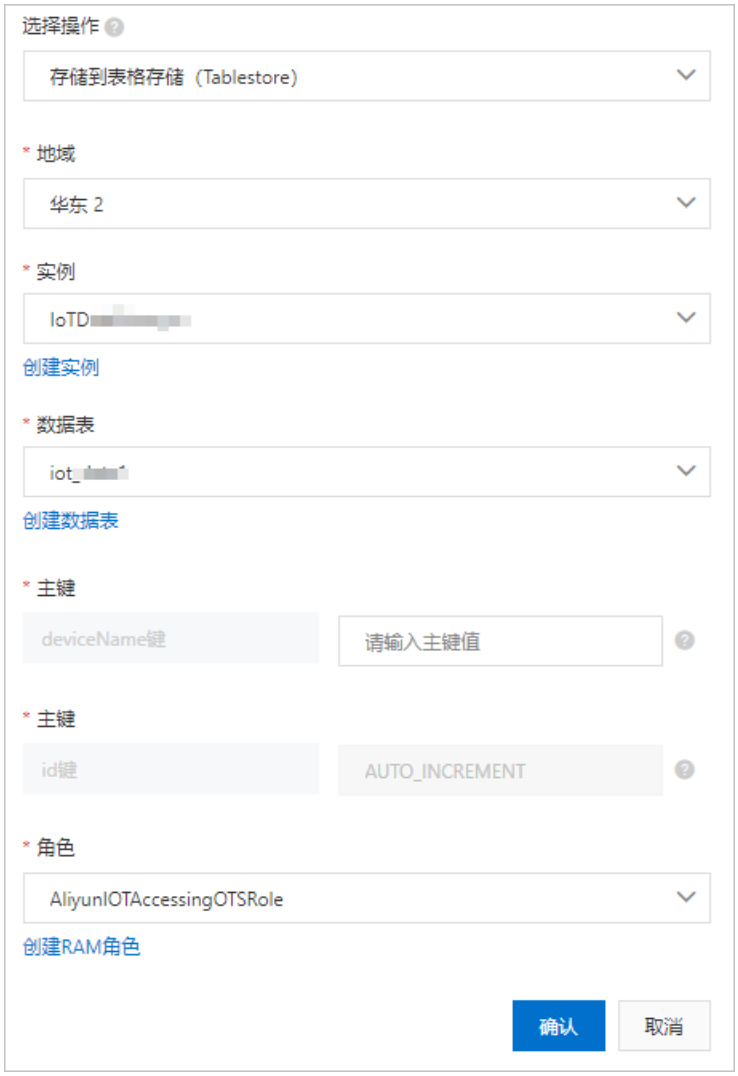
   > 注意 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，按照界面提示，设置参数，单击**确认**。

> **说明** 仅支持转发JSON格式数据。



参数	说明
选择操作	选择**存储到表格存储（Tablestore）**。
地域	选择表格存储所在地域。
实例	选择表格存储实例。 您可以单击**创建实例**，跳转到表格存储控制台，创建表格存储实例，请参见表格存储文档。

参数	说明
数据表	选择接收数据的表格存储数据表。 您可以单击**创建数据表**，跳转到表格存储控制台，创建表格存储数据表。
主键	配置表格存储数据表主键对应的值，需设置为规则SQL中SELECT的某字段值。数据流转时，该值将被存为主键对应的值。 **说明** ○ 支持配置为变量格式 `${}` ，如${deviceName}，表示该主键对应的值为消息中 `deviceName` 的值。 ○ 如果主键类型是**自增列**，这一列主键无需填值，表格存储会自动生成这一主键列的值。所以，自增列主键值，系统已自动设置为 `AUTO_INCREMENT` ，且不能编辑。 更多自增列主键说明，请参见主键列自增。
角色	授权物联网平台将数据写入表格存储。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。
8. 测试。
   i. 登录物联网平台控制台，进入对应实例。
   ii. 在左侧导航栏，选择**监控运维 > 设备模拟器**。
   iii. 选择调试用的设备，依次单击**上行指令调试**、**属性上报**、**启动设备模拟器**。

iv. 在**默认模块**下，设置测试数据，单击**发送指令**。

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

4.8.5. 数据转发到DataHub

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

前提条件

- 已创建DataHub Project和用于接收数据的Topic。DataHub使用方法，请参见DataHub文档。

  > **注意** 企业版实例中，DataHub Project所在地域，必须与企业版实例所在地域一致。

- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

操作步骤

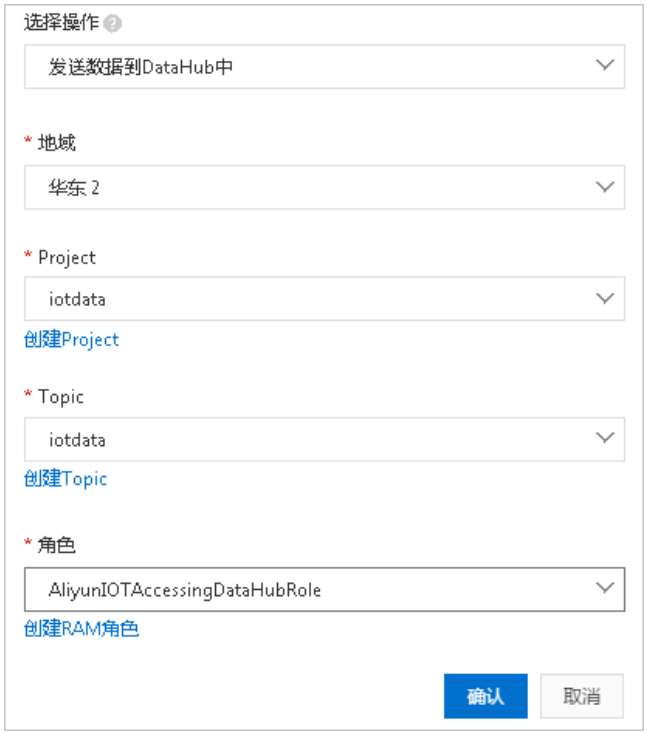
1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，按照界面提示，设置参数，单击**确认**。



参数	说明
选择操作	选择**发送数据到DataHub中**。
地域	选择DataHub所在地域。
Project	选择DataHub Project。 您可以单击**创建Project**，跳转到DataHub控制台，创建DataHub Project，请参见DataHub文档。
Topic	选择接收数据的DataHub Topic。 选择Topic后，规则引擎会自动获取Topic中的Schema，规则引擎筛选出来的数据将会映射到对应的Schema中。 **说明** ◦ 将数据映射到Schema时，需使用 `${}` ，否则存入表中的将会是一个常量。 ◦ Schema与规则引擎的数据类型必须保持一致，否则无法存储。 您可以单击**创建Topic**，跳转到DataHub控制台，创建DataHub Topic。
角色	授权物联网平台将数据写入DataHub。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

通过大数据平台搭建设备监控大屏

4.8.6. 数据转发到云数据库RDS

您可以设置规则引擎数据流转，将处理后的数据转发到云数据库RDS版（以下简称RDS）的实例中存储。本文介绍将数据流转到数据目的地的完整流程。

前提条件

- 已在您的物联网平台实例所在地域创建专有网络下的MySQL或SQL Server版RDS实例，并创建数据库和数据表。RDS使用方法，请参见云数据库RDS文档。
- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

限制说明

- 仅支持同地域转发。例如：华东2（上海）的物联网平台实例数据只能转发到华东2（上海）的RDS数据表中。
- 仅支持转发到专有网络（VPC）下的RDS实例。
- 支持MySQL实例和SQL Server实例。
- 支持普通数据库和高权限数据库的转发。
- 仅支持转发JSON格式数据。

操作步骤

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**存储到云数据库（RDS）中**。按照界面提示，设置其他信息，单击**确认**。

参数	描述
选择操作	选择**存储到云数据库（RDS）中**。
地域	固定为您的物联网平台实例所在地域。
RDS实例	选择RDS实例。
数据库	输入数据库名。 **说明** 如果是高权限数据库，需要您手动输入数据库名称。

参数	描述
账号	输入RDS实例的用户账号。此账号应具有该数据库的读写权限，否则规则引擎无法将数据写入RDS。 **说明** 规则引擎获得账号后，仅将规则匹配的数据写进数据库中，不会做其他操作。
密码	输入登录RDS实例的密码。
表名	输入数据库中已建立的数据表名。规则引擎将把数据写入这张表上。
键	输入RDS数据表的一个字段。规则引擎将把数据存入该字段中。
值	输入您在数据处理SQL中指定的Topic中的消息的一个字段，作为输入数据表字段（键）的值。 **说明** ○ 值与键的数据类型需保持一致，否则无法存储成功。 ○ 可输入一个变量，如 `${deviceName}` 。
角色	授权物联网平台将数据写入RDS。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。
8. 在RDS控制台的**数据安全性**页面，设置和查看白名单。配置完成后，规则引擎为了连接RDS，会在RDS的白名单中添加下列IP。若以下IP未出现，请手动添加。具体操作，请参见云数据库RDS文档。
   - 华东2：`100.104.53.192/26,100.104.148.64/26,100.104.6.192/26,100.104.143.128/26,100.104.76.0/24,100.104.73.128/26,100.104.200.64/26,100.104.40.64/26,100.104.3.0/26,100.104.29.128/26,100.104.121.0/26,100.104.84.64/26`
   - 亚太东南1（新加坡）：`100.104.106.0/24`
   - 美国（硅谷）：`100.104.8.0/24`
   - 美国（弗吉尼亚）：`100.104.133.64/26`
   - 德国（法兰克福）：`100.104.160.192/26`
   - 日本（东京）：`100.104.160.192/26`

4.8.7. 数据转发到消息服务

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

前提条件

- 已创建消息服务主题，并在该主题下创建推送类型为HTTP或队列的订阅。消息服务使用方法，请参见消息服务MNS文档。

  > **注意** 企业版实例中，消息服务主题所在地域必须与该企业版实例所在地域一致。

- 已创建数据转发规则和编写处理数据的SQL，请参见设置数据流转规则。

数据转发流程

- 设备发送数据到服务端。

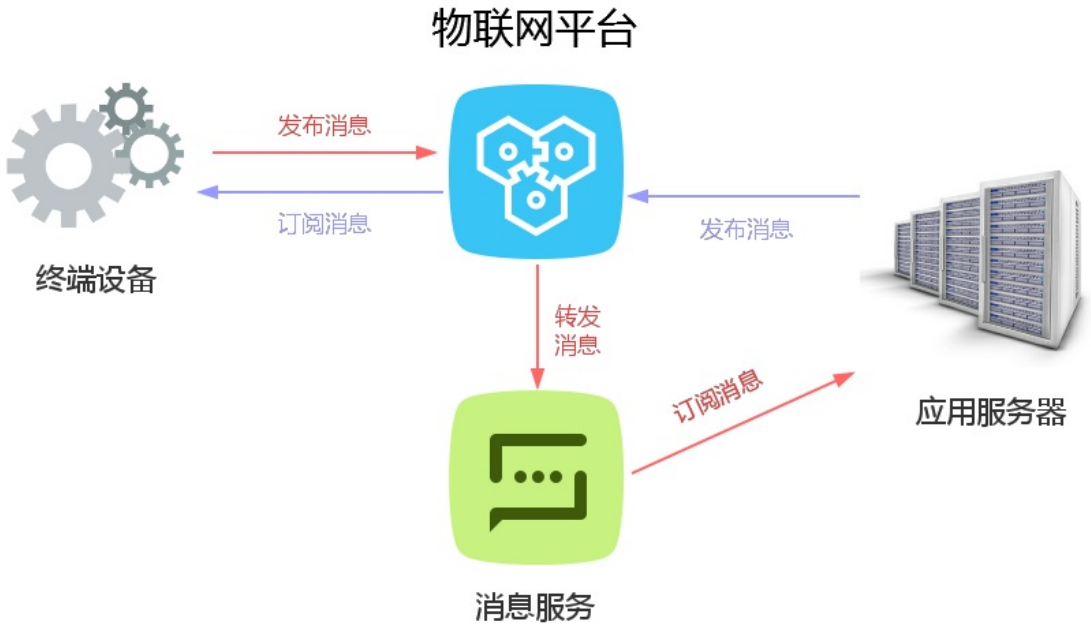
  设备发布消息到物联网平台中，物联网平台通过规则引擎，将消息进行处理并转发到消息服务的主题中。然后，您的应用服务器调用消息服务的接口订阅消息。

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

- 服务端发送数据到设备。

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

数据流转示意图如下。



操作步骤

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**发送数据到消息服务（MNS）中**。按照界面提示，设置其他信息，单击**确认**。

参数	描述
选择操作	选择**发送数据到消息服务（MNS）中**。
地域	选择消息服务所在地域。
主题	选择接收数据的消息服务主题。 消息服务会将接收到的消息发送给该主题下的HTTP和队列类型的订阅。目前，物联网平台推送至消息服务主题的消息，仅可以通过HTTP或队列两种方式订阅。 您可以单击**创建主题**，跳转到消息服务MNS控制台，创建消息服务主题以及订阅。请参见消息服务MNS文档。

参数	描述
角色	授权物联网平台将数据写入消息服务。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

4.8.8. 数据转发到时序数据库

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

前提条件

- 已在华东2（上海）地域创建专有网络下的TSDB实例。详细内容，请参见时序数据库（TSDB）文档。

  > **注意** 时序数据库（TSDB）实例于2022年04月06日停止新购，建议您将数据流转到时序数据库（Lindorm）。使用方法，请参见数据转发到时序数据库（Lindorm）。
  >
  > 对于2022年04月06日（不包含）前已经购买的TSDB实例支持正常续费和升配。

- 已创建数据转发规则和编写处理数据的SQL。具体操作，请参见设置数据流转规则。

限制说明

- 仅支持同地域转发。例如：华东2（上海）的物联网平台实例数据只能转发到华东2（上海）的TSDB实例中。
- 仅支持转发到专有网络（VPC）下的TSDB实例。
- 仅支持转发JSON格式数据。
- 转发的消息中，除了配置为timestamp和tag值的字段外，其他字段都将作为metric写入时序数据库。metric的数据类型支持数值型、字符串，其他类型会导致写入数据库失败。

操作步骤

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**存储到时序数据库（TSDB）中**。按照界面提示，设置其他信息，单击**确认**。

参数	描述
选择操作	选择**存储到时序数据库（TSDB）中**。
地域	固定为您的物联网平台实例所在地域：华东2（上海）。
TSDB实例	选择数据转发目标为您已创建的专有网络（VPC）下的TSDB实例。
metric数据类型	选择metric的数据类型。支持**数值型**和**字符串**。 更多信息，可单击帮助按钮查看。

参数	描述
timestamp	时间戳。支持： ○ 使用转义符 `${}` 表达式，例如 `${time}` ，表示取值为数据源Topic消息中time字段对应的值。 ○ 使用数据流转函数 `timestamp()` ，表示取值为数据流转服务器的时间戳。 ○ 输入值，必须为Unix时间戳，例如1404955893000。
tag名称	设置标记数据的标签名。支持中文汉字、英文字母、数字和特殊字符，包括：半角冒号（:）、逗号（,）、英文句号（.）、单引号（'）、正斜线（/）、短划线（-）、下划线（_）、圆括号（()）、方括号（[]）。
tag值	设置标签值。支持： ○ 使用转义符 `${}` 表达式。例如，数据源Topic的消息结构中，包含一个位置属性，标识符为city，则可以指定标签值为 `${city}` ，表示消息中city字段对应的值。建议使用此方式。 ○ 使用数据流转函数规定的一些函数，例如 `deviceName()` ，表示标签值为设备名称。支持的函数，请参见函数列表。 ○ 输入常量，例如beijing。支持输入中文汉字、英文字母、数字和特殊字符，包括：半角冒号（:）、逗号（,）、英文句号（.）、单引号（'）、正斜线（/）、短划线（-）、下划线（_）、圆括号（()）、方括号（[]）。 **说明** ○ 最多可添加8个tag名称、tag值。 ○ 需保证TSDB能够获取到配置的tag名称和值，如果获取不到任意一个tag的名称和值，会导致写入数据库失败。
角色	授权物联网平台将数据写入TSDB。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

数据流转示例

示例规则的SQL:

```
SELECT time,city,power,distance FROM "/alprodu****/myDevice/user/update";
```

规则引擎根据SQL处理数据和写入数据到TSDB如下。

1. 根据该SQL，规则引擎从Topic `/alprodu****/myDevice/user/update` 的消息中，筛选出time、city、power、和distance字段内容，作为转发的消息内容。

   通过以上SQL处理后的转发消息内容示例如下：

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

2. 根据已配置的数据流转操作，规则引擎向TSDB中写入的两条数据。

   示例中写入TSDB的数据如下：

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

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

   写入TSDB的数据说明：

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

操作样例

通过TSDB实现楼宇环境监测

4.8.9. 数据转发到函数计算

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

前提条件

- 已创建函数计算的服务和函数，并完成函数配置，验证函数能正常执行。函数计算使用方法，请参见函数计算文档。

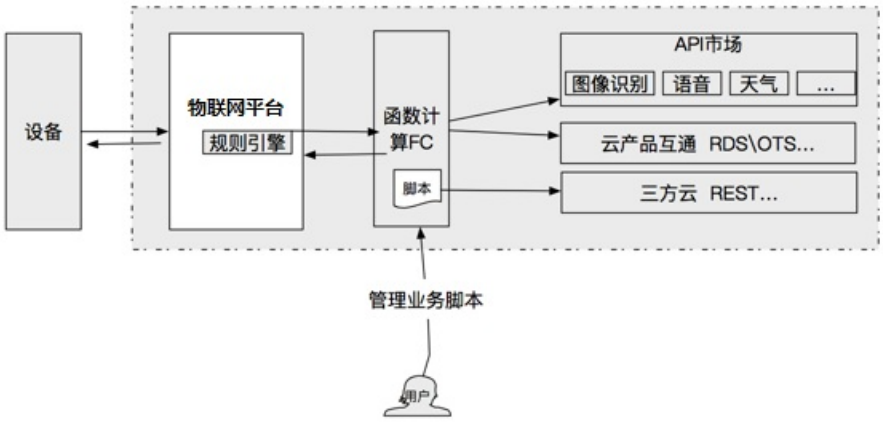
  > **注意** 企业版实例中，函数计算的服务和函数所在地域，必须与企业版实例所在地域一致。

- 已创建数据转发规则并编写处理数据的SQL，请参见设置数据流转规则。

背景信息

使用规则引擎数据转发功能，将设备中的数据转发到函数计算，函数计算执行函数的业务脚本，最终实现丰富的业务功能。

数据流转示意图如下。



操作步骤

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

> 注意 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**发送数据到函数计算（FC）中**。按照界面提示，设置其他信息，单击**确认**。

参数	说明
选择操作	选择**发送数据到函数计算（FC）中**。
地域	选择函数计算所在地域。
服务	选择函数计算服务。 您可单击**创建服务**，跳转到函数计算控制台创建服务。服务的详细说明，请参见管理服务。

参数	说明
函数版本选择方式	可选： ○ **使用默认版本**：使用函数计算的默认版本LATEST。 ○ **选择版本**：选择您为函数计算服务发布的版本。 您可单击**创建版本**，跳转到函数控制台创建版本。版本的详细内容，请参见管理版本。 ○ **选择别名**：选择您为函数计算服务版本设置的别名。 您可单击**创建别名**，跳转到函数控制台创建别名。别名的详细内容，请参见管理别名。
函数	选择接收数据的函数。 您可以单击**创建函数**，跳转到函数计算控制台创建函数。函数的详细说明，请参见管理函数。
授权	授权物联网平台将数据写入函数计算。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。
8. 测试。
   i. 根据规则SQL中定义的Topic类型，向目标Topic发送消息。Topic使用说明，请参见什么是Topic。设备上下行消息调试，请参见设备模拟器。
   ii. 登录函数计算控制台，在函数详情页面的**调用日志**页签查看函数执行记录；单击右上方**监控大盘**，查看函数的监控统计。

   > **注意** 监控数据统计会有5分钟的延时。

示例

推送设备上报数据到钉钉群

5.云产品流转（新版）

5.1. 概述

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

背景信息

与云产品流转的旧版相比，新版提供了解析器功能，可加工处理更复杂消息数据和云产品交互。例如在一个解析器中，可以配置多个Topic作为数据源，流转数据到其他Topic或云产品中。

云产品流转介绍，请参见什么是云产品流转。

功能说明

功能	描述
创建数据源	添加待转发的消息Topic。一个数据源支持添加多个Topic。
创建数据目的	添加数据转发的目的Topic或云产品。
配置解析器	创建解析器，关联已添加的数据源和数据目的，配置解析器脚本，实现数据流转。

使用限制

限制项	描述	限制
解析器	一个实例最多包含解析器总数。	1,000
数据源	一个解析器最多关联数据源总数。	1
	一个数据源最多包含Topic总数。	1,000
数据目的	一个解析器最多关联数据目的总数。	10
	一个数据目的最多包含的操作总数。	1
	一个解析器最多关联异常数据目的总数。	1
解析脚本	一个解析器的脚本内容大小限制。	120 KB
	一个解析器的脚本中循环执行流转函数的最大次数。 流转函数详细信息，请参见流转数据到数据目的函数。	100

使用流程

1. 添加待流转的数据源。
2. 添加转发到的数据目的。
3. 配置并启动解析器。

相关文档

- 脚本语法：解析器脚本的编辑和语法说明。
- 函数列表：解析器脚本中支持的函数列表。
- 数据格式：基础通信Topic、物模型通信Topic消息经物模型解析后的数据格式。解析器脚本中字段需按照解析后的数据格式编写。

5.2. 设置数据流转解析器

5.2.1. 添加待流转的数据源

通过规则引擎的云产品流转功能，物联网平台可将指定Topic的数据，流转至其他Topic和其他阿里云产品中。本文介绍添加消息Topic作为数据源的具体操作。

背景信息

配置数据流转解析器时，需关联已添加的消息Topic，作为待流转的数据。每个数据源中最多添加1,000个消息Topic。

解析器脚本中可使用获取消息上下文的函数 `topic(n)` ，获取消息来源的Topic。更多信息，请参见数据流转支持的函数。

创建数据源

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据源**页签，然后单击**创建数据源**。
6. 输入数据源名称和描述，单击**确定**。

   数据源名称支持中文、英文字母、日文、数字、下划线（_）和短划线（-），长度为1~30个字符，一个中文及日文占2个字符。

   根据页面提示，可直接进入**数据源详情**页面。您可单击右上角**编辑**，修改数据源名称和描述。

7. 在**数据源详情**页面，单击**添加Topic**，在弹出对话框中，根据页面提示，选择需要处理的消息Topic，然后单击**确定**。

   支持的Topic如下：

> **注意** 对于基础型（MQTT）实例下产品和设备，及尊享型实例下MQTT云网关产品和设备，仅支持选择**自定义**、**设备状态变化通知**和**设备生命周期变更**，若选择**自定义**需手动输入完整具体的Topic。更多说明，请参见消息通信说明。

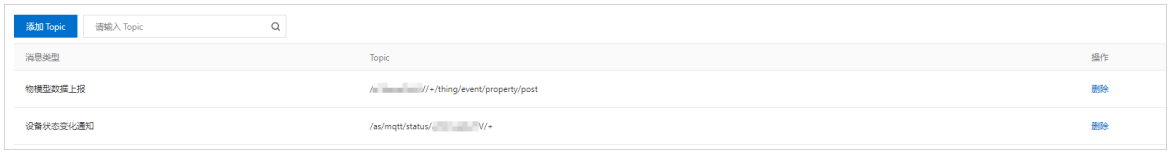
Topic说明

Topic	说明	相关文档
自定义	流转自定义数据格式消息的Topic，与自定义Topic的格式相同。格式为：`/${productKey}/${deviceName}/user/${TopicShortName}`。 其中`${TopicShortName}`为自定义的Topic类，即自定义Topic的后缀。 支持使用通配符（+）和（#）： ○ 全部设备（`+`）：指定产品下所有设备。 ○ `/user/#`：指定设备的所有自定义Topic。	自定义Topic
设备状态变化通知	流转设备上下线状态变更消息的Topic：`/as/mqtt/status/${productKey}/${deviceName}`。	设备上下线状态
物模型数据上报	包含： ○ 流转设备上报属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/post`。 ○ 流转设备上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`。 ○ 流转设备批量上报属性数据的Topic：`/${productKey}/${deviceName}/thing/property/batch/post`。 ○ 流转设备批量上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/batch/post`。 ○ 流转设备响应云端命令返回消息的Topic：`/${productKey}/${deviceName}/thing/downlink/reply/message`。	○ 设备上报属性 ○ 设备上报事件 ○ 设备属性批量上报 ○ 设备事件批量上报 ○ 设备下行指令结果
	对应设备Topic如下： ○ 设备上报属性的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/post`。 ○ 设备上报事件的Topic：`/sys/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`、`/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:{tsl.event.identifier}/post`。 ○ 设备批量上报属性、事件数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/batch/post`。	○ 设备上报属性 ○ 设备上报事件 ○ 设备批量上报属性、事件

Topic	说明	相关文档
设备生命周期变更	流转设备创建、删除、禁用、启用等消息的Topic：`/${productKey}/${deviceName}/thing/lifecycle`。	设备生命周期变更
网关发现子设备上报	网关设备特有的Topic：`/${productKey}/${deviceName}/thing/list/found`，将发现的子设备信息上报给物联网平台，然后进行流转。	网关发现子设备
设备拓扑关系变更	网关设备特有Topic：`/${productKey}/${deviceName}/thing/topo/lifecycle`，流转子设备和网关之间的拓扑关系建立和解除消息的Topic。	设备拓扑关系变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/topo/change`。	通知网关拓扑关系变化
设备标签变更	流转设备标签信息变更的Topic：`/${productKey}/${deviceName}/thing/deviceinfo/update`。	设备标签变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/update`。	标签信息上报
物模型历史数据上报	包含： ○ 流转设备上报历史属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/history/post`。 ○ 流转设备上报历史事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/history/post`。	○ 历史属性上报 ○ 历史事件上报
	对应设备上报物模型历史数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/history/post`。	物模型历史数据上报
OTA升级设备状态通知	包含： ○ 流转设备上报OTA升级结果的Topic：`/${productKey}/${deviceName}/ota/upgrade`。 ○ 流转设备上报OTA升级进度的Topic：`/${productKey}/${deviceName}/ota/progress/post`。	○ OTA升级状态通知 ○ OTA升级进度通知
	对应设备上报升级进度的Topic：`/ota/device/progress/${productKey}/${deviceName}`。	设备上报升级进度
OTA模块版本号上报	流转设备上报OTA模块版本号变更的Topic：`/${productKey}/${deviceName}/ota/version/post`。	OTA模块版本号变更通知
	对应设备上报OTA模块版本的Topic：`/ota/device/inform/${productKey}/${deviceName}`。	设备上报OTA模块版本

Topic	说明	相关文档
OTA升级批次状态通知	物联网平台通知OTA升级批次状态变化的Topic：`/${productKey}/${packageId}/${jobId}/ota/job/status`。	OTA升级批次状态通知
任务事件	包含： ○ 流转设备任务状态通知的Topic：`/sys/uid/${uid}/job/${jobId}/lifecycle`。 ○ 流转实例迁移任务状态通知的Topic：`/sys/uid/${uid}/distribution/${jobId}/lifecycle`。 **说明** 迁移产品的名称为实例迁移的任务名称。	○ 设备任务的状态通知 ○ 实例迁移任务的状态通知
孪生节点属性变更	流转数字孪生节点属性数据的Topic：`/sys/uid/${uid}/digitaltwin/${dtInstanceId}/${nodeId}/property/update`。	数字孪生节点属性变更

8. （可选）重复上一步操作，可在当前数据源下，添加多个消息Topic。
添加Topic后，若有业务需求，您也可在**数据源详情**页面，删除已添加的消息Topic。



9. （可选）重复步骤5~8，添加多个数据源。

后续步骤

添加转发到的数据目的

5.2.2. 添加转发到的数据目的

通过规则引擎的云产品流转功能，物联网平台可将指定Topic的数据，流转至其他Topic和其他阿里云产品中。本文介绍添加数据目的对应的操作步骤。

背景信息

配置数据流转解析器时，需关联已添加的数据目的，作为数据正常或异常流转的目的。每个数据目的中最多添加1个数据转发操作。

解析器脚本中可使用数据流转函数，实现数据流转到其他Topic或云产品的目的。更多信息，请参见流转数据到数据目的的函数。

创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，配置以下参数，单击**确定**。

参数	描述
数据目的名称	自定义名称。支持中文、英文字母、日文、数字、下划线（_）和短划线（-），长度为1~30个字符，一个中文及日文占2个字符。
数据目的描述	数据目的描述信息。
选择操作	设置数据转发目的地。数据转发配置的具体示例，请参见数据流转使用示例目录下的具体文档。

7. （可选）重复步骤5~6，添加多个数据目的。

后续步骤

配置解析器

5.2.3. 配置解析器

通过规则引擎的云产品流转功能，物联网平台可将指定Topic的数据，流转至其他Topic和其他阿里云产品中。本文介绍设置数据流转解析器的完整操作步骤，依次是创建解析器、关联数据源和数据目的、配置流转数据的解析脚本。

前提条件

已添加数据源和数据目的，请参见：

- 添加待流转的数据源。
- 添加转发到的数据目的。

创建解析器

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

   > **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 在**解析器**页签，单击**创建解析器**。

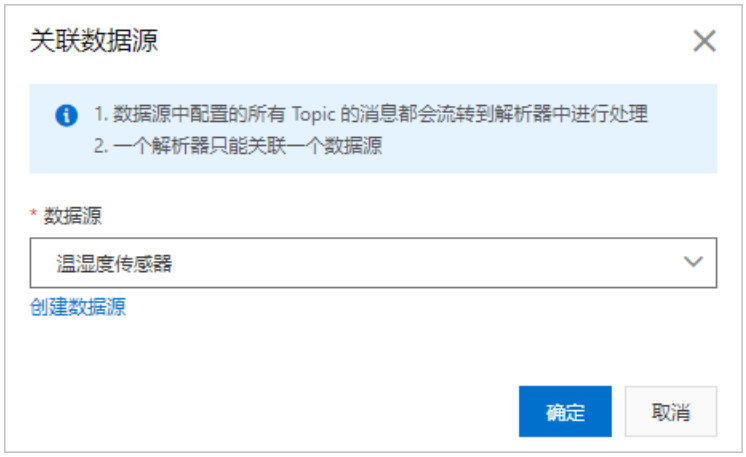
6. 配置**解析器名称**和**解析器描述**内容，单击**确定**。

   解析器名称支持中文、英文字母、日文、数字、下划线（_）和短划线（-），长度为1~30个字符，一个中文及日文占2个字符。

   根据页面提示，可直接进入**解析器详情**页面。您可单击右上角**编辑**，修改解析器名称和描述。

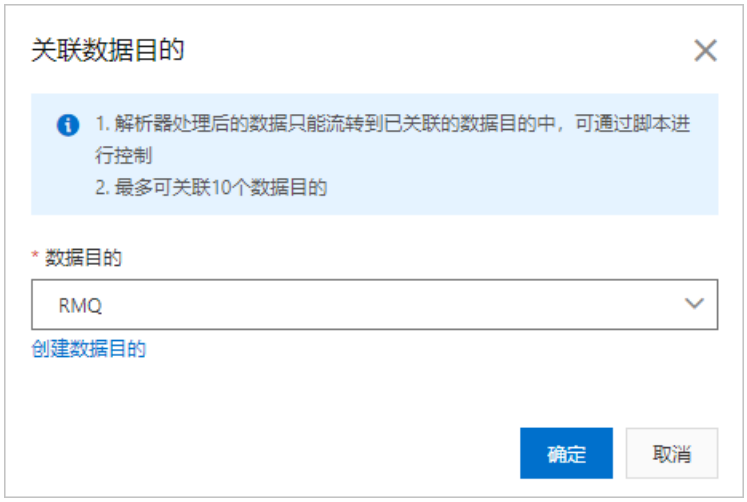
关联数据源和数据目的

1. 在**解析器详情**页面，单击**关联数据源**。
2. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源，单击**确定**。



3. 在**解析器详情**页面，单击**数据目的**。

4. 单击**关联数据目的**，在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的，单击**确定**。



5. 单击**异常数据目的**下的**关联数据目的**，将重试失败的错误消息转发至指定位置。

> 注意
>
> - 最多支持添加一个错误操作。
> - 正常操作和错误操作的转发目的地不能是相同的云产品。例如，不能同时转发到表格存储。
> - 错误消息转发失败后，不会再进行重试。
> - 这里的错误消息仅针对因其他云产品实例问题导致的规则引擎转发失败错误。

消息转发至云产品失败后，会进行重试。若重试失败，将根据错误操作数据转发的设置转发错误消息。

错误消息格式：

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

错误消息参数说明如下：

参数	说明
ruleName	规则名称。
topic	消息来源Topic。
productKey	产品ProductKey。
deviceName	设备名称。
messageId	云端消息ID。
base64OriginalPayload	Base64编码后的原始数据。
failures	错误详情。可能会有多个。

参数	说明
actionType	出错操作的类型。
actionRegion	出错操作的地域。
actionResource	出错操作的目的资源。
errorMessage	错误信息。

配置并启动解析器

1. 在**解析器详情**页面，单击**解析器**。
2. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。

   数据转发函数和脚本的使用示例，请参见数据流转使用示例目录下的具体文档。
3. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   - **Topic**：输入的Topic，其数据格式与脚本解析逻辑相符即可。
   - **Payload数据**：输入数据的格式需符合规则引擎的数据格式。其中：
     - 自定义Topic数据的格式，是设备上报的原始数据格式。
     - 基础通信Topic和物模型通信Topic数据的格式说明，请参见数据格式。

   运行结果中会显示脚本中声明的变量值和执行的数据流转函数。

   调试成功后，会直接将调试数据写入对应云产品。您可登录对应云产品控制台，查看流转的数据。
4. 单击**发布**。
5. 所有设置完成后，返回至**云产品流转**页面的**解析器**页签，单击解析器对应的**启动**。解析器启动后，数据即可按照解析脚本进行转发。

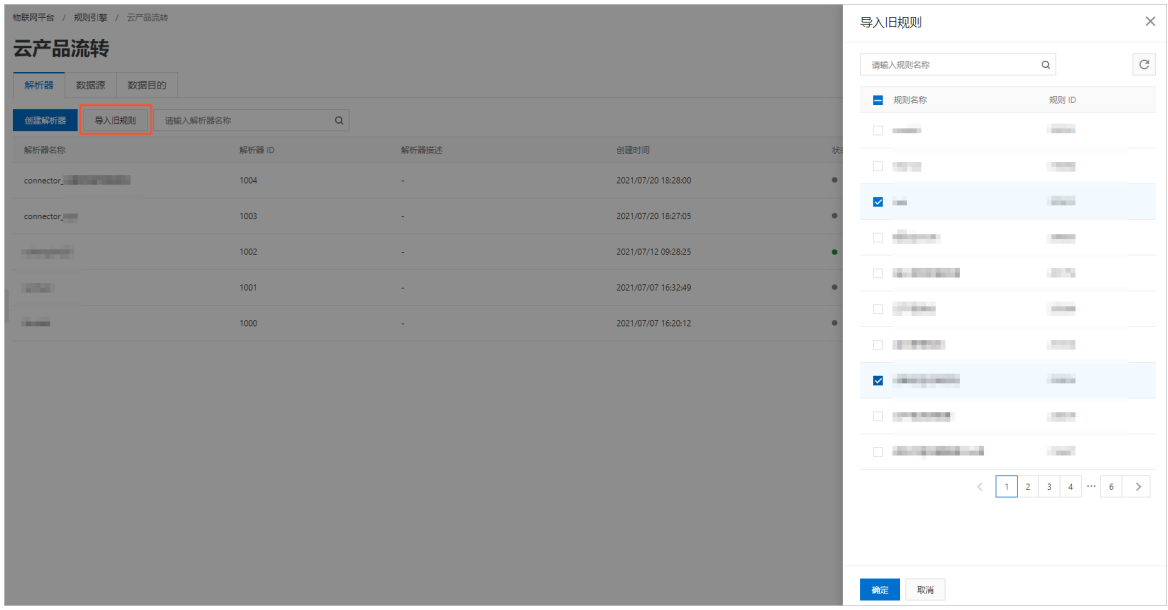
   您也可以单击：
   - **查看**：在**解析器详情**页面，修改解析器的具体设置。
   - **删除**：删除对应解析器。

     > **说明** 运行中的解析器不可删除。
   - **停止**：停止对应解析器转发数据。

导入旧规则

您可根据业务需求，直接导入旧版中已配置的流转规则，再重新配置数据源、数据目的和解析脚本。导入旧规则步骤如下：

1. 在**解析器**页签，单击**导入旧规则**。



2. 在右侧面板的规则列表中，选中需要导入的规则，单击**确定**。
   旧规则导入后，会显示在解析器列表中，且以 `"connector_"+${旧规则名称}` 重新命名。

   您可根据业务需求，在**解析器详情**页面，修改解析器名称，重新配置数据源、数据目的和解析脚本。

5.3. 脚本语法

物联网平台提供的脚本解析器，类似于JavaScript语言，与SQL表达式相比，可加工处理复杂消息数据和云产品交互。脚本语言的解析能力包括获取消息内容、转换数据格式、处理字符串、组装JSON格式数据、处理二进制数据和流转数据等。本文介绍如何编写解析脚本。

背景信息

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

脚本示例

本文以上报的属性数据为例，输入数据如下：

```
{
    "deviceType": "CustomCategory",
    "iotId": "JCp9***",
    "requestId": "1626948228247",
    "checkFailedData": {
    },
    "productKey": "a1o***",
    "gmtCreate": 1626948134445,
    "deviceName": "Device1",
    "items": {
        "Temperature": {
            "value": 38,
            "time": 1626948134319
        },
        "Humidity": {
            "value": 25,
            "time": 1626948134319
        }
    }
}
```

解析和处理数据的示例如下：

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//筛选出上报的温湿度值。
var h = getOrNull(data, "items", "Humidity", "value");
var t = data.items.Temperature.value;
// 设置温度值大于38时触发规则，转发数据到云数据库RDS。
// RDS表结构为id[自增主键]、deviceName、temperature、humidity、time，在writeRds方法中，可以按column:value的形式，将值写入对应的列。
if (t > 38) {
    writeRds(1000, {"devcieName":deviceName(), "temperature":t, "time":timestamp(), "humidity":h});
}
```

解析处理的数据源必须转换为JSON格式数据，即数组或者嵌套的JSON数据。

脚本文件中支持使用JSONPath和函数 `getOrNull()` 获取其中的字段值。详细的使用说明，请参见LanguageManual UDF和getOrNull()。

例如以上示例中，可使用 `getOrNull(data, "items", "Humidity", "value");` ，获取到值 `25` ；使用 `data.items.Temperature.value` 获取到值 `38` ；使用 `data.iotId` ，获取到值 `JCp9***` 。

> **注意** 在脚本文件中获取指定字段值时，若指定字段不存在，则：
> - 使用函数方法，会返回值 `null` ，脚本可继续向下执行。
> - 使用JSONPath方法，脚本会出现空指针，中断执行。

标识符

代码中常量、变量和其他自定义字段，需使用标识符定义。标识符支持大小写英文字母、数字和下划线（_），不能以数字开头。

以下关键词和保留字不能作为标识符使用。

- 关键词：`for`、`break`、`continue`、`if`、`else`、`true`、`false`、`var`、`new`、`null`和`return`。
- 保留字：`breakdo`、`instanceof`、`typeof`、`case`、`catch`、`finally`、`void`、`switch`、`while`、`debugger`、`function`、`this`、`with`、`default`、`throw`、`delete`、`in`、`try`、`as`、`from`、`classenum`、`extends`、`super`、`const`、`export`、`import`、`await`、`implementslet`、`let`、`private`、`public`、`interface`、`package`、`protected`、`static`、`yield`。

数据类型

代码中常量、变量和其他自定义字段支持数据类型有：Number、Boolean、String、Byte、Map、Array。

常量可取值为null，数值型常量的取值类型包括十进制整型、十六进制整型和浮点型。

流程控制语句

物联网平台支持使用`for`循环和`if...else`条件语句。其中`for`循环，支持使用关键词`break`（跳出循环）和`continue`（跳出本次循环）。

> **注意** 若使用`for`语句循环执行流转函数，循环次数不能超过100。流转函数详细信息，请参见流转数据到数据目的函数。

操作符

- 逻辑运算：`&&`、`||`。

  逻辑条件结果为非布尔型时，null（空值）表示false，否则为true。例如`null && "x"`返回false，`null || "x"`返回true。
- 数学运算：`*`、`/`、`%`、`+`、`-`。

  操作数据必须为数值型，否则会抛出异常。
- 条件判断：`>`、`=>`、`<`、`<=`、`==`（仅支持值比较）、`!=`。

注释

脚本中支持多行注释（`/* ${comments}*/`）和单行注释（`// ${comments}`）。

相关文档

- 脚本中支持使用的函数说明，请参见函数列表。
- 配置解析脚本进行数据流转的示例，请参见数据流转使用示例目录下的具体文档。

5.4. 函数列表

您可以在编写解析器脚本时使用以下函数，实现多样化数据处理。函数列表包括数据类型转换函数、流转数据到数据目的函数和基础数据类型的函数等。本文介绍函数的表达式及其使用说明。

除本文支持的函数表达式外，云产品流转新版的解析器脚本中还支持旧版的函数。详细内容，请参见数据流转支持的函数。

数据类型转换函数

表达式	说明
toBoolean(Object)	Object值转换为Boolean值。参数支持以下类型的取值： • Boolean：返回对应的布尔值。 • Number：参数为0，返回false。否则，返回true。 • String：参数为 `"true"` ，返回true。否则，返回false。 • 参数值为null：返回false。
toNumber(Object)	Object值转换为Number值。参数支持以下类型的取值： • Boolean：参数为true，返回1，参数为false，返回0。 • Number：返回对应的数值。 • String：按照数值类型解析。 • 参数值为null：返回0。
toString(Object)	Object值转换为String值。 参数为任意值，返回对应String值。参数值为null，返回空字符。 当参数为二进制类型时，按照UTF-8编码返回对应值。
toMap(Object)	Object值转换为Map值。参数支持以下类型的取值： • Map：返回对应的值。 • String：按照JSON格式解析为Map。 • 参数值为null：返回空Map。
toArray(Object)	Object值转换为Array值。参数支持以下类型的取值： • Array：返回对应的值。 • String：按照JSON格式解析为Array。 • 参数值为null：返回空Array。
toBinary(Object)	Object值转换为二进制值。参数支持以下类型的取值： • Binary：返回对应的值。 • String：按照UTF-8编码返回值。 • 参数值为null：返回空Binary。

时间类型转换函数

函数表达式	说明

函数表达式	说明
format_date(timestamp, patten, timeZone)	将时间戳毫秒值转换为指定格式的时间。返回String类型的时间。 • timestamp：时间戳毫秒值。 • patten：待转换的时间格式。例如 `yyyy-MM-dd HH:mm:ss` 。 • timeZone：时区。例如GMT、UTC、CST等。可为空，默认为当前系统所在时区。
to_timestamp(dateString, patten, timeZone)	将指定格式的时间转换为时间戳毫秒值。返回数值类型的时间戳。 • dateString：时间字符串。 • patten：指定的时间格式。例如 `yyyy-MM-dd HH:mm:ss` 。 • timeZone：时区。例如GMT、UTC、CST等。可为空，默认为当前系统所在时区。

流转数据到数据目的函数

以下函数表达式中的入参destinationId是数据目的ID，payload是消息内容。

函数表达式	说明	示例
writeAmqp(destinationId, payload, tag)	流转数据到AMQP消费组。 tag：可选参数。设置 `tag` 后，所有通过该操作流转到AMQP服务端订阅消费组里的消息都会携带该 `tag` 。 tag长度不超过128个字符，可以输入常量或变量。 • 常量支持输入中文汉字、英文字母、数字。 • 变量代表解析脚本处理后的JSON数据中key对应的value值。如果取不到value值，则消息不携带tag。	数据转发到AMQP服务端订阅消费组
writeDatahub(destinationId, data)	流转数据到DataHub。 data：写入DataHub的数据。仅支持Map和Binary类型数据。	数据转发到DataHub
writeFc(destinationId, data)	流转数据到函数计算（FC）。 data：流转到FC的数据。	数据转发到函数计算
writeKafka(destinationId, payload, key)	流转数据到消息对列Kafka。 key：可选参数，流转到Kafka Topic消息中携带的key。 您可以在Kafka消息消费端，通过标签进行消息路由。	数据转发到消息队列Kafka
writeMns(destinationId, payload)	流转数据到消息服务（MNS）。	数据转发到消息服务

函数表达式	说明	示例
writeMq(destinationId, payload, tag)	流转数据到消息队列（RocketMQ）。 tag：可选参数。设置标签后，所有通过该操作流转到RocketMQ对应Topic里的消息都会携带该标签。您可以在RocketMQ消息消费端，通过标签进行消息过滤。 tag长度不超过128个字符，可以输入常量或变量。 • 常量支持输入中文汉字、英文字母、数字。 • 变量代表解析脚本处理后的JSON数据中key对应的value值。如果取不到value值，则消息不携带tag。	数据转发到消息队列RocketMQ
writeTableStore(destinationId, data，flowType)	流转数据到表格存储（Tablestore）。 • data：写入表格存储的数据。仅支持Map类型数据，其中键（Key）值对应表格列名，值（Value）对应列值。 data中必须包含Tablestore的主键。 • flowType：可选参数。设置非主键字段的数据类型。 ◦ flowType为true时，非主键字段都按照String数据类型流转到Tablestore中。 ◦ flowType为false或不传入时，非主键字段按照对应值的数据类型流转到Tablestore中。	数据转发到表格存储
writeRds(destinationId, data)	流转数据到云数据库RDS。 data：写入云数据库表的数据。仅支持Map类型数据，其中键（Key）值对应数据库表列名，值（Value）对应列值。	数据转发到云数据库RDS
writeTsdb(destinationId, timestamp, metricName, value, tag)	流转数据到时序数据库（TSDB）或实例内的时序数据库中。 • timestamp：时间戳。 • metricName：TSDB中存储的指标名称。 • value：TSDB中存储的数据点值。支持String和Map类型数据。 • tag：设置标记数据的标签键值对，Map类型数据。	数据转发到实例内的时序数据存储、数据转发到实例外的时序数据库

函数表达式	说明	示例

writeIotTopic(destinationId, topic, payload, deviceName)	流转数据到另一个Topic。
	• Topic：数据转发目的地Topic，支持以下Topic。
	  ○ 自定义Topic：该自定义Topic的设备操作权限需为订阅，即所属设备可订阅这个Topic，获取转发的消息。
	  ○ 物模型数据下发Topic：`/sys/${productKey}/${deviceName}/thing/service/property/set`。该Topic为设备接收设置属性值指令的Topic，设备从该Topic接收转发数据，并根据数据内容，设置属性值。用于目的地Topic所属设备根据转发的数据更改属性值的场景。
	Topic中`${productKey}`必须与destinationId对应数据目的中设置的产品相同。函数中Topic必须指定具体Topic，不支持使用通配符。
	更多信息，请参见创建数据目的。
	注意 目的地Topic所属设备若未订阅该Topic，则收不到转发的消息。
	例如使用MQTT.fx工具接入物联网平台的设备A，不会自动订阅设备Topic，若设备B向设备A的Topic：`thing/service/property/set`转发数据，而设备A未手动订阅该Topic，则设备A收不到设备B转发的消息。
	设备Topic自动订阅的详细说明，请参见自动订阅Topic说明。
	• deviceName：对于基础型（MQTT）实例下产品和设备，及尊享型实例下MQTT云网关产品和设备，流转数据到另一个Topic时，需传入该参数。其他业务场景无需传入该参数。
	数据格式为Map类型，Key为`"deviceName"`，Value为`deviceName()`，表示发送该Topic消息的设备名称。
	更多说明，请参见消息通信说明。
	注意 对于基础型（MQTT）实例下产品和设备，及尊享型实例下MQTT云网关产品和设备，流转的数据目的Topic仅支持自定义Topic，且不支持跨产品流转，即仅支持流转到当前产品下的自定义Topic。
		数据转发到另一Topic

函数表达式	说明	示例
writeLindorm(destinationId, timestamp, tags, fields)	流转数据到时序数据库（Lindorm）。 • timestamp：时间戳，表示该数据记录对应的生成时间。 • tags：标签，表示指标项针对的具体对象属性。一个标签由一个标签键（Key）和一个标签值（Value）组成。Map类型数据。 • fields：数据记录，一条数据记录可以有多个字段值，表示指标项的不同对象。支持Object数据类型。	数据转发到时序数据库（Lindorm）

基本数据类型支持的函数

- Map类型。

表达式	说明
[Object]	获取指定键（ `Key` ）对应的 `Value` 。
size()	获取Map数据中键值对数量。
containKey(String)	判断Map数据中是否包含指定的键。
keySet()	获取Map数据中键的集合，返回类型为Array。
remove(Object)	移除Map数据中指定键对应的键值对。
put(Object, Object)	在Map数据中添加键值对。
putAll(map)	在Map数据中批量添加一组Map数据。

- Array类型。

函数	说明
[int]	获取指定索引位置的值。数组首位的索引值为0。
contains(Object)	判断数组中是否包含指定元素。
remove(Object)	移除数组中指定的元素。
removeAt(int)	移除数组中指定索引位置的元素。
add(Object)	在数组末尾中添加元素。
add(index, Object)	在对应index索引位置添加元素。
addAll(array)	在数组中添加另一个数组。
size()	获取数组中元素个数。

- String类型。

函数	说明
substring(start, end)	截取从 `start` 位置到 `end-1` 位置的字符串。 `end` 不传时，截取到字符串末尾。
length()	获取字符串长度。
split(String)	按照分割字符，分割字符串。
startsWith(String)	查看字符串是否以指定的字符串开头。
endsWith(String)	查看字符串是否以指定的子字符串结尾。
indexOf(String, index)	从指定index索引位置开始，指定字符串在字符串中首次出现的位置。 index不传入，默认为0。

其他函数

表达式	说明
endWith(input, suffix)	判断字符串input中是否以字符串suffix结尾。 返回结果类型为布尔型。
getDeviceTag(key)	返回 `key` 所对应的设备标签。如果设备没有该 `key` 对应的标签，则返回值为空。
getOrNull(data, "items", ……)	返回JSON格式数据data中指定字段的值。 该函数中传入的字段个数没有限制，但必须逐级传入。函数返回最后一个字段的值，若最后一个字段不存在或值为空，则函数返回值为null。 例如data中数据： `"items": { "Humidity": { "value": 25, "time": 1626948134319 } }` 函数示例如下： • `getOrNull(data, "items")` 返回字段items值 `"Humidity": {"value": 25,"time": 1626948134319}` 。 • `getOrNull(data, "items", "Humidity", "value")` 返回字段value值 `25` 。 • `getOrNull(data, "items", "Temperature")` ，字段Temperature不存在，返回值null。

表达式	说明
payload(textEncoding)	返回设备发布消息payload的转义数据。textEncoding表示payload的转义字符编码，取值如下： • 不传入参数：默认按照UTF-8编码转换为字符串，即 `payload()` 等价于 `payload('utf-8')` 。 • `'json'` ：将payload数据转换成Map格式变量。如果payload不是JSON格式，则返回异常。 • `'binary'` ：将payload数据转换成二进制变量进行透传。

相关文档

您可结合支持的脚本语法，调用函数将数据流转到数据目的地。脚本的使用说明，请参见脚本语法。

5.5. 数据流转使用示例

5.5.1. 数据转发到另一Topic

您可以设置将解析脚本处理完的数据，转发到另一个Topic中，实现M2M通信或者更多其他通信场景。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

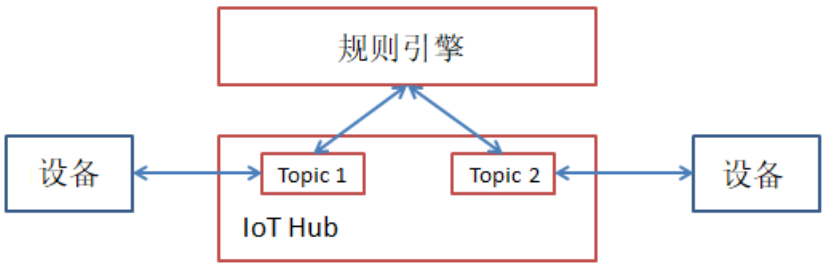
前提条件

已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。

背景信息

规则引擎数据转发功能将Topic 1中的数据转发到Topic 2内。

数据流转示意图如下。



创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

   本示例配置如下图所示。

参数	说明
选择操作	选择**发布到另一个Topic**。

参数	说明
产品	选择转发的目的地Topic所属产品。 具体Topic，需在解析脚本中通过函数 `writeIotTopic(destinationId, topic, payload)` 设置。函数说明，请参见数据流转函数。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeIotTopic(1000, "/sys/a1I***/room3/thing/service/property/set", data)
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。

5.5.2. 数据转发到AMQP服务端订阅消费组

您可以使用规则引擎将设备发送到物联网平台的消息，经过解析脚本处理后，再转发到AMQP服务端订阅消费组，并通过AMQP客户端消费消息。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建AMQP服务端订阅消费组，作为数据转发目的地。消费组的创建，请参见创建消费组。

创建数据目的

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

   > **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	描述
选择操作	选择**发布到AMQP服务端订阅消费组**。
消费组	选择一个已创建的消费组作为数据转发目标。单击**创建消费组**可以进行消费组创建。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeAmqp(1000, data, "调试");
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。

配置AMQP客户端消费消息

数据转发到AMQP消费组后，您的服务器需通过AMQP客户端消费消息。AMQP客户端开发说明，请参见AMQP客户端接入说明。

AMQP客户端开发示例，请参见：

- Java SDK接入示例
- .NET SDK接入示例

- Node.js SDK接入示例
- Python 2.7 SDK接入示例
- Python3 SDK接入示例
- PHP SDK接入示例
- Go SDK接入示例

5.5.3. 数据转发到消息队列RocketMQ

您可以使用规则引擎，将物联网平台数据转发到消息队列（RocketMQ）中存储，以实现消息从设备、物联网平台、RocketMQ到应用服务器之间的全链路高可靠传输能力。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建消息队列（RocketMQ）实例和用于接收数据的Topic。RocketMQ使用方法，请参见RocketMQ快速入门。

> **注意** 目前，RocketMQ实例所在地域必须与当前物联网平台实例所在地域保持一致。存量的跨地域实例下数据流转配置不受影响，仍可继续正常流转数据。

创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	说明
选择操作	选择**发送数据到消息队列（RocketMQ）中**。
地域	选择RocketMQ所在地域。
实例	选择RocketMQ实例。 您可以单击**创建实例**，跳转到消息队列控制台，创建RocketMQ实例，请参见消息队列文档。
Topic	选择用于接收物联网平台数据的RocketMQ Topic。 您可以单击**创建Topic**，跳转到消息队列控制台，创建RocketMQ Topic。
授权	授权物联网平台将数据写入RocketMQ。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。

5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。

   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeMq(1000, data, "调试");
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。

   参数示例如下：

   运行结果如下，表示脚本执行成功。

7. 单击**发布**。

8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。
9. 在RocketMQ控制台，查看是否成功接收到消息。

5.5.4. 数据转发到表格存储

您可以使用规则引擎数据流转功能，将数据转发到表格存储（Tablestore）中。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建表格存储实例和用于接收数据的数据表。表格存储使用方法，请参见表格存储文档。

> **注意** 企业版实例中，表格存储实例所在地域，必须与企业版实例所在地域一致。

创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎** > **云产品流转**。

4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

   > **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

   > **说明** 仅支持转发JSON格式数据。

参数	说明
选择操作	选择**存储到表格存储（Tablestore）**。
地域	选择表格存储所在地域。
实例	选择表格存储实例。 您可以单击**创建实例**，跳转到表格存储控制台，创建表格存储实例，请参见表格存储文档。
数据表	选择接收数据的表格存储数据表。 您可以单击**创建数据表**，跳转到表格存储控制台，创建表格存储数据表。
角色	授权物联网平台将数据写入表格存储。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//获取上报的属性值
var h = data.items.Humidity.value;
var t = data.items.Temperature.value;
// 表中添加主键deviceName、id，在writeTableStore方法中，可以按column:value的形式，将温湿度值写入对应的列
writeTableStore(1000, {"devcieName":deviceName(), "id":timestamp(), "temperature":t, "humidity":h});
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。
9. 数据推送成功后，在表格存储接收数据的数据表的**数据管理**页签，查看是否成功接收到指定数据。

5.5.5. 数据转发到DataHub

您可以使用规则引擎将数据转到DataHub上，再由DataHub将数据流转至实时计算、MaxCompute等服务中，以实现更多计算场景。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建DataHub Project和用于接收数据的Topic。DataHub使用方法，请参见DataHub文档。

> **注意** 企业版实例中，DataHub Project所在地域，必须与企业版实例所在地域一致。

创建数据目的

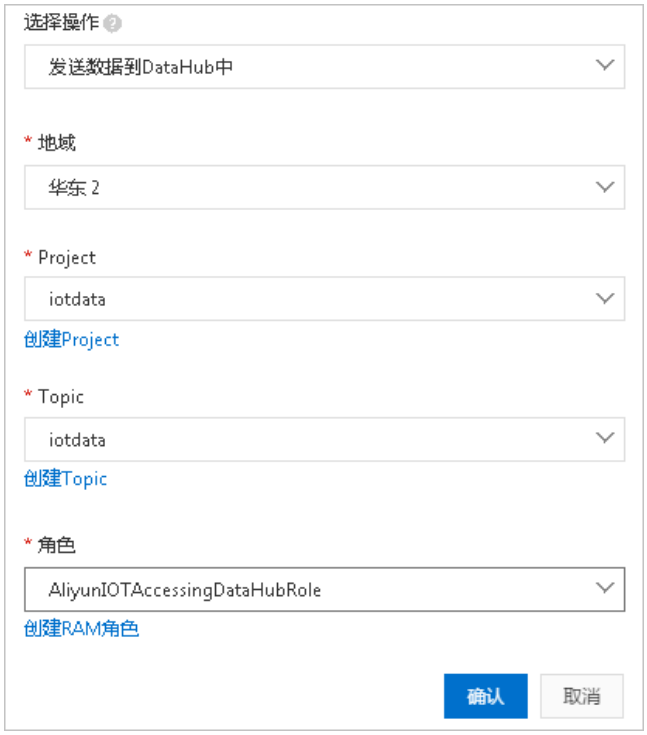
1. 登录物联网平台控制台。
2. 在**实例概览**页面，找到对应的实例，单击实例进入**实例详情**页面。

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。



参数	说明
选择操作	选择**发送数据到DataHub中**。
地域	选择DataHub所在地域。
Project	选择DataHub Project。 您可以单击**创建Project**，跳转到DataHub控制台，创建DataHub Project，请参见DataHub文档。
Topic	选择接收数据的DataHub Topic。 选择Topic后，规则引擎会自动获取Topic中的Schema，规则引擎筛选出来的数据将会映射到对应的Schema中。 **说明** ○ 将数据映射到Schema时，需使用 `${}` ，否则存入表中的将会是一个常量。 ○ Schema与规则引擎的数据类型必须保持一致，否则无法存储。 您可以单击**创建Topic**，跳转到DataHub控制台，创建DataHub Topic。

参数	说明
角色	授权物联网平台将数据写入DataHub。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeDatahub(1000, data)
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。

5.5.6. 数据转发到云数据库RDS

您可以设置规则引擎数据流转，将处理后的数据转发到云数据库RDS版（以下简称RDS）的实例中存储。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。

- 已在您的物联网平台实例所在地域，创建专有网络下的MySQL或SQL Server版RDS实例，并创建数据库和数据表。RDS使用方法，请参见云数据库RDS文档。

限制说明

- 仅支持同地域转发。例如：华东2（上海）的物联网平台实例数据只能转发到华东2（上海）的RDS数据表中。
- 仅支持转发到专有网络（VPC）下的RDS实例。
- 支持MySQL实例和SQL Server实例。
- 支持普通数据库和高权限数据库的转发。
- 仅支持转发JSON格式数据。

创建数据目的

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

   > **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	描述
选择操作	选择**存储到云数据库（RDS）中**。
地域	固定为您的物联网平台实例所在地域。
RDS实例	选择RDS实例。
数据库	输入数据库名。 **说明** 如果是高权限数据库，需要您手动输入数据库名称。
账号	输入RDS实例的用户账号。此账号应具有该数据库的读写权限，否则规则引擎无法将数据写入RDS。 **说明** 规则引擎获得账号后，仅将规则匹配的数据写进数据库中，不会做其他操作。
密码	输入登录RDS实例的密码。
表名	输入数据库中已建立的数据表名。规则引擎将把数据写入这张表上。
角色	授权物联网平台将数据写入RDS。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//筛选出上报的温湿度值。
var h = data.items.Humidity.value;
var t = data.items.Temperature.value;
// RDS表结构为id[自增主键] deviceName、temperature、humidity、time，在writeRds方法中，可以按column:value的形式，将值写入对应的列。
// 设置温度值大于38时触发规则。
if (t > 38) {
    writeRds(1000, {"deviceName":deviceName(), "temperature":t, "time":timestamp(), "humidity":h});
}
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。

   参数示例如下：

   运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。
9. 在RDS控制台的**数据安全性**页面，设置和查看白名单。配置完成后，规则引擎为了连接RDS，会在RDS的白名单中添加下列IP。若以下IP未出现，请手动添加。具体操作，请参见云数据库RDS文档。
   - 华东2：`100.104.53.192/26,100.104.148.64/26,100.104.6.192/26,100.104.143.128/26,100.104.76.0/24,100.104.73.128/26,100.104.200.64/26,100.104.40.64/26,100.104.3.0/26,100.104.29.128/26,100.104.121.0/26,100.104.84.64/26`
   - 亚太东南1（新加坡）：`100.104.106.0/24`
   - 美国（硅谷）：`100.104.8.0/24`
   - 美国（弗吉尼亚）：`100.104.133.64/26`
   - 德国（法兰克福）：`100.104.160.192/26`
   - 日本（东京）：`100.104.160.192/26`

5.5.7. 数据转发到消息服务

您可以使用规则引擎数据流转功能，将设备数据转发到消息服务主题中，服务端再从消息服务主题中订阅消息，实现设备端与服务端之间高性能的消息闭环传输。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建消息服务主题，并在该主题下创建推送类型为HTTP或队列的订阅。消息服务使用方法，请参见消息

服务MNS文档。

> **注意** 企业版实例中，消息服务主题所在地域必须与该企业版实例所在地域一致。

背景信息

设备端与服务端通过消息服务转发数据的流程，请参见数据转发流程。

创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	描述
选择操作	选择**发送数据到消息服务（MNS）中**。
地域	选择消息服务所在地域。
主题	选择接收数据的消息服务主题。 消息服务会将接收到的消息发送给该主题下的HTTP和队列类型的订阅。目前，物联网平台推送至消息服务主题的消息，仅可以通过HTTP或队列两种方式订阅。 您可以单击**创建主题**，跳转到消息服务MNS控制台，创建消息服务主题以及订阅。请参见消息服务MNS文档。
角色	授权物联网平台将数据写入消息服务。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeMns(1000, data)
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。

5.5.8. 数据转发到实例外的时序数据库

您可以配置数据流转规则，将处理过的数据转发到时序数据库（TSDB）的中。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。

- 已在华东2（上海）地域创建专有网络下的TSDB实例。详细内容，请参见时序数据库（TSDB）文档。

> **注意** 时序数据库（TSDB）实例于2022年04月06日停止新购，建议您将数据流转到时序数据库（Lindorm）。使用方法，请参见数据转发到时序数据库（Lindorm）。
>
> 对于2022年04月06日（不包含）前已经购买的TSDB实例支持正常续费和升配。

背景信息

本文示例中，写入时序数据库的数据如下：

- timestamp：使用函数 `timestamp()` 获取设备上报数据的当前时间。
- tag：使用函数 `deviceName()` 获取设备名称，写入标签 `{"deviceName":deviceName()}` 。
- metric：使用JSONPath方法，获取设备上报的物模型属性Temperature和Humidity的值，分别写入温、湿度的时序数据。

限制说明

- 仅支持同地域转发。例如：华东2（上海）的物联网平台实例数据只能转发到华东2（上海）的TSDB实例中。
- 仅支持转发到专有网络（VPC）下的TSDB实例。
- 仅支持转发JSON格式数据。
- 转发的消息中，除了配置为timestamp和tag值的字段外，其他字段都将作为metric写入时序数据库。metric的数据类型支持数值型、字符串，其他类型会导致写入数据库失败。

创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

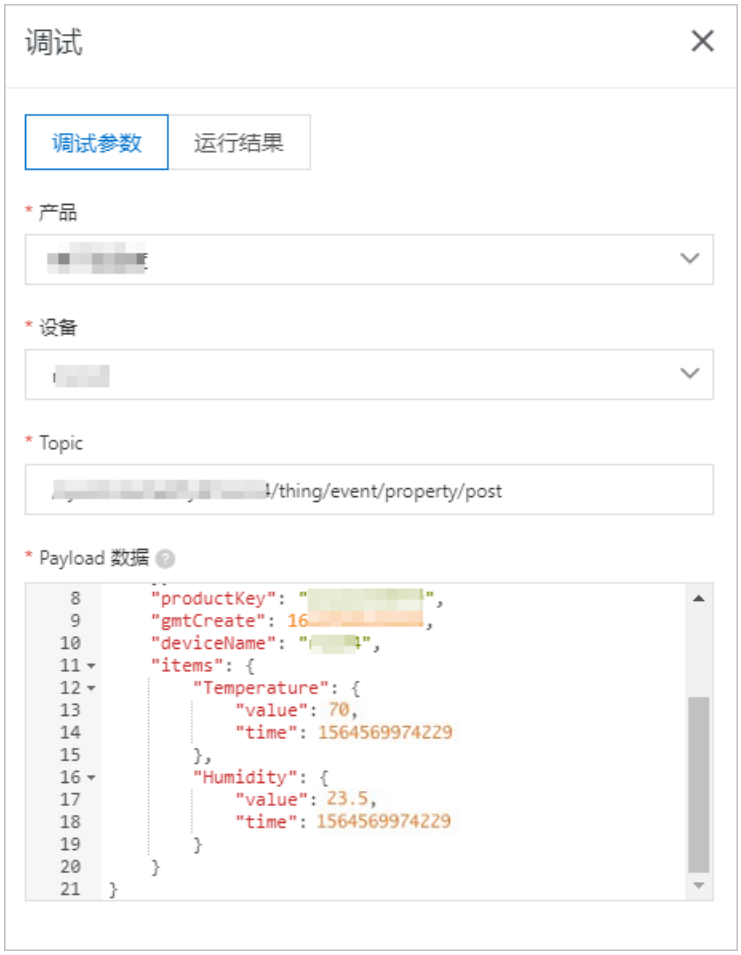
参数	描述
选择操作	选择**存储到时序数据库（TSDB）中**。
地域	与您的物联网平台实例所在地域一致。
TSDB实例	选择数据转发目的为您已创建的专有网络（VPC）下的TSDB实例。
metric数据类型	选择metric的数据类型。支持**数值型**和**字符串**。 更多信息，可单击帮助按钮查看。
角色	授权物联网平台将数据写入TSDB。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//筛选出上报的温湿度值。
var h = data.items.Humidity.value;
var t = data.items.Temperature.value;
//直接流转物模型上报数据。
writeTsdb(1000,timestamp(),"temperature", t , {"deviceName":deviceName()});
writeTsdb(1000,timestamp(),"humidity", h , {"deviceName":deviceName()});
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：



运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。

5.5.9. 数据转发到函数计算

您可以使用规则引擎数据流转，将数据转发至函数计算（FC）中，然后由函数计算运行函数脚本进行业务处理。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已创建函数计算的服务和函数，并完成函数配置，验证函数能正常执行。函数计算使用方法，请参见函数

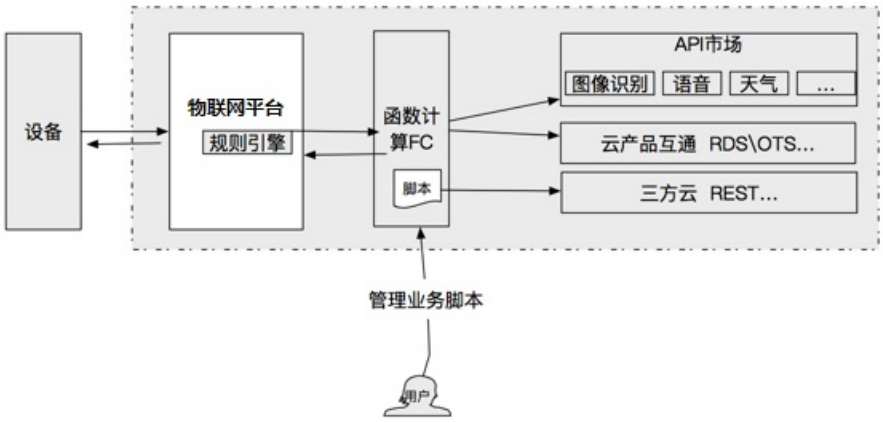
计算文档。

> **注意** 企业版实例中，函数计算的服务和函数所在地域，必须与企业版实例所在地域一致。

背景信息

使用规则引擎数据转发功能，将设备中的数据转发到函数计算，函数计算执行函数的业务脚本，最终实现丰富的业务功能。

数据流转示意图如下。



创建数据目的

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	说明
选择操作	选择**发送数据到函数计算（FC）中**。
地域	选择函数计算所在地域。
服务	选择函数计算服务。 您可单击**创建服务**，跳转到函数计算控制台创建服务。服务的详细说明，请参见管理服务。

参数	说明
函数版本选择方式	可选： ○ **使用默认版本**：使用函数计算的默认版本LATEST。 ○ **选择版本**：选择您为函数计算服务发布的版本。 您可单击**创建版本**，跳转到函数控制台创建版本。版本的详细内容，请参见管理版本。 ○ **选择别名**：选择您为函数计算服务版本设置的别名。 您可单击**创建别名**，跳转到函数控制台创建别名。别名的详细内容，请参见管理别名。
函数	选择接收数据的函数。 您可以单击**创建函数**，跳转到函数计算控制台创建函数。函数的详细说明，请参见管理函数。
授权	授权物联网平台将数据写入函数计算。 如您还未创建相关角色，单击**创建RAM角色**，跳转到RAM控制台，创建角色和授权策略，请参见创建RAM角色。

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//直接流转物模型上报数据。
writeFc(1000, data)
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：

运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。
9. 登录函数计算控制台，在函数详情页面的**调用日志**页签，查看函数执行记录。单击右上角**监控大盘**，查看函数的监控统计。

> 注意 监控数据统计会有5分钟的延时。

5.5.10. 数据转发到时序数据库（Lindorm）

您可以配置数据流转的解析规则，将处理过的数据转发到时序数据库（Lindorm）中。本文以物模型数据上报Topic为例，介绍流转消息数据的完整流程。

前提条件

- 已创建数据源*DataSource*，并添加**物模型数据上报**Topic。具体步骤，请参见添加待流转的数据源。
- 已通过云原生多模数据库Lindorm控制台创建Lindorm实例，相关配置请根据业务需求选择。具体操作，请参见创建实例。

  > **注意**
  > - 仅支持同地域转发。所以Lindorm实例所在地域必须与当前物联网平台实例所在地域保持一致。
  > - Lindorm实例的**商品类型**仅支持**Lindorm（包年包月）**和**Lindorm（按量付费）**，且选择数据引擎的**时序节点数量**必须不小于1。

- 已集成Lindorm时序引擎的Java Native SDK，完成创建时序数据库、数据表，以及创建管理数据库的账号和密码，以及支持多实例时序数据存储的配置。具体操作，请参见Java Native SDK。

背景信息

本文示例中，写入时序数据库的数据如下：

- timestamp：使用函数 `timestamp()` 获取设备上报数据的当前时间。
- tags：使用函数 `deviceName()` 获取设备名称，写入标签 `{"device_id": deviceName(), "region": "cn-shanghai"}` 。
- fields：使用JSONPath方法，获取设备上报的物模型属性Temperature和Humidity的值，写入温、湿度的时序数据。

使用限制

目前，仅新版云产品流转功能支持将数据流转到时序数据库（Lindorm）中。

创建数据目的

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

   > **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. （可选）在**云产品流转**页面，单击右上角**体验新版**，进入新版功能页面。

> **说明** 如果您已执行过此操作，再次进入**云产品流转**页面，会直接进入新版功能页面。

5. 单击**数据目的**页签，然后单击**创建数据目的**。
6. 在**创建数据目的**对话框，输入数据目的的名称，例如*DataPurpose*，按照以下参数说明，完成配置，然后单击**确定**。

参数	描述
选择操作	选择**存储到时序数据库（Lindorm）中**。
授权	单击**授权**，系统创建角色和授权策略，授权物联网平台将数据写入Lindorm。详细说明，请参见数据转发到时序数据库（Lindorm）服务关联角色。 **说明** 若已授权，忽略此操作。
地域	与您的物联网平台实例所在地域一致。
Lindorm实例	选择数据转发目的为您已创建的专有网络（VPC）下的Lindorm实例。
数据库	输入Lindorm实例下接收数据的目标数据库名称。

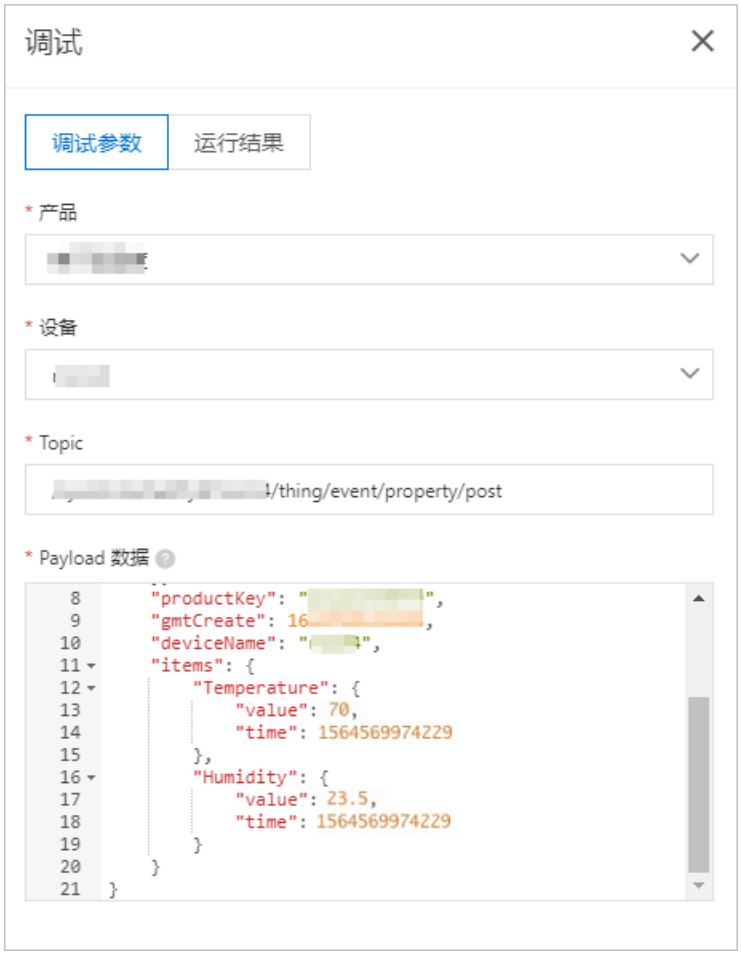
参数	描述
表名	输入目标数据库下接收数据的时序数据表名称。
账号	输入管理时序数据库的账号和密码。 数据库管理账号和密码说明，请参见用户及权限管理。
密码	

配置并启动解析器

1. 创建解析器，例如*DataParser*。具体操作，请参见创建解析器。
2. 在**解析器详情**页面，关联数据源。
   i. 在配置向导的**数据源**下，单击**关联数据源**。
   ii. 在弹出的对话框中，单击**数据源**下拉列表，选择已创建的数据源*DataSource*，单击**确定**。
3. 在**解析器详情**页面，关联数据目的。
   i. 单击配置向导的**数据目的**，然后单击**数据目的**列表右上方的**关联数据目的**。
   ii. 在弹出的对话框中，单击**数据目的**下拉列表，选择已创建的数据目的*DataPurpose*，单击**确定**。
   iii. 在数据目的的列表，查看并保存**数据目的ID**，例如为**1000**。
   后续解析脚本中，需使用此处的**数据目的ID**。
4. 在**解析器详情**页面，单击**解析器**。
5. 在脚本输入框，输入解析脚本。脚本编辑方法，请参见脚本示例。
   函数参数说明，请参见函数列表。

```
//通过payload函数，获取设备上报的消息内容，并按照JSON格式转换。
var data = payload("json");
//筛选出上报的温湿度值。
var h = data.items.Humidity.value;
var t = data.items.Temperature.value;
var fields = {"temperature":t, "humidity":h};
var tags = {"device_id": deviceName(), "region": "cn-shanghai"};
var timestamp = timestamp();
//流转物模型上报数据。
writeLindorm(1000, timestamp, tags, fields);
```

6. 单击**调试**，根据页面提示，选择产品和设备，输入Topic和Payload数据，验证脚本可执行。
   参数示例如下：



运行结果如下，表示脚本执行成功。

7. 单击**发布**。
8. 回到**云产品流转**页面的**解析器**页签，单击解析器*DataParser*对应的**启动**按钮，启动解析器。
9. 模拟设备上报物模型数据，查看流转结果。具体操作，请参见设备模拟器。

   设备上报数据流转到Lindorm时序数据库后，您可通过Lindorm TSDB SDK的查询接口查看流转的数据。

6.数据格式

物联网平台的云产品流转和服务端订阅，是基于Topic中的数据格式来处理和传递数据的。Topic可分为自定义Topic、基础通信Topic和物模型Topic，其中自定义Topic中的数据格式由您定义。本文介绍基础通信Topic和物模型通信Topic数据的数据格式。

规则引擎流转与设备原始通信的Topic说明

本文介绍设备上报数据经过规则引擎转化后的数据格式，设备通信的原始数据格式说明，请参见Alink协议目录下对应文档。Topic对应说明如下表。

Topic说明

Topic	说明	相关文档
自定义	流转自定义数据格式消息的Topic，与自定义Topic的格式相同。格式为：`/${productKey}/${deviceName}/user/${TopicShortName}`。 其中`${TopicShortName}`为自定义的Topic类，即自定义Topic的后缀。 支持使用通配符（+）和（#）： • 全部设备（`+`）：指定产品下所有设备。 • `/user/#`：指定设备的所有自定义Topic。	自定义Topic
设备状态变化通知	流转设备上下线状态变更消息的Topic：`/as/mqtt/status/${productKey}/${deviceName}`。	设备上下线状态
物模型数据上报	包含： • 流转设备上报属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/post`。 • 流转设备上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`。 • 流转设备批量上报属性数据的Topic：`/${productKey}/${deviceName}/thing/property/batch/post`。 • 流转设备批量上报事件数据的Topic：`/${productKey}/${deviceName}/thing/event/batch/post`。 • 流转设备响应云端命令返回消息的Topic：`/${productKey}/${deviceName}/thing/downlink/reply/message`。	• 设备上报属性 • 设备上报事件 • 设备属性批量上报 • 设备事件批量上报 • 设备下行指令结果

Topic	说明	相关文档
	对应设备Topic如下： • 设备上报属性的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/post`。 • 设备上报事件的Topic：`/sys/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`、`/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:{tsl.event.identifier}/post`。 • 设备批量上报属性、事件数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/batch/post`。	• 设备上报属性 • 设备上报事件 • 设备批量上报属性、事件
设备生命周期变更	流转设备创建、删除、禁用、启用等消息的Topic：`/${productKey}/${deviceName}/thing/lifecycle`。	设备生命周期变更
网关发现子设备上报	网关设备特有的Topic：`/${productKey}/${deviceName}/thing/list/found`，将发现的子设备信息上报给物联网平台，然后进行流转。	网关发现子设备
设备拓扑关系变更	网关设备特有Topic：`/${productKey}/${deviceName}/thing/topo/lifecycle`，流转子设备和网关之间的拓扑关系建立和解除消息的Topic。	设备拓扑关系变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/topo/change`。	通知网关拓扑关系变化
设备标签变更	流转设备标签信息变更的Topic：`/${productKey}/${deviceName}/thing/deviceinfo/update`。	设备标签变更
	对应设备上报数据的Topic：`/sys/${productKey}/${deviceName}/thing/deviceinfo/update`。	标签信息上报
物模型历史数据上报	包含： • 流转设备上报历史属性数据的Topic：`/${productKey}/${deviceName}/thing/event/property/history/post`。 • 流转设备上报历史事件数据的Topic：`/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/history/post`。	• 历史属性上报 • 历史事件上报
	对应设备上报物模型历史数据的Topic：`/sys/${productKey}/${deviceName}/thing/event/property/history/post`。	物模型历史数据上报

Topic	说明	相关文档
OTA升级设备状态通知	包含： • 流转设备上报OTA升级结果的Topic：`/${productKey}/${deviceName}/ota/upgrade`。 • 流转设备上报OTA升级进度的Topic：`/${productKey}/${deviceName}/ota/progress/post`。	• OTA升级状态通知 • OTA升级进度通知
	对应设备上报升级进度的Topic：`/ota/device/progress/${productKey}/${deviceName}`。	设备上报升级进度
OTA模块版本号上报	流转设备上报OTA模块版本号变更的Topic：`/${productKey}/${deviceName}/ota/version/post`。	OTA模块版本号变更通知
	对应设备上报OTA模块版本的Topic：`/ota/device/inform/${productKey}/${deviceName}`。	设备上报OTA模块版本
OTA升级批次状态通知	物联网平台通知OTA升级批次状态变化的Topic：`/${productKey}/${packageId}/${jobId}/ota/job/status`。	OTA升级批次状态通知
任务事件	包含： • 流转设备任务状态通知的Topic：`/sys/uid/${uid}/job/${jobId}/lifecycle`。 • 流转实例迁移任务状态通知的Topic：`/sys/uid/${uid}/distribution/${jobId}/lifecycle`。 **说明** 迁移产品的名称为实例迁移的任务名称。	• 设备任务的状态通知 • 实例迁移任务的状态通知
孪生节点属性变更	流转数字孪生节点属性数据的Topic：`/sys/uid/${uid}/digitaltwin/${dtInstanceId}/${nodeId}/property/update`。	数字孪生节点属性变更

设备上下线状态

Topic：`/as/mqtt/status/${productKey}/${deviceName}`

通过该Topic获取设备的上下线状态。

设备上线的数据格式：

```
{
    "status":"online",
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "productKey":"al12345****",
    "deviceName":"deviceName1234",
    "time":"2018-08-31 15:32:28.205",
    "utcTime":"2018-08-31T07:32:28.205Z",
    "lastTime":"2018-08-31 15:32:28.195",
    "utcLastTime":"2018-08-31T07:32:28.195Z",
    "clientIp":"192.0.2.1"
}
```

设备下线的数据格式：

```
{
    "status":"offline",
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "offlineReasonCode":427,
    "productKey":"al12345****",
    "deviceName":"deviceName1234",
    "time":"2018-08-31 15:32:28.205",
    "utcTime":"2018-08-31T07:32:28.205Z",
    "lastTime":"2018-08-31 15:32:28.195",
    "utcLastTime":"2018-08-31T07:32:28.195Z",
    "clientIp":"192.0.2.1"
}
```

参数说明：

参数	类型	说明
status	String	设备状态。 • *online*：上线。 • *offline*：离线。
iotId	String	设备在平台内的唯一标识。
offlineReasonCode	Integer	设备下线时，返回的错误码。详细说明，请参见设备行为错误码。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
lastTime	String	该参数为历史存量字段，已无实际意义。
utcLastTime	String	

参数	类型	说明
Time	String	设备上、下线的时间。 收到消息的时间不是实际设备上下线时间。设备上下线顺序可按照Time排序。 例如，您依次收到3条消息： 1. 上线：`2018-08-31 10:02:28.195`。 2. 下线：`2018-08-31 10:01:28.195`。 3. 下线：`2018-08-31 10:03:28.195`。 这3条消息展示了，设备先下线，再上线，最后下线的过程。
utcTime	String	设备上、下线的UTC时间。
clientIp	String	设备公网出口IP。

设备属性上报

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

通过该Topic获取设备上报的属性信息。

数据格式：

```
{
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "requestId":"2",
    "productKey":"al12345****",
    "deviceName":"deviceName1234",
    "gmtCreate":1510799670074,
    "deviceType":"Ammeter",
    "items":{
        "Power":{
            "value":"on",
            "time":1510799670074
        },
        "Position":{
            "time":1510292697470,
            "value":{
                "latitude":39.9,
                "longitude":116.38
            }
        }
    },
    "checkFailedData":{
        "attribute_8":{
            "time": 1510292697470,
            "value": 715665571,
            "code":6304,
            "message":"tsl parse: params not exist -> attribute_8"
        }
    }
}
```

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识。
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
gmtCreate	Long	数据流转消息产生时间。
deviceType	String	设备所属品类。
items	Object	设备数据。

参数	类型	说明
Power	String	属性标识符。产品所具有的属性名称请参见产品的TSL描述。 如果是自定义模块属性，属性标识符格式为 **模块标识符:属性标识符** （中间为半角冒号）。例如，物模型自定义模块标识符为test，数据格式为： `"items":{ "test:Power":{ "value":"on", "time":1510799670074 }, "test:Position":{ "time":1510292697470, "value":{ "latitude":39.9, "longitude":116.38 } } }`
Position		
attribute_8		
checkFailedData	Object	未通过物模型数据校验的数据。
value	根据TSL定义	属性值。
time	Long	上报属性的时间，如果设备没有上报数据，默认采用在物联网平台生成的时间。
code	Integer	数据未通过物模型数据校验时，返回的错误码。详细说明，请参见设备端接收的错误码。
message	String	数据未通过物模型数据校验时，返回的错误码信息，包含错误原因和具体的错误参数。

设备事件上报

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

通过该Topic获取设备上报的事件信息。

数据格式：

```
{
    "identifier":"BrokenInfo",
    "name":"损坏率上报",
    "type":"info",
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "requestId":"2",
    "productKey":"X5eCzh6****",
    "deviceName":"5gJtxDVeGAkaEztpisjX",
    "gmtCreate":1510799670074,
    "value":{
        "Power":"on",
        "Position":{
            "latitude":39.9,
            "longitude":116.38
        }
    },
    "checkFailedData":{
    },
    "time":1510799670074
}
```

参数说明：

参数	类型	说明
identifier	String	事件的标识符。 如果是自定义模块事件，事件标识符格式为 `模块标识符:事件标识符` （中间为半角冒号）。例如，物模型自定义模块标识符为test，数据格式为： `"test:identifier":"BrokenInfo",`
name	String	事件的名称。
type	String	事件类型，事件类型参见产品的TSL描述。
iotId	String	设备在平台内的唯一标识。
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
gmtCreate	Long	数据流转消息产生时间。

参数	类型	说明
value	Object	事件的输出参数信息。如以上示例中的两个参数Power（电源）和Position（位置）的信息。 `{ "Power":"on", "Position":{ "latitude":39.9, "longitude":116.38 } }` **注意** • 仅当事件的输出参数均通过检验时，输出参数均显示在value中。此时checkFailedData为空。 • 当事件的输出参数有任一参数未通过检验时，输出参数均显示在checkFailedData中。此时value为空。
checkFailedData	Object	未通过物模型数据校验的数据信息。 若事件的输出参数未通过检验，则checkFailedData为： `{ "value":{ "Power":"on", "Position":{ "latitude":39.9, "longitude":116.38 } } "time":1524448722000, "code":6304, "message":"tsl parse: params not exist -> type" }` 其中： • value：Object类型。事件的输出参数信息。 • time：Long类型。事件生成的时间戳。 • code：Integer类型。数据未通过物模型数据校验时，返回的错误码。详细说明，请参见设备端接收的错误码。 • message：String类型。数据未通过物模型数据校验时，返回的错误码信息，包含错误原因和具体的错误参数。
time	Long	上报事件的时间，如果设备没有上报数据，默认采用物联网平台生成的时间。

设备属性批量上报

Topic： `/${productKey}/${deviceName}/thing/property/batch/post`

通过该Topic获取设备批量上报的属性信息。

数据格式：

```
{
    "productKey": "al12345****",
    "deviceName": "deviceName1234",
    "instanceId": "iot-0***",
    "requestId": "2",
    "payload": {
        "Power": [{
                "value": "on",
                "time": 1524448722000
            },
            {
                "value": "off",
                "time": 1524448722001
            }
        ],
        "WF": [{
                "value": 3,
                "time": 1524448722000
            },
            {
                "value": 4,
                "time": 1524448722009
            }
        ]
    }
}
```

参数说明：

参数	类型	说明
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
instanceId	String	设备所属实例的实例ID。
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
payload	Object	设备数据。

参数	类型	说明
Power	String	属性标识符。产品所具有的属性名称请参见产品的TSL描述。 如果是自定义模块属性，属性标识符格式为 **模块标识符:属性标识符** （中间为半角冒号）。例如，物模型自定义模块标识符为test，数据格式为： `"payload":{ "test:Power":[ { "value":"on", "time":1510799670074 }, { "value": "off", "time": 1524448722001 }], "test:WF":[ { "value": 3, "time": 1524448722000 }, { "value": 4, "time": 1524448722009 }] } }`
WF		
value	根据TSL定义	属性值。
time	Long	上报属性的时间，如果设备没有上报数据，默认采用在物联网平台生成的时间。

设备事件批量上报

Topic： `/${productKey}/${deviceName}/thing/event/batch/post`

通过该Topic获取设备批量上报的事件信息。

数据格式：

```
{
    "productKey": "al12345****",
    "deviceName": "deviceName1234",
    "instanceId": "iot-0***",
    "requestId": "2",
    "payload": {
        "alarmEvent": [{
                "value": {
                    "Power": "on",
                    "WF": "2"
                },
                "time": 1524448722000
            },
            {
                "value": {
                    "Power": "on",
                    "WF": "2"
                },
                "time": 1524448723000
            }
        ]
    }
}
```

参数说明：

参数	类型	说明
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
instanceId	String	设备所属实例的实例ID。
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
payload	Object	设备数据。
alarmEvent	List	事件的标识符。
value	Object	事件的参数。 以上示例中，Power和WF是事件的参数名称。
time	Long	上报事件的时间，如果设备没有上报数据，默认采用在物联网平台生成的时间。

设备生命周期变更

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

通过该Topic获得设备创建、删除、禁用、启用等消息。

数据格式：

```
{
    "action": "create|delete|enable|disable",
    "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
    "productKey": "al5eCzh****",
    "deviceName": "5gJtxDVeGAkaEztpisjX",
    "deviceSecret": "wsde***",
    "messageCreateTime": 1510292739881
}
```

参数说明：

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

设备拓扑关系变更

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

通过该Topic获得子设备和网关之间拓扑关系建立和解除信息。

数据格式：

```
{
    "action" : "create|delete|enable|disable",
    "gwIotId": "dfaejVQHk6VSLmmBJfrf00107e****",
    "gwProductKey": "al5eCzh****",
    "gwDeviceName": "deviceName1234",
    "devices": [
        {
          "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
          "productKey": "ala4Czh****",
          "deviceName": "deviceName1234"
       }
    ],
    "messageCreateTime": 1510292739881
}
```

参数说明：

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

网关发现子设备

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

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

数据格式：

```
{
    "gwIotId":"dfaew9VQHk6VSLmmBJfrf00107e****",
    "gwProductKey":"al12345****",
    "gwDeviceName":"deviceName1234",
    "devices":[
        {
            "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
            "productKey":"alr56g9****",
            "deviceName":"deviceName1234"
        }
    ]
}
```

参数说明：

参数	类型	说明
gwIotId	String	网关设备在平台内的唯一标识。

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

设备下行指令结果

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

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

数据格式：

```
{
    "gmtCreate":1510292739881,
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "productKey":"al12355****",
    "deviceName":"deviceName1234",
    "requestId":"2",
    "code":200,
    "message":"success",
    "topic":"/sys/al12355****/deviceName1234/thing/service/property/set",
    "data":{
    },
    "checkFailedData":{
        "value": {
            "PicID": "15194139"
        },
        "code":6304,
        "message":"tsl parse: params not exist -> PicID"
    }
}
```

参数说明：

参数	类型	说明
gmtCreate	Long	UTC时间戳。
iotId	String	设备在平台内的唯一标识。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。

参数	类型	说明
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
code	Integer	设备回复的结果状态码，说明参见下表结果状态码。
message	String	设备回复的结果状态码信息。
topic	String	下发指令给设备的Topic信息。
data	Object	设备返回的结果。Alink格式数据直接返回设备处理结果，透传格式数据则需要经过脚本转换。
checkFailedData	Object	未通过物模型数据校验的数据。
value	根据TSL定义	未通过物模型数据校验的属性值、服务参数值。 以上示例中，PicID是未通过检验的参数名称。
code	Integer	数据未通过物模型数据校验时，返回的错误码。详细说明，请参见设备端接收的错误码。
message	String	数据未通过物模型数据校验时，返回的错误码信息，包含错误原因和具体的错误参数。

结果状态码

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

错误码相应解决办法，请参见设备端接收的错误码。

历史属性上报

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

通过该Topic获取设备上报的物模型历史属性数据。

数据格式：

```
{
    "iotId":"4z819VQHk6VSLmmBJfrf00107e****",
    "requestId":"2",
    "productKey":"12345****",
    "deviceName":"deviceName1234",
    "gmtCreate":1510799670074,
    "deviceType":"Ammeter",
    "items":{
        "Power":{
            "value":"on",
            "time":1510799670074
        },
        "Position":{
            "time":1510292697470,
            "value":{
                "latitude":39.9,
                "longitude":116.38
            }
        }
    },
    "checkFailedData":{
        "attribute_8":{
            "time": 1510292697470,
            "value": 715665571,
            "code":6304,
            "message":"tsl parse: params not exist -> attribute_8"
        }
    }
}
```

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识。
requestId	String	设备上报消息中的Id，String类型的数字，取值范围为0~4294967295，且每个消息ID在当前设备中具有唯一性。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
gmtCreate	Long	数据流转消息产生时间。
deviceType	String	设备所属品类。
items	Object	设备数据。

参数	类型	说明
Power	String	属性标识符。产品所具有的属性名称请参见产品的TSL描述。 如果是自定义模块属性，属性标识符格式为 **模块标识符:属性标识符** （中间为半角冒号）。例如，物模型自定义模块标识符为test，数据格式为： `"items":{ "test:Power":{ "value":"on", "time":1510799670074 }, "test:Position":{ "time":1510292697470, "value":{ "latitude":39.9, "longitude":116.38 } } }`
Position		
attribute_8		
checkFailedData	Object	未通过物模型数据校验的数据。
value	根据TSL定义	属性值。
time	Long	上报属性的时间，如果设备没有上报数据，默认采用在物联网平台生成的时间。
code	Integer	数据未通过物模型数据校验时，返回的错误码。详细说明，请参见设备端接收的错误码。
message	String	数据未通过物模型数据校验时，返回的错误码信息，包含错误原因和具体的错误参数。

历史事件上报

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

通过该Topic获取设备上报的历史事件数据。

数据格式：

```
{
    "identifier":"BrokenInfo",
    "name":"损坏率上报",
    "type":"info",
    "iotId":"4z819VQHk6VSLmmBJfrf00107e***",
    "requestId":"2",
    "productKey":"X5eCzh6***",
    "deviceName":"5gJtxDVeGAkaEztpisjX",
    "value":{
        "Power":"on",
        "Position":{
            "latitude":39.9,
            "longitude":116.38
        }
    },
    "checkFailedData":{
    },
    "time":1510799670074
}
```

参数说明：

参数	类型	说明
identifier	String	事件的标识符。 如果是自定义模块事件，事件标识符格式为 `模块标识符:事件标识符` 。例如，物模型自定义模块标识符为test，数据格式为： `"test:identifier":"BrokenInfo",`
name	String	事件的名称。
type	String	事件类型，事件类型参见产品的TSL描述。
iotId	String	设备在平台内的唯一标识。
requestId	String	
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
gmtCreate	Long	数据流转消息产生时间。

参数	类型	说明
value	Object	事件的输出参数信息。如以上示例中的两个参数Power（电源）和Position（位置）的信息。 `{ "Power":"on", "Position":{ "latitude":39.9, "longitude":116.38 } }` **注意** • 仅当事件的输出参数均通过检验时，输出参数均显示在value中。此时checkFailedData为空。 • 当事件的输出参数有任一参数未通过检验时，输出参数均显示在checkFailedData中。此时value为空。
checkFailedData	Object	未通过物模型数据校验的数据信息。 若事件的输出参数未通过检验，则checkFailedData为： `{ "value":{ "Power":"on", "Position":{ "latitude":39.9, "longitude":116.38 } } "time":1524448722000, "code":6304, "message":"tsl parse: params not exist -> type" }` 其中： • value：Object类型。事件的输出参数信息。 • time：Long类型。事件生成的时间戳。 • code：Integer类型。数据未通过物模型数据校验时，返回的错误码。详细说明，请参见设备端接收的错误码。 • message：String类型。数据未通过物模型数据校验时，返回的错误码信息，包含错误原因和具体的错误参数。
time	Long	上报事件的时间，如果设备没有上报数据，默认采用物联网平台生成的时间。

OTA升级状态通知

Topic： `/${productKey}/${deviceName}/ota/upgrade`

通过该Topic获得OTA升级状态消息，即设备OTA升级成功或失败的消息。

> **说明** 如果设备有未完成的升级任务，又对该设备发起批量升级任务，后发起的升级任务将升级失败。这种情况下，物联网平台不流转升级失败消息。

数据格式：

```
{
    "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
    "productKey": "X5eCzh6****",
    "deviceName": "deviceName1234",
    "moduleName": "default",
    "status": "SUCCEEDED|FAILED|CANCELED",
    "messageCreateTime": 1571323748000,
    "srcVersion": "1.0.1",
    "destVersion": "1.0.2",
    "desc": "success",
    "jobId": "wahVIzGkCMuAUE2gDERM02****",
    "taskId": "y3tOmCDNgpR8F9jnVEzC01****"
}
```

参数说明：

参数	类型	说明
iotId	String	设备ID，设备的唯一标识符。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
moduleName	String	OTA模块名称。
status	String	升级状态。 • *SUCCEEDED*：升级成功。 • *FAILED*：升级失败。 • *CANCELED*：升级已取消。
messageCreateTime	Long	消息产生时间戳，单位毫秒。
srcVersion	String	升级前的版本。
destVersion	String	升级目标版本。
desc	String	升级状态描述信息。
jobId	String	升级批次ID，升级批次的唯一标识符。
taskId	String	设备升级记录的唯一标识符。

OTA升级进度通知

Topic： `/${productKey}/${deviceName}/ota/progress/post`

通过该Topic获得OTA升级中进度消息。

数据格式：

```
{
    "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
    "productKey": "X5eCzh6****",
    "deviceName": "deviceName1234",
    "moduleName":"default",
    "status":"IN_PROGRESS",
    "step": "90",
    "messageCreateTime": 1571323748000,
    "srcVersion":"1.0.1",
    "destVersion":"1.0.2",
    "desc": "success",
    "jobId": "wahVIzGkCMuAUE2gDERM02****",
    "taskId": "y3tOmCDNgpR8F9jnVEzC01****"
}
```

参数说明：

参数	类型	说明
iotId	String	设备ID，设备的唯一标识符。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
moduleName	String	OTA模块名称。
status	String	升级状态。此处取固定值：IN_PROGRESS（升级中）。
step	Integer	设备上报的升级进度。
messageCreateTime	Long	消息产生时间戳，单位毫秒。
srcVersion	String	升级前的版本。
destVersion	String	升级目标版本。
desc	String	升级状态描述信息。
jobId	String	升级批次ID，升级批次的唯一标识符。
taskId	String	设备升级记录的唯一标识符。

OTA模块版本号变更通知

Topic： `/${productKey}/${deviceName}/ota/version/post`

通过该Topic获得设备上报的OTA模块版本号。当设备上报OTA模块版本号，且版本号有变更时进行转发。

数据格式：

```
{
    "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
    "deviceName": "deviceName1234",
    "productKey": "X5eCzh6****",
    "moduleName": "BarcodeScanner",
    "moduleVersion": "1.0.3",
    "messageCreateTime": 1571323748000
}
```

参数说明：

参数	类型	说明
iotId	String	设备ID，设备的唯一标识符。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
moduleName	String	模块名称。
moduleVersion	String	模块版本号。
messageCreateTime	Long	消息产生时间戳，单位毫秒。

OTA升级批次状态通知

Topic： `/${productKey}/${packageId}/${jobId}/ota/job/status`

通过该Topic获得OTA升级批次状态变更消息。

数据格式：

```
{
    "productKey": "X5eCzh6****",
    "moduleName": "BarcodeScanner",
    "packageId": "wahVIzGkCMuAUE2***",
    "jobId": "wahVIzGkCMuAUE2gDERM02****",
    "state": "IN_PROGRESS",
    "messageCreateTime": 1571323748000
}
```

参数说明：

参数	类型	说明
productKey	String	设备所属产品的唯一标识。
moduleName	String	模块名称。

参数	类型	说明
packageId	String	升级包ID。是使用CreateOTAFirmware创建升级包时，返回的FirmwareId参数。
jobId	String	升级批次ID，升级批次的唯一标识符。
state	String	升级批次状态，取值： • PLANNED：未开始升级。 • IN_PROGRESS：升级中。 • COMPLETED：升级完毕。 • CANCELED：已取消。
messageCreateTime	Long	消息产生时间戳，单位毫秒。

设备标签变更

Topic： `/${productKey}/${deviceName}/thing/deviceinfo/update`

通过该Topic获得设备的标签变更信息。

数据格式：

```
{
  "action": "UPDATE|DELETE|DELETEALL"
  "iotId": "4z819VQHk6VSLmmBJfrf00107e****",
  "productKey": "X5eCzh6****",
  "deviceName": "deviceName1234",
  "deletedAttrKeyList": ["abc", "def", "rng"],
  "value": [
    {
      "attrKey": "tagKey",
      "attrValue": "tagValue"
    }
  ],
  "messageCreateTime": 1510799670074
}
```

参数说明：

参数	类型	说明
action	String	标签变更的类型。可取值： • UPDATE：更新或新增标签。 • DELETE：删除指定标签。 • DELETEALL：清空所有标签。
iotId	String	设备ID，设备的唯一标识符。
productKey	String	设备所属产品的唯一标识。

参数	类型	说明
deviceName	String	设备名称。
deletedAttrKeyList	List	删除标签key的列表。 仅当action为DELETE时，显示该参数。
value	List	标签数据。
attrKey	String	标签Key。
attrValue	String	标签Value。
messageCreateTime	Long	消息产生时间戳，单位毫秒。

设备任务的状态通知

Topic：`/sys/uid/${uid}/job/${jobId}/lifecycle`

通过该Topic获得设备任务（设备批量属性设置任务、设备批量服务调用任务、自定义任务）状态变更消息。

其中，`${uid}`为阿里云账号ID，可登录物联网平台控制台，单击账号头像，跳转至**安全设置**页面查看。

数据格式：

```
{
    "jobId": "4z819VQHk6VSLmm***ee200",
    "jobType": "CUSTOM_JOB",
    "status": "INITIALIZING",
    "messageCreateTime": 1510292739881
}
```

参数说明：

参数	类型	说明
jobId	String	任务ID，任务的全局唯一标识符。
jobType	String	任务类型。 • `SET_PROPERTY`：设备批量属性设置任务。 • `INVOKE_SERVICE`：设备批量服务调用任务。 • `CUSTOM_JOB`：自定义任务。

参数	类型	说明
status	String	任务状态，取值： • INITIALIZING：初始化。 • WAITING：待调度。 • IN_PROGRESS：执行中。 • COMPLETED：已完成。 • CANCELLING：取消中。 • CANCELLED：已取消。 • REMOVING：删除中。
messageCreateTime	Long	消息产生时间戳，单位毫秒。

实例迁移任务的状态通知

Topic： `/sys/uid/{uid}/distribution/{jobId}/lifecycle`

通过该Topic获得实例迁移任务状态的变更消息。

其中，`${uid}` 为阿里云账号ID，可登录物联网平台控制台，将鼠标指针移动到账号头像，查看**账号ID**。

数据格式：

```
{
    "jobId": "4z819VQHk6VSLmmxxxxxxxxxxee200",
    "status": "GRAY_EXECUTING",
    "messageCreateTime": 1510292739881,
    "type":"INSTANCE_UPGRADE",
    "sourceInstanceId":"iotx-oxssharez200",
    "targetInstanceId":"iot-es5v4***",
    "successDevices":[{
      "productKey":"al12***",
      "deviceName":"deviceName1",
      "iotId":"4z81frf00107e***"
    }]
}
```

参数说明：

参数	类型	说明
jobId	String	任务ID，任务的全局唯一标识符。

参数	类型	说明
status	String	任务状态，取值： • GRAY_EXECUTING：灰度中。 • GRAY_FINISHED：灰度完成。 • ALL_EXECUTING：迁移中。 • ALL_FINISHED：迁移完成。 • ALL_PAUSE：迁移暂停。 • ROLL_BACK_EXECUTING：回滚中。 • ROLL_BACK_PAUSE：暂停回滚。
messageCreateTime	Long	消息产生时间戳，单位毫秒。
type	String	任务类型。固定取值为 `INSTANCE_UPGRADE` 。
sourceInstanceId	String	实例迁移的源公共实例ID。旧版公共实例ID为 `iotx-oxssharez200` 。 实例的详细说明，请参见实例概述。
targetInstanceId	String	实例迁移的目标企业版实例ID。
successDevices	List	成功迁移的设备信息： • 灰度迁移和全量迁移中，从公共实例成功迁移到目标企业版实例的设备信息。 • 回滚迁移时，从企业版实例成功迁移回公共实例的设备信息。
productKey	String	设备所属产品的唯一标识。
deviceName	String	设备名称。
iotId	String	设备在平台内的唯一标识。

数字孪生节点属性变更

Topic： `/sys/uid/${uid}/digitaltwin/${dtInstanceId}/${nodeId}/property/update` 。

通过该Topic获取数字孪生节点属性数据的变更信息。

其中， `${uid}` 为阿里云账号ID，可登录物联网平台控制台，将鼠标指针移动到账号头像，查看**账号ID**。

数据格式：

```
{
    "nodeId": "4z819VQHk6VS0107e***",
    "dtInstanceId": "al12345****",
    "gmtCreate": 1510799670074,
    "items": {
        "Power": {
            "value": "on",
            "time": 1510799670074
        },
        "Position": {
            "time": 1510292697470,
            "value": {
                "latitude": 39.9,
                "longitude": 116.38
            }
        }
    }
}
```

参数说明：

参数	类型	说明
nodeId	String	数字孪生节点的ID。
dtInstanceId	String	数字孪生体的ID。
gmtCreate	Long	数据流转消息产生时间。
items	Object	数字孪生节点下变更的属性数据。如以上示例中属性Power（电源）和Position（位置）的信息。 其中value为属性值，time为属性值变更时间。 `{ "Power": { "value": "on", "time": 1510799670074 }, "Position": { "time": 1510292697470, "value": { "latitude": 39.9, "longitude": 116.38 } } }`

7.时序数据存储管理

时序数据存储是一种高性能、低成本、稳定可靠的在线时序数据库服务，具有高效读写、高压缩比存储等优点。针对物联网设备数据采集场景，时序数据存储能解决由于设备采集点数量巨大、数据采集频率高造成的存储成本高、写入和查询分析效率低的问题。

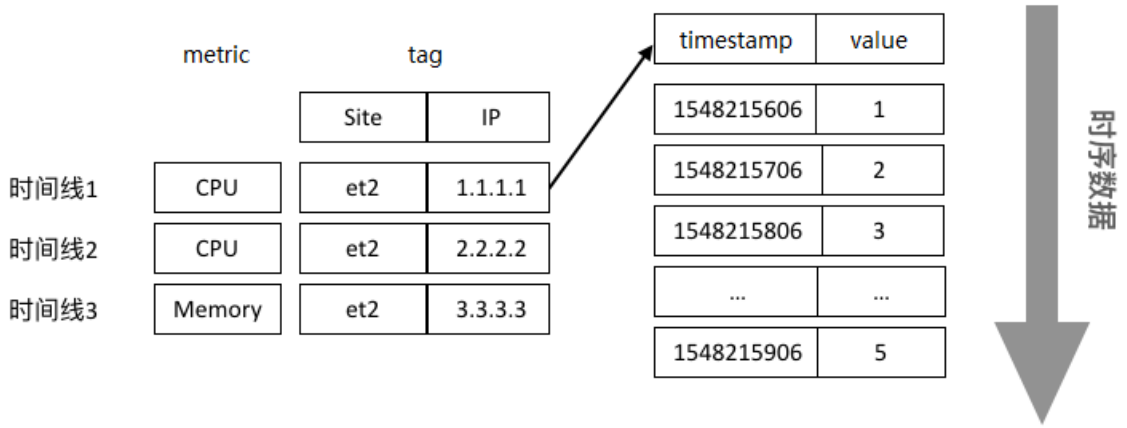
如果您购买了企业版实例，可以享受时序数据存储服务。公共实例不提供此服务。关于如何购买企业版实例，请参见购买企业版实例。

> **注意** 时序数据库（TSDB）实例于2022年04月06日停止新购，建议您将数据流转到时序数据库（Lindorm）。使用方法，请参见数据转发到时序数据库（Lindorm）。
>
> 对于2022年04月06日（不包含）前已经购买的TSDB实例支持正常续费和升配。

什么是时序数据

时序数据存储将一个metric与一组tag的组合称为一条时间线，在一条时间线下面，连续时间点的采样数据则为时序数据。

例如：下图中有3条时间线，`{"metric": "cpu","tags": "site": "et2", "ip": "1.1.1.1"}` 这个metric与tag的组合为一条时间线，在同一条时间线下面存储连续的时序数据 `(timestamp,value)` 。



时序数据存储功能

时序数据存储提供以下功能：

- 时序数据高效读写：
  - 数据写入：支持通过配置数据流转规则，将数据转发到实例内的时序数据存储进行数据写入；也支持通过SDK写入数据。最多可以支持每秒千万数据点的写入。
  - 数据查询：支持通过SDK使用SQL，或通过JDBC连接使用TSQL进行数据的查询操作。百万数据点的读取，响应时间小于5秒。
- 数据管理：

  您可以通过控制台设置数据的有效期。开启并设置数据时效后，系统将过期数据及时标记为失效，并自动在特定时间清除。
- 高压缩比存储：

单个数据点的平均使用存储空间仅需要1~2个字节，相较于常规存储降低90%存储使用空间。高压缩比存储同时能加快数据写入的速度。

- 数据安全：

  为充分保证数据的可用性，时序数据存储默认采取三副本策略。

开通时序数据存储

时序数据库（TSDB）实例于2022年04月06日停止新购开通，对于2022年04月06日（不包含）前已经开通的TSDB实例支持正常续费和升配。

您可在**实例概览**页面，单击对应实例的**升配**、**续费**购买需要的时序数据存储的规格，包括以下项目：

项目	说明
时序数据写入TPS	每秒钟写入时序数据存储的最大数据条数，达到上限后会写入失败。 时序数据写入TPS选择5千条/秒时，不支持TSQL。
时序数据存储空间	时序数据达到存储空间上限后会写入失败。 默认时序数据时效为永久，购买后时序数据时效可配置，请参见管理时序数据存储。

> **说明** 当实例被释放时，时序数据存储也将同时被释放。

管理时序数据存储

开通时序数据存储后，您可以设置时序数据时效，查看已使用的存储空间、时间线数，查看用户名、密码、连接地址，重新创建时序存储用户。

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在**实例详情**页面下方，查看实时时序存储写入TPS、实时时序存储空间数据，及对应的监控图表，设置相关报警规则。

   单击相应区域框中的**报警配置**，在云监控控制台，设置阈值报警规则。具体操作，请参见创建阈值报警规则。

4. 在**实例详情**页面，单击右上角的**查看开发配置**，查看下方的**时序存储配置**。
   - 查看时序存储用户名和密码。

     单击**查看**，短信校验码验证通过后显示密码。

     用户名、密码、连接地址用于查询时序存储中的数据，购买企业版实例时，会自动生成。
   - 重新创建时序存储用户。

     > **注意** 重新创建时序存储用户后，旧的用户名和密码将失效，会导致正在进行的时序数据查询失败，请及时更新SDK使用的用户名和密码。

     单击**重新创建**，输入新的用户名、密码，短信校验码验证通过后，创建成功。
   - 设置时序数据存储时效。

     购买企业版实例时，时效配置默认关闭。您可打开**存储时效**开关，设置保留数据天数。取值范围为1~365天。

通过云产品流转写入数据

您可以通过配置数据流转规则，将数据转发到实例内的时序数据存储。本示例使用云产品流转旧版功能，新版云产品流转配置方法，请参见数据转发到实例内的时序数据存储。

> **说明**
>
> - 在设置转发之前，请参见设置数据流转规则，创建数据转发规则和编写处理数据的SQL。
> - 只支持JSON格式数据转发。
> - 转发的消息中，除了配置的timestamp、tag值字段外，其他字段都将作为metric写入时序数据存储。metric的数据类型支持数值型、字符串，其他类型会导致写入失败。

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，选择**规则引擎 > 云产品流转**。
4. 单击规则对应的**查看**，进入**数据流转规则**页面。

> **注意** 若当前页面为云产品流转新版页面，需先单击右上角**返回旧版**，再单击目标规则对应的查看。

5. 单击**转发数据**一栏对应的**添加操作**。
6. 在**添加操作**对话框中，选择操作为**存储到实例内的时序数据存储中**。按照界面提示，设置其他信息，单击**确认**。

参数	描述
选择操作	选择**存储到实例内的时序数据存储中**。
地域	固定为您的物联网平台实例所在地域。
metric数据类型	选择metric的数据类型。支持**数值型**和**字符串**。 更多信息，可单击帮助按钮查看。
timestamp	时间戳。支持： ○ 使用转义符 `${}` 表达式，例如 `${time}` ，表示取值为数据源Topic消息中time字段对应的值。 ○ 使用数据流转函数 `timestamp()` ，表示取值为数据流转服务器的时间戳。 ○ 输入值，必须为Unix时间戳，例如1404955893000。
tag名称	设置标记数据的标签名。支持中文汉字、英文字母、数字、和特殊字符，包括：半角冒号（:）、逗号（,）、英文句号（.）、单引号（'）、正斜线（/）、短划线（-）、下划线（_）、圆括号（()）、方括号（[]）。
tag值	设置标签值。支持： ○ 使用转义符 `${}` 表达式。例如，数据源Topic的消息结构中，包含一个位置属性，标识符为city，则可以指定标签值为 `${city}` ，表示消息中city字段对应的值。建议使用此方式。 ○ 使用数据流转函数规定的一些函数，例如 `deviceName()` ，表示标签值为设备名称。支持的函数，请参见函数列表。 ○ 输入常量，例如beijing。支持输入中文汉字、英文字母、数字、和特殊字符包括：半角冒号（:）、逗号（,）、英文句号（.）、单引号（'）、正斜线（/）、短划线（-）、下划线（_）、圆括号（()）、方括号（[]）。 **说明** ○ 最多可添加8个tag键值对。 ○ 需保证时序数据存储能够获取到配置的tag键值对，如果获取不到任意一个tag键值对，会导致写入数据库失败。

7. 回到**云产品流转**页，单击规则对应的**启动**按钮启动规则。

下面给出一个数据流转示例。

示例规则的SQL:

```
SELECT time,city,power,distance, FROM "/alprodu****/myDevice/user/update";
```

规则引擎根据SQL处理数据和写入数据到时序数据存储如下。

1. 根据该SQL，规则引擎从Topic `/alprodu****/myDevice/user/update` 的消息中，筛选出time、city、power和distance字段内容，作为转发的消息内容。

   通过以上SQL处理后的转发消息内容示例如下：

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

2. 根据已配置的数据流转操作，规则引擎向时序数据存储中写入两条数据。

   示例中写入时序数据存储的数据如下：

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

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

通过SDK写入数据

SDK使用Point类表示一个时间点。一个 Point对象表示一个时间序列（时间线）某个时刻上的数据。

1. 获取SDK。

   使用Maven做项目构建工具，在pom.xml文件的<dependencies>标签中添加hitsdb-client依赖。代码如下：

   ```
   <dependency>
       <groupId>com.aliyun</groupId>
       <artifactId>hitsdb-client</artifactId>
       <version>0.2.7</version>
   </dependency>
   ```

   > **说明** SDK版本号必须为0.2.7及以上。

2. 配置客户端。

   客户端的所有配置均由TSDBConfig类进行配置。示例代码如下：

   ```
   //实例详情页面中的时序存储SQL连接地址、端口。
   String connectString = "XXX";
   int port = XXX;
   //实例详情页面中的时序数据存储用户名、密码。
   String username = "XXX";
   String password = "XXXX";
   TSDBConfig config = TSDBConfig.address(connectString, port)
       .basicAuth(username, password)
       // 网络连接池大小，默认为64。
       .httpConnectionPool(64)
       // HTTP 等待时间，单位为秒，默认为90秒。
       .httpConnectTimeout(90)
       // IO 线程数，默认为1。
       .ioThreadCount(1)
       .config();
   ```

3. 构造时间点。

Point（时间点）有多种构造方法，形式比较多样化。下面提供三种构造时间点的方法示例。

> 说明 Point的构建至少需要一个tag键值对。

示例一

构建一个时间点。用单位为秒的时间戳表示时间，指定Point数据的Metric与多个Tag。

```
// 以'秒'为时间戳。
int timestamp = (int)(System.currentTimeMillis()/1000);
// 构造Point。
Point point = Point.metric("test1")
                    .tag("tagk1", "tagv1")
                    .tag("tagk2", "tagv2")
                    .tag("tagk3", "tagv3")
                   .timestamp(timestamp).value(123.456)
                   .build();
```

示例二

构建一个时间点。用单位为毫秒的时间戳表示时间，指定Point数据的Metric，Tag使用Map形式的键值对。

```
// 也可以毫秒为时间戳。
long timestamp = System.currentTimeMillis();
// 使用HashMap表示Tags。
Map<String,String> tagsMap = new HashMap<String,String>();
tagsMap.put("tagk1", "tagv1");
tagsMap.put("tagk2", "tagv2");
// 构造Point。
Point point = Point.metric("test1")
                    .tag(tagsMap)
                   .value(timestamp,123.456)
                   .build();
```

示例三

构建一个时间点。使用*java.util.Date*表示时间。

```
// 使用java.util.Date表示时间。
Point point = Point.metric("test1")
                    .tag("tagk1", "tagv1")
                   .value(new Date,123.456)
                   .build();
```

4. 写入数据。

   SDK有两种写数据的方式：同步阻塞的写数据、异步非阻塞的写数据。

   - 异步非阻塞的写数据

     异步写数据的方式比较简单，只要有Point就可以提交数据。Client会自动帮助您批量地提交数据，不需要您手动打包。如果没有特殊需求，推荐您使用异步非阻塞的写数据的方式。示例代码：

```
// 构建Point。
Point point = Point.metric("test1")
                   .tag("tagk1", "tagv1")
                   .value(timestamp, Math.random())
                   .build();
// 直接提交数据。
tsdb.put(point);
```

当您使用异步写数据的方式时，若需要获取Put接口的响应结果，则需要设置回调接口。示例代码：

```
final AtomicLong num = new AtomicLong();
// 回调对象。
BatchPutCallback cb = new BatchPutCallback() {
    @Override
    public void response(String address, List<Point> input, Result output) {
        long afterNum = num.addAndGet(input.size());
        System.out.println("成功处理" + input.size() + ",已处理" + afterNum);
    }
    @Override
    public void failed(String address, List<Point> input, Exception ex) {
        ex.printStackTrace();
        long afterNum = num.addAndGet(input.size());
        System.out.println("失败处理" + input.size() + ",已处理" + afterNum);
    }
};
HiTSDBConfig config = HiTSDBConfig
                        .address("example.hitsdb.com", 8242)
                        .listenBatchPut(cb)     // 设置回调接口。
                        .config();
tsdb = HiTSDBClientFactory.connect(config);
```

SDK提供了以下三种类型的PutCallback接口来根据需要选择返回的数据内容：

- BatchPutCallback：即调用 `POST /api/put` 。
- BatchPutSummaryCallback：即调用 `POST /api/put?summary=true` 。
- BatchPutDetailsCallback：即调用 `POST /api/put?details=true` 。

> **说明**
>
> - 不要在回调方法中做耗时操作。若有此需要，可以再把操作交给其他工作线程执行。
> - 回调方法 `response()` 与 `failed()` 被调用的时机是不一样的：
>   - `response()` ：指的是写入请求合法且被服务端处理后执行的回调。此时可以根据回调对象种类的不同，获取一些不同的反馈信息。
>   - `failed()` ：指的是写入请求本身存在问题导致请求不合法（例如报文非法，触发限流，鉴权失败等）被服务端拒绝服务时所执行的回调。
>
> 因此，建议在实现PutCallback接口时，不要忘记实现 `failed()` 方法。如果不实现的话，则会执行默认的 `failed()` 方法，其本身是一个空方法。

- 同步阻塞的写数据

假设我们现在需要构建500个时间点提交给时序数据存储，示例代码：

```
List<Point> points = new ArrayList<Point>();
// 构建Point。
for(int i = 0; i<500; i++) {
    long timestamp = System.currentTimeMillis();
    Point point = Point.metric("test1")
                  .tag("tagk1", "tagv1")
                  .value(timestamp, Math.random())
                  .build();
    // 手动打包数据。
    points.add(point);
}
// 手动打包后提交数据。
tsdb.putSync(points)
```

> **说明** 出于写入性能的考虑，同步写的方式一般需要您手动将数据点打包成一批数据，并且建议一批数据包含500～1000个数据点。此外，也建议多个线程并发进行提交。

您可以根据需要选择返回的数据内容：

- 返回空对象，无内容。本质是调用 `POST /api/put` 。

  ```
  Result result = tsdb.putSync(ps);
  ```

- 返回提交概述，包含成功数和返回数。本质是调用 `POST /api/put?summary=true` 。

  ```
  SummaryResult summaryResult = tsdb.putSync(ps,SummaryResult.class);
  ```

- 返回提交概述。包含成功数、返回数和失败原因。本质是调用 `POST /api/put?details=true` 。

  ```
  DetailsResult detailsResult = tsdb.putSync(ps,DetailsResult.class);
  ```

通过SDK使用SQL查询数据

您可以通过SDK使用SQL查询已写入实例内时序数据存储的数据。

1. 获取SDK。

   使用Maven做项目构建工具，在pom.xml文件的<dependencies>标签中添加hitsdb-client依赖。代码如下：

   ```
   <dependency>
       <groupId>com.aliyun</groupId>
       <artifactId>hitsdb-client</artifactId>
       <version>0.2.7</version>
   </dependency>
   ```

   > **说明** SDK版本号必须为0.2.7及以上。

2. 配置客户端。

   客户端的所有配置均由TSDBConfig类进行配置。示例代码如下：

```
//实例详情页面中的时序存储SQL连接地址、端口。
String connectString = "XXX";
int port = XXX;
//实例详情页面中的时序数据存储用户名、密码。
String username = "XXX";
String password = "XXXX";
TSDBConfig config = TSDBConfig.address(connectString, port)
    .basicAuth(username, password)
    // 网络连接池大小，默认为64。
    .httpConnectionPool(64)
    // HTTP 等待时间，单位为秒，默认为90秒。
    .httpConnectTimeout(90)
    // IO 线程数，默认为1。
    .ioThreadCount(1)
    .config();
```

3. 查询数据。

   示例代码如下：

```
TSDB tsdbClient = TSDBClientFactory.connect(config);
//按标签筛选数据。
Map<String, String> tags = new HashMap<>();
String metric = "XXX";
long now = System.currentTimeMillis();
LastPointQuery query = LastPointQuery.builder()
    .timestamp(now)
    .backScan(-1)
    .msResolution(true)
    .sub(LastPointSubQuery.builder(metric, tags).build()).build();
List<LastDataValue> lastDataValues = tsdbClient.queryLast(query);
System.out.println(lastDataValues);
```

> **说明** SDK不能用于将数据写入实例内的时序数据存储。

通过JDBC连接使用TSQL查询数据

您可以通过JDBC连接使用TSQL进行数据的查询操作。

> **说明** 时序数据写入TPS为5千条/秒时，不支持TSQL。

1. 引入Driver依赖。

   运行时要求：

   - Java 1.8 Runtime。
   - 配置实例JBDC访问，获取JDBC URL。

   TSQL的JDBC driver的依赖已经发布到Maven仓库，这里以Maven来管理项目（tsql_jdbc_app）为例，把下面的内容加入*pom.xml*。获得一个包括所有依赖.jar文件，以及应用程序.class文件的JAR包。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/x
sd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.alibaba.tsdb.tsql</groupId>
    <artifactId>tsql_jdbc_app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.apache.drill.exec/drill-jdbc -->
        <dependency>
            <groupId>org.apache.drill.exec</groupId>
            <artifactId>drill-jdbc-all</artifactId>
            <version>1.15.0</version>
            <exclusions>
                <exclusion>
                    <groupId>org.slf4j</groupId>
                    <artifactId>log4j-over-slf4j</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <configuration>
                    <archive>
                        <manifest>
                            <mainClass>com.alibaba.tsdb.tsql.TsqlJdbcSampleApp</mainClas
s>
                        </manifest>
                    </archive>
                    <descriptorRefs>
                        <descriptorRef>jar-with-dependencies</descriptorRef>
                    </descriptorRefs>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id> <!-- this is used for inheritance merges
-->
                        <phase>package</phase> <!-- bind to the packaging phase -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

2. 在Java应用项目下，创建一个*package com.alibaba.tsdb.tsql*文件，并创建一个Java源文件*TsqlJdbcSam*

pleApp。

在以下示例中，根据您的实际情况，修改：

- host：实例详情页面中的时序存储TSQL连接地址。
- port：实例详情页面中的时序存储TSQL端口。
- sql: 您所需要执行的TSQL查询语句。

```
package com.alibaba.tsdb.tsql;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TsqlJdbcSampleApp {
    public static void main(String[] args) throws Exception {
        Connection connection = null;
        Statement stmt = null;
        try {
            // step 1: Register JDBC driver
            Class.forName("org.apache.drill.jdbc.Driver");
            // hostname or address of TSDB instance.
            String host = "ts-uf64t3199j58j8251.tsql.hitsdb.rds.aliyuncs.com";
            // port for TSQL JDBC service
            int port = 3306;
            String jdbcUrl = String.format("jdbc:drill:drillbit=%s:%s", host, port);
            // step 2: Open connection
            System.out.println("Connecting to database @ " + jdbcUrl + "  ...");
            connection = DriverManager.getConnection(jdbcUrl);
            // step 3: Create a statement
            System.out.println("Creating statement ...");
            stmt = connection.createStatement();
            // step 4: Execute a query using the statement.
            String sql = "select hostname, `timestamp`, `value` " +
                "from tsdb.`cpu.usage_system` " +
                "where `timestamp` between '2019-03-01' and '2019-03-01 00:05:00'";
            ResultSet rs = stmt.executeQuery(sql);
            // step 5: Extract data from ResultSet.
            int row = 0;
            System.out.println("hostname\ttimestamp\tvalue");
            System.out.println("-----------------------------------------------------");
            while (rs.next()) {
                row++;
                System.out.println(rs.getString("hostname") + "\t" + rs.getTimestamp("ti
mestamp") + "\t" +rs.getDouble("value"));
            }
            System.out.println("-----------------------------------------------------");
            System.out.println( row + "rows returned");
        } catch(SQLException se){
            //Handle errors for JDBC
            se.printStackTrace();
        }catch(Exception e){
            //Handle errors for Class.forName
            e.printStackTrace();
        }finally{
```

```
            //finally block used to close resources
            try{
                if(stmt!=null)
                    stmt.close();
            }catch(SQLException se2){
            }// nothing we can do
            try{
                if(connection!=null)
                    connection.close();
            }catch(SQLException se){
                se.printStackTrace();
            }//end finally try
        }//end try
        System.out.println("Goodbye!");
    }
}
```

3. 编译与执行。

   在项目根目录下，执行下面的Maven命令：

   ```
   maven clean install
   ```

   执行完成后，您将在项目目录*/targets*下获得可执行程序*tsql_jdbc_app-1.0-SNAPSHOT-jar-with-dependencies.jar*。

   执行该应用程序：

   ```
   java -jar target/tsql_jdbc_app-1.0-SNAPSHOT-jar-with-dependencies.jar
   ```

JDBC协议使用限制

TSQL的JDBC协议存在功能限制。在使用TSQL JDBC协议前请对照检查，具体限制和说明如下：

- TSQL目前仅支持时序数据查询和时序元数据查询，不支持数据写入，修改和删除。
- 时序数据存储没有事务（Transaction）的支持。

具体JDBC接口的限制说明如下：

接口	方法	TSDB JDBC支持情况
Connection	setAutoCommit(boolean)	仅允许true作为传入参数。
Connection	getAutoCommit()	返回true。
Connection	commit()	调用会引发异常：SQLFeatureNotSupportedException。
Connection	rollback()	调用会引发异常：SQLFeatureNotSupportedException。
Connection	setTransactionIsolation(int level)	仅允许TRANSACTION_NONE。
Connection	getTransactionIsolation()	仅允许TRANSACTION_NONE。

接口	方法	TSDB JDBC支持情况
Connection	setSavePoint()	调用会引发异常SQLFeatureNotSupportedException。
Connection	setSavePoint(String name)	调用会引发异常SQLFeatureNotSupportedException。
Connection	rollback(Savepoint savepoint)	调用会引发异常SQLFeatureNotSupportedException。
Connection	releaseSavePoint(Savepoint savepoint)	调用会引发异常SQLFeatureNotSupportedException。
Connection	setNetworkTimeout()	调用会引发异常SQLFeatureNotSupportedException。
Connection	getNetworkTimeout()	调用会引发异常SQLFeatureNotSupportedException。

8.场景联动

8.1. 什么是场景联动

场景联动是规则引擎中，一种开发自动化业务逻辑的可视化编程方式，您可以通过可视化的方式定义设备之间联动规则，并部署规则至物联网平台云端或者边缘端。

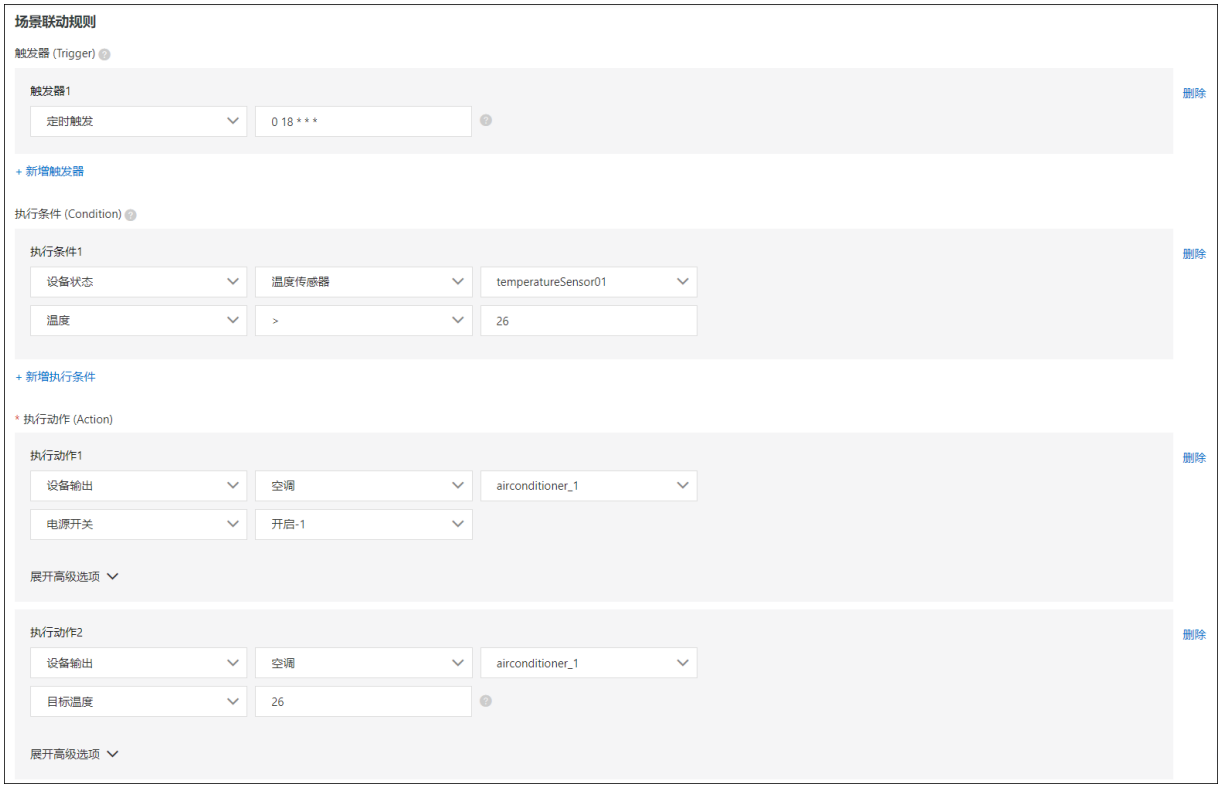
您需在物联网平台控制台，**规则引擎 > 场景联动**页面中创建场景联动规则。每个场景联动规则由触发器（Trigger）、执行条件（Condition）、执行动作（Action）三个部分组成。这种规则模型称为TCA模型。

> **注意** 仅华东2（上海）地域下，企业版实例和新版公共实例，支持场景联动功能。
>
> 目前，场景联动已升级为事件响应增值服务。
>
> - 对于已配置并使用场景联动的用户，不受影响。
> - 对于需使用场景联动的新用户，请使用事件响应增值服务。详细内容，请参见事件响应服务。
>
> 实例的详细说明，请参见实例概述。

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

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

参数设置如下图。



参数说明如下：

参数	描述	并列关系
触发器	定时为每天18:00触发该规则。时间的cron表达式写作方法，请参见CRONTAB网页。	或（||）
执行条件	获取温度传感器上报的数据，若室内温度高于26摄氏度，则执行动作。	与（&&）
执行动作	空调开关设置为打开；空调目标温度设置为26摄氏度。	与（&&）

创建场景联动规则的更多设置说明，请参见云端场景联动。

8.2. 云端场景联动

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

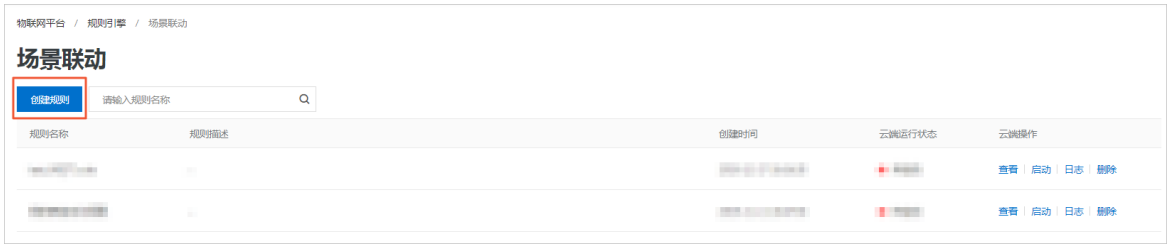
前提条件

已完成边缘实例的创建。具体操作步骤请参见环境搭建。

> **注意** 仅华东2（上海）地域下，企业版实例和新版公共实例，支持场景联动功能。
>
> 目前，场景联动已升级为事件响应增值服务。
>
> - 对于已配置并使用场景联动的用户，不受影响。
> - 对于需使用场景联动的新用户，请使用事件响应增值服务。详细内容，请参见事件响应服务。
>
> 实例的详细说明，请参见实例概述。

创建场景联动

1. 登录物联网平台控制台，选择华东2（上海）地域，单击对应实例。
2. 在左侧导航栏选择**规则引擎 > 场景联动**。
3. 单击**创建规则**。



4. 设置参数，然后单击**确认**。

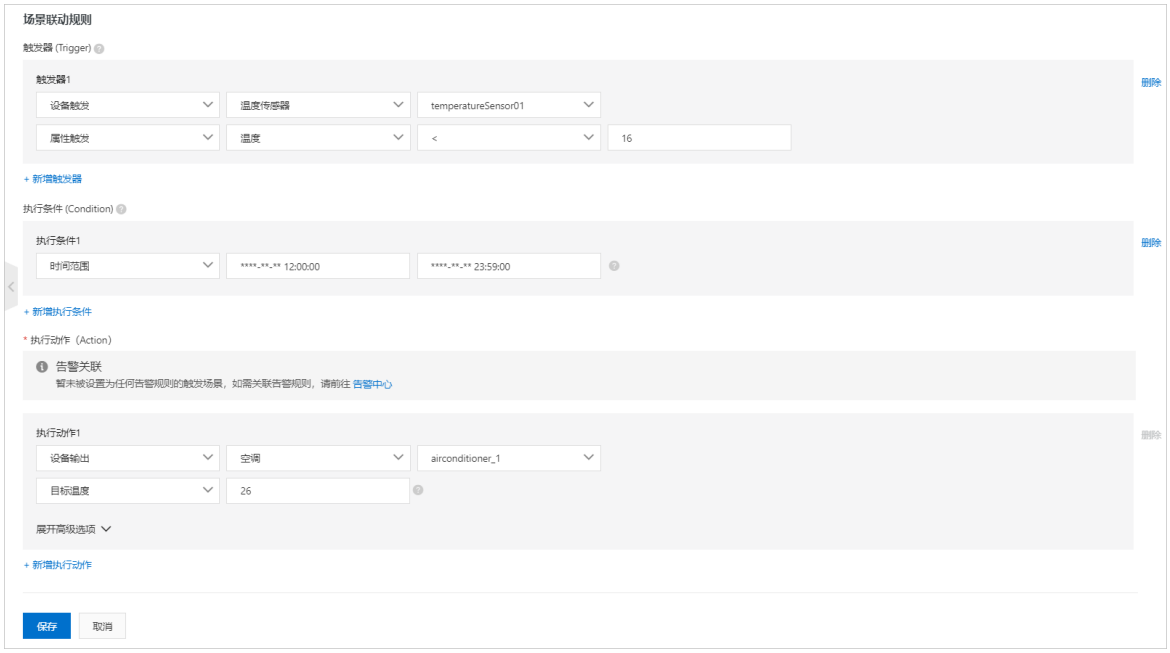
参数	描述
规则名称	设置具体规则的名称。支持中文、英文字母、数字、下划线（_）和短划线（-），长度限制为1~30个字符，中文字算两位字符。
规则描述	为规则添加描述，可以为空。

5. 完成场景联动的创建后，在弹出的对话框中单击**前往编辑**，管理配置场景联动。

   您也可以在场景联动名称右侧单击**查看**，管理配置场景联动。

   以空调设备自动化为例：在12:00至23:59之间，当温度传感器上报的室内温度低于16摄氏度时，空调设备开始工作，且设置温度为26摄氏度。

   具体参数设置，请见下图。



   单击页面右上角**编辑**，可更改场景联动规则名称，其余参数说明请见下表。

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

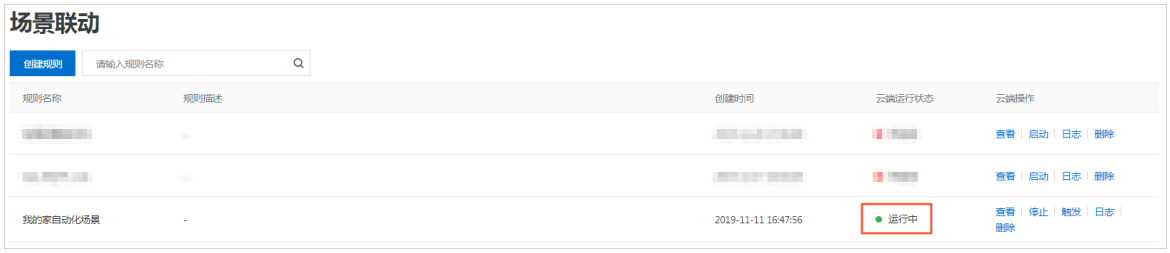
参数	描述
执行条件	执行条件集。只有满足执行条件的数据，才能触发执行动作。可设置为**设备状态**或**时间范围**。可以为一个规则创建多个执行条件，执行条件之间是和（and）关系。 ○ 设置为**设备状态**，则需选择已创建的产品、该产品下的某个设备、和设备功能中的某个属性或事件。 ○ 设置为**时间范围**，则需设置起始时间和结束时间，格式为 `yyyy-mm-dd hh24:mi:ss` 。 上图示例中，设置为**时间范围**：时间在12:00至23:59之间，则触发执行动作。
执行动作	需执行的动作。您可以设置多个动作。某一动作执行失败时，不影响其他动作。 ○ 设置为**设备输出**，则需选择已创建的产品、该产品下的某个设备、和设备功能中的某个属性或服务（只有可写的属性或服务才能被设为执行动作）。当触发器和执行条件均被满足时，执行已定义的设备属性或服务的相关动作。 ○ 设置为**规则输出**，则需嵌套另外一个规则，即调用其他规则。被调用规则中的触发器会被跳过，直接进行执行条件检查。若执行条件满足，则执行该规则中定义的执行动作。 例如，被调用规则为A，那么规则A中的触发器被跳过，直接检查其执行条件内容，若满足规则A的全部执行条件（多个执行条件之间为&&关系）内容，则会执行规则A中的执行动作。 ○ 设置为**函数输出**，则需要选择一个已创建的函数。当触发器和执行条件均被满足时，运行已选定的函数。创建函数相关内容，请参见使用控制台创建函数。 ○ 设置为**告警输出**，则需要将该场景联动规则关联到告警中心。当触发器和执行条件均被满足时，触发告警。单击**告警中心**，前往告警中心设置告警规则。详细操作，请参见告警中心。 上图示例中，设置为**设备输出**：指定的空调设备，执行设置温度为26摄氏度的动作。
延时执行	展开高级选项后的参数。设置延时时间后，执行动作会延迟执行。取值范围为0~86400秒。

运行场景联动

场景联动创建成功后，您可在**场景联动**页面中，启动此场景联动。

启动场景联动操作：

1. 在物联网平台控制台对应实例页面，左侧导航栏选择**规则引擎 > 场景联动**。
2. 找到要启动的场景联动，单击右侧操作栏中的**启动**，使规则状态为**运行中**。



启动场景联动后：

- 若场景联动在云端运行，则需要为场景联动中的设备配置消息路由，使得设备的属性和事件能够发送到IoT Hub（云端）。消息路由的配置，请参见设置消息路由。
- 若场景联动在边缘端运行，则需要先**停止**其在云端的运行，再分配场景联动到边缘实例中，分配方法，请参见本文下方场景联动其他操作。

查看日志

您可以查看该场景联动的日志，并且可在详情中查看运行结果。

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

1. 在物联网平台控制台对应实例页面，左侧导航栏选择**规则引擎 > 场景联动**。
2. 找到要查看日志的场景联动，单击右侧操作栏中的**日志**。
3. 单击**详情**，查看该条日志的详情信息。

> **说明** 若有执行状态为失败的日志，可单击对应操作栏中的**详情**，查看场景联动执行失败的详细信息。

场景联动其他操作

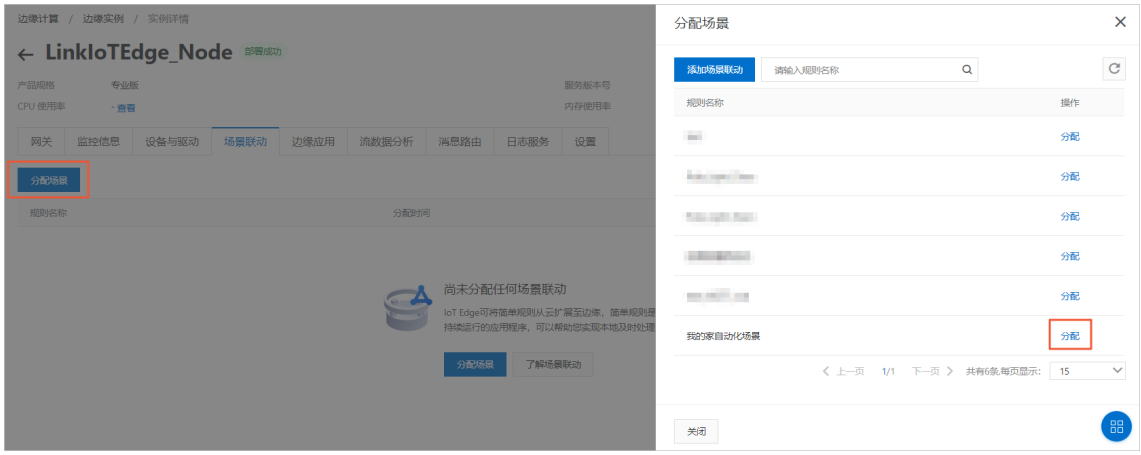
- 删除场景联动：
  i. 在**场景联动**页签中，找到需要删除的场景联动规则名称。
  ii. 单击规则名称右侧的**删除**，在弹出的对话框中单击**确认**，删除该条场景联动规则。
- 触发场景规则：

  在**启动**场景联动规则后，方可显示**触发**操作按钮。
  i. 在**场景联动**页签中，找到已启动的需要触发的场景联动规则名称。
  ii. 单击规则名称右侧的**触发**，表示手动触发规则一次，即忽略已管理配置的触发器，直接执行所有执行条件和执行动作。
- 在边缘实例中运行场景联动：

  您需要根据如下步骤，部署场景联动到边缘实例。

  > **注意** 请确保已停止场景联动在云端的运行。

  i. 登录边缘计算控制台，在左侧导航栏选择**边缘实例**，单击“前提条件”中已创建的边缘实例右侧的**查看**。
  ii. 在**实例详情**页面，选择**场景联动**，单击**分配场景**。
  iii. 在**分配场景**对话框，单击待分配场景联动规则名称后的**分配**，然后单击**关闭**。



iv. 分配场景联动后，重新部署边缘实例。

9.MQTT同步通信（RRPC）

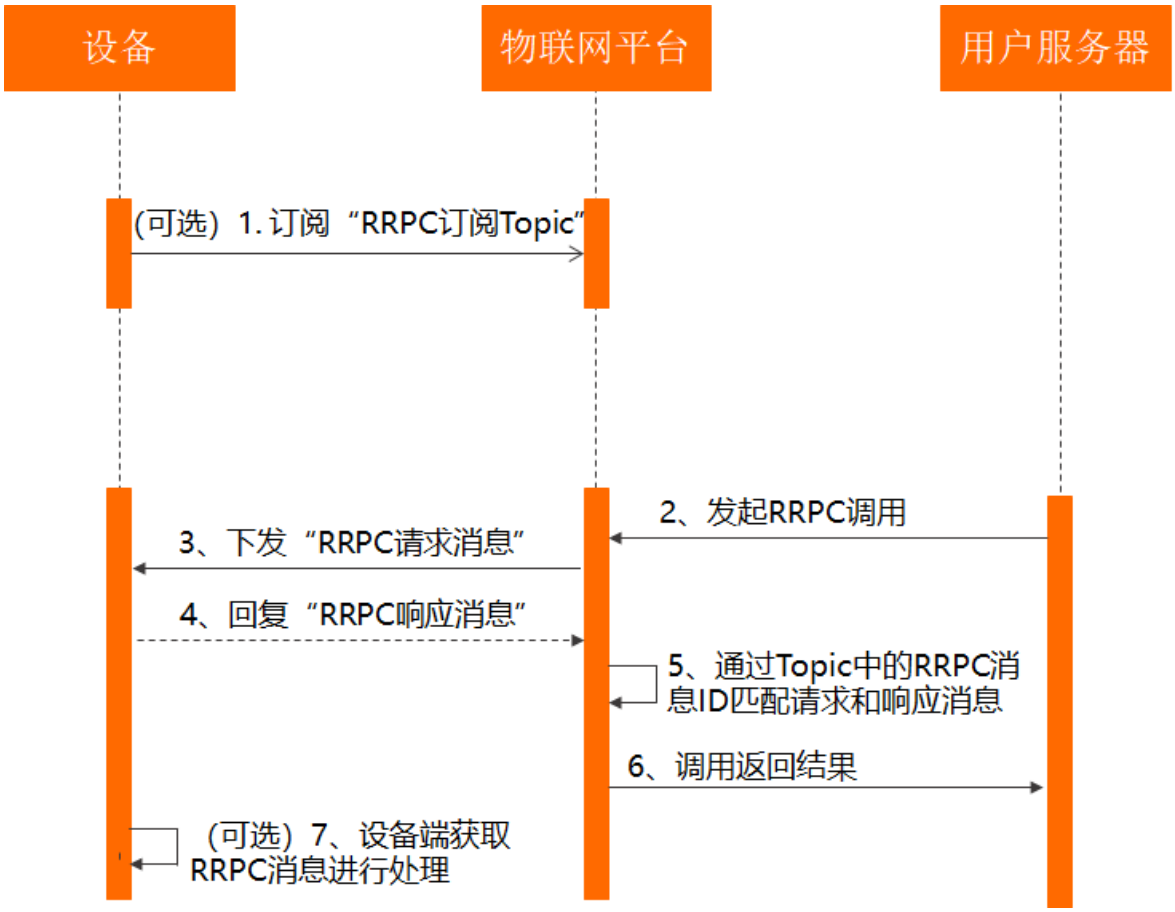
9.1. 什么是RRPC

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

名词解释

名词	说明
RRPC	是Revert-RPC的简称。RPC（Remote Procedure Call）是指采用客户机/服务器模式，您不需要了解底层技术协议，即可远程请求服务。RRPC则可以实现由服务端请求设备端，并能够使设备端响应的功能。
RRPC订阅Topic	设备端订阅RRPC消息时传递的Topic，含有通配符。
RRPC请求消息	物联网平台下发给设备端的消息。
RRPC响应消息	设备端回复给物联网平台的消息。
RRPC消息ID	物联网平台为每次RRPC调用，生成的唯一消息ID，可标识不同RRPC消息。

RRPC原理



具体流程如下：

1. （可选）设备端订阅RRPC相关Topic。

   物联网平台根据MQTT协议，仅定制了RRPC消息的请求和响应机制。设备端若要获取物联网平台下发的RRPC消息数据，并做应对处理，需要在设备端先订阅Topic，然后开发数据处理逻辑。

   - 设备可通过发送SUB指令订阅指定Topic，实现从物联网平台获取消息。开发方法，请参见设备接入Link SDK文档中订阅Topic的代码示例。
   - 在设备端开发处理RRPC消息数据的方法，请参见RRPC功能开发示例。
2. 用户服务器调用物联网平台的RRpc接口。
3. 物联网平台收到服务器端RRPC调用请求，向设备下发一条RRPC请求消息。消息体为用户传入的数据，Topic为物联网平台定义的Topic，且携带RRPC消息ID。
4. 设备收到下行消息后，按照指定Topic格式（包含之前物联网平台下发的唯一RRPC消息ID），回复一条RRPC响应消息给物联网平台。
5. 物联网平台提取RRPC消息ID，和之前的RRPC请求消息匹配。
6. 物联网平台将响应结果返回给用户服务器。
7. （可选）设备对已订阅Topic的消息进行处理。

> **说明** 调用RRpc接口时，可能出现以下情况：
>
> - 设备不在线，物联网平台会向用户服务器，返回设备离线的错误信息。
> - 设备未在超时时间（8秒）内回复RRPC响应消息，物联网平台会向用户服务器，返回超时的错误信息。

使用Topic

不同Topic格式使用方法不同：

- 基础通信Topic使用方法，请参见调用RRPC通信相关Topic。
- 自定义Topic使用方法，请参见调用自定义Topic。

RRPC调用实践示例，请参见远程控制树莓派服务器。

9.2. 调用RRPC通信相关Topic

RRPC支持调用RRPC通信相关Topic与物联网平台通信。本文介绍RRPC通信相关Topic和接入方法。

RRPC通信相关Topic

RRPC通信相关Topic格式如下：

Topic	格式	说明
RRPC订阅	`/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/+`	订阅物联网平台下发的RRPC请求消息。
RRPC请求消息	`/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/${messageId}`	物联网平台下发的RRPC请求消息。

Topic	格式	说明
RRPC响应消息	`/sys/${YourProductKey}/${YourDeviceName}/rrpc/response/${messageId}`	设备上行的RRPC响应消息。

Topic格式中：

- *${YourProductKey}*：您设备所属产品的**ProductKey**。
- *${YourDeviceName}*：您设备的名称。
- *${messageId}*：服务端调用物联网平台的RRpc接口向设备下发消息时，物联网平台生成的唯一的RRPC消息ID，可用于区分不同的RRPC消息。

RRPC接入

1. 物联网平台发送RRPC消息。

   服务端调用物联网平台的RRpc接口向设备发送消息。接口调用方法，请参见RRpc。

   以使用Java Link SDK为例，调用方式如下：

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

   > **说明** 请登录OpenAPI开发者门户，在线调用RRpc接口，查看物联网平台中多种语言的云端SDK调用示例。

2. 设备端返回RRPC响应的Topic。

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

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

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

示例：远程控制树莓派服务器

9.3. 调用自定义Topic

RRPC支持调用自定义Topic与物联网平台通信，且相关Topic中包含了您自定义的完整Topic。本文介绍RRPC自定义Topic和接入方法。

前提条件

支持使用以下Link SDK开发的设备，通过自定义Topic与物联网平台通信：

- C Link SDK
- Android Link SDK

- Node.js Link SDK
- Python Link SDK

自定义Topic

Topic	格式	说明
RRPC订阅	`/ext/rrpc/+/${topic}`	您需订阅物联网平台下发的RRPC请求消息中的自定义Topic。
RRPC请求消息	`/ext/rrpc/${messageId}/${topic}`	物联网平台下发的RRPC请求消息。
RRPC响应消息	`/ext/rrpc/${messageId}/${topic}`	设备上行的RRPC响应消息。

以上Topic格式中：

- *${topic}*：您在物联网平台的完整自定义Topic，即为 `/${productKey}/${deviceName}/user/${TopicShortName}` 。

  其中 `${TopicShortName}` 为自定义的Topic类，即自定义Topic的后缀。更多信息，请参见自定义Topic。
- *${messageId}*：服务端调用物联网平台的RRpc接口向设备下发消息时，物联网平台生成的唯一的RRPC消息ID，可用于区分不同的RRPC消息。

例如设备device1的自定义Topic `a18wP***/device1/user/get` ，发起RRPC消息的Topic为 `/ext/rrpc/121307410***/a18wP***/device1/user/get` 。

RRPC接入

1. 从物联网平台发送RRPC消息。

   服务端调用物联网平台API的RRpc接口向设备发送消息。更多信息，请参见RRpc。

   以使用物联网平台云端Java SDK为例，调用方式如下。

   使用自定义Topic格式时，您需要确保使用的物联网平台云端Java SDK（aliyun-java-sdk-iot）为6.0.0及以上版本。

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

   调用RRpc接口的示例：

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

> **说明** 请登录OpenAPI开发者门户，在线调用RRpc接口，查看物联网平台的多种语言云端SDK调用示例。

2. 设备端接入。
   - 对于使用C Link SDK、Android Link SDK、Python Link SDK的设备，无需做任何特殊操作。
   - 对于使用Node.js Link SDK的设备，需要在开发设备前，将Link SDK下载到本地，修改*src*文件夹中的*model.js*文件，在genConnectPrarms函数的clientId中添加`ext=1`，再使用该Link SDK进行设备开发。

     原clientId为：

     ```
     clientId:`${this.clientId}|securemode=${this.securemode },signmethod=hmac${this.signAlgorithm},timestamp=${this.timestamp},${extra}`,
     ```

     添加`ext=1`后，clientId为：

     ```
     clientId:`${this.clientId}|securemode=${this.securemode },signmethod=hmac${this.signAlgorithm},timestamp=${this.timestamp},${extra},ext=1`,
     ```

3. 设备端返回RRPC响应的Topic。

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

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

   关于通过Python语言，调用设备端Link SDK响应RRPC的方法，请参见RRPC能力。

10.广播通信

物联网平台支持广播通信，即向指定产品下的全量设备（设备无需订阅广播Topic），或订阅了指定Topic的所有设备发送消息。设备在线，即可收到服务器发送的广播消息。本文以在公共实例下向全量在线设备广播消息为例，介绍广播通信的具体配置流程。

背景信息

- 向全量在线设备广播消息

  业务服务器调用PubBroadcast接口，传入指定产品的ProductKey和MessageContent消息内容，产品的全量在线设备可通过广播Topic `/sys/${productKey}/${deviceName}/broadcast/request/${MessageId}`，收到MessageContent消息内容。

  广播Topic中的MessageId是物联网平台生成的消息ID，成功发送消息后，将作为PubBroadcast接口的返回数据返回业务服务器。

  例如，厂家有多个智能门锁接入物联网平台，可通过业务服务器向全部在线设备发送一条相同的指令，使某个密码失效。

- 向订阅了指定Topic的所有设备广播消息

  设备端订阅相同的广播Topic，业务服务器调用PubBroadcast接口，传入指定产品的ProductKey、MessageContent消息内容和要接收广播消息的Topic全称（格式为：`/broadcast/${productKey}/自定义字段`），已订阅广播Topic的在线设备，收到MessageContent消息内容。

  > 注意
  > - 广播Topic是在设备开发时编码定义的，无需在物联网平台控制台创建。
  > - 一个广播Topic最多可被1,000个设备订阅。如果您的设备超过数量限制，您可以对设备进行分组。例如，如果您有5,000个设备，您可以将设备按每组1,000个，而分成5组。您需要分5次调用广播Topic，自定义字段分别设置为group1、group2、group3、group4、group5，然后让每组设备分别订阅各自分组的广播Topic。

  广播接口调用的更多详细说明，请参见PubBroadcast。

使用限制

- 广播消息仅推送给产品下当前在线的设备。
- 指定在线设备广播时，需指定Topic订阅广播，广播接口最大调用频次：1次/秒。

- 全量在线设备广播时，无需订阅广播Topic，广播接口最大调用频次：1次/分钟。
- 广播消息体报文最大为64 KB。

> **注意** 广播消息不会被实例的消息上下行TPS规格限流。例如，消息上下行TPS为100条/秒，当前在线设备为500个时，每次调用广播接口，消息不会只发送给100个在线设备，而是可以发送给500个在线设备。

准备开发环境

本示例中，设备端和云端均使用Java语言的SDK，需先准备Java开发环境。您可从Java 官方网站下载，并安装Java最新开发环境。

添加Maven项目依赖

新建项目，在*pom.xml*中，添加以下Maven依赖。

```
<dependencies>
  <dependency>
     <groupId>com.aliyun.alink.linksdk</groupId>
     <artifactId>iot-linkkit-java</artifactId>
     <version>1.2.0.1</version>
     <scope>compile</scope>
  </dependency>
  <dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-core</artifactId>
      <version>3.7.1</version>
  </dependency>
  <dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-iot</artifactId>
      <version>7.6.0</version>
  </dependency>
  <dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.2</version>
  </dependency>
</dependencies>
```

创建产品和设备

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

> **注意** 目前华东2（上海）、华北2（北京）、华南1（深圳）地域开通了企业版实例服务。其他地域，请跳过此步骤。

3. 在左侧导航栏，单击**设备管理 > 产品**。
4. 单击**创建产品**，创建**智能门锁**产品。具体操作，请参见创建产品。
5. 在左侧导航栏，单击**设备管理 > 设备**，然后在**智能门锁**产品下，创建三个设备。具体操作，请参见批量创建设备。

配置设备端SDK

- 配置设备端接入物联网平台。
  - 配置设备认证信息。

```
final String productKey = "<yourProductKey>";
final String deviceName = "<yourDeviceName>";
final String deviceSecret = "<yourDeviceSecret>";
final String region = "<yourRegionID>";
```

productKey、deviceName和deviceSecret是设备证书信息，请在物联网平台控制台，对应实例下，选择**设备管理 > 设备**，单击设备对应的**查看**，进入**设备详情**页获取。

region是设备所属地域。region的表达方法，请参见地域和可用区中的Region ID。

- 设置初始化连接参数，包括MQTT配置、设备信息和初始状态。

```
LinkKitInitParams params = new LinkKitInitParams();
//LinkKit底层是MQTT协议，设置MQTT配置。
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = productKey;
config.deviceName = deviceName;
config.deviceSecret = deviceSecret;
config.channelHost = productKey + ".iot-as-mqtt." + region + ".aliyuncs.com:1883";
//设备信息。
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
//报备的设备初始状态。
Map<String, ValueWrapper> propertyValues = new HashMap<String, ValueWrapper>();
params.mqttClientConfig = config;
params.deviceInfo = deviceInfo;
params.propertyValues = propertyValues;
```

  channelHost 是MQTT接入地址，公共实例和企业版实例中接入地址的获取方法，请参见查看实例终端节点。

- 初始化连接。

```
//连接并设置连接成功以后的回调函数。
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
     @Override
     public void onError(AError aError) {
         System.out.println("Init error:" + aError);
     }
     //初始化成功以后的回调。
     @Override
     public void onInitDone(InitResult initResult) {
         System.out.println("Init done:" + initResult);
     }
 });
```

- 在回调函数onInitDone中，通过前缀来识别广播Topic，广播Topic的前缀为：`/sys/${productKey}/${deviceName}/broadcast/request/`。

```
public void onInitDone(InitResult initResult) {
                //设置下行消息到来时的回调函数。
                IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
                    //此处定义收到下行消息以后的回调函数。
                    @Override
                    public void onNotify(String connectId, String topic, AMessage aMessag
e) {
                        //过滤得到广播消息。
                        if(topic.startsWith(broadcastTopic)){
                            System.out.println(
                                    "received broadcast message from topic=" + topic + ",
\npayload=" + new String((byte[])aMessage.getData()));
                        }
                    }
                    @Override
                    public boolean shouldHandle(String s, String s1) {
                        return false;
                    }
                    @Override
                    public void onConnectStateChange(String s, ConnectState connectState)
{
                    }
                };
                LinkKit.getInstance().registerOnNotifyListener(notifyListener);
            }
```

配置服务端SDK

配置云端Java SDK发送广播消息。

- 配置身份认证信息。

```
 String regionId = "<yourRegionID>";
 String accessKey = "<yourAccessKey>";
 String accessSecret = "<yourAccessSecret>";
 final String productKey = "<yourProductKey>";
```

- 配置服务端调用云端API PubBroadcast广播消息。

```
//设置client的参数。
DefaultProfile profile = DefaultProfile.getProfile(regionId, accessKey, accessSecret);
IAcsClient client = new DefaultAcsClient(profile);
PubBroadcastRequest request = new PubBroadcastRequest();
//设置广播消息的产品productKey。
request.setProductKey(productKey);
//设置消息的内容，一定要用base64编码，否则乱码。
request.setMessageContent(Base64.encode("{\"pwd\":\"2892nd6Y\"}"));
//设置实例ID，获取ID方法，请参见如何获取实例ID。
request.setIotInstanceId("iot-cn-***");
```

- 服务端发送广播消息。

```
try {
    PubBroadcastResponse response = client.getAcsResponse(request);
System.out.println("broadcast pub success: broadcastId =" + response.getMessageId());
} catch (Exception e) {
    System.out.println(e);
}
```

验证操作

先运行各设备的Link SDK代码，使设备上线，然后运行云端SDK代码，调用接口PubBroadcast向设备广播消息。

云端SDK中，向设备端广播的消息内容为：`"{\"pwd\":\"2892nd6Y\"}"`。

设备端本地日志都将显示收到广播消息：`{\"pwd\":\"2892nd6Y\"}`。

门锁1：

门锁2：

门锁3：

附录：Demo

查看以下完整的配置代码Demo，其中包含物联网平台云端SDK和设备端SDK示例。

- 向全量在线设备广播消息：下载PubBroadcast Demo。
- 向订阅了指定Topic的所有设备广播消息：下载Broadcast Demo。