阿里云
物联网平台

用户指南

文档版本：20190219

法律声明

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

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

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 产品与设备

本章节将介绍如何使用控制台创建、管理产品与设备。

1.1 创建产品(基础版)

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

背景信息

物联网平台有两个规格，基础版和高级版。本文档介绍如何在控制台上创建基础版产品。

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 产品，单击创建产品。
3. 选择版本为基础版，单击下一步。

4. 完成要创建的产品参数设置，单击完成。

页面参数设置如下：

参数	描述
产品名称	为产品命名。产品名称在账号内具有唯一性。例如，可以填写为产品型号。支持中文、英文字母、数字和下划线，长度限制4~30，一个中文汉字算2位。
节点类型	设备或网关。 · 设备：指不能挂载子设备的设备。这种设备可以直连物联网平台，也可以作为网关的子设备连接物联网平台。 · 网关：指可以挂载子设备的直连设备。网关具有子设备管理模块，维持子设备的拓扑关系，并且可以将拓扑关系同步到云端。 网关与子设备的关系，请参见网关与子设备。

参数	描述
使用 ID² 认证	该产品下的设备是否使用ID²认证。 ID²认证提供设备与物联网平台的双向身份认证能力，通过建立轻量化的安全链路（iTLS）来保障数据的安全性。 说明: · 仅在产品创建时能够启用ID²认证。并且，产品创建成功后，不能更改是否使用ID²认证的状态。 · 选择使用ID²认证，需购买ID²服务。请参见*ID²*设备身份认证用户手册。
产品描述	输入文字，用以描述产品。字数限制为100。

预期结果

产品创建成功后，页面自动跳转回产品列表页面。您可以查看或编辑产品信息。

后续操作

产品创建成功后，您需要创建产品对应的设备。具体请参考单个创建设备或批量创建设备。

1.2 创建产品(高级版)

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

背景信息

物联网平台有两个规格，基础版和高级版。本文档介绍如何在控制台上创建高级版产品。

操作步骤

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

3. 选择版本为高级版，单击下一步。

4. 按照页面提示填写信息，然后单击完成。

页面参数设置如下：

参数	描述
产品名称	为产品命名。产品名称在账号内具有唯一性。例如，可以填写为产品型号。支持中文、英文字母、数字和下划线，长度限制4~30，一个中文汉字算2位。
所属分类	选择品类，为该产品定义物模型。 可选择为： · 自定义品类：需根据实际需要，定义产品功能。 · 任一既有功能模板。 选择任一物联网平台预定义的品类，快速完成产品的功能定义。选择产品模板后，您可以在该模板基础上，编辑、修改、新增功能。 阿里云物联网平台提供多种品类，并为对应产品预定义了相关功能。如您选择智能城市 > 能源管理 > 电表设备类型模板中，已预定义用电量、电压、电流、总累积量等电表标准功能。
节点类型	· 设备：指不能挂载子设备的设备。这种设备可以直连物联网平台，也可以作为网关的子设备连接物联网平台 · 网关：指可以挂载子设备的直连设备。网关具有子设备管理模块，可以维持子设备的拓扑关系，并且将拓扑关系同步到云端。 网关与子设备的关系，请参见网关与子设备。
是否接入网关 说明： 当节点类型为设备时出现的参数。	该产品是否会接入网关产品，成为网关产品的子设备。 · 是：接入网关。需在连网与数据下，选择接入网关协议。 · 否：不接入网关。需在连网与数据下，选择连网方式。
接入网关协议 说明： 当是否接入网关选择为是时出现的参数。	接入网关协议，即该产品作为子设备时与网关的通讯协议类型。 · 自定义：表示子设备和网关之间是其它标准或私有协议。 · Modbus：表示子设备和网关之间的通讯协议是 Modbus。 · OPC UA：表示子设备和网关之间的通讯协议是 OPC UA。 · ZigBee：表示子设备和网关之间的通讯协议是 ZigBee。 · BLE：表示子设备和网关之间的通讯协议是 BLE。
连网方式 说明： 当是否接入网关选择为否时出现的参数。	为设备选择连网方式： · WiFi · 蜂窝（2G/3G/4G） · 以太网 · 其他

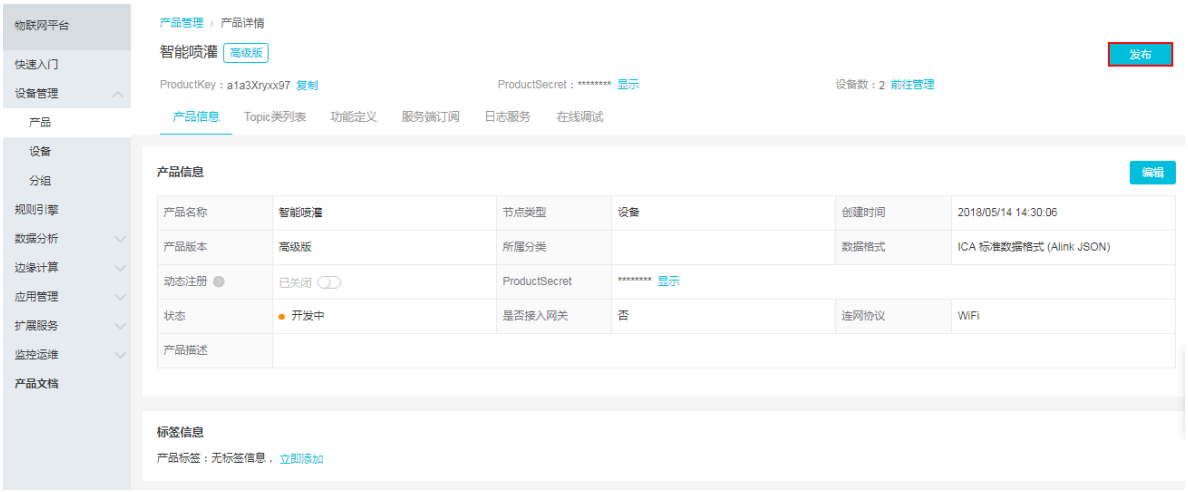
参数	描述
数据格式	设备上下行的数据格式，可选择ICA 标准数据格式（Alink JSON）或透传/自定义。 · ICA 标准数据格式（Alink JSON）：是物联网平台高级版为开发者提供的设备与云端的数据交换协议，采用 JSON 格式。 · 透传/自定义：如果您希望使用自定义的串口数据格式，可以选择透传/自定义。此时，您需将自定义格式的数据通过数据解析转换为 Alink JSON 格式，才能与云端进行通信。
使用ID²认证	该产品下的设备是否使用ID²认证。 ID²认证提供设备与物联网平台的双向身份认证能力，通过建立轻量化的安全链路（iTLS）来保障数据的安全性。 说明: · 仅在产品创建时能够启用ID²认证。并且，产品创建成功后，不能更改是否使用ID²认证的状态。 · 选择使用ID²认证，需购买ID²服务。请参见ID²设备身份认证用户手册。
产品描述	可输入文字，用来描述产品信息。字数限制为100。

产品创建成功后，页面自动跳转回产品列表页面。

后续操作

1. 在产品列表中，单击该产品的查看按钮，设置消息通信、物模型（功能定义）、服务端订阅等功能。
2. 参考设备开发指南手册完成设备开发。

3. 在该产品的详情页中，单击发布按钮，发布产品。



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

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

已发布的产品支持撤销发布。

1.3 创建设备

1.3.1 批量创建设备

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

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，单击批量添加。
3. 选择一个已创建的产品。选择后，新创建的设备将继承该产品的功能和特性。

4. 选择设备名称的添加方式。两种添加方式：

   · 自动生成：您无需为要创建的设备指定名称，您只需填写设备数量，系统将为每个设备自动生成DeviceName。

   · 批量上传：您需为每个要创建的设备指定名称。选择为批量上传后，您可以单击页面上蓝色字体的下载.csv模板下载模板。在模板表格中输入设备名称并保存，然后单击上传文件，将已填好设备名称的表格上传至控制台。

   > 说明：
   > 上传的文件需符合以下要求：
   > - 设备名称长度为4-32个字符，可包含英文字母、数字和特殊字符（连字符、下划线、@符号、点号和英文冒号）。
   > - 列表中的设备名称不可重复，且不可与该产品下已有设备的名称重复。
   > - 一个文件中最多可包含1,000个名称。
   > - 文件大小不超过2 MB。

5. 单击确认，完成创建批量设备。

6. 单击下载设备证书，下载本批次设备的设备证书。

预期结果

产品创建完成后，您可以进入设备管理下的批次管理界面，

· 单击查看详情查看对应批次创建设备的详细信息。

· 单击下载CSV下载该批次设备的证书，方便产线统一烧录。

1.3.2 单个创建设备

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

操作步骤

1. 登录物联网平台控制台。
2. 左侧导航栏选择设备管理 > 设备，单击添加设备。
3. 选择一个已创建的产品。选择后，新创建的设备将继承该产品定义好的功能和特性。
4. （可选）填入DeviceName。如果不填，系统将自动生成一个DeviceName，用以标识设备。

> **说明:**
> DeviceName（即设备名称）需在产品内具有唯一性。可作为设备的唯一标识符，用于与 IoT Hub 进行通信。

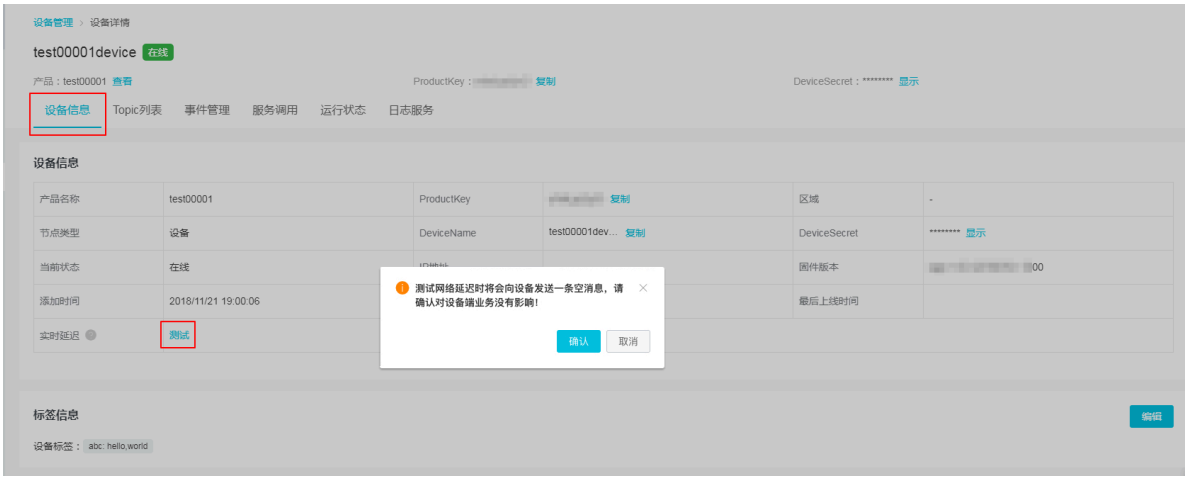
5. 单击确认。完成设备创建。

   设备创建完成后，将自动弹出查看设备证书弹框。您可以查看、复制设备证书信息。设备证书又名设备三元组，由设备 ProductKey、DeviceName、和 DeviceSecret组成，是设备与物联网平台进行通信的重要身份认证，建议您妥善保管。

   - ProductKey：物联网平台为您创建的产品颁发的全局唯一标识符。
   - DeviceName：设备在产品内的唯一标识符，用于设备认证和通信。
   - DeviceSecret：物联网平台为设备颁发的设备秘钥，用于认证加密，需与DeviceName成对使用。

6. 单击已创建设备对应操作栏中的查看，进入设备详情页面，查看设备详情。

   在设备信息页签下，单击实时延迟后的测试，查看您设备的网络延迟情况。



1.4 物模型

1.4.1 概述

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

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

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

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

物模型使用流程

1. 使用控制台，新增物模型，或导入物模型。
2. 设备端参考*Link Kit SDK*文档，进行物模型开发。
3. 开发完成后，设备端可以上报属性和事件；云端可以设置设备属性，调用设备服务。

1.4.2 新增物模型

本文讲述如何通过控制台操作，定义物模型。

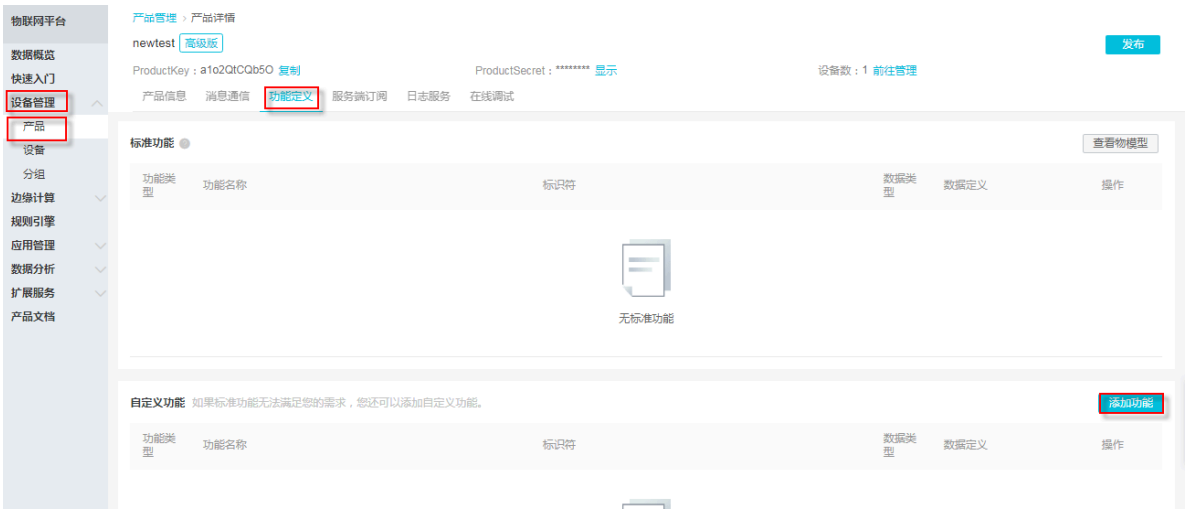
操作步骤

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

4. 单击功能定义。

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

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



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

属性参数设置如下：

参数	描述
功能名称	属性的名称，如用电量。同一产品下功能名称不能重复。 支持中文、大小写字母、数字、短划线和下划线，且必须以中文、英文或数字开头，不超过30个字符。 如果您创建产品时选择了功能模板，输入功能名称时，将从标准功能库中筛选匹配的标准属性，供您选择。 **说明:** 当接入网关协议为Modbus时，不支持标准属性，仅支持自定义属性。

参数	描述
标识符	属性唯一标识符，在产品中具有唯一性。即 Alink JSON 格式中的identifier的值，作为设备上报该属性数据的 Key，云端根据该标识符校验是否接收数据。可包含英文、数字、下划线，长度不超过50个字符，如 PowerComsuption。 说明: 不能用以下系统保留参数作为标识符: set、get、post、time、value。
数据类型	- int32：32位整型。需定义取值范围、步长和单位符号。 - float：单精度浮点型。需定义取值范围、步长和单位符号。 - double：双精度浮点型。需定义取值范围、步长和单位符号。 - enum：枚举型。定义枚举项的参数值和参数描述，如 1-加热模式、2-制冷模式。 - bool：布尔型。采用 0 或 1 来定义布尔值，如 0-关、1-开。 - text：字符串。需定义字符串的数据长度，最长支持 2048 字节。 - date：时间戳。格式为 string 类型的 UTC 时间戳，单位：毫秒。 - struct：JSON对象。定义一个 JSON 结构体，新增 JSON 参数项，如定义灯的颜色是由 Red、Green、Blue 三个参数组成的结构体。不支持结构体嵌套。 - array：数组。需声明数组内元素的数据类型，可选择int32、float、double、text或struct。需确保同一个数组元素类型相同，数组长度最长不超过 128 个元素。 说明: 当设备协议为Modbus时，无需设置该参数。
步长	属性值和事件以及服务中输入输出参数值变化的最小粒度。数据类型为int32、float、double时，需要设置步长。
单位	单位可选择为无或根据实际情况选择。
读写类型	- 读写：请求读写的方法支持 GET（获取）和 SET（设置）。 - 只读：请求只读的方法仅支持 GET（获取）。 说明: 当接入网关协议为Modbus时，无需设置该参数。
描述	输入文字，对该功能进行说明或备注。长度限制为100字。

参数	描述
扩展描述 说明: 当接入网关协议为Modbus时	- 操作类型：包括输入状态（只读）、线圈状态（读写）、保持寄存器（读写）、输入寄存器（只读）。 - 寄存器地址：十六进制，必须以0x开头，如0xFE，限制范围是0x0~0xFFFF。 - 原始数据类型：支持int16、uint16、int32、uint32、int64、uint64、float、double、string、bool、自定义（原始数据）多种数据类型。 - 寄存器数据个数：操作类型为输入状态/线圈状态时限制为1~2000，为保持寄存器/输入寄存器时限制为1~125。 - 交换寄存器内高低字节：把寄存器内16位数据的前后8个bits互换。 - 交换寄存器顺序：把原始数据32位数据的bits互换。 - 缩放因子：不能为0，默认为1，可以为负数。 - 采集间隔：数据采集间隔，单位ms，不能小于10。 - 数据上报方式：分为按时上报和变更上报。

参数	描述
扩展描述 说明：当接入网关协议为OPC UA时	节点名称。需保证属性维度下唯一。

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

服务参数设置如下：

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

参数	描述
输出参数	设置该服务的出参，可选。 单击新增参数，在弹窗对话框中添加服务出参。 当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。 说明: - 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。 - 您可以直接选择某个属性作为出参，也可以自定义参数，如将已定义的属性土壤湿度作为出参，则云端调用自动喷灌服务时，将返回当前土壤湿度的数据。 - 一个服务最多支持定义20个出参。
扩展描述 说明: 当接入网关协议为OPC UA时	节点名称。需保证服务维度下唯一。

参数	描述
描述	输入文字，对该服务功能进行说明或备注。长度限制为100字。

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

事件参数设置如下：

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

参数	描述
标识符	事件唯一标识符，在产品下具有唯一性。即 Alink JSON 格式中该事件的identifier的值，作为设备上报该事件数据的 Key，如 ErrorCode。 说明: 不能用以下系统保留参数作为标识符： set、get、post、time、value。
事件类型	- 信息：指设备上报的一般性通知，如完成某项任务等。 - 告警：设备运行过程中主动上报的突发或异常情况，告警类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。 - 故障：设备运行过程中主动上报的突发或异常情况，故障类信息，优先级高。您可以针对不同的事件类型进行业务逻辑处理和统计分析。
输出参数	该事件的出参。单击新增参数，在弹窗对话框中添加一个服务出参。您可以直接选择某个属性作为出参，也可以自定义参数。如，将已定义的属性电压作为出参，则设备上报该故障事件时，将携带当前设备的电压值，用于进一步判断故障原因。 当接入网关协议为OPC UA时，需设置参数索引，用于标记参数的顺序。 说明: - 不能用以下系统保留参数作为输出参数的标识符：set、get、post、time、value。 - 一个事件最多支持定义20个出参。
扩展描述 说明: 当接入网关协议为OPC UA时	节点名称。需保证事件维度下唯一。
描述	输入文字，对该事件功能进行说明或备注。长度限制为100字。

1.4.3 导入物模型

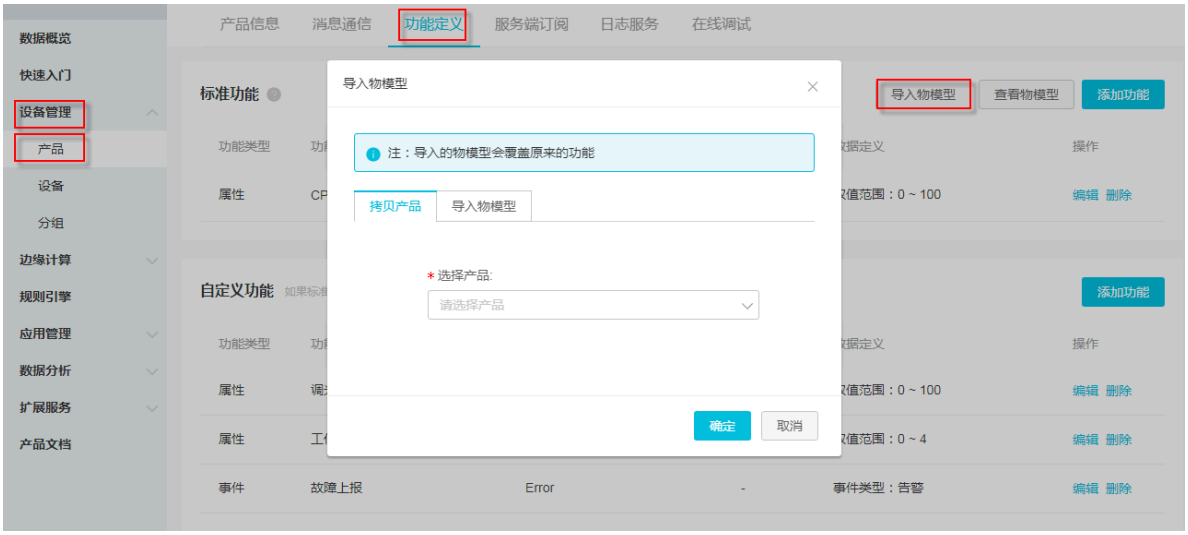
本文讲述如何将已有物模型导入新创建的产品。

操作步骤

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

4. 单击功能定义 > 导入物模型为产品导入物模型。

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



可通过以下两种方法导入物模型:

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

  如果需要修改某些功能，单击功能对应的编辑按钮，即可修改该功能。

· 导入物模型：将自定义的物模型脚本粘贴进编辑框中，然后单击确定。

1.4.4 物模型格式

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

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

物模型的 JSON 字段说明如下:

```
{
    "schema":"物的TSL描述schema",
    "link":"云端系统级uri,用来调用服务/订阅事件",
    "profile":{
        "productKey":"产品key"
    },
    "properties":[
        {
            "identifier":"属性唯一标识符(产品下唯一)",
            "name":"属性名称",
            "accessMode":"属性读写类型，只读(r)，读写(rw)",
            "required":"是否是标准功能的必选属性"
            "dataType":{
```

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

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

若该产品是设备，且设备协议为Modbus或OPC UA时，可查看物模型扩展配置。

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

```
        "specs": {
          "registerCount": "寄存器的数据个数”，string，customized data特
有",
          "swap16": "把寄存器内16位数据的前后8个bits互换（byte1byte2 ->
byte2byte10），除 string，customized data外，其他数据类型特有",
          "reverseRegister": "Ex:把原始数据32位数据的bits互换（byte1byte2
byte3byte4 ->byte3byte4byte1byte2”，除 string，customized data外，其他数
据类型特有"
        }
      },
      "scaling": "缩放因子",
      "pollingTime": "采集间隔，单位是ms",
      "trigger": "数据的上报方式，目前有 按时上报:1和变更上报:2"
    }
  ]
}
```

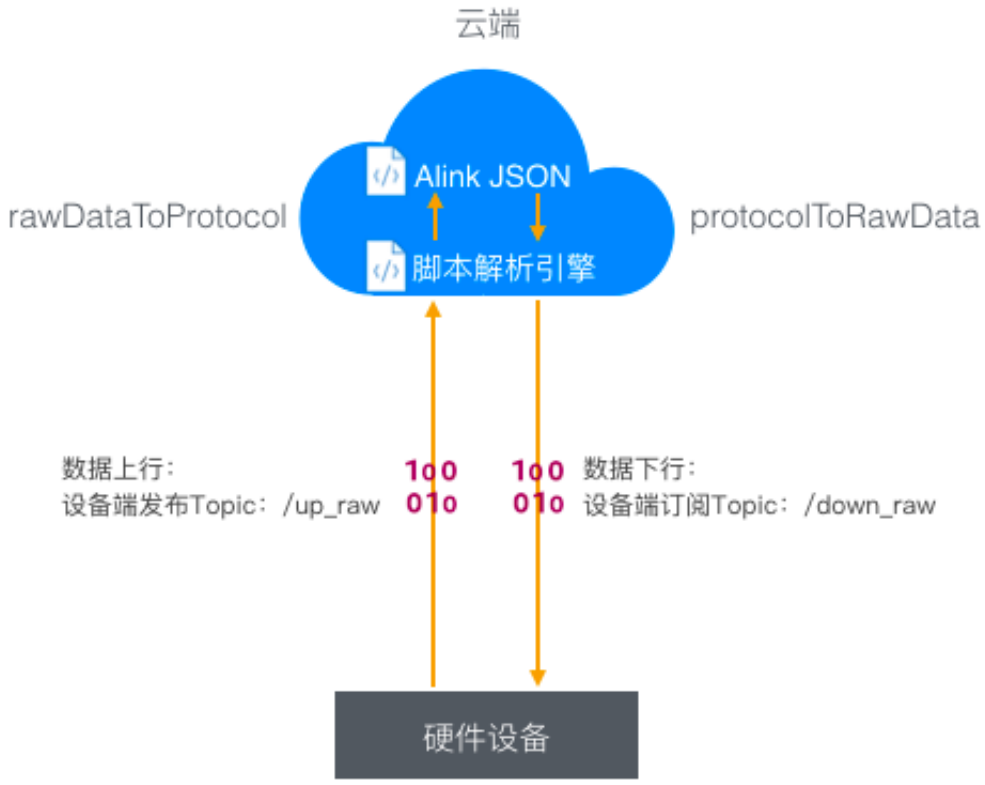
1.5 数据解析

由于低配置且资源受限，或者对网络流量有要求的设备，不适合直接构造JSON数据与物联网平台通信，可将原数据透传到物联网平台。您需在物联网平台控制台，编写数据解析脚本，用于将设备上下行数据分别解析为物联网平台定义的标准格式（Alink JSON）和设备的自定义数据格式。

关于数据解析

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

数据解析流程：



透传格式数据上下行完整流程，请参见*Alink*协议文档中，“设备上报透传格式的属性或事件”章节和“调用设备服务或设置属性”章节。

脚本格式

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

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

脚本编辑和测试过程简介

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

1. 登录物联网平台控制台。
2. 在左侧导航栏中，单击设备管理 > 产品。
3. 单击创建产品，创建一个高级版产品，并将数据格式选择透传/自定义格式。请参见创建产品(高级版)。
4. 在产品详情页面，单击数据解析进入脚本编辑页面。在下方编辑框中编写数据解析脚本。目前支持编写 JavaScript 语言。详情请参见下一章节编写脚本示例。

编写脚本时，您可以

- 单击全屏，支持全屏查看或编辑脚本。单击退出全屏即可退出当前编辑界面。
- 单击页面底部的保存草稿，系统将保存本次编辑的结果。您下次再进入平台时，将提示您最近一次保存的草稿，您可以选择恢复编辑或删除草稿。
  - 草稿不会进入到脚本解析的运行环境中，保存草稿不会影响已经提交的正式脚本。
  - 每次保存草稿将覆盖上一次保存的草稿。

5. 脚本编写完成后，在模拟输入栏中，输入数据，然后单击运行测试编写的脚本是否能正常解析数据。模拟数据及返回结果，请参见以下测试数据解析脚本章节。
6. 确认脚本能正常运行，并能正确解析数据后，单击提交将脚本提交到运行环境中。设备数据上下行时，系统将自动调用脚本对数据进行转换。

7. 真实设备上报数据测试。

   a. 注册设备。

   b. 将设备证书（ProductKey、DeviceName和DeviceSecret）烧录到开发完成的设备端SDK中。

   c. 模拟设备上报数据。

   d. 在物联网平台控制台，该设备详情页上运行状态栏下，查看物联网平台接收到的设备运行数据。

编写脚本示例

以下以设备数据格式为16进制，有三个属性prop_float、prop_int16和prop_bool的产品为例，介绍数据解析脚本格式和内容。

1. 创建一个高级版产品，并且数据格式选择为透传/自定义格式。然后，为该产品定义以下三个属性。请参见新增物模型文档中，“自定义属性”章节。

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

2. 在设备通信协议中做如下定义：

   表 1-1: 设备上报数据请求

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

字段	字节数
属性prop_float	4字节

表 1-2: 设备上报数据响应

字段	字节数
帧类型	1字节
请求ID	4字节
结果code	1字节

表 1-3: 设置属性请求

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

表 1-4: 属性设置响应

字段	字节数
帧类型	1字节
请求ID	4字节
结果code	1字节

3. 编写脚本。

脚本中需定义支持以下两个方法：

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

脚本示例Demo如下：

```
var COMMAND_REPORT = 0x00; //属性上报
var COMMAND_SET = 0x01; //属性设置
var COMMAND_REPORT_REPLY = 0x02; //上报数据返回结果
var COMMAND_SET_REPLY = 0x03; //属性设置设备返回结果
var COMMAD_UNKOWN = 0xff;    //未知的命令
```

```
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //标准
ALink JSON格式topic， 设备上传属性数据到云端
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'; //标准ALink
 JSON格式topic， 云端下发属性控制指令到设备端
var ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set'; //标
准ALink JSON格式topic，设备上报属性设置的结果到云端
/*
示例数据:
设备上报数据
传入参数 ->
    0x00000000010032010000000
输出结果 ->
    {"method":"thing.event.property.post","id":"1","params":{"
prop_float":0,"prop_int16":50,"prop_bool":1},"version":"1.0"}

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

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

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

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

测试数据解析脚本

输入以上示例脚本后，在编辑器下方输入模拟数据，验证脚本的运行情况。输入模拟数据后，单击运行，系统将调用该脚本模拟进行数据解析。运行结果将显示在右侧的运行结果区域中。

- 模拟解析设备上报的属性数据。

  选择模拟类型为设备上报数据，输入以下16进制格式数据的设备上报数据，然后单击运行。

  ```
  0x00002233441232013fa00000
  ```

  数据解析引擎会按照脚本规则，将16进制透传数据转换为JSON格式数据。运行结果栏将显示解析结果。

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

- 模拟解析设备上报属性数据后，物联网平台下发的返回结果数据。

  模拟类型选择设备接收数据，输入以下JSON格式数据，然后单击运行。

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

  数据解析引擎会将JSON格式数据转换为以下16进制数据。

  ```
  0x0200003039c8
  ```

- 模拟解析设备接收到的物联网平台下发的属性设置数据。

  选择模拟类型为设备接收数据，输入以下JSON格式数据，然后单击运行。

  ```
  {
  ```

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

数据解析引擎会将JSON格式数据转换为以下16进制数据。

```
0x0100003039014d0142f6e76d
```

- **模拟解析设备返回的属性设置结果数据。**

  选择模拟类型为设备上报数据，输入以下16进制格式数据，然后单击运行。

  ```
  0x0300223344c8
  ```

  数据解析引擎会将16进制透传数据转换为以下JSON格式数据。

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

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

本地环境调试脚本的方法

目前物联网平台数据解析不支持debug调试。建议开发过程，先在本地开发、调试完成后，再将脚本拷贝到物联网控制台的脚本编辑器中。可参考如下方式调用。

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

问提排查指南

设备连接物联网平台，上报数据后，若脚本解析运行正常，在设备详情页运行状态标签页，将会显示设备上报的数据。

若设备已经上报了数据，但是页面上却没有显示数据，如下图所示：

若出现这种情况，可在监控运维 > 日志服务中，选择物模型数据分析，查看设备和物联网平台属性相关的通信日志。

问题排查过程如下：

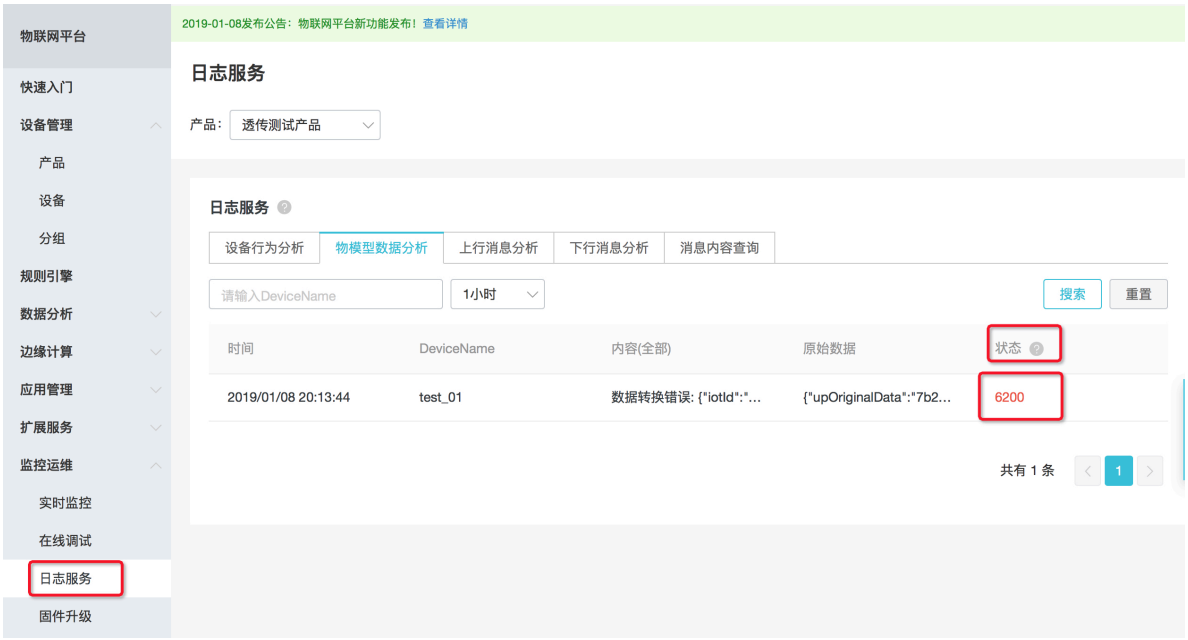
1. 在日志服务中查看设备上报的数据记录。日志中会显示脚本转化后的数据和原数据。
2. 结合日志说明文档，查看错误码的信息

3. 按照错误码提示，结合脚本和设备上报的数据排查问题。

下面列举一些错误。

- 脚本不存在。

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



- Alink method不存在。

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

可以查询日志中的原始数据，如：

```
{"upOriginalData":"7b226d6574686f64223a227468696e672e657665
6e742e70726f70657274792e706f7374222c226964223a2231222c227061
72616d73223a7b2270726f705f666c6f6174223a31302c2270726f705f69
```

```
6e743136223a33302c2270726f705f626f6f6c223a317d2c227665727369
6f6e223a22312e30227d","upTransformedData":{}}
```

可以看到脚本转换后的数据为空。上报的数据开头是`0x7b`。协议规定命令是`0x00~0x03`，因而无法识别`0x7b`。对于不能识别的命令，默认返回就是空对象。这是脚本本身有问题，需修改脚本。

1.6 Topic

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

1.6.1 什么是Topic

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

什么是Topic类?

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

您创建产品后，物联网平台会为该产品创建系统 Topic 类。您还可以根据业务需求，自定义Topic类。基础版产品和高级版产品均支持自定义Topic 类。

- 在产品的Topic类列表页，创建自定义*Topic*类。
- 其他功能用到的Topic类，如固件升级等，请参考具体功能文档中的Topic相关章节进行创建。

您可以在产品详情页的Topic类列表页，查看该产品的所有Topic类。

在您创建设备后，产品Topic类会自动映射到设备上。您无需单独为每个设备授权Topic。

图 1-1: Topic 自动生成示意图

关于Topic类的说明：

- Topic类中，以正斜线(/)进行分层，区分每个类目。其中，有两个类目为既定类目：`${YourProductKey}`表示产品的标识符ProductKey；`${YourDeviceName}`表示设备名称。
- 类目命名只能包含字母，数字和下划线(_)。每级类目不能为空。
- 设备操作权限：发布表示设备可以往Topic发布消息；订阅表示设备可以从Topic订阅消息。
- 系统Topic类是由系统预定义的Topic类，不支持用户自定义，不采用`/${YourProductKey}`开头。例如，高级版中，针对物模型所提供的Topic类一般以`/sys/`开头；固件升级相关的Topic类以`/ota/`开头；设备影子的Topic类以`/shadow/`开头。

什么是Topic?

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

您可以在物联网平台控制台，对应的设备详情页的Topic类列表页，查看该设备支持的具体Topic。

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

- 使用规则引擎来转发设备数据，需配置相关Topic。在设置规则引擎时，配置的Topic中可使用通配符，且同一个类目中只能出现一个通配符。

表 1-5: Topic 通配符

通配符	描述
#	这个通配符必须出现在Topic的最后一个类目，代表本级及下级所有类目。例如， Topic：`/${YourProductKey}/device1/user/#`，可以代表`/${YourProductKey}/device1/user/update`和`/${YourProductKey}/device1/user/update/error`。
+	代表本级所有类目。例如，Topic：`/${YourProductKey}/+/user/update`，可以代表`/${YourProductKey}/device1/user/update`和`/${YourProductKey}/device2/user/update`。

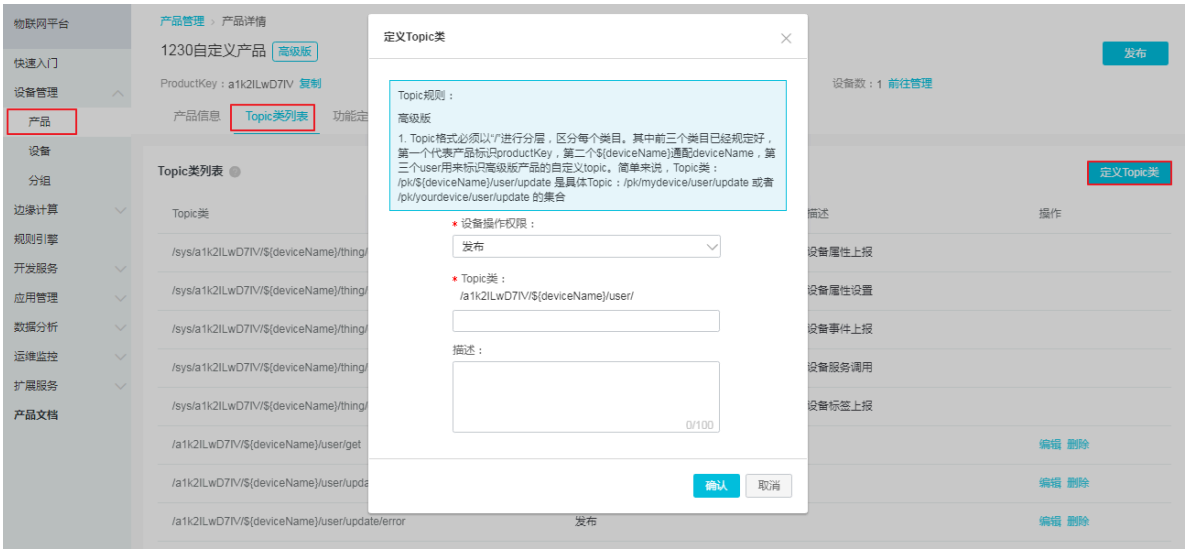
1.6.2 自定义Topic

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

操作步骤

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

5. 定义Topic类。



- 设备操作权限：设备对该Topic的操作权限，可设置为发布、订阅、发布和订阅。
- Topic类：根据页面上方的Topic规则设置Topic类的自定义类目名称。
- 描述：对自定义的Topic类进行描述，可以为空。

6. 单击确认。

Topic类中的通配符

自定义Topic类时，您可以使用通配符。通配符内容请参考什么是*Topic*。其中：

- #代表本级及下级所有类目。
- +代表本级所有类目。

> **说明:**
> 创建带通配符的Topic类时，需注意：
>
> - 只有设备操作权限为订阅的产品，才支持使用通配符。
> - 通配符#只能在Topic类的最后一个类目。
> - 带通配符的Topic不支持在设备的Topic列表页面执行发布消息操作。

1.7 标签

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

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

标签包括产品标签、设备标签和分组标签。标签的结构为`Key:Value`。

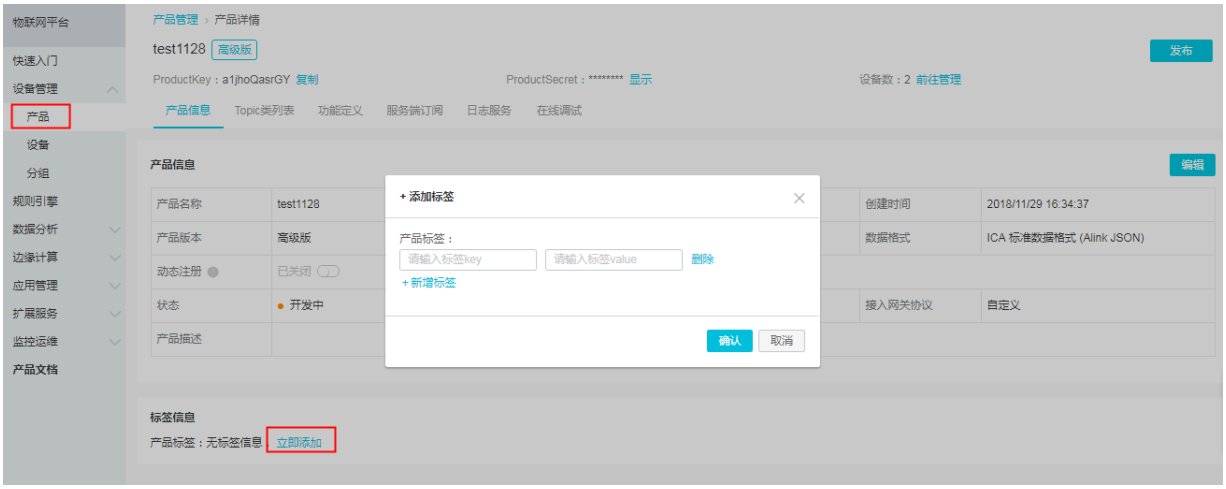
本文将详细讲解产品标签、设备标签与分组标签的创建。

产品标签

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

添加产品标签操作步骤：

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

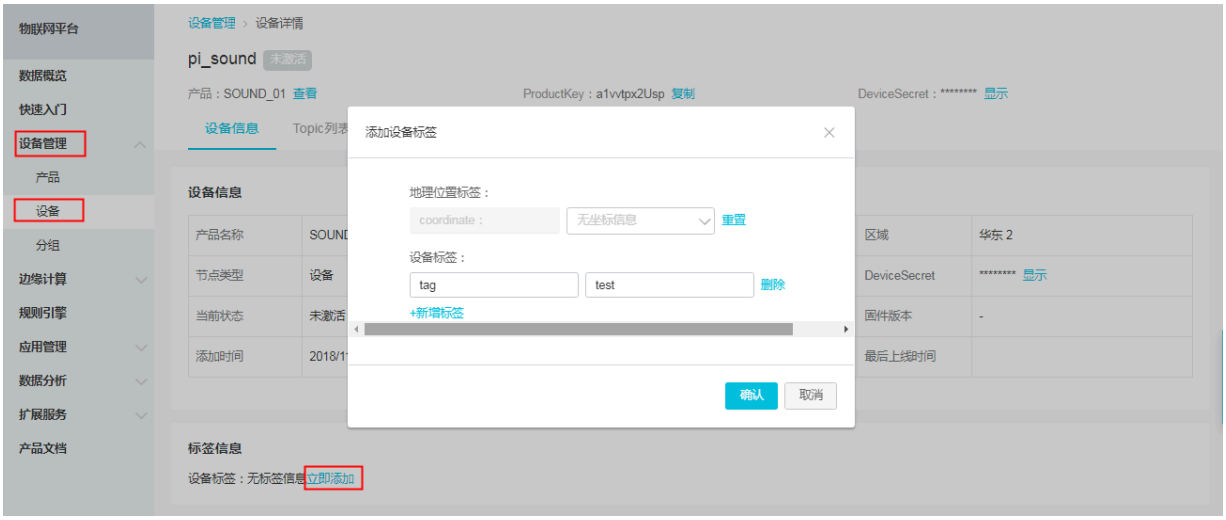


设备标签

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

在控制台添加设备标签的操作步骤：

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



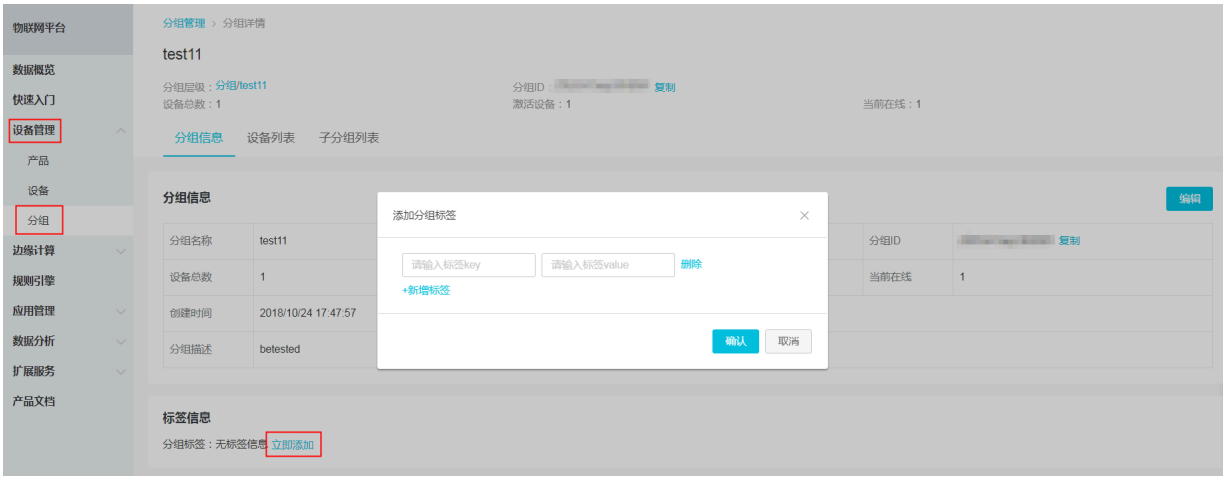
设备标签信息会跟随设备在系统内部流转。并且，物联网平台可以基于规则引擎，将设备标签发给阿里云其他云服务。

分组标签

设备分组用于跨产品管理设备。分组标签通常描述的是对一个分组下所有设备和子分组所具有的共性信息，如分组下的设备所在的地域、空间等。需在创建分组后，再为该分组添加标签。

添加分组标签操作步骤：

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



1.8 网关与子设备

1.8.1 网关与子设备

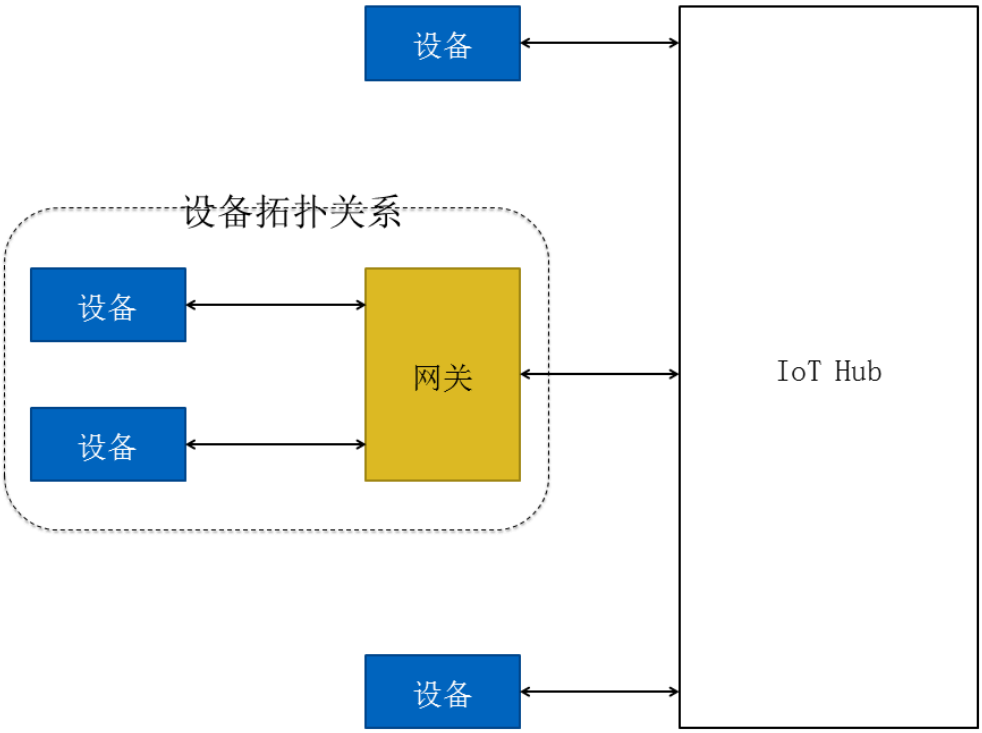
物联网平台支持设备直连，也支持设备挂载在网关上，作为网关的子设备，由网关直连。

网关与设备

创建产品与设备时，需要选择节点类型。平台目前支持两种节点类型：设备和网关。

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

网关与子设备的拓扑关系如下图所示：



接入流程

网关连接上云后，由网关将拓扑关系同步至云端，代替子设备完成设备认证、消息上传、指令接收等与平台的通信。而子设备将由网关统一管理。

1. 网关可参考普通设备接入流程，接入物联网平台。具体请参考*Link Kit SDK*文档。
2. 子设备接入物联网平台有两种方式：
   - 使用一机一密的认证方式，像普通设备一样，预烧录设备证书信息（ProductKey、DeviceName和DeviceSecret），然后接入物联网平台。
   - 使用一型一密的认证方式，在控制台打开动态注册开关，并预注册子设备的DeviceName。由网关代替子设备进行注册，云端校验子设备DeviceName，校验通过后，动态下发DeviceSecret。然后子设备通过设备证书（ProductKey、DeviceName和DeviceSecret）接入物联网平台。

1.8.2 子设备通道管理

当您使用高级版产品时，可以为网关设备添加子设备通道，添加完成后，该网关设备可以使用这些通道管理子设备。当前支持添加Modbus协议、OPC UA协议和自定义协议三种通道。

1. 左侧导航栏选择设备管理 > 设备。
2. 在设备管理页面，找到网关设备，单击查看，进入设备详情页。

3. 单击子设备通道管理，并添加不同协议的管理通道。

- Modbus

  在Modbus页面，单击添加Modbus通道，并根据页面提示，输入参数。

参数	描述
通道名称	网关下需要唯一
传输模式	支持RTU和TCP两种
当传输模式为RTU时，需设置以下参数：	
选择串口	如/dev/tty0、/dev/tty1
波特率	从下拉列表中选择
数据位	支持5、6、7、8四种
校验位	支持无校验、奇校验、偶校验三种
停止位	支持1、1.5、2三种
当传输模式为TCP时，需设置以下参数：	
IP地址	输入点分十进制格式的地址
端口号	输入0~65535范围的整数

- OPC UA

  单击OPC UA > 添加OPC UA通道，并根据页面提示，输入参数。

参数	描述
通道名称	网关下需要唯一
连接地址	如opc.tcp://localhost:4840
用户名	非必填
密码	非必填

参数	描述
方法调用超时时间	单位为秒

- 自定义

  a. 单击自定义 > 添加自定义通道。

  b. 在弹出界面上，设置通道名称。

  c. 添加自定义配置。

> 说明:
> 自定义配置仅支持JSON格式。建议您在本地写好后，粘贴进来。

1.8.3 子设备管理

您可以在网关下面关联子设备，并将子设备配置下发至网关。

前提条件

- 若子设备接入网关协议为Modbus或OPC UA，子设备接入网关前，该网关下应配置好对应的子设备通道，具体请参考子设备通道管理文档。
- 2018年9月4日之前创建的产品及设备，也可以作为子设备添加进来，但仅支持添加拓扑关系，不支持关联子设备通道和自定义配置。

操作步骤

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

4. 在弹出页面上，设置要关联的子设备相关信息。

参数	描述
产品	选择子设备对应的产品名称。

参数	描述
设备	选择子设备对应的设备名称。
如果该设备使用Modbus协议	
关联通道	必选，从网关下的子设备管理通道中，选择该子设备使用的关联通道。
从站号	输入1~247之间的整数值。
如果该设备使用OPC UA协议	
关联通道	必选，从网关下的子设备管理通道中，选择该子设备使用的关联通道。
节点路径	如Objects/Device1，Objects是固定根节点，后面Device1是到设备节点路径上的节点名称，以/分隔。
如果该设备使用自定义协议	
关联通道	非必选，从网关下的子设备管理通道中，选择该子设备使用的关联通道。
自定义配置	若选择了关联通道，需自定义配置。只支持JSON格式。

5. 子设备信息配置完成后，可回到网关设备详情页，单击配置下发，将子设备所属产品的物模型和扩展配置，以及子设备与网关之间的连接配置信息，一并下发至网关。

> **说明:**
> 如果该网关设备已被添加到边缘实例中，当您单击配置下发后，页面会自动跳转到边缘实例详情页。您需在边缘实例页面通过部署操作进行配置下发。

6. 子设备信息配置完成后，可在对应子设备详情页，查看相应信息。此处可单击编辑，修改相关信息。

后续操作

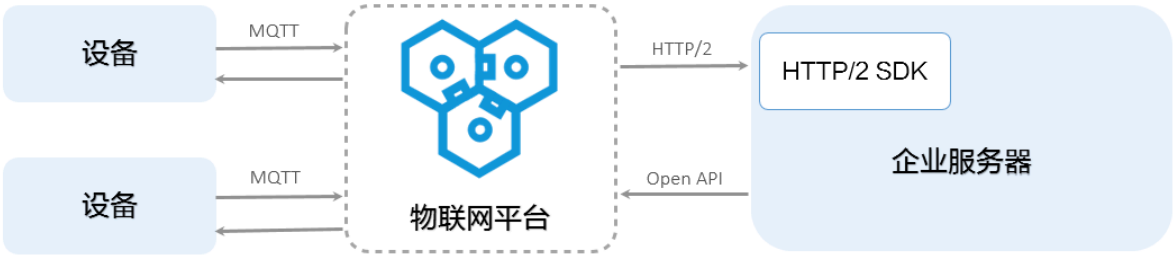
- 您可以使用*Alink*协议自行开发设备，并参见网关配置下发章节，将此处网关与子设备的配置下发至设备端。
- 若您使用了边缘计算节点，可参考设备接入简介进行配置。

1.9 服务端订阅

1.9.1 什么是服务端订阅

服务端可以直接订阅产品下配置的所有类型的消息。

目前，新版物联网平台通过HTTP/2通道进行消息流转。配置HTTP/2服务端订阅后，物联网平台会将消息通过HTTP/2通道推送至服务端。通过接入HTTP/2 SDK，企业服务器可以直接从物联网平台接收消息。HTTP/2 SDK提供身份认证、Topic订阅、消息发送和消息接收能力，并支持设备接入和云端接入能力。HTTP/2 SDK适用于物联网平台与企业服务器之间的大量消息流转，也支持设备与物联网平台之间的消息收发。



说明:

旧版物联网平台用户使用阿里云消息服务（MNS）进行消息流转，您可以将服务端订阅方式升级为HTTP/2方式。如果您继续使用在MNS这种方式，物联网平台将设备消息推送至MNS，服务端应用通过监听MNS队列接收设备消息。

1.9.2 开发指南(Java)

本文档介绍如何配置服务端订阅、接入HTTP/2 Java版SDK、进行身份认证和设置消息接收接口。

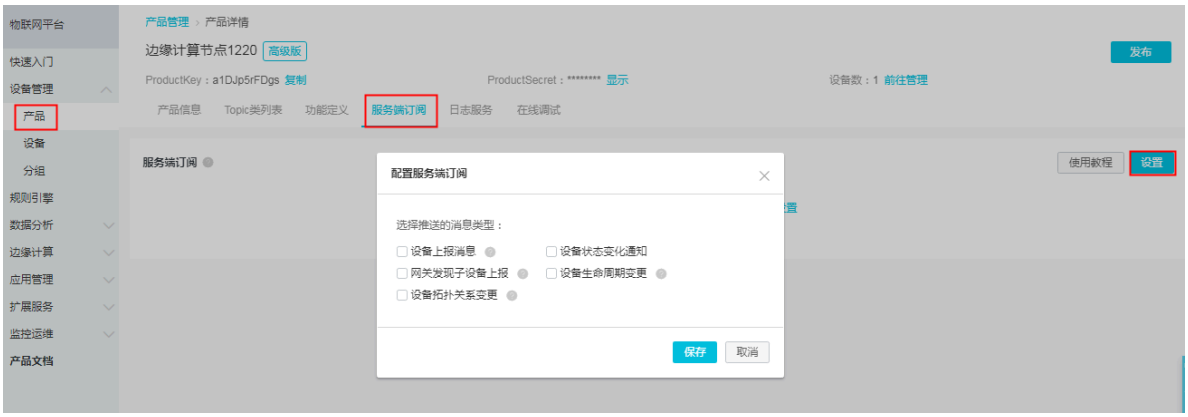
SDK配置方法：

- 有关设备端接入配置，请下载*iot-http2-sdk-demo*，并参考使用*HTTP/2 SDK(Java)*建连接入物联网平台。
- 有关服务端订阅配置，请下载*HTTP/2 Java SDK demo*，参考本文以下章节进行配置。

以下简单介绍服务端订阅的开发流程。

配置服务端订阅

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



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

  高级版产品的设备上报消息，包括设备上报的自定义数据和属性、事件、属性设置响应、服务调用响应的物模型数据。而基础版产品只包括设备上报的自定义数据。

  例如，有一个高级版产品，有3个Topic类，分别是：

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

  那么，服务端订阅会推送具有发布权限的Topic类中的消息，即`/${YourProductKey}/${YourDeviceName}/user/update`和`/sys/${YourProductKey}/${YourDeviceName`

}/thing/event/property/post中的消息。其中，/sys/${YourProductKey}/${YourDeviceName}/thing/event/property/post中的数据已经过系统处理。

- 设备状态变化通知：指一旦该产品下的设备状态变化时通知的消息，例如设备上线、下线的消息。设备状态消息的发送 Topic 为 /as/mqtt/status/${YourProductKey}/${YourDeviceName}。勾选后，可以通过 HTTP/2 SDK 接收设备状态变化的通知消息。
- 网关发现子设备上报：高级版产品特有的消息类型，网关可以将发现的子设备上报，需要网关上的应用程序支持。
- 设备拓扑关系变更：高级版产品特有的消息类型，指的是子设备和网关之间的拓扑关系建立和解除。
- 设备生命周期变更：高级版产品特有的消息类型，包括设备创建、删除、禁用、启用等消息的订阅。

接入 SDK

在工程中添加 maven 依赖接入 SDK。

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

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

身份认证

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

建立链接示例如下：

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

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

```
        // 构造客户端
        MessageClient client = MessageClientFactory.messageClient(
profile);

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

accessKey 即您的账号的 AccessKey ID， accessSecret 即 AccessKey ID对应的 AccessKey Secret。请登录阿里云控制台，将光标移至您的账号头像上，在选项框中选择 accesskeys 查看您的 AccessKey ID 和 AccessKey Secret；选择安全设置查看您的账号 ID。

regionId为您的物联网平台服务地域。

设置消息接收接口

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

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

consume 方法的返回值决定 SDK 是否发送 ACK。

设置消息接收接口的方法如下：

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

其中，

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

  消息体包含的内容如下：

  ```
  public class Message {
      // 消息体
      private byte[] payload;
      // Topic
  ```

```
    private String topic;
    // 消息ID
    private String messageId;
    // QoS
    private int qos;
}
```

- 具体请参考消息体格式。
- `messageClient.setMessageListener("/${YourProductKey}/#",messageCallback);`用于设置回调。本示例中，设置为指定 Topic 回调。

  您可以设置为指定 Topic 回调，也可以设置为通用回调。

  - 指定 Topic 回调

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

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

    示例：

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

  - 通用回调

    未指定 Topic 回调的消息，则调用通用回调。

    设置通用回调方法：

    ```
    messageClient.setMessageListener(messageCallback);
    ```

```
//当收到消息topic未匹配到已定指的Topic 回调时，调用该回调
```

- 设置回复 ACK。

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

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

    设置为手动回复 ACK 后，需要调用`MessageClient.ack()`方法回复 ACK，参数为 `MessageToken`。`MessageToken` 参数值可在接收消息中获取。

    手动回复 ACK 的方法：

    ```
    messageClient.ack(messageToken);
    ```

消息体格式

- 设备状态通知：

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

参数	类型	说明
status	String	设备状态，online上线，offline离线
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
time	String	发送通知的时间点
utcTime	String	发送通知的UTC时间点
lastTime	String	状态变更时最后一次通信时间
utcLastTime	String	状态变更时最后一次通信的UTC时间

参数	类型	说明
clientIp	String	设备公网出口IP

说明:

为避免消息时序紊乱造成影响，建议您根据lastTime来维护最终设备状态。

- 设备生命周期变更：

```
{
    "action" : "create|delete|enable|disable",
    "iotId" : "4z819VQHk6VSLmmBJfrf00107ee201",
    "productKey" : "12345565569",
    "deviceName" : "deviceName1234",
    "deviceSecret" : "",
    "messageCreateTime": 1510292739881
}
```

参数	类型	描述
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	设备所属产品的ProductKey
deviceName	String	设备名称
deviceSecret	String	设备密钥，仅在action为create时包含该参数
messageCreateTime	Long	消息产生的时间戳，单位为毫秒

- 设备拓扑关系变更：

```
{
    "action" : "add|remove|enable|disable",
    "gwIotId": "4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey": "1234556554",
    "gwDeviceName": "deviceName1234",
    "devices": [
        {
          "iotId": "4z819VQHk6VSLmmBJfrf00107ee201",
          "productKey": "12345565569",
          "deviceName": "deviceName1234"
       }
    ],
    "messageCreateTime": 1510292739881
```

```
}
```

参数	类型	说明
action	String	- add：新增拓扑关系 - remove：移除拓扑关系 - enable：启用拓扑关系 - disable：禁用拓扑关系
gwIotId	String	网关设备在平台内的唯一标识
gwProductKey	String	网关设备所属产品的ProductKey
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	子设备所属产品的ProductKey
deviceName	String	子设备名称
messageCreateTime	Long	消息产生的时间戳，单位为毫秒

- 网关发现子设备上报：

```
{
    "gwIotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey":"1234556554",
    "gwDeviceName":"deviceName1234",
    "devices":[
        {
            "iotId":"4z819VQHk6VSLmmBJfrf00107ee201",
            "productKey":"12345565569",
            "deviceName":"deviceName1234"
        }
    ]
}
```

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

参数	类型	说明
deviceName	String	子设备名称

1.9.3 开发指南(.NET)

本文档介绍如何配置服务端订阅、接入HTTP/2 .NET版SDK、进行身份认证和设置消息接收接口。

有关*HTTP/2 .NET SDK*配置的具体信息，

- 关于设备端接入配置，请下载*H2*设备接入*Demo*，并参考配置*HTTP/2 .NET SDK*接入物联网平台。
- 关于服务端订阅配置，请下载服务端订阅*Demo*，参考本文以下章节进行配置。

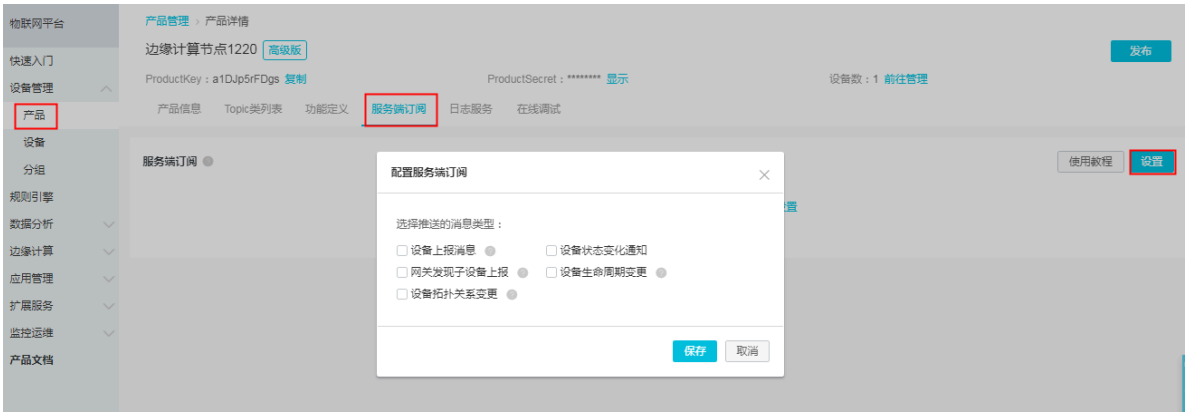
> 说明:
> 使用Demo程序时，需要使用NLog包。请访问NLog官网下载，建议版本号：4.5.11。

以下简单介绍服务端订阅的开发流程。

配置服务端订阅

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

5. 选择推送的消息类型。



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

  高级版产品的设备上报消息，包括设备上报的自定义数据和属性、事件、属性设置响应、服务调用响应的物模型数据。而基础版产品只包括设备上报的自定义数据。

  例如，有一个高级版产品，有3个Topic类，分别是：

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

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

接入 SDK

在工程中添加依赖包*iotx-as-http2-net-sdk.dll*。

身份认证

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

建立链接示例如下：

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

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

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

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

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

`accessKey` 即您的账号的 AccessKey ID， `accessSecret` 即 AccessKey ID对应的 AccessKey Secret。请登录阿里云控制台，将光标移至您的账号头像上，在选项框中选择 accesskeys 查看您的 AccessKey ID 和 AccessKey Secret；选择安全设置查看您的账号 ID。

`regionId`为您的物联网平台服务地域。

设置消息接收接口

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

消息接收接口如下：

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

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

设置消息接收接口的方法如下：

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

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

其中，

- 参数 Http2ConsumeMessage 是消息回执的消息体。

  消息体包含的内容如下：

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

- 具体请参考消息体格式。

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

  您可以设置为指定 Topic 回调，也可以设置为通用回调。

  - 指定 Topic 回调

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

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

    示例：

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

  - 通用回调

    未指定 Topic 回调的消息，则调用通用回调。

    设置通用回调方法：

    ```
    new DefaultHttp2MessageCallback()
    ```

- 设置回复 ACK。

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

  - 自动回复 ACK：设置为自动回复 ACK 后，若 `IHttp2MessageCallback.consume` 的返回值为 `ConsumeAction.CommitSuccess` 则 SDK 会发送 ACK；返回 `ConsumeAction`

.CommitFailure 或抛出异常，则不会返回 ACK。对于 QOS>0 且未回复 ACK 的消息，服务端会重新发送。

- 手动回复 ACK：通过ConsumeAction.CommitFailure设置手动回复 ACK。

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

  手动回复 ACK 的方法：

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

消息体格式

- 设备状态通知：

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

参数	类型	说明
status	String	设备状态，online上线，offline离线
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
time	String	发送通知的时间点
utcTime	String	发送通知的UTC时间点
lastTime	String	状态变更时最后一次通信时间
utcLastTime	String	状态变更时最后一次通信的UTC时间
clientIp	String	设备公网出口IP

说明：

为避免消息时序紊乱造成影响，建议您根据lastTime来维护最终设备状态。

- 设备生命周期变更：

```
{
    "action" : "create|delete|enable|disable",
    "iotId" : "4z819VQHk6VSLmmBJfrf00107ee201",
    "productKey" : "12345565569",
```

```
    "deviceName" : "deviceName1234",
    "deviceSecret" : "",
    "messageCreateTime": 1510292739881
}
```

参数	类型	描述
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	设备所属产品的ProductKey
deviceName	String	设备名称
deviceSecret	String	设备密钥，仅在action为create时包含该参数
messageCreateTime	Long	消息产生的时间戳，单位为毫秒

- 设备拓扑关系变更：

```
{
    "action" : "add|remove|enable|disable",
    "gwIotId": "4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey": "1234556554",
    "gwDeviceName": "deviceName1234",
    "devices": [
        {
          "iotId": "4z819VQHk6VSLmmBJfrf00107ee201",
          "productKey": "12345565569",
          "deviceName": "deviceName1234"
       }
    ],
    "messageCreateTime": 1510292739881
}
```

参数	类型	说明
action	String	- add：新增拓扑关系 - remove：移除拓扑关系 - enable：启用拓扑关系 - disable：禁用拓扑关系
gwIotId	String	网关设备在平台内的唯一标识
gwProductKey	String	网关设备所属产品的ProductKey
gwDeviceName	String	网关设备名称
devices	Object	变更的子设备列表

参数	类型	说明
iotId	String	子设备在平台内的唯一标识
productKey	String	子设备所属产品的ProductKey
deviceName	String	子设备名称
messageCreateTime	Long	消息产生的时间戳，单位为毫秒

- 网关发现子设备上报：

```
{
    "gwIotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey":"1234556554",
    "gwDeviceName":"deviceName1234",
    "devices":[
        {
            "iotId":"4z819VQHk6VSLmmBJfrf00107ee201",
            "productKey":"12345565569",
            "deviceName":"deviceName1234"
        }
    ]
}
```

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

1.9.4 使用限制

当您使用服务端订阅时，请注意以下限制。

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

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

1.9.5 使用MNS订阅设备消息

物联网平台基础版产品支持云端应用通过监听MNS队列，获取设备消息。本章讲解使用MNS订阅设备消息的配置方法。

操作步骤

1. 在物联网平台控制台上，为产品配置服务端订阅，实现平台将消息自动转发至MNS。
   a) 单击设备管理 > 产品，查看刚刚创建的产品。
   b) 单击服务端订阅，在推送MNS区域，单击设置，勾选设备上报消息或设备状态变化通知。
      - 设备上报消息，指平台将设备上报的数据自动转发至MNS。
      - 设备状态变化通知，指平台将设备上下线的消息自动推送至MNS。
   c) 订阅完成后，MNS将自动新建一个消息队列。消息队列信息显示在控制台上。

   配置完成大约1分钟后生效。
2. 通过监听MNS队列，接收设备消息。

   具体操作，请参考*MNS*文档。

   本文示例中使用MNS Java SDK监听消息，涉及以下信息填写。

   - 在pom.xml文件中，添加如下依赖：

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

```
</dependency>
```

- 接收消息时，需填入以下信息：

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

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

- 填写接收设备消息的逻辑：

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

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

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

- 运行程序，完成对MNS队列的监听。

3. 启动设备，上报消息。

   可参考*SDK*文档，查看设备上报消息的具体内容。

4. 检查云端应用是否监听到设备消息。若成功监听，将获得如下所示消息代码。

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

参数	说明
messageid	物联网平台生成的消息ID，19位大小。
messagetype	消息类型。 · status：设备状态通知 · upload：设备上报消息

参数	说明
topic	云端监听到的信息来自哪个Topic。 · 当messagetype=status时，为null · 当messagetype=upload时，为具体Topic
payload	Base64编码的数据。 · 当messagetype=status时，数据是平台的通知数据，数据格式为： `{ "status":"online|offline",//设备状态 "productKey":"12345565569",//产品唯一标识, ProductKey "deviceName":"deviceName1234",//设备名称 "time":"2018-08-31 15:32:28.205",//发送通知时间点 "utcTime":"2018-08-31T07:32:28.205Z",//发送通知UTC时间点 "lastTime":"2018-08-31 15:32:28 .195",//状态变更时最后一次通信时间 "utcLastTime":"2018-08-31T07:32:28.195Z ",//状态变更时最后一次通信UTC时间 "clientIp":"xxx.xxx.xxx.xxx"//设备端公网出口IP }` 说明： 为避免消息时序紊乱造成影响，建议您根据lastTime来维护最终设备状态。 · 当messagetype=upload时，数据是设备发布到Topic中的原始数据
timestamp	时间戳，以Epoch时间表示。

1.10 设备分组

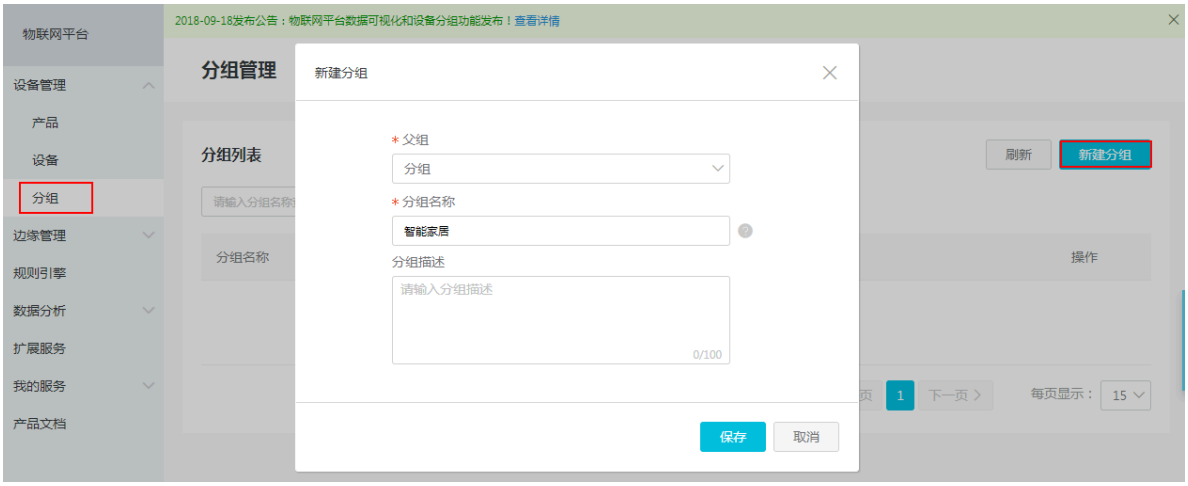
物联网平台提供设备分组功能。您可以通过设备分组来进行跨产品管理设备。本章节介绍如何在物联网平台控制台创建设备分组和管理分组。

操作步骤

1. 登录物联网平台控制台。
2. 单击设备管理 > 分组进入分组管理页面。
3. 单击新建分组，设置分组参数，并单击保存。

说明：

一个阿里云账号下最多可创建1,000个分组，包括分组和子分组。



参数信息解释如下：

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

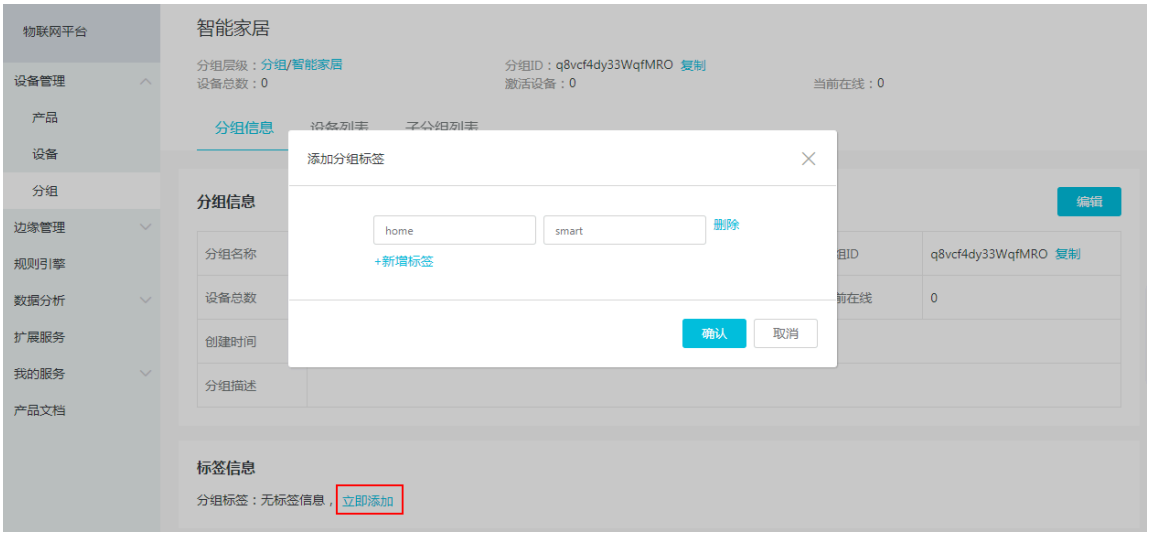
4. 在分组管理页面，单击已创建分组对应的查看操作按钮，进入分组详情页面。
5. （可选）为分组添加标签，即自定义分组标识，以便灵活管理分组。

a) 单击标签信息栏下的立即添加，并输入标签的key和value。

b) 单击确认添加标签。

说明：

一个分组最多可添加100个标签。



6. 单击设备列表 > 添加设备到分组。勾选设备，将选中设备添加至指定分组。

说明:

- 单次最多添加200个设备。单个分组最多添加20,000个设备。
- 一个设备最多可以被添加到10个分组中。

添加设备到分组页面右上方的全部和已选择按钮说明:

- 单击全部，显示所有设备列表。
- 单击已选择，显示您已勾选的设备列表。

7. （可选）单击子分组列表 > 新建分组，为分组添加子分组。

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

a) 选择该子分组的父组，输入子分组名称和描述，然后单击保存。

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

c) 单击设备列表 > 添加设备到分组，然后为该子分组添加设备。

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

说明：

- 一个分组最多可包含100个子分组。
- 分组只支持三级嵌套，即分组>子分组>子子分组。
- 一个子分组只能隶属于一个父组。
- 分组的嵌套关系创建后不能修改，只能删除后重新创建。
- 分组下有子分组时，不能直接删除分组。需子分组全部删除后，才能删除父组。

2 规则引擎

2.1 数据流转

2.1.1 数据流转概览

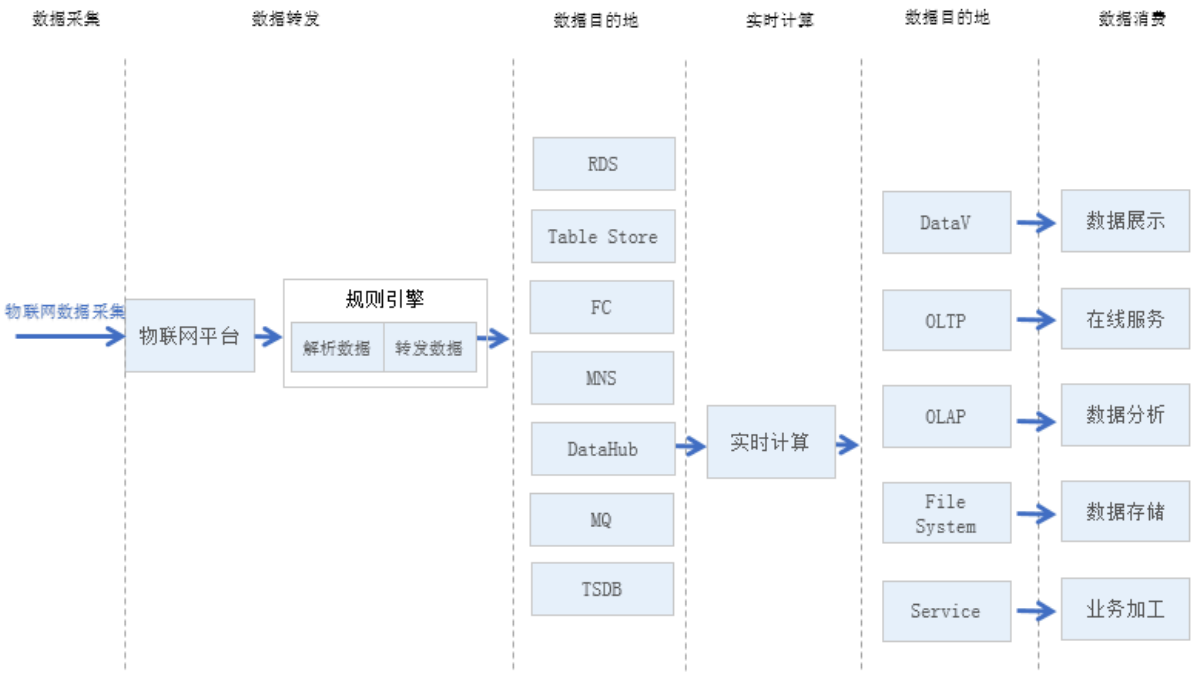
使用物联网平台规则引擎的数据流转功能，可将Topic中的数据消息转发至其他Topic或其他阿里云产品进行存储或处理。

什么是数据流转

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

- 将数据转发到另一个Topic中以实现M2M通信。
- 将数据转发到*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*MQ*、消息服务实现高可靠消费数据·。

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



数据流转使用限制

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

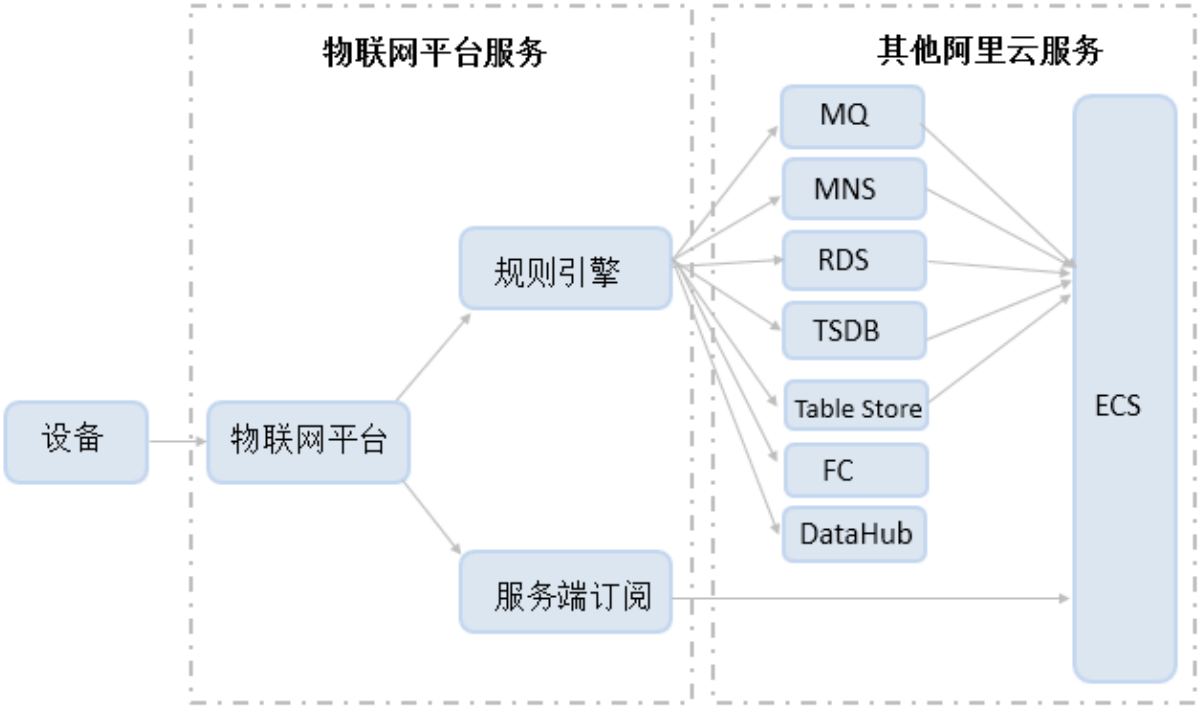
数据流转使用指南

- 基础版产品
  1. 设置数据流转规则：如何设置一条转发规则。
  2. SQL表达式：规则中SQL表达式的写法详解。
  3. 函数列表：规则中SQL表达式支持的函数列表。
  4. 数据流转过程：使用规则引擎数据流转过程中的数据流转格式。
  5. 地域和可用区：确认支持的目的云产品及地域信息。
  6. 使用实例：不同云产品目的地的转发设置详解。
- 高级版产品
  1. 设置数据流转规则：如何设置一条转发规则。
  2. SQL表达式：规则中SQL表达式的写法详解。
  3. 函数列表：规则中SQL表达式支持的函数列表。
  4. 数据流转过程：使用规则引擎数据流转过程中的数据流转格式。
  5. 数据格式(高级版)：高级版产品Topic中的数据格式。
  6. 地域和可用区：确认支持的目的云产品及地域信息。
  7. 使用实例：不同云产品目的地的转发设置详解。

2.1.2 数据流转方案对比

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

设备数据流转方案汇总



目前，流转方式有两类:

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

规则引擎 vs 服务端订阅

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

表 2-1: 规则引擎各方案对比

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

流转目标	适用场景	优点	缺点
云数据库RDS版	适合单纯的数据保存场景。	数据直接写入数据库。	-
时序时空数据库(TSDB)	合根据设备数据进行业务分析和监控的场景。	数据直接写入时序数据库。	-
DataHub	适合需对数据进行分析处理的场景。	数据直接写入DataHub。	-
表格存储 (Table Store)	适合单纯的数据存储场景。	数据直接写入表格存储实例。	-
函数计算(Function Compute)	需要简化设备开发过程，且对设备数据进行一定自由度的处理的场景。	· 数据处理自由度高。 · 功能多。 · 无需部署。	费用略高。

服务端订阅

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

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

规则引擎 + 消息队列 (MQ)

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

· 因为MQ性能突出，推荐通过MQ接收设备消息。

- MQ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	参考文档
- MQ的公网测试集群不能用于生产环境。 - MQ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。	- MQ的公网测试集群可以支持开发者本地接收消息，但是稳定性很差，请勿用于线上生产环境。 - 对于较大的设备消息量场景(> 5000 QPS)或者稳定性要求特别高的场景， 推荐使用 MQ 铂金版。 - 规则引擎数据流转失败，然后再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的影响防护。	- 设置数据流转规则 - 数据转发到*MQ* - 设备上报消息到您的服务器 - *MQ*使用指南

规则引擎 + 消息服务(MNS)

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

使用限制	使用注意	参考文档
请参见消息服务使用限制文档。	规则引擎数据流转失败后，再重试失败数次后，会丢弃消息。另外，消息类产品存在延迟的可能，业务场景一定要做好消息丢失或者延迟送达的防护。	- 设置数据流转规则 - 数据转发到*MNS* - *MNS*使用指南 - 消息服务 *MNS* 和消息队列 *MQ* 产品对比

规则引擎 + 函数计算

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

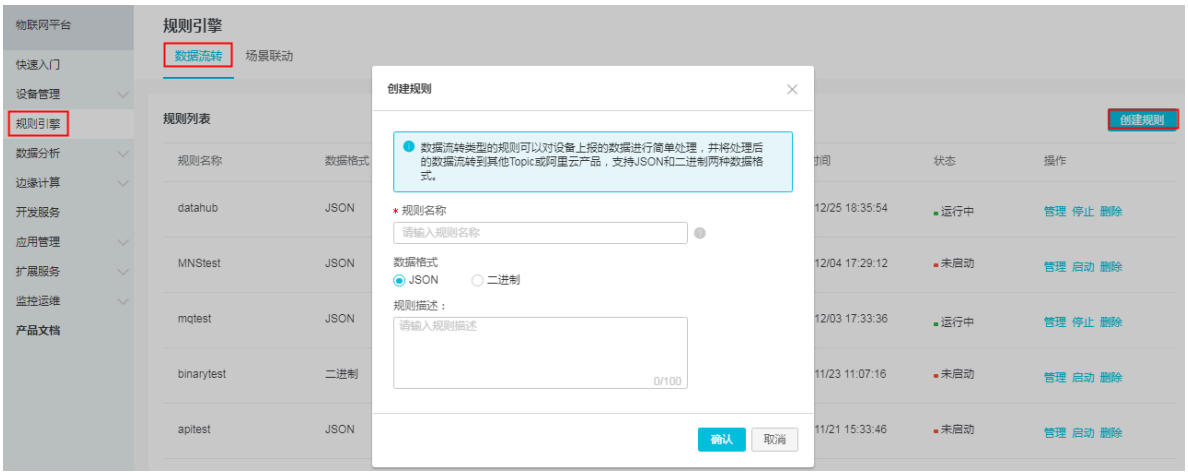
使用限制	使用注意	参考文档
请参见函数计算使用限制。	· 适用于对于设备消息有定制化的处理需求或者需简化开发运维的场景。 · 规则引擎数据流转失败后，再重试失败数次后，会丢弃消息。业务场景一定要做好消息丢失或者延迟送达的影响防护。	· 设置数据流转规则 · 转发数据到函数计算 · 实践案例(温湿度计上报数据到钉钉群机器人) · 函数计算使用指南

2.1.3 设置数据流转规则

本文将为您详细讲解如何设置一条完整的数据流转规则。设置过程依次是创建规则、编写处理数据的SQL、设置数据流转目的地。

操作步骤

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

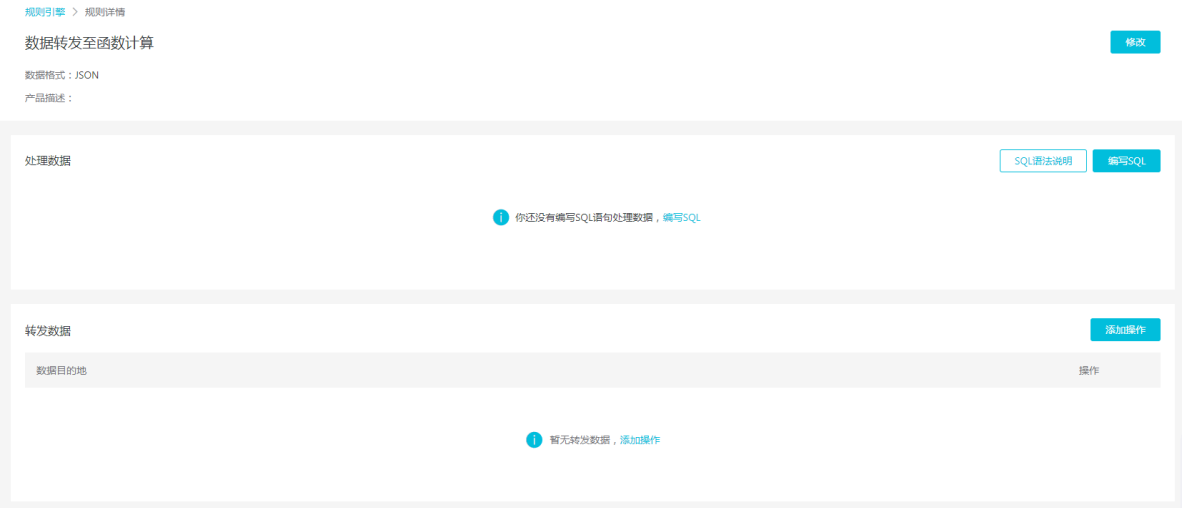


- 规则名称：输入规则名称，用以区别各条规则。名称支持中文汉字、英文字母、数字、下划线和连字符。长度为1-30字节。一个中文汉字占二字节。
- 数据格式：支持JSON和二进制。

> 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。

- 如果数据格式为二进制的数据，不支持转发至表格存储、时序时空数据库和云数据库RDS版。

4. 找到刚创建的规则，单击管理，进入规则详情页，设置规则具体信息。

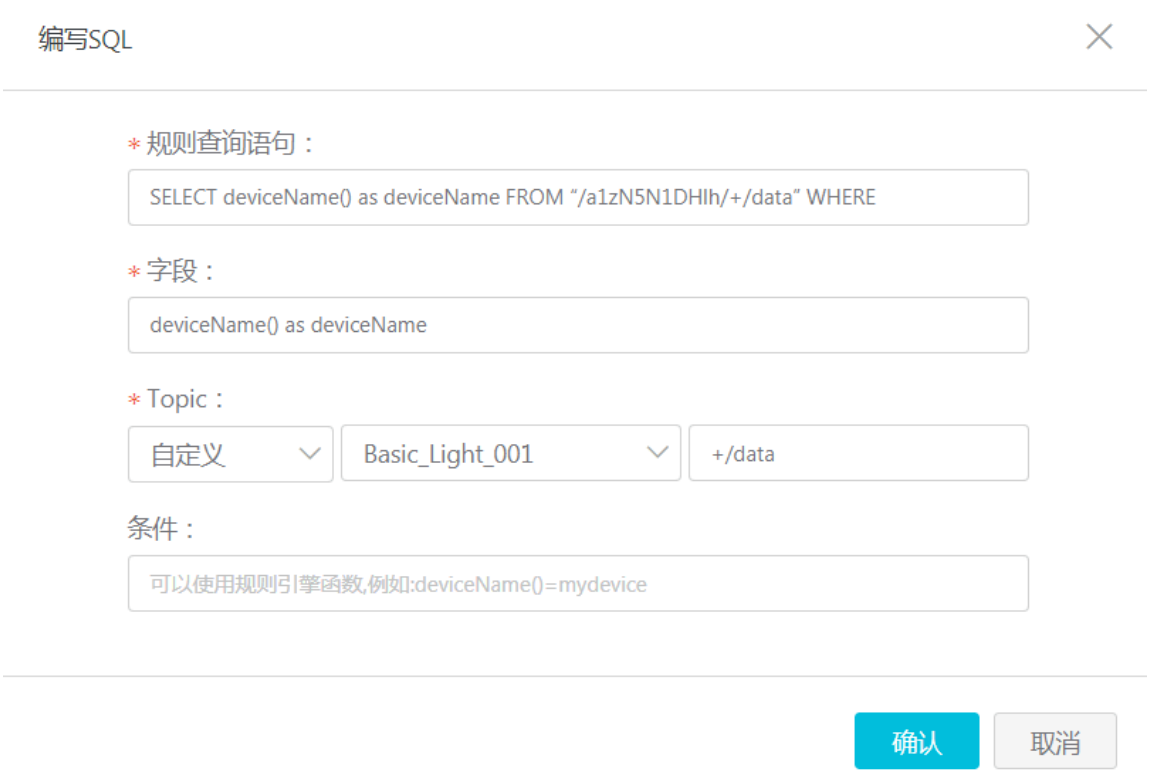


a) 单击编写SQL，编写SQL，设置数据处理的具体规则。

例如：以下SQL可以将deviceName从Basic_Light_001产品下，后缀为data的自定义Topic类中取出。

说明:

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



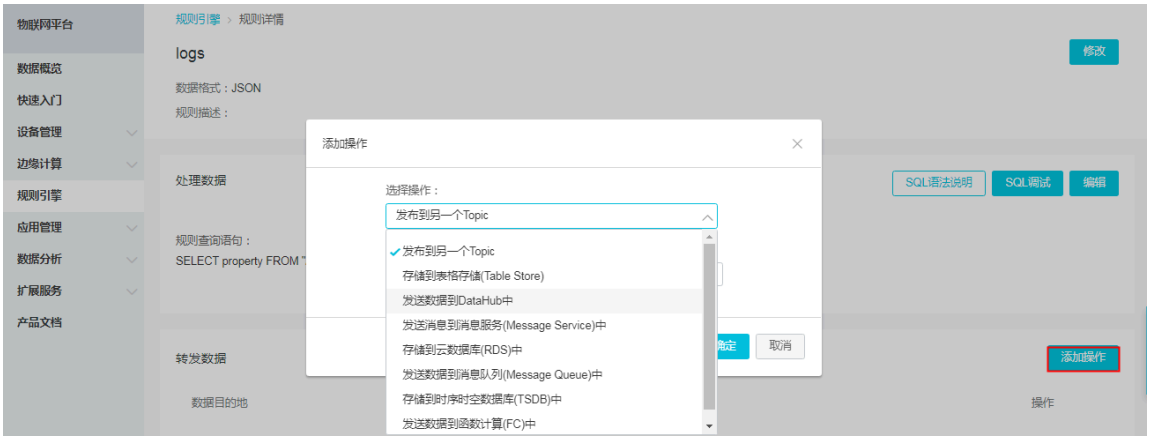
参数设置解释如下，具体可参考*SQL*表达式和函数列表。

- 规则查询语句：此处根据字段、Topic和条件，系统自动补充完整规则查询语句。
- 字段：指定消息内容字段，此示例填`deviceName() as deviceName`。
- Topic：选择需要处理的消息Topic。
  - 自定义：选择自定义后，表明该Topic是您自定义的产品Topic。此时您需要在选择产品后，补充完整该Topic。
  - sys：选择sys后，表明该Topic是系统定义的产品Topic。此时您需要在选择产品后，还需要选择设备，选择系统定义的某个Topic。
- 条件：规则触发条件。

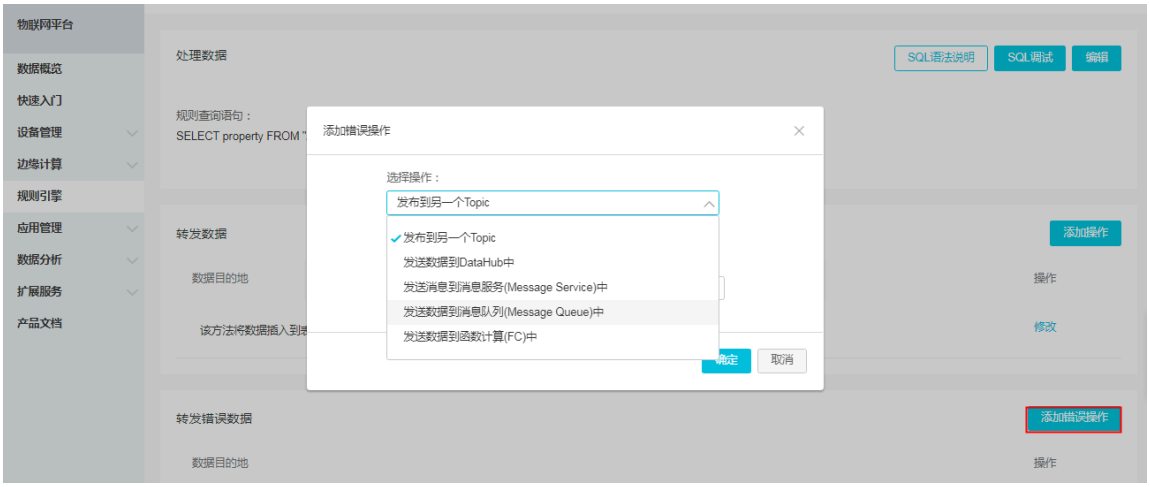
b) 单击转发数据一栏的添加操作，将数据转发至目的云产品。数据转发的具体实例，请参考使用实例目录下的具体文档。

说明:

最多可为一个规则创建10个数据转发操作。



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



说明：

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

```
{
      "ruleName":"",
      "topic":"",
      "productKey":"",
      "deviceName":"",
      "messageId":"",
```

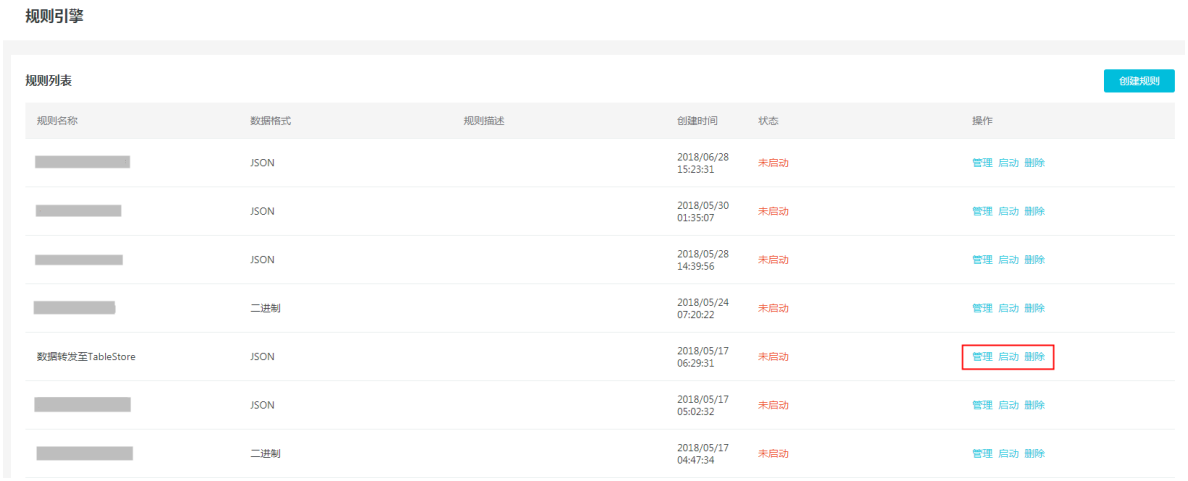
```
      "base64OriginalPayload":"",
      "failures":[
        {
          "actionType":"OTS",
          "actionRegion":"cn-shanghai",
          "actionResource":"table1",
          "errorMessage":""
        },
        {
          "actionType":"RDS",
          "actionRegion":"cn-shanghai",
       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

}
```

参数说明如下：

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

5. 返回至规则引擎的数据流转页签栏，单击启动。规则启动后，数据即可按照规则进行转发。



您也可以：

- 单击管理修改规则具体设置。
- 单击删除删除对应规则。

> 说明：
> 运行中的规则不可删除。

- 单击停止停止对应规则转发数据。

2.1.4 SQL表达式

建立数据流转规则时，可以编写SQL来解析和处理数据。其中，二进制格式的数据不做解析，直接透传。本文主要讲解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，这样就可以使用SQL处理。为便于理解，我们将数据流转的一条规则抽象为一条SQL表达（类试MySQL语法）：

例如某环境传感器用于火灾预警，可以采集温度、湿度及气压数据，上报数据内容如下：

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

假定温度大于38，湿度小于40时，需要触发报警，可以编写如下的SQL语句：

```
SELECT temperature as t, deviceName() as deviceName, location FROM /
ProductA/+/update WHERE temperature > 38 and humidity < 40
```

当上报的数据中，温度大于38且湿度小于40时，会触发该规则，并且解析数据中的温度、设备名称、位置，用于进一步处理。

FROM

FROM 可以填写Topic。而且，Topic中的设备名（deviceName）一级类目可以填写通配符+（代表本级所有类目），用于匹配需要处理的设备消息Topic。当有符合Topic规则的消息到达

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，将忽略此消息）。您可以使用`topic()`函数引用具体的Topic值。

上文例子中，"FROM /ProductA/+/update"语句表示该SQL仅处理符合/ProductA/+/update格式的消息，具体匹配参考 *Topic*。

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，比如`deviceName()`。

  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`。

  内置的SQL函数可以参考函数列表。

  例如上文，"SELECT temperature as t, deviceName() as deviceName, location"语句，其中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。

- 二进制数据格式

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

说明:

SELECT语句中的字段最多支持50个。

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。具体条件表达式列表见下方表格。

  上文例子中， "WHERE temperature > 38 and humidity < 40" 表示温度大于38且湿度小于40时，才会触发该规则，执行配置。

- 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。 转发数据动作中的表达式需要使用 `${表达式}` 引用对应的值。

对于上文例子，配置转发动作时，可以${t}、${deviceName}和${loaction}获取SQL解析结果，如果要将数据存储到TableStore，配置中可以使用${t}、${deviceName}和${loaction}。

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。

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

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

条件表达式支持列表

操作符	描述	举例
=	相等	color = ‘red’
<>	不等于	color <> ‘red’
AND	逻辑与	color = ‘red’ AND siren = ‘on’

OR	逻辑或	color = ‘red’ OR siren = ‘on’
( )	括号代表一个整体	color = ‘red’ AND (siren = ‘on’ OR isTest)
+	算术加法	4 + 5
-	算术减	5 - 4
/	除	20 / 4
*	乘	5 * 4
%	取余数	20 % 6
<	小于	5 < 6
<=	小于或等于	5 <= 6
>	大于	6 > 5
>=	大于或等于	6 >= 5
函数调用	支持函数，详细列表请参考函数列表。	deviceId()
JSON属性表达式	可以从消息payload以JSON表达式提取属性。	state.desired.color,a.b.c[0].d
CASE … WHEN … THEN … ELSE … END	Case 表达式（不支持嵌套）	CASE col WHEN 1 THEN ‘Y’ WHEN 0 THEN ‘N’ ELSE ‘’ END as flag
IN	仅支持枚举，不支持子查询。	比如， 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。	比如， where c1 like ‘%abc’ , where c1 not like ‘%def%’

2.1.5 函数列表

规则引擎提供多种函数，您可以在编写SQL时使用它们，实现多样化数据处理。

使用方法

您可以在SQL语句中，使用函数获取数据或者对数据做处理。

例如：以下示例中，用到了deviceName()，abs(number)，topic(number)三个函数。

```
SELECT case flag when 1 then '开灯' when 2 then '关灯' else '' end flag
, deviceName(),abs(temperature) tmr FROM "/topic/#" WHERE temperature>
10 and topic(2)='123'
```

说明:

使用函数时，通常单引号代表常量，不带引号或双引号代表变量。如`select “a” a1, ‘a’ a2, a a3`中，`a1`与`a3`等效，`a2`代表常量a。

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)	字符串连接。 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值是否以suffix结尾。
exp(number)	返回指定数字的指定次幂。
floor(number)	返回一个最接近它的整数，它的值小于或等于这个浮点数。
log(n, m)	返回自然对数。 如果不传m值，则返回log(n)。
lower(string)	返回小写字符串。
mod(n, m)	n%m 余数。
nanvl(value, default)	返回属性值。 若属性值为null，则返回default。
newuuid()	返回一个随机uuid字符串。

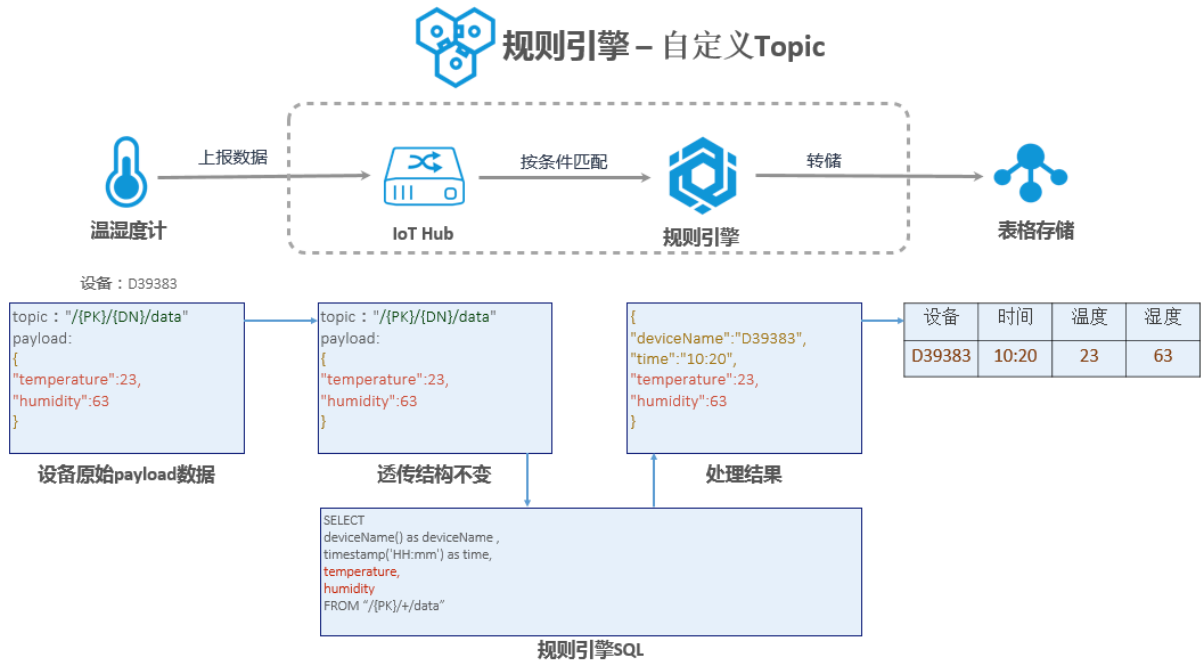
函数名	函数说明
payload(textEncoding)	返回设备发布消息的payload转字符串。 字符编码默认UTF-8，即 payload()默认等价于payload（‘utf-8’）。
power(n,m)	返回n的m次幂。
rand()	返回[0~1)之间随机数。
replace(source, substring, replacement)	对某个目标列值进行替换。 示例：replace(field,’a’,’1’)。
sin(n)	返回n值的正弦。
sinh(n)	返回n值的双曲正弦（hyperbolic sine）。
tan(n)	返回n值的正切。
tanh(n)	返回n值的双曲正切（hyperbolic tangent）。
timestamp(format)	返回当前系统时间，格式化后返回当前地域的时间。 format可选，如果为空则返回当前系统时间戳毫秒值，比如timestamp() = 1543373798943，timestamp('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'') = 2018-11-28T10:56:38Z。
timestamp_utc(format)	返回当前系统时间，格式化后返回UTC时间。 format可选，如果为空则返回当前系统时间戳毫秒值，比如timestamp_utc() = 1543373798943，timestamp_utc('yyyy-MM-dd\'T\'HH:mm:ss\'Z\'') = 2018-11-28T02:56:38Z。
topic(number)	返回Topic分段信息。 如，有一个Topic： /abcdef/ghi。使用函数 topic()，则 返回 “ /abcdef/ghi”； 使用 topic(1)，则 返回 “abcdef” ； 使用 topic(2) ，则返回 “ghi”。
upper(string)	返回大写字符。
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。

2.1.6 数据流转过程

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

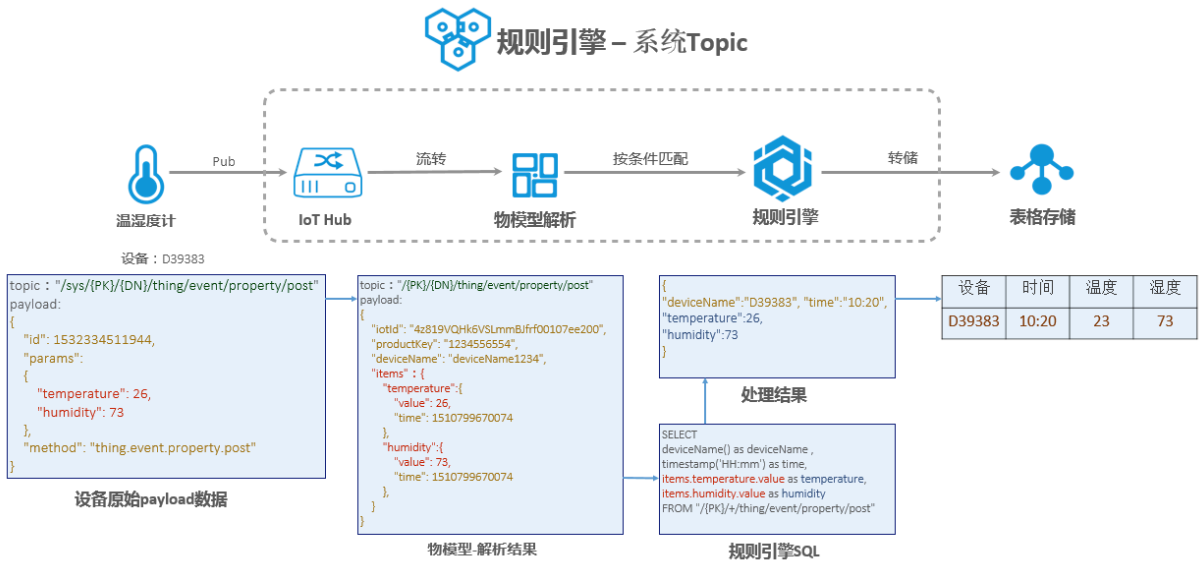
自定义Topic

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



系统Topic

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



说明:

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

2.1.7 数据格式(高级版)

使用规则引擎，您需要基于Topic编写SQL处理数据。基础版Topic和高级版自定义的Topic，数据格式是您自定义的，物联网平台不做处理。高级版系统默认的Topic，物联网平台定义了Topic格

式，此时您需要根据平台定义的数据格式，处理数据。本文将为您讲解高级版系统默认Topic的数据格式。

设备属性上报

通过该Topic获取设备上报的属性信息。

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

数据格式：

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

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
deviceType	String	设备类型
items	Object	设备数据
Power	String	属性名称，产品所具有的属性名称请参考TSL描述
Position	String	属性名称，产品所具有的属性名称请参考TSL描述
value	根据TSL定义	属性值
time	Long	属性产生时间，如果设备没有上报默认采用云端生成时间

参数	类型	说明
gmtCreate	Long	数据流转消息产生时间

设备上报事件

通过该topic获取设备上报的事件信息。

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

数据格式：

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

参数说明：

参数	类型	说明
iotId	String	设备在平台内的唯一标识
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
type	String	事件类型，事件类型参考TSL描述
value	Object	事件的参数
Power	String	事件参数名称
Position	String	事件参数名称
time	Long	事件产生时间，如果设备没有上报默认采用远端时间
gmtCreate	Long	数据流转消息产生时间

设备生命周期变更

通过该Topic获得设备创建、删除、禁用、启用等消息。

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

数据格式：

```
{
    "action" : "create|delete|enable|disable",
    "iotId" : "4z819VQHk6VSLmmBJfrf00107ee200",
    "productKey" : "X5eCzh6fEH7",
    "deviceName" : "5gJtxDVeGAkaEztpisjX",
    "deviceSecret" : "",
    "messageCreateTime": 1510292739881
}
```

参数说明：

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

设备拓扑关系变更

通过该Topic获得子设备和网关之间拓扑关系建立和解除信息。

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

数据格式：

```
{
    "action" : "add|remove|enable|disable",
    "gwIotId": "4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey": "1234556554",
    "gwDeviceName": "deviceName1234",
    "devices": [
        {
          "iotId": "4z819VQHk6VSLmmBJfrf00107ee201",
          "productKey": "12345565569",
          "deviceName": "deviceName1234"
```

```
        }
    ],

    "messageCreateTime": 1510292739881
}
```

参数说明：

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

网关发现子设备数据流转

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

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

数据格式：

```
{
    "gwIotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "gwProductKey":"1234556554",
    "gwDeviceName":"deviceName1234",
    "devices":[
        {
            "iotId":"4z819VQHk6VSLmmBJfrf00107ee201",
            "productKey":"12345565569",
            "deviceName":"deviceName1234"
        }
    ]
}
```

参数说明：

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

设备下行指令结果数据流转

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

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

数据格式：

```
{
    "gmtCreate":1510292739881,
    "iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "productKey":"1234556554",
    "deviceName":"deviceName1234",
    "requestId":1234,
    "code":200,
    "message":"success",
    "topic":"/sys/1234556554/deviceName1234/thing/service/property/set
",
    "data":{

    }
}
```

参数说明：

参数	类型	说明
gmtCreate	Long	UTC时间戳
iotId	String	设备在平台内的唯一标识
productKey	String	设备所属产品的唯一标识
deviceName	String	设备名称
requestId	Long	阿里云产生和设备通信的信息id
code	Integer	调用的结果信息
message	String	结果信息说明

参数	类型	说明
data	Object	设备返回的结果，非透传之间返回设备结果，透传则需要经过脚本转换

返回信息：

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

设备上下线状态数据流转

通过该Topic获取设备的上下线状态。

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

数据格式：

```
{
    "productKey":"1234556554",
    "deviceName":"deviceName1234",
    "gmtCreate":1510799670074,
    "deviceType":"Ammeter",
    "iotId":"4z819VQHk6VSLmmBJfrf00107ee200",
    "action":"online|offline",
    "status":{
        "value":"1",
        "time":1510292697471
    }
}
```

参数说明：

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

参数	类型	说明
status	Object	设备状态
value	String	状态值，1上线，0离线
time	Long	设备上下线时间
gmtCreate	Long	数据流转消息产生时间
action	String	设备状态变更动作，online上线，offline离线

2.1.8 地域和可用区

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

表 2-2: 地域和可用区列表

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

2.2 数据流转使用示例

2.2.1 数据转发到另一Topic

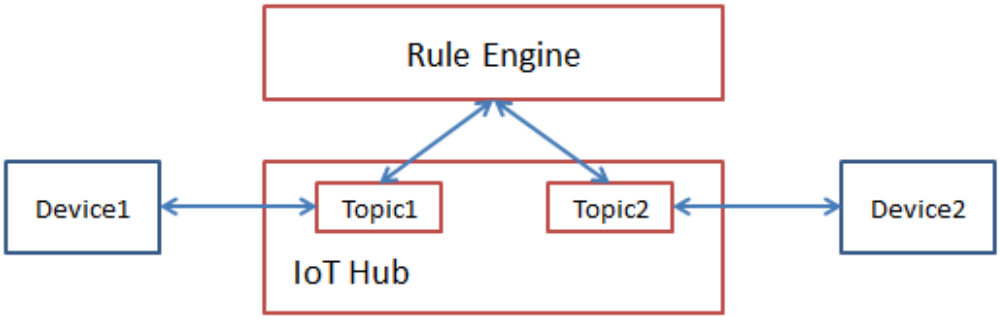
您可以将依据SQL规则处理完的数据，设置转发到另一个Topic中，实现M2M或者更多其他场景。

前提条件

在设置转发之前，您需要参考设置数据流转规则编写SQL完成对数据的处理。

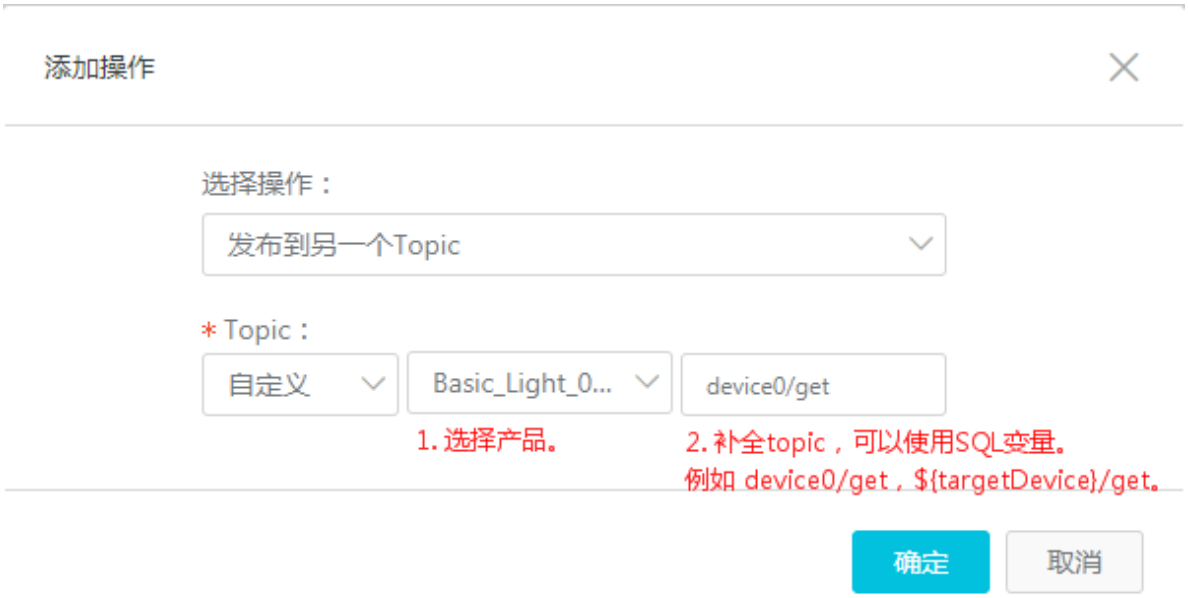
背景信息

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



操作步骤

1. 单击数据转发一栏的添加操作。出现添加操作页面。



2. 按照页面提示，设置参数。

   · 选择操作：此处选择发布到另一个Topic。

   · Topic：选择您需要把数据转发到哪一个Topic中。

     - 自定义：填写您自定义的产品Topic。在选择产品后，还需补充完整该Topic。您可以使用${}表达式引用上下文值。例如，填写${devicename}/get表示从消息中筛选出devicename信息，转发到后缀为get的Topic中。
     - sys：选择系统定义的Topic。在选择产品后，还需要选择设备，选择系统定义的某个Topic。

2.2.2 数据转发到MQ

您可以使用规则引擎，将物联网数平台据转发到消息队列（以下简称为MQ）中。从而实现消息从设备、物联网平台、MQ到应用服务器之间的全链路高可靠传输能力。

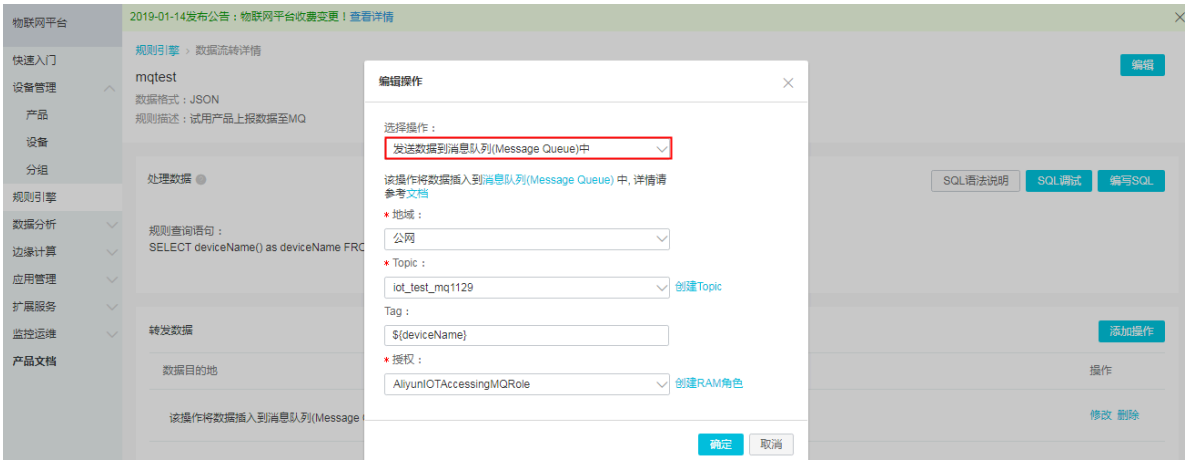
前提条件

在设置转发之前，您需参考设置数据流转规则，创建规则，编写用于处理数据的SQL。

本文仅介绍数据转发中的添加操作。

操作步骤

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



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

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

预期结果

规则启动之后，物联网平台即可将数据写入MQ的Topic，您可以使用MQ实现消息收发，具体使用方法，请参考*MQ*订阅消息。

2.2.3 数据转发到表格存储

您可以使用规则引擎，将数据转发到表格存储（Table Store）中。

前提条件

在设置转发之前，您需要参考设置数据流转规则编写SQL完成对数据的处理。

操作步骤

1. 单击数据转发一栏的添加操作，出现添加操作页面。选择存储到表格存储(Table Store)。

说明:

不支持二进制格式的数据转发至表格存储。

2. 按照界面提示，设置参数。

   - 选择操作：选择为存储到表格存储(Table Store)。
   - 地域、实例、数据表：选择表格存储数据表的基本信息，用于数据存储。
   - 主键：表格存储数据表都包含主键，选择好数据表之后，控制台将自动读出该表的主键，此处需要您配置主键的值。
   - 角色：授权平台将数据写入表格存储。此处需要您先创建一个具有表格存储写入权限的角色，然后将该角色赋予给规则引擎。这样，规则引擎才可以将处理完成的数据写入数据表中。

后续操作

示例

使用SQL抽取JSON数据：`{"device":"bike","message":"hello","data":[{...}]}`。因业务需要，需将此JSON数据存入表格存储中，主键是`device`，`id`，`message`。

配置及效果：

1. 规则会自动匹配主键是否为自增列，如果是自增列，将自动返填`AUTO_INCREMENT`，并且不能编辑，如主键id。
2. 在控制台配置主键的值，输入`${device}`。当有消息过来并触发规则，主键device就会存入JSON中device的value值。主键message同理。

   说明：
   `${}`是转义符，如果不输入该转义符，存入的将会是一个常量。
3. 主键配置完成后，平台将自动解析JSON中除主键外的key值，并依据key值自动创建表格存储的属性列。此例中，将会自动创建data属性列，并在该列下存入对应的value值。

   说明：
   该示例中data字段的值将会以完整的JSON字符串存入表格存储指定表格的data属性列中。

2.2.4 数据转发到DataHub

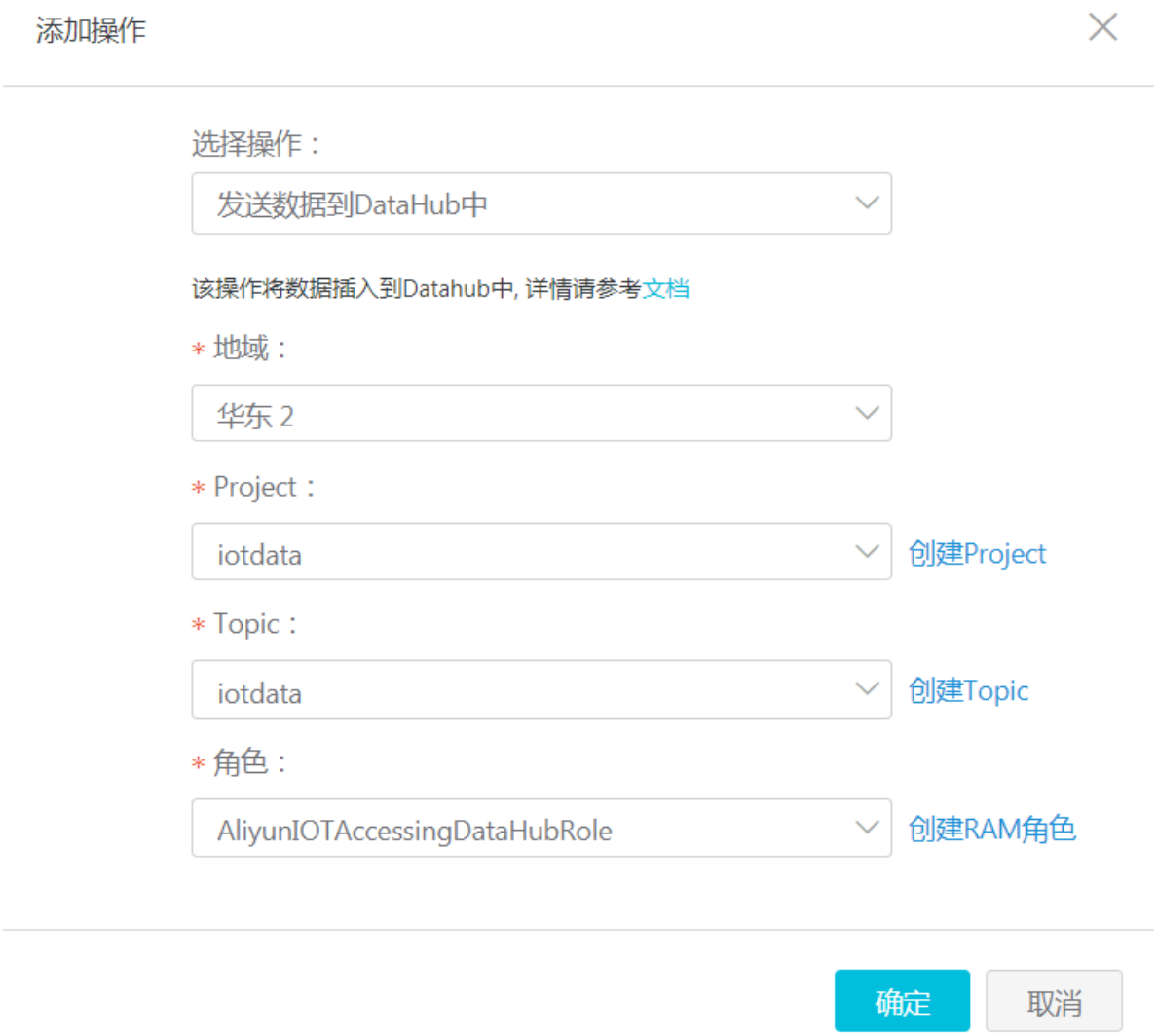
物联网平台用于接入和管理设备，数据存储和计算将交给阿里云其他产品。比如，您可以使用规则引擎将数据转到*DataHub*上，由DataHub为下游流计算、MaxCompute等提供实时数据，帮助用户实现更多计算场景。

前提条件

在设置转发之前，您需要参考设置数据流转规则编写SQL完成对数据的处理。

操作步骤

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



2. 按照界面提示，设置参数。

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

  说明:

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

2.2.5 数据转发到云数据库（RDS）

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

使用须知

- 只支持同地域转发，不支持跨区转发。比如，华东2节点的平台数据只能转发到华东2的RDS中。
- 只支持转发到VPC实例。
- 只支持MySQL实例。
- 支持普通数据库和高权限数据库的转发。
- 不支持二进制格式数据转发至RDS。

准备工作

在设置转发之前，您需要参考设置数据流转规则编写SQL完成对数据的处理。

操作步骤

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


添加操作
选择操作：
存储到云数据库(RDS)中
该操作将数据插入到云数据库(RDS)中, 详情请参考云数据库(RDS)中, 详情请参考文档
特别提醒：此操作仅针对专有网络的RDS实例，并且将会在您的RDS白名单中添加一条记录100.104.123.0/24，用于IoT访问您的数据库，请勿删除。
区域：
华东2
* VPC实例：
rm-uf63b1cji3z6xgpk5
创建实例
* MySQL数据库：
asdf
* 账号：
请选择
创建账号
* 请输入密码：
请输入该账号的密码
* 表名：
请输入已存在的表名
* 键：
RDS表中的字段
* 值：
Topic中的字段
删除
添加字段
* 角色：
请选择角色
创建RAM角色
确定
取消

2. 按照界面提示，设置参数。

- 选择操作：选择存储到云数据库(RDS)中。
- VPC实例、MySQL数据库：根据您的业务选择当前区域下的VPC实例和MySQL数据库。

> 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、密码：输入数据库的账号、密码。此账号密码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，仅负责将规则匹配的数据写进数据库中，不会做其他操作。

- 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。
- 字段：此处输入数据表的字段，规则引擎将把处理后的数据存入该字段中。
- 值：此处填写输入数据表字段的值。可以使用转义符`$`，格式为`${key}`，即提取Topic中Key对应的Value值作为输入值。

  例如，当规则引擎的SQL为：`SELECT key FROM mytopic`，且RDS数据库有一张表，表中有`tem`字段，类型是`String`时，

  控制台配置，请在字段处填入RDS数据表的字段`tem`，值处填入规则引擎筛选出来的JSON字段`${key}`。

> 说明:
> - 值处需使用`${}`，否则填入表中的将是一个常量。
> - 字段与值的数据类型需保持一致，否则无法存储成功。

3. 配置完成后，规则引擎为了连接RDS，会在RDS的白名单中添加下列IP。若这些IP未出现，请手动添加。

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

   RDS控制台的白名单示例如下：

2.2.6 数据转发到MNS

您可以使用规则引擎，将处理后的数据转发到消息服务（MNS）中。两者结合使用，实现设备端与服务端之间高性能的消息闭环传输。

数据转发流程

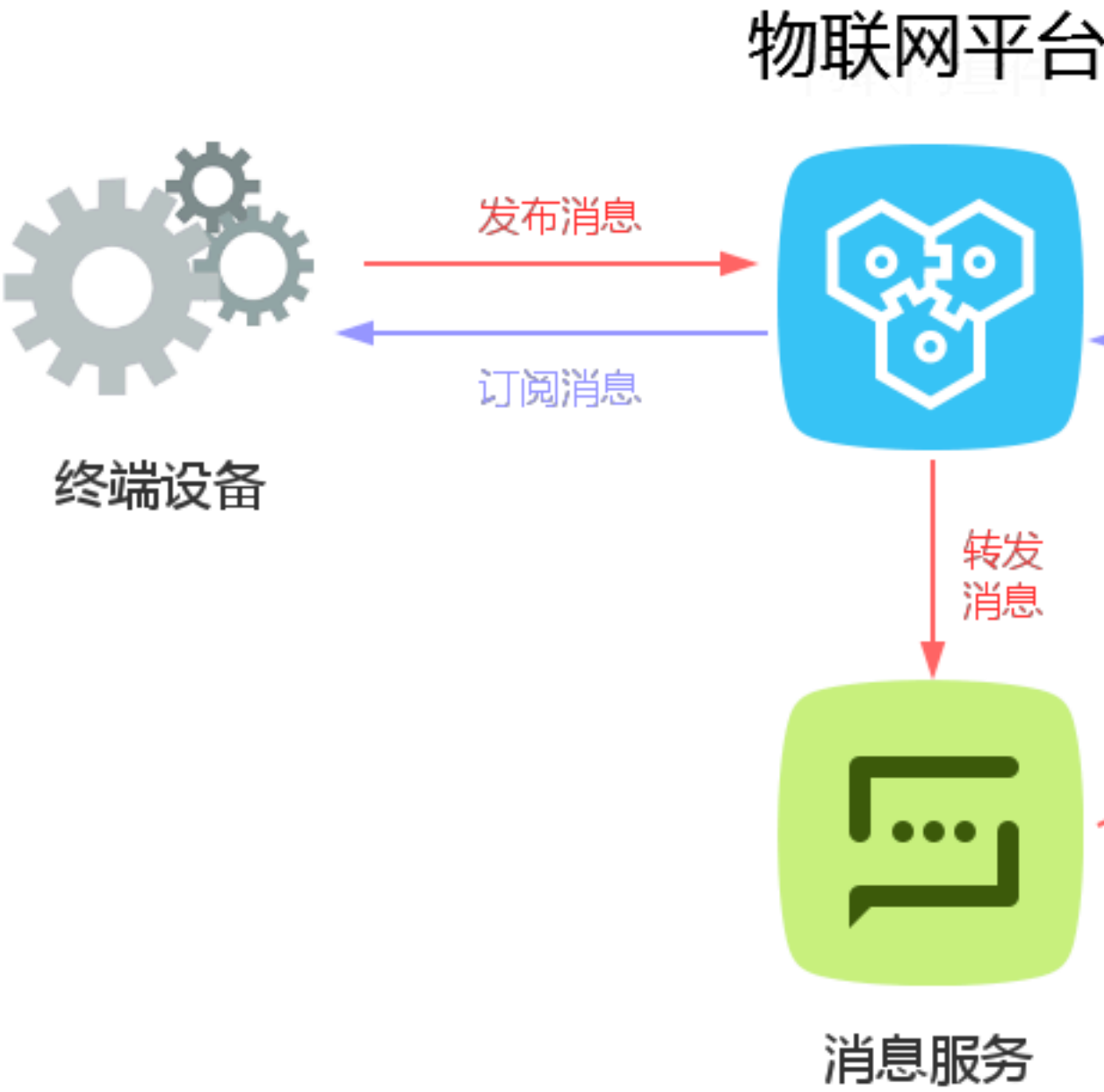
- 设备发送数据到服务端

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

  这种方式优势是MNS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

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



操作步骤

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

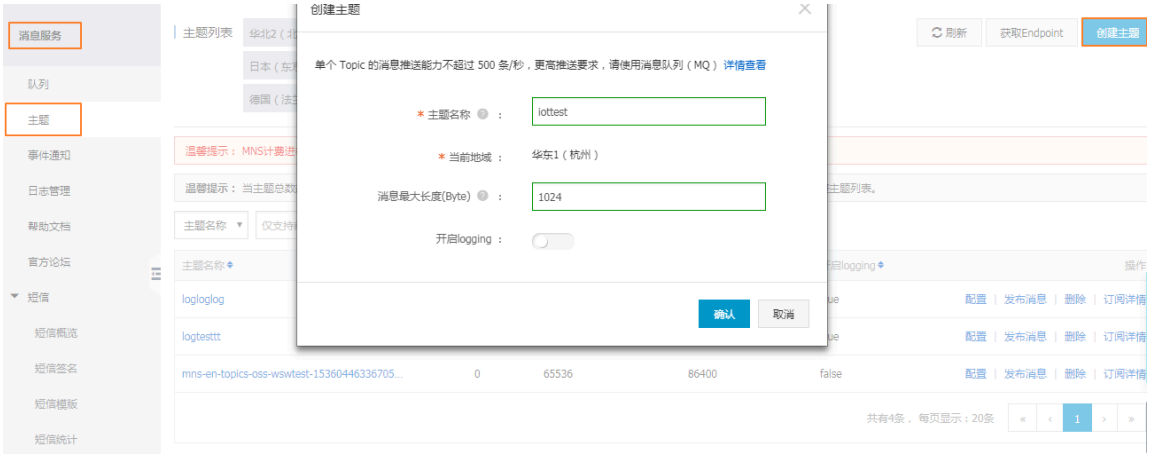
   物联网平台必须经过您的授权才能对您的消息服务主题写入数据。所以，您需要创建一个具有消息服务写入数据权限的角色，然后将该角色授予给物联网平台。这样规则引擎才能将处理过后的数据写入消息服务中。

   如需帮助，请参见管理 RAM 角色。

2. 在消息服务控制台，创建接收消息的主题。

   a. 在消息服务控制台中，单击主题 > 创建主题。

   b. 在创建主题对话框中，输入主题信息，然后单击确定。

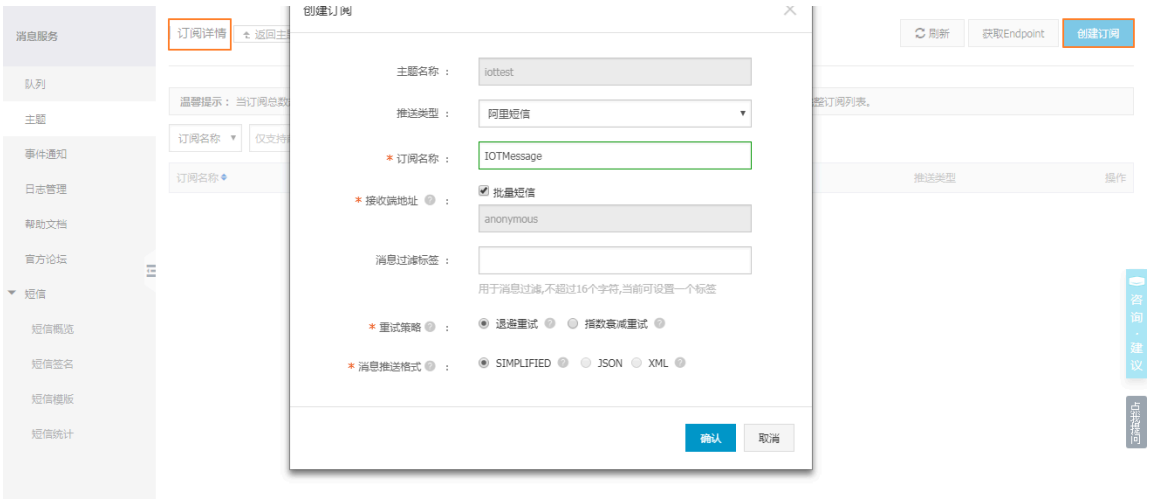


   c. 在主题列表，找到创建的主题，单击其对应的订阅详情操作按钮。

   d. 在订阅详情页，单击创建订阅。

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

   您可以根据自己的业务需求创建多个订阅者。



   关于消息服务的使用方法，请参见消息服务产品文档

3. 在物联网平台控制台规则引擎页，单击创建规则，然后创建一个规则。

4. 在规则引擎页，单击新建规则对应的管理操作按钮。

5. 在规则详情页，为该规则编写用于处理数据的SQL。如需帮助，请参见设置规则引擎和SQL表达式。

6. 在规则详情页，单击数据转发栏对应的添加操作。

7. 在添加操作对话框中，输入数据转发目标的消息服务主题信息。

   参数说明：

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

8. 返回规则引擎页面，单击该规则对应的启动操作按钮启动规则。

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

2.2.7 数据转发到TSDB

您可以配置规则引擎将处理过的数据转发到时序时空数据库（*TSDB*）的实例中。

使用须知

- 目前转发至TSDB仅发布在华东2节点。
- 只支持专有网络下TSDB实例。
- 只支持同region转发。例如：华东2节点的设备数据只能转发到华东2的TSDB中。
- 只支持JSON格式数据转发。

准备工作

在设置转发之前，您需要参考设置数据流转规则编写SQL完成对数据的处理。

操作步骤

1. **单击数据转发一栏的添加操作，出现添加操作页面。选择存储到时序时空数据库(TSDB)中。**

2. 按照界面提示，设置参数。

- 选择操作：此处选择存储到时序时空数据库(TSDB)中。
- 地域、实例：选择处理后的数据将要存入哪个数据库实例中。
- timestamp：此处的值必须为Unix时间戳，如1404955893000。有如下两种配置方式：
  - 使用转义符`${}`表达式，例如`${time}`。此时timestamp的值为指定Topic对应数据包中time字段对应的值。建议使用此方式。
  - 使用规则引擎函数`timestamp()`，此时timestamp的值为规则引擎服务器的时间戳。
- tag：必须使用常量配置，值有三种配置方式：
  - 使用转义符`${}`表达式，例如`${city}`，此时值为指定Topic对应数据包中city字段对应的值。建议使用此方式。
  - 使用规则引擎函数规定的一些函数，例如`deviceName()`，此时值为设备名称。
  - 使用常量配置，例如beijing，此时值为beijing。
- 授权：勾选同意物联网平台向TSDB写数据。此时，规则引擎会向时序时空数据库实例中添加网络白名单，用于IoT访问您的数据库， 请勿删除这些IP段。

示例

规则引擎的SQL：

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

配置的规则如操作步骤所示。

发送的消息：

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

规则引擎会向时序时空数据库中写入的两条数据：

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

```
数据: timestamp:1513677897, [metric:power value:93.0]
```

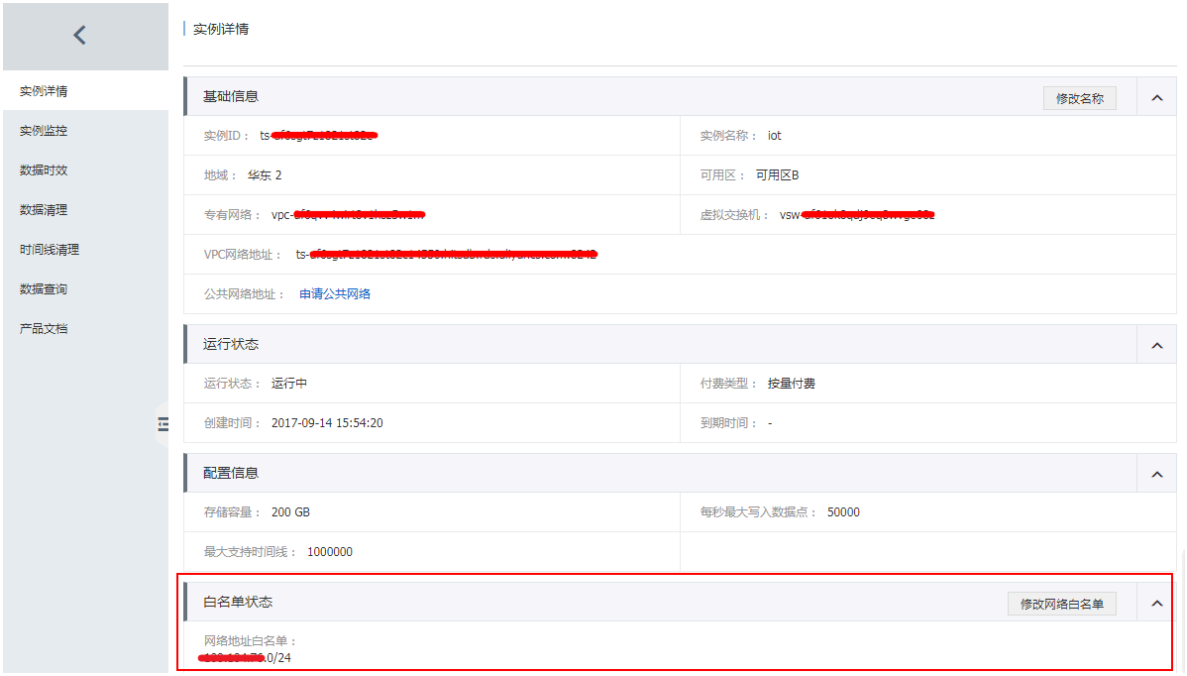
```
tag:device=myDevice,product=bikes,cityName=beijing
```

注意

- 发送的消息中除了在规则引擎中配置为timestamp或者tag值的字段外，其他字段都将作为metric写入时序时空数据库。如上例所示除time和city以外，distance和power作为两个metric写入数据库。
- metric的值只能为数值类型，否则会导致写入数据库失败。
- 用户要保证在规则引擎中配置的tag-值键值对能够获取到，如果获取不到任意一个tag-值键值对，会导致写入数据库失败。
- tag-值键值对限制最多输入8个。
- metric、tag和tag值只可包含大小写英文字母、中文、数字，以及特殊字符`-_./():,[]='`
- 规则引擎在TSDB的白名单中添加以下IP段用以访问数据库。若IP未出现，请手动添加。

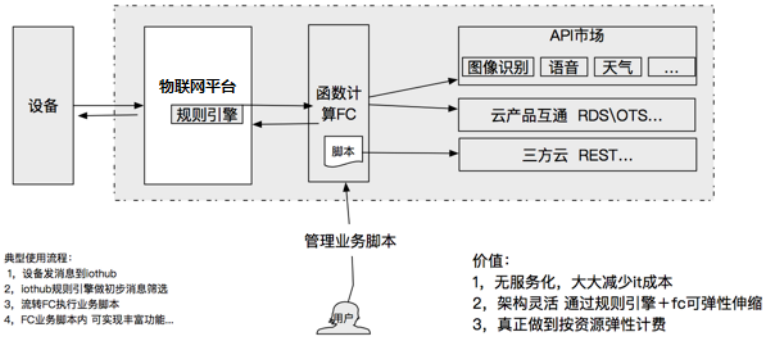
  华东2：100.104.76.0/24

  TSDB控制台白名单示例如下：



2.2.8 转发数据到函数计算

您可以使用规则引擎，将处理后的数据转发至函数计算（Function Compute）中。



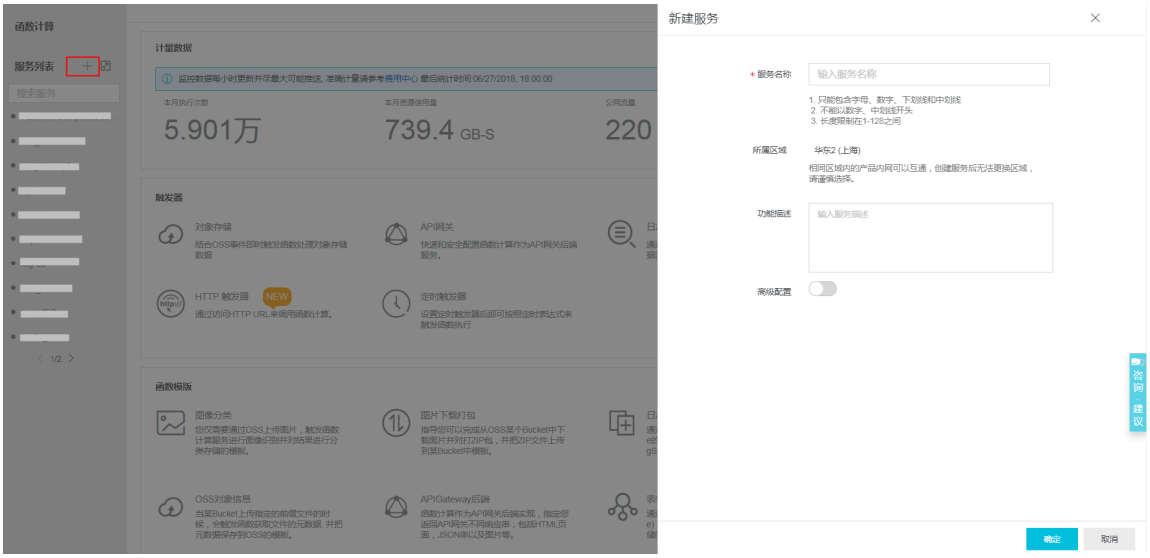
操作流程概述：

1. **函数计算控制台创建好服务、函数。**
2. **创建规则，将IoT数据处理并转发到函数计算中，启动规则。**
3. **使用配置了规则的Topic发送一条消息。**
4. **查看函数计算服务实时监控大盘查询函数执行情况，或者根据函数的具体业务逻辑查看结果是否正确。**

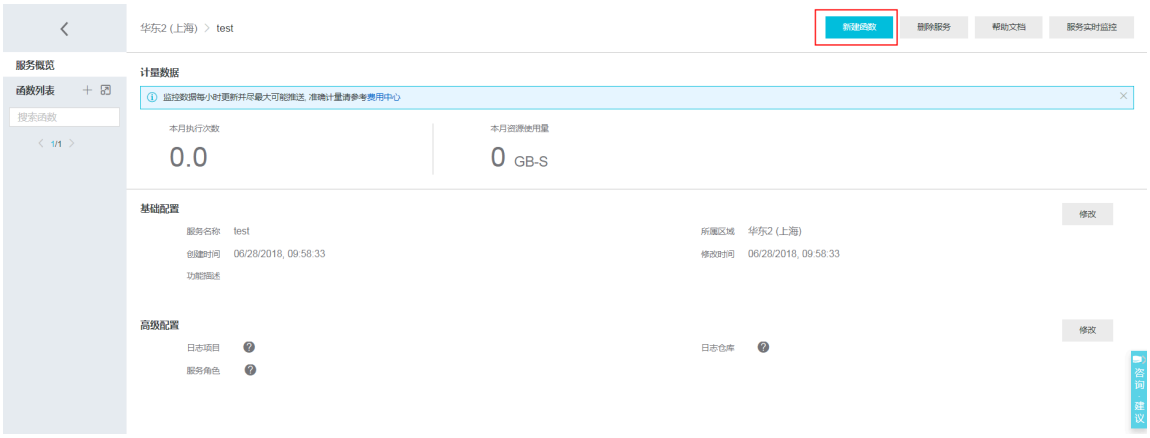
操作步骤

1. 登录函数计算控制台，创建服务与函数。

   a. 创建服务。其中，服务名称必须填写，其余参数请根据您的需求设置。



   b. 创建服务成功后，创建函数。



   c. 选择函数模板，此处以空白函数模板为示例。

d. 设置函数参数。

此处设置函数的逻辑为直接在函数计算中显示获取的数据。

其中，

所在服务：选择*1.a*中创建的服务。

函数名称：设置您的函数名称。

运行环境：设置函数运行的环境，此示例中选择java8。

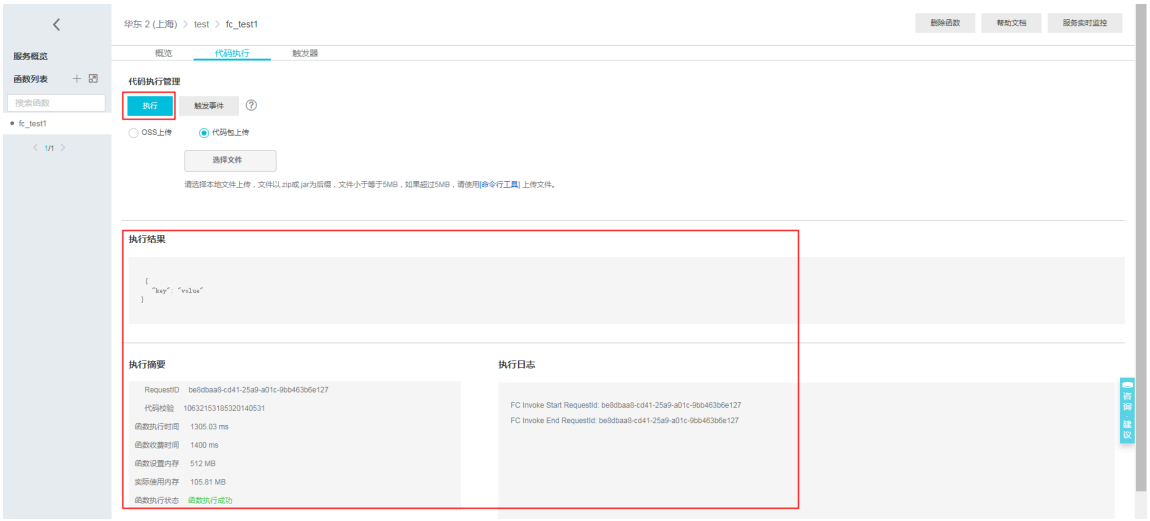
代码配置：上传您的代码。

函数入口：设置您的函数在函数计算系统运行的调用入口，此示例中必须设置为`com.aliyun.fc.FcDemo::handleRequest`。

其余参数的值请根据您的需求，参见函数计算设置。

e. 函数执行验证。

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

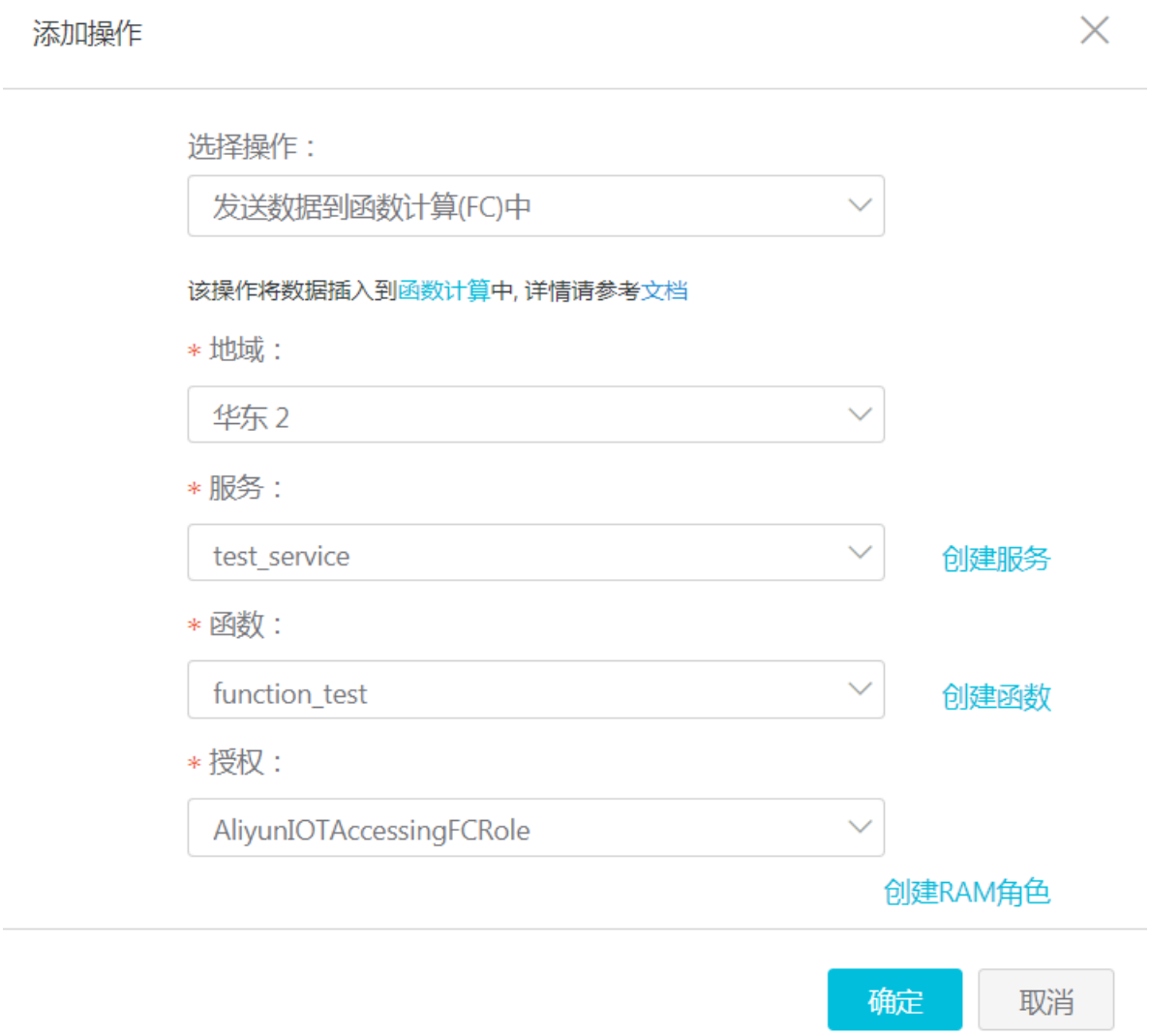


2. 函数正常执行后，配置IoT规则引擎。
3. 在设置转发之前，您需要参考设置规则引擎编写SQL完成对数据的处理。

> **说明:**
> JSON格式和二进制格式都支持转发到函数计算中

4. 单击规则名称，进入规则详情页面。

5. 单击数据转发一栏的添加操作，并在弹出的添加操作页面中设置参数。



- 选择操作：发送数据到函数计算(FC)中。
- 地域：根据您的业务选择将处理后的数据转发至某个地域。如果该地域没有对应资源，需要到函数计算控制台创建相应的资源。

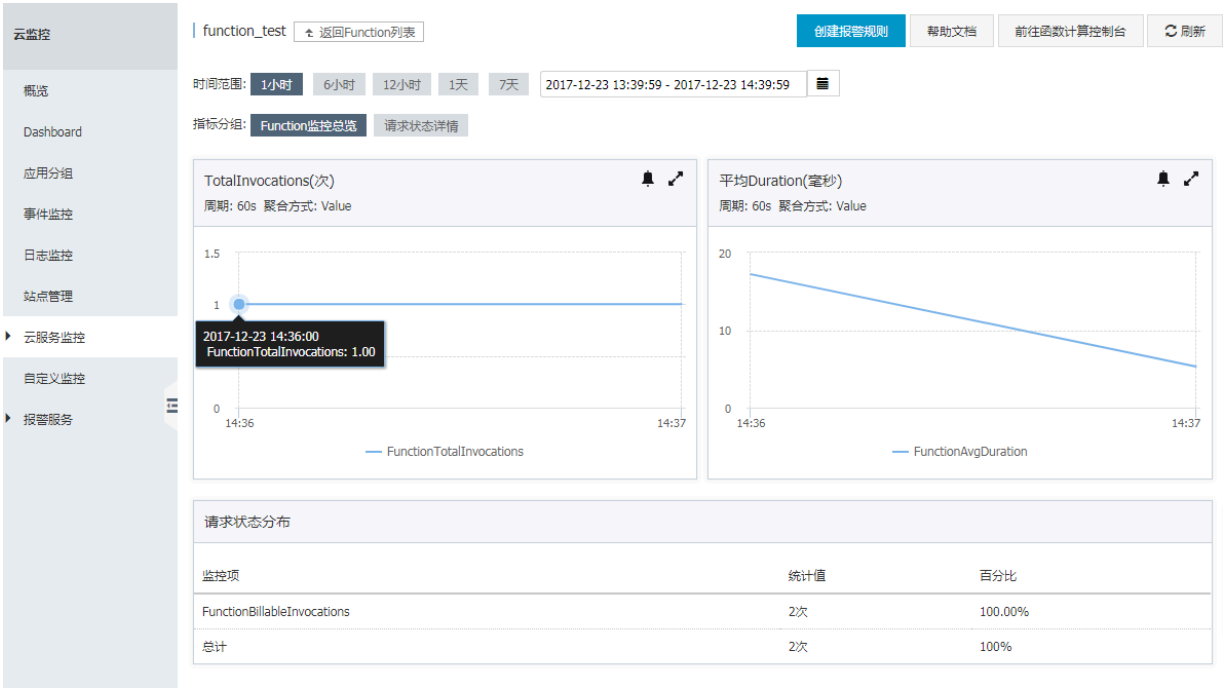
> **说明:**
> 目前支持转发数据至函数计算的地域包括：华东2（上海）、新加坡和日本（东京）。

- 服务：根据您的地域选择服务。若没有服务，请单击创建服务。
- 函数：根据您的地域选择函数。若没有函数，请单击创建函数。
- 角色：授权物联网平台操作函数的权限。此处需要您先创建一个具有执行函数权限的角色，然后将该角色赋予给规则引擎。

6. 启动规则。规则运行后，IoT将根据编写的SQL，将处理后的数据转发到函数计算中。函数计算根据您函数定义的逻辑，直接将接收到的数据显示在函数计算控制台上。

结果验证

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



2.3 场景联动

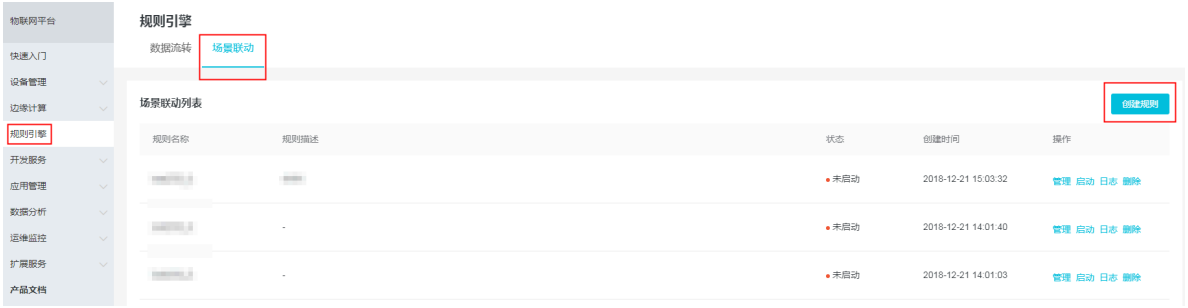
2.3.1 云端管理场景联动

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，经过过滤条件过滤，执行预定的业务逻辑，输出数据到设备或者其他规则，实现海量设备根据场景的联动。

目前场景联动只支持高级版产品。

操作步骤

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



3. 设置参数，然后单击确认。

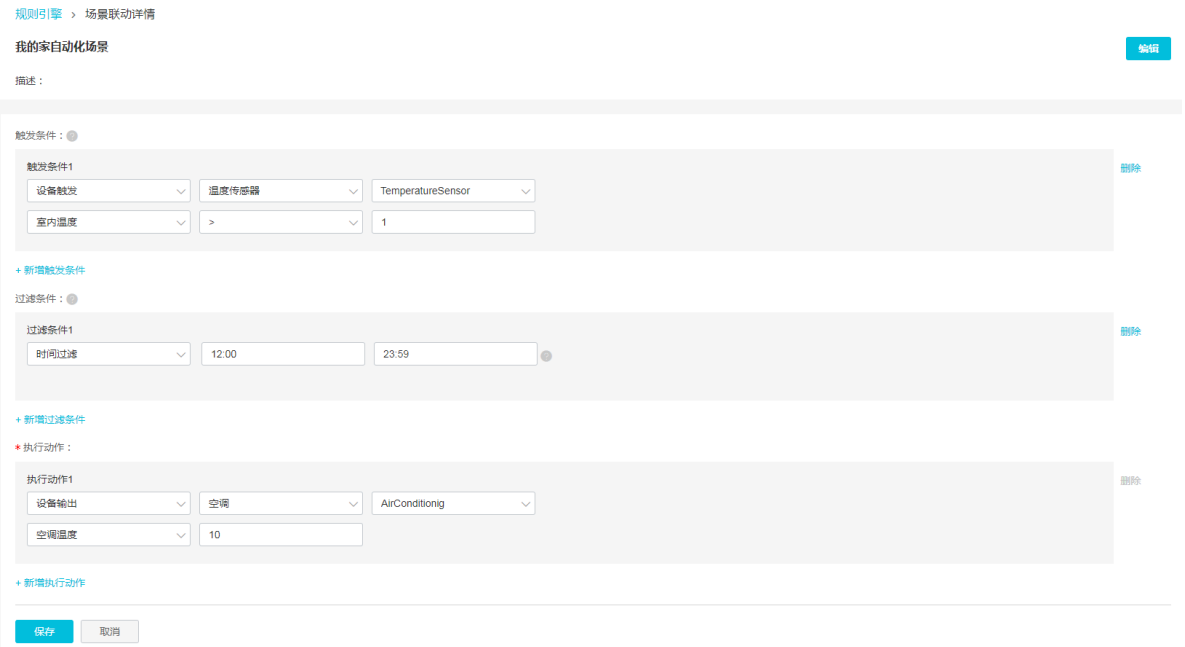
参数	描述
规则名称	设置具体规则的名称。支持中文、英文字母、数字、下划线和短划线，长度限制 1 ~ 30个字符，中文字算两位字符。
规则描述	为规则添加描述，可以为空。

4. 完成场景联动的创建后在弹窗中单击前往编辑，管理配置场景联动。

   您也可以在场景联动名称右侧单击管理，管理配置场景联动。

   以空调设备自动化为例：在 12:00 至 23:59 之间，当温度传感器上报的室内温度高于1摄氏度时，空调设备执行控制室内温度为10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
触发条件	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条件时，则触发过滤条件判断。可以为一个规则创建多个触发条件，触发条件之间是或（or）关系。 · 设置为设备触发，则需选择已创建的产品名称、设备名称、和设备属性或事件。 · 设置为定时触发，则需填写时间点。时间点格式为 cron 表达式。cron表达式的构成：分、小时、日、月、一周内的某天（0或7表示周日，1-6分别表示周一至周六），每项之间用空格隔开。如，每天18点整的cron表达式为：0 18 * * *（其中星号（*）是通配符）；每周五18点整的表达式为：0 18 * * 5。cron表达式具体写作方法，请参见 *CRONTAB* 网页。 上图示例中，设置为设备触发：以温度传感器上报的室内温度高于1摄氏度作为触发条件。

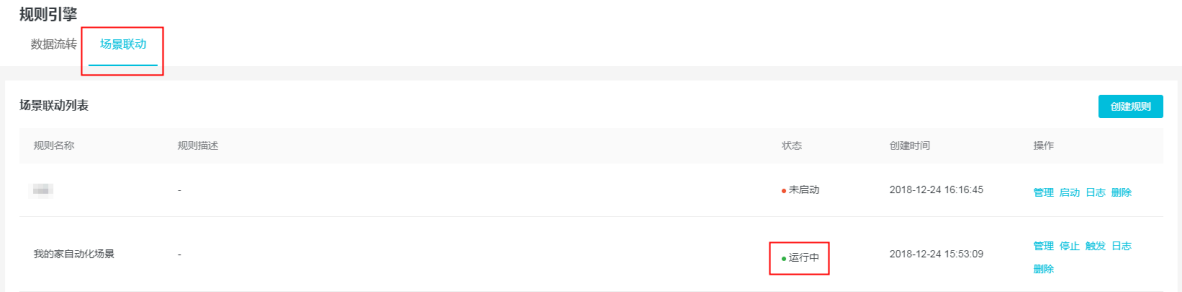
参数	描述
过滤条件	过滤条件集。只有满足过滤条件的数据，才能触发执行动作。可设置为设备过滤或时间过滤。可以为一个规则创建多个过滤条件，过滤条件之间是和（and）关系。 · 设置为设备过滤，则需选择已创建的产品名称、设备名称、和设备功能中的属性或事件。 · 设置为时间过滤，则需设置起始时间和结束时间。 上图示例中，设置为时间过滤：时间在 12:00 至 23:59 之间，则触发执行动作。
执行动作	需执行的动作。可设置为设备输出或规则输出。您可以设置多个动作。某一动作执行失败时，不影响其他动作。 · 设置为设备输出，则需选择已创建的产品名称、设备名称、和设备属性或服务（只有可写的属性或服务才能被设为执行动作）。当触发条件和过滤条件均被满足时，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条件将被跳过，直接进行过滤条件检查。若过滤条件满足，则执行该规则中定义的执行动作。 上图示例中，设置为设备输出：指定的空调设备执行控制温度为10摄氏度。

运行场景联动

场景联动创建成功后，您可在规则引擎 > 场景联动页面中，启动此场景联动。

启动场景联动操作：

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



当设备有数据上报，并且上报数据满足触发条件时，该场景联动便会在云端运行。

若要使场景联动在边缘实例节点中运行，您需将场景联动部署到实例中。具体操作，请参见部署边缘实例。

查看日志

您可以查看该场景联动的日志，并且可在详情中查看运行结果。

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

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

3. 单击详情，查看该条日志的详情信息。

2.3.2 什么是场景联动

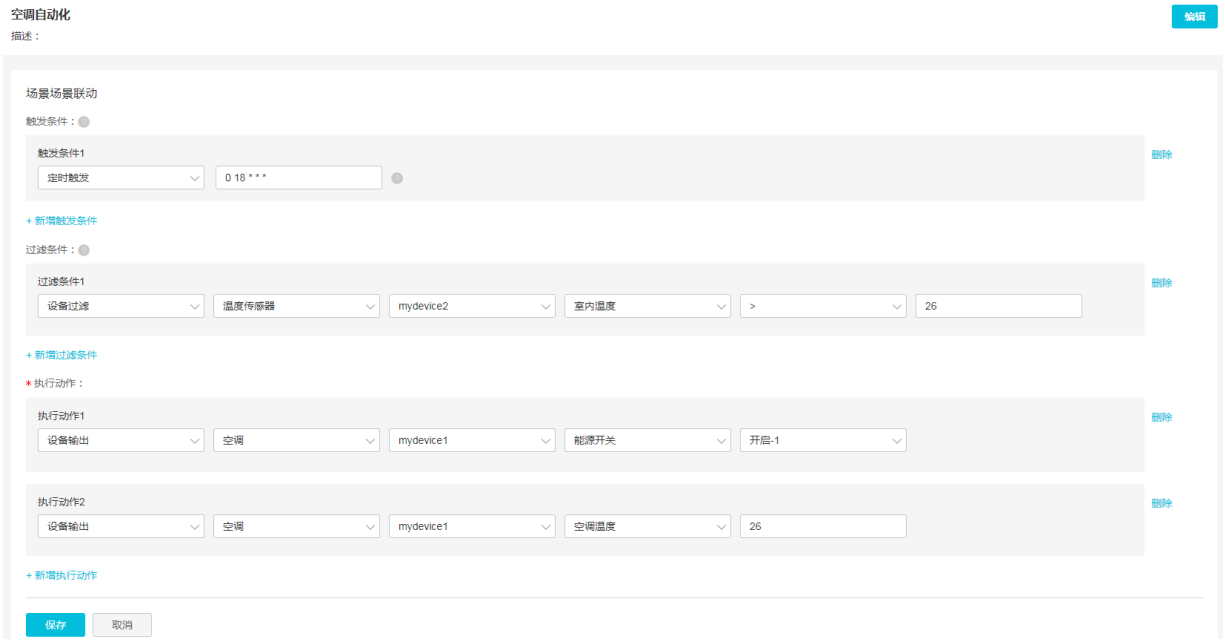
场景联动是规则引擎中，一种开发自动化业务逻辑的可视化编程方式，您可以通过可视化的方式定义设备之间联动规则，将规则部署至云端或者边缘端。

首先，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条件（Trigger）、过滤条件（Condition）、执行动作（Action）三个部分组成。这种规则模型称为 TCA 模型。

当触发条件触发这条场景联动规则后，系统通过判断过滤条件是否已满足，来决定是否执行规则中定义的执行动作。如果满足过滤条件，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
触发条件	定时为每天18:00。时间的cron表达式写作方法，请参见*CRONTAB*网页。
过滤条件	温度传感器探测到室内温度高于 26 摄氏度。
执行动作	空调开关设置为打开；空调控制温度设置为 26 摄氏度。

创建场景联动规则的更多设置说明，请参见云端管理场景联动。

3 监控运维

3.1 在线调试

3.1.1 真实设备调试

设备端开发完成后，您可以使用物联网平台的在线调试功能，从控制台下发指令给设备端进行功能测试。目前仅高级版产品下的设备支持此功能。

操作步骤

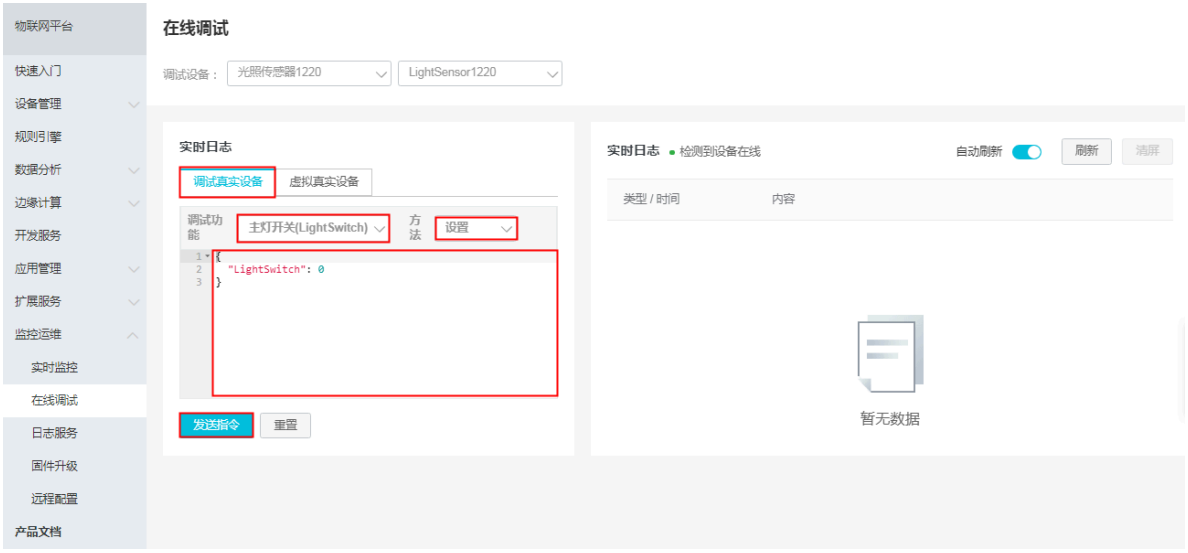
1. 在物联网控制台，左侧导航栏选择监控运维 > 在线调试。
2. 在在线调试页，选择本次调试的设备。

   选择设备后，页面会自动跳转至调试设置页。

3. 选择调试真实设备。
4. 选择要调试的功能。

> **说明:**
>
> · 选择调试属性时，需选择方法为设置或获取。

· 选择调试事件时，需选择方法为获取。



5. 发送调试指令。

· 设置属性：在空白栏，输入属性参数，格式为`{"Yourpropertyidentifier": xxxx }`，然后单击发送指令。属性设置结果，可查看设备实时日志。

· 获取属性：单击发送指令，空白栏处显示设备最新上报的属性内容。

· 调用服务：在空白栏，输入服务入参，格式为`{"Yourserviceinputparam": xxxx}`，然后单击发送指令。服务调用结果，可查看设备实时日志。

· 获取事件：单击发送指令，空白栏处显示设备设备最新上报的事件内容。

3.1.2 虚拟设备调试

物联网平台提供虚拟设备功能，供云端应用开发测试使用。目前仅高级版支持该功能。

背景信息

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

使用限制:

· 连续推送的最小时间间隔为1秒。

· 最多连续推送1000条消息。

· 每天最多可使用100次推送按钮推送调试信息。

操作步骤

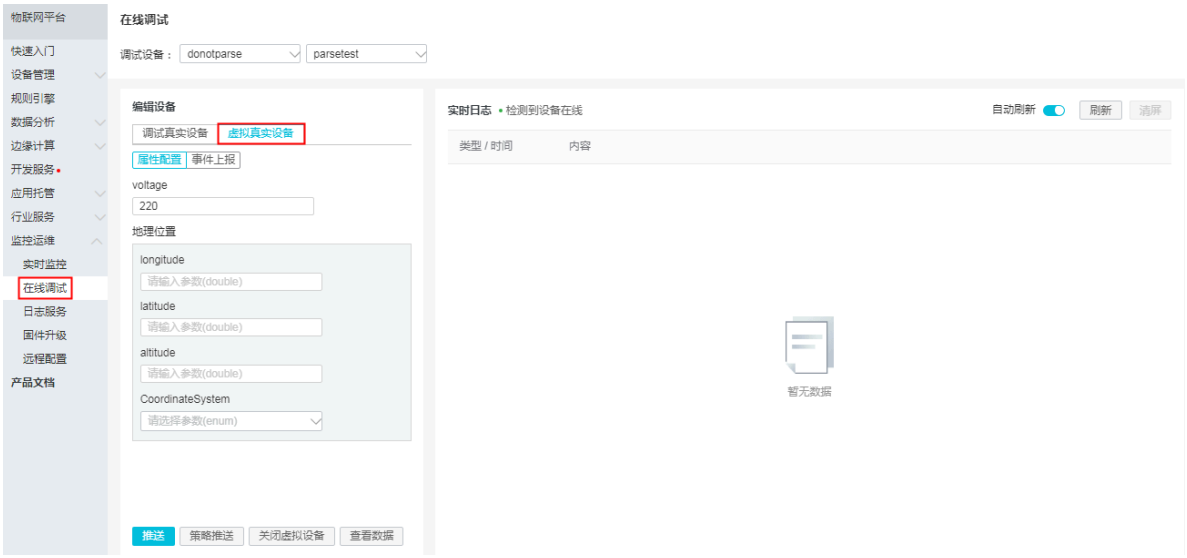
1. 登录物联网平台控制台。

2. 在左侧导航栏，选择监控运维 > 在线调试。
3. 在在线调试页，选择本次调试的设备。

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

   说明:
   当真实设备在线或被禁用时，虚拟设备将会启动失败。
5. 设置模拟推送的内容。可模拟推送属性和事件。

   例如，下图示例中，推送属性配置，`Voltage` 属性值设置为`220`。



6. 选择推送方式。
   - 推送：仅推送一次。
   - 策略推送：
     - 定时推送：在设置好的时间推送数据。
     - 连续推送：在设置好的时间段内，按照固定时间间隔，推送数据。时间间隔单位为秒。
7. 数据推送成功后，单击查看数据，查看结果。

3.2 日志服务

物联网平台提供日志服务功能。您可以在物联网平台控制台日志服务页，查询设备日志。本文主要介绍如何查询设备日志、日志类型和失败日志说明。

查询设备日志

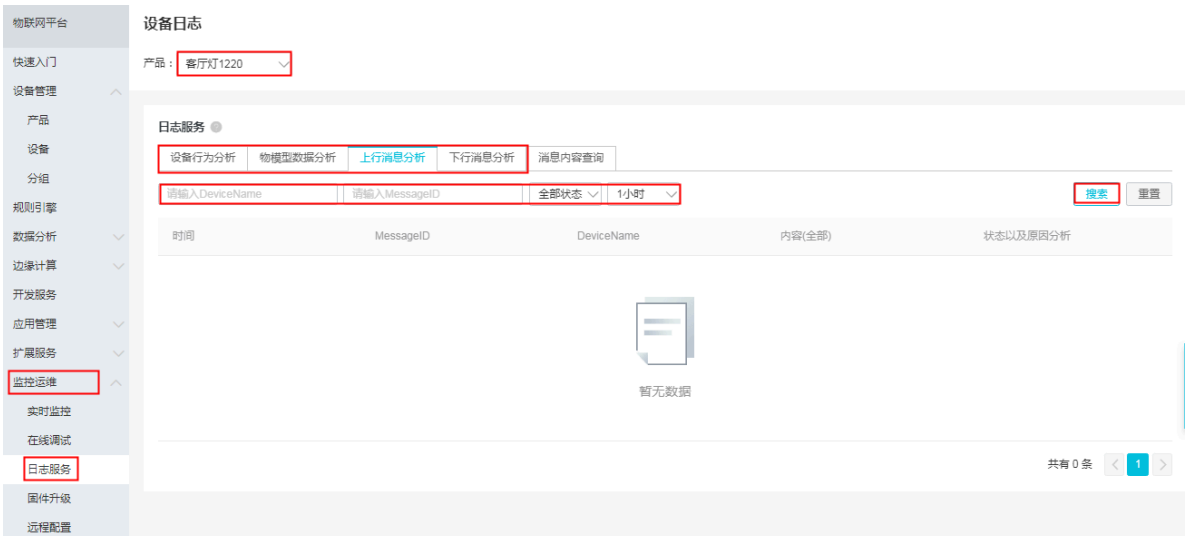
日志类型分为四类：

- 设备行为分析
- 上行消息分析
- 下行消息分析
- 物模型数据分析

其中物模型数据分析为高级版产品特有日志类型，基础版产品只有设备行为分析、上行消息分析、下行消息分析三种日志类型。

查询设备日志途径：

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



物联网平台支持的搜索条件：

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

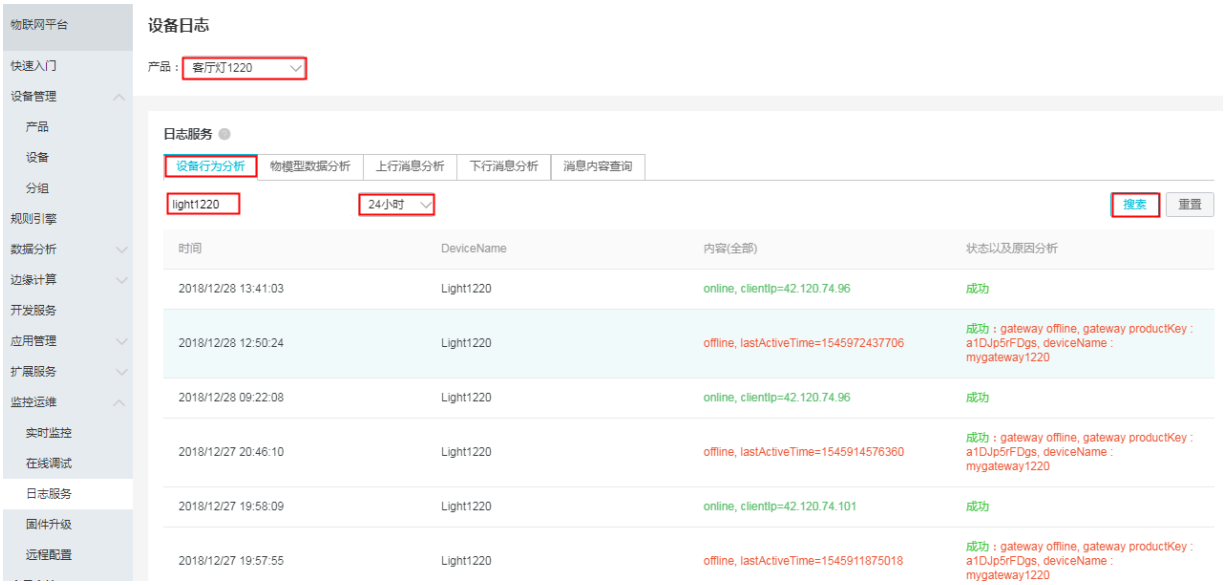
说明：

- 以下章节中，日志说明中的{}符号表示变量，实际日志中会显示为具体的内容。
- 日志内容均为英文。
- 状态为失败的日志，若非system error的原因，都是由于用户使用不当或者产品限制导致的，请仔细排查。

设备行为分析

设备行为主要有设备上线（online）和设备下线(offline)的日志。

可按DeviceName和时间范围来筛选日志。操作界面如下图所示：



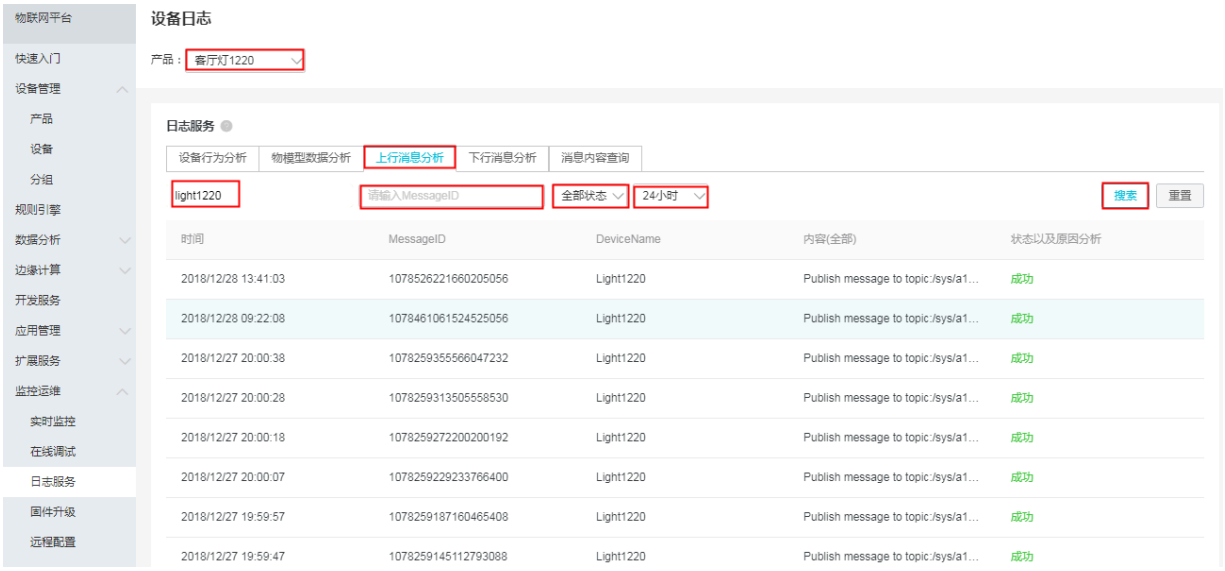
设备离线原因中英文对查表

英文	中文
Kicked by the same device	被使用相同设备证书的设备踢下线。
Connection reset by peer	TCP连接被对端重置。
Connection occurs exception	通信异常，物联网平台服务端主动关闭连接。
Device disconnect	设备主动发送MQTT断开连接请求。
Keepalive timeout	心跳超时，物联网平台服务端关闭连接。
gateway offline	该子设备对应的网关设备已下线。

上行消息分析

上行消息日志主要包括：设备发送消息到Topic，消息流转到规则引擎，和规则引擎转发消息到其他云产品。

可按DeviceName、MessageID（可选）、状态、时间范围来筛选日志。操作界面如下图所示：



上行消息失败日志说明

说明:

下表中包括日志内容（打印的英文日志和中文注释），失败原因（打印的英文日志），失败原因说明。

内容	失败原因	失败原因中文说明
Device publish message to topic:{},QoS={},protocolMessageId:{}（设备发送消息到topic:{},QoS={},protocolMessageId:{}）	Rate limit:{maxQps}, current qps:{}	限流{最大流量}，当前QPS:{}。
	No authorization	未授权。
	System error	系统错误。
	Bad Request	参数错误，例如topic、payload或CoAP协议的option、token等参数传错或没有上传。
send message to RuleEngine，topic:{} protocolMessageId:{}IoT Hub（发送消息给规则引擎, topic:{} protocolMessageId:{}）	{eg，too many requests}	失败信息，例如太多请求导致被限流。
	System error	系统错误。

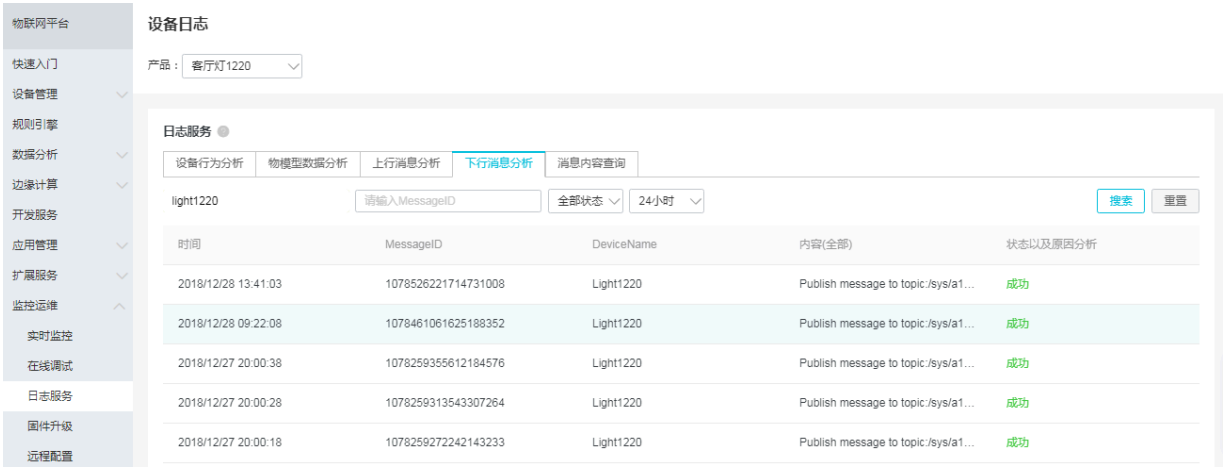
内容	失败原因	失败原因中文说明
Transmit data to DataHub, project:{},topic:{},from IoT topic:{}（规则引擎发送数据到Datahub,project:{}，topic:{},来自IoT的topic:{}）	DataHub Schema:{} is invalid!	DataHub Schema:{}无效，类型不匹配。
	DataHub IllegalArgumentException:{}	DataHub参数异常:{}。
	Write record to dataHub occurs error! errors:[code:{},message:{}]	数据写入到DataHub时出错，错误:[code:{},message{}]。
	Datahub ServiceException:{}	Datahub服务异常:{}。
	System error	系统错误。
Transmit data to MNS, queue:{},theme:{},from IoT topic:{}（发送数据到MNS, queue:{},topic:{},来自IoT的topic:{}）	MNS IllegalArgumentException:{}	MNS参数异常:{}。
	MNS ServiceException:{}	MNS服务异常:{}。
	MNS ClientException:{}	MNS客户端异常:{}。
	System error	系统错误
Transmit data to MQ,topic:{},from IoT topic:{}（规则引擎发送数据到MQ,topic:{},来自IoT的topic:{}）	MQ IllegalArgumentException:{}	MQ参数异常:{}。
	MQ ClientException:{}	MQ客户端异常:{}。
	System error	系统错误。
Transmit data to TableStore,instance:{}, tableName:{},from IoT topic:{}（规则引擎发送数据到TableStore，实例名:{},表名:{},来自IoT的topic:{}）	TableStore IllegalArgumentException:{}	TableStore参数异常:{}。
	TableStore ServiceException:{}	TableStore服务异常:{}。
	TableStore ClientException:{}	TableStore客户端异常:{}。
	System error	系统错误。
Transmit data to RDS, instance:{},databaseName:{},tableName:{},from IoT topic:{}（规则引擎发送数据到RDS,实例名:{},数据库名:{},表名:{}）	RDS IllegalArgumentException:{}	RDS参数异常:{}。
	RDS CannotGetConnectionException:{}	RDS无法连接:{}。
	RDS SQLException:{}	RDS SQL语句异常。
	System error	系统错误。

内容	失败原因	失败原因中文说明
Republish topic, from topic:{} to target topic:{}（规则引擎转发topic,从topic:{}到目标topic:{}）	System error	系统错误。
RuleEngine receive message from IoT topic:{}（规则引擎接收消息，来自IoT的topic:{}）	Rate limit:{maxQps}, current qps:{}	限流{最大流量}，当前QPS:{}。
	System error	系统错误。
Check payload, payload:{}（检测payload,payload:{}）	Payload is not json	Payload的JSON格式不合法。

下行消息分析

下行消息日志主要是云端发送消息到设备的日志。

可按DeviceName、MessageID、执行状态、时间范围来筛选日志。操作界面如下图所示：



下行消息失败日志说明

说明:

其中包括内容（打印的英文日志和中文注释），失败原因（打印的英文日志），失败原因的中文注释。

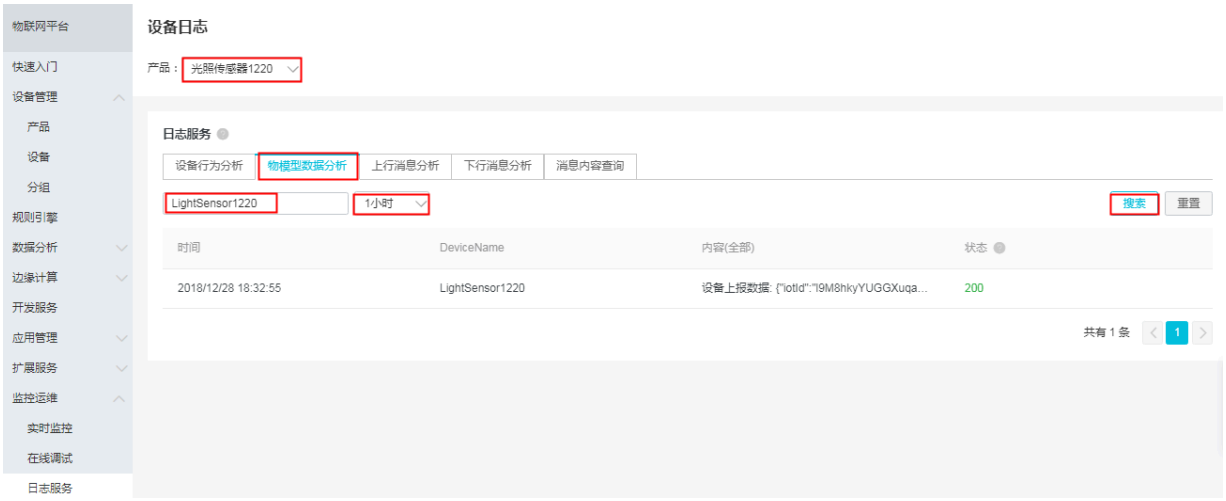
内容	失败原因	失败原因说明
Publish message to topic:{},protocolMessageId:{}（推送消息给topic:{},protocolMessageId:{}）	No authorization	未授权。

内容	失败原因	失败原因说明
Publish message to device, QoS={}（物联网平台发送消息给设备，QoS={}）	IoT hub cannot publish messages	因为服务端没有收到设备的puback，会一直发消息，直到超过50条的阈值。物联网平台就会无法再发送消息。
	Device cannot receive messages	设备端接受消息的通道阻塞，可能由于网络慢或者设备端消息能力不足导致了服务端发送消息失败。
	Rate limit:{maxQps}, current qps:{}	限流{最大流量}，当前QPS:{}。
Publish RRPC message to device（物联网平台发送RRPC消息给设备）	IoT hub cannot publish messages	设备端一直没有回复response，而且服务端一直发送消息，超出阈值导致发送失败。
	Response timeout	设备响应超时。
	System error	系统错误。
RRPC finished（RRPC结束）	{e.g rrpcCode}	错误信息，会打印出相应的RRPC码，比如UNKNOW、TIMEOUT、OFFLINE或HALFCONN。
Publish offline message to device （物联网平台发送离线消息给设备）	Device cannot receive messages	设备端接受消息的通道阻塞。可能由于网络慢或者设备端消息能力不足导致了服务端发送消息失败。

物模型数据分析

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

Alink数据格式产品下的物模型数据分析，可按DeviceName和时间范围来筛选日志。操作界面如下图所示：



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

表 3-1: 日志格式说明

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

参数	说明
upTransformedData	数据格式为透传（自定义）时，上行数据经数据解析后的数据，格式为Alink JSON。

调用服务和设置属性失败日志说明

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

错误码	含义	原因	排查
9201	设备已下线。	设备不在线时，会报这个错。	在控制台，查看设备的在线状态。
9200	设备没有激活。	设备没有在物联网平台激活。新设备注册后需要进行数据上报。	在控制台，查看设备的在线状态。
6208	设备被禁用。	设备被禁用时，属性设置、服务调用被禁止使用。	在控制台，查看设备的状态。如果设备被禁用，启用该设备然后重试操作即可。
6300	TSL校验时，method不存在。	TSL校验时，服务的标识符，没有在设备所属产品中定义，在TSL中不存在。	在控制台产品详情中，查看设备所属产品的功能定义，并与服务的标识符核对，校验标识符中是否包含不可见字符等。
6206	查询服务定义出错。	调用服务时，会查询服务的定义信息，如果服务不存在会报这个错误。	在控制台产品详情中，查看设备所属产品的功能定义，查看传入的服务是否存在。如果存在，请校验传入的参数中是否包含不可见字符。
6200	脚本不存在。	对于透传（自定义）格式的产品，下行服务调用时，会调用产品脚本进行数据的转换。如果脚本不存在，会报这个错误。	在控制台产品详情中，查看产品的脚本是否存在。如果存在，请重新保存脚本后再尝试操作。
6201	脚本转换后数据为空。	脚本执行正常，但是脚本中返回的数据为空。如rawDataToProtocol返回null，protocolToRawData返回null或者空数组。	查看脚本的内容，确认在什么情况下返回数据为空。

错误码	含义	原因	排查
6207	数据的格式错误。	下行同步调用时，或者设备上报数据时出现。 下行同步调用时，可能有如下原因： · 设备返回的数据格式错误。 · 对自定义/透传格式脚本进行解析后的数据格式错误。 · 服务调用传入的参数格式不正确。	参考*API*接口文档及TSL，查看服务需要的数据格式。同时参考*Alink*协议文档，查看对应的数据格式。
系统异常错误码			
5159	获取TSL中属性信息时报错。	系统异常。	可以通过提交工单排查。
5160	获取TSL中事件信息时报错。		
5161	获取TSL中事件信息时报错。		
6661	查询租户信息时异常。		
6205	下行服务调用异常。		

上报属性和上报事件失败日志说明

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

错误码	含义	原因	排查
6106	上报的属性数据过多。	设备一次上报的有效属性个数不能超过200个。	查看设备属性上报的日志，检查上报的属性个数。或者查看设备本地的日志，查看上报的数据。

错误码	含义	原因	排查
6300	TSL校验时，method不存在。	TSL校验时，设备上报的Alink（标准）格式数据，或者自定义（透传）格式数据经过脚本转换后，没有Alink协议要求的method参数。	查看设备属性上报的日志，查看上报的数据。 或者查看设备的本地日志，查看上报的数据。
6320	TSL校验时，属性信息不存在。	查询设备的TSL时，没有查询到设备的属性信息。	在控制台产品详情中，查看设备所属产品的功能定义，查看属性定义是否存在。不存在时，定义相应的属性。
6450	Alink协议格式的数据中method不存在。	设备上报的Alink标准格式数据，或者自定义/透传格式数据经过脚本解析为Alink标准格式数据后无method。	查看设备属性上报的日志，检查设备上报的数据中是否有method参数。或者查看设备本地的日志。
6207	数据的格式错误。	下行同步调用时，或者设备上报数据时出现。 设备上报数据时，可能原因为：设备上报的Alink数据格式，或者调用脚本解析后返回的数据格式不是JSON格式。	参考*Alink*协议文档，查看对应数据格式，并按格式要求上报数据。
系统异常错误码			
6452	限流。	请求量过大，触发限流。	可以通过提交工单排查。
6760	租户的空间存储超出配额。	系统异常。	可以通过提交工单排查。

调用服务和设置属性的reply失败日志说明

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

错误码	含义	原因	排查
6452	限流。	请求量过大，触发限流。 说明: 服务调用非透传格式数据时，可能返回的结果。请经过TSL再次校验。	可以通过提交工单排查。

TSL公共错误码

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

错误码	含义	原因	排查
6321	TSL中，属性对应的标识符identifier不存在。	系统异常。	可以通过提交工单排查。
6317	TSL模型有错误。	系统异常。	可以通过提交工单排查。
6302	参数不存在。	TSL校验服务的入参时，服务要求的参数没有传。	在控制台产品详情中，查看设备所属产品的功能定义。服务查询调用、属性设置，查询对应服务的入参，核对传入的参数。
6306	传入的参数，不符合TSL整形数据的规范。	TSL校验时： · 参数类型，和TSL中定义的类型不一致。 · 传入的参数取值范围不符合功能定义时设置的参数范围。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型。
6307	传入的参数，不符合TSL中32位浮点数据的规范。	TSL校验时： · 参数类型，和TSL中定义的类型不一致。 · 传入的参数取值范围不符合功能定义时设置的参数范围。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型和参数的取值范围。

错误码	含义	原因	排查
6322	传入的参数，不符合TSL中64位浮点数据的规范。	TSL校验时： · 参数类型，和TSL中定义的类型不一致。 · 传入的参数取值范围不符合功能定义时设置的参数范围。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型和参数的取值范围。
6308	传入的参数，不符合TSL中布尔类型数据的规范。	TSL校验时： · 参数类型，和TSL中定义的类型不一致。 · 传入的参数取值范围不符合功能定义时设置的参数范围。	在控制台产品详情中，查看设备所属产品的功能定义和对应服务的入参，核对传入的参数类型。
6309	传入的参数，不符合TSL中枚举类型数据的规范。	TSL校验时，参数类型和TSL中定义的类型不一致。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6310	传入的参数，不符合TSL中字符类型数据的规范。	TSL校验时： · 参数类型，和TSL中定义的类型不一致。 · 传入的字符类型的参数长度超过限制。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6311	传入的参数，不符合TSL中日期类型数据的规范。	TSL校验时： · 传入的参数类型，需要和TSL中定义的类型完全一致。 · 传入的字符类型判断不是UTC时间戳的字符格式时会报错。	在控制台中产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6312	传入的参数，不符合TSL中结构体类型数据的规范。	TSL校验时： · 传入的参数类型，需要和TSL中定义的类型完全一致。 · 结构体类型中参数的个数和TSL中定义不一致时会报这个错。	在控制台产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。

错误码	含义	原因	排查
6304	校验的参数，在TSL结构体中不存在。	TSL校验时，传入的参数在结构体中不存在。	在控制台产品详情中查看设备所属产品的功能定义，对应服务的入参，核对传入的参数类型。
6324	校验参数时，数组类型的参数不符合规范。	TSL校验时： · 传入的数组类型的参数不符合TSL定义时，会报这个错。 · 数组中参数个数超过了TSL中定义的最大个数。	· 在控制台产品详情中，查看设备所属产品的功能定义，检查对应数组的定义。 · 查看设备上报的日志，检查设备上报的数据中数组内元素的个数。
6328	校验参数时，传入的参数不是数组类型。	TSL校验时，传入的参数如果不是数组类型，会报这个错。	在控制台中产品详情中查看设备所属产品的功能定义，查看对应服务的入参，查询类型为数据的参数，然后检查传入的对应参数是否是数组类型。
6325	校验参数时，传入的数组类型参数中的元素类型，目前不支持该类型。	TSL校验参数时报错，数组中元素的类型目前只支持整形、32位浮点类型、64位浮点类型、字符串类型、结构体类型。	检查传入的数组元素类型是否是目前支持的类型。
系统异常错误码			
6318	TSL解析时系统异常。	系统异常。	可以通过提交工单排查。
6329	校验参数时，TSL中数组规范解析出错。		
6323	TSL中参数规范格式错误。		
6316	TSL中解析参数报错。		
6314	TSL不支持的数据类型。		
6301	通过TSL校验参数格式时报错。		
数据解析脚本相关			

错误码	含义	原因	排查
26010	请求过于频繁被限流。	请求过于频繁。	可以通过提交工单排查。
26001	脚本内容为空。	执行脚本时获取脚本内容，不存在。	在控制台查询产品的脚本是否存在。如果存在，则是否正常保存。应是正式的脚本，不是草稿。
26002	脚本执行时异常。	脚本执行正常，但脚本编写有问题，如脚本中语法错误。	在控制台使用相同的参数去执行脚本，查看具体的错误信息，修改脚本。注意控制台只提供了脚本的基础运行环境，并不会对脚本进行详细的校验。建议脚本需在本地经过详细的自验后，再进行保存。
26006	脚本执行时必要的方法不存在。	脚本执行正常，脚本内容有误。脚本编写要求有protocolToRawData和rawDataToProtocol这两个服务，如果执行时不存在，会报错。	在控制台查询脚本的内容，查看protocolToRawData和rawDataToProtocol服务是否存在。
26007	脚本执行时返回的结果格式不正确。	脚本执行正常，但返回的结果不符合格式要求。 脚本编写要求有protocolToRawData和rawDataToProtocol这两个服务。 protocolToRawData返回byte[]数组， rawDataToProtocol要求返回JSON对象。如果脚本返回的结果不符合这两种格式，返回时会报这个错。 如设备上报数据后，会返回结果给设备。返回的结果也会经过脚本进行解析，如果对于返回结果不做处理，可能会返回不符合要求的类型。	在控制台查看脚本，获取脚本内容。按照输入参数，在本地执行脚本并查看返回结果的格式是否符合要求。

消息内容查询

设备发送的payload内容可通过消息内容查询获取。

通过Message ID搜索payload内容。目前，只支持查询QoS1的消息。

显示结果中，您可以选择显示的内容格式为Base64编码或原始数据。

3.3 固件升级

物联网平台提供固件升级与管理服务。首先，设置设备端支持OTA服务。然后，在控制台上传新的固件，并将固件升级消息推送给设备，设备即可在线升级。本文将为您讲解如何设置固件升级和管理固件。

前提条件

固件升级功能前，请确保设备端支持OTA升级服务。

- 如果您使用设备端SDK，请参考*设备OTA升级*。
- 如果您使用AliOS Things，请参考*AliOS Things技术文档*。

操作步骤

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

> **说明:**
> 为提供更好的服务，物联网平台提供了产品版本统计功能，所以对原固件升级进行了全新改版。您首次进入改版后的固件升级页面时，需手动将之前上传的固件与产品关联。固件与产品是一一对应关系，所以您只能为一个固件关联一个产品。详情请参考控制台指引。

3. 在固件升级页，单击新增固件。

> **说明:**
> 一个阿里云账号下最多可有100个固件。

4. 在添加固件对话框中，输入固件信息，并上传固件文件。

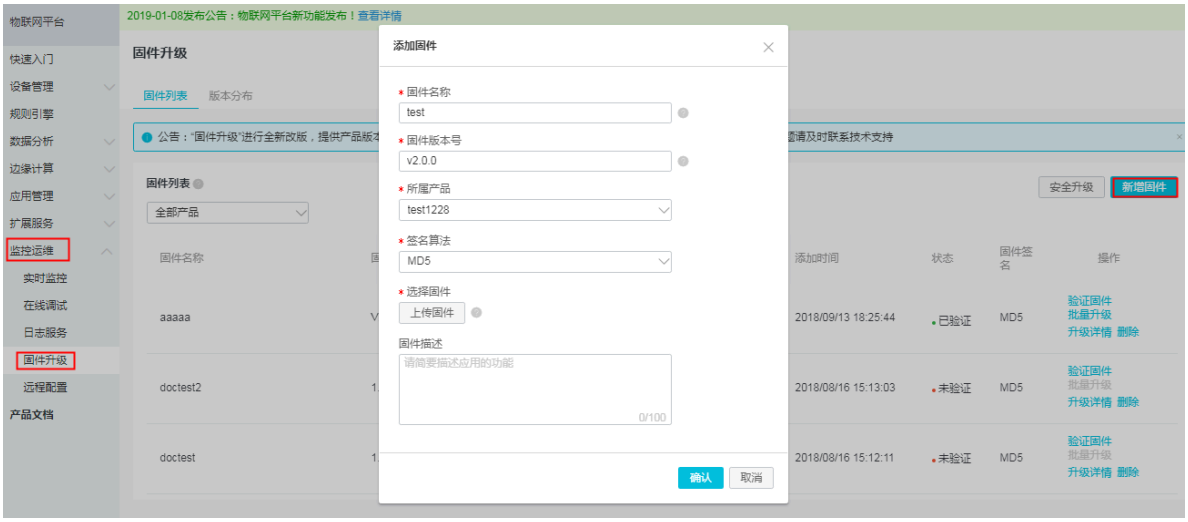


表 3-2: 参数说明

参数	描述
固件名称	设置固件名称。仅支持中文、英文字母、数字和下划线，且不能以下划线开头。长度限制为4~32字符。
固件版本号	输入固件版本号。仅支持英文字母、数字、点号、连字符和下划线。长度限制为1~64字符。
所属产品	选择固件所属产品。
签名算法	仅支持MD5和SHA256。
上传固件	上传固件文件。文件大小不能超过150M，仅支持bin，tar，gz，zip类型的文件。

5. （可选）若您的产品设备使用的是搭载AliOS Things的芯片，可以使用安全升级功能。

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

   a) 在固件升级页，单击安全升级。

   b) 在安全升级开启对话框中，将待升级的AliOS Things产品对应的安全升级按钮置为开。

      当安全升级功能为开时，可单击对应的复制按钮，复制公钥，用于设备端签名。

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

说明:

固件上传至物联网平台后，必须使用少量设备验证固件是否可用。确认测试设备升级成功后，才能用于批量升级设备固件。可以发起多次验证固件。

参数	描述
升级方式	· 整包：将整个升级包推送至设备。 · 差分：物联网平台将提取升级包与前一个版本的差异，仅将差异部分推送至设备。差分升级可有效减少升级对设备资源的占用。 说明: - 仅AliOS Things设备mk3060，esp8266两个型号支持差分升级功能。 - 差分升级的目标固件大小不能小于20 KB。
待升级版本号	下拉选项框中，展示当前产品下所有设备的当前固件版本号。选择一个或者多个待升级的固件版本。 当您选择待升级的版本号后，该固件版本对应的设备将展示在DeviceName的下拉选项中。
DeviceName	选择用于此次测试的设备。
差分的固件名称 说明: 升级方式为差分时的参数。	选择待差分升级的固件名称。

参数	描述
切片大小规格 说明: 升级方式为差分时的参数。	将固件进行切片传输，选择固件切片规格。支持不切片、2 MB、64 KB、32 KB四种规格。

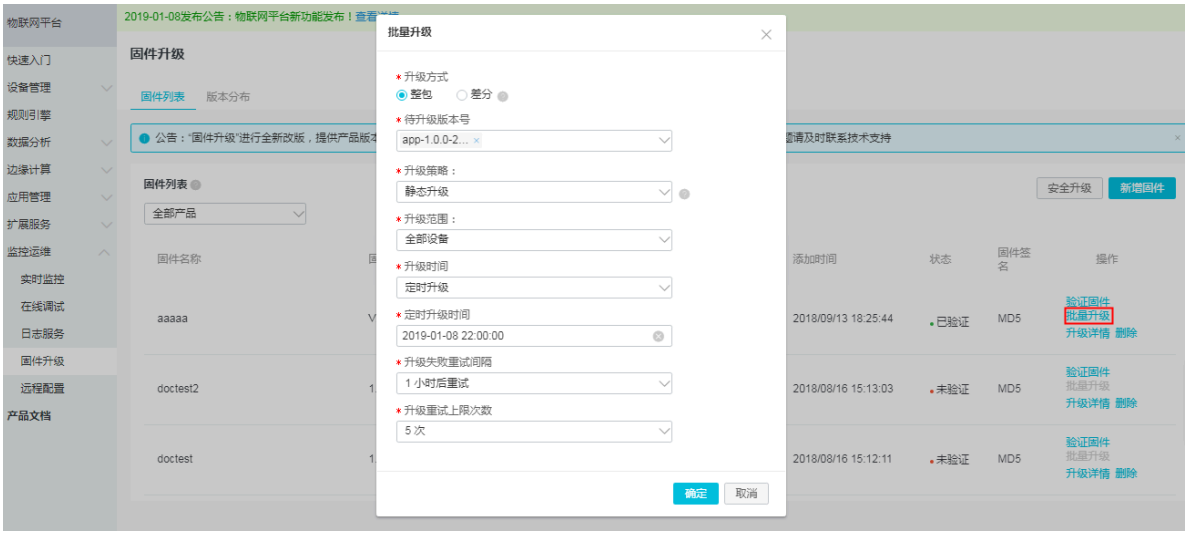
说明:

- 设备接收固件升级通知:
  - 通过MQTT协议接入物联网平台的设备，在线时可以立即接收到升级通知；不在线的设备下次接入时，系统会再次推送升级通知。
  - 其他接入方式（如CoAP或者HTTPS）的设备都是短连接，在线时可以立即收到升级通知；不在线时，无法收到通知。
- 只要您进行了固件验证操作，固件状态都会从未验证变为已验证，与设备的实际升级结果无关。单击对应固件的升级详情查看设备固件是否升级成功。

7. 固件验证可用后，可单击对应固件的批量升级，设置参数，向批量设备定向推送升级通知。

说明:

批量升级前，请确保该固件已验证可用。



参数	描述
升级方式	· 整包：将升级包整包推送到设备端进行升级。需设置待升级的固件版本号。 · 差分：将升级包与某一待升级固件间的差异提取出来，仅将差异部分推送至设备端进行升级。需设置待升级的固件版本号、差分的固件名称和切片大小。
待升级版本号	下拉选项框中，展示当前产品下所有设备的当前固件版本号。选择待升级的固件版本。
差分的固件名称 说明: 升级方式为差分时的参数。	待差分升级的固件名称。
切片大小规格 说明: 升级方式为差分时的参数。	将固件进行切片传输，选择固件切片规格。支持四种规格: · 不切片 · 2 MB · 64 KB · 32 KB
升级策略	· 静态升级：仅升级满足指定条件的当前设备。 · 动态升级：满足指定条件的设备都将收到升级通知。动态升级将持续维护需升级的设备范围，包括当前已经上报版本号的设备和新激活的设备。

参数	描述
升级范围	· 全部设备：升级该产品下全部的设备。 · 定向升级：选择定向升级后，在侧边弹出窗口中，选择要升级的设备。仅升级被选中的设备。 说明: 定向升级的待升级版本为多选。默认选中您之前已输入的待升级版本号。如果您未设置待升级版本号，则默认选中全部版本。 · 区域升级：升级实际地理位置在指定区域的设备。
升级时间	指定设备固件升级的时间。 · 立即升级：设置完成后立即进行固件升级。 · 定时升级：需设定升级时间，等到指定时间自动进行升级。定时时间范围是5分钟~7天。 说明: 仅升级策略为静态升级时，支持定时升级。 设置定时升级时间后，在升级详情页的待升级页签下看到设备的待升级状态为：定时待升级（定时：xxxx-xx-xx xx:xx:xx）。
升级失败重试间隔	如果升级失败，在什么时候进行重试升级。可选： · 不重试 · 立即重试 · 10分钟后重试 · 30分钟后重试 · 1小时候重试 · 24小时候重试
升级重试上限次数	选择升级失败后，最多可重试几次。可选： · 1次 · 2次 · 5次

预期结果

单击该固件的升级详情查看升级状态。

· 待升级：已选中的待升级设备。两种待升级状态：离线待升级和定时待升级（定时：xxxx-xx-xx xx:xx:xx）。
  - 如果设备既是离线状态又是定时升级状态，显示为定时待升级；
  - 如果当定时时间到，启动升级任务时，设备为离线状态，则显示为离线待升级。

- 升级中：设备收到升级通知并上传升级进度。如果设备没有返回任何进度信息，则升级进度显示为0。
- 升级成功：本次升级成功的设备。
- 升级失败：本次升级失败的设备及简要的升级失败原因。以下原因可能造成设备升级失败：
  - 待升级的设备中有些设备还未结束上一次的升级动作。这部分设备升级失败。等设备完成升级动作后，可尝试再次升级。
  - 设备在实际升级过程中出现如下载失败、校验失败、解压失败等错误。可以尝试再次升级。

固件升级成功后，您可以单击固件升级页的版本分布页签，选择具体产品，查看固件版本分布状态。

- 固件版本分布：该产品下所有设备固件版本占比情况。展示前5个占比最高固件名和版本号，其余固件归属为“其他”。
- 固件版本占比：该产品下所有设备固件版本占比情况。
- 设备列表：该产品下所有设备。可按照不同固件版本进行查询。

3.4 远程配置

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

前提条件

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

远程配置说明

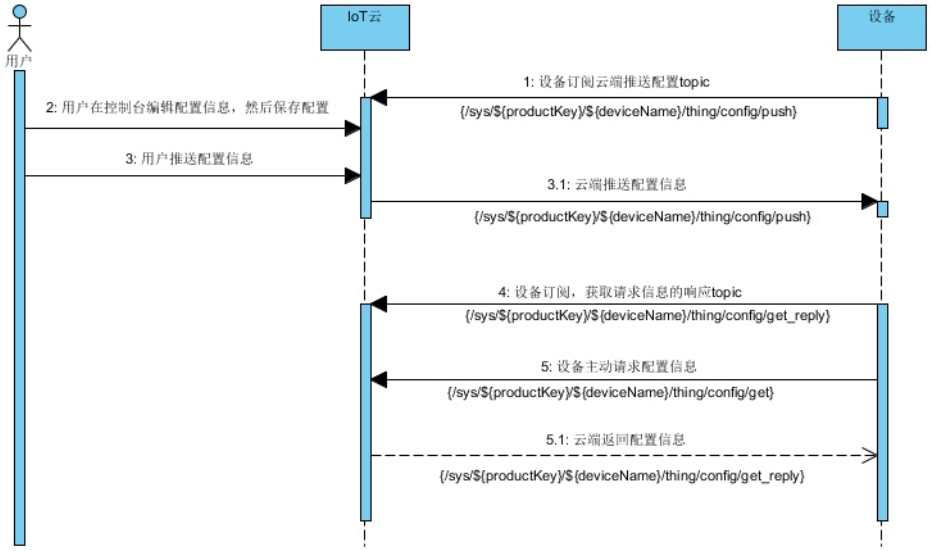
很多场景下，开发者需要更新设备的配置信息，包括设备的系统参数、网络参数、本地策略等。通常情况下，是通过固件升级的方式更新设备的配置信息。但是，这将加大固件版本的维护工作，并且需要设备中断运行以完成更新。为了解决上述问题，物联网平台提供了远程配置更新的功能，设备无需重启或中断运行即可在线完成配置信息的更新。

物联网平台提供的远程配置功能，支持：

- 开启/关闭远程配置。
- 在线编辑配置文件，并管理版本。

- 批量更新设备配置信息。
- 设备主动请求更新配置信息。

远程配置流程图如下：



远程配置大致分为三部分：

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

远程配置操作流程

远程配置使用分为两种场景，一种是云端推送配置信息给设备端，一种是设备端主动请求配置信息。根据场景的不同，远程配置的步骤也有所区别。

场景一：云端推送配置信息给设备端

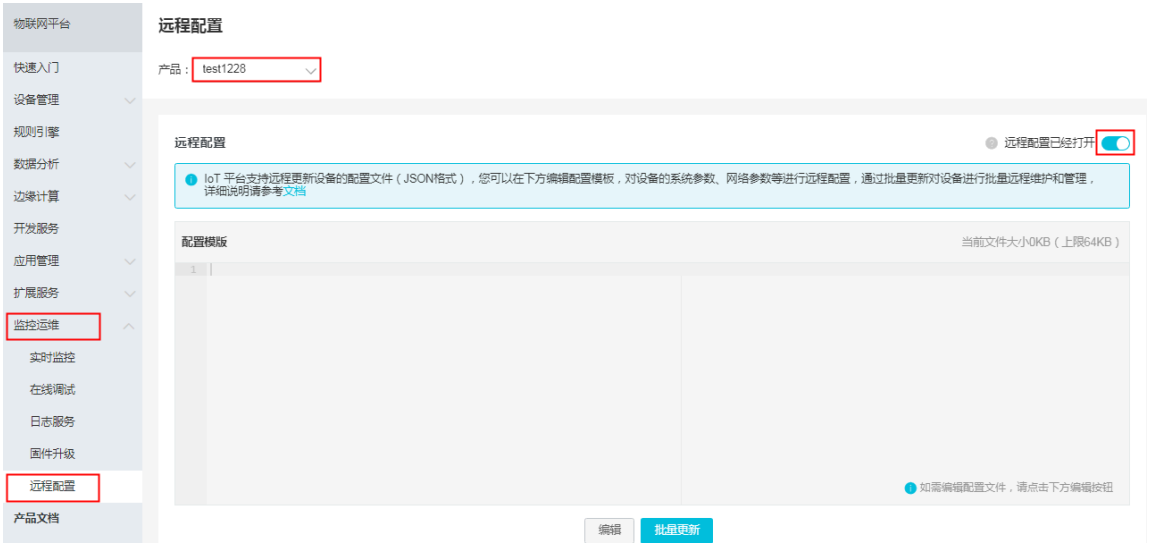
在物联网平台控制台，向某一产品下的所有设备批量推送配置文件。

1. 设备上线，并订阅推送配置信息的Topic `/sys/${productKey}/${deviceName}/thing/config/push`。

2. 在物联网平台控制台中，编辑配置文件。

a. 登录物联网平台的控制台，选择监控运维 > 远程控制。

b. 选择产品，打开远程配置开关。



说明:

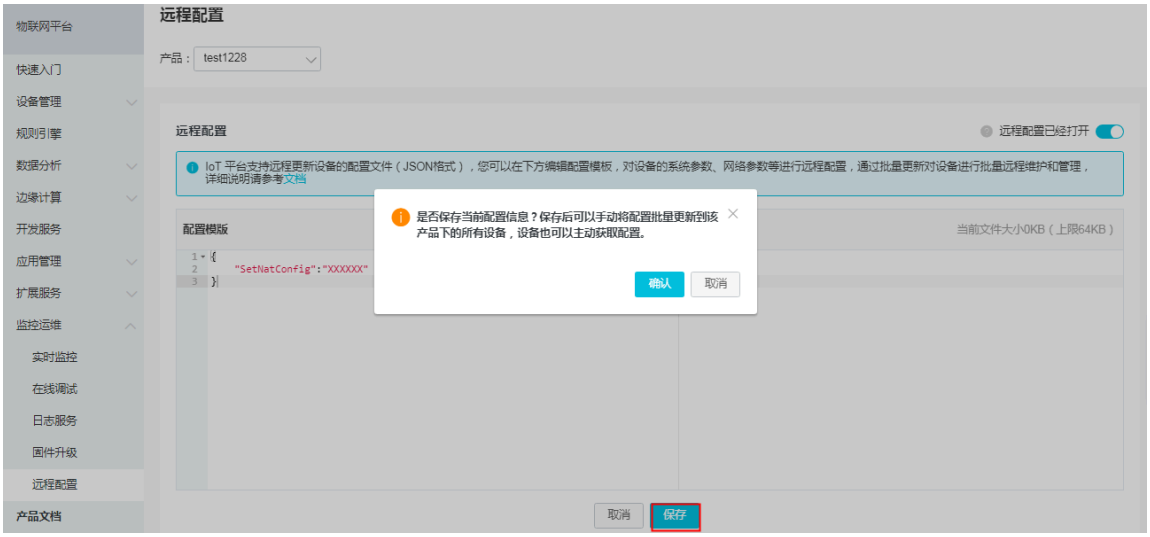
- 必须开启产品的远程配置功能后，才可以编辑配置信息。
- 切换为关闭状态，即关闭该产品的远程配置功能。
- 产品配置模板适用于该产品下的所有设备。目前，不支持在控制台向单个设备推送配置文件。

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

说明:

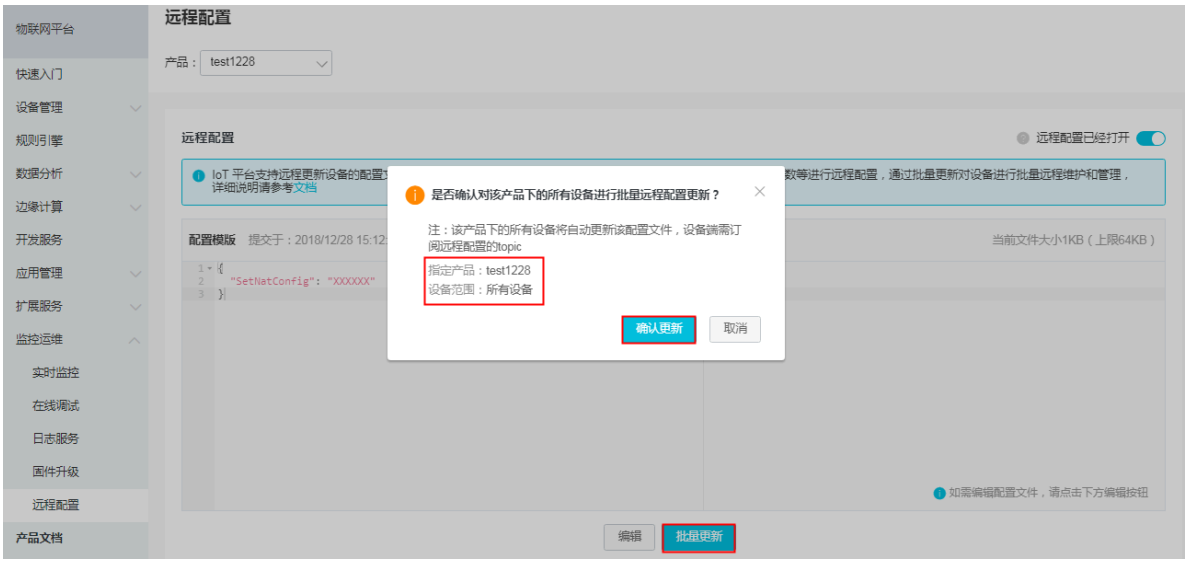
- 远程配置文件为JSON格式。物联网平台对配置内容没有特殊要求，但系统会对提交的配置文件进行JSON格式校验，避免错误格式引起配置异常。
- 配置文件最大支持64 KB。编辑框右上角将实时显示当前文件的大小。超过64KB的配置文件无法提交。

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



3. 向设备批量推送配置文件。单击批量更新，物联网平台会向该产品下的所有设备批量推送配置文件。

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



说明:

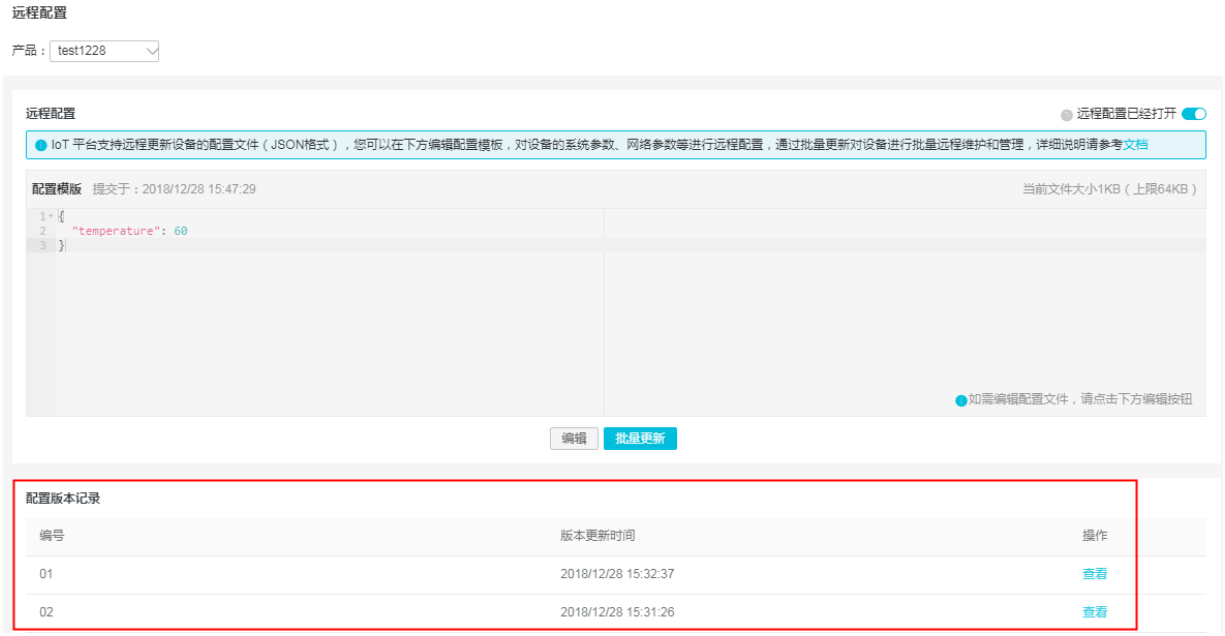
- 批量更新频率限制：一小时内仅允许操作一次 。

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

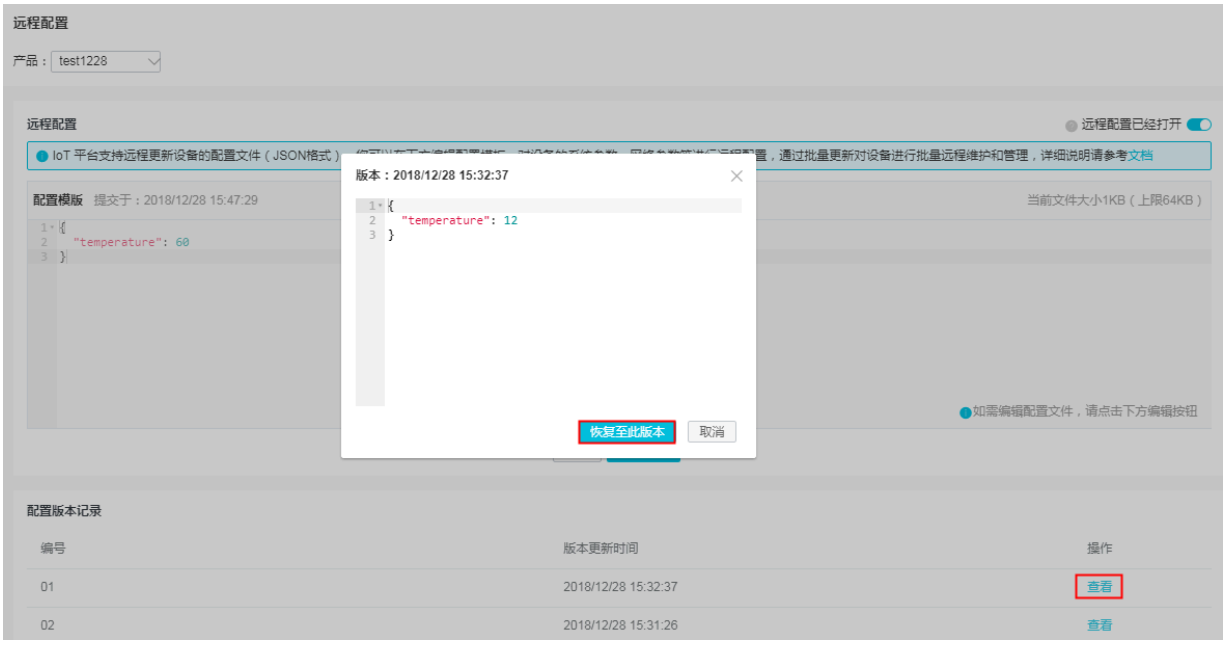
4. 设备端接收云端下发的配置信息后，自动更新配置。

查看和管理配置文件版本：

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



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



场景二：设备主动请求配置信息

在一些场景中，设备需要主动查询配置信息。可按如下步骤进行远程配置：

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

4 扩展服务

4.1 三维数据可视化

三维数据可视化服务通过空间建模展示设备实时状态，方便您查看设备状态并进行管理。

三维数据可视化主要功能包括：

- 支持拖拽建模
- 支持IoT设备联动
- 支持单空间搜索

目前仅支持高级版产品使用三维数据可视化服务。

开通服务

三维数据可视化为扩展服务，需开通后方能使用。

登录物联网平台的控制台，选择扩展服务 > 三维数据可视化，在服务详情页面单击开通服务。

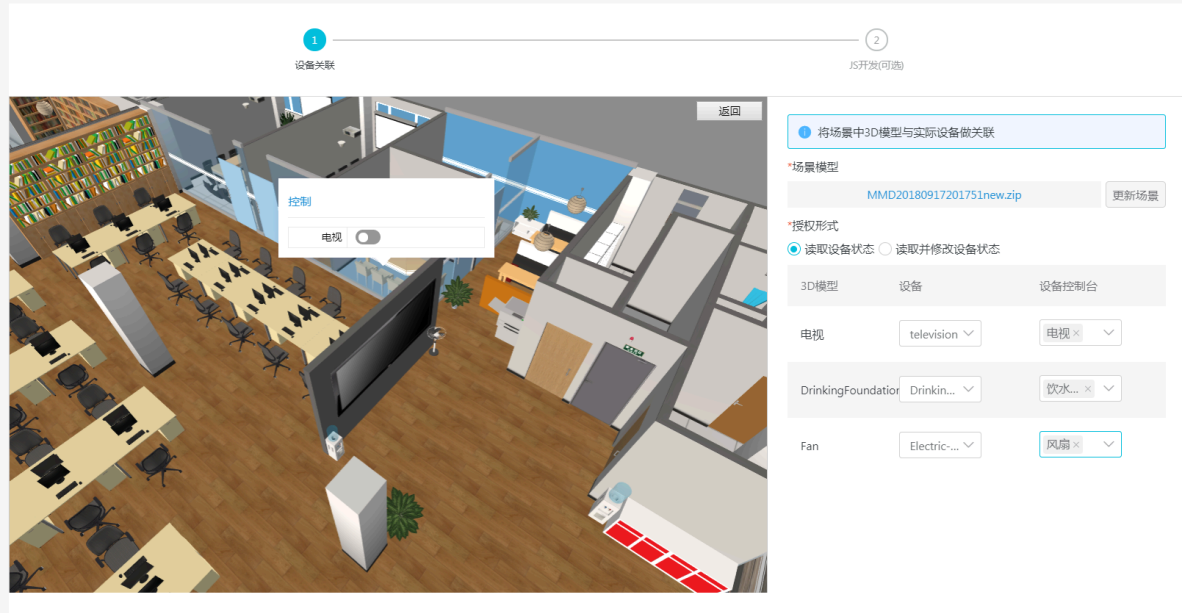
创建场景

1. 登录物联网平台控制台。
2. 单击我的服务 > 三维数据可视化。
3. 在我的场景下，单击“+”号图标。
4. 根据参数说明设置场景参数，并单击确定。

参数	说明
场景名称	设置您的场景名称。
目标产品或设备组	选择部署场景的目标产品或设备组。 目标产品或设备组中包含的设备，与您使用的场景模型关联。创建设备组的操作请参见设备分组。
场景模型	上传场景模型。 说明: 您可以下载IoT提供的模型编辑器，参考搭建工具手册，制作您自己的场景模型。

5. 配置设备关联参数。

   下图以智能家居为例，展示设备关联内容。



参数	说明
场景模型	您可以更新已上传的场景模型。
授权形式	授权形式，可分为读取设备状态与读取并修改设备状态。 · 选择读取设备状态表示您授权第三方读取您的设备状态，展示在页面上。 · 选择读取并修改设备状态表示您授权第三方读取您的设备状态，并且操作您的设备来改变设备状态，主要用于对设备的远程控制。
3D模型、设备、设备控制台	将您3D场景模型中的设备与IoT平台创建的设备关联起来，并将设备属性展示出来。 · 3D模型：模型中的设备 · 设备：IoT控制台创建的设备 · 设备控制台：IoT控制台创建设备时，设置的设备属性。 说明: · 设备属性具体信息请参考新增物模型和导入物模型。 · 设备控制台处也支持展示设备的位置信息，位置信息需要通过设备标签来配置。

6. 单击保存，可保存您设置的模型与设备的关联关系。
7. 单击下一步，对已创建好的场景模型进行JS补充，具体操作方法请参见搭建工具手册。
8. 单击保存，可保存您修改的代码。

9. 单击发布 > 确定，将您创建的场景模型发布到三维数据可视化服务中。

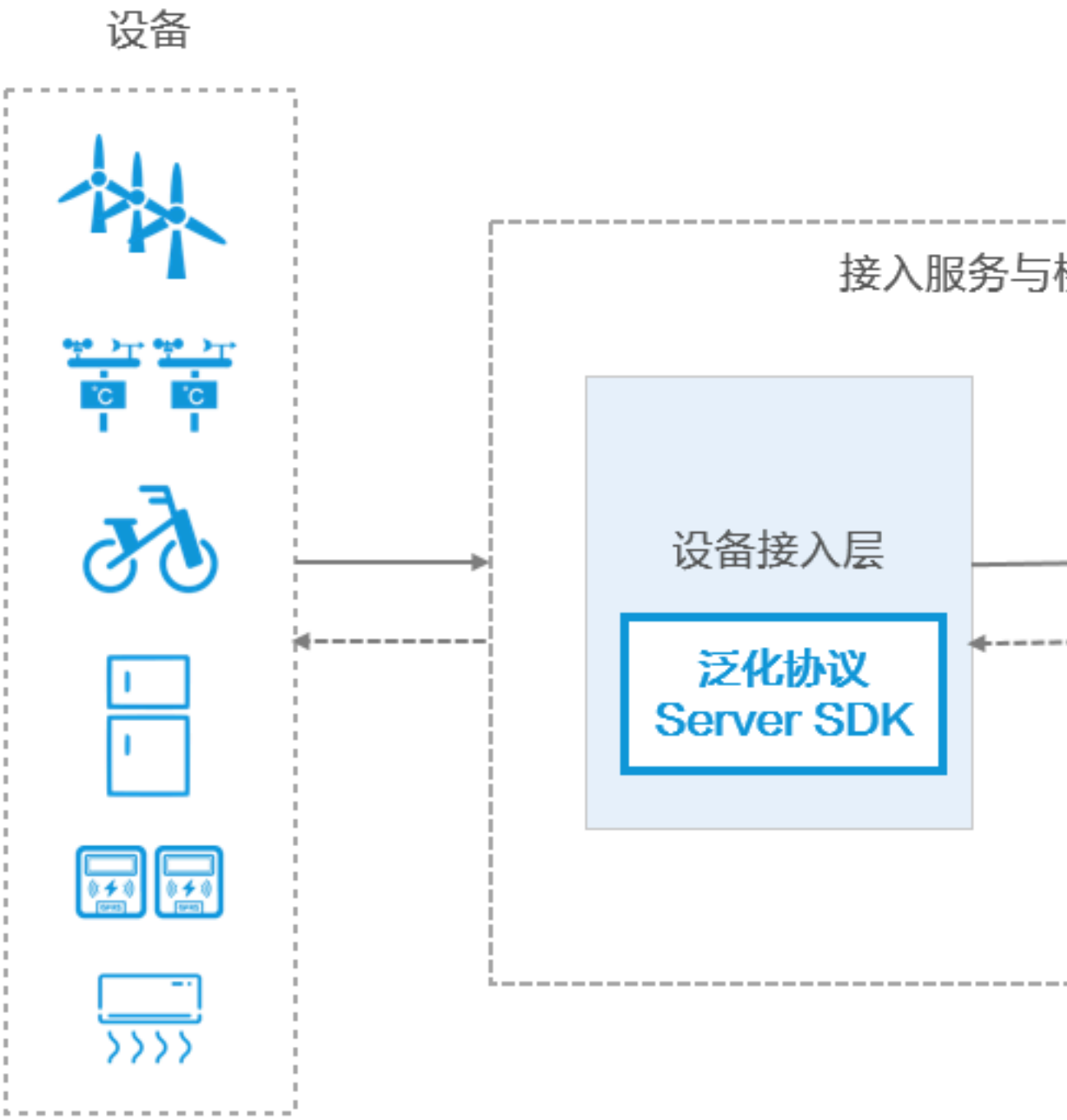
5 泛化协议

5.1 概览

阿里云物联网平台已经支持基于MQTT、CoAP和HTTP等一些协议的通信，其他类型协议，如消防协议GB/T 26875.3-2011、Modbus、JT808等暂未接入，在特定场景下，部分设备也可能无法直接接入物联网平台。此时，您需要使用泛化协议SDK，快速构建桥接服务，搭建设备或平台与阿里云物联网平台的双向数据通道。

泛化协议SDK

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



泛化协议一共提供两个SDK：核心SDK与Server SDK。

- 泛化协议核心SDK

  核心SDK抽象了如Session管理、配置管理等通用能力，在设备与物联网平台中间，扮演网桥角色，代替设备与物联网平台通信，有效降低了开发者适配物联网平台的复杂性。其主要功能如下：

  - 提供非持久化的Session管理能力。
  - 提供基于配置文件的配置管理能力。
  - 提供设备连接管理能力。
  - 提供上行通讯能力。
  - 提供下行通讯能力。
  - 支持设备认证。

  当您的设备已经使用传统方式连接物联网，现在希望将已有设备或平台桥接至阿里云物联网平台时，请使用核心SDK。

- Server SDK

  泛化协议Server SDK在核心SDK基础上提供了基于TCP/UDP的设备端接入服务。主要功能如下：

  - 支持基于TCP/UDP传输的任意协议。
  - 支持TLS/SSL加密传输。
  - 支持接入容量水平扩展。
  - 基于Netty的通信服务。
  - 提供自动化可定制的设备连接管理能力。

  若您希望从头构建接入服务，请使用附带了Socket通信功能的Server SDK。

开发和部署

控制台创建产品与设备

参考创建产品(高级版)章节，创建网桥产品与设备。获取网桥三元组。

> 说明:
> 网桥是个虚拟概念，您可以使用任意设备的三元组信息作为网桥的三元组。

SDK依赖

泛化协议SDK目前仅提供Java语言版本，支持JDK 1.8及以上版本。Maven依赖如下：

```
<!-- 核心SDK -->
<dependency>
    <groupId>com.aliyun.openservices</groupId>
```

```
    <artifactId>iot-as-bridge-sdk-core</artifactId>
    <version>1.0.0</version>
</dependency>

<!-- Server SDK -->
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-as-bridge-sdk-server</artifactId>
    <version>1.0.0</version>
</dependency>
```

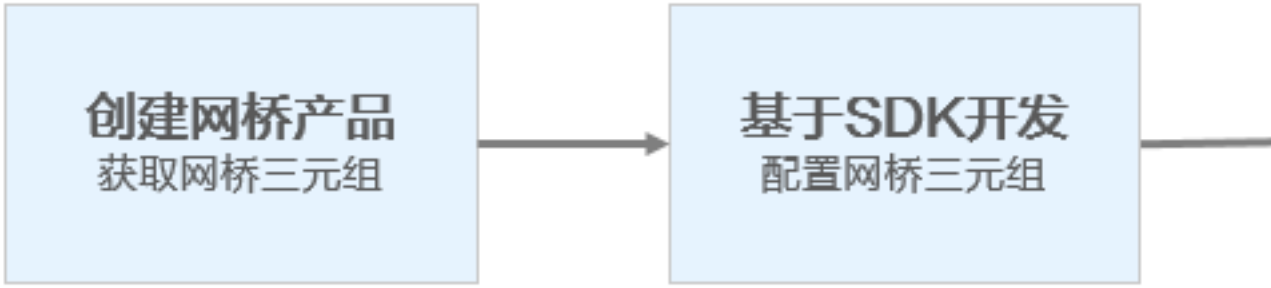
开发SDK

核心*SDK*开发和*Server SDK*开发为您简单介绍了开发流程。具体实现细节，请参考Javadoc。

部署服务

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

以基于阿里云云服务器ECS为例，上线流程如下:



5.2 核心SDK开发

基于泛化协议核心SDK，您可以将物联网平台桥接服务与已有的接入服务或平台进行高效整合，使设备或者服务得以快速接入阿里云物联网平台。

准备工作

泛化协议核心SDK概念、功能及Maven依赖，请参考概览。

配置管理

泛化协议核心SDK默认使用基于文件的配置管理。自定义配置请参见自定义组件>配置管理。

- **支持格式包括Java Proterties、JSON以及可读性较好的*HOCON*（JSON超集）。**

- 支持结构化配置项以提升可维护性。
- 可以通过Java系统属性覆盖文件配置，如java -Dmyapp.foo.bar=10。
- 支持配置文件分离和嵌套引用。

表 5-1: application.conf

网桥是个虚拟概念，您可以使用任意设备的三元组信息作为网桥的三元组（productKey，deviceName，和deviceSecret。

参数	是否必需	描述
productKey	是	网桥所属产品的 ProductKey。
deviceName	否	网桥对应的DeviceName，默认值为ECS MAC地址。
deviceSecret	否	网桥对应的DeviceSecret。
http2Endpoint	是	HTTP2网关服务地址。 HTTP2网关服务地址的结构为：`${UID}.iot-as-http2.${RegionId}.aliyuncs.com:443`。 其中， · 变量`${UID}`需替换成您的阿里云账号ID。登录阿里云控制台，将光标移至您的账号头像上，在选项框中单击安全设置，跳转至账号管理页面，查看您的账号ID。 · 变量`${RegionId}`需替换成您的服务所在地域。如，某用户的地域为上海，HTTP2网关服务地址是 `123456789.iot-as-http2.cn-shanghai.aliyuncs.com:443`。 RegionId 的表达方法，请参见通用参考地域与可用区
authEndpoint	是	设备认证服务地址。 设备认证服务地址结构为：`https://iot-auth.${RegionId}.aliyuncs.com/auth/bridge`。 其中，变量`${RegionId}`需替换成您的服务所在地域。如，地域为上海，则认证服务地址就是 `https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`。 RegionId 的表达方法，请参见通用参考地域与可用区
popClientProfile	是	Open API调用客户端相关配置，参见下表*API Client*配置。

表 5-2: API Client配置

参数	是否必需	描述
accessKey	是	Open API调用用户身份标识。
accessSecret	是	Open API调用用户身份标识对应的秘钥。
name	是	调用指定地域API的节点名称。
region	是	调用指定地域API的节点ID。
product	是	产品名称，除非特殊情况否则请配置为 `Iot` 。
endpoint	是	调用指定地域API的节点地址。 节点地址结构 为：`iot.${RegionId}.aliyuncs.com`。 其中，变量`${RegionId}`需替换成您的服务所在地域。 如上海节点的 endpoint 为：`iot.cn-shanghai.aliyuncs.com`。 RegionId 的表达方法，请参见通用参考地域与可用区

devices.conf

配置设备在阿里云物联网平台的映射，即配置设备三元组。自定义配置文件请参见自定义组件>配置管理。

```
XXXX // 表示设备的原始标识
{
    productKey: "",
    deviceName: "",
    deviceSecret: ""
}
```

开发接口

初始化

`com.aliyun.iot.as.bridge.core.BridgeBootstrap`负责初始化设备到阿里云物联网平台的通信。创建新的BridgeBootstrap实例后，网关基本配置的管理组件会被初始化。使用自定义的配置管理，请参见自定义组件>配置管理。

请通过以下接口完成初始化。

- `bootstrap()`：不实现消息下行功能并进行初始化。
- `bootstrap(DownlinkChannelHandler handler)`：采用开发者指定的`DownlinkChannelHandler`进行初始化。

示例代码：

```
BridgeBootstrap bootstrap = new BridgeBootstrap();
// 不实现下行通讯
bootstrap.bootstrap();
```

设备上线

当设备建立连接或者发送连接请求时，应发起设备上线操作，只有设备上线后，才能与阿里云物联网平台通讯。设备上线分为两个部分：本地Session的初始化以及设备认证。

1. Session初始化

   泛化协议SDK提供非持久化的本地Session管理功能，关于自定义Session管理，请参见自定义组件>*Session*管理。

   创建接口：

   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel)`
   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel, int heartBeatInterval)`
   - `com.aliyun.iot.as.bridge.core.model.Session.newInstance(String originalIdentity, Object channel, int heartBeatInterval, int heartBeatProbes)`

   其中`originalIdentity`表示原始协议中设备的唯一标识，如序列号、编号等；`channel`为设备连接至桥接服务的通信通道如Netty的Channel；`heartBeatInterval`与`heartBeatProbes`用于自动的心跳过期监测，分别代表心跳的间隔时间（以秒为单位）以及允许的最大心跳丢失次数，超过该次数后将发送心跳超时事件，开发者可通过注册`com.aliyun.iot.as.bridge.core.session.SessionListener`来处理心跳超时事件。

2. 设备认证

   设备本地Session初始化完成后请调用`com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler.doOnline(Session newSession, String originalIdentity, String... credentials)`完成本地设备认证和阿里云物联网平台上线认证，并根据结果允许设备后续通信或者断开连接。默认无本地认证，到阿里云物联网平台上线认证由SDK提供支持，如需使用自定义本地认证，请参见自定义组件>连接认证。

   示例代码：

   ```
   UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
   Session session = Session.newInstance(device, channel);
   ```

```
boolean success = uplinkHandler.doOnline(session, originalIdentity);
if (success) {
    // 上线成功，接受后续通信请求
} else {
    // 上线失败，拒绝后续通信请求，断开连接（如有）
}
```

设备下线

当设备连接断开或者探测到需要断开连接的时候，应发起设备下线操作，通过接口`com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler.doOffline(String originalIdentity)`实现设备下线。

示例代码：

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
uplinkHandler.doOffline(originalIdentity);
```

上报数据

开发者可通过`com.aliyun.iot.as.bridge.core.handler.UplinkChannelHandler`上报数据到阿里云物联网平台。操作包括以下三步：获取上行数据关联的设备、找到设备关联的Session、上报数据到阿里云物联网平台。请通过以下接口完成数据上报。

说明：
请开发者务必注意控制是否上报数据以及安全相关前置处理 。

- `CompletableFuture doPublishAsync(String originalIdentity, String topic, byte[] payload, int qos)`：异步发送数据并立即返回，开发者需根据future判断发送结果。
- `CompletableFuture doPublishAsync(String originalIdentity, ProtocolMessage protocolMsg)`：异步发送数据并立即返回，开发者需根据future判断发送结果。
- `boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg, int timeout)`：异步发送数据并等待发送结果返回。
- `boolean doPublish(String originalIdentity, String topic, byte[] payload, int qos, int timeout)`：异步发送数据并等待发送结果返回。

示例代码：

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
DeviceIdentity identity = ConfigFactory.getDeviceConfigManager().
getDeviviceIdentity(device);
if (identity == null) {
    // 设备未映射到阿里云物联网平台上的设备，丢弃消息
    return;
}
```

```
Session session = SessionManagerFactory.getInstance().getSession(
device);
if (session == null) {
    // 设备尚未上线，上报数据到阿里云物联网平台前请务必确保设备已上线，请上线设备或
者丢弃消息
}
boolean success = uplinkHandler.doPublish(session, topic, payload, 0,
10);
if(success) {
    // 上报数据到阿里云物联网平台成功
} else {
    // 上报数据到阿里云物联网平台失败
}
```

下行消息

泛化协议SDK提供了`com.aliyun.iot.as.bridge.core.handler.DownlinkChannelHandler`作为下行到设备的数据分发处理器，支持单播和广播(如果云端下发的数据不包含特定设备信息则为广播)。

示例代码：

```
public class SampleDownlinkHandler implements DownlinkChannelHandler {
    @Override
    public boolean pushToDevice(Session session, String topic, byte[]
payload) {
        // 处理设备消息推送
    }

    @Override
    public boolean broadcast(String topic, byte[] payload) {
        // 处理广播
    }
}
```

自定义组件

开发者可以自定义设备连接认证、Session管理以及配置管理组件，如果期望使用自定义的组件，请务必在调用BridgeBootstrap初始化之前完成这些组件的初始化和替换。

连接认证

自定义设备连接认证需实现接口`com.aliyun.iot.as.bridge.core.auth.AuthProvider`，并在BridgeBootstrap初始化前调用`com.aliyun.iot.as.bridge.core.auth.AuthProviderFactory.init(AuthProvider customizedProvider)`将认证组件替换为自定义实现。

Session管理

自定义Session管理需实现接口`com.aliyun.iot.as.bridge.core.session.SessionManager`，并在BridgeBootstrap初始化前调用`com.aliyun.iot.as.bridge.core.session`

`.SessionManagerFactory.init(SessionManager<?> customizedSessionManager)`将Session组件替换为自定义实现。

配置管理

自定义配置管理需实现接口`com.aliyun.iot.as.bridge.core.config.DeviceConfigManager`和`com.aliyun.iot.as.bridge.config.BridgeConfigManager`，并在BridgeBootstrap初始化前调用 `com.aliyun.iot.as.bridge.core.config.ConfigFactory.init(BridgeConfigManager bcm, DeviceConfigManager dcm)`将配置管理组件替换为自定义实现，其中参数均可为空，如果为空则使用泛化协议SDK默认实现。

5.3 Server SDK开发

5.3.1 UDP Server开发接口

基于泛化协议SDK的UDP Server开发接口，您可以快速构建以UDP为传输协议的接入服务，并将其与阿里云物联网平台桥接。

启动引导

com.aliyun.iot.as.bridge.server.BridgeServerBootstrap是启动Socket服务器和桥接服务的引导类。创建新的BridgeServerBootstrap实例后，会初始化基于配置文件的基本配置组件。

示例代码：

```
BridgeServerBootstrap bootstrap = new BridgeServerBootstrap(new
UdpDecoderFactory() {
    @Override
    public MessageToMessageDecoder newInstance() {
        // 返回解码器
    }
}, new UdpEncoderFactory() {
    @Override
    public MessageToMessageEncoder<?> newInstance() {
        // 返回编码器
    }
}, new UdpBasedProtocolAdaptorHandlerFactory() {
    @Override
    public CustomizedUdpBasedProtocolHandler newInstance() {
        // 返回协议适配器
    }
});
try {
    bootstrap.start();
} catch (BootException | ConfigException e) {
    // 处理启动失败
}
```

实例化UDP类型BridgeServerBootstrap

- com.aliyun.iot.as.bridge.server.channel.factory.UdpDecoderFactory: 协议解码器工厂类具体实现，负责如何创建一个新的解码器实例，用于上行数据的解析，线程安全，可为null。
- com.aliyun.iot.as.bridge.server.channel.factory.UdpEncoderFactory: 协议编码器工厂类具体实现，负责如何创建一个新的编码器实例，用于下行数据到协议编码的转换，线程安全，可为null。
- com.aliyun.iot.as.bridge.server.channel.factory.UdpBasedProtocolAdaptorHandlerFactory: 协议适配器工厂类具体实现，负责如何创建一个新的协议适配器实例，用于解码器解析出来的数据到云端数据的转换，线程安全，不可为null。

启动Socket Server

创建BridgeServerBootstrap完成后，需调用com.aliyun.iot.as.bridge.server.BridgeServerBootstrap.start()完成Socket Server启动。

协议解码

协议解码组件需派生自io.netty.handler.codec.MessageToMessageDecoder<I>，具体内容请参考*MessageToMessageDecoder*文档。

示例代码：

```
public class SampleDecoder extends  MessageToMessageDecoder<DatagramPacket> {
    @Override
    protected void decode(ChannelHandlerContext ctx, DatagramPacket in, List<Object> out) throws Exception {
        // 解码协议
    }
}
```

协议编码

协议编码组件需派生自io.netty.handler.codec.MessageToMessageEncoder<I>，具体内容请参考*MessageToMessageEncoder*文档。

示例代码：

```
public class SampleEncoder extends MessageToMessageEncoder<T>{
    @Override
    protected void encode(ChannelHandlerContext ctx, T msg, ByteBuf out) throws Exception {
        // 协议编码
    }
```

```
}
```

协议适配器

为了降低开发成本，提高泛化协议设备接入与桥接服务的开发效率，Server SDK为协议适配器提供了可扩展、可定制的基础能力类com.aliyun.iot.as.bridge.server.channel.CustomizedUdpBasedProtocolHandler。其封装了与阿里云物联网平台的对接细节，使您可以专注于协议本身的业务，无需关注具体的对接细节。泛化协议的协议处理适配器需派生自该类。

设备上线

当设备建立连接或发送连接请求时，应触发设备上线操作。设备上线分为两个部分：网桥本地Session的初始化以及设备认证。

1. Session初始化

   请参见核心SDK>设备上线>Session初始化。

2. 设备认证

   本地Session初始化完成后即可调用doOnline(Session newSession, String originalIdentity, String... credentials)或doOnline(String originalIdentity, String... credentials)完成本地设备认证和阿里云物联网平台上线认证。认证成功，设备可以进行通信；认证失败，设备将断开连接。

   示例代码：

```
Session session = Session.newInstance(device, channel);
boolean success = doOnline(session, originalIdentity);
if (success) {
    // 上线成功，接受后续通信请求
} else {
    // 上线失败，拒绝后续通信请求，断开连接（如有）
}
```

设备下线

当设备连接断开或者网桥探测到需要断开连接时，应触发设备下线操作。泛化协议Server SDK自身提供了设备连接断开时自动下线设备的能力，开发者只需负责处理主动触发设备下线操作的情形即可，下线某个设备可通过接口 doOffline(Session session)实现。

上报数据

协议适配器需override channelRead(ChannelHandlerContext ctx, Object msg)方法，该方法是所有设备上行数据的入口，msg为decoder所返回的数据。

上报数据到物联网平台包括三步：获取上行数据关联的设备、找到设备关联的Session、上报数据到物联网平台。需通过以下接口完成数据上报：

- CompletableFuture doPublishAsync(String originalIdentity, String topic, byte[] payload, int qos)：异步发送数据并立即返回，开发者需根据future判断发送结果。
- CompletableFuture doPublishAsync(String originalIdentity, ProtocolMessage protocolMsg)：异步发送数据并立即返回，开发者需根据future判断发送结果。
- boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg, int timeout)：异步发送数据并等待发送结果返回。
- boolean doPublish(String originalIdentity, String topic, byte[] payload, int qos, int timeout)：异步发送数据并等待发送结果返回。

示例代码：

```
DeviceIdentity identity = ConfigFactory.getDeviceConfigManager().
getDeviviceIdentity(device);
if (identity == null) {
    // 设备未成功映射到阿里云物联网平台上的设备，丢弃消息
    return;
}
Session session = SessionManagerFactory.getInstance().getSession(
device);
if (session == null) {
    // 设备尚未上线，请上线设备或者丢弃消息。上报数据到物联网平台前请务必确保设备已
上线。
}
boolean success = doPublish(session, topic, payload, 0, 10);
if(success) {
    // 成功上报数据到阿里云物联网平台
} else {
    // 上报数据到阿里云物联网平台失败
}
```

下行消息

暂不支持

5.3.2 TCP Server开发接口

基于泛化协议SDK的TCP Server开发接口，您可以快速构建以TCP为传输协议的接入服务，并将其与阿里云物联网平台桥接。

启动引导

com.aliyun.iot.as.bridge.server.BridgeServerBootstrap是启动Socket服务器和桥接服务的引导类。创建新的BridgeServerBootstrap实例后，会初始化基于配置文件的配置管理组件。

示例代码：

```
BridgeServerBootstrap bootstrap = new BridgeServerBootstrap(new
TcpDecoderFactory() {
    @Override
    public ByteToMessageDecoder newInstance() {
        // 返回解码器
    }
```

```
}, new TcpEncoderFactory() {
    @Override
    public MessageToByteEncoder<?> newInstance() {
        // 返回编码器
    }
}, new TcpBasedProtocolAdaptorHandlerFactory() {
    @Override
    public CustomizedTcpBasedProtocolHandler newInstance() {
        // 返回协议适配器
    }
}, new DefaultDownlinkChannelHandler());
try {
    bootstrap.start();
} catch (BootException | ConfigException e) {
    // 处理启动失败
}
```

实例化TCP类型BridgeServerBootstrap

- com.aliyun.iot.as.bridge.server.channel.factory.TcpDecoderFactory: 协议解码器工厂类具体实现，负责如何创建一个新的解码器实例，用于上行数据的解析，线程安全，可为null。
- com.aliyun.iot.as.bridge.server.channel.factory.TcpEncoderFactory: 协议编码器工厂类具体实现，负责如何创建一个新的编码器实例，用于下行数据到协议编码的转换，线程安全，可为null。
- com.aliyun.iot.as.bridge.server.channel.factory.TcpBasedProtocolAdaptorHandlerFactory: 协议适配器工厂类具体实现，负责如何创建一个新的协议适配器实例，用于解码器解析出来的数据到云端数据的转换，线程安全，不可为null。
- com.aliyun.iot.as.bridge.core.handler.DownlinkChannelHandler: 云端下行数据分发处理器，支持单播和广播。单播默认直接转发云端下发的数据到设备，广播需开发者自定义具体实现。可为null，为null时表示不支持下行。

启动Socket Server

创建BridgeServerBootstrap完成后，需调用com.aliyun.iot.as.bridge.server.BridgeServerBootstrap.start()完成Socket Server启动。

协议解码

协议解码组件需派生自io.netty.handler.codec.ByteToMessageDecoder，具体内容请参考 *ByteToMessageDecoder文档*。

示例代码：

```
public class SampleDecoder extends ByteToMessageDecoder {
    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<
Object> out) throws Exception {
        // 解码协议
    }
```

```
}
```

协议编码

协议编码组件需派生自io.netty.handler.codec.MessageToByteEncoder<I>，具体内容请参考*MessageToByteEncoder*文档。

示例代码：

```
public class SampleEncoder extends MessageToByteEncoder<String>{
    @Override
    protected void encode(ChannelHandlerContext ctx, String msg,
ByteBuf out) throws Exception {
        // 协议编码
    }
}
```

协议适配器

为了降低开发成本，提高泛化协议设备接入与桥接服务的开发效率，泛化协议Server SDK为协议适配器提供了可扩展、可定制的基础能力类com.aliyun.iot.as.bridge.server.channel.CustomizedTcpBasedProtocolHandler。其封装了与阿里云物联网平台的对接细节，使您可以专注于协议本身的业务，无需关注具体的对接细节。泛化协议的协议处理适配器需派生自该类。

设备上线

当设备建立连接或者发送连接请求时，应触发设备上线操作。设备上线分为两个部分：网桥本地Session的初始化以及设备认证。

1. Session初始化

   具体内容请参见核心*SDK*>设备上线>*Session*初始化。

2. 设备认证

   本地Session初始化完成后即可调用doOnline(ChannelHandlerContext ctx, Session newSession, String originalIdentity, String... credentials)完成本地设备认证和阿里云物联网平台上线认证。认证成功，设备可以进行通信；认证失败，设备将断开连接。

   示例代码：

   ```
   Session session = Session.newInstance(device, channel);
   boolean success = doOnline(session, originalIdentity);
   if (success) {
       // 上线成功，接受后续通信请求
   } else {
       // 上线失败，拒绝后续通信请求，断开连接（如有）
   }
   ```

设备下线

当设备连接断开或者网桥探测到需要断开连接时，应触发设备下线操作。Server SDK自身提供了设备连接断开时自动下线设备的能力，开发者只需负责处理主动触发设备下线操作的情形即可，下线某个设备可通过接口doOffline(Session session)实现。

上报数据

协议适配器需override channelRead(ChannelHandlerContext ctx, Object msg)方法，该方法是所有设备上行数据的入口，msg为decoder所返回的数据。

上报数据到物联网平台包括三步：获取上行数据关联的设备、找到设备关联的Session、上报数据到物联网平台。需通过以下接口完成数据上报：

- CompletableFuture doPublishAsync(Session session, String topic, byte[] payload, int qos)：异步发送数据并立即返回，开发者需根据future判断发送结果。
- CompletableFuture doPublishAsync(Session session, ProtocolMessage protocolMsg)：异步发送数据并立即返回，开发者需根据future判断发送结果。
- boolean doPublish(Session session, ProtocolMessage protocolMsg, int timeout)：异步发送数据并等待发送结果返回。
- boolean doPublish(Session session, String topic, byte[] payload, int qos, int timeout)：异步发送数据并等待发送结果返回。

示例代码：

```
DeviceIdentity identity = ConfigFactory.getDeviceConfigManager().
getDeviviceIdentity(device);
if (identity == null) {
    // 设备未成功映射到阿里云物联网平台上的设备，丢弃消息
    return;
}
Session session = SessionManagerFactory.getInstance().getSession(
device);
if (session == null) {
    // 设备尚未上线，请上线设备或者丢弃消息。上报数据到物联网平台前请务必确保设备已
上线。
}
boolean success = doPublish(session, topic, payload, 0, 10);
if(success) {
    // 成功上报数据到阿里云物联网平台
} else {
    // 上报数据到阿里云物联网平台失败
}
```

下行消息

具体请参考核心*SDK*>下行消息文档。

SDK提供了com.aliyun.iot.as.bridge.core.handler.DefaultDownlinkChannelHandler作为下行数据的分发处理器。支持单播和广播，其中单播默认直接转发云端下发的数据到设备，广播需开发者自定义具体实现。开发者可以通过派生子类实现行为定制。

示例代码：

```
import io.netty.channel.Channel;
import io.netty.channel.ChannelFuture;
...

public class SampleDownlinkChannelHandler implements DownlinkCh
annelHandler {
    @Override
    public boolean pushToDevice(Session session, String topic, byte[]
payload) {
        // 从设备对应的session中取出通信通道
        Channel channel = (Channel) session.getChannel().get();
        if (channel != null && channel.isWritable()) {
            String body = new String(payload, StandardCharsets.UTF_8);
            // 发送下行数据到设备
            ChannelFuture future = channel.pipeline().writeAndFlush(
body);
            future.addListener(ChannelFutureListener.FIRE_EXCEP
TION_ON_FAILURE);
            return true;
        }
        return false;
    }

    @Override
    public boolean broadcast(String topic, byte[] payload) {
        throw new RuntimeException("not implemented");
    }
}
```

5.3.3 Server SDK开发

基于泛化协议Server SDK，您可以快速搭建桥接物联网平台的接入服务，使设备或者服务快速接入阿里云物联网平台。

准备工作

泛化协议Server SDK概念、功能及Maven依赖，请参考概览。

配置管理

泛化协议Server SDK默认提供基于文件的配置管理，可通过在*application.conf*中增加 socketServer 配置项设置如下表所示的Socket Server相关参数。自定义配置管理请参见自定义组件>配置管理。

名称	描述	是否必须
address	连接服务监听地址，支持网卡名称如eth1，IPv4地址前缀如10.30，建议明确指定	否
backlog	TCP连接backlog的数量	否

名称	描述	是否必须
ports	连接服务监听端口，可指定多个，默认为：9123	否
listenType	socket server类型。取值为udp或tcp，默认为tcp。不区分大小写	否
broadcastEnabled	是否支持udp广播。当listenType为udp时有效，默认为true	否
unsecured	是否支持不加密的TCP连接。当listenType为tcp时有效	否
keyPassword	证书库密码。当listenType为tcp时有效	否 说明: 只有同时配置keyPassword、keyStoreFile和keyStoreTyp会生效，否则配置将被忽略。
keyStoreFile	证书库文件地址。当listenType为tcp时有效	否
keyStoreType	证书库类型。当listenType为tcp时有效	否

开发接口

以下两篇文档假设开发者对基于Netty的开发已有基本了解。关于Netty的知识，请参考*Netty*文档。

- *TCP*类型*Server*开发接口
- *UDP*类型*Server*开发接口

自定义组件

除基于文件的配置管理外，开发者还可以自定义server的配置管理。

自定义配置管理需实现接口com.aliyun.iot.as.bridge.server.config.BridgeServerConfigManager，调用 com.aliyun.iot.as.bridge.server.config.ServerConfigFactory.init(BridgeServerConfigManager bcm)将默认配置管理组件替换为自定义组件，完成自定义组件的初始化。然后，再启动网桥。

6 RRPC

6.1 什么是RRPC

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

名词解释

- RRPC：远程同步调用。
- RRPC 请求消息：云端下发给设备端的消息。
- RRPC 响应消息：设备端回复给云端的消息。
- RRPC 消息id：云端为每次RRPC调用生成的唯一消息id。
- RRPC 订阅Topic：设备端订阅RRPC消息时传递的Topic，含有通配符。

RRPC原理

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

Topic格式

不同Topic格式使用方法不同。

- 系统Topic使用方法参见*系统Topic*。
- 自定义Topic使用方法参见*自定义Topic*。

6.2 系统Topic

RRPC支持调用系统Topic与云端通信，且相关Topic包含ProductKey和DeviceName。

系统Topic

RRPC调用的系统Topic格式如下：

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

其中，${YourProductKey}和${YourDeviceName}是您设备的三元组信息，${messageId}是云端生成的唯一的RRPC消息id。

RRPC接入

1. 云端接入工作。

   调用RRPC的API，将您的设备接入云端SDK，详细调用方法请见*RRPC*。

   以使用Java SDK为例，调用方式：

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

2. 返回RRPC响应Topic。

   设备端收到RRPC请求Topic之后，需要根据RRPC请求Topic的格式，返回对应的RRPC响应Topic消息。

   从收到的Topic中（/sys/${YourProductKey}/${YourDeviceName}/rrpc/request/${messageId}）提取出messageId，然后拼装出对应的RRPC响应Topic的格式，发送给云端。

6.3 自定义Topic

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

自定义Topic

RRPC调用的自定义Topic格式如下：

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

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

RRPC接入

1. 云端接入。

   调用RRPC的API，将您的设备接入云端SDK，详细调用方法请见*RRPC*。

   以使用Java SDK为例，调用方式：

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

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

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

2. 设备端接入。

   从云端下发自定义格式Topic的RRPC调用命令到设备端时，设备端必须在进行MQTT CONNECT协议设置时，在clientId中增加ext=1参数。MQTT接入请参考*MQTT-TCP*连接通信。

   例如，原来传递的clientId为：

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

   则添加ext=1参数后，传递的clientId为：

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

   说明：

   云端和设备端之间RRPC调用使用自定义topic的条件：

- 云端传递的Topic字段不为空
- 设备端在connect时传递了ext=1参数

3. 回复RRPC响应Topic。

   RRPC请求Topic和响应Topic格式是一样的，不需要提取messageId，直接将请求Topic作为响应Topic即可。

7 设备影子

7.1 设备影子介绍

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

- 每个设备有且只有一个设备影子，设备可以通过MQTT获取和设置设备影子以此来同步状态，该同步可以是影子同步给设备，也可以是设备同步给影子。
- 应用程序可以通过物联网平台的云端API，获取和设置设备影子，获取设备最新状态，并将期望状态下发给设备。

场景一

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

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

场景二

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

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

场景三

- 设备网络不稳定，导致设备频繁上下线，应用程序发送控制指令给设备时，设备掉线，指令无法下达到设备。
  - 通过QoS=1或者2实现，但是该方法对于服务端的压力比较大，一般不建议使用。
  - 使用设备影子机制，应用程序发送控制指令，指令携带时间戳保存在设备影子中。当设备掉线重连时，获取指令并根据时间戳确定是否执行。
- 设备真实掉线，指令发送失败。设备再上线时，设备影子功能通过指令加时间戳的模式，保证设备不会执行过期指令。

7.2 设备影子JSON详解

本文档介绍设备影子的JSON格式表达方法。

设备影子JSON文档示例：

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

JSON属性描述，如下表表 7-1: JSON属性说明所示。

表 7-1: JSON属性说明

属性	描述
desired	设备的预期状态。仅当设备影子文档具有预期状态时，才包含desired部分。 应用程序向desired部分写入数据，更新事物的状态，而无需直接连接到该设备。
reported	设备的报告状态。设备可以在reported部分写入数据，报告其最新状态。 应用程序可以通过读取该参数值，获取设备的状态。 JSON文档中也可以不包含reported部分，没有reported部分的文档同样为有效影子JSON文档。

属性	描述
metadata	当用户更新设备状态文档后，设备影子服务会自动更新metadata的值。 设备状态的元数据的信息包含以 Epoch 时间表示的每个属性的时间戳，用来获取准确的更新时间。
timestamp	影子文档的最新更新时间。
version	用户主动更新版本号时，设备影子会检查请求中的version值是否大于当前版本号。 如果大于当前版本号，则更新设备影子，并将version值更新到请求的版本中，反之则会拒绝更新设备影子。 该参数更新后，版本号会递增，用于确保正在更新的文档为最新版本。 version参数为long型。为防止参数溢出，您可以手动传入`-1`将版本号重置为`0`。

说明:

设备影子支持数组。更新数组时必须全量更新，不能只更新数组的某一部分。

更新数组数据示例:

- 初始状态:

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

- 更新:

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

- 最终状态:

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

```
}
```

7.3 设备影子数据流

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

设备影子数据流转Topic

物联网平台为每个设备预定义了两个Topic实现数据流转。定义的Topic都以以下固定格式呈现：

- Topic：*/shadow/update/${YourProductKey}/${YourDeviceName}*

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

- Topic：*/shadow/get/${YourProductKey}/${YourDeviceName}*

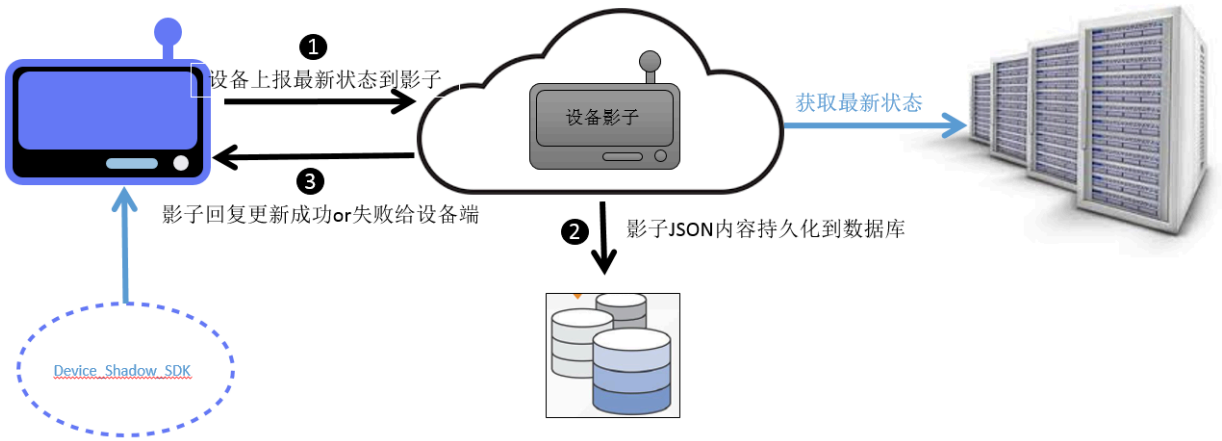
  设备影子更新状态到该Topic，设备订阅此Topic获取最新消息。

以下章节中，以某个具体灯泡设备为例，说明设备、设备影子以及应用程序之间的通信。在本文示例中，产品的ProductKey是a1PbRCFQWfX；设备名称DeviceName是lightbulb。设备以QoS=1发布消息和订阅定义的两个Topic。

设备主动上报状态

处理流程图如图 *7-1:* 设备主动上报所示。

图 7-1: 设备主动上报



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

   发送的JSON消息格式：

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

   参数说明如表 7-2: 上报参数说明所示。

   表 7-2: 上报参数说明

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

2. 当设备影子接收到灯泡上报状态时，成功更新影子文档。

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

```
}
```

3. 更新设备影子之后，设备影子会返回结果给设备（灯泡），即发送消息到设备订阅的Topic `/shadow/get/a1PbRCFQWfX/lightbulb`中。

   · 若更新成功，发送到该Topic中的消息为：

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

   · 若更新失败，发送到该Topic中的消息为：

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

错误码说明如表 *7-3:* 错误码说明所示。

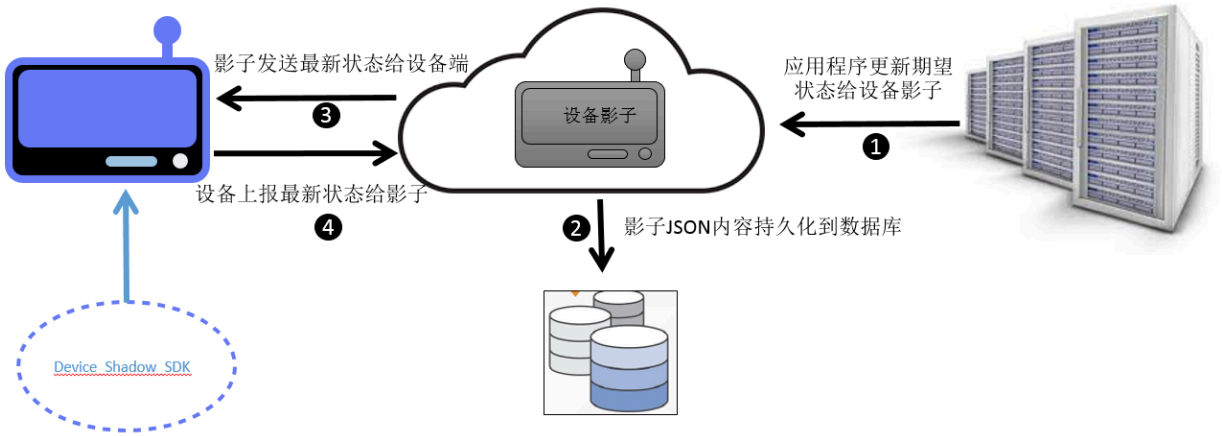
表 7-3: 错误码说明

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

应用程序改变设备状态

处理流程图如图 *7-2: 应用程序改变设备状态*所示。

图 7-2: 应用程序改变设备状态



1. 应用程序下发指令给设备影子，更改灯泡状态。

   应用程序发消息到Topic `/shadow/update/a1PbRCFQWfX/lightbulb/`中，消息码流如下：

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

2. 设备影子接收到应用程序发出更新请求后，更新其影子文档为：

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

```
    }
  },
  "timestamp": 1469564576,
  "version": 2
}
```

3. 设备影子更新完成后，发送返回结果到Topic`/shadow/get/a1PbRCFQWfX/lightbulb`中。返回结果信息构成由设备影子决定。

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

4. 如果设备灯泡在线，并且订阅了Topic`/shadow/get/a1PbRCFQWfX/lightbulb`，就会收到消息，并根据请求文档中`desired`的值更新状态，将灯泡颜色变成绿色。灯泡更新完状态后，上报最新状态到物联网平台。

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

如果有时间戳判断指令过期，也可以选择不更新。

5. 最新状态上报成功后，设备端发消息到Topic`/shadow/update/a1PbRCFQWfX/lightbulb`中清空desired属性。消息如下：

```
{
```

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

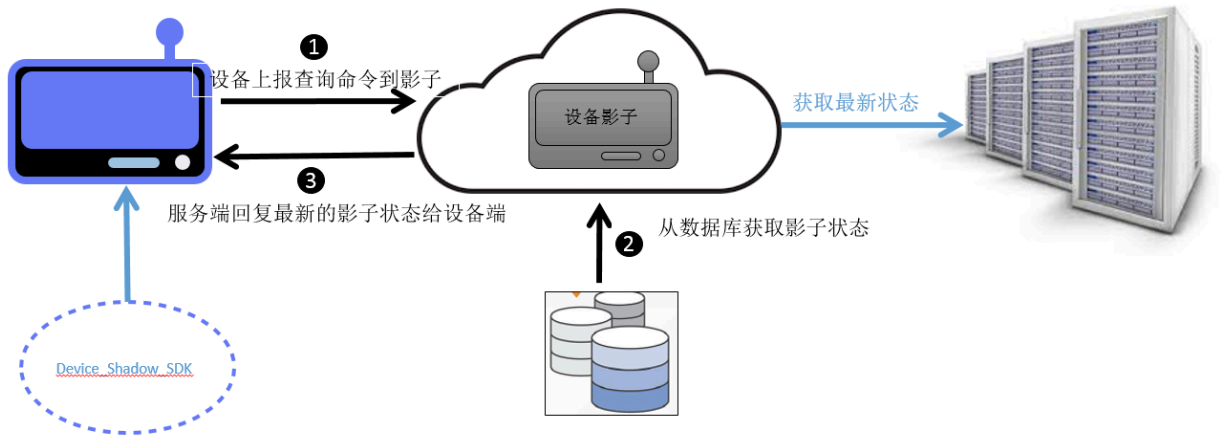
6. 上报状态成功后，设备影子会同步更新，此时的影子文档如下：

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

设备主动获取设备影子内容

处理流程图如图 7-3: 获取设备影子内容所示。

图 7-3: 获取设备影子内容



1. 灯泡主动发送以下消息到Topic/shadow/update/a1PbRCFQWfX/lightbulb中，获取设备影子中保存的灯泡最新状态。

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

```
}
```

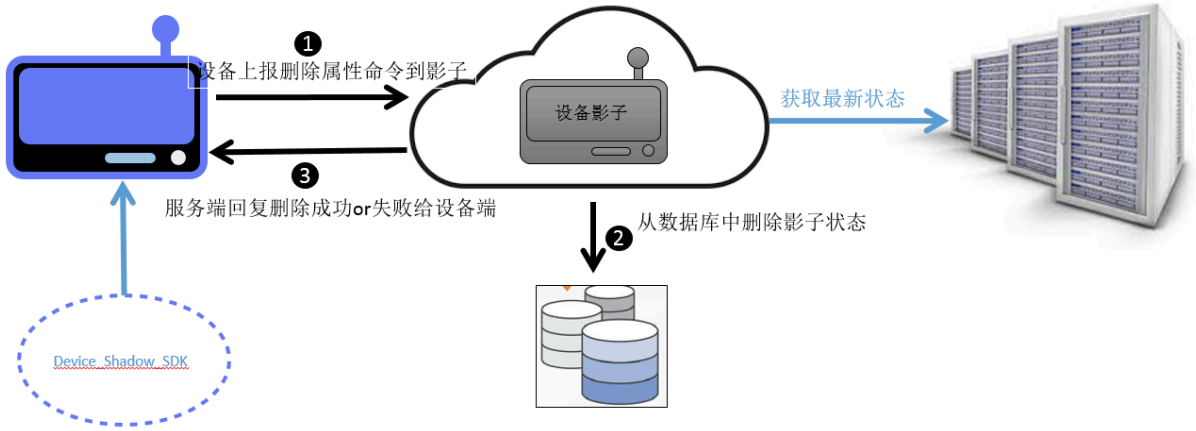
2. 当设备影子收到这条消息时，发送最新状态消息到Topic`/shadow/get/a1PbRCFQWfX/lightbulb`中。灯泡通过订阅该Topic获得消息。消息内容如下：

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

设备端删除影子属性

处理流程图如图 *7-4:* 删除影子属性所示。

图 7-4: 删除影子属性



设备灯泡要删除设备影子中保存的某条属性状态，发送删除影子属性的JSON内容到Topic/shadow/update/a1PbRCFQWfX/lightbulb中。具体的请求消息格式见以下示例。

删除属性时，只需要把method设置为delete，并且属性的值设置为null即可。

- 删除影子某一属性的JSON格式

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

- 删除影子全部属性的JSON格式

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

8 NTP服务

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

原理介绍

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

说明:

只有设备端与云端成功建立连接之后，才能通过NTP服务进行校准。

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

接入流程

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

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

说明:

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

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

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

说明:

时间戳数字的格式，支持Long和String。

3. 设备端收到服务端回复的消息，payload中包含以下信息：

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

```
}
```

4. 设备端计算出当前精确的unix时间。

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

使用示例

比如设备上时间是100，服务端时间是1000，链路延时是10，服务端从接收到发送经过了5。

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

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

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

9 账号与登录

本章将详细介绍IoT控制台账号、登录的操作。

9.1 使用阿里云主账号登录控制台

阿里云主账号具有该账号下所有资源的完全操作权限，并且可以修改账号信息。

使用主账号登录 IoT 控制台

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

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

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

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

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

进入物联网控制台后，您便可以进行产品管理、设备管理、规则管理等操作。

使用主账号创建访问控制

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

9.2 RAM授权管理

本章节将详细介绍物联网平台账号权限控制相关事宜。

9.2.1 RAM 和 STS 介绍

RAM 和 STS 是阿里云提供的权限管理系统。了解 RAM 和 STS 的详情，请参见访问控制产品帮助文档。

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

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

背景介绍

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

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

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

使用场景示例，请参见使用示例。

基本概念

使用 RAM 和 STS 涉及以下基本概念：

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

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

使用示例

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

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

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

为什么 RAM 和 STS 的使用这么复杂?

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

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

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

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

9.2.2 自定义权限

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

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

授权策略是 JSON 格式的字符串，需包含以下参数：

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

Action 定义

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

下面介绍一些典型的 Action 定义示例。

- 定义单个 API。

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

- 定义多个API。

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

- 定义所有只读 API。

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

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

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

- **定义所有读写 API**。

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

```
        "dhs:GetTopic"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ots:ListInstance",
        "ots:ListTable",
        "ots:DescribeTable",
        "ots:GetInstance"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "ons:OnsRegionList",
        "ons:OnsInstanceInServiceList",
        "ons:OnsTopicList",
        "ons:OnsTopicGet"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "hitsdb:DescribeRegions",
        "hitsdb:DescribeHiTSDBInstanceList",
        "hitsdb:DescribeHiTSDBInstance",
        "hitsdb:ModifyHiTSDBInstanceSecurityIpList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "fc:ListServices",
        "fc:GetService",
        "fc:GetFunction",
        "fc:ListFunctions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": [
        "log:ListShards",
        "log:ListLogStores",
        "log:ListProject"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Action": "ram:PassRole",
      "Resource": "*",
      "Effect": "Allow",
      "Condition": {
        "StringEquals": {
          "acs:Service": "iot.aliyuncs.com"
        }
      }
    },
```

```
    {
      "Action": [
        "cms:QueryMetricList"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

Condition 定义

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

访问 IP 限制

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

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

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

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

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

```
      }
    ],
    "Version": "1"
  }
```

HTTPS 访问限制

访问控制可以限制是否通过 HTTPS 访问。

示例：限制必须通过 HTTPS 请求访问。

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

MFA 访问限制

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

示例：限制必须通过 MFA 请求访问。

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

访问时间限制

访问控制可以限制请求的访问时间，即只允许或拒绝在某个时间点范围之前的请求。

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

```
{
  "Statement": [
```

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

典型使用场景

结合以上对 Action、Resource 和 Condition 的定义，下面介绍一些典型使用场景的授权策略定义和授权方法。

允许访问的授权策略示例

场景：定义访问 IP 地址为 10.101.168.111/24 网段的用户访问 IoT 的权限，且要求只能在 2019-01-01 00:00:00 之前访问和通过 HTTPS 访问。

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "iot:*",
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.168.111/24"
          ]
        },
        "DateLessThan": {
          "acs:CurrentTime": "2019-01-01T00:00:00+08:00"
        },
        "Bool": {
          "acs:SecureTransport": "true"
        }
      }
    }
  ],
  "Version": "1"
}
```

拒绝访问的授权策略示例

场景：拒绝访问 IP 地址为 10.101.169.111 的用户对 IoT 执行读操作。

```
{
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "iot:Query*",
        "iot:List*",
```

```
        "iot:Get*",
        "iot:BatchGet*"
      ],
      "Resource": "*",
      "Condition": {
        "IpAddress": {
          "acs:SourceIp": [
            "10.101.169.111"
          ]
        }
      }
    }
  ],
  "Version": "1"
}
```

授权策略创建成功后，在访问控制 RAM 控制台，用户管理页面，将此权限授予子账号用户。获得授权的子账号用户就可以进行权限中定义的操作。创建子账号和授权操作帮助，请参见子账号访问。

9.2.3 IoT API 授权映射表

定义授权策略，为RAM用户授予具体某些API的访问权限。

为RAM用户授权的具体方法，请参见自定义权限。

下表中列举的物联网平台 API 名称，即您在创建物联网平台相关授权策略时，参数 Action 的可选值。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
CreateProduct	iot:CreateProduct	*	创建产品。
UpdateProduct	iot:UpdateProduct	*	修改产品。
QueryProduct	iot:QueryProduct	*	查询产品信息。
QueryProductList	iot:QueryProductList	*	查询产品列表。
DeleteProduct	iot:DeleteProduct	*	删除产品。
CreateProductTags	iot:CreateProductTags	*	创建产品标签。
UpdateProductTags	iot:UpdateProductTags	*	更新产品标签。
DeleteProductTags	iot:DeleteProductTags	*	删除产品标签。
ListProductTags	iot:ListProductTags	*	查询产品标签。
ListProductByTags	iot:ListProductByTags	*	根据标签查询产品。
RegisterDevice	iot:RegisterDevice	*	注册设备。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
QueryDevice	iot:QueryDevice	*	查询指定产品下的所有设备列表。
DeleteDevice	iot:DeleteDevice	*	删除设备。
QueryPageByApplyId	iot:QueryPageByApplyId	*	查询批量注册的设备信息。
BatchGetDeviceState	iot:BatchGetDeviceState	*	批量获取设备状态。
BatchRegisterDeviceWithApplyId	iot:BatchRegisterDeviceWithApplyId	*	根据ApplyId批量申请设备。
BatchRegisterDevice	iot:BatchRegisterDevice	*	批量注册设备（随机生成设备名）。
QueryBatchRegisterDeviceStatus	iot:QueryBatchRegisterDeviceStatus	*	查询批量注册设备的处理状态和结果。
BatchCheckDeviceNames	iot:BatchCheckDeviceNames	*	批量自定义设备名称。
QueryDeviceStatistics	iot:QueryDeviceStatistics	*	获取设备的统计数量。
QueryDeviceEventData	iot:QueryDeviceEventData	*	获取设备的事件历史数据。
QueryDeviceServiceData	iot:QueryDeviceServiceData	*	获取设备的服务记录历史数据。
SetDeviceProperty	iot:SetDeviceProperty	*	设置设备的属性。
SetDevicesProperty	iot:SetDevicesProperty	*	批量设置设备属性。
InvokeThingService	iot:InvokeThingService	*	调用设备的服务。
InvokeThingsService	iot:InvokeThingsService	*	批量调用设备服务。
QueryDevicePropertyStatus	iot:QueryDevicePropertyStatus	*	查询设备的属性快照。
QueryDeviceDetail	iot:QueryDeviceDetail	*	查询设备详情。
DisableThing	iot:DisableThing	*	禁用设备。
EnableThing	iot:EnableThing	*	解除设备的禁用状态。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
GetThingTopo	iot:GetThingTopo	*	查询设备拓扑关系。
RemoveThingTopo	iot:RemoveThingTopo	*	移除设备拓扑关系。
NotifyAddThingTopo	iot:NotifyAddThingTopo	*	通知云端增加设备拓扑关系。
QueryDevicePropertyData	iot:QueryDevicePropertyData	*	获取设备的属性历史数据。
QueryDevicePropertiesData	iot:QueryDevicePropertiesData	*	批量查询指定设备的属性上报数据。
GetGatewayBySubDevice	iot:GetGatewayBySubDevice	*	根据挂载的子设备信息查询对应的网关设备信息。
SaveDeviceProp	iot:SaveDeviceProp	*	为指定设备设置标签。
QueryDeviceProp	iot:QueryDeviceProp	*	查询指定设备的标签列表。
DeleteDeviceProp	iot:DeleteDeviceProp	*	删除设备标签。
QueryDeviceByTags	iot:QueryDeviceByTags	*	根据标签查询设备。
CreateDeviceGroup	iot:CreateDeviceGroup	*	创建分组。
UpdateDeviceGroup	iot:UpdateDeviceGroup	*	更新分组信息。
DeleteDeviceGroup	iot:DeleteDeviceGroup	*	删除分组。
BatchAddDeviceGroupRelations	iot:BatchAddDeviceGroupRelations	*	添加设备到分组。
BatchDeleteDeviceGroupRelations	iot:BatchDeleteDeviceGroupRelations	*	将设备从分组中删除。
QueryDeviceGroupInfo	iot:QueryDeviceGroupInfo	*	查询分组详情。
QueryDeviceGroupList	iot:QueryDeviceGroupList	*	查询分组列表。
SetDeviceGroupTags	iot:SetDeviceGroupTags	*	添加或更新分组标签。
QueryDeviceGroupTagList	iot:QueryDeviceGroupTagList	*	查询分组标签列表。
QueryDeviceGroupByDevice	iot:QueryDeviceGroupByDevice	*	查询指定设备所在的分组列表。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
QueryDeviceListByDeviceGroup	iot:QueryDeviceListByDeviceGroup	*	查询分组中的设备列表。
QuerySuperDeviceGroup	iot:QuerySuperDeviceGroup	*	根据子分组ID查询父分组信息。
QueryDeviceGroupByTags	iot:QueryDeviceGroupByTags	*	根据标签查询设备分组。
StartRule	iot:StartRule	*	启动规则。
StopRule	iot:StopRule	*	暂停规则。
ListRule	iot:ListRule	*	查询规则列表。
GetRule	iot:GetRule	*	查询规则详情。
CreateRule	iot:CreateRule	*	创建规则。
UpdateRule	iot:UpdateRule	*	修改规则。
DeleteRule	iot:DeleteRule	*	删除规则。
CreateRuleAction	iot:CreateRuleAction	*	创建规则中的数据转发方法。
UpdateRuleAction	iot:UpdateRuleAction	*	修改规则中的数据转发方法。
DeleteRuleAction	iot:DeleteRuleAction	*	删除规则中的数据转发方法。
GetRuleAction	iot:GetRuleAction	*	查询规则中的数据转发方法的详细信息。
ListRuleActions	iot:ListRuleActions	*	获取规则中的数据转发方法列表。
Pub	iot:Pub	*	发布消息。
PubBroadcast	iot:PubBroadcast	*	向订阅了指定产品广播Topic的所有设备发送消息。
RRpc	iot:RRpc	*	发送消息给设备并得到设备响应。
CreateProductTopic	iot:CreateProductTopic	*	创建产品Topic类。
DeleteProductTopic	iot:DeleteProductTopic	*	删除产品Topic类。
QueryProductTopic	iot:QueryProductTopic	*	查询产品Topic类列表。
UpdateProductTopic	iot:UpdateProductTopic	*	修改产品Topic类。

IoT API	RAM 授权操作（Action）	资源（Resource）	接口说明
CreateTopicRouteTable	iot:CreateTopicRouteTable	*	新建Topic间的消息路由关系。
DeleteTopicRouteTable	iot:DeleteTopicRouteTable	*	删除Topic路由关系。
QueryTopicReverseRouteTable	iot:QueryTopicReverseRouteTable	*	查询指定Topic订阅的源Topic。
QueryTopicRouteTable	iot:QueryTopicRouteTable	*	查询向指定Topic订阅消息的目标Topic。
GetDeviceShadow	iot:GetDeviceShadow	*	查询设备的影子信息。
UpdateDeviceShadow	iot:UpdateDeviceShadow	*	修改设备的影子信息。

9.2.4 子账号访问

子账号用户可以登录物联网控制台管理您的 IoT 资源，和使用子账号的 AccessKeyId 和AccessKeySecret 调用 IoT API。

您需先创建子账号，并通过授权策略授予子账号访问物联网平台的权限。创建自定义授权策略的方法，请参见自定义权限。

创建子账号

如果您已有子账号，请忽略此操作。

1. 用主账号登录访问控制 *RAM* 控制台。
2. 在左侧导航栏中，单击用户管理。
3. 单击新建用户。
4. 输入用户信息，并勾选为该用户自动生成 AccessKey前的复选框，再单击确定。

> 说明:
> 单击确定后，弹出保存 AccessKey 的提示。这是下载该子账号的 AccessKey 的唯一机会。请及时保存该 AccessKey 信息，并妥善保管。子账号用户调用 API 时，需传入该 AccessKey 信息。

5. 设置初始登录密码。

   a. 在用户管理页面，找到刚创建的用户名，单击管理，进入用户详情页面。

   b. 单击启用控制台登录。

   c. 为该子账号设置一个初始密码，并勾选要求该账号下次登录成功后重置密码前的复选框，再单击确定。

6. 启用多因素认证设备（可选）。

   在用户详情页面，单击启用虚拟MFA设备。

子账号创建完成后，子账号用户便可通过RAM 用户登录链接登录阿里云官网和控制台。RAM 用户的登录链接，请在访问控制 RAM 控制台的概览页面查看。

但是，在获得授权之前，该子账号无法访问您的阿里云资源。下一步，为子账号授予物联网平台的访问权限。

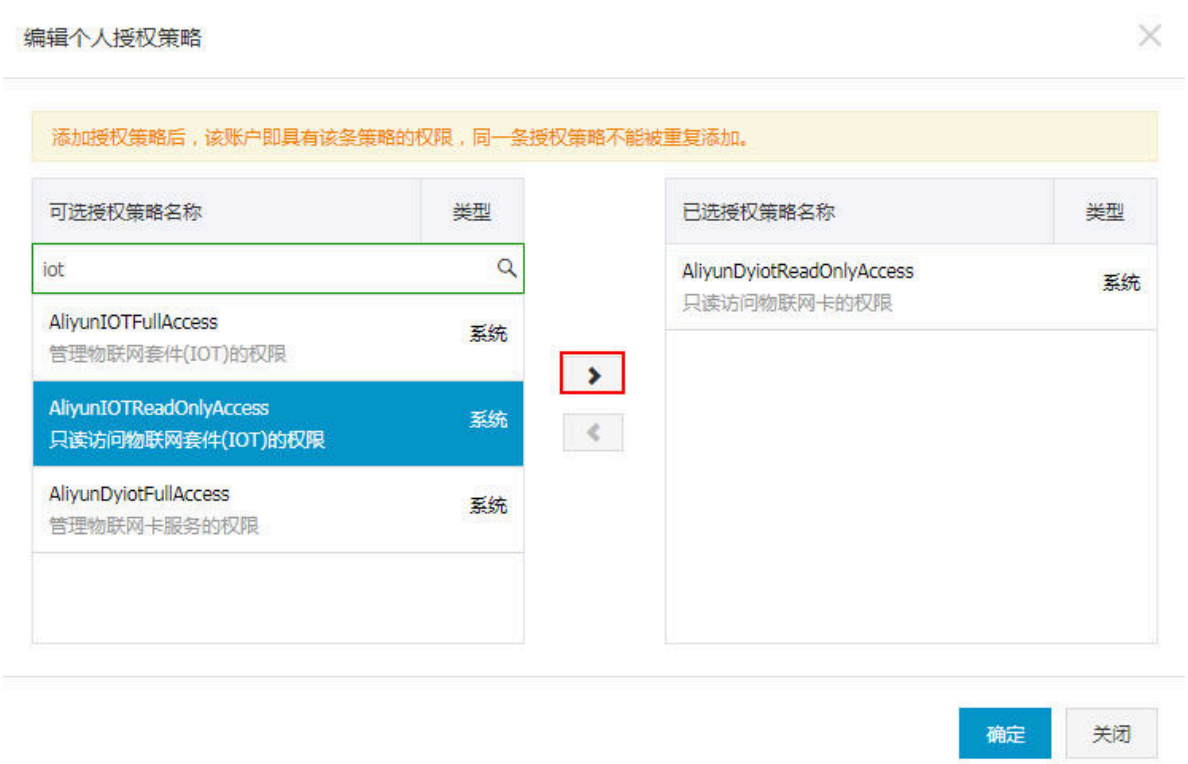
授权子账号访问 IoT

在访问控制 RAM 控制台中，您可以在用户管理页面，为单个子账号进行授权；也可以在群组管理页面，为整个群组授予相同的权限。下面我们以为单个子账号授权为例，介绍授权操作流程。

1. 用主账号登录访问控制 *RAM* 控制台。
2. 在左侧导航栏中，单击用户管理。
3. 找到要授权的子账号的用户名，单击授权操作按钮。
4. 在授权对话框中，选择您要授予该子账号的 IoT 授权策略，单击页面中间向右箭头将选中权限移入已选授权策略名称，再单击确定。

说明:

如果您要为子账号授予自定义权限，请先创建授权策略。授权策略的创建方法，请参见自定义权限。



授权成功后，子账号用户便可访问授权策略中定义的资源，和进行授权策略中定义的操作。

子账号登录控制台

阿里云主账号登录是从阿里云官网主页直接登录，但是子账号需从RAM 用户登录页面登录。

1. 获取RAM 用户登录页面的链接地址。

   用主账号登录访问控制 RAM 控制台，在概览页面查看RAM 用户登录链接，并将链接地址分发给子账号的用户。

2. 子账号用户访问RAM 用户登录页面，并使用子账号和子账号密码登录。

   说明:
   子账号登录格式为：子用户名称@企业别名，如：username@company-alias。并且，首次登录成功后，需修改登录密码。

3. 单击页面右上角控制台按钮，进入管理控制台。
4. 单击产品与服务，选择物联网平台，即可进入物联网控制台。

子账号用户登录物联网控制台后，便可在控制台中，进行已获授权的操作。

9.2.5 进阶使用 STS

STS 权限管理系统是比访问控制（RAM）更为严格的权限管理系统。使用 STS 权限管理系统进行资源访问控制，需通过复杂的授权流程，授予子账号用户临时访问资源的权限。

子账号和授予子账号的权限均长期有效。删除子账号或解除子账号权限，均需手动操作。发生子账号信息泄露后，如果无法及时删除该子账号或解除权限，可能给您的阿里云资源和重要信息带来危险。所以，对于关键性权限或子账号无需长期使用的权限，您可以通过 STS 权限管理系统来进行控制。

图 9-1: 子账号获得临时访问权限的操作流程

步骤一：创建角色

RAM 角色是一种虚拟用户，是承载操作权限的虚拟概念。

1. 使用阿里云主账号登录访问控制 *RAM* 控制台。
2. 单击角色管理 > 新建角色，进入角色创建流程。
3. 选择用户角色。
4. 填写信息类型步骤中，直接使用默认的当前云账号信息，单击下一步。
5. 输入角色名称和备注后，单击创建。
6. 单击关闭或授权。

   如果您已创建要授予该角色的授权策略，则单击 授权，并对角色进行授权。

   如果您还没有创建授权策略，则单击关闭。然后，在策略管理中为该角色新建授权策略。

步骤二：创建角色授权策略

角色授权策略，即定义要授予角色的资源访问权限。

1. 在访问控制 *RAM* 控制台主页，单击策略管理 > 新建授权策略。
2. 选择空白模板。
3. 输入授权策略名称和策略内容，再单击新建授权策略。

   授权策略内容的编写方法参考，请单击授权策略格式定义查看。

   授权策略内容示例：IoT 资源只读权限。

```
{
```

```
    "Version":"1",
    "Statement":[
        {
            "Action":[
                "rds:DescribeDBInstances",
                "rds:DescribeDatabases",
                "rds:DescribeAccounts",
                "rds:DescribeDBInstanceNetInfo"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":"ram:ListRoles",
            "Effect":"Allow",
            "Resource":"*"
        },
        {
            "Action":[
                "mns:ListTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "dhs:ListProject",
                "dhs:ListTopic",
                "dhs:GetTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "ots:ListInstance",
                "ots:ListTable",
                "ots:DescribeTable"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "log:ListShards",
                "log:ListLogStores",
                "log:ListProject"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Effect":"Allow",
            "Action":[
                "iot:Query*",
                "iot:List*",
                "iot:Get*",
                "iot:BatchGet*"
            ],
            "Resource":"*"
        }
    ]
```

```
}
```

授权策略内容示例：IoT 资源读写权限。

```
{
    "Version":"1",
    "Statement":[
        {
            "Action":[
                "rds:DescribeDBInstances",
                "rds:DescribeDatabases",
                "rds:DescribeAccounts",
                "rds:DescribeDBInstanceNetInfo"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":"ram:ListRoles",
            "Effect":"Allow",
            "Resource":"*"
        },
        {
            "Action":[
                "mns:ListTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "dhs:ListProject",
                "dhs:ListTopic",
                "dhs:GetTopic"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "ots:ListInstance",
                "ots:ListTable",
                "ots:DescribeTable"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Action":[
                "log:ListShards",
                "log:ListLogStores",
                "log:ListProject"
            ],
            "Resource":"*",
            "Effect":"Allow"
        },
        {
            "Effect":"Allow",
            "Action":"iot:*",
            "Resource":"*"
        }
    ]
```

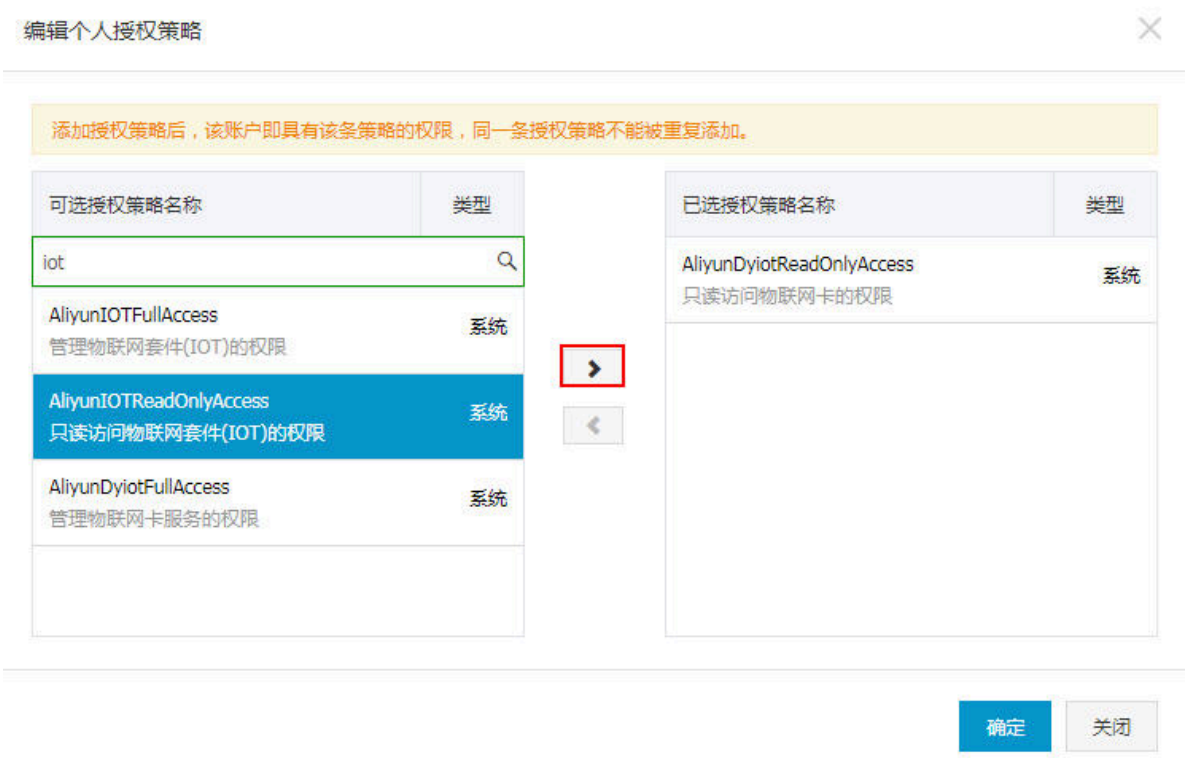
```
}
```

授权策略创建成功后，您就可以将该授权策略中定义的权限授予角色。

步骤三：为角色授权

角色获得授权后，才具有资源访问权限。

1. 在访问控制 *RAM* 控制台主页，单击角色管理。
2. 找到要授权的角色，单击授权。
3. 在授权对话框中，选择要授予角色的自定义授权策略，单击中间的向右箭头，将选中的授权策略移至已选授权策略名称下，再单击确定。



授权完成后，该角色就具有了授权策略定义的权限。您可以单击该角色对应的管理操作按钮，进入角色详情页，查看该角色的基本信息和权限信息。

下一步，为子账号授予可以扮演该角色的权限。

步骤四：授予子账号角色扮演权限

虽然经过授权后，该角色已拥有了授权策略定义的访问权限，但角色本身只是虚拟用户，需要子账号用户扮演该角色，才能进行权限允许的操作。若任意子账号都可以扮演该角色，也会带来风险，因此只有获得角色扮演权限的子账号用户才能扮演角色。

授权子账号扮演角色的方法：先新建一个Resource参数值为角色 ID 的自定义授权策略，然后用该授权策略为子账号授权。

1. 在访问控制 *RAM* 控制台主页，单击策略管理 > 新建授权策略。
2. 选择空白模板。
3. 输入授权策略名称和策略内容，再单击新建授权策略。

> 说明:
> 授权策略内容中，参数Resource 的值需为角色 Arn。在角色管理页面，单击角色对应的管理按钮，进入角色详情页面中，查看角色的 Arn 。

角色授权策略示例:

```
{
    "Version":"1",
    "Statement":[
        {
            "Effect":"Allow",
            "Action":"iot:QueryProduct",
            "Resource":"角色Arn"
        }
    ]
}
```

4. 授权策略创建成功后，返回访问控制 RAM 控制台主页。
5. 单击左侧导航栏中的用户管理按钮，进入子账号管理页面。
6. 找到要授权的子账号，并单击对应的授权按钮。
7. 在授权对话框中，选择刚新建的角色授权策略，单击中间的向右箭头将授权策略移至已选授权策略名称下，再单击确定。

授权完成后，子账号便有了可以扮演该角色的权限，就可以使用 STS 获取扮演角色的临时身份凭证，和进行资源访问。

步骤五：子账号获取临时身份凭证

获得角色授权的子账号用户，可以通过直接调用 API 或使用 SDK 来获取扮演角色的临时身份凭证：AccessKeyId、AccessKeySecret、和 SecurityToken。STS API 和 STS SDK 详情，请参见访问控制文档中*API* 参考（*STS*）和*SDK* 参考（*STS*）。

使用 API 和 SDK 获取扮演角色的临时身份凭证需传入以下参数:

- RoleArn：需要扮演的角色 Arn。
- RoleSessionName：临时凭证的名称（自定义参数）。
- Policy：授权策略，即为角色增加一个权限限制。通过此参数限制生成的Token的权限。不指定此参数，则返回的 Token 将拥有指定角色的所有权限。
- DurationSeconds：临时凭证的有效期。单位是秒，最小为 900，最大为 3600，默认值是 3600。

· id 和 secret：指需要扮演该角色的子账号的 AccessKeyId 和 AccessKeySecret。

获取临时身份凭证示例

API 示例：子账号用户通过调用 STS 的AssumeRole接口获得扮演该角色的临时身份凭证。

```
https://sts.aliyuncs.com?Action=AssumeRole
&RoleArn=acs:ram::1234567890123456:role/iotstsrole
&RoleSessionName=iotreadonlyrole
&DurationSeconds=3600
&Policy=<url_encoded_policy>
&<公共请求参数>
```

SDK 示例：子账号用户使用 STS 的 Python 命令行工具接口获得扮演该角色的临时身份凭证。

```
$python ./sts.py AssumeRole RoleArn=acs:ram::1234567890123456:role/
iotstsrole RoleSessionName=iotreadonlyrole Policy='{"Version":"1","
Statement":[{"Effect":"Allow","Action":"iot:*","Resource":"*"}]}'
DurationSeconds=3600 --id=id --secret=secret
```

请求成功后，将返回扮演该角色的临时身份凭证：AccessKeyId、AccessKeySecret、和 SecurityToken。

步骤六：子账号临时访问资源

获得扮演角色的临时身份凭证后，子账号用户便可以在调用 SDK 的请求中传入该临时身份凭证信息，扮演角色。

Java SDK 示例：子账号用户在调用请求中，传入临时身份凭证的 AccessKeyId、AccessKeySecret 和 SecurityToken 参数，创建 IAcsClient 对象。

```
IClientProfile  profile = DefaultProfile.getProfile("cn-hangzhou",
AccessKeyId,AccessSecret) ;
RpcAcsRequest request.putQueryParameter("SecurityToken", Token);
IAcsClient client = new DefaultAcsClient(profile);
AcsResponse response = client.getAcsResponse(request);
```