阿里云
物联网平台

设备端开发指南

文档版本：20190515

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 下载设备端SDK

物联网平台提供各类设备端SDK，简化开发过程，使设备快速上云。

前提条件

在设备端开发之前，您需要首先完成控制台所需操作，以获取设备开发阶段的必要信息，包括设备信息、Topic信息等。具体内容请参考用户指南。

基于设备端SDK开发

您可以在设备中集成物联网平台提供的SDK，实现物联网平台接入。

设备接入流程请参考设备接入引导。

实际开发中，请根据开发时使用的语言、平台，选用合适的设备端SDK。Link Kit SDK包含：

- *C SDK*
- *Android SDK*
- *NodeJS SDK*
- *Java SDK*
- *Python SDK*
- *iOS SDK*

> **说明:**
> 不同语言、平台的SDK功能可能有所不同，具体请查询各*SDK*支持的功能。

如果以上SDK不满足您的需求，可以按照以下模板发送邮件至`linkkitSDK-query@list.alibaba-inc.com`联系我们。

- 邮件主题：`SDK开发语言/平台咨询`
- 邮件内容：

```
公司名称:
联系人:
联系电话:
设备开发语言/平台:
需求描述:
贵司的产品规模及开发计划:
```

基于Alink协议开发

如果提供的设备端SDK无法满足您的需求，可以参考*Alink*协议文档自行开发。

泛化协议SDK

您可以使用泛化协议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。

- 核心SDK
- Server SDK

HTTP/2 SDK

您可以使用HTTP/2 SDK，建立设备端与阿里云物联网平台的数据通道。例如，将HTTP/2 SDK嵌入您的服务器，直接接受设备上报的消息。

- HTTP/2 SDK (Java)
- HTTP/2 SDK (.NET)

2 设备安全认证

为保障设备安全，物联网平台为设备颁发证书，包括产品证书（ProductKey和ProductSecret）与设备证书（DeviceName和DeviceSecret）。其中，设备证书与设备一一对应，以确保设备的唯一合法性。设备通过协议接入IoT Hub之前，需依据不同的认证方案，上报产品证书和设备证书，云端通过认证后，方可接入物联网平台。针对不同的使用环境，物联网平台提供了多种认证方案。

物联网平台目前提供三种认证方案，分别是：

- 一机一密：每台设备烧录自己的设备证书。
- 一型一密：同一产品下设备烧录相同产品证书。
- 子设备认证：网关连接上云后，子设备的认证方案。

三种方案在易用性和安全性上各有优势，您可以根据设备所需的安全等级和实际的产线条件灵活选择，方案对比如下图所示。

表 2-1: 认证方案对比

对比项	一机一密	一型一密	子设备注册
设备端烧录信息	ProductKey、DeviceName、DeviceSecret	ProductKey、ProductSecret	ProductKey
云端是否需要开启	无需开启，默认支持。	需打开动态注册开关	需打开动态注册开关
是否需要预注册DeviceName	需要，确保产品下DeviceName唯一	需要，确保产品下DeviceName唯一	需要预注册
产线烧录要求	逐一烧录设备证书，需确保设备证书的安全性	批量烧录相同的产品证书，需确保产品证书的安全存储	子设备批量烧录相同的产品证书，但需要确保网关的安全性
安全性	较高	一般	一般
是否有配额限制	有，单个产品50万上限	有，单个产品50万上限	有，单个网关最多可注册200个子设备
其他外部依赖	无	无	依赖网关的安全性保障

2.1 概览

物联网平台使用ProductKey标识产品，DeviceName标识设备，设备证书（ProductKey、DeviceName和DeviceSecret）校验设备合法性。设备通过协议接入物联网

平台之前，需依据不同的认证方法，上报产品与设备信息，认证通过后，方可接入物联网平台。针对不同的使用环境，物联网平台提供了多种认证方案。

物联网平台目前提供三种认证方案，分别是：

- 一机一密：每台设备烧录自己的设备证书（ProductKey、DeviceName和DeviceSecret）。
- 一型一密：同一产品下设备烧录相同产品证书（ProductKey和ProductSecret）。
- 子设备动态注册：网关连接上云后，子设备通过动态注册获取DeviceSecret。

三种方案在易用性和安全性上各有优势，您可以根据设备所需的安全等级和实际的产线条件灵活选择，方案对比如下图所示。

表 2-2: 认证方案对比

对比项	一机一密	一型一密	子设备动态注册
设备端烧录信息	ProductKey、DeviceName、DeviceSecret	ProductKey、ProductSecret	ProductKey
云端是否需要开启动态注册	无需开启，默认支持。	需打开动态注册开关	需打开动态注册开关
是否需要预注册DeviceName	需要，确保产品下DeviceName唯一	需要，确保产品下DeviceName唯一	需要，确保产品下DeviceName唯一
产线烧录要求	逐一烧录设备证书，需确保设备证书的安全性	批量烧录相同的产品证书，需确保产品证书的安全存储	· 网关可以本地获取子设备ProductKey · 将子设备ProductKey烧录在网关上
安全性	较高	一般	一般
是否有配额限制	有，单个产品50万上限	有，单个产品50万上限	有，单个网关最多可注册1500个子设备
其他外部依赖	无	无	依赖网关的安全性保障

2.2 一机一密

一机一密认证方法，即预先为每个设备烧录其唯一的设备证书（ProductKey、DeviceName和DeviceSecret）。当设备与物联网平台建立连接时，物联网

平台对其携带的设备证书信息进行认证。认证通过，物联网平台激活设备，设备与物联网平台间才可传输数据。

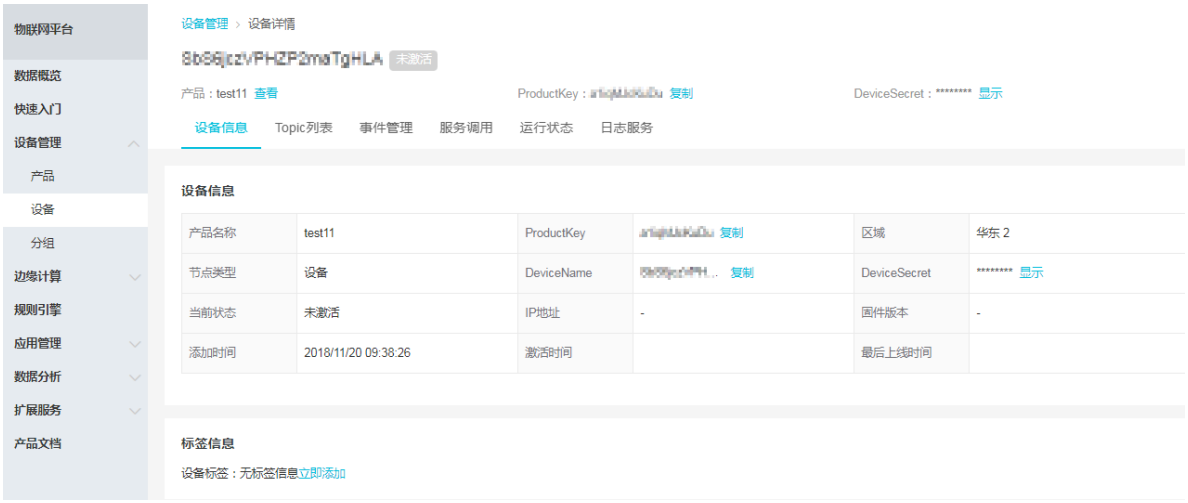
背景信息

一机一密认证方式的安全性较高，推荐使用。

使用流程示意图：

操作步骤

1. 在*IoT*控制台，创建产品。如需帮助，请参考用户指南中，创建产品、批量创建设备或单个创建设备。
2. 在产品下，添加设备，并获取设备证书信息。



3. 将设备证书信息烧录至设备中。

   烧录设备证书信息操作步骤如下：

   a) 下载设备端SDK。

   b) 初始化设备端SDK。在设备端SDK中，填入设备的证书信息（ProductKey、DeviceName和DeviceSecret）。

      初始化一机一密认证方式的设备端SDK，请参见*Link Kit SDK*文档中，各语言SDK的《认证与连接》文档。

   c) 根据实际需求，完成设备端SDK开发，如，OTA开发、子设备接入、设备物模型开发、设备影子开发等。

   d) 在产线上，将已开发完成的设备SDK烧录至设备中。

4. 设备上电联网后，向物联网平台发起认证激活请求。
5. 物联网平台对设备证书进行校验。认证通过后，与设备建立连接，设备便可通过发布消息至Topic和订阅Topic消息，与物联网平台进行数据通信。

2.3 一型一密

一型一密安全认证方式下，同一产品下所有设备可以烧录相同固件（即烧录ProductKey和ProductSecret）。设备发送激活请求时，物联网平台进行身份确认，认证通过，下发该设备对应的DeviceSecret。

背景信息

> **说明:**
> 采用一型一密方式认证，设备烧录相同固件，存在产品证书泄露风险。您可以在产品详情页面，手动关闭动态注册开关，拒绝新设备的认证请求。

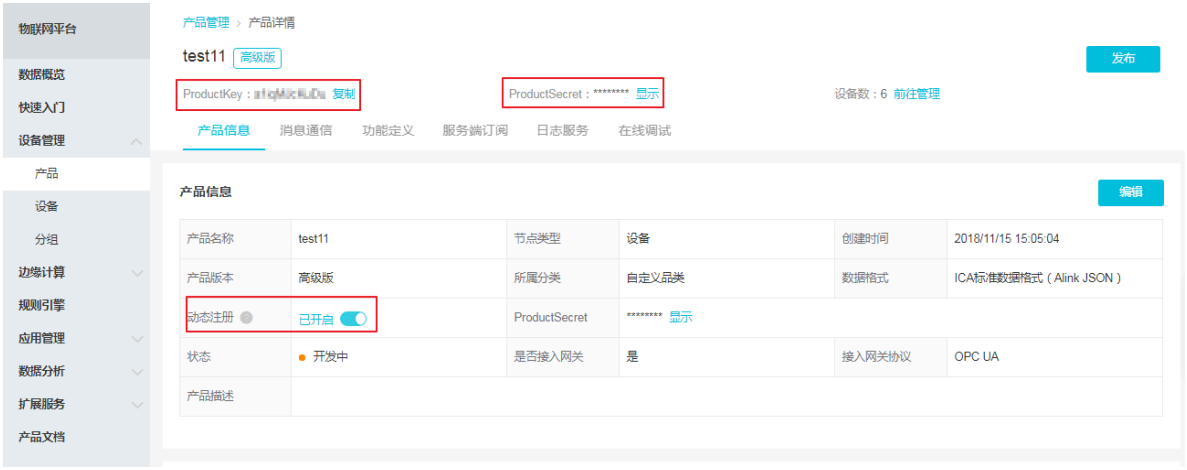
使用流程示意图：

操作步骤

1. 在*IoT*控制台，创建产品。如需帮助，请参考用户指南中，创建产品、批量创建设备或单个创建设备。
2. 在已创建产品的产品详情页面，开启动态注册开关。系统将进行短信验证，以确认是您本人操作。

> **说明:**

若设备发出激活请求时，系统校验发现该开关未开启，将拒绝新设备的动态激活请求。已激活设备不受影响。



3. 在该产品下，添加设备。添加成功的设备状态为未激活。

   因设备激活时会校验DeviceName，建议您采用可以直接从设备中读取到的ID，如设备的MAC地址、IMEI或SN号等，作为DeviceName使用。

4. 为设备烧录产品证书。

   烧录产品证书操作步骤如下：

   a) 下载设备端SDK。

   b) 初始化设备端SDK，开通设备端SDK一型一密注册。在设备端SDK中，填入产品证书（ProductKey和ProductSecret）。

      开通设备端SDK一型一密注册，请参见*Link Kit SDK*文档中，各语言SDK的《认证与连接》文档。

   c) 根据实际需求，完成设备端SDK开发，如，OTA开发、子设备接入、设备物模型开发、设备影子开发等。

   d) 在产线上，将已开发完成的设备SDK烧录至设备中。

5. 设备上电联网后，通过一型一密发起认证请求。

6. 物联网平台校验通过后，动态下发该设备对应的DeviceSecret。至此，设备获得连接云端所需的设备证书信息（ProductKey、DeviceName和DeviceSecret），可以与云端建立连接，通过发布消息到Topic和订阅Topic消息，与云端进行数据通信。

说明：

设备在物联网平台有且仅有唯一一组设备证书信息（唯一一组ProductKey、DeviceName和DeviceSecret）。

未激活的设备，使用ProductKey、DeviceName，可以反复注册获取DeviceSecret，每次获取的DeviceSecret都不同。

已激活的设备，DeviceSecret唯一。若需要重新激活该设备，请首先在物联网平台上删除设备，重新注册，使用新的ProductKey、DeviceName获取DeviceSecret。

3 设备多协议连接

3.1 MQTT协议规范

MQTT是基于TCP/IP协议栈构建的异步通信消息协议，是一种轻量级的发布/订阅信息传输协议。MQTT在时间和空间上，将消息发送者与接受者分离，可以在不可靠的网络环境中进行扩展。适用于设备硬件存储空间有限或网络带宽有限的场景。物联网平台支持设备使用MQTT协议接入。

支持版本

目前阿里云支持MQTT标准协议接入，兼容3.1.1和3.1版本协议，具体的协议请参考 *MQTT 3.1.1* 和 *MQTT 3.1* 协议文档。

与标准MQTT的区别

- 支持MQTT 的 PUB、SUB、PING、PONG、CONNECT、DISCONNECT和UNSUB等报文。
- 支持clean session。
- 不支持will、retain msg。
- 不支持QoS2。
- 基于原生的MQTT Topic上支持RRPC同步模式，服务器可以同步调用设备并获取设备回执结果。

安全等级

- TCP通道基础 + TLS协议（TLSV1、 TLSV1.1和TLSV1.2 版本）：安全级别高。
- TCP通道基础＋芯片级加密（ID^{2}硬件集成）： 安全级别高。
- TCP通道基础＋对称加密（使用设备私钥做对称加密）：安全级别中。
- TCP方式（数据不加密）： 安全级别低。

Topic规范

Topic定义及分类，请查看什么是*Topic*。

系统默认通信类Topic可前往控制台设备详情页查看，功能类Topic可前往具体功能文档页查看。

3.2 MQTT-TCP连接通信

本文档主要介绍基于TCP的MQTT连接，并提供了两种连接方式：MQTT客户端直连和使用HTTPS认证再连接。

> 说明:
> 在进行MQTT CONNECT协议设置时，注意：
> - 如果同一个设备证书（ProductKey、DeviceName和DeviceSecret）同时用于多个物理设备连接，可能会导致客户端频繁上下线。因为新设备连接认证时，原使用该证书已连接的设备会被迫下线，而设备被下线后，又会自动尝试重新连接。
> - MQTT连接模式中，默认开源SDK下线后会自动重连。您可以通过日志服务查看设备行为。

MQTT客户端直连

1. 推荐使用TLS加密。如果使用TLS加密，需要下载根证书。
2. 使用MQTT客户端连接服务器。连接方法，请参见开源MQTT客户端参考。如果需了解MQTT协议，请参考 *http://mqtt.org* 相关文档。

   > 说明:
   > 若使用第三方代码，阿里云不提供技术支持。

3. MQTT 连接。

   建议您使用设备端SDK接入物联网平台。如果您自行开发接入，连接参数如下，签名可参考MQTT连接签名示例。

连接域名	`${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com:1883` *${YourProductKey}*请替换为您的产品key。 *${YourRegionId}*请参考地域和可用区替换为您的Region ID。

可变报头（variable header）：Keep Alive	Connect指令中需包含Keep Alive（保活时间）。保活心跳时间取值范围为30至1200秒。如果心跳时间不在此区间内，物联网平台会拒绝连接。建议取值300秒以上。如果网络不稳定，将心跳时间设置高一些。

MQTT的Connect报文参数	`mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,timestamp=132323232|"`
	`mqttUsername: deviceName+"&"+productKey`
	`mqttPassword: sign_hmac(deviceSecret,content)`
	
	`mqttPassword`：sign签名需把提交给服务器的参数按字典排序后，根据signmethod加签。
	content的值为提交给服务器的参数（ProductKey、DeviceName、timestamp和clientId），按照字母顺序排序，然后将参数值依次拼接。
	· clientId：表示客户端ID，建议使用设备的MAC地址或SN码，64字符内。
	· timestamp：表示当前时间毫秒值，可以不传递。
	· mqttClientId：格式中||内为扩展参数。
	· signmethod：表示签名算法类型。支持hmacmd5，hmacsha1和hmacsha256，默认为hmacmd5。
	· securemode：表示目前安全模式，可选值有2（TLS直连模式）和3（TCP直连模式）。
	
	示例：
	假设`clientId = 12345, deviceName = device, productKey = pk, timestamp = 789, signmethod=hmacsha1, deviceSecret=secret`，那么使用TCP方式提交给MQTT的参数如下：
	
	`mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=789|`
	`mqttUsername=device&pk`
	`mqttPassword=hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();`
	
	加密后的Password为二进制转16制字符串，示例结果为：
	
	`FAFD82A3D602B37FB0FA8B7892F24A477F851A14`

使用HTTPS认证再连接

1. 认证设备。

   使用HTTPS进行设备认证，认证域名为：`https://iot-auth.${YourRegionId}.aliyuncs.com/auth/devicename`。其中，*${YourRegionId}*请参考地域和可用区替换为您的Region ID。

- 认证请求参数信息：

参数	是否可选	获取方式
productKey	必选	ProductKey，从物联网平台的控制台获取。
deviceName	必选	DeviceName，从物联网平台的控制台获取。
sign	必选	签名，格式为hmacmd5(deviceSecret,content)，content值是将所有提交给服务器的参数（version、sign、resources和signmethod除外），按照字母顺序排序， 然后将参数值依次拼接（无拼接符号）。
signmethod	可选	算法类型。支持hmacmd5，hmacsha1和hmacsha256，默认为hmacmd5。
clientId	必选	表示客户端ID，64字符内。
timestamp	可选	时间戳。时间戳不做时间窗口校验。
resources	可选	希望获取的资源描述，如MQTT。 多个资源名称用逗号隔开。

- 返回参数：

参数	是否必选	含义
iotId	必选	服务器颁发的一个连接标记，用于赋值给MQTT connect报文中的username。
iotToken	必选	token有效期为7天，赋值给MQTT connect报文中的password。
resources	可选	资源信息，扩展信息比如MQTT服务器地址和CA证书信息等。

- x-www-form-urlencoded请求示例：

```
POST /auth/devicename HTTP/1.1
Host: iot-auth.cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
productKey=123&sign=123&timestamp=123&version=default&clientId=123
&resources=mqtt&deviceName=test
```

```
sign = hmac_md5(deviceSecret, clientId123deviceNametestprodu
ctKey123timestamp123)
```

- 请求响应：

```
HTTP/1.1 200 OK
Server: Tengine
Date: Wed, 29 Mar 2017 13:08:36 GMT
Content-Type: application/json;charset=utf-8
Connection: close
{
     "code" : 200,
     "data" : {
        "iotId" : "42Ze0mkxxxxxxxx1AlTP",
        "iotToken" : "0d7fdeb9dc1f4344a2xxxxxxxxcb0bcb",
        "resources" : {
            "mqtt" : {
               "host" : "xxx.iot-as-mqtt.cn-shanghai.aliyuncs.com
",
               "port" : 1883
        }
      }
      },
      "message" : "success"
}
```

2. 连接MQTT。

   a. 下载物联网平台 root.crt，建议使用TLS1.2。

   b. 设备端连接阿里云MQTT服务器地址。使用设备认证返回的MQTT地址和端口进行通信。

   c. 采用TLS建立连接。客户端通过CA证书验证物联网平台服务器；物联网平台服务器通过MQTT协议体内connect报文信息验证客户端，其中，username=iotId，password=iotToken，clientId=自定义的设备标记（建议MAC或SN）。

   如果iotId或iotToken非法，则连接失败，收到的connect ack为3。

   错误码说明如下：

   - 401: request auth error：在这个场景中，通常在签名匹配不通过时返回。
   - 460: param error：参数异常。
   - 500: unknown error：一些未知异常。
   - 5001: meta device not found：指定的设备不存在。
   - 6200: auth type mismatch：未授权认证类型错误。

MQTT保活

设备端在保活时间间隔内，至少需要发送一次报文，包括ping请求。

如果物联网平台在保活时间内无法收到任何报文，物联网平台会断开连接，设备端需要进行重连。

连接保活时间的取值范围为30至1200秒。建议取值300秒以上。

3.3 MQTT-WebSocket连接通信

物联网平台支持基于WebSocket的MQTT协议。您可以首先使用WebSocket建立连接，然后在WebSocket通道上，使用MQTT协议进行通信，即MQTT over WebSocket。

背景信息

使用WebSocket方式主要有以下优势：

- 使基于浏览器的应用程序可以像普通设备一样，具备与服务端建立MQTT长连接的能力。
- WebSocket方式使用443端口，消息可以顺利穿过大多数防火墙。

操作步骤

1. 证书准备。

   WebSocket可以使用ws和wss两种方式，ws就是普通的WebSocket连接，wss就是增加了TLS加密。如果使用wss方式进行安全连接，需要使用和TLS直连一样的根证书。

2. 客户端选择。

   直接使用官方客户端，只需要替换连接URL即可。其他语言版本客户端或者是自主接入，请参考开源MQTT客户端参考，使用前请阅读相关客户端的说明，是否支持WebSocket方式。

3. 连接说明。

   使用WebSocket方式进行连接，区别主要在MQTT连接URL的协议和端口号，MQTT连接参数和TCP直接连接方式完全相同，其中要注意securemode参数，使用wss方式连接时securemode=2，使用ws方式连接时securemode=3。

   - 连接域名，华东2节点：${YourProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com:443

     ${YourProductKey}请替换为您的产品key。
   - 可变报头（variable header）：Keep Alive

     Connect指令中需包含Keep Alive（保活时间）。保活心跳时间取值范围为30至1200秒。如果心跳时间不在此区间内，物联网平台会拒绝连接。建议取值300秒以上。如果网络不稳定，将心跳时间设置高一些。

     设备端在保活时间间隔内，至少需要发送一次报文，包括PING请求。

     如果物联网平台在保活时间内无法收到任何报文，物联网平台会断开连接，设备端需要进行重连。
   - MQTT的Connect报文参数如下：

```
mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,
timestamp=132323232|"
mqttUsername: deviceName+"&"+productKey
mqttPassword: sign_hmac(deviceSecret,content)sign签名需要把以下参数按
字典序排序后，再根据signmethod加签。
content=提交给服务器的参数（productKey,deviceName,timestamp,clientId
），按照字母顺序排序，然后将参数值依次拼接
```

其中，

- clientId：表示客户端ID，建议mac或sn，64字符内。
- timestamp：表示当前时间毫秒值，可选。
- mqttClientId：格式中||内为扩展参数。
- signmethod：表示签名算法类型。
- securemode：表示目前安全模式，可选值有2（wss协议）和3（ws协议）。

参考示例，如果预置前提如下：

```
clientId = 12345, deviceName = device,  productKey = pk,  timestamp
 = 789, signmethod=hmacsha1, deviceSecret=secret
```

· 使用ws方式

- 连接域名

```
ws://pk.iot-as-mqtt.cn-shanghai.aliyuncs.com:443
```

- 连接参数

```
mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=
789|
mqttUsername=device&pk
mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedevicep
roductKeypktimestamp789").toHexString();
```

· 使用wss方式

- 连接域名

```
wss://pk.iot-as-mqtt.cn-shanghai.aliyuncs.com:443
```

- 连接参数

```
mqttclientId=12345|securemode=2,signmethod=hmacsha1,timestamp=
789|
mqttUsername=device&pk
mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedevicep
roductKeypktimestamp789").toHexString();
```

建议您使用设备端SDK接入物联网平台。如果您自行开发接入，可参考*MQTT*连接签名示例。

3.4 MQTT连接签名示例

若您不使用阿里云提供的设备端SDK，而是使用其他方式，自己进行开发使您的设备使用MQTT协议与物联网平台连接，您可以参见本文提供的签名代码示例进行MQTT连接签名。

说明

推荐您使用阿里云提供的设备端SDK。使用阿里云提供的任何一种语言的设备端SDK，则不用您自己配置签名机制。请访问 下载设备端SDK查看阿里云提供的SDK下载路径。

如果您不使用阿里云提供的设备端SDK，而是使用其他方式将您的设备接入物联网平台，需了解：

- 需要您自己保证连接的稳定性、MQTT连接保活和MQTT连接断开重连。
- 不使用设备端SDK连接阿里云物联网平台导致的连接问题，阿里云不负责相关的技术支持。
- 如果您要使用物联网平台提供的OTA、物模型、一型一密等多种功能，需您自己去编写这些功能的实现。这将会耗费较多的开发时间、以及bug修复时间。

签名计算代码示例

若您不使用阿里云物联网平台设备端SDK，可点击以下链接，访问相关代码示例页面。

- *sign_mqtt.c*：实现签名函数的代码示例。
- *sign_api.h*：定义签名函数用到的数据结构的代码示例。
- *sign_sha256.c*：签名函数可能使用的算法实现代码示例。如果您自己的平台上有HMACSHA256的实现，可以不编译本文件，但是需要提供函数`utils_hmac_sha256()`给*sign_mqtt.c*中的API调用。
- 用于测试签名函数的代码示例。

签名函数API接口说明

函数原型	`int32_t IOT_Sign_MQTT(iotx_mqtt_region_types_t region,`
	`iotx_dev_meta_info_t *meta,`
	`iotx_sign_mqtt_t *signout);`

函数功能	根据输入的IoT设备身份认证信息，输出连接到阿里云物联网平台时所需要的域名、MQTT ClientID、MQTT Username、MQTT Password。之后，您可以将这些信息提供给MQTT Client用于连接阿里云物联网平台。

输入参数	输入参数内容包括：
	· region：指定设备需要连接的阿里云站点。
	代码示例：
	`typedef enum {`
	`IOTX_CLOUD_REGION_SHANGHAI, /* Shanghai */`
	`IOTX_CLOUD_REGION_SINGAPORE, /* Singapore */`
	`IOTX_CLOUD_REGION_JAPAN, /* Japan */`
	`IOTX_CLOUD_REGION_USA_WEST, /* America */`
	`IOTX_CLOUD_REGION_GERMANY, /* Germany */`
	`IOTX_CLOUD_REGION_CUSTOM, /* Custom setting */`
	`IOTX_CLOUD_DOMAIN_MAX /* Maximum number of domain */`
	`} iotx_mqtt_region_types_t;`
	· meta：指定设备的身份认证信息。
	说明：
	API调用者需为meta分配内存。
	代码示例：
	`typedef struct _iotx_dev_meta_info {`
	`char product_key[IOTX_PRODUCT_KEY_LEN + 1];`
	`char product_secret[IOTX_PRODUCT_SECRET_LEN + 1];`
	`char device_name[IOTX_DEVICE_NAME_LEN + 1];`
	`char device_secret[IOTX_DEVICE_SECRET_LEN + 1];`
	`} iotx_dev_meta_info_t;`
	其中包含的参数：
	- product_key：设备所属产品的ProductKey。
	- product_secret：设备所属产品的ProductSecret。
	- device_name：设备名称DeviceName。
	- device_secret：设备的DeviceSecret。

输出参数	signout：输出的数据，该数据将用于MQTT连接。
	代码示例：
	`typedef struct {`
	`char hostname[DEV_SIGN_HOSTNAME_MAXLEN];`
	`uint16_t port;`
	`char clientid[DEV_SIGN_CLIENT_ID_MAXLEN];`
	`char username[DEV_SIGN_USERNAME_MAXLEN];`
	`char password[DEV_SIGN_PASSWORD_MAXLEN];`
	`} iotx_sign_mqtt_t;`
	其中包含参数：
	· hostname：完整的阿里云物联网站点域名。
	· port：阿里云站点的端口号。
	· clientid：MQTT建立连接时需要指定的ClientID。
	· username：MQTT建立连接时需要指定的Username。
	· password：MQTT建立连接时需要指定的Password。

返回值	· 0：表示成功。 · -1：表示输入参数非法而失败。

签名API使用示例

以下以sign_test.c中的测试代码为例。

```
#include <stdio.h>
#include <string.h>
#include "sign_api.h"  //包含签名所需的各种数据结构定义

//下面的几个宏用于定义设备的阿里云身份认证信息：ProductKey、ProductSecret、
DeviceName、DeviceSecret
//在实际产品开发中，设备的身份认证信息应该是设备厂商将其加密后存放于设备Flash中或者
某个文件中，
//设备上电时将其读出后使用
#define EXAMPLE_PRODUCT_KEY     "a1X2bEn****"
#define EXAMPLE_PRODUCT_SECRET  "7jluWm1zql7b****"
#define EXAMPLE_DEVICE_NAME     "example1"
#define EXAMPLE_DEVICE_SECRET   "ga7XA6KdlEeiPXQPpRbAjOZXwG8y****"

int main(int argc, char *argv[])
{
    iotx_dev_meta_info_t meta_info;
    iotx_sign_mqtt_t sign_mqtt;

    memset(&meta_info, 0, sizeof(iotx_dev_meta_info_t));
    //下面的代码是将上面静态定义的设备身份信息赋值给meta_info
    memcpy(meta_info.product_key, EXAMPLE_PRODUCT_KEY, strlen(
EXAMPLE_PRODUCT_KEY));
    memcpy(meta_info.product_secret, EXAMPLE_PRODUCT_SECRET, strlen(
EXAMPLE_PRODUCT_SECRET));
    memcpy(meta_info.device_name, EXAMPLE_DEVICE_NAME, strlen(
EXAMPLE_DEVICE_NAME));
    memcpy(meta_info.device_secret, EXAMPLE_DEVICE_SECRET, strlen(
EXAMPLE_DEVICE_SECRET));
```

```
    //调用签名函数，生成MQTT连接时需要的各种数据
    IOT_Sign_MQTT(IOTX_CLOUD_REGION_SHANGHAI, &meta_info, &sign_mqtt);

    ...

}
```

3.5 CoAP协议规范

协议版本

支持 RFC 7252 Constrained Application Protocol协议，具体请参考：*RFC 7252*。

通道安全

使用 DTLS v1.2保证通道安全，具体请参考：*DTLS v1.2*。

开源客户端参考

http://coap.technology/impls.html。

说明:

若使用第三方代码, 阿里云不提供技术支持

阿里CoAP约定

- 暂时不支持资源发现。
- 仅支持UDP协议，目前支持DTLS和对称加密两种安全模式。
- URI规范，CoAP的URI资源和MQTT Topic保持一致，参考*MQTT规范*。

说明与限制

- Topic规范和MQTT Topic一致，CoAP协议内 `coap://host:port/topic/${topic}`接口对于所有`${topic}`和MQTT Topic可以复用。
- 客户端缓存认证返回的token是请求的令牌。
- 传输的数据大小依赖于MTU的大小，建议在1 KB以内。
- 仅华东2（上海）地域支持CoAP通信。

3.6 CoAP连接通信

物联网平台支持CoAP协议连接通信。CoAP协议适用在资源受限的低功耗设备上，尤其是NB-IoT的设备使用。本文介绍基于CoAP协议进行设备接入的流程，及使用DTLS和对称加密两种认证方式下的自主接入流程。

基础流程

基于CoAP协议将NB-IoT设备接入物联网平台的流程如下图所示：

基础流程说明如下：

1. 在设备端NB-IoT模块中，集成阿里云物联网平台SDK。厂商在物联网平台的控制台申请设备证书（ProductKey、DeviceName和DeviceSecret）并烧录到设备中。
2. NB-IoT设备通过运营商的蜂窝网络进行入网。需要联系当地运营商，确保设备所属地区已经覆盖NB网络，并已具备NB-IoT入网能力。
3. 设备入网成功后，NB设备产生的流量数据及产生的费用数据，将由运营商的M2M平台管理。此部分平台能力由运营商提供。
4. 设备开发者可通过 CoAP/UDP 协议，将设备采集的实时数据上报到阿里云物联网平台，借助物联网平台，实现海量亿级设备的安全连接和数据管理能力。并且，可通过规则引擎，将数据转发至阿里云的大数据产品、云数据库、表格存储等服务中进行处理。
5. 物联网平台提供相关的数据开放接口和消息推送服务，可将数据转发到业务服务器中，实现设备资产与实际应用的快速集成。

使用对称加密自主接入

1. 连接CoAP服务器，endpoint地址为`${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`。

   其中：

   - `${YourProductKey}`：请替换为您申请的产品Key。
   - `${port}`：端口。使用对称加密时端口为5682。

2. 设备认证。

   设备认证请求：

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"a1NUjcVkHZS","deviceName":"ff1a11e7c0
8d4b3db2b1500d8e0e55","clientId":"a1NUjcVkHZS&ff1a11e7c08d4b3db2b1
500d8e0e55","sign":"F9FD53EE0CD010FCA40D14A9FEAB81E0", "seq":"10"}
```

   表 3-1: 设备认证参数说明

参数	说明
Method	请求方法。只支持POST方法。
URL	URL地址，取值： `/auth`。
Host	endpoint地址。取值格式：*`${YourProductKey}`*`.coap.cn-shanghai.link.aliyuncs.com`。其中，变量*`${YourProductKey}`*需替换为您的产品Key。
Port	端口，取值：`5682`。
Accept	设备接收的数据编码方式。目前，支持两种方式：application/json和application/cbor。
Content-Format	设备发送给物联网平台的上行数据的编码格式，目前，支持两种方式： application/json和application/cbor。

参数	说明
payload	设备认证信息内容，JSON数据格式。具体参数，请参见下表payload说明。

表 3-2: payload 说明

字段名称	是否必需	说明
productKey	是	设备三元组信息中ProductKey的值，是物联网平台为产品颁发的全局唯一标识。从物联网平台的控制台获取。
deviceName	是	设备三元组信息中DeviceName的值，在注册设备时自定义的或自动生成的设备名称。从物联网平台的控制台获取。
ackMode	否	通信模式。取值： · 0：request/response 是携带模式，即客户端发送请求到服务端后，服务端处理完业务，回复业务数据和ACK。 · 1：request/response 是分离模式，即客户端发送请求到服务端后，服务端先回复一个确认ACK，然后再处理业务后，回复业务数据。 若不传入此参数，则默认为携带模式。
sign	是	签名。 签名计算格式为`hmacmd5(DeviceSecret,content)`。 其中，`content`为将所有提交给服务器的参数（除`version`、`sign`、`resources`和`signmethod`外），按照英文字母升序，依次拼接排序（无拼接符号）。 签名计算示例： `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123seq123timestamp1524448722000)`
signmethod	否	算法类型，支持hmacmd5和hmacsha1。
clientId	是	客户端ID，长度需在64字符内。建议使用设备的的MAC地址或SN码作为clientId的值。

字段名称	是否必需	说明
timestamp	否	时间戳。目前，时间戳不做时间窗口校验。

返回结果示例：

```
{"random":"ad2b3a5eb51d64f7","seqOffset":1,"token":"MZ8m37hp01
w1SSqoDFzo0010500d00.ad2b"}
```

表 3-3: 返回参数说明

字段名称	说明
random	用于后续上、下行加密，组成加密Key。
seqOffset	认证seq偏移初始值。
token	设备认证成功后，返回的token值。

3. 上报数据。

上报数据请求：

```
POST /topic/${topic}
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(标识token), 2089(seq)
```

表 3-4: 上报数据参数说明

字段名称	是否必需	说明
Method	是	请求方法。支持POST方法。
URL	是	传入格式：`/topic/${topic}`。其中，变量*${topic}*需替换为设备数据上行Topic。
Host	是	endpoint地址。传入格式：`${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`。其中，*${YourProductKey}*需替换为设备所属产品的Key。
Port	是	端口。取值：5682。
Accept	是	设备接收的数据编码方式。目前，支持两种方式：application/json和application/cbor。

字段名称	是否必需	说明
Content-Format	是	上行数据的编码格式，服务端对此不做校验。目前，支持两种方式：application/json和application/cbor。
payload	是	待上传的数据经高级加密标准（AES）加密后的数据。
CustomOptions	是	option值有2088和2089两种类型，说明如下： · 2088：表示token，取值为设备认证后返回的token值。 说明: 每次上报数据都需要携带token信息。如果token失效，需重新认证获取token。 · 2089：表示seq，取值需比设备认证后返回的seqOffset值更大，且在认证生效周期内不重复的随机值。使用AES加密该值。 option返回示例： `number:2090(云端消息ID)`

消息上行成功后，返回成功状态码，同时返回物联网平台生成的消息ID。

使用DTLS自主接入

1. 连接CoAP服务器，endpoint 地址为`${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`。

   其中：

   · *${YourProductKey}*：请替换为您申请的产品Key。

   · *${port}*：端口。使用DTLS时，端口为5684。

2. 使用DTLS安全通道，需下载根证书。
3. 设备认证。使用auth接口认证设备，获取token。上报数据时，需携带token信息。

   设备认证请求：

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
```

```
payload: {"productKey":"ZG1EvTEa7NN","deviceName":"NlwaSPXsCp
TQuh8FxBGH","clientId":"mylight1000002","sign":"bccb3d2618afe74b3eab
12b94042f87b"}
```

参数（除 Port参数外。使用DTLS时的Port为 5684）及payload内容说明，可参见参数说明中。

返回结果示例：

```
response: {"token":"f13102810756432e85dfd351eeb41c04"}
```

表 3-5: 返回码说明

Code	描述	Payload	备注
2.05	Content	认证通过：Token对象	正确请求。
4.00	Bad Request	no payload	请求发送的Payload非法。
4.01	Unauthorized	no payload	未授权的请求。
4.03	Forbidden	no payload	禁止的请求。
4.04	Not Found	no payload	请求的路径不存在。
4.05	Method Not Allowed	no payload	请求方法不是指定值。
4.06	Not Acceptable	no payload	Accept不是指定的类型。
4.15	Unsupported Content-Format	no payload	请求的content不是指定类型。
5.00	Internal Server Error	no payload	auth服务器超时或错误。

4. 上行数据。

   设备发送数据到某个Topic。

   自定义Topic，在物联网平台的控制台，设备所属产品的产品详情页的Topic类列表栏中进行创建。

   目前，只支持发布权限的Topic用于数据上报，如`/${YourProductKey}/${YourDeviceName}/pub`。假设当前设备名称为device，产品Key为a1GFjLP3xxC，那么您可以调用

`a1GFjLP3xxC.coap.cn-shanghai.link.aliyuncs.com:5684/topic/a1GFjLP3xxC/device/pub` 地址来上报数据。

上报数据请求：

```
POST /topic/${topic}
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(标识token)
```

表 3-6: 上报数据请求参数说明

参数	是否必需	说明
Method	是	请求方法。支持POST方法。
URL	是	`/topic/${topic}`。其中，变量`${topic}`需替换为当前设备对应的Topic。
Host	是	endpoint地址。取值格式：`${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`。其中，变量`${YourProductKey}`需替换为您的产品Key。
Port	是	端口，取值：5684。
Accept	是	设备接收的数据编码方式。目前，支持两种方式：application/json和application/cbor。
Content-Format	是	上行数据的编码格式，服务端对此不做校验。目前，支持两种方式：application/json和application/cbor。
CustomOptions	是	· number取值：2088。 · token为设备认证#auth#返回的token值。 **说明:** 每次上报数据都需要携带token信息。如果token失效，需重新进行设备认证获取token。

3.7 HTTP协议规范

HTTP协议版本

· 支持 Hypertext Transfer Protocol — HTTP/1.0 协议，具体请参考：*RFC 1945*

- 支持 Hypertext Transfer Protocol — HTTP/1.1 协议，具体请参考：*RFC 2616*

通道安全

使用HTTPS(Hypertext Transfer Protocol Secure协议)保证通道安全。

- 不支持? 号形式传参数。
- 暂时不支持资源发现。
- 仅支持HTTPS。
- URI规范, HTTP的URI资源和MQTT TOPIC保持一致，参考*MQTT规范*。

3.8 HTTP连接通信

物联网平台支持使用HTTP接入，目前仅支持HTTPS协议。本文介绍使用HTTP连接通信的接入流程。

限制说明

- HTTP通信方式适合单纯的数据上报的场景。
- HTTP服务器地址为`https://iot-as-http.cn-shanghai.aliyuncs.com`。
- 目前，仅中国（上海）地域支持HTTP通信。
- 只支持HTTPS协议。
- Topic规范和*MQTT的Topic规范*一致。使用HTTP协议连接，上报数据请求：`https://iot-as-http.cn-shanghai.aliyuncs.com/topic/${topic}`。其中，Topic变量`${topic}`的值可以与MQTT连接通信的Topic 相复用。不支持以`?query_String=xxx`格式传参。
- 数据上行接口传输的数据大小限制为128 KB。
- HTTP请求只支持POST方式。
- 设备认证请求的HTTP header中的`Content-Type`必须为`application/json`。
- 数据上报请求的HTTP header中的`Content-Type`必须为`application/octet-stream`。
- 设备认证返回的token会在一定周期后失效（目前token有效期是7天），请务必考虑token失效逻辑的处理。

接入流程

接入流程主要包含进行设备认证以获取设备token和采用获取的token进行持续地数据上报。

1. 认证设备，获取设备的token。

   endpoint地址：`https://iot-as-http.cn-shanghai.aliyuncs.com`

   认证设备请求：

```
POST /auth HTTP/1.1
Host: iot-as-http.cn-shanghai.aliyuncs.com
Content-Type: application/json
body: {"version":"default","clientId":"mylight1000002","signmethod
":"hmacsha1","sign":"4870141D4067227128CBB4377906C3731CAC221C","
productKey":"ZG1EvTEa7NN","deviceName":"NlwaSPXsCpTQuh8FxBGH","
timestamp":"1501668289957"}
```

表 3-7: 参数说明

参数	说明
Method	请求方法。支持POST方法。
URL	`/auth`，URL地址，只支持HTTPS。
Host	endpoint地址：`iot-as-http.cn-shanghai.aliyuncs.com`
Content-Type	设备发送给物联网平台的上行数据的编码格式。目前只支持`application/json`。若使用其他编码格式，会返回参数错误。
body	设备认证信息。JSON数据格式。具体信息，请参见下表*body参数*。

表 3-8: body参数

字段名称	是否必需	说明
productKey	是	设备所属产品的Key。可从物联网平台的控制台设备详情页获取。
deviceName	是	设备名称。从物联网平台的控制台设备详情页获取。
clientId	是	客户端ID。长度为64字符内，建议以MAC地址或SN码作为clientId。
timestamp	否	时间戳。校验时间戳15分钟内的请求有效。时间戳格式为数值，值为自GMT 1970年1月1日0时0分到当前时间点所经过的毫秒数。

字段名称	是否必需	说明
sign	是	签名。 签名计算格式为hmacmd5(DeviceSecret,content)。 其中，content为将所有提交给服务器的参数（除version、sign和signmethod外），按照英文字母升序，依次拼接排序（无拼接符号）的结果。 签名示例： 假设clientId = 12345，deviceName = device，productKey = pk，timestamp = 789，signmethod=hmacsha1，deviceSecret=secret，那么签名计算为hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();。最后二进制数据转十六进制字符串，大小写不敏感。
signmethod	否	算法类型，支持hmacmd5和hmacsha1。 若不传入此参数，则默认为hmacmd5。
version	否	版本号。若不传入此参数，则默认default。

设备认证返回结果示例：

```
body:
{
  "code": 0,//业务状态码
  "message": "success",//业务信息
  "info": {
    "token":  "6944e5bfb92e4d4ea3918d1eda3942f6"
  }
}
```

说明：

- 请将返回的token值缓存到本地。
- 每次上报数据时，都需要携带token信息。如果token失效，需要重新认证设备获取token。

表 3-9: 错误码说明

code	message	备注
10000	common error	未知错误。
10001	param error	请求的参数异常。
20000	auth check error	设备鉴权失败。

code	message	备注
20004	update session error	更新失败。
40000	request too many	请求次数过多，流控限制。

2. 上报数据。

   发送数据到某个Topic。

   若要发送数据到自定义Topic，需先在物联网平台的控制台，设备所属产品的产品详情页，Topic类列表栏中创建Topic类。请参见自定义Topic。

   如Topic为/${YourProductKey}/${YourDeviceName}/pub。假设当前设备名称为device123，产品Key为a1GFjLPXXXX，那么您可以调用 https://iot-as-http.cn-shanghai.aliyuncs.com/topic/a1GFjLPXXXX/device123/pub地址来上报数据。

   上报数据请求：

```
POST /topic/${topic} HTTP/1.1
Host: iot-as-http.cn-shanghai.aliyuncs.com
password:${token}
Content-Type: application/octet-stream
body: ${your_data}
```

   表 3-10: 上报数据参数说明

参数	说明
Method	请求方法。支持POST方法。
URL	/topic/${topic}。其中，变量${topic}需替换为数据发往的目标Topic。只支持HTTPS。
Host	endpoint地址： iot-as-http.cn-shanghai.aliyuncs.com。
password	放在Header中的参数，取值为调用设备认证接口auth返回的token值。
Content-Type	设备发送给物联网平台的上行数据的编码格式。目前仅支持application/octet-stream。若使用其他编码格式，会返回参数错误。
body	发往${topic}的数据内容。

   返回结果示例：

```
body:
{
  "code": 0,//业务状态码
  "message": "success",//业务信息
  "info": {
```

```
    "messageId": 892687627916247040,
  }
}
```

表 3-11: 错误码说明

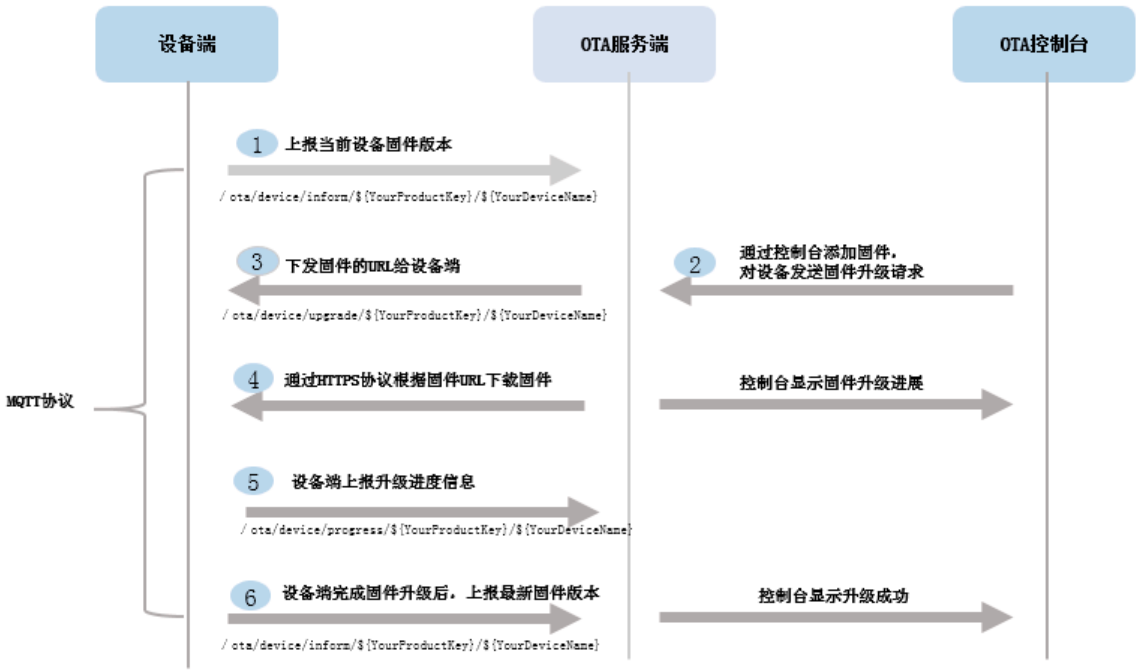
code	message	备注
10000	common error	未知错误。
10001	param error	请求的参数异常。
20001	token is expired	token失效。需重新调用auth进行鉴权，获取token。
20002	token is null	请求header中无token信息。
20003	check token error	根据token获取identify信息失败。需重新调用auth进行鉴权，获取token。
30001	publish message error	数据上行失败。
40000	request too many	请求次数过多，流控限制。

4 设备OTA升级

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

OTA固件升级流程

MQTT协议下固件升级流程如下图所示：



固件升级Topic：

- 设备端上报固件版本给物联网平台

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

- 设备端订阅该topic接收物联网平台的固件升级通知

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

- 设备端上报固件升级进度

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

说明：

- 设备固件版本号只需要在系统启动过程中上报一次即可，不需要周期循环上报。

- 从物联网平台控制台发起批量升级，设备升级操作记录状态是待升级。

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

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

数据格式说明

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

1. 设备连接OTA服务，必须上报版本号。

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

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

   - `id`：消息ID号，保留值。
   - `version`：设备当前固件版本号。
2. 在物联网平台控制台上，逐步添加固件、验证固件和发起批量升级固件。

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

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

   ```
   {
     "code": "1000",
     "data": {
       "size": 432945,
       "version": "2.0.0",
       "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com
   /nopoll_0.4.4.tar.gz?    Expires=1502955804&OSSAccessKeyId=
   ```

```
XXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&
security-  token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5
D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasu
mZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceusbFbpjzJ6xaCAGxypQ12iN%2B%
2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%2BpKWSKuGfLC1dysQcO1wEP4K
%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTknxR7ARasaBq
helc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tRjofHWmojNz
BJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McKARWct06MWV
9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK8CfNOkfTuc
MGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6IzvJsClXTnbJ
BMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2FoaGk6BAF8
hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e"
  },
  "id": 1507707025,
  "message": "success"
}
```

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

4. 设备收到URL之后，通过HTTPS协议根据URL下载固件。

> **说明:**
> 设备需在固件URL下发后的24小时内下载固件，否则该URL失效。

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

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

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

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

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

常见下载固件错误

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

5 子设备相关错误码

本文汇总了子设备上线失败、下线和消息发送失败的相关错误码。

错误信息发送

- 对于直连设备，当云端发生业务上的错误时，设备客户端可以通过TCP连接断开感知到。
- 子设备使用网关通道与物联网平台服务端通信。当子设备通过网关通道和服务端的通信出现异常时，由于网关物理通道仍然保持着，因此必须通过物理通道向子设备客户端发送错误消息，才能使子设备客户端感知到错误。

响应格式

子设备和物联网平台通信异常时，物联网平台通过网关通道发送一条MQTT错误消息给网关。发送错误消息的Topic格式，请见以下具体场景章节。

错误消息为SON格式：

```
{
id:子设备请求参数中上报的消息ID
code: 错误码(成功为200)
message: 错误信息
}
```

子设备上线失败

错误消息发送Topic：`/ext/session/{gw_productKey}/{gw_deviceName}/combine/login_reply`

表 5-1: 上线失败错误码说明

code	message	备注
460	request parameter error	参数格式错误，比如JSON格式错误，或者其他认证参数错误。
429	too many requests	认证被限流。单个设备认证过于频繁，一分钟内子设备上线次数超过5次会被限流。
428	too many subdevices under gateway	单个网关下子设备数目超过最大值。目前一个网关下，最多可有1500个子设备。
6401	topo relation not exist	子设备和当前网关之间没有拓扑关系。
6100	device not found	子设备不存在。
521	device deleted	子设备已被删除。

code	message	备注
522	device forbidden	子设备已被禁用。
6287	invalid sign	认证失败，用户名密码错误。
500	server error	云端异常。

子设备主动下线异常

发送到Topic：`/ext/session/{gw_productKey}/{gw_deviceName}/combine/logout_reply`

表 5-2: 主动下线异常

code	message	备注
460	request parameter error	参数格式错误，JSON格式错误，或者参数错误。
520	device no session	子设备会话不存在。子设备已经下线，或者没有上线过。
500	server error	云端处理异常。

子设备被踢下线

发送到Topic：`/ext/error/{gw_productKey}/{gw_deviceName}`

表 5-3: 被踢下线

code	message	备注
427	device connect in elsewhere	设备重复登录。有使用相同设备证书信息的设备连接物联网平台，导致当前连接被断开。
521	device deleted	设备已被删除。
522	device forbidden	设备已被禁用。
6401	topo relation not exist	网关和子设备的拓扑关系已被解除。

子设备发送消息失败

发送到topic：`/ext/error/{gw_productKey}/{gw_deviceName}`

表 5-4: 发送消息失败

code	message	备注
520	device session error	子设备会话错误。 · 子设备会话不存在，可能子设备没有上线，也可能已经被下线。 · 子设备会话在线，但是并不是通过当前网关会话上线的。

6 基于Alink协议开发

6.1 Alink协议

物联网平台为设备端开发提供了SDK，这些SDK已封装了设备端与物联网平台的交互协议。您可以直接使用设备端SDK来进行开发。如果嵌入式环境复杂，已提供的设备端SDK不能满足您的需求，请参考本文，自行封装Alink协议数据，建立设备与物联网平台的通信。

物联网平台为设备端开发提供的各语言SDK，请参见设备端SDK。

Alink协议是针对物联网开发领域设计的一种数据交换规范，数据格式是JSON，用于设备端和物联网平台的双向通信，更便捷地实现和规范了设备端和物联网平台之间的业务数据交互。

以下为您介绍Alink协议下，设备的上线流程和数据上下行原理。

上线流程

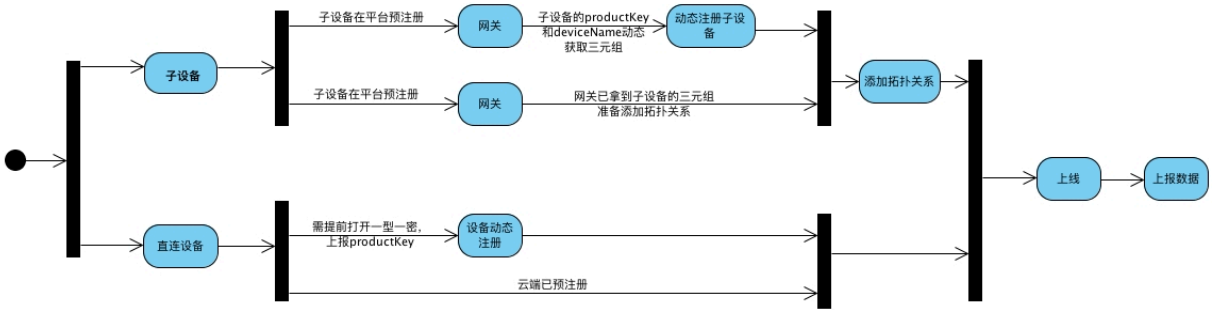
设备上线流程，可以按照设备类型，分为直连设备接入与子设备接入。主要包括：设备注册、上线和数据上报三个流程。

直连设备接入有两种方式：

- 使用一机一密方式提前烧录设备证书(ProductKey、DeviceName和DeviceSecret)，注册设备，上线，然后上报数据。
- 使用一型一密动态注册提前烧录产品证书（ProductKey和ProductSecret），注册设备，上线，然后上报数据。

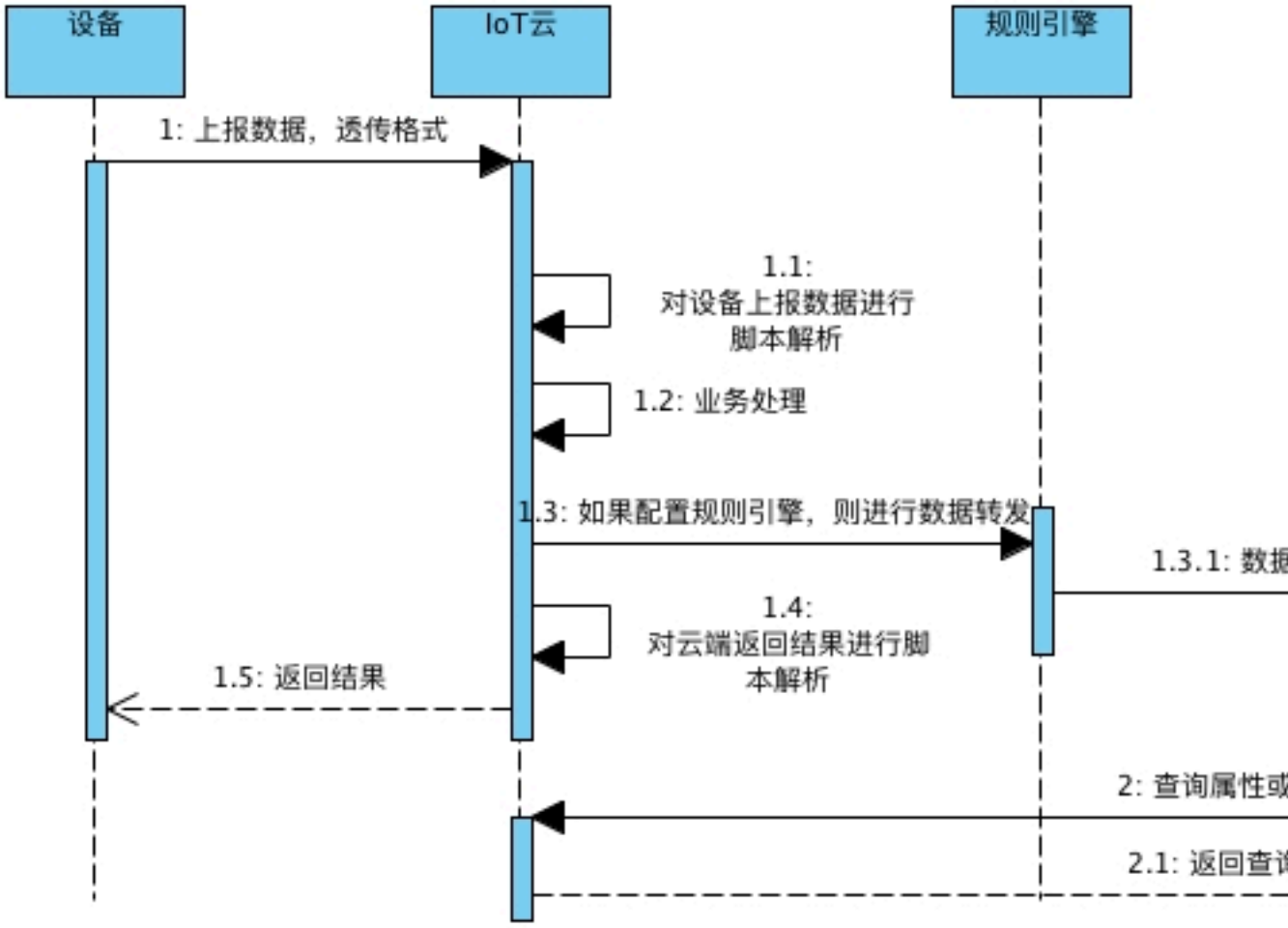
子设备接入流程通过网关发起，具体接入方式有两种：

- 使用一机一密提前烧录设备证书(ProductKey、DeviceName和DeviceSecret)，子设备上报设备证书给网关，网关添加拓扑关系，复用网关的通道上报数据。
- 使用动态注册方式提前烧录ProductKey，子设备上报ProductKey和DeviceName给网关，物联网平台校验DeviceName成功后，下发DeviceSecret。子设备将获得的设备证书信息上报网关，网关添加拓扑关系，通过网关的通道上报数据。



设备上报属性或事件

- **透传格式（透传/自定义）数据**



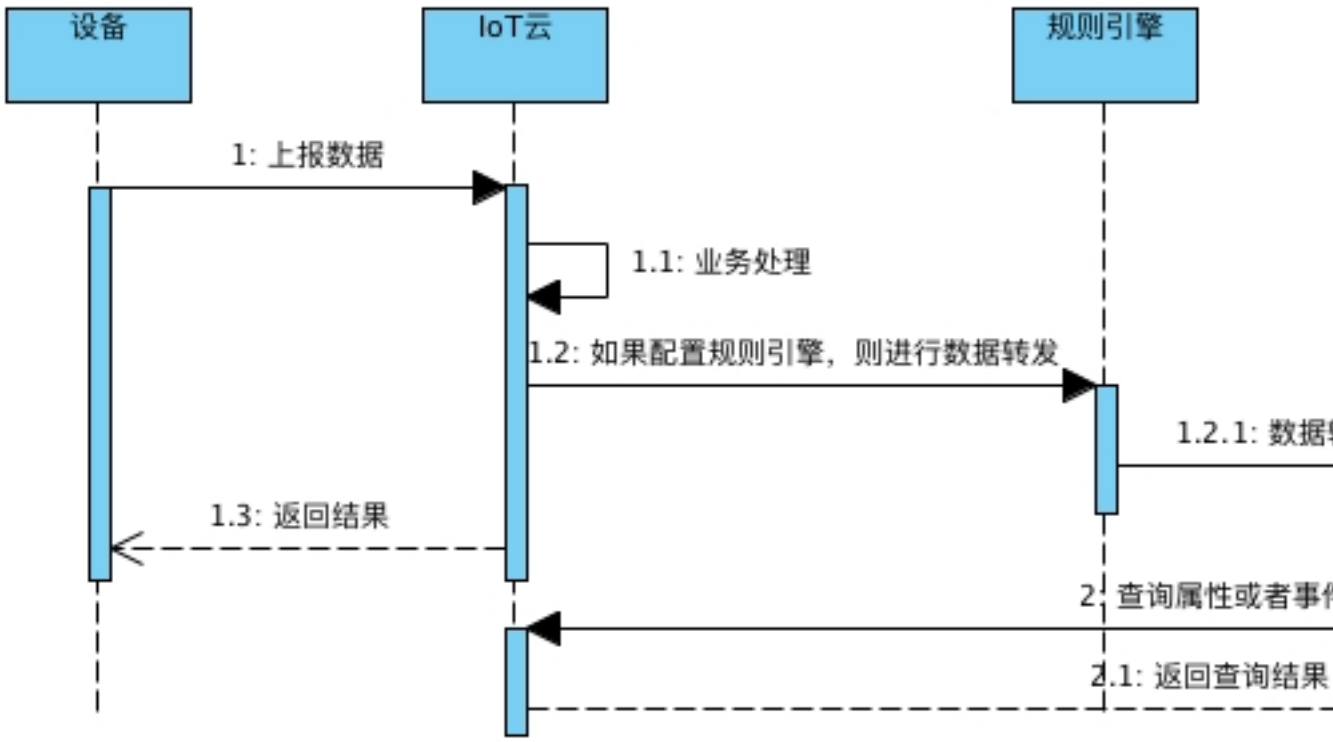
1. 设备通过透传格式数据的Topic，上报透传数据。
2. 物联网平台通过数据解析脚本先对设备上报的数据进行解析。调用脚本中的`rawDataToProtocol`方法，将设备上报的数据转换为物联网平台标准数据格式（Alink JSON格式）。
3. 物联网平台使用转换后的Alink JSON格式数据进行业务处理。

> **说明:**
> 如果配置了规则引擎数据流转，则会将数据流转到规则引擎配置的目的云产品中。

- 规则引擎获取到的数据是经过脚本解析之后的数据。
- 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/property/post`，获取设备属性。
- 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`，获取设备事件。

4. 调用数据解析脚本中的`protocolToRawData`方法，对结果数据进行格式转换，将数据解析为设备可以接收的数据格式。
5. 推送解析后的返回结果数据给设备。
6. 您可以通过*QueryDevicePropertyData*接口查询设备上报的属性历史数据，通过*QueryDeviceEventData*接口查询设备上报的事件历史数据。

· 非透传格式（Alink JSON）数据



1. 设备使用非透传格式数据的Topic，上报数据。
2. 物联网平台进行业务处理。

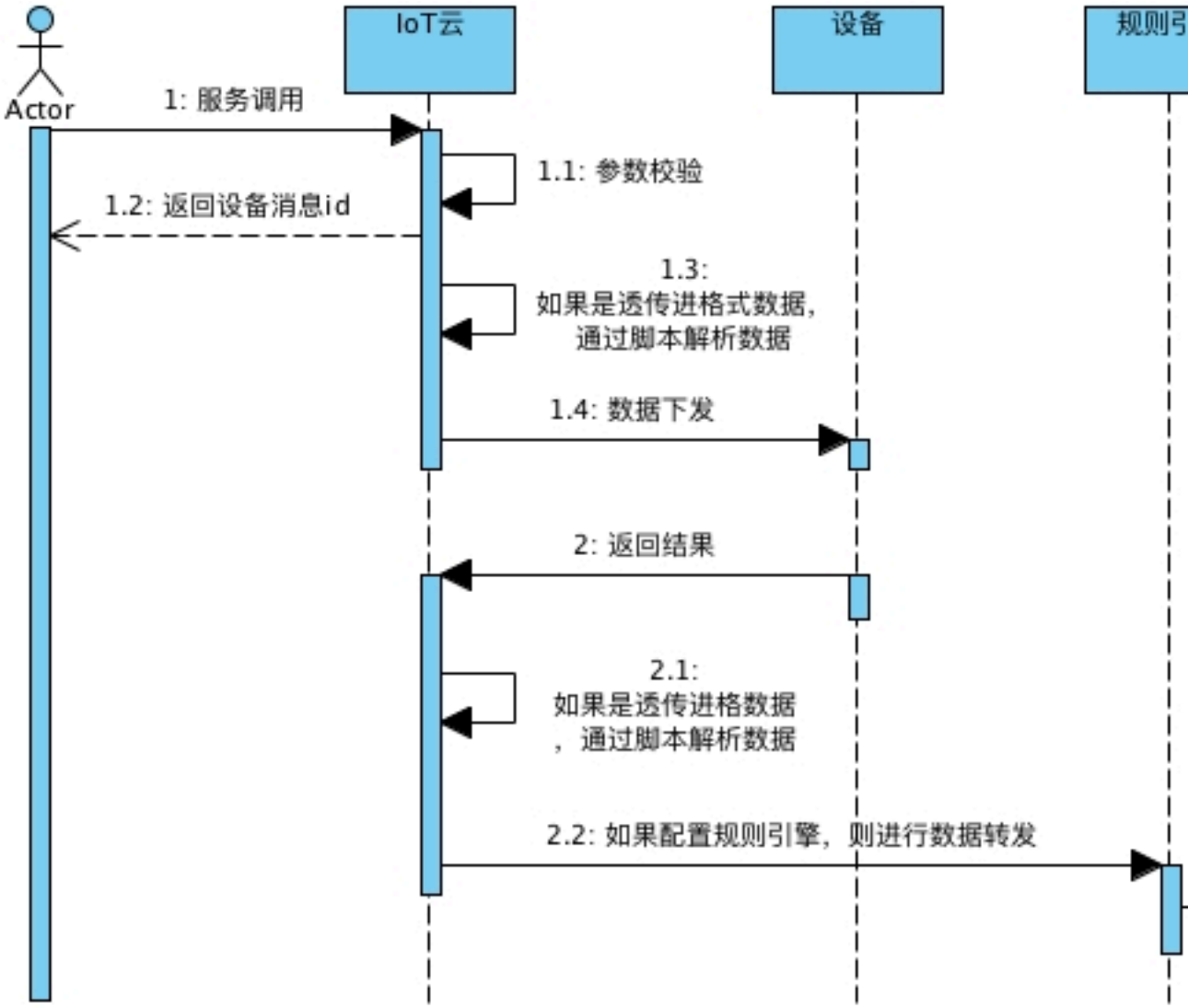
> **说明:**
> 如果配置了规则引擎，则通过规则引擎将数据输出到规则引擎配置的目的云产品中。
>
> - 在配置规则引擎数据流转，编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/property/post`，获取设备属性。

- 在配置规则引擎数据流转， 编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`，获取设备事件。

3. 物联网平台返回处理结果。
4. 您可以通过*QueryDevicePropertyData*接口查询设备上报的属性历史数据，通过*QueryDeviceEventData*接口查询设备上报的事件历史数据。

调用设备服务或设置属性

- 异步服务调用或属性设置



1. 设置属性或调用服务。

> 说明:
>
> - 设置属性：调用*SetDeviceProperty*接口为设备设置具体属性。

- 调用服务：调用*InvokeThingService*接口来异步调用服务（定义服务时，调用方式选择为异步的服务即为异步调用）。

2. 物联网平台对您提交的参数进行校验。
3. 物联网平台采用异步调用方式下发数据给设备，并返回调用操作结果。若没有报错，则结果中携带下发给设备的消息ID。

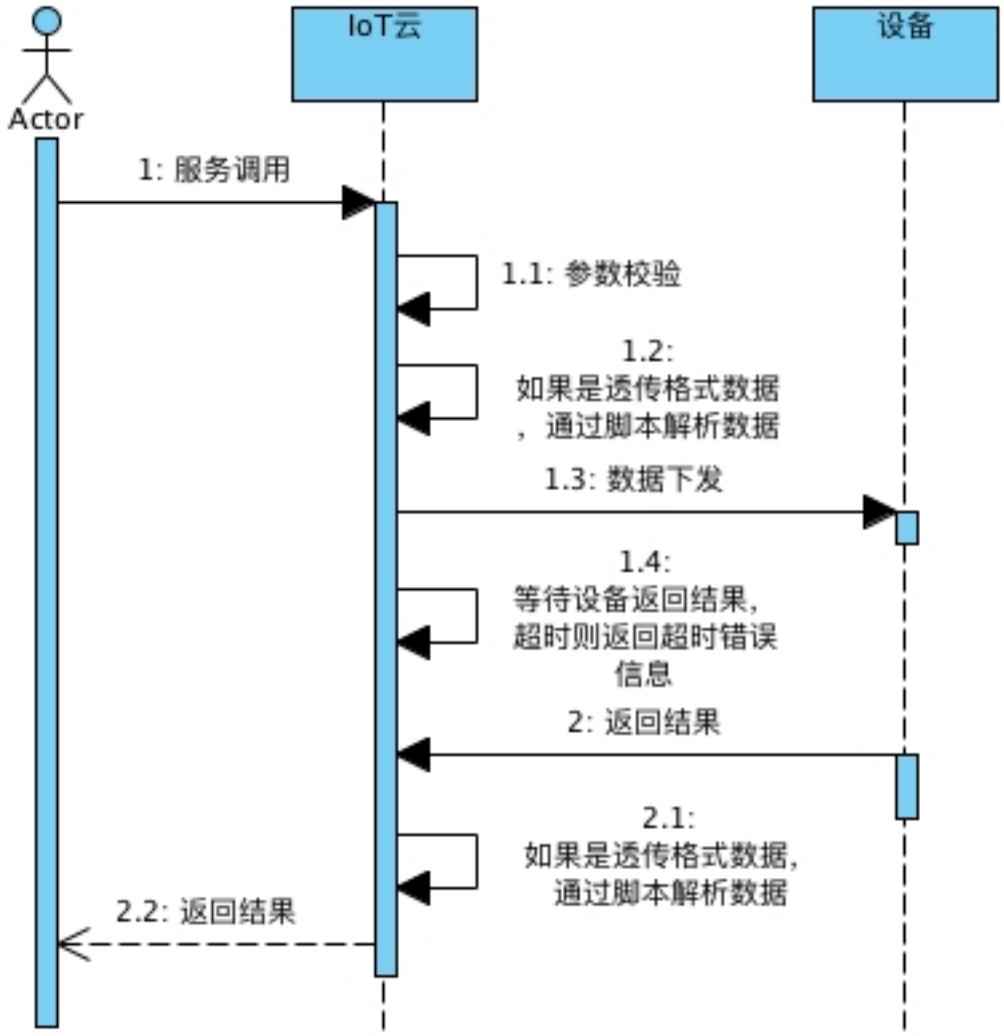
> 说明:
> 对于透传格式（透传/自定义）数据，则会调用数据解析脚本中的`protocolToRawData`方法，对数据进行数据格式转换后，再将转换后的数据下发给设备。

4. 设备收到数据后，进行业务处理。

> 说明:
> - 如果是透传格式（透传/自定义）数据，则使用透传格式数据的Topic。
> - 如果是非透传格式（Alink JSON）数据，则使用非透传格式数据的Topic。

5. 设备完成业务处理后，返回处理结果给物联网平台。
6. 物联网平台收到处理结果的后续操作：
   - 如果是透传格式（透传/自定义）数据，将调用数据解析脚本中的`rawDataToProtocol`方法，对设备返回的结果进行数据格式转换。
   - 如果配置了规则引擎数据流转，则将数据流转到规则引擎配置的目的Topic或云产品中。
     - 在配置规则引擎数据流转， 编写处理数据的SQL时，将Topic定义为：`/sys/{productKey}/{deviceName}/thing/downlink/reply/message`，获取异步调用的返回结果。
     - 对于透传格式（透传/自定义）数据，规则引擎获取到的数据是经过脚本解析之后的数据。

- 同步服务调用



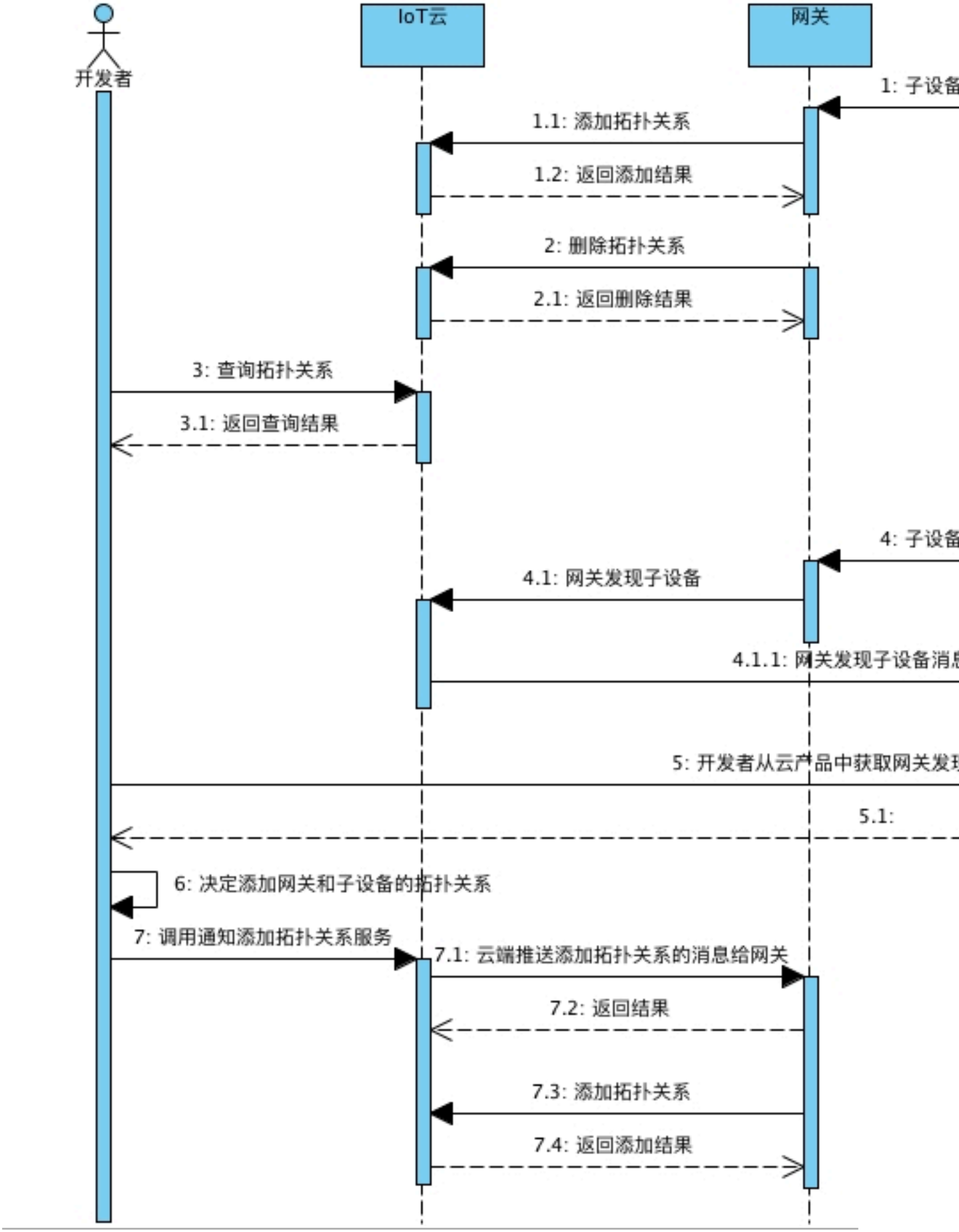
1. 通过调用*InvokeThingService*接口来调用同步服务（定义服务时，调用方式选择为同步的服务即为同步调用）。
2. 物联网平台对您提交的参数进行校验。
3. 使用同步调用方式，调用RRPC的Topic，下发数据给设备，物联网平台同步等待设备返回结果。

   说明:
   对于透传格式（透传/自定义）数据，则会先调用数据解析脚本中的`protocolToRawData`方法，对数据进行数据格式转换后，再将格式转换后的数据下发给设备。
4. 设备完成处理业务后，返回处理结果。若超时，则返回超时的错误信息。
5. 物联网平台收到设备处理结果后，返回结果给调用者。

   说明:

对于透传格式（透传/自定义）数据，物联网平台会调用脚本中的`rawDataToProtocol`方法，对设备返回的结果数据进行格式转换后，再将转换后的结果数据发送给调用者。

拓扑关系


开发者
IoT云
网关
1: 子设备
1.1: 添加拓扑关系
1.2: 返回添加结果
2: 删除拓扑关系
2.1: 返回删除结果
3: 查询拓扑关系
3.1: 返回查询结果
4: 子设备
4.1: 网关发现子设备
4.1.1: 网关发现子设备消息
5: 开发者从云产品中获取网关发现
5.1:
6: 决定添加网关和子设备的拓扑关系
7: 调用通知添加拓扑关系服务
7.1: 云端推送添加拓扑关系的消息给网关
7.2: 返回结果
7.3: 添加拓扑关系
7.4: 返回添加结果

1. 子设备连接到网关后，网关通过添加拓扑关系Topic，添加拓扑关系，物联网平台返回添加的结果。
2. 网关可以通过删除拓扑关系的Topic，来删除网关和子设备的拓扑关系。
3. 开发者可以调用*GetThingTopo*接口来查询网关和子设备的拓扑关系。
4. 当添加拓扑关系需要第三方介入时，可以通过下面的步骤添加拓扑关系。
   a. 网关通过发现设备列表的Topic，上报发现的子设备信息。
   b. 物联网平台收到上报数据后，如果配置了规则引擎，可以通过规则引擎将数据流转到对应的云产品中，开发者可以从云产品中获取该数据。
   c. 当开发者从云产品中获取了网关发现的子设备后，可以决定是否添加与网关的拓扑关系。如果需要添加拓扑关系，可以调用*NotifyAddThingTopo*接口通知网关发起添加拓扑关系。
   d. 物联网平台收到*NotifyAddThingTopo*接口调用后，会通知添加拓扑关系的Topic将命令推送给网关。
   e. 网关收到通知添加拓扑关系的命令后，通过添加拓扑关系Topic，添加拓扑关系。

 说明:

- 网关通过Topic：`/sys/{productKey}/{deviceName}/thing/topo/add`，添加拓扑关系。
- 网关通过Topic：`/sys/{productKey}/{deviceName}/thing/topo/delete`，删除拓扑关系。
- 网关通过Topic：`/sys/{productKey}/{deviceName}/thing/topo/get`，获取网关和子设备的拓扑关系。
- 网关通过发现设备列表的Topic：`/sys/{productKey}/{deviceName}/thing/list/found`，上报发现的子设备信息。
- 网关通过Topic：`/sys/{productKey}/{deviceName}/thing/topo/add/notify`，通知网关设备对子设备发起添加拓扑关系

6.2 设备身份注册

设备上线之前您需要对设备进行身份注册，标识您的设备。

接入物联网平台的设备身份注册有两种方式：

- 使用一机一密的方式。首先，在物联网平台注册设备，获取设备证书信息（ProductKey，DeviceName， DeviceSecret）做为设备唯一标识。然后，将设备证书信息预烧录到固件，固件在完成上线建连后即可向云端上报数据。

- 使用动态注册的方式，包括直连设备使用一型一密动态注册和子设备动态注册。
  - 直连设备使用一型一密动态注册的流程：
    1. 在物联网平台预注册设备，并获取产品证书（ProductKey和ProductSecret）。预注册设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
    2. 在控制台开启产品的动态注册开关。
    3. 将产品证书烧录至固件。
    4. 设备向云端发起身份认证。云端认证成功后，下发DeviceSecret。
    5. 设备使用设备证书与云端建立连接。
  - 子设备动态注册流程：
    1. 在物联网平台预注册设备，获取ProductKey。预注册设备时，可以使用设备的MAC地址或SN序列号等作为DeviceName。
    2. 在控制台开启动态注册开关。
    3. 将子设备ProductKey烧录至固件或网关上。
    4. 网关代替子设备向云端发起身份注册。

子设备的动态注册

上行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/sub/register`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/sub/register_reply`

请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.sub.register"
}
```

响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "iotId": "12344",
      "productKey": "1234556554",
      "deviceName": "deviceName1234",
```

```
      "deviceSecret": "xxxxxx"
    }
  ]
}
```

参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	List	设备动态注册的参数。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。
iotId	String	设备的唯一标识ID。
deviceSecret	String	设备秘钥。
method	String	请求方法，取值thing.sub.register。
code	Integer	结果信息。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6402	topo relation cannot add by self	设备不能将自己添加为自己的子设备。
401	request auth error	签名校验失败。

直连设备使用一型一密动态注册

直连设备动态注册，通过HTTP请求进行。 使用时需在控制台上开通该产品的一型一密动态注册功能。

- URL模板： `https://iot-auth.cn-shanghai.aliyuncs.com/auth/register/device`
- HTTP方法： POST

请求数据格式

```
POST /auth/register/device  HTTP/1.1
Host: iot-auth.cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
```

```
productKey=1234556554&deviceName=deviceName1234&random=567345&sign=
adfv123hdfdh&signMethod=HmacMD5
```

响应数据格式

```
{
  "code": 200,
  "data": {
    "productKey": "1234556554",
    "deviceName": "deviceName1234",
    "deviceSecret": "adsfweafdsf"
  },
  "message": "success"
}
```

参数说明

参数	类型	说明
Method	String	POST
Host	String	endpoint地址：iot-auth.cn-shanghai.aliyuncs.com。
Content-Type	String	设备发送给物联网平台的上行数据的编码格式。
productKey	String	产品唯一标识。
deviceName	String	设备名称。
random	String	随机数。
sign	String	签名。 加签方法： 1. 将所有提交给服务器的参数（sign,signMethod除外）按照字母顺序排序, 然后将参数和值依次拼接（无拼接符号）。 2. 对加签内容，通过signMethod指定的加签算法，使用产品的ProductSecret，进行加签。 示例如下： `sign = hmac_sha1(productSecret, deviceNamedeviceName1234productKey1234556554random123)`
signMethod	String	签名方法， 目前支持hmacmd5、hmacsha1、hmacsha256。
code	Integer	结果信息。
deviceSecret	String	设备秘钥。

6.3 添加拓扑关系

子设备身份注册后，需由网关向物联网平台上报网关与子设备的拓扑关系，然后进行子设备上线。

子设备上线过程中，物联网平台会校验子设备的身份和与网关的拓扑关系。所有校验通过，才会建立并绑定子设备逻辑通道至网关物理通道上。子设备与物联网平台的数据上下行通信与直连设备的通信协议一致，协议上不需要露出网关信息。

删除拓扑关系后，子设备不能再通过网关上线。系统将提示拓扑关系不存在，认证不通过等错误。

添加设备拓扑关系

数据上行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/topo/add`
- 响应Topic：`sys/{productKey}/{deviceName}/thing/topo/add_reply`

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554",
      "sign": "xxxxxx",
      "signmethod": "hmacSha1",
      "timestamp": "1524448722000",
      "clientId": "xxxxxx"
    }
  ],
  "method": "thing.topo.add"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	List	请求入参。

参数	类型	说明
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。
sign	String	签名。 加签算法： 1. 将所有提交给服务器的参数（sign，signMethod除外）按照字母顺序排序，然后将参数和值依次拼接（无拼接符号）。 2. 对加签内容，需通过signMethod指定的加签算法，使用设备的DeviceSecret值，进行签名计算。 签名计算示例： `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp1524448722000)`
signmethod	String	签名方法，支持hmacSha1，hmacSha256，hmacMd5，Sha256。
timestamp	String	时间戳。
clientId	String	设备本地标记，非必填，可以和productKey&deviceName保持一致。
method	String	请求方法，取值：thing.topo.add。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。。
6402	topo relation cannot add by self	设备不能把自己添加为自己的子设备。
401	request auth error	签名校验授权失败。

删除设备的拓扑关系

网关类型的设备， 可以通过该Topic上行请求删除它和子设备之间的拓扑关系。

数据上行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/topo/delete`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/topo/delete_reply`

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为`1.0`。
params	List	请求参数。
deviceName	String	子设备名称。
productKey	String	子设备产品Key。
method	String	请求方法。取值`thing.topo.delete`。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。

参数	类型	说明
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

获取设备的拓扑关系

数据上行

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

网关类型的设备，可以通过该Topic获取该设备和子设备的拓扑关系。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.topo.get"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数，可为空。

参数	类型	说明
method	String	请求方法，取值thing.topo.get

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。

发现设备列表上报

数据上行

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

在一些场景下，网关可以发现新接入的子设备。发现后，需将新接入子设备的信息上报云端，然后通过数据流转到第三方应用，选择将哪些子设备接入该网关。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.list.found"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
```

```
  "data":{}
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数，可为空。
method	String	请求方法，取值thing.list.found。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6250	product not found	上报的子设备产品不存在。
6280	devicename not meet specs	上报的子设备的名称不符规范，设备名称支持英文字母、数字和特殊字符-_@.:，长度限制4~32。

通知网关添加设备拓扑关系

数据下行

- **请求Topic：** /sys/{productKey}/{deviceName}/thing/topo/add/notify
- **响应Topic：** /sys/{productKey}/{deviceName}/thing/topo/add/notify_reply

通知网关设备对子设备发起添加拓扑关系，可以配合发现设备列表上报功能使用。可以通过数据流转获取设备返回的结果，数据流转Topic为/{productKey}/{deviceName}/thing/downlink/reply/message.

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.add.notify"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

请求参数说明

参数	类型	说明
id	String	数据下行消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数，可为空。
method	String	请求方法，取值thing.topo.add.notify。
deviceName	String	子设备的名称。
productKey	String	子设备的产品Key。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
data	Object	请求成功时的返回结果。

6.4 子设备上下线

子设备上线之前，需在物联网平台为子设备注册身份，建立子设备与网关的拓扑关系。子设备上线时，物联网平台会根据拓扑关系进行子设备身份校验，以确定子设备是否具备使用网关通道的能力。

说明:

子设备上下线消息，只支持QoS=0，不支持QoS=1。

子设备上线

说明:

一个网关下，同时在线的子设备数量不能超过1500。若在线子设备数量达到1500个后，新的子设备上线请求将被拒绝。

数据上行

- 请求Topic：`/ext/session/${productKey}/${deviceName}/combine/login`
- 响应Topic：`/ext/session/${productKey}/${deviceName}/combine/login_reply`

说明:

因为子设备通过网关通道与物联网平台通信，以上Topic为网关设备的Topic。Topic中变量*${productKey}*和*${deviceName}*需替换为网关设备的对应信息。

Alink请求数据格式

```
{
  "id": "123",
  "params": {
    "productKey": "123",
    "deviceName": "test",
    "clientId": "123",
    "timestamp": "123",
    "signMethod": "hmacmd5",
    "sign": "xxxxxx",
    "cleanSession": "true"
  }
}
```

说明:

消息体中，参数productKey和deviceName的值是子设备的对应信息。

Alink响应数据格式

```
{
  "id":"123",
```

```
  "code":200,
  "message":"success"
  "data":""
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
params	Object	请求入参。
deviceName	String	子设备的设备名称。
productKey	String	子设备所属的产品Key。
sign	String	子设备签名。签名方法与直连设备签名方法相同。 签名方法： 1. 将所有提交给服务器的参数（sign，signMethod 和 cleanSession除外）按照字母顺序排序，然后将参数和值依次拼接（无拼接符号）。 2. 对加签内容，通过signMethod指定的加签算法，并使用设备的DeviceSecret值，进行签名计算。 示例如下： `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp123)`
signMethod	String	签名方法，支持hmacSha1，hmacSha256，hmacMd5，Sha256。
timestamp	String	时间戳。
clientId	String	设备端标识。可以为设备的productKey&deviceName。
cleanSession	String	· 如果取值是true，则清理所有子设备离线时的消息，即所有未接收的QoS1消息将被清除。 · 如果取值是false，则不清理子设备离线时的消息。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
message	String	结果信息。

参数	类型	说明
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
429	rate limit, too many subDeviceOnline msg in one minute	单个设备认证过于频繁被限流。
428	too many subdevices under gateway	网关下同时在线子设备过多。
6401	topo relation not exist	网关和子设备没有拓扑关系。
6100	device not found	子设备不存在。
521	device deleted	子设备已被删除。
522	device forbidden	子设备已被禁用。
6287	invalid sign	子设备密码或者签名错误。

子设备下线

数据上行

- 请求Topic：`/ext/session/{productKey}/{deviceName}/combine/logout`
- 响应Topic：`/ext/session/{productKey}/{deviceName}/combine/logout_reply`

说明：

因为子设备通过网关通道与物联网平台通信，以上Topic为网关设备的Topic。Topic中变量*${productKey}*和*${deviceName}*需替换为网关设备的对应信息。

Alink请求数据格式

```
{
  "id": 123,
  "params": {
    "productKey": "xxxxx",
    "deviceName": "xxxxx"
  }
}
```

说明：

消息体中，参数productKey和deviceName的值是子设备的对应信息。

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "message": "success",
  "data": ""
}
```

请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
params	Object	请求入参。
deviceName	String	子设备的设备名称。
productKey	String	子设备所属的产品Key。

响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。
code	Integer	返回结果，200代表成功。
message	String	结果信息。
data	Object	请求成功时的返回结果。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
520	device no session	子设备会话不存在。

子设备接入具体可参考设备身份注册，错误码参考错误码说明。

6.5 设备属性、事件、服务

如果产品定义了物模型，设备可以按照属性、事件、服务协议分别上报数据。物模型（属性、事件、服务）数据格式请参考物模型数据格式文档。本文仅讲解如何上报数据。

设备的数据上报方式有两种：ICA标准数据格式 (Alink JSON)和透传/自定义。两者二选一，推荐您使用Alink JSON方式。

- ICA 标准数据格式 (Alink JSON)：设备按照物联网平台定义的标准数据格式生成数据，然后上报数据。具体格式，请参见本文示例。
- 透传/自定义：设备上报原始数据如二进制数据流，阿里云物联网平台会运行您在控制台提交的数据解析脚本，将原始数据转成标准数据格式后，再进行业务处理。而云端返回的是标准Alink JSON格式，返回结果经数据解析后，再推送给设备。

设备上报属性

上行（透传）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

请求数据为设备上报的原始报文，示例如下。

```
0x020000007b00
```

云端返回数据格式如下：

```
{
    "id":"123",
    "code":200,
    "method":"thing.event.property.post"
    "data":{}
}
```

上行（Alink JSON）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/event/property/post`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/event/property/post_reply`

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": {
      "value": "on",
      "time": 1524448722000
```

```
    },
    "WF": {
      "value": 23.6,
      "time": 1524448722000
    }
  },
  "method": "thing.event.property.post"
}
```

表 6-1: 请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数。如以上示例中，设备上报了的两个属性Power和WF。具体属性信息，包含属性上报时间（time）和上报的属性值（value）。
time	Long	属性上报时间。
value	object	上报的属性值。
method	String	请求方法，取值thing.event.property.post。

Alink响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 6-2: 响应参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字。
code	Integer	结果状态码。 具体参考设备端通用code。
data	String	请求成功时，返回的数据。

表 6-3: 错误码

错误码	消息	描述
460	request parameter error	请求参数错误。

错误码	消息	描述
6106	map size must less than 200	一次最多只能上报200条属性。
6313	tsl service not available	物模型校验服务不可用。 物联网平台根据您定义的TSL中属性格式，校验上报的属性信息。 校验后，将过滤掉不合格的属性，仅保留合格属性。即使全部属性都被过滤，也代表着校验成功。 若校验服务不可用，报6313错误码。

设备上报至平台的属性信息，可以使用规则引擎数据流转功能，转发至其他云产品。具体Topic和数据格式请参考设备属性上报。

设置设备属性

下行（透传）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/model/down_raw`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/model/down_raw_reply`

下行（Alink JSON）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/service/property/set`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/service/property/set_reply`

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "temperature": "30.5"
  },
  "method": "thing.service.property.set"
}
```

表 6-4: 请求参数说明

参数	类型	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。

参数	类型	说明
params	Object	属性设置参数。如以上示例中，设置属性：`{ "temperature": "30.5" }`。
method	String	请求方法，取值`thing.service.property.set`。

Alink响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 6-5: 响应参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字。
code	Integer	结果状态码， 具体参考设备端通用*code*。
data	String	请求成功时，返回的数据。

您可以使用规则引擎数据流转功能，将属性设置返回的结果，转发至其它云产品。具体Topic和数据格式请参考设备下行指令结果数据流转。

设备事件上报

上行（透传）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

请求数据为设备上报的原始报文，示例如下。

```
0xff0000007b00
```

云端返回数据格式如下：

```
{
    "id":"123",
    "code":200,
    "method":"thing.event.{tsl.event.identifier}.post"
    "data":{}
}
```

上行（Alink JSON）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post_reply`

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "Power": "on",
      "WF": "2"
    },
    "time": 1524448722000
  },
  "method": "thing.event.{tsl.event.identifier}.post"
}
```

表 6-6: 请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	List	上报事件的参数。
value	Object	具体的事件信息。如以上示例中 `{ "Power": "on", "WF": "2" }`
time	Long	事件生成的时间戳，类型为UTC毫秒级时间。
method	String	请求方法：`thing.event.{tsl.event.identifier}.post`。 说明：`{tsl.event.identifier}`为物模型中，事件的标识符（Identifier）。请参见物模型文档。

Alink响应数据格式：

```
{
```

```
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 6-7: 响应参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字。
code	Integer	结果状态码，具体参考设备端通用code。 说明: 物联网平台会对设备上报的事件做校验。通过产品的TSL描述判断上报的事件是否符合定义的事件格式。不合格的事件会直接被过滤掉，并返回失败的错误码。
data	String	请求成功时，返回的数据。

Alink格式示例

假设产品中定义了一个alarm事件，它的TSL描述如下：

```
{
  "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json
",
  "link": "/sys/${productKey}/airCondition/thing/",
  "profile": {
    "productKey": "${productKey}，请替换为您的ProductKey",
    "deviceName": "airCondition，请替换为您的Devicename"
  },
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",
      "desc": "风扇警报",
      "type": "alert",
      "required": true,
      "outputData": [
        {
          "identifier": "errorCode",
          "name": "错误码",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ],
      "method": "thing.event.alarm.post"
    }
  ]
```

```
}
```

当设备上报事件时，Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "errorCode": "error"
    },
    "time": 1524448722000
  },
  "method": "thing.event.alarm.post"
}
```

设备上报至平台的事件信息，可以使用规则引擎数据流转功能，转发至其他云产品。具体Topic和数据格式，请参考设备上报事件。

设备服务调用

下行（透传）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/model/down_raw`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/model/down_raw_reply`

下行（Alink JSON）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}_reply`

服务调用方式

支持同步调用和异步调用。物模型定义服务时，需设置此项。

- 同步方式：物联网平台直接使用RRPC同步方式下行推送请求。设备RRPC的集成方式，请参见什么是*RRPC*。
- 异步方式：物联网平台则采用异步方式下行推送请求，设备也采用异步方式返回结果。

  只有当前服务选择为异步调用方式，物联网平台才会订阅该异步响应Topic。异步调用的结果，可以使用规则引擎数据流转功能获取。对应Topic和数据格式，请参考设备下行指令结果。

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
```

```
    "Power": "on",
    "WF": "2"
  },
  "method": "thing.service.{tsl.service.identifier}"
}
```

表 6-8: 请求参数说明

参数	类型	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
params	Map	服务调用参数。包含服务标识符和服务的值。如以上示例中： `{ "Power": "on", "WF": "2" }`
method	String	请求方法：thing.service.{tsl.service.identifier}。 说明：{tsl.service.identifier}为物模型中定义的服务标识符（Identifier）。请参见新增物模型。

Alink响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 6-9: 返回参数说明

参数	类型	说明
id	String	消息ID号，String类型的数字。
code	Integer	结果状态码，具体参考设备端通用code。
data	String	返回的结果信息。 data参数的值和物模型定义相关。如果服务没有返回结果，则data的值为空。如果服务有返回结果，则返回的数据会严格遵循服务的定义。

Alink格式示例

比如产品中定义了服务SetWeight，它的TSL描述如下：

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.
json",
  "profile": {
    "productKey": "testProduct01"
  },
  "services": [
    {
      "outputData": [
        {
          "identifier": "OldWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "max": "200",
              "step": "1"
            },
            "type": "double"
          },
          "name": "OldWeight"
        },
        {
          "identifier": "CollectTime",
          "dataType": {
            "specs": {
              "length": "2048"
            },
            "type": "text"
          },
          "name": "CollectTime"
        }
      ],
      "identifier": "SetWeight",
      "inputData": [
        {
          "identifier": "NewWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "max": "200",
              "step": "1"
            },
            "type": "double"
          },
          "name": "NewWeight"
        }
      ],
      "method": "thing.service.SetWeight",
      "name": "设置重量",
      "required": false,
      "callType": "async"
    }
  ]
```

```
}
```

当调用服务时，Alink请求数据格式：

```
{
  "method": "thing.service.SetWeight",
  "id": "105917531",
  "params": {
    "NewWeight": 100.8
  },
  "version": "1.0.0"
}
```

Alink响应数据格式：

```
{
  "id": "105917531",
  "code": 200,
  "data": {
    "CollectTime": "1536228947682",
    "OldWeight": 100.101
  }
}
```

网关批量上报数据

网关类型的设备可以批量上报属性和事件，也可以代其子设备批量上报属性和事件。

 说明：

- 一次最多可上报200个属性，20个事件。
- 子设备数据条数限制为20。

上行（透传）

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

请求数据为设备上报的原始报文，示例如下。

```
0xff0000007b00
```

云端返回数据格式如下：

```
{
  "id": "123",
  "code": 200,
  "method": "thing.event.property.pack.post",
  "data": {}
}
```

上行（Alink JSON）

- **请求Topic：** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post`
- **响应Topic：** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post_reply`

Alink 请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "properties": {
      "Power": {
        "value": "on",
        "time": 1524448722000
      },
      "WF": {
        "value": { },
        "time": 1524448722000
      }
    },
    "events": {
      "alarmEvent1": {
        "value": {
          "param1": "on",
          "param2": "2"
        },
        "time": 1524448722000
      },
      "alertEvent2": {
        "value": {
          "param1": "on",
          "param2": "2"
        },
        "time": 1524448722000
      }
    },
    "subDevices": [
      {
        "identity": {
          "productKey": "",
          "deviceName": ""
        },
        "properties": {
          "Power": {
            "value": "on",
            "time": 1524448722000
          },
          "WF": {
            "value": { },
            "time": 1524448722000
          }
        },
        "events": {
          "alarmEvent1": {
            "value": {
              "param1": "on",
              "param2": "2"
            },
            "time": 1524448722000
```

```
          },
          "alertEvent2": {
            "value": {
              "param1": "on",
              "param2": "2"
            },
            "time": 1524448722000
          }
        }
      }
    ]
  },
  "method": "thing.event.property.pack.post"
}
```

表 6-10: 请求参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数。
properties	Object	属性，包含属性标识符、属性值(value) 和属性生成的时间(time)。
events	Object	事件，包含事件标识符、事件值(value) 和事件生成的时间(time)
subDevices	Object	子设备信息。
productKey	String	子设备产品key。
deviceName	String	子设备名称。
method	String	请求方法，取值thing.event.property.pack.post。

Alink 响应数据格式：

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

表 6-11: 响应参数说明

参数	类型	说明
id	String	消息ID，String类型的数字。

参数	类型	说明
code	Integer	返回结果，200代表成功。  说明: 系统会校验设备、拓扑关系、及上报的属性和事件都否符合产品物模型（TSL）中的定义。如果其中任何一项校验不通过，则上报数据失败。
data	Object	请求成功时的返回结果。

6.6 设备期望属性值

在云端设置设备期望属性值后，若设备在线，将实时更新设备属性状态；若设备离线，期望属性值将缓存云端，待设备上线后，获取期望属性值，并更新属性状态。本文讲述设备期望属性值的数据格式。

获取期望属性值

上行（Alink JSON）

设备向云端请求获取设备属性的期望值。

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/property/desired/get`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/property/desired/get_reply`

Alink请求数据格式

```
{
    "id" : "123",
    "version":"1.0",
    "params" : [
        "power",
        "temperature"
    ],
    "method":"thing.property.desired.get"
}
```

Alink响应数据格式

```
{
    "id":"123",
    "code":200,
    "data":{
  "power": {
   "value": "on",
   "version": 2
  }
 }
```

```
}
```

表 6-12: 请求参数描述

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	List	要获取期望值的属性标识符（Identifier）列表。 如示例中列举了两个属性的标识符： `[ "power", "temperature" ]`
method	String	请求方法，取值thing.property.desired.get。

表 6-13: 返回参数描述

参数	类型	说明
id	String	消息ID号。
code	Integer	结果信息， 具体参考设备端通用code。
data	Object	返回的期望值信息。 示例中，返回了属性power的期望值数据，包含期望值value和当前期望值版本version。 `{ "power": { "value": "on", "version": 2 } }` 说明: 若未在云端设置过该属性的期望值，或期望属性值已被清空，返回对象中不包含该属性的标识符。如示例中，属性temperature无期望值，返回数据中不包含该属性标识符。 data中所包含的参数具体说明，请见下表data。

表 6-14: data

参数	类型	说明
key	String	key即属性的标识符。如示例中为power。
value	Object	期望属性值。
version	Integer	当前期望属性值的版本。  说明: 首次设置期望属性值后，期望值版本为1。以后每次设置后，期望值版本号自动加1。

清空期望属性值

上行（Alink JSON）

设备清除云端设备的期望属性值。

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/property/desired/delete`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/property/desired/delete_reply`

Alink请求数据格式

```
{
    "id" : "123",
    "version":"1.0",
    "params" :  {
  "power": {
   "version": 1
  },
  "temperature": {
  }
 },
    "method":"thing.property.desired.delete"
}
```

Alink响应数据格式

```
{
    "id":"123",
    "code":200,
    "data":{
 }
```

```
}
```

表 6-15: 请求参数描述

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	List	要清除期望值的属性信息列表。传入数据包含属性的标识符和期望值版本version。如： `{ "power": { "version": 1 }, "temperature": { } }` params所包含的参数具体说明，请见下表params。
method	String	请求方法，取值thing.property.desired.delete。

表 6-16: params

参数	类型	说明
key	String	key即属性的标识符。如示例中，列出了power和temperature两个属性标识符。
version	Integer	要删除期望属性值的版本号。 说明： · version版本号可从Topic/sys/{productKey}/{deviceName}/thing/property/desired/get 获取。 · 如果指定version为2，则表示云端最新版本是2时执行清除。如果指定版本为2，但是云端最新版本是3，则忽略这个清除请求。 · 若请求中，未指定要清除的期望值版本version，则不验证版本号，该属性的期望值将被清除。

表 6-17: 返回参数描述

参数	类型	说明
id	String	消息ID号。

参数	类型	说明
code	Integer	结果信息， 具体参考设备端通用code。
data	String	返回数据。

6.7 子设备配置下发网关

云端将网关下所有子设备的连接配置和各子设备所属产品的物模型下发给网关。

配置下发

请求Topic：`/sys/{productKey}/{deviceName}/thing/model/config/push`

Alink配置推送数据格式：

```
{
  "id": 123,
  "version": "1.0",
  "method": "thing.model.config.push",
  "data": {
    "digest":"",
    "digestMethod":"",
    "url": ""
  }
}
```

参数说明：

参数	类型	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
method	String	请求方法，取值thing.model.config.push。
data	Object	数据。
digest	String	签名，用于校验从url中获取的数据完整性。
digestMethod	String	签名方法，默认sha256。
url	String	存储子设备配置数据的对象存储URL，用于从对象存储OSS中获取子设备配置数据。

子设备配置数据格式

对象存储URL中的子设备配置数据示例：

```
{
  "modelList": [
    {
```

```
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
          "outputData": "",
          "identifier": "AngleSelfAdaption",
          "inputData": [
            {
              "identifier": "test01",
              "index": 0
            }
          ],
          "displayName": "test01"
        }
      ],
      "properties": [
        {
          "identifier": "identifier",
          "displayName": "test02"
        },
        {
          "identifier": "identifier_01",
          "displayName": "identifier_01"
        }
      ],
      "events": [
        {
          "outputData": [
            {
              "identifier": "test01",
              "index": 0
            }
          ],
          "identifier": "event1",
          "displayName": "abc"
        }
      ]
    },
    {
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "properties": [
        {
          "originalDataType": {
            "specs": {
              "registerCount": 1,
              "reverseRegister": 0,
              "swap16": 0
            },
            "type": "bool"
          },
          "identifier": "test01",
          "registerAddress": "0x03",
          "scaling": 1,
          "operateType": "inputStatus",
          "pollingTime": 1000,
          "trigger": 1
        },
        {
          "originalDataType": {
            "specs": {
              "registerCount": 1,
```

```
              "reverseRegister": 0,
              "swap16": 0
            },
            "type": "bool"
          },
          "identifier": "test02",
          "registerAddress": "0x05",
          "scaling": 1,
          "operateType": "coilStatus",
          "pollingTime": 1000,
          "trigger": 2
        }
      ]
    },
    {
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "properties": [
        {
          "identifier": "test_02",
          "customize": {
            "test_02": 123
          }
        },
        {
          "identifier": "test_01",
          "customize": {
            "test01": 1
          }
        }
      ]
    }
  ],
  "serverList": [
    {
      "baudRate": 1200,
      "protocol": "RTU",
      "byteSize": 8,
      "stopBits": 2,
      "parity": 1,
      "name": "modbus01",
      "serialPort": "0",
      "serverId": "D73251B427****"
    },
    {
      "protocol": "TCP",
      "port": 8000,
      "ip": "192.168.0.1",
      "name": "modbus02",
      "serverId": "586CB066D****"
    },
    {
      "password": "XIJTginONohPEUAyZ******==",
      "secPolicy": "Basic128Rsa15",
      "name": "server_01",
      "secMode": "Sign",
      "userName": "123",
      "serverId": "55A9D276A7E****",
      "url": "tcp:00",
      "timeout": 10
    },
    {
      "password": "hAaX5s13gwX2JwyvUk******==",
```

```
      "name": "service_09",
      "secMode": "None",
      "userName": "1234",
      "serverId": "44895C63E3F****",
      "url": "tcp:00",
      "timeout": 10
    }
  ],
  "deviceList": [
    {
      "deviceConfig": {
        "displayNamePath": "123",
        "serverId": "44895C63E3FF4013924CEF31519A****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test_02"
    },
    {
      "deviceConfig": {
        "displayNamePath": "1",
        "serverId": "55A9D276A7****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test_03"
    },
    {
      "deviceConfig": {
        "slaveId": 1,
        "serverId": "D73251B4277****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test01"
    },
    {
      "deviceConfig": {
        "slaveId": 2,
        "serverId": "586CB066D6****"
      },
      "productKey": "a1ZlGQv****",
      "deviceName": "test02"
    }
  ],
  "tslList": [
    {
      "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/
schema.json",
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
          "outputData": [],
          "identifier": "set",
          "inputData": [
            {
              "identifier": "test02",
              "dataType": {
                "specs": {
                  "unit": "mm",
                  "min": "0",
                  "max": "1"
                },
                "type": "int"
              },
```

```
            "name": "测试功能02"
          }
        ],
        "method": "thing.service.property.set",
        "name": "set",
        "required": true,
        "callType": "async",
        "desc": "属性设置"
      },
      {
        "outputData": [
          {
            "identifier": "test01",
            "dataType": {
              "specs": {
                "unit": "m",
                "min": "0",
                "max": "1"
              },
              "type": "int"
            },
            "name": "测试功能01"
          },
          {
            "identifier": "test02",
            "dataType": {
              "specs": {
                "unit": "mm",
                "min": "0",
                "max": "1"
              },
              "type": "int"
            },
            "name": "测试功能02"
          }
        ],
        "identifier": "get",
        "inputData": [
          "test01",
          "test02"
        ],
        "method": "thing.service.property.get",
        "name": "get",
        "required": true,
        "callType": "async",
        "desc": "属性获取"
      }
    ],
    "properties": [
      {
        "identifier": "test01",
        "dataType": {
          "specs": {
            "unit": "m",
            "min": "0",
            "max": "1"
          },
          "type": "int"
        },
        "name": "测试功能01",
        "accessMode": "r",
        "required": false
      },
      {
```

```
          "identifier": "test02",
          "dataType": {
            "specs": {
              "unit": "mm",
              "min": "0",
              "max": "1"
            },
            "type": "int"
          },
          "name": "测试功能02",
          "accessMode": "rw",
          "required": false
        }
      ],
      "events": [
        {
          "outputData": [
            {
              "identifier": "test01",
              "dataType": {
                "specs": {
                  "unit": "m",
                  "min": "0",
                  "max": "1"
                },
                "type": "int"
              },
              "name": "测试功能01"
            },
            {
              "identifier": "test02",
              "dataType": {
                "specs": {
                  "unit": "mm",
                  "min": "0",
                  "max": "1"
                },
                "type": "int"
              },
              "name": "测试功能02"
            }
          ],
          "identifier": "post",
          "method": "thing.event.property.post",
          "name": "post",
          "type": "info",
          "required": true,
          "desc": "属性上报"
        }
      ]
    },
    {
      "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/
schema.json",
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
          "outputData": [],
          "identifier": "set",
          "inputData": [
            {
              "identifier": "identifier",
```

```
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
            },
            "name": "7614"
          },
          {
            "identifier": "identifier_01",
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
            },
            "name": "测试功能1"
          }
        ],
        "method": "thing.service.property.set",
        "name": "set",
        "required": true,
        "callType": "async",
        "desc": "属性设置"
      },
      {
        "outputData": [
          {
            "identifier": "identifier",
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
            },
            "name": "7614"
          },
          {
            "identifier": "identifier_01",
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
            },
            "name": "测试功能1"
          }
        ],
        "identifier": "get",
        "inputData": [
          "identifier",
          "identifier_01"
        ],
        "method": "thing.service.property.get",
        "name": "get",
        "required": true,
        "callType": "async",
        "desc": "属性获取"
      },
      {
        "outputData": [],
        "identifier": "AngleSelfAdaption",
        "inputData": [
          {
```

```
            "identifier": "test01",
            "dataType": {
              "specs": {
                "min": "1",
                "max": "10",
                "step": "1"
              },
              "type": "int"
            },
            "name": "参数1"
          }
        ],
        "method": "thing.service.AngleSelfAdaption",
        "name": "角度自适应校准",
        "required": false,
        "callType": "async"
      }
    ],
    "properties": [
      {
        "identifier": "identifier",
        "dataType": {
          "specs": {
            "length": "2048"
          },
          "type": "text"
        },
        "name": "7614",
        "accessMode": "rw",
        "required": true
      },
      {
        "identifier": "identifier_01",
        "dataType": {
          "specs": {
            "length": "2048"
          },
          "type": "text"
        },
        "name": "测试功能1",
        "accessMode": "rw",
        "required": false
      }
    ],
    "events": [
      {
        "outputData": [
          {
            "identifier": "identifier",
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
            },
            "name": "7614"
          },
          {
            "identifier": "identifier_01",
            "dataType": {
              "specs": {
                "length": "2048"
              },
              "type": "text"
```

```
              },
              "name": "测试功能1"
            }
          ],
          "identifier": "post",
          "method": "thing.event.property.post",
          "name": "post",
          "type": "info",
          "required": true,
          "desc": "属性上报"
        },
        {
          "outputData": [
            {
              "identifier": "test01",
              "dataType": {
                "specs": {
                  "min": "1",
                  "max": "20",
                  "step": "1"
                },
                "type": "int"
              },
              "name": "测试参数1"
            }
          ],
          "identifier": "event1",
          "method": "thing.event.event1.post",
          "name": "event1",
          "type": "info",
          "required": false
        }
      ]
    },
    {
      "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/
schema.json",
      "profile": {
        "productKey": "a1ZlGQv****"
      },
      "services": [
        {
          "outputData": [],
          "identifier": "set",
          "inputData": [
            {
              "identifier": "test_01",
              "dataType": {
                "specs": {
                  "min": "1",
                  "max": "100",
                  "step": "1"
                },
                "type": "int"
              },
              "name": "参数1"
            },
            {
              "identifier": "test_02",
              "dataType": {
                "specs": {
                  "min": "1",
                  "max": "100",
                  "step": "10"
```

```
            },
            "type": "double"
          },
          "name": "参数2"
        }
      ],
      "method": "thing.service.property.set",
      "name": "set",
      "required": true,
      "callType": "async",
      "desc": "属性设置"
    },
    {
      "outputData": [
        {
          "identifier": "test_01",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "1"
            },
            "type": "int"
          },
          "name": "参数1"
        },
        {
          "identifier": "test_02",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "10"
            },
            "type": "double"
          },
          "name": "参数2"
        }
      ],
      "identifier": "get",
      "inputData": [
        "test_01",
        "test_02"
      ],
      "method": "thing.service.property.get",
      "name": "get",
      "required": true,
      "callType": "async",
      "desc": "属性获取"
    }
  ],
  "properties": [
    {
      "identifier": "test_01",
      "dataType": {
        "specs": {
          "min": "1",
          "max": "100",
          "step": "1"
        },
        "type": "int"
      },
      "name": "参数1",
      "accessMode": "rw",
```

```
          "required": false
        },
        {
          "identifier": "test_02",
          "dataType": {
            "specs": {
              "min": "1",
              "max": "100",
              "step": "10"
            },
            "type": "double"
          },
          "name": "参数2",
          "accessMode": "rw",
          "required": false
        }
      ],
      "events": [
        {
          "outputData": [
            {
              "identifier": "test_01",
              "dataType": {
                "specs": {
                  "min": "1",
                  "max": "100",
                  "step": "1"
                },
                "type": "int"
              },
              "name": "参数1"
            },
            {
              "identifier": "test_02",
              "dataType": {
                "specs": {
                  "min": "1",
                  "max": "100",
                  "step": "10"
                },
                "type": "double"
              },
              "name": "参数2"
            }
          ],
          "identifier": "post",
          "method": "thing.event.property.post",
          "name": "post",
          "type": "info",
          "required": true,
          "desc": "属性上报"
        }
      ]
    }
  ]
}
```

参数	类型	描述
modelList	Object	网关下面所有子设备的产品扩展信息。详见modelList说明。

参数	类型	描述
serverList	Object	网关下面所有的子设备连接通道。详见serverList说明。
deviceList	Object	网关下面所有子设备的连接配置。详见deviceList说明。
tslList	Object	网关下面所有子设备的TSL描述。请参见物模型格式。

modelList说明

设备协议目前支持Modbus、OPC UA和自定义协议，不同协议的扩展信息不一致。

- Modbus协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "properties": [
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      },
      "identifier": "test01",
      "registerAddress": "0x03",
      "scaling": 1,
      "operateType": "inputStatus",
      "pollingTime": 1000,
      "trigger": 1
    },
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      },
      "identifier": "test02",
      "registerAddress": "0x05",
      "scaling": 1,
      "operateType": "coilStatus",
      "pollingTime": 1000,
      "trigger": 2
    }
  ]
```

```
}
```

参数说明如下：

参数	类型	说明
identifier	String	属性、事件和服务的标识符。
operateType	String	参数类型，取值可以为： - 线圈状态：coilStatus - 输入状态：inputStatus - 保持寄存器：holdingRegister - 输入寄存器：inputRegister
registerAddress	String	寄存器地址。
originalDataType	Object	数据原始类型。
type	String	取值如下： int16，uint16，int32，uint32，int64，uint64，float，double，string，customized data
specs	Object	描述信息。
registerCount	Integer	寄存器的数据个数。
swap16	Integer	把寄存器内16位数据的前后8个bits互换。0表示false、1表示true。
reverseRegister	Integer	把原始数据32位数据的bits互换。 - 0：不互换（false） - 1：互换（true）
scaling	Integer	缩放因子。
pollingTime	Integer	采集间隔。
trigger	Integer	数据的上报方式： - 1：按时上报 - 2：变更上报

- OPC UA协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "services": [
    {
      "outputData": "",
      "identifier": "AngleSelfAdaption",
      "inputData": [
```

```
        {
          "identifier": "test01",
          "index": 0
        }
      ],
      "displayName": "test01"
    }
  ],
  "properties": [
    {
      "identifier": "identifier",
      "displayName": "test02"
    },
    {
      "identifier": "identifier_01",
      "displayName": "identifier_01"
    }
  ],
  "events": [
    {
      "outputData": [
        {
          "identifier": "test01",
          "index": 0
        }
      ],
      "identifier": "event1",
      "displayName": "abc"
    }
  ]
}
```

参数说明如下：

参数	类型	说明
services	Object	服务。
properties	Object	属性。
events	Object	事件。
outputData	Object	输出参数，如事件上报数据、服务调用的返回结果。
identifier	String	属性、事件或服务的标识符。
inputData	Object	输入参数，如服务的入参。
index	Integer	索引信息。
displayName	String	属性、事件或服务的名称。

- 自定义协议下子设备的产品扩展信息数据示例：

```
{
  "profile": {
    "productKey": "a1ZlGQv****"
  },
  "properties": [
    {
```

```
      "identifier": "test_02",
      "customize": {
        "test_02": 123
      }
    },
    {
      "identifier": "test_01",
      "customize": {
        "test01": 1
      }
    }
  ]
}
```

参数说明如下：

参数	类型	说明
productKey	String	产品Key，物联网平台为产品颁发的唯一标识符。
properties	Object	属性信息集合。
identifier	String	属性标识符。
customize	Object	属性的自定义扩展信息，即定义属性时，扩展描述中填入的数据。

serverList说明

子设备连接通道的信息分为Modbus和OPC UA协议两种。

- Modbus协议下的连接通道数据示例：

```
[
  {
    "baudRate": 1200,
    "protocol": "RTU",
    "byteSize": 8,
    "stopBits": 2,
    "parity": 1,
    "name": "modbus01",
    "serialPort": "0",
    "serverId": "D73251B427****"
  },
  {
    "protocol": "TCP",
    "port": 8000,
    "ip": "192.168.0.1",
    "name": "modbus02",
    "serverId": "586CB066D****"
  }
]
```

参数	类型	说明
protocol	String	协议类型，TCP或者RTU。
port	Integer	端口号。

参数	类型	说明
ip	String	IP地址。
name	String	子设备通道名称。
serverId	String	子设备通道ID。
baudRate	Integer	波特率。
byteSize	Integer	字节数。
stopBits	Integer	停止位。
parity	Integer	奇偶校验位。 - E：偶校验 - O：奇校验 - N：无校验
serialPort	String	串口号。

- OPC UA协议下的连接通道数据示例：

```
{
  "password": "XIJTginONohPEUAyZx******==",
  "secPolicy": "Basic128Rsa15",
  "name": "server_01",
  "secMode": "Sign",
  "userName": "123",
  "serverId": "55A9D276A7E****",
  "url": "tcp:00",
  "timeout": 10
}
```

参数说明如下：

参数	类型	说明
password	String	密码，采用AES算法加密。具体说明见下文OPC UA的password加密方式。
secPolicy	String	加密策略，取值：None、Basic128Rsa15、Basic256。
secMode	String	加密模式，取值：None、Sign、SignAndEncrypt。
name	String	子设备通道名称。
userName	String	用户名。
serverId	String	子设备通道ID。
url	String	服务器连接地址。

参数	类型	说明
timeout	Integer	超时时间。

OPC UA的password加密方式

加密算法采用AES算法，使用128位（16字节）分组，缺省的mode为CBC，缺省的padding为PKCS5Padding， 秘钥使用设备的DeviceSecret，加密后的内容采用BASE64进行编码。

加解密示例代码：

说明：

Java版本请采用Java 9、Java 8u161、Java 7u171、Java 6u181及以上版本。Java版本过低时运行以下示例会报错，此类报错具体解释，请参考 Java Bug Database。

```
private static String instance = "AES/CBC/PKCS5Padding";

    private static String algorithm = "AES";

    private static String charsetName = "utf-8";
    /**
     * 加密算法
     *
     * @param data              待加密内容
     * @param deviceSecret      设备的秘钥
     * @return
     */
    public static String aesEncrypt(String data, String deviceSecret
) {
        try {
            Cipher cipher = Cipher.getInstance(instance);
            byte[] raw = deviceSecret.getBytes();
            SecretKeySpec key = new SecretKeySpec(raw, algorithm);
            IvParameterSpec ivParameter = new IvParameterSpec(
deviceSecret.substring(0, 16).getBytes());
            cipher.init(Cipher.ENCRYPT_MODE, key, ivParameter);
            byte[] encrypted = cipher.doFinal(data.getBytes(
charsetName));

            return new BASE64Encoder().encode(encrypted);
        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    public static String aesDecrypt(String data, String deviceSecret
) {
        try {
            byte[] raw = deviceSecret.getBytes(charsetName);
            byte[] encrypted1 = new BASE64Decoder().decodeBuffer(
data);
            SecretKeySpec key = new SecretKeySpec(raw, algorithm);
            Cipher cipher = Cipher.getInstance(instance);
            IvParameterSpec ivParameter = new IvParameterSpec(
deviceSecret.substring(0, 16).getBytes());
```

```
            cipher.init(Cipher.DECRYPT_MODE, key, ivParameter);
            byte[] originalBytes = cipher.doFinal(encrypted1);
            String originalString = new String(originalBytes,
charsetName);
            return originalString;
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return null;
    }

    public static void main(String[] args) throws Exception {
        String text = "test123";
        String secret = "testTNmjyWHQzniA8wEkTNmjyWH****";
        String data = null;
        data = aesEncrypt(text, secret);
        System.out.println(data);
        System.out.println(aesDecrypt(data, secret));
    }
```

deviceList说明

- **Modbus协议下子设备的连接配置数据示例：**

```
{
  "deviceConfig": {
    "slaveId": 1,
    "serverId": "D73251B42777****"
  },
  "productKey": "a1ZlGQv****",
  "deviceName": "test01"
}
```

参数说明如下：

参数	类型	取值
deviceConfig	Object	设备连接配置信息。
slaveId	Integer	从站ID。
serverId	String	子设备通道ID。
productKey	String	子设备的产品Key。
deviceName	String	子设备名称。

- **OPC UA协议**

```
{
  "deviceConfig": {
    "displayNamePath": "123",
    "serverId": "44895C63E3FF4013924CEF31519A****"
  },
  "productKey": "a1ZlGQv****",
  "deviceName": "test_02"
```

```
}
```

参数说明如下：

参数	类型	说明
deviceConfig	Object	设备连接配置信息。
productKey	String	子设备的产品Key。
deviceName	String	子设备名称。
displayNamePath	String	自定义的展示名称。
serverId	String	子设备通道ID。

6.8 设备禁用、删除

网关类设备可以禁用和删除子设备。

禁用设备

下行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/disable`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/disable_reply`

设备禁用提供设备侧的通道能力，云端使用异步的方式推送该消息， 设备订阅该Topic。具体禁用功能适用于网关类型设备，网关可以使用该功能禁用相应的子设备。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.disable"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	取值	说明
id	String	消息ID号，由物联网平台生成。

参数	取值	说明
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数， 为空即可。
method	String	请求方法，取值thing.disable。
code	Integer	结果信息， 具体参考设备端通用code。

恢复禁用

下行

- 请求Topic：/sys/{productKey}/{deviceName}/thing/enable
- 响应Topic：/sys/{productKey}/{deviceName}/thing/enable_reply

设备恢复禁用提供设备侧的通道能力，云端使用异步的方式推送消息，设备订阅该topic。具体恢复禁用功能适用于网关类型设备，网关可以使用该功能恢复被禁用的子设备。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.enable"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	取值	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数， 为空即可。
method	String	请求方法，取值thing.enable。
code	Integer	结果信息， 具体参考设备端通用code。

删除设备

下行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/delete`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/delete_reply`

删除设备提供设备侧的通道能力，云端使用异步的方式推送消息，设备订阅该topic。具体删除设备的功能适用于网关类型设备，网关可以使用该功能删除相应的子设备。

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	取值	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数， 为空即可。
method	String	请求方法，取值thing.delete。
code	String	结果信息， 具体参考设备端通用code。

6.9 设备标签

设备上报部分信息，如厂商、设备型号等，可以保存为设备标签。

标签信息上报

上行

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

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature",
      "attrValue": "36.8"
    }
  ],
  "method": "thing.deviceinfo.update"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明:

参数	类型	说明
id	String	消息ID号。上行消息ID需您自定义。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	请求参数，包含标签的键attrKey和值attrValue。params元素个数不超过200个。
method	String	请求方法，取值thing.deviceinfo.update。
attrKey	String	标签Key。 · 长度不超过64字节。 · 仅允许字符集为英文大小写字母、数字和下划线。 · 首字符必须是字母或者下划线。
attrValue	String	标签的值。可包含中文汉字、英文字母、数字、下划线(_)、连字符(-)和点号(.)。
code	Integer	结果信息， 200表示成功。

错误码

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

删除标签信息

上行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/deviceinfo/delete`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/deviceinfo/delete_reply`

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature"
    }
  ],
  "method": "thing.deviceinfo.delete"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明：

参数	类型	说明
id	String	消息ID号。上行消息ID需您自定义。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为`1.0`。
params	Object	请求参数，包含要删除的标签键`attrKey`参数。
method	String	请求方法，取值`thing.deviceinfo.delete`。
attrKey	String	要删除标签的Key。
code	Integer	结果信息， 200表示成功。

错误信息

错误码	消息	描述
460	request parameter error	请求参数错误。
6100	device not found	设备不存在。

6.10 TSL模板

设备可以通过上行请求获取设备的TSL模板。

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

Alink请求数据格式

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.dsltemplate.get"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {
    "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.
json",
    "link": "/sys/1234556554/airCondition/thing/",
    "profile": {
      "productKey": "1234556554",
      "deviceName": "airCondition"
    },
    "properties": [
      {
        "identifier": "fan_array_property",
        "name": "风扇数组属性",
        "accessMode": "r",
        "required": true,
        "dataType": {
          "type": "array",
          "specs": {
            "size": "128",
            "item": {
              "type": "int"
            }
          }
        }
      }
    ],
    "events": [
      {
        "identifier": "alarm",
        "name": "alarm",
        "desc": "风扇警报",
        "type": "alert",
        "required": true,
        "outputData": [
          {
            "identifier": "errorCode",
            "name": "错误码",
            "dataType": {
              "type": "text",
```

```
            "specs": {
              "length": "255"
            }
          }
        }
      ],
      "method": "thing.event.alarm.post"
    }
  ],
  "services": [
    {
      "identifier": "timeReset",
      "name": "timeReset",
      "desc": "校准时间",
      "inputData": [
        {
          "identifier": "timeZone",
          "name": "时区",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "512"
            }
          }
        }
      ],
      "outputData": [
        {
          "identifier": "curTime",
          "name": "当前的时间",
          "dataType": {
            "type": "date",
            "specs": {}
          }
        }
      ],
      "method": "thing.service.timeReset"
    },
    {
      "identifier": "set",
      "name": "set",
      "required": true,
      "desc": "属性设置",
      "method": "thing.service.property.set",
      "inputData": [
        {
          "identifier": "fan_int_property",
          "name": "风扇整数型属性",
          "accessMode": "rw",
          "required": true,
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "100",
              "unit": "g/ml",
              "unitName": "毫升"
            }
          }
        }
      ],
      "outputData": []
    },
    {
```

```
        "identifier": "get",
        "name": "get",
        "required": true,
        "desc": "属性获取",
        "method": "thing.service.property.get",
        "inputData": [
          "array_property",
          "fan_int_property",
          "batch_enum_attr_id",
          "fan_float_property",
          "fan_double_property",
          "fan_text_property",
          "fan_date_property",
          "batch_boolean_attr_id",
          "fan_struct_property"
        ],
        "outputData": [
          {
            "identifier": "fan_array_property",
            "name": "风扇数组属性",
            "accessMode": "r",
            "required": true,
            "dataType": {
              "type": "array",
              "specs": {
                "size": "128",
                "item": {
                  "type": "int"
                }
              }
            }
          }
        ]
      }
    ]
  }
}
```

参数说明:

参数	取值	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
params	Object	为空即可。
method	String	请求方法，取值thing.dsltemplate.get。
productKey	String	产品的Key，示例中取值为1234556554。
deviceName	String	设备名称，示例中取值为airCondition。
data	Object	物的TSL描述，具体参考物模型文档。

错误码

错误码	消息	描述
460	request parameter error	请求参数错误。
6321	tsl: device not exist in product	设备不存在。

6.11 固件升级

物联网平台提供固件升级与管理服务。本文介绍固件升级数据上下行Topic和Alink协议下，固件升级的上下行数据格式，包括设备上报固件版本、物联网平台推送固件信息、设备上报升级进度和设备请求获取最新固件信息。

关于固件升级开发设置流程，请参考设备OTA升级和用户指南固件升级。

设备上报固件版本

数据上行

- Topic：/ota/device/inform/${YourProductKey}/${YourDeviceName}

  设备通过这个Topic上报当前使用的固件版本信息。

Alink请求数据格式

```
{
  "id": 1,
  "params": {
    "version": "1.0.1"
  }
}
```

参数说明

参数	取值	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	设备固件的版本信息。

物联网平台推送固件信息

数据下行

- Topic：/ota/device/upgrade/${YourProductKey}/${YourDeviceName}

  物联网平台通过这个Topic推送固件信息， 设备订阅该Topic可以获得固件信息。

Alink请求数据格式

```
{
  "code": "1000",
```

```
  "data": {
    "size": 432945,
    "version": "2.0.0",
    "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e",
    "sign": "93230c3bde425a9d7984a594ac55ea1e",
    "signMethod": "Md5"
  },
  "id": 1507707025,
  "message": "success"
}
```


参数说明

参数	取值	说明
id	String	消息ID号，由物联网平台生成。
message	String	结果信息。
code	String	状态码。
version	String	设备固件的版本信息。
size	Long	固件大小，单位：字节。
url	String	固件在对象存储（OSS）上的存储地址。
sign	String	固件签名。
signMethod	String	签名方法， 目前支持Md5，Sha256两种签名方法。
md5	String	作为保留字段，兼容老的设备信息。当签名方法为Md5时，除了会给sign赋值外还会给md5赋值。

设备上报升级进度

数据上行

- Topic:/ota/device/progress/${YourProductKey}/${YourDeviceName}

  固件升级过程中，设备可以通过这个Topic上报固件升级的进度百分比。

Alink请求数据格式


```
{
  "id": 1,
```

```
  "params": {
    "step": "-1",
    "desc": "固件升级失败，请求不到固件信息"
  }
}
```

参数说明

参数	取值	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
step	String	固件升级进度信息。 取值范围： · [1，100] 之间的数字：表示升级进度百分比。 · -1 ： 表示升级失败。 · -2： 表示下载失败。 · -3： 表示校验失败。 · -4 ： 表示烧写失败。
desc	String	当前步骤的描述信息。如果发生异常，此字段可承载错误信息。

设备请求固件信息

数据上行

· Topic：/ota/device/request/${YourProductKey}/${YourDeviceName}

设备通过这个Topic主动请求固件信息。

Alink请求数据格式

```
{
  "id": 1,
  "params": {
    "version": "1.0.1"
  }
}
```

参数说明

参数	取值	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	设备固件的版本信息。

物联网平台收到设备请求后，响应请求。物联网平台响应数据格式如下：

- 下发固件信息。返回数据格式如下：

```
{
"code":"1000",
"data":{
"size":93796291,
"sign":"f8d85b250d4d787a9f483d89a9747348",
"version":"1.0.1.9.20171112.1432",
"url":"https://the_firmware_url",
"signMethod":"Md5",
"md5":"f8d85b250d4d787a9f483d89a9747348"
},
"id":8758548588458,
"message":"success"
}
```

- 无固件信息下发。返回数据格式如下：

```
:{
"code":500,
"message":"none upgrade operation of the device."
}
```

返回参数说明，请参见物联网平台推送固件信息章节中的参数说明。

6.12 远程配置

本文档介绍设备主动请求配置信息和物联网平台推送配置信息的Topic及Alink数据格式。远程配置的具体使用方法，请参考用户指南远程配置文档。

设备主动请求配置信息

上行

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

Alink请求数据格式

```
{
  "id": 123,
  "version": "1.0",
  "params": {
    "configScope": "product",
    "getType": "file"
  },
  "method": "thing.config.get"
}
```

Alink响应数据格式

```
{
```

```
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  }
}
```

参数说明

参数	类型	说明
id	String	消息ID号。需定义为String类型的数字，且设备维度唯一。
version	String	协议版本号，目前协议版本号为1.0。
configScope	String	配置范围， 目前只支持产品维度配置。 取值：product。
getType	String	获取配置类型。 目前支持文件类型，取值：file。
method	String	请求方法，取值：thing.config.get。
configId	String	配置文件的ID。
configSize	Long	配置文件大小，按字节计算。
sign	String	签名。
signMethod	String	签名方法，仅支持Sha256。
url	String	存储配置文件的对象存储（OSS）地址。
code	Integer	结果码。返回200表示成功，返回其他状态码，表示失败。具体参考设备端通用code。

错误码

错误码	消息	描述
6713	thing config function is not available	产品的远程配置功能不可用，需要在控制台，远程配置页面打开配置开关。
6710	no data	没有配置的数据。

配置推送

下行

- 请求Topic：`/sys/{productKey}/{deviceName}/thing/config/push`
- 响应Topic：`/sys/{productKey}/{deviceName}/thing/config/push_reply`

设备订阅该Topic后，您在物联网控制台批量推送配置信息时，物联网平台采用异步推送方式向设备推送信息。

Alink请求数据格式：

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  },
  "method": "thing.config.push"
}
```

Alink响应数据格式

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

参数说明

参数	类型	说明
id	String	消息ID号，由物联网平台生成。
version	String	协议版本号，目前协议版本号为1.0。
configScope	String	配置范围， 目前只支持产品维度配置。 取值：product。
getType	String	获取配置类型，目前支持文件类型，取值：file。
configId	String	配置的ID。
configSize	Long	配置大小，按字节计算。
sign	String	签名。
signMethod	String	签名方法，仅支持sha256。
url	String	存储配置文件的对象存储（OSS）地址。
method	String	请求方法，取值：thing.config.push。
code	Integer	结果信息， 具体参考设备端通用code。

您可以使用规则引擎数据流转功能，将设备返回的响应结果转发至其他Topic和其他阿里云服务中。具体的设备响应数据Topic和数据格式请参考设备下行指令结果数据流转。

6.13 设备端通用code

设备通用code信息，用于表达云端下行推送时设备侧业务处理的返回结果。

错误码	消息	描述
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，设备端处理不过来时可以使用
100000-110000	自定义的错误信息	从100000到110000的错误码用于设备自定义错误信息，和云端错误信息加以区分