目录

- 产品架构 1.1
- 平台介绍 2.1
  - 产品背景 2.1.1
  - 产品架构 2.1.2
  - 产品特点 2.1.3
  - 产品优势 2.1.4
  - 边缘网关 2.1.5
  - 设备和应用接入 2.1.6
  - 部分应用案例 2.1.7
  - 名词解释 2.1.8
- 平台入驻 2.2
  - 注册登录 2.2.1
  - 实名认证 2.2.2
- 快速入门 2.3
  - 业务使用全流程 2.3.1
  - 使用模拟器体验全流程 2.3.2
  - 使用Postman模拟应用接入 2.3.3
- 设备接入 2.4
  - 创建产品 2.4.1
  - 创建设备 2.4.2
    - 创建单个设备 2.4.2.1
    - 批量创建设备 2.4.2.2
  - 消息通信Topic 2.4.3
    - 产品Topic类 2.4.3.1
    - 设备Topic类 2.4.3.2
  - 物模型 2.4.4
  - 数据解析 2.4.5
  - 在线调试 2.4.6
  - 使用开放协议自主接入 2.4.7
    - MQTT协议接入 2.4.7.1
      - 使用MQTT.fx接入 2.4.7.1.1
      - 使用SDK接入 2.4.7.1.2
    - CoAP协议接入 2.4.7.2
    - LwM2M协议接入 2.4.7.3
    - HTTP协议接入 2.4.7.4
- 消息通信 2.5
  - 服务端订阅 2.5.1

规则引擎 2.5.2
规则引擎推送实例 2.5.3
产品管理 2.6
物模型 2.6.1
添加协议参数 2.6.1.1
添加协议命令 2.6.1.2
数据解析 2.6.2
远程配置 2.6.3
设备管理 2.7
设备生命周期管理 2.7.1
创建单个设备 2.7.1.1
批量创建设备 2.7.1.2
设备导出 2.7.1.3
禁用启用与删除 2.7.1.4
设备详情 2.7.2
设备影子 2.7.3
上下行数据 2.7.4
网关与设备 2.7.5
KLink 2.7.6
远程升级 2.7.7
硬件升级 2.7.7.1
软件升级 2.7.7.2
数据分析 2.8
地理分析 2.8.1
总体趋势 2.8.2
指标趋势 2.8.3
指标聚合 2.8.4
边缘网关 2.9
硬件网关 2.9.1
概述 2.9.1.1
硬件介绍 2.9.1.2
功能介绍 2.9.1.3
工程配置 2.9.1.4
管理后台 2.10
用户管理 2.10.1
配额管理 2.10.2
品类管理 2.10.3
数据分析 2.10.4
系统监控 2.10.5
应用开发指南及API 2.11

设备管理 2.11.1
新增设备 2.11.1.1
导入设备 2.11.1.2
获取批次下所有设备 2.11.1.3
更改设备名称 2.11.1.4
查询设备详情 2.11.1.5
批量查询状态 2.11.1.6
删除设备 2.11.1.7
数据管理 2.11.2
查询历史上下行数据 2.11.2.1
查询设备影子 2.11.2.2
查询设备指标聚合 2.11.2.3
查询设备指标趋势 2.11.2.4
设备命令相关 2.11.3
下发控制命令 2.11.3.1
查询命令状态 2.11.3.2
查询历史控制命令 2.11.3.3
错误码 2.11.4
数据订阅 2.11.5
资料下载 2.12

- 产品背景

产品背景

各类IoT产品和解决方案的飞速发展，为市场提供了新的机遇，从而根本上转变了竞争激烈的的商业环境。越来越多的企业已经通过物联技术实现产品升级甚至企业转型，从而在大趋势中牢牢占据领先位置。 虽然价值显而易见，但挑战严峻，尤其是在技术层面：

- 技术链条长，意味着研发团队规模不小，且包含各技术领域人才，然而IoT领域人才稀缺；
- 技术稳定期长，需要大量的实战经验解决可能存在的各种难题；
- 海量连接和数据存储及处理该如何解决？如城域级别的物联网项目可能要求支撑百万级的并发连接量或十万级的QPS；
- IoT场景未知、多样、复杂，如何快速应对？如在园区场景中，可能会涉及到几百种不同型号的设备以及若干复杂的跨子系统联动需求；
- 数据安全性如何保障？传统物联系统仅依靠对比IMEI等基础的鉴权机制，以及孱弱的防攻击能力，可能会导致巨大风险；
- 不同项目中对部署方式有不同的要求，如公有云部署、私有云部署甚至直接部署在物理机上；
- ……

本质上，企业需要一个性能稳定、扩展灵活且轻量化的系统，能够针对不同的业务形态，将设备和业务的解耦，进而实现各种复杂的场景和业务流程。正是在这一背景下，IoT OS应运而生。

- 产品背景

产品背景

各类IoT产品和解决方案的飞速发展，为市场提供了新的机遇，从而根本上转变了竞争激烈的的商业环境。越来越多的企业已经通过物联技术实现产品升级甚至企业转型，从而在大趋势中牢牢占据领先位置。 虽然价值显而易见，但挑战严峻，尤其是在技术层面：

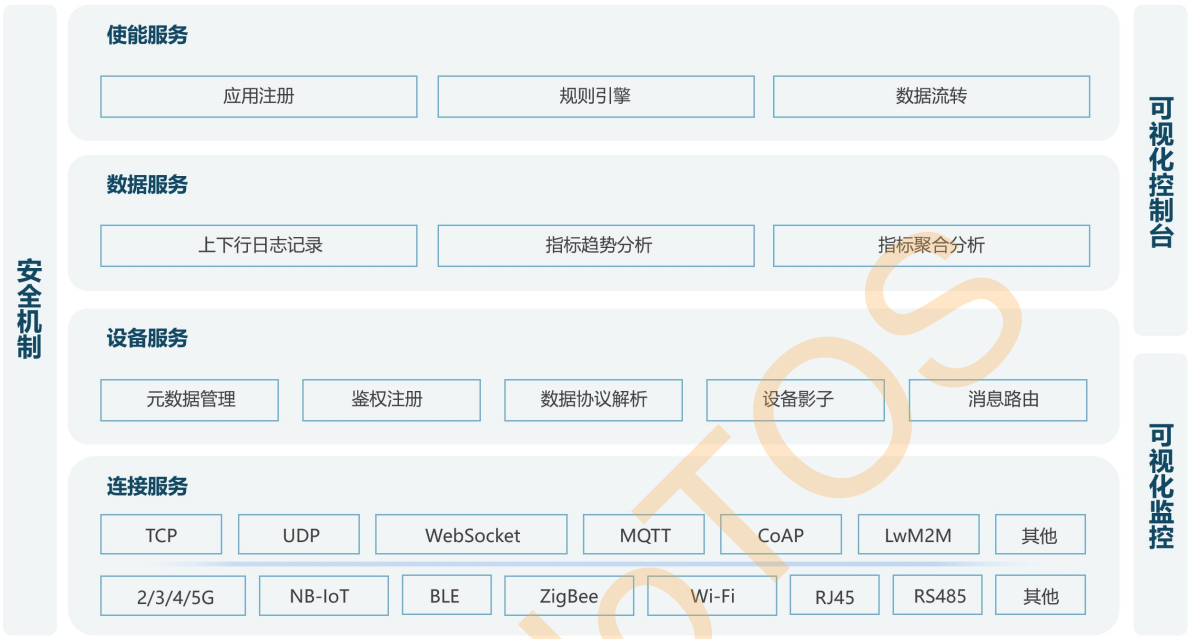
- 技术链条长，意味着研发团队规模不小，且包含各技术领域人才，然而IoT领域人才稀缺；
- 技术稳定期长，需要大量的实战经验解决可能存在的各种难题；
- 海量连接和数据存储及处理该如何解决？如城域级别的物联网项目可能要求支撑百万级的并发连接量或十万级的QPS；
- IoT场景未知、多样、复杂，如何快速应对？如在园区场景中，可能会涉及到几百种不同型号的设备以及若干复杂的跨子系统联动需求；
- 数据安全性如何保障？传统物联系统仅依靠对比IMEI等基础的鉴权机制，以及孱弱的防攻击能力，可能会导致巨大风险；
- 不同项目中对部署方式有不同的要求，如公有云部署、私有云部署甚至直接部署在物理机上；
- ……

本质上，企业需要一个性能稳定、扩展灵活且轻量化的系统，能够针对不同的业务形态，将设备和业务的解耦，进而实现各种复杂的场景和业务流程。正是在这一背景下，IoT OS应运而生。

- 产品架构

产品架构

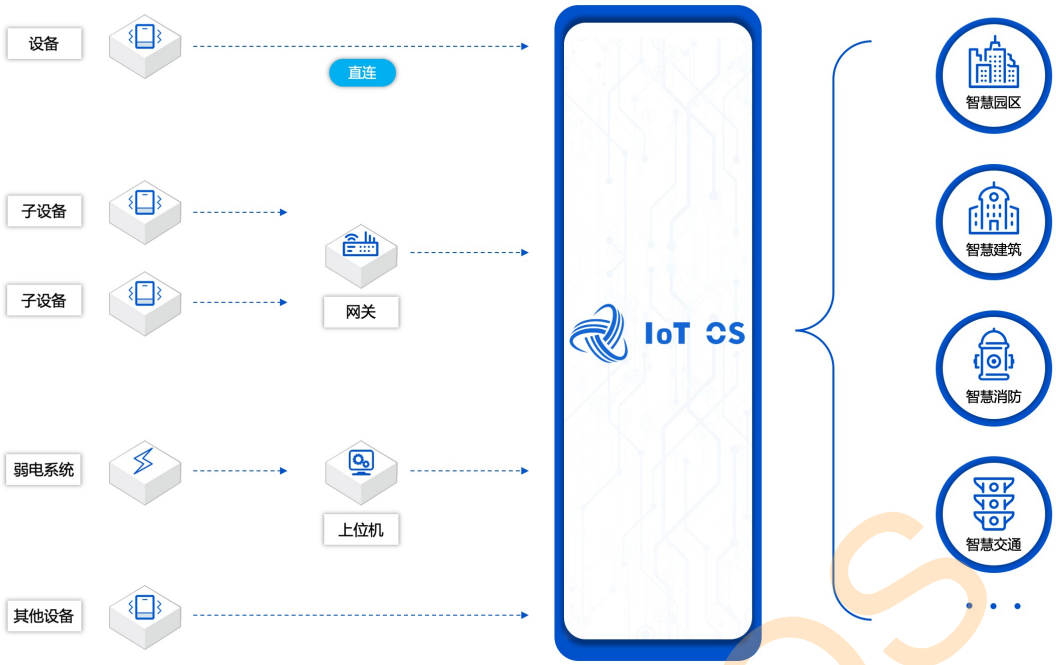
IoT OS是一款物联网云操作系统，为万物互联提供可靠安全稳定的终端接入、协议适配、消息路由、数据存储和分析、应用使能等核心功能。其 功能架构 如下：



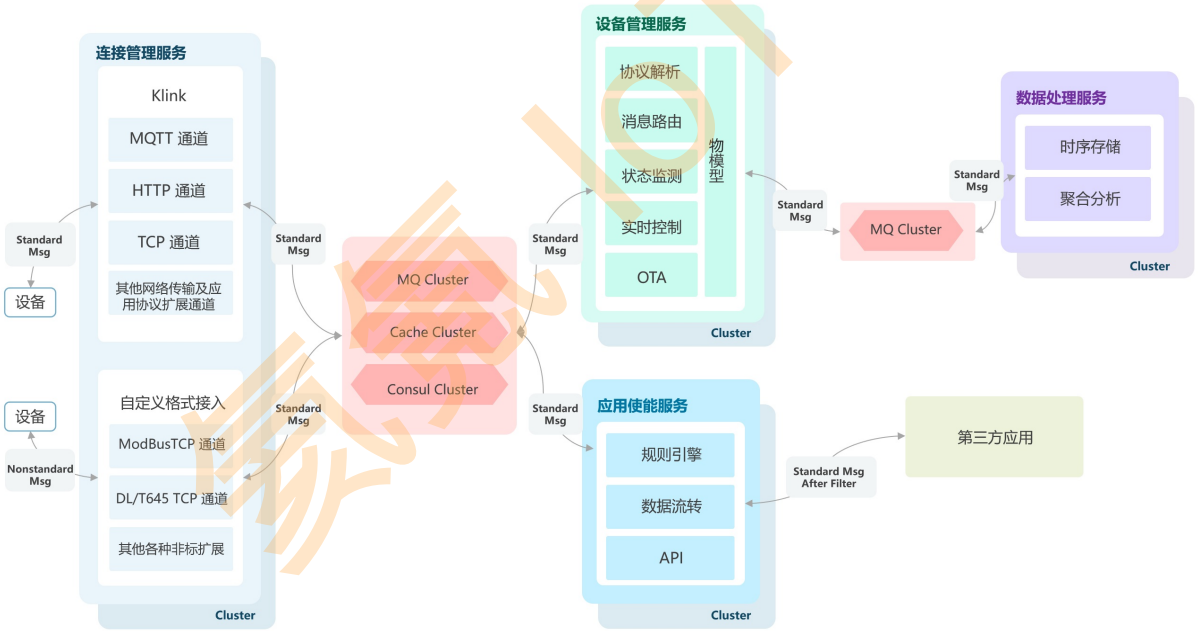
上图中各层服务负责的功能分别如下：

- 连接服务：提供设备联网功能，支持设备通过各种无线或有线的通信方式接入网络，并支持各种网络传输协议；
- 设备服务：提供设备基础管理功能，包括设备的鉴权管理、数据协议解析、消息路由、设备影子数据及元数据管理功能；
- 数据服务：提供设备数据的基本管理功能，包括设备的上下行日志存储，以及一些数据指标的聚合分析，如平均值、最大、最小值等；
- 使能服务：提供应用使能服务，主要是为上层或第三方应用提供按规则和条件进行数据订阅和数据转发的服务。包括应用注册、规则引擎、数据流转服务；
- 其他：包括基础的安全服务、控制台和监控服务。安全服务提供基本物联网安全机制；可视化控制台提供IoT OS与客户交互的界面；可视化监控提供服务可用性及风险监控能力。

IoT OS在物联网应用中常处于如下图所示位置：



IoT OS本质上是一系列物联网微服务的集合，采用分布式架构，应用程序和服务组件均不存在单点风险。其 技术架构如下：



- 产品特点

产品特点

1. 高性能

IoT OS单集群可支持百万级设备连接和十万级QPS请求，具有毫秒级延迟，可支撑超大规模物联网设备接入需求。

2. 电信级高可靠

IoT OS采用大规模分布式、高可用、高可靠集群，并提供多重安全机制。

3. 多协议支持

IoT OS支持主流物联网协议，包括MQTT、CoAP、LwM2M和HTTP等，并配套若干款边缘硬件网关实现PLC、OPC等接入。同时，IoT OS还提供边缘侧开发框架，支持更多私有协议的快速接入。

4. 网络透明

IoT OS支持各种蜂窝网络(2G、3G、4G/NB-IoT) 、有线网络和无线网络，亦能支持5G中的mMTC（海量物联）和uRLLC（高可靠低时延）两大场景。

5. 数据持久化和实时分析

IoT OS内置高性能分布式时序数据库，支持海量数据的持久化和实时分析，也支持与用户提供的数据库、大数据平台或消息列对接，方便用户对数据进行存储与消费。

6. 规则引擎

IoT OS提供基于SQL的高速规则引擎，支持用户设置复杂规则实时过滤海量数据，减少应用侧压力，使客户聚焦业务开发。

7. 可伸缩

基于良好的架构设计，IoT OS具备强大的可伸缩能力。根据业务压力，既能以数百台规模的超大集群形式运行，也可以在超小配置上运行，并支持以公有云、私有云、物理机及Docker容器和K8S等方式进行灵活部署，因此能非常好地适应端-边-云体系。

IoT OS具备以上诸多特点，因此特别适合在以下场景中使用：

- 应用中有大规模感知设备接入，设备量可能是百万级，请求量和数据存储需求巨大；
- 应用中的感知设备多元异构，通信方式、传输协议、业务数据等复杂多样；
- 处于架构优化的目的，需要将感知层与应用层解耦，实现与复杂应用系统的快速对接；
- 需要构建统一的PaaS级平台，实现标准规范的接入及应用开发；
- 需要构建端-边-云体系，实现跨区域统一化管理；

IoT OS目前被广泛应用于车联网、城市物联网、工业物联网、智能家居等跨行业领域。

- 产品优势

产品优势

IoT OS的方案相较于传统开发平台有着下表所列的优势。

	传统开发平台	IoT OS
设备接入	需要搭建基础设施，联合嵌入式开发人员与云端开发人员共同开发。开发工作量大、效率低。	提供设备端SDK，快速连接设备上云，效率高。同时支持全球设备接入、异构网络设备接入、多环境下设备接入和多协议设备接入。
性能	自行实现扩展性架构，极难做到从设备粒度调度服务器、负载均衡等基础设施。	具有百万级设备的长连接能力、百万级并发处理能力，架构支撑水平性扩展。
安全	需要额外开发、部署各种安全措施，保障设备数据安全是个极大挑战。	提供多重防护，保障设备数据安全。设备认证保障设备安全与唯一性。传输加密保障数据不被篡改。
简单易用	需要购买服务器搭建负载均衡分布式架构，需要花费大量人力物力开发"接入 + 计算 + 存储"一整套物联网系统。	一站式设备管理，实时监控设备运行状态。快速、灵活、简便的搭建复杂物联网应用。

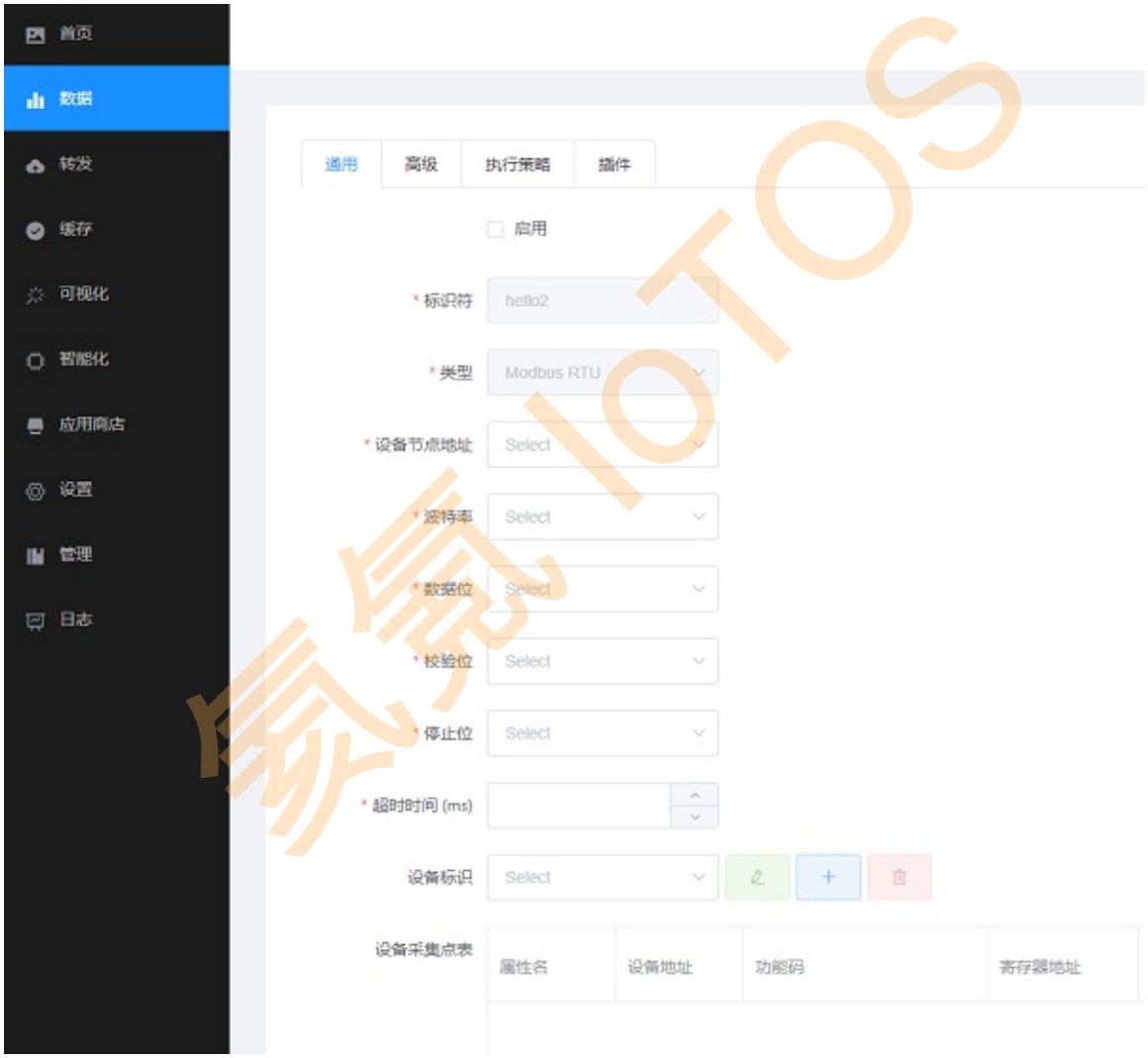
- 边缘网关

边缘网关

物联网协议网关是靠近工业设备、传感器等物理设备的网络边缘侧，亦称边缘网关，主要担负物联网协议转换的功能。边缘网关是物联网系统中端侧设备数据通往云端的最后一层物理实体，因此对实现端边云协同计算具有重要意义。

IoT OS配套若干款边缘网关，均为基于物联网架构设计的工业级嵌入式软硬件一体化设备。可实现工业现场各种设备的数据采集、存储、转发、系统维护、域名管理等功能。用户可在PC、手机上使用浏览器进行数据监控和参数配置。 边缘网关具备对下（自动化系统）协议解析能力（通讯协议：Modbus、DL/T645、CJ/T188等；总线协议：OPC UA、BACNET/IP、KNX等）；具备对上（IT系统）的协议对接能力（MQTT、TCP、UDP、HTTP），对上的通讯能力（以太网、Wi-Fi、4G）；具备对下（采集）和对上（转发）的私有协议二次开发功能。

边缘网关内置Web-UI，方便用户进行配置管理。



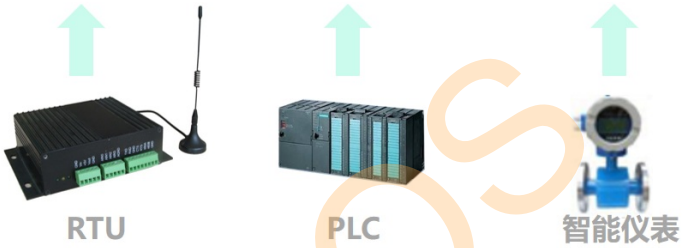
边缘网关可广泛应用于光伏电、风电站、小水电、配电室、抽油机、灌区、管道、环境监测站、消防监控室、农业大棚、空气监测站、实验室、工控中心、无人值守站等场站监控和园区能源管理系统以及各种在线监测系统等。

如下图展示了边缘网关在智慧工厂中的典型应用：

Web UI 终端 >>

智能网关 >>

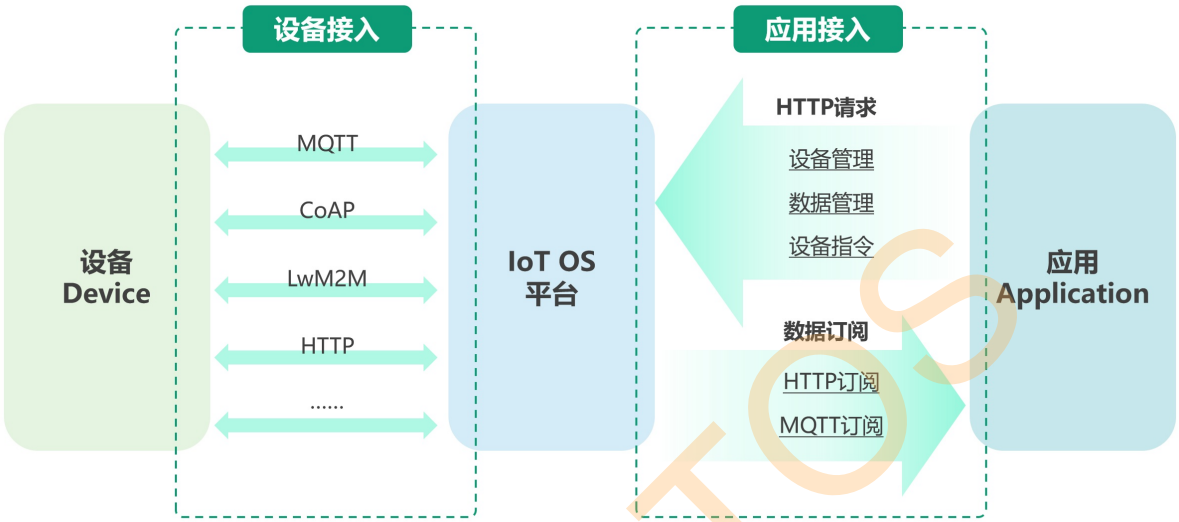
硬件设备 >>



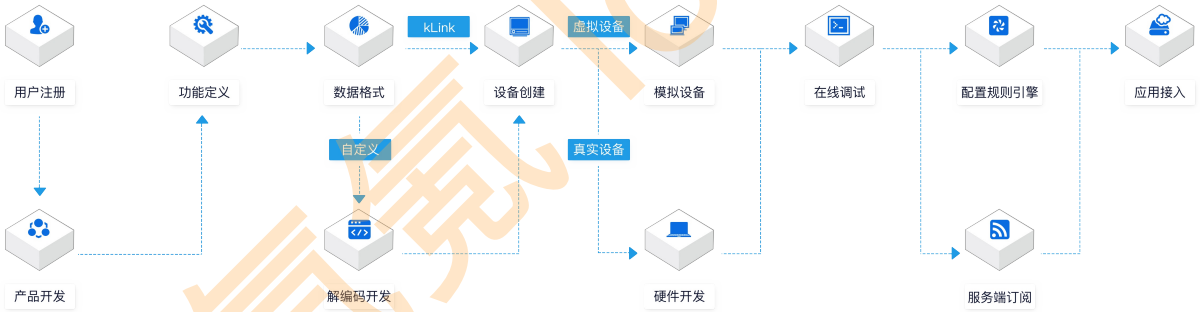
- 设备和应用接入

设备和应用接入

IoT OS的南向能力体现在各类多元异构设备的接入，北向能力则体现在对各种业务系统、应用场景的支撑能力。



用户只需登录WEB UI，基于其提供的一站式接入能力，编写少量代码，即可快速实现设备上云和应用开发。



- 部分应用案例

部分应用案例

1. 电信运营商DMP

随着通信技术的发展，物联网逐渐深入到各行各业中。其中运营商提供2/3/4G卡以及NB-IoT卡，为物联网通信中的重要组成部分。物联网设备通信方式、网络传输层、应用层、业务层协议多样繁杂，为更好帮助客户进行物联网项目落地，需要从产业化、生态化规划建设物联网设备管理平台（DMP）。

某电信运营商基于IoT OS建设DMP，构建起行业竞争壁垒。该DMP以WEB方式提供给行业从业者使用，帮助合作商降低整个产业的运营成本，提高产业的运营质量和效率，并通过新的产业生态为客户创造新的体验和社会价值。通过信息服务、整合渠道等方法扩大业务知名度及影响力。提高用户黏性，抢占产业化运营市场先机，继而促进物联网流量卡的深度运营。

2. 智慧园区综合管理平台

中国智慧城市建设快速发展，已成为全球最大智慧城市实施国。在国家大力扶持下，全国园区智慧化建设如火如荼。但集成商、实施方在智慧园区建设过程中也遇到了巨大的挑战。如园区感知系统里各类设备的通信方式、传输协议，数据协议均多样且复杂，专业性极强，协调各厂商对接设备协议是一项特别繁杂且工作量极其巨大的任务，等等。

合作伙伴以IoT OS作为某智慧园区综合管理平台项目的物联网中台，通过IoT OS实现了在知识领域、技术领域、开发过程、实施过程等各个方面和环节上的各层解耦，实现系统建设的合理分层和标准化。并且实现与第三方数据中台、业务中台的融合打通，从而解决了各子系统互相隔离、信息孤岛的难题。

3. 城市级智慧消防监控平台

当前，在全世界范围内开始充分重视对"智慧城市"的构建。作为"智慧城市"构建中的一个重要的组成部分，"智慧消防"建设也逐渐得到中国各级消防监管机构的充分重视，其对"智慧城市"的良好构建与发展具有极大的推动作用。

合作伙伴以IoT OS为核心建设某城市级智慧消防监管平台，具备大规模消防基础设施联网、火灾实时感知、数据精准分析、告警定向推送等能力。该平台充分发挥IoT OS异构设备多协议扩展接入能力，高效集成如NB-IoT独立烟感、2G独立电气火灾监测设备等新型消防感知设备，并通过消防用户信息传输装置这类边缘硬件网关管理传统消防基础设施。为减少火灾损失甚至预防火灾发生、应对突发事件等加上一道强有力的技术保障。

4. 智慧纺织工业平台

2017年11月27日中国国务院发布的《深化“互联网+先进制造业”发展工业互联网的指导意见》指出，工业互联网通过系统构建网络、平台、安全三大功能体系，打造人、机、物全面互联的新型网络基础设施，形成智能化发展的新兴业态和应用模式。

合作伙伴基于IoT OS打造的纺织业平安智慧工厂平台，服务对象包括企业客户、机器制造商、政府监管部门等，在设备管理、数据采集、消防预防等方面提供服务，同时对接各类行业软件、硬件、通信商展开深度合作，形成生态效应。该平台响应浙江省“企业数字化制造，行业平台化服务”的宗旨，立足于纺织印染行业，以新技术新业态新模式致力于推动传统轻纺行业生产、管理和营销模式的变革。平台以纺机使用环节与用户的交互为切入点，通过采集设备的运行数据、生产数据、能耗数据，与企业运营系统中获取的人员数据、财务数据等进行深度融合，为企业进行赋能，将极大提升当地纺织企业的数字化管理能力及安全生产整体水平。

- 名词解释

名词解释

本节主要介绍IoT OS中相关的名词解释。

名词	解释
产品	设备的集合，通常指一组具有相同功能的设备。物联网平台为每个产品颁发全局唯一的ProductKey。
设备	归属于某个产品下的具体设备。拥有唯一的DeviceId。设备可以直接连接物联网平台，也可以作为子设备通过网关连接物联网平台。
产品品类	产品分类标识。
网关	能够直接连接物联网平台的设备，且具有子设备管理功能，能够代理子设备连接云端。
终端子设备	本质上也是设备。终端子设备不能直接连接物联网平台，只能通过网关连接。
中继设备	无法直接连接物联网平台的设备，但拥有子设备管理功能，只能通过网关连接。
普通设备	本质上也是设备。可以直接连接物联网平台，且不能挂载子设备。
发布	操作Topic的权限类型，对应的英文名称为Pub。可以往此类Topic中发布消息。
订阅	操作Topic的权限类型，对应的英文名称为Sub。可以从此类Topic中订阅消息。
自定义参数	产品物模型下对应的参数，由用户自定义，对应具体设备的属性。
命令	产品物模型下对应的命令，分为上报帧和下发帧。
数据解析脚本	对采用自定义数据格式的设备，通过编写数据解析脚本，可以将设备上报的数据，转换为服务端支持的KLink JSON数据格式。另一方面解析脚本可以将服务端下发的KLink JSON格式数据，转换为设备支持的数据格式。
设备影子	即设备快照，用于存储设备自定义参数的最新值。
上下行数据	设备和服务端通信的所有数据，包括原始数据、编解码数据、服务端响应数据。
规则引擎	通过创建、配置规则，以实现数据流转。
服务端订阅	一种快捷的消息流转订阅模式，可选择数据上报通知、数据变化通知、控制响应通知、设备上下线通知。
PK	即ProductKey，物联网平台颁发的产品唯一标识码。
ProductSecret	由物联网平台颁发的产品密钥，通常与ProductKey成对出现，主要用于设备注册。该参数很重要，需要您保管好，不能泄露。
devSecret	设备秘钥，用于设备注册，该参数很重要，需要您保管好，不能泄露。
Topic	每个设备唯一，是发布（Pub）/订阅（Sub）消息的传输中介。可以向Topic发布或者订阅消息。
D => C	表示指令传输方向为从设备端传向云端。
C => D	表示指令传输方向为从云端传向设备端。

- 注册登录
  - 用户注册
  - 用户登录

注册登录

注册登录功能是IoT OS工作在多租户模式下的必须步骤，但如果用户仅使用IoT OS作为某项目开发中的基础组件（类似数据库），则直接使用superadmin账号（类似数据库的root账户）即可，忽略此章节。

用户注册

在登录页面点击“点击注册”按钮，跳转至注册页面，如下图所示。用户选择用户类型（个人注册或企业注册），依次输入用户名、密码、确认密码和验证码。点击“立即注册”按钮可完成注册流程。

【说明】

```
1. 用户名校验格式为：6-24位只含字母、数字、“_”、“-”的字符；
2. 密码校验格式为：8-20 个字符，需同时包含数字、字母、特殊符号（!@#$%^&*()等非空格），重复密码必须和第一次输入的密码保持一致；
3. 点击验证码图片可刷新验证码；
4. 注册成功后，用户类型不可修改。
```

用户登录

用户在IoT OS登录页面中输入由平台方提供的登录账号及密码，进行登录。


IOT OS
卓越的
物联网连接管理


登录
IOT OS
账号:
superadmin
密码:
下次自动登录
登录
点击注册
物联网连接管理系统

- 实名认证
  - 个人用户认证
  - 企业用户认证

实名认证

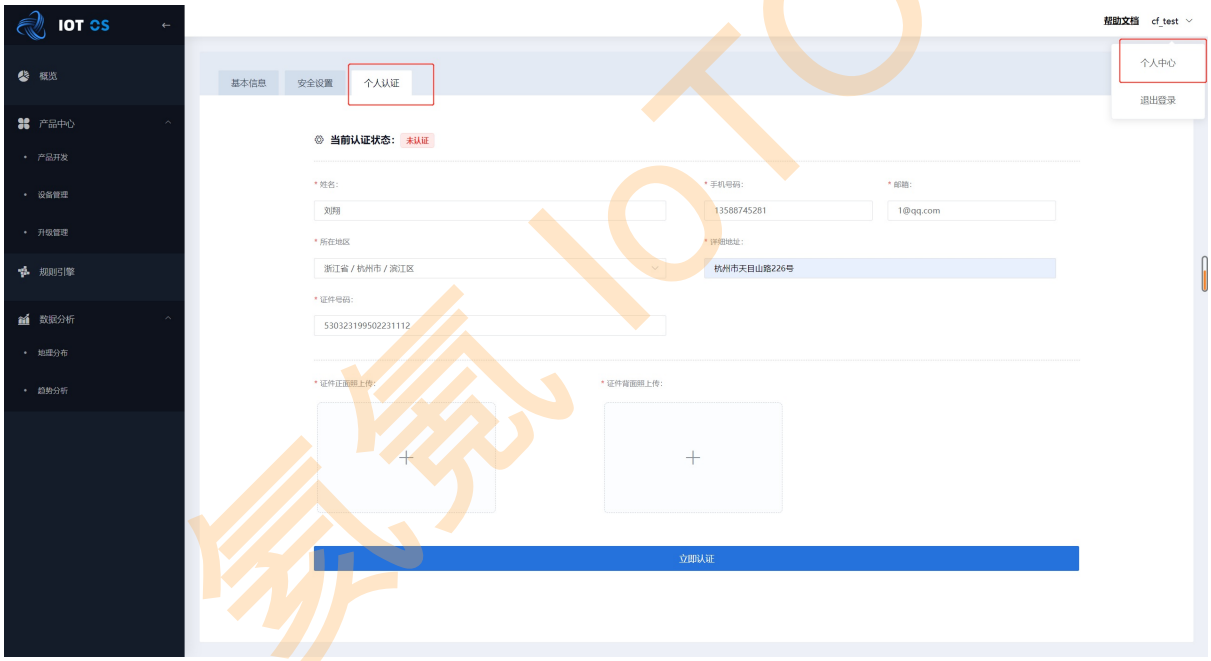
实名认证功能是IoT OS工作在多租户模式下的必须步骤，但如果用户仅使用IoT OS作为某项目开发中的基础组件（类似数据库），则直接使用superadmin账号（类似数据库的root账户）即可，忽略此章节。

个人用户认证

账户注册成功后，使用该账号登录至平台，点击“个人中心”-“个人认证”，填写姓名、手机号、邮箱、身份证号，选择对应的省市区，填入详情地址，然后上传身份证正反面照片，点击“立即认证”按钮即可完成认证流程。

【说明】

```
1. “认证中”和“认证成功”两种状态下，相关信息不可修改；
2. 若管理员审核失败，用户登录可以登录平台，在个人认证页面查看具体拒绝原因，修改相关信息后再次提交认证申请。
```



企业用户认证

注册账户成功后，使用该账号登录至平台，点击“个人中心”-“个人认证”，填写企业名称、联系人姓名、联系人电话、邮箱，选择所在地区，填入详情地址，然后选择所属行业、证件类型、填入企业证书编码，上传企业证件照，点击“立即认证”按钮即可完成认证流程。

【说明】

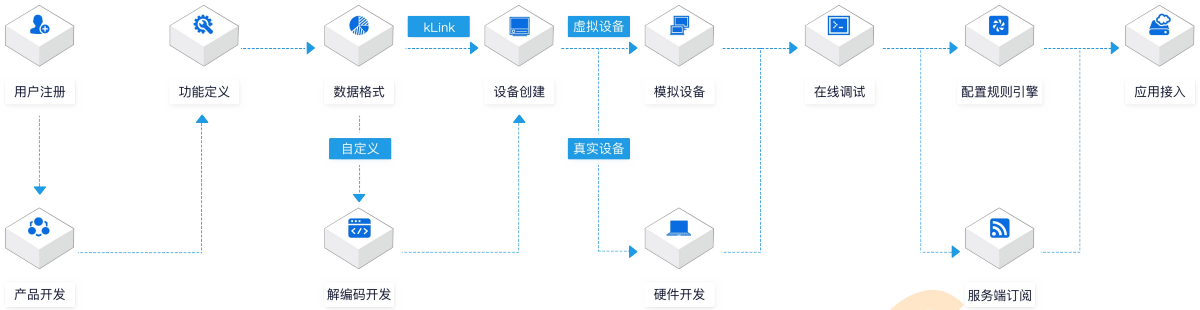
```
上传企业证件照时，工商营业执照、组织机构代码证、税务登记证，
三证选一上传即可（首选工商营业执照），支持（*.png,*.jpg,*.jpeg,*.gif）格式，大小不超过2M。
```


IOT OS
概览
产品中心
产品开发
设备管理
升级管理
规则引擎
数据分析
地理分布
趋势分析
帮助文档
gcf_test2
基本信息
安全设置
企业认证
当前认证状态：
未认证
* 企业名称：
嘉新企业
* 联系人姓名：
展辉
* 联系人电话：
13455456768
* 所在地区
重庆市 / 重庆郊县 / 垫江县
* 详细地址：
天目山路226号
* 邮箱：
2323@qq.com
选择行业
农、林、牧、渔业 / 农业 / 谷物种植
* 证件类型：
社会统一信用代码
* 企业证书编码：
343352132034Z
* 企业证件照上传：
工商营业执照，组织机构代码证，税务登记证，三证选一上传即可，最好是工商营业执照
支持（*.png,*.jpg,*.jpeg,*.gif）格式，大小不超过2M
立即认证

- 业务使用全流程

业务使用全流程

设备接入IoT OS及应用开发流程一览:



1. 用户注册登录

- 平台注册
- 实名认证

若IoT OS运行在多租户模式下，用户需要先在平台注册并且实名认证之后才能进行产品开发操作； 若IoT OS作为某项目开发中的基础组件，请直接使用superadmin账户，忽略此步骤。

2. 产品创建

- 创建产品

用户创建产品供设备接入使用。

3. 功能定义

- 添加协议参数
- 添加协议命令

除了产品的创建，还需要对产品的物模型进行定义。

4. 编解码开发

非必须，若为自定义数据格式，则需要开发编解码插件。

5. 创建设备

- 创建设备

完成产品创建后，创建隶属于此产品的设备。

在文档"设备管理"中有更详细的介绍。

6. 在线调试

使用设备模拟器或真实设备接入平台，在线调试。

7. 消息处理

配置规则引擎或服务端订阅。

8. 应用接入

通过IoT OS提供的HTTP API，北向应用进行开发接入。

- 使用模拟器体验全流程
  - 前提条件
  - MQTT设备接入快速体验
    - 第一步 创建产品
    - 第二步 创建设备
    - 第三步 获取连接信息
    - 第四步 获取模拟器
    - 第五步 设备模拟连接
    - 第六步 命令下发
    - 第七步 命令上报
  - CoAP设备接入快速体验
    - 第一步 创建产品
    - 第二步 创建设备
    - 第三步 获取连接信息
    - 第四步 获取模拟器
    - 第五步 设备模拟连接
    - 第六步 命令下发
    - 第七步 命令上报

使用模拟器体验全流程

本文通过使用模拟器来模拟设备，快速完成设备接入的流程，以方便用户快速熟悉平台各个功能的大致使用。

【注意】

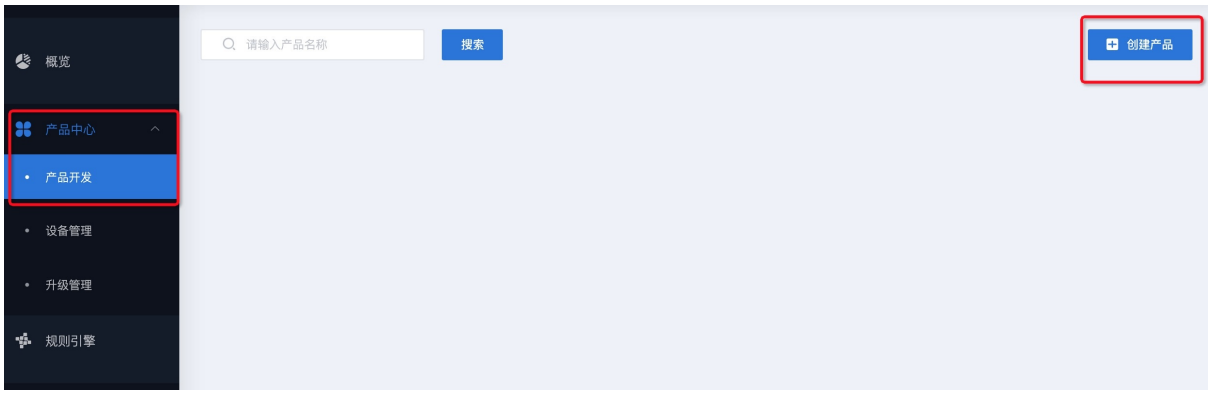
> 此处只是简单使用了各个模块的部分基础功能，更多更详细的功能介绍可以参看文档后面的章节。

前提条件

- 完成平台用户注册
- 完成用户实名认证

MQTT设备接入快速体验

第一步 创建产品



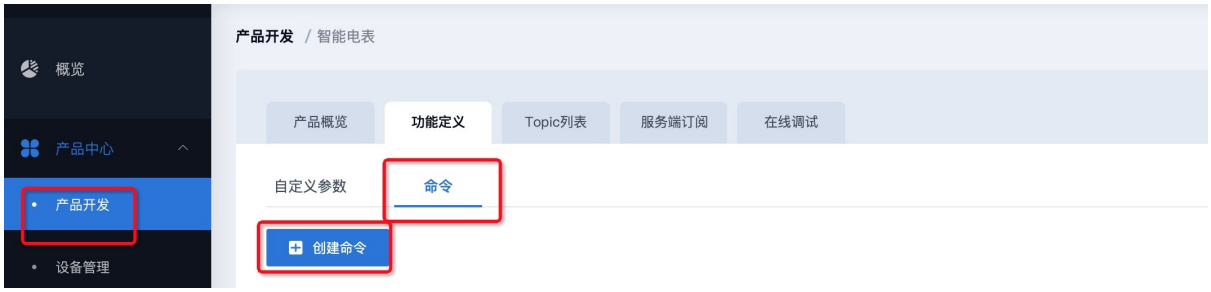
登录平台，进入"产品中心"-"产品开发"，点击右上角"创建产品"按钮。

如图所示书写相关信息，配置相关详情可参考创建产品。点击右边"创建"按钮完成创建。

此时回到"产品开发"可以看到刚刚创建的产品卡片。如图所示。

点击产品卡片，进入产品概览。点击"功能定义"，对产品功能进行定义。

点击"自定义参数"-"创建参数"，如上图编写参数。



再点击"命令"-"创建命令"，分别创建上报命令和下发命令。

上报命令：


修改命令
* 命令名称:
上报开关
* 标识符:
reportPower
* 命令类型:
上报帧
待选参数 0/0
请输入参数名称
无数据
已选参数 0/1
请输入参数名称
开关
取 消
确 定

下发命令：

第二步 创建设备

完成产品创建后，进入"产品中心"-"设备管理"，点击右上角"创建设备"按钮。

如图所示填写设备信息。其中"所属产品为第一步中创建的产品"。 "设备ID"则需要为唯一标识。

点击"确定"按钮完成设备创建。

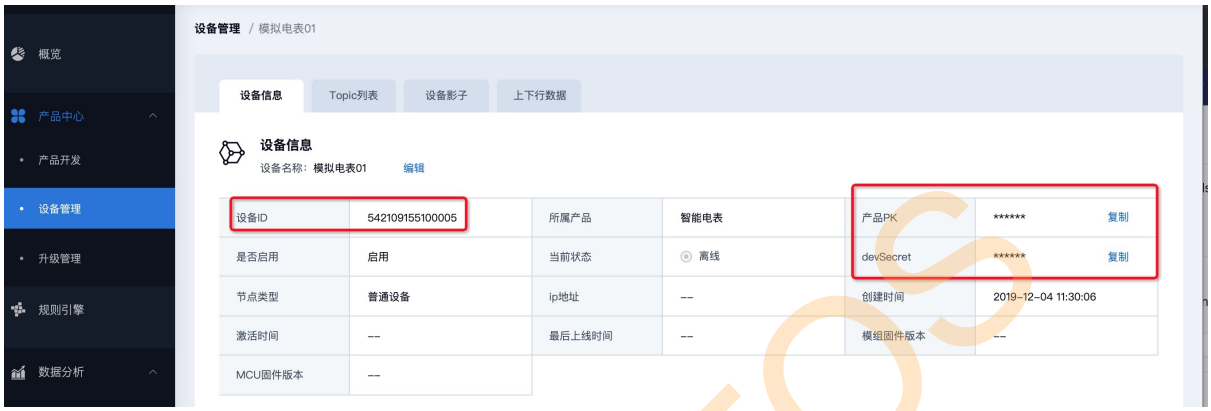
第三步 获取连接信息

完成设备创建后，回到"产品中心"-"产品开发"中，点击第一步创建的产品卡片。进入产品概览，可以获取响应的连接信息。

其中主要使用到的参数是"设备接入信息"。 此例中用到的是MQTT接入方式：117.50.16.141:1883

进入"设备管理"，选择创建的设备，点击"查看"。

即可查看设备详细信息。



如图所示可以得到设备的设备ID、设备所属产品pk以及devSecret。

点击"Topic列表"可以获得MQTT连接需要用到的Topic。

通过获取到的数据进行处理得到连接参数。处理方式参考MQTT协议接入。

此例中连接数据为：

参数名	参数值
服务器IP	117.50.16.141
服务器端口	1883
客户端ID	dev:fb4327cdcfe5432e82ad217572bd8ebd:542109155100005

第四步 获取模拟器

点击"在线调试"中的"下载设备模拟器"按钮即可获得设备模拟器。

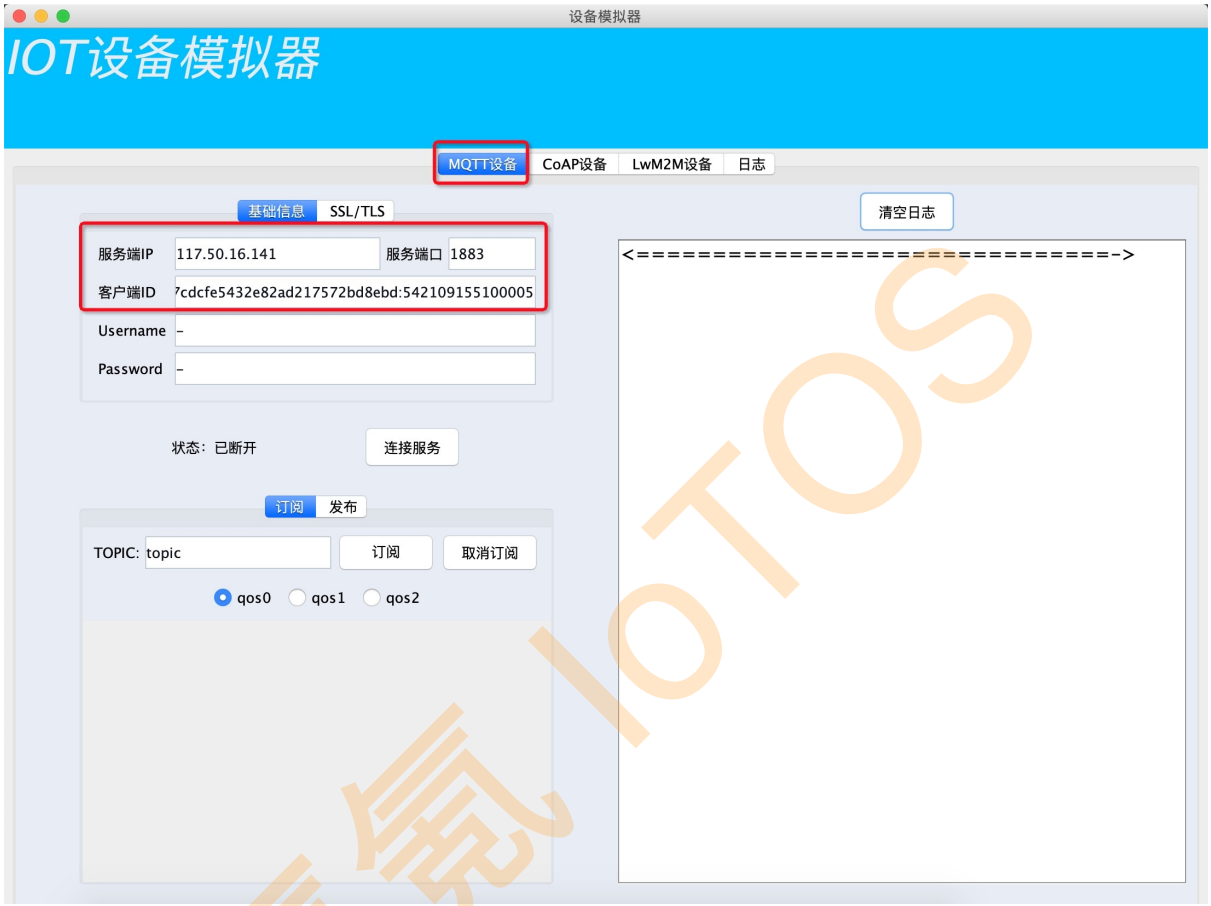
此处使用的是IoT OS提供的模拟器，也可以使用第三方模拟器 `MQTT.fx` 来进行模拟连接。若使用 `MQTT.fx` ,则可以参考使用MQTT.fx接入进行模拟。

【注意】：

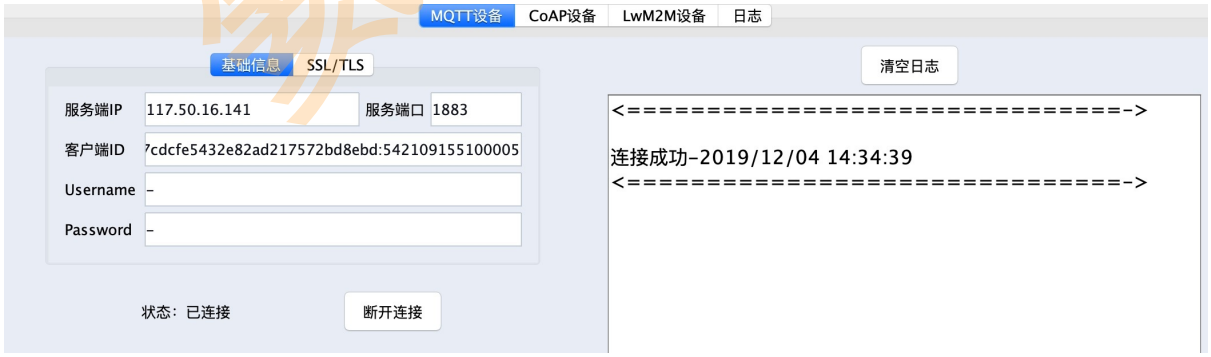
```
IoT OS提供的模拟器需要运行在windows操作系统下，解压后运行"windows启动器.bat"即可。
```

第五步 设备模拟连接

按照第三步中获取的连接信息填写到设备模拟器中。

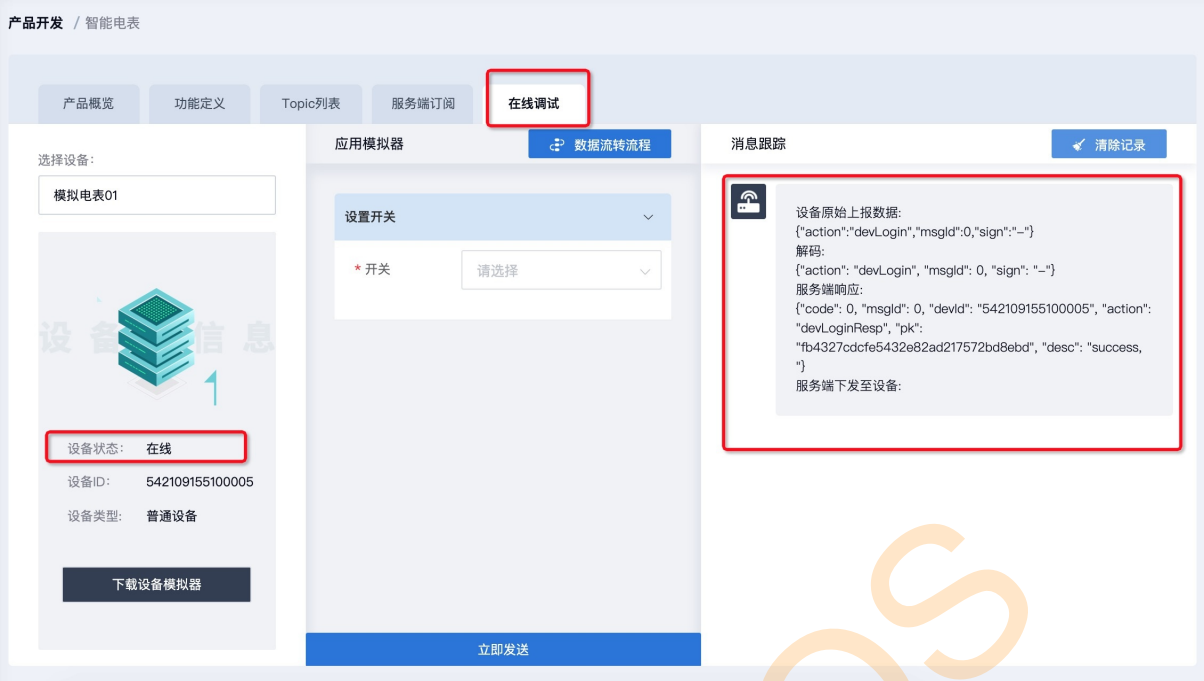


此例中创建的产品不需要设备登录验证，所以不需要写Username和Password。



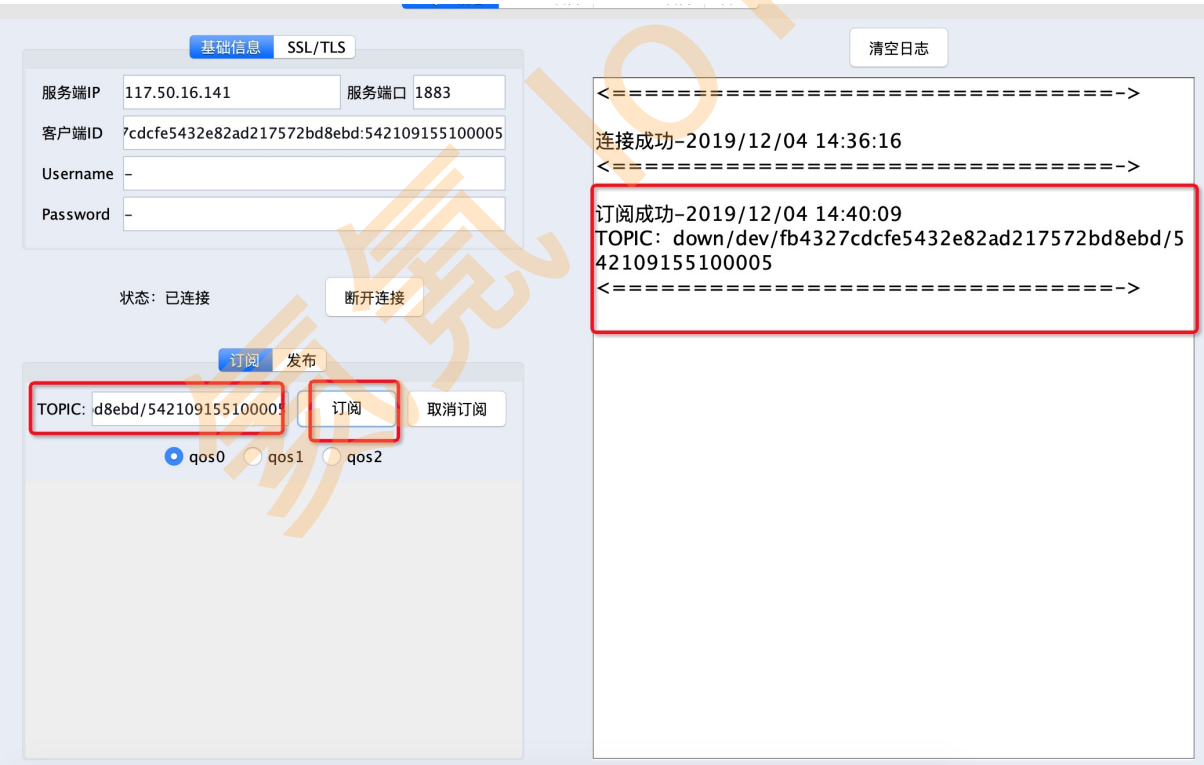
如图所示设备连接成功。

平台上，进入产品开发，点击创建的产品，点击"在线调试"，选择正确的设备。如图可以看到设备连接成功，右边消息跟踪获得了设备连接的数据。



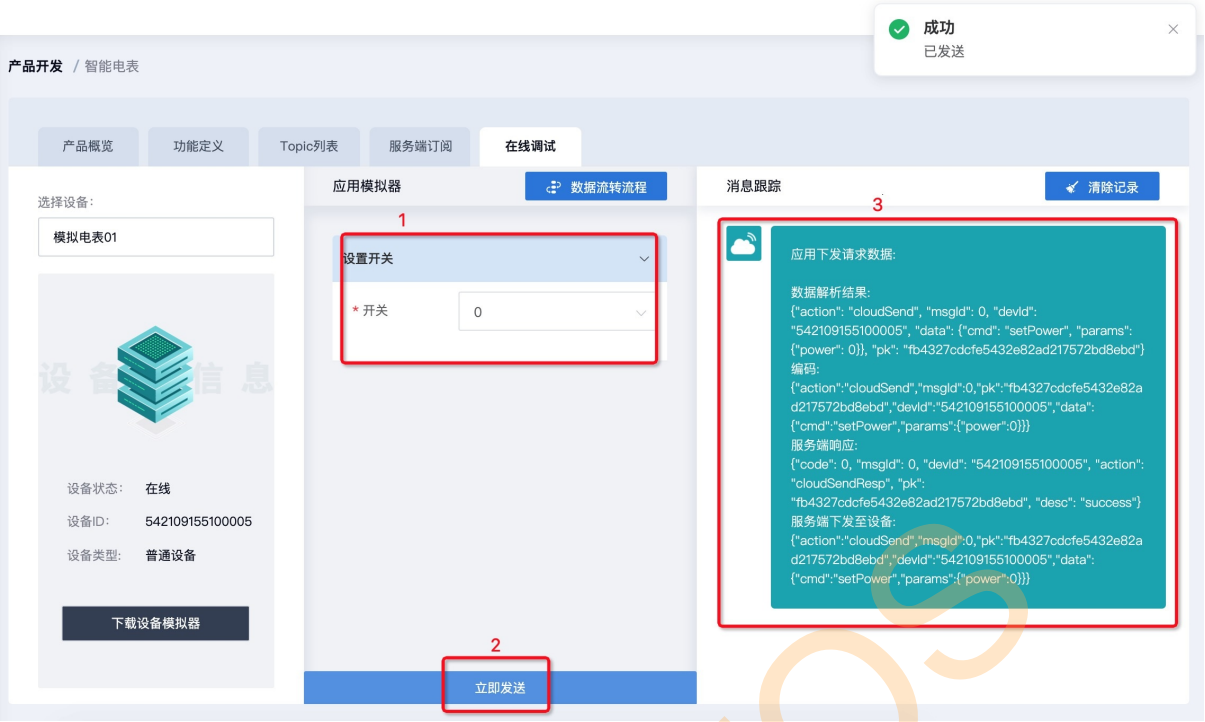
第六步 命令下发

为接收到平台下发的命令，必须先在模拟器订阅下发命令的Topic，此例的Topic在第三步中获取到。



如图将Topic输入后点击"订阅"，日志显示订阅成功。

回到平台-"在线调试"



【1】：点击"设置开关"中的枚举值选择，如图选择0。（具体功能由创建产品时定义）

【2】：点击"立即发送"按钮发送命令。

【3】：平台显示下发日志。

此时在模拟器也可以顺利收到平台下发的命令。

如图所示："power"为创建产品时创建的参数，数值为0。

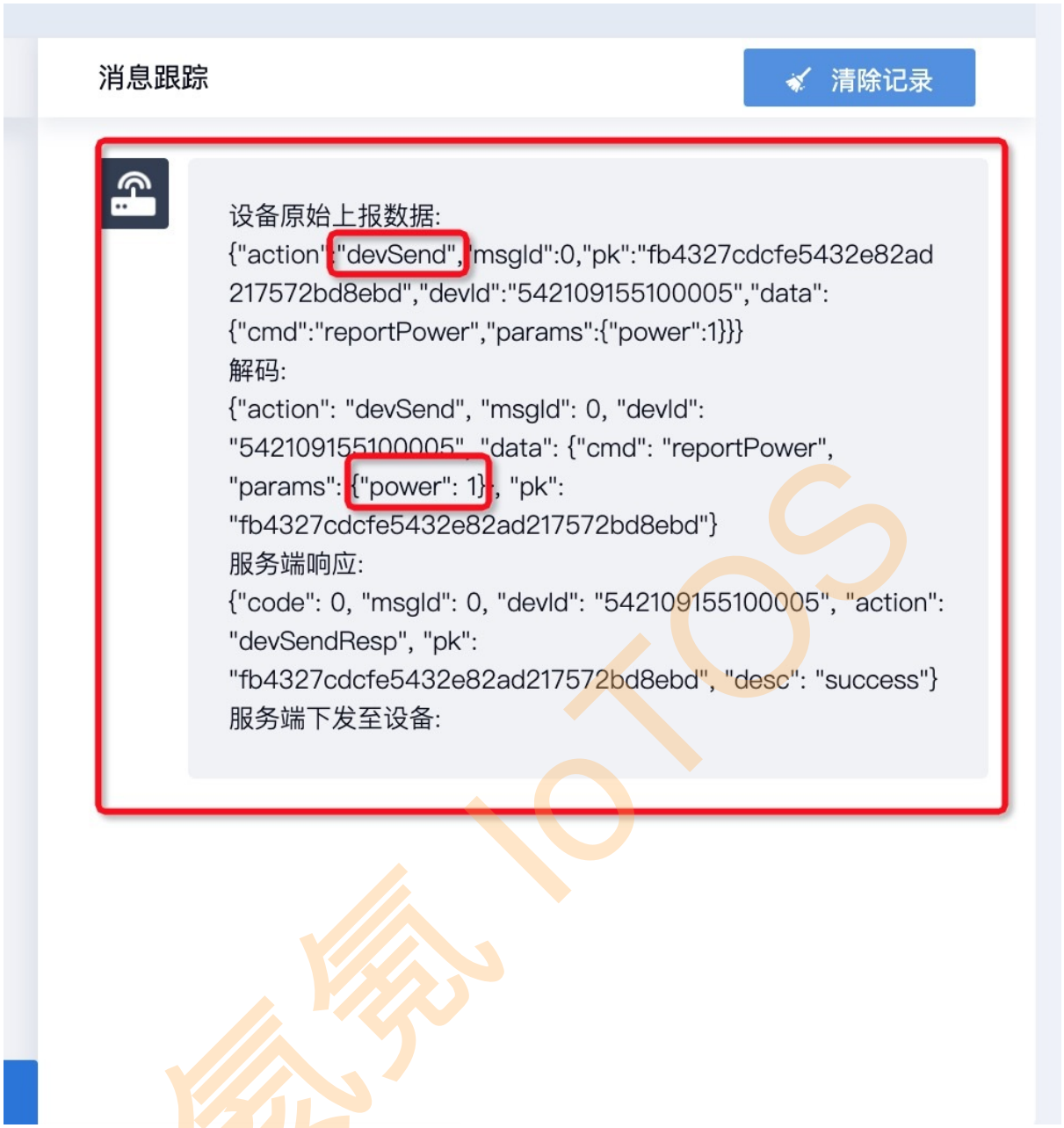
第七步 命令上报

使用模拟器，对平台进行数据上报。切换到"发布"，填入第三步获取用以发布的Topic，以KLink格式写完数据后，点击"发布"。

【1】：模拟器打印上报数据日志。

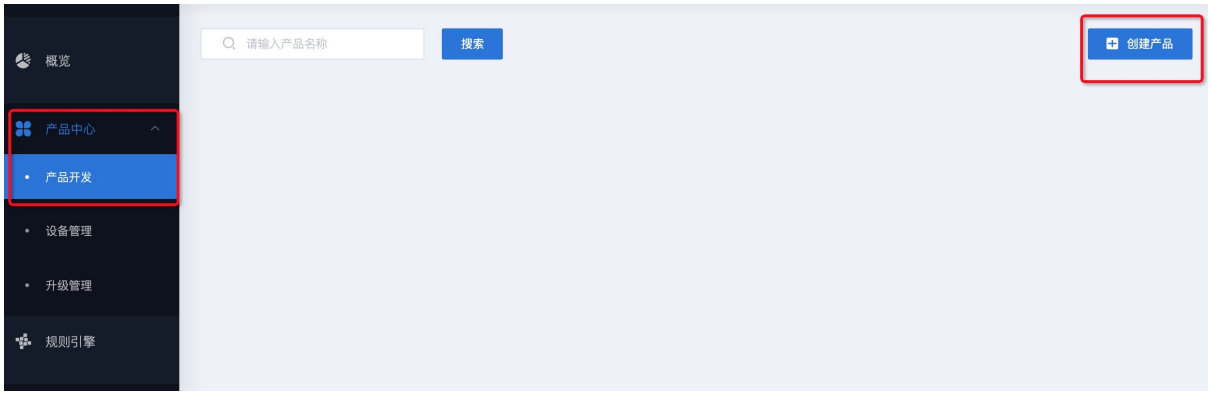
【2】：平台下发设备上报回复命令。

回到平台-"在线调试"可以看到平台收到了来自设备上报的数据。

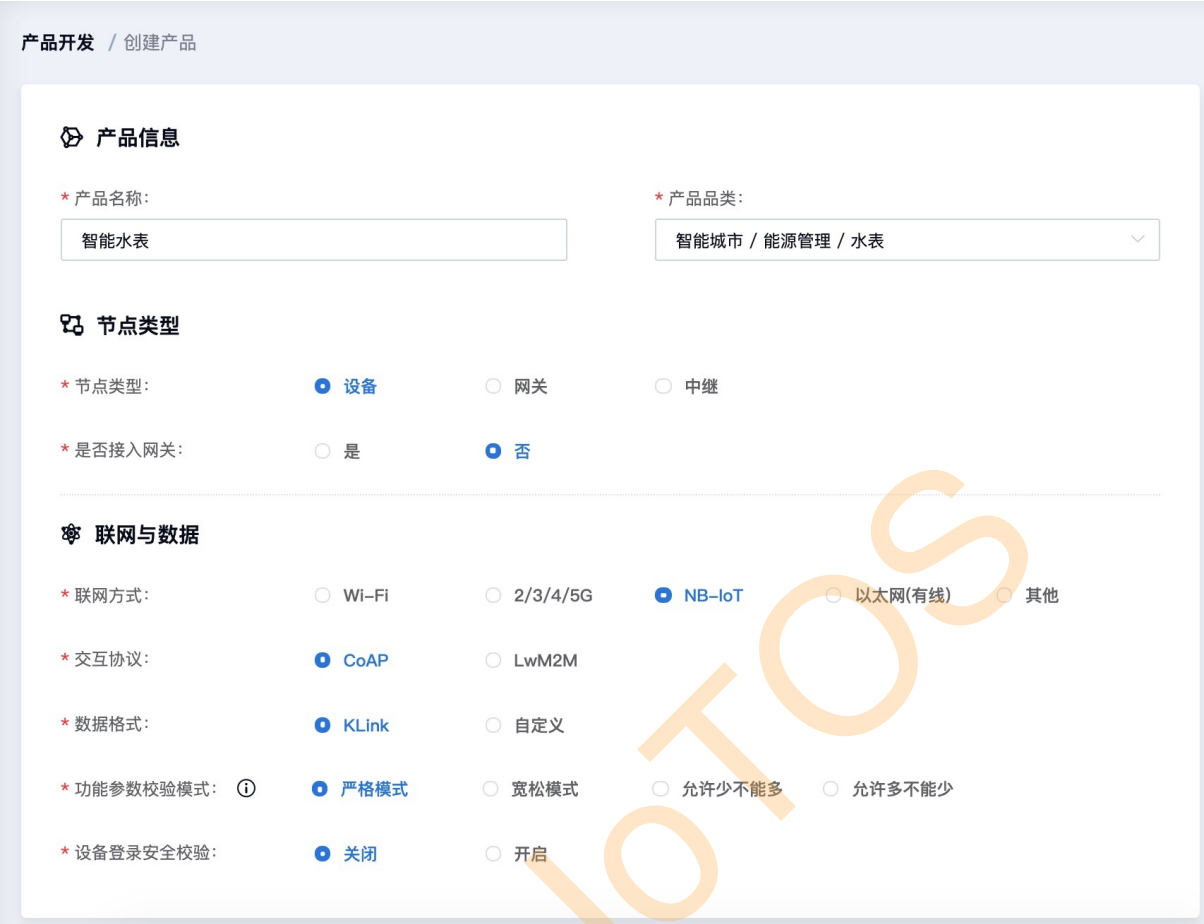


CoAP设备接入快速体验

第一步 创建产品



登录平台，进入"产品中心"-"产品开发"，点击右上角"创建产品"按钮。



如图所示书写相关信息，配置相关详情可参考创建产品。

点击右边"创建"按钮完成创建。

此时回到"产品开发"可以看到刚刚创建的产品卡片。如图所示。

点击产品卡片，进入产品概览。点击"功能定义"，对产品功能进行定义。

点击"自定义参数"-"创建参数"，如上图编写参数。

再点击"命令"-"创建命令"，分别创建上报命令和下发命令。

上报命令：


修改命令
* 命令名称:
上报开关
* 标识符:
reportPower
* 命令类型:
上报帧
待选参数
0/0
请输入参数名称
无数据
已选参数
0/1
请输入参数名称
开关
取 消
确 定

下发命令：

第二步 创建设备

完成产品创建后，进入"产品中心"-"设备管理"，点击右上角"创建设备"按钮。

如图所示填写设备信息。其中"所属产品为第一步中创建的产品"。 "设备ID"则需要为唯一标识。

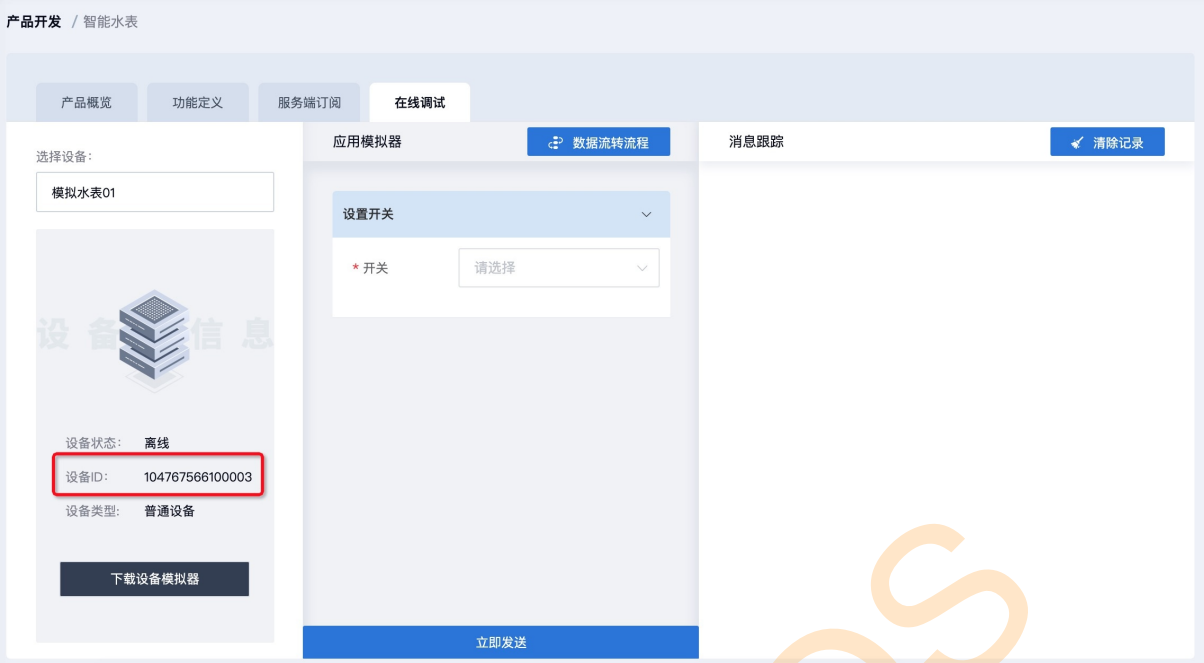
点击"确定"按钮完成设备创建。

第三步 获取连接信息

完成设备创建后，回到"产品中心"-"产品开发"中，点击第一步创建的产品卡片。进入产品概览，可以获取响应的连接信息。

其中主要使用到的参数是"设备接入信息"。 此例中用到的是CoAP接入方式：117.50.16.141:15683

再进入"在线调试"选择再第二步创建的设备，可以看到设备的唯一标识。



此例中的设备标识为：104767566100003

第四步 获取模拟器

点击"在线调试"中的"下载设备模拟器"按钮即可获得设备模拟器。

【注意】:

```
IoT OS提供的模拟器需要运行在windows操作系统下，解压后运行"windows启动器.bat"即可。
```

第五步 设备模拟连接

按照第三步中获取的连接信息填写到设备模拟器中。

如图选择"CoAP设备"，填写完连接信息后，点击"注册设备"。

如图所示设备注册成功。

第六步 命令下发

回到平台-"在线调试"

【1】：点击"设置开关"中的枚举值选择，如图选择0。（具体功能由创建产品时定义）

【2】：点击"立即发送"按钮发送命令。

【3】：平台显示下发日志。

此时在模拟器也可以顺利收到平台下发的命令。

如图所示："power"为创建产品时创建的参数，数值为0。

第七步 命令上报

使用模拟器，对平台进行数据上报。

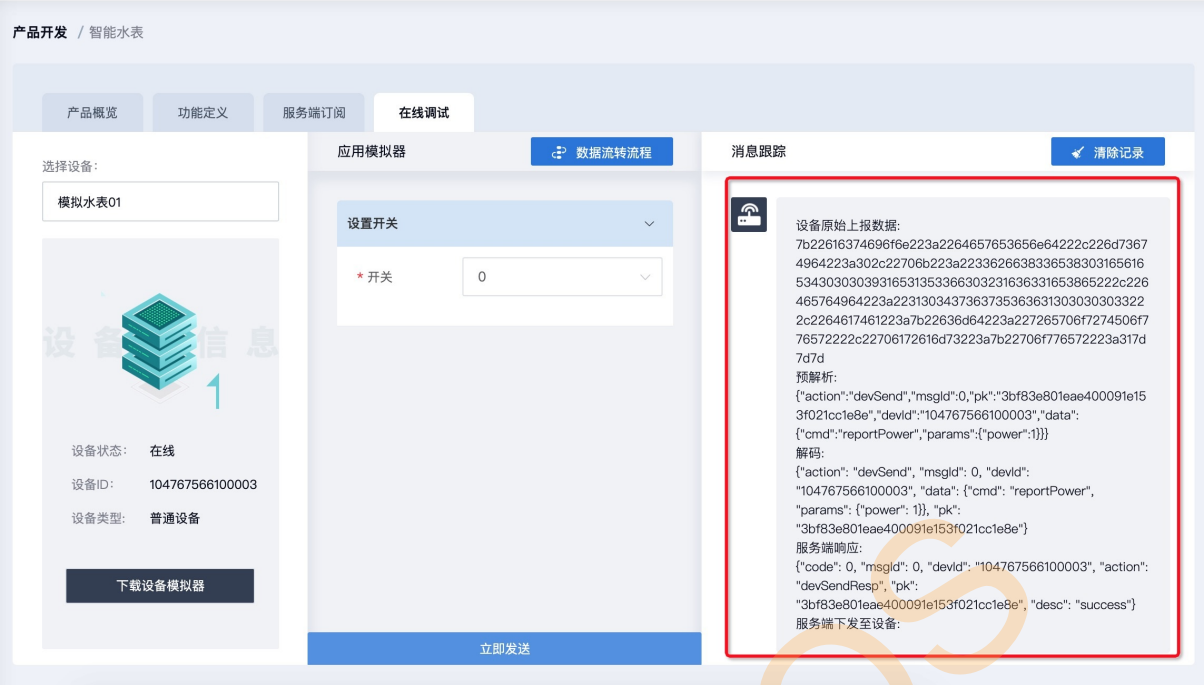
【1】：使用KLink格式进行设备上报命令。

【2】：点击"数据上报"按钮进行上报。

【3】：模拟器打印上报数据日志。

【4】：平台下发设备上报回复命令。

回到平台-"在线调试"可以看到平台收到了来自设备上报的数据。


产品开发 / 智能水表
产品概览
功能定义
服务端订阅
在线调试
选择设备:
模拟水表01
设备状态: 在线
设备ID: 104767566100003
设备类型: 普通设备
下载设备模拟器
应用模拟器
数据流转流程
设置开关
* 开关
0
立即发送
消息跟踪
清除记录
设备原始上报数据:
7b22616374696f6e223a2264657653656e64222c226d7367
4964223a302c22706b223a223362663833653830316561 6
534303030393165313533663032316363316538652 22c226
465764964223a223130343736373536363130303030322
2c2264617461223a7b22636d64223a227265706f7274506f7
76572222c22706172616d73223a7b22706f776572223a317d
7d7d
预解析:
{"action":"devSend","msgId":0,"pk":"3bf83e801eae400091e15
3f021cc1e8e","devId":"104767566100003","data":
{"cmd":"reportPower","params":{"power":1}}}
解码:
{"action": "devSend", "msgId": 0, "devId":
"104767566100003", "data": {"cmd": "reportPower",
"params": {"power": 1}}, "pk":
"3bf83e801eae400091e153f021cc1e8e"}
服务端响应:
{"code": 0, "msgId": 0, "devId": "104767566100003", "action":
"devSendResp", "pk":
"3bf83e801eae400091e153f021cc1e8e", "desc": "success"}
服务端下发至设备:

- 使用Postman模拟应用接入
  - 前提条件
  - 计算鉴权token
  - 获取应用接入信息
  - 使用Postman模拟查看设备详情

使用Postman模拟应用接入

本平台向北向开发者开放了相关接口，开发者可以通过接口文档进行相关开发。

本文通过Postman模拟应用接入平台。

前提条件

- 完成平台用户注册
- 完成用户实名认证
- 完成产品创建
- 完成设备创建

此处产品创建参考模拟器全流程中创建的产品。

登录平台，进入"产品中心"-"产品开发"，点击右上角"创建产品"按钮。

如图所示书写相关信息，此配置为不需要进行校验的普通设备。配置相关详情可参考创建产品。

点击右边"创建"按钮完成创建。

此时回到"产品开发"可以看到刚刚创建的产品卡片。如图所示。

完成产品创建后，进入"产品中心"-"设备管理"，点击右上角"创建设备"按钮。

如图所示填写设备信息。其中"所属产品为第一步中创建的产品"。 "设备ID"则需要为唯一标识。

点击"确定"按钮完成设备创建。

计算鉴权token

遵循安全鉴权方式计算token

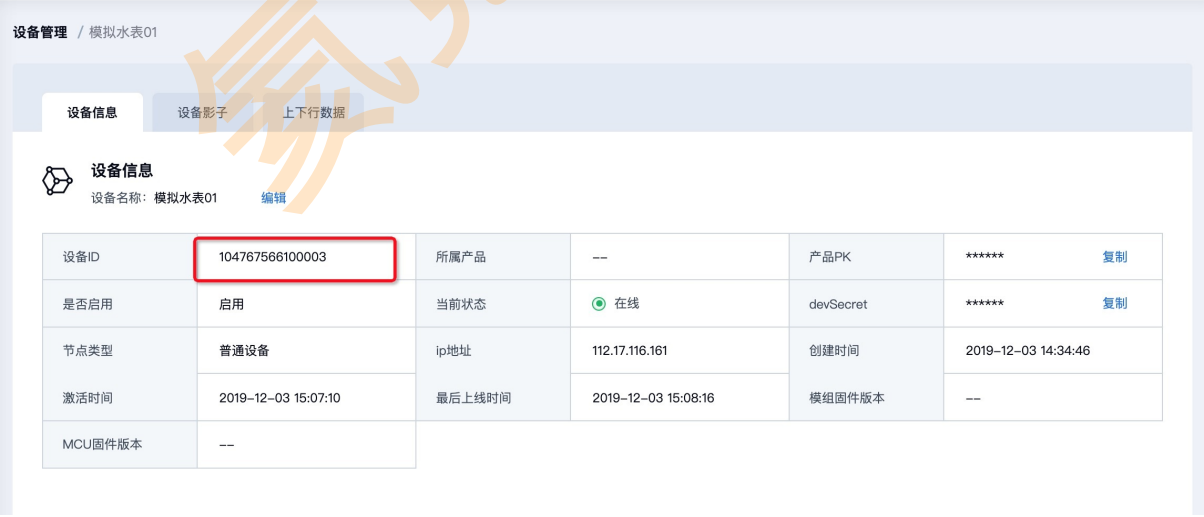
token获取方式参考使用AccessKey计算token。

获取应用接入信息

点击创建的产品卡片，进入"产品概览"。

图中应用接入方式显示了应用接入IP和端口。

点击刚创建的设备-"查看"。



使用Postman模拟查看设备详情

首先获取"查看设备详情"接口信息。点击查询设备详情接口。

在"产品开发"-"产品概览"中获得产品pk。

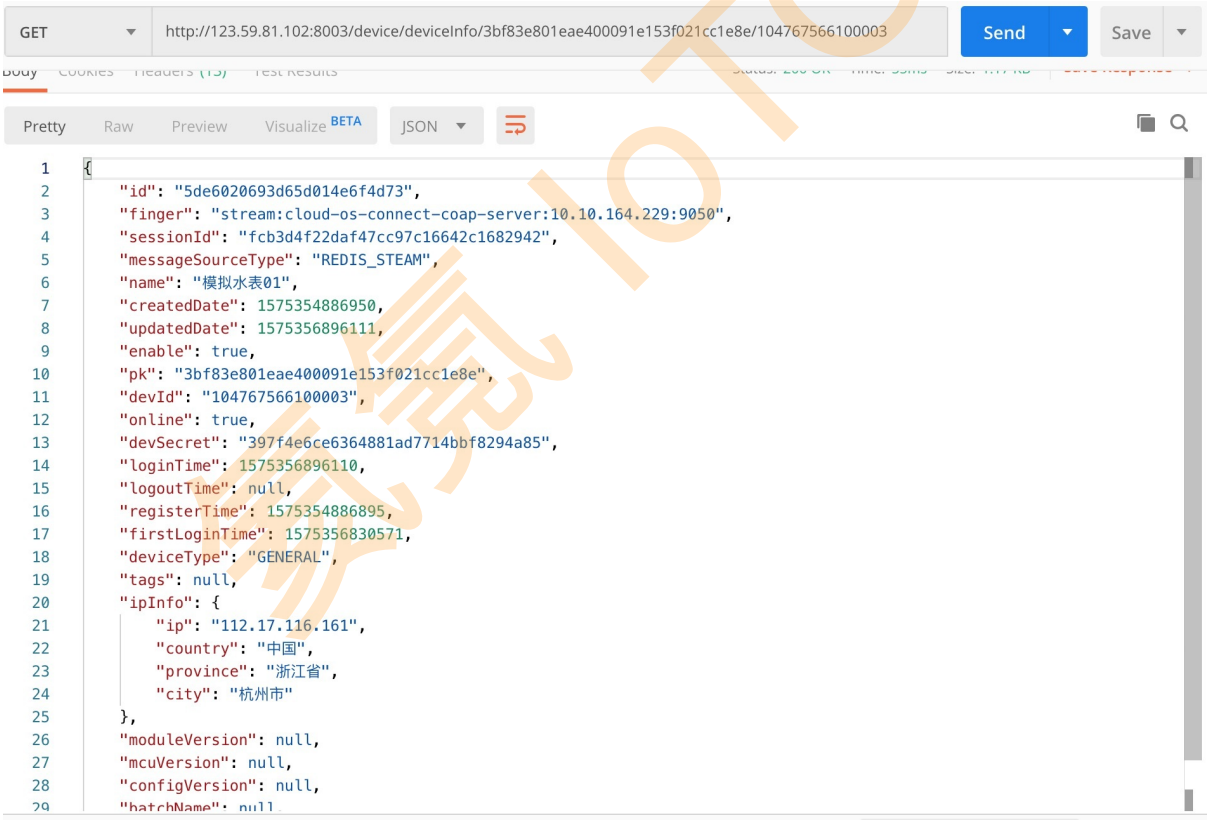
此例中产品pk:3bf83e801eae400091e153f021cc1e8e

在"设备管理"-"设备信息"中获得设备ID。

此例中设备ID为：104767566100003

在Postman中填写相应的接口和连接信息，如图所示

发送请求后获得到返回数值，如图所示

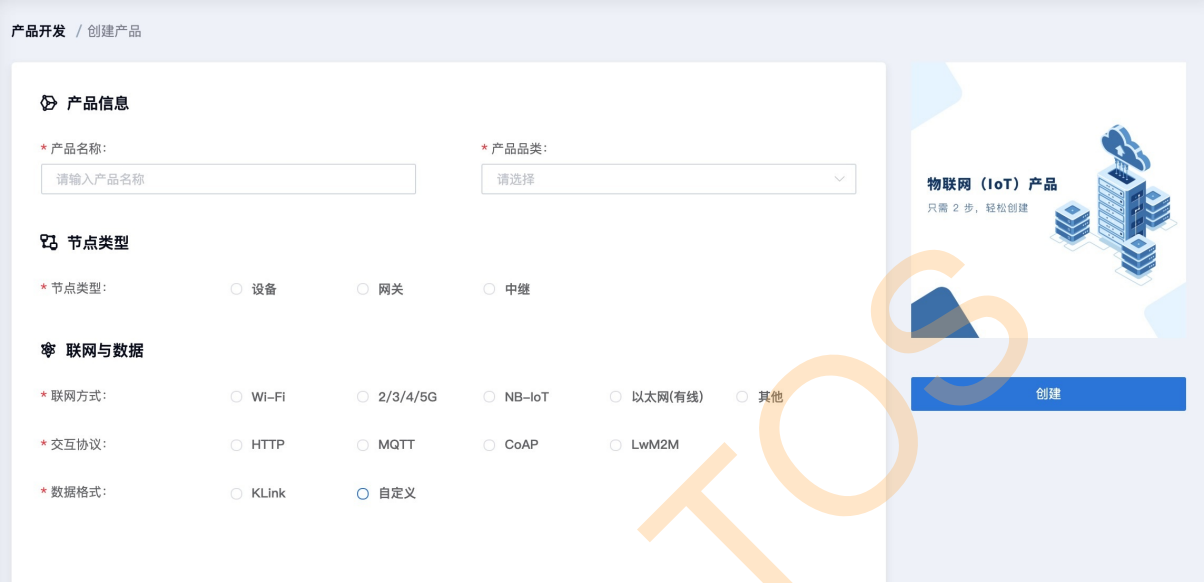



```
{
    "id": "5de6020693d65d014e6f4d73",
    "finger": "stream:cloud-os-connect-coap-server:10.10.164.229:9050",
    "sessionId": "fcb3d4f22daf47cc97c16642c1682942",
    "messageSourceType": "REDIS_STEAM",
    "name": "模拟水表01",
    "createdDate": 1575354886950,
    "updatedDate": 1575356896111,
    "enable": true,
    "pk": "3bf83e801eae400091e153f021cc1e8e",
    "devId": "104767566100003",
    "online": true,
    "devSecret": "397f4e6ce6364881ad7714bbf8294a85",
    "loginTime": 1575356896110,
    "logoutTime": null,
    "registerTime": 1575354886895,
    "firstLoginTime": 1575356830571,
    "deviceType": "GENERAL",
    "tags": null,
    "ipInfo": {
        "ip": "112.17.116.161",
        "country": "中国",
        "province": "浙江省",
        "city": "杭州市"
    },
    "moduleVersion": null,
    "mcuVersion": null,
    "configVersion": null,
    "batchName": null,
```

- 创建产品

创建产品

登录IoT OS后，在左侧导航栏选择“产品开发”，点击“创建产品”按钮，可新增产品，页面如下图所示，产品新增成功后在产品列表中可查看新创建的产品。



字段说明

产品名称：该产品在后台中的显示名称，支持中英文字符、数字；

产品品类：产品所属品类，在后台提供的品类中进行选择；

IoT OS支持的节点类型如下清单：

节点类型	是否接入网关	说明
设备	否	普通设备，可直连物联网平台，不能挂载子设备，也不能作为子设备挂载到网关下。
	是	终端子设备，不能直接连接物联网平台，而是通过网关设备接入物联网平台，不能再挂载终端子设备或中继设备。
网关	/	可以挂载终端子设备或中继设备的直连设备。
中继	/	不能直接连接物联网平台，而是通过网关设备接入物联网平台的设备，可以挂载终端子设备。

IoT OS支持的联网与数据格式如下清单：

联网方式	可选交互协议	可选数据格式
Wi-Fi、2/3/4/5G、以太网（有线）、其他	MQTT	KLink
		自定义
	HTTP	KLink
NB-IoT	CoAP	KLink
		自定义
		KLink

	LwM2M	自定义

IoT OS支持的数据格式包括：

KLink格式：物联网平台为开发者提供的设备与云端之间数据交换的标准协议，采用JSON格式。

自定义格式：自定义的数据格式，选择该格式后需在该产品的“数据解析”页面中提交数据解析脚本，将上行的自定义数据转换成标准KLink格式，同时将下行的KLink数据解析为设备自定义的格式，设备才能与云端进行通信。

设备登录注册校验：

说明：√表示可进行配置，×表示禁止或默认关闭

	普通设备	终端子设备	网关设备	中继设备
设备登录安全校验	√	×	√	×
设备动态注册	×	√	×	√
设备动态注册安全校验	×	√	×	√
远程配置	×	×	√	×
功能参数校验	√	√	√	√

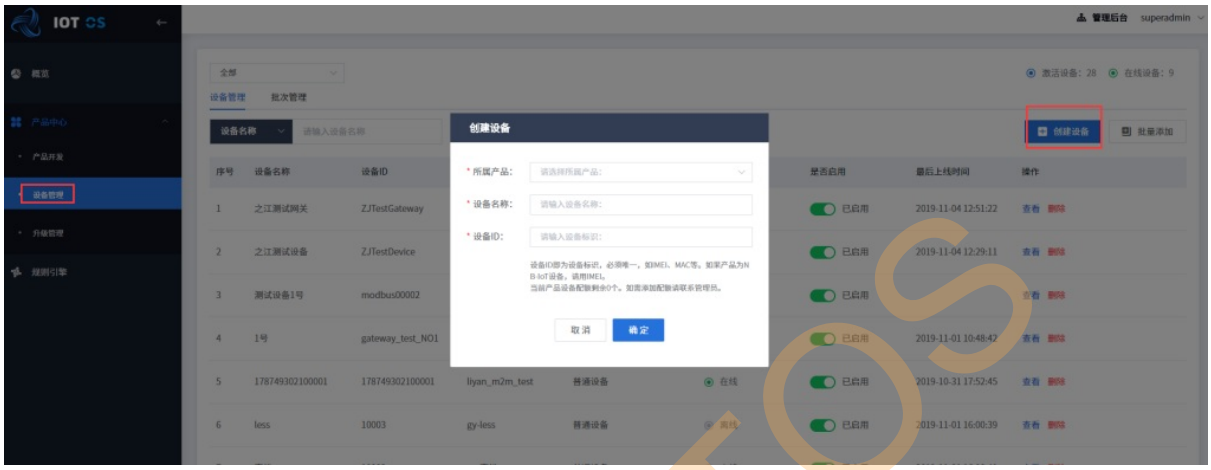
校验内容	前提条件		校验类型	备注	说明
	节点类型	数据格式			
功能参数校验模式	全部类型	KLink	严格模式	设备上报帧中的参数需要完全匹配功能定义命令中的参数列表，参数不能多也不能少。	数据格式为自定义时，功能参数校验默认为严格模式。
			宽松模式	设备上报帧中的参数不完全匹配功能定义命令中的参数列表，参数可以多也可以少。	
			允许少不能多	设备上报帧中的参数少于功能定义命令中的参数列表，但是不能多。	
			允许多不能少	设备上报帧中的参数多于功能定义命令中的参数列表，但是不能少。	
设备登录安全校验	普通设备或者网关设备	KLink	开启	设备与云端进行通信时，云端先校验设备登录信息，校验不通过则拒绝连接。	其他情形下的产品，默认为设备登录安全不校验。
			关闭	设备与云端进行通信时，不校验设备登录信息。	
设备动	接入网关的终端子设备		不允许	设备ID必须先在平台添加，才能连接云端。	

态注册	或者中继设备		允许	设备ID不必先从平台添加好，可通过网关发送register进行动态注册后，再与云端上报或者接收下发数据。	其他情形下的产品，设备ID必须先在平台添加，才能与云端上报或者接收下发数据。
设备动态注册校验	接入网关的终端子设备或者中继设备	KLink	开启	设备在云端进行注册时，云端先校验设备注册的信息，校验不通过则拒绝连接。	
			关闭	设备在云端进行注册时，不校验设备注册信息。	
远程配置	网关	KLink	开启	支持对该产品下的设备进行远程配置更新。	其他类型设备不允许远程更新配置。
			关闭	不支持对该产品下的设备进行远程配置更新。	

- 创建单个设备

创建单个设备

产品创建成功后，用户需要先将设备信息添加到平台，进行设备注册，设备才能正常连接IoT OS。IoT OS支持创建单个设备，也支持批量创建设备。 登录IoT OS后，在左侧导航栏中选择“设备管理”，在设备管理页面，单击“创建设备”，在创建设备对话框输入设备信息，点击确定即可完成添加。



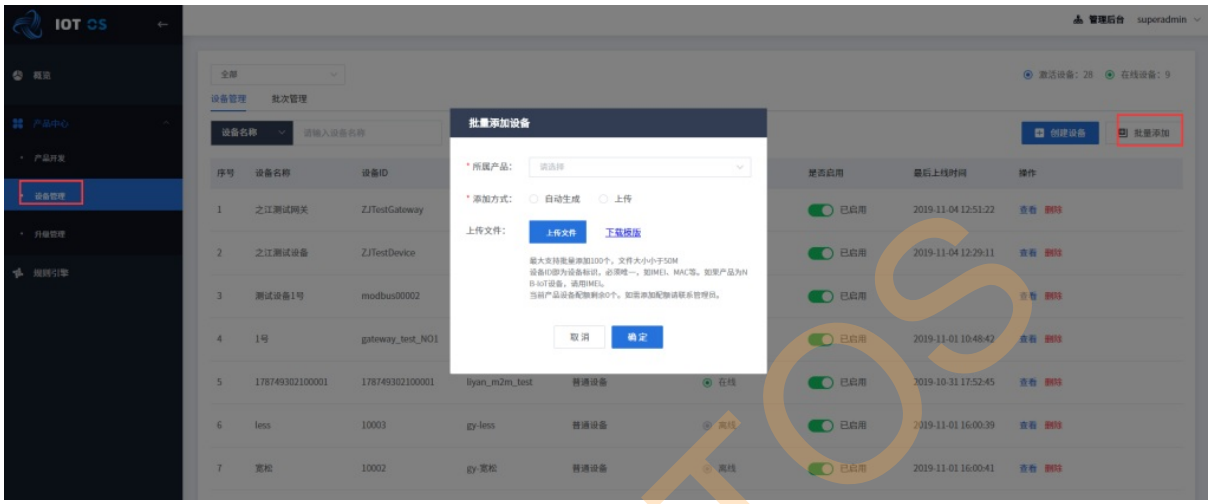
【说明】

1. 所属产品：设备所属产品，新创建的设备会继承该产品定义好的功能和特性；
2. 设备名称：支持2~32长度的字符；
3. 设备ID：设备唯一标识，NB-IoT设备在该处填写设备的IMEI；
4. 配额剩余数：所选产品下还可添加的设备个数。新建的产品默认最多允许添加10个设备。选择所属产品后，对话框下面的文案中会提示该产品还剩余的设备配额，若配额为0，该产品将不允许再添加设备，需要向平台管理员申请增加设备额度。

- 批量创建设备

批量创建设备

IoT OS支持通过导入CSV文件或服务端自动生成两种方式批量添加设备。登录IoT OS后，在左侧导航栏中选择“设备管理”，在设备管理页面，单击“批量添加”，在批量添加设备的对话框中，选择批量生成的方式，填入设备信息，点击确定，可批量创建设备。



【说明】

1．所属产品：设备所属产品，新创建的设备会继承该产品定义好的功能和特性；
2．添加方式：
【自动生成】无需指定设备ID，填写设备数量后，系统自动生成设备ID；
【上传】需要指定要添加的所有设备ID。单击“下载模板”，下载表格模板，在模板中填写设备ID，然后将填写好的表格上传到控制台。

3．批量添加额度：单次批量添加，最大允许添加100个设备，超过100个设备，需要分批次添加。

【注意】

若所选产品剩余的设备额度小于要添加的设备数量，只能添加剩余额度数量的设备，超出的部分会添加失败。

- 消息通信Topic

消息通信Topic

物联网平台中，若交互协议为MQTT的产品，服务端跟设备端是通过Topic来实现消息通信的。Topic是针对设备的概念，Topic类是针对产品的概念。产品的Topic类会自动映射到产品下的所有设备中，生成用于消息通信的具体设备Topic。

- 产品Topic类

产品Topic类

Topic类是一类Topic的集合。产品创建成功后，会自动生成两个Topic类，在“产品开发”页面，单击交互协议为MQTT的产品卡片，单击“Topic列表”标签可查看生成的Topic。

【说明】

1. 生成的两个Topic类分别对应发布和订阅两个操作权限，格式固定。
2. 发布Topic：设备往该Topic发布消息，格式为up/dev/{productPK}/{devId}。
3. 订阅Topic：设备订阅该Topic获取消息，格式为down/dev/{productPK}/{devId}。
4. productPK为产品PK，每个产品唯一。

- 设备Topic类

设备Topic类

设备创建好后，设备所隶属的产品下的Topic类会自动映射到设备上。真正用于消息通信的是具体的设备Topic。用户可点击左侧边栏中的“设备管理”，单击设备列表中设备的“查看”按钮，点击“Topic列表”，可查看和复制具体的设备Topic，用于设备消息通信。

【说明】

设备对应的Topic是从产品Topic类映射出来，根据设备ID动态创建的。每个设备的Topic唯一，只能被该设备用于消息通信。

- 物模型
  - 物模型简介
  - 物模型定义

物模型

物模型简介

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

物模型定义

在设备接入在线调试之前，需要在平台"产品开发"-选择产品-"功能定义"中建立符合用户要求的物模型。只有定义好了物模型的各种参数才能顺利完成设备接入在线调试。 物模型定义详细操作可见添加协议参数和添加协议命令。

- 数据解析
  - 数据解析脚本

数据解析

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

数据解析脚本

用户创建产品时会选择产品下设备的种类，包括设备的传输数据格式，当用户定义产品时选择的数据格式为KLink时，用户只需保证设备传输数据符合KLink格式即可。KLink格式详情可见KLink。

若用户定义产品时选择的是自定义格式，则用户需要在"产品开发"-选择产品-"数据解析"中填写数据解析脚本。数据解析脚本编写详情可见数据解析。

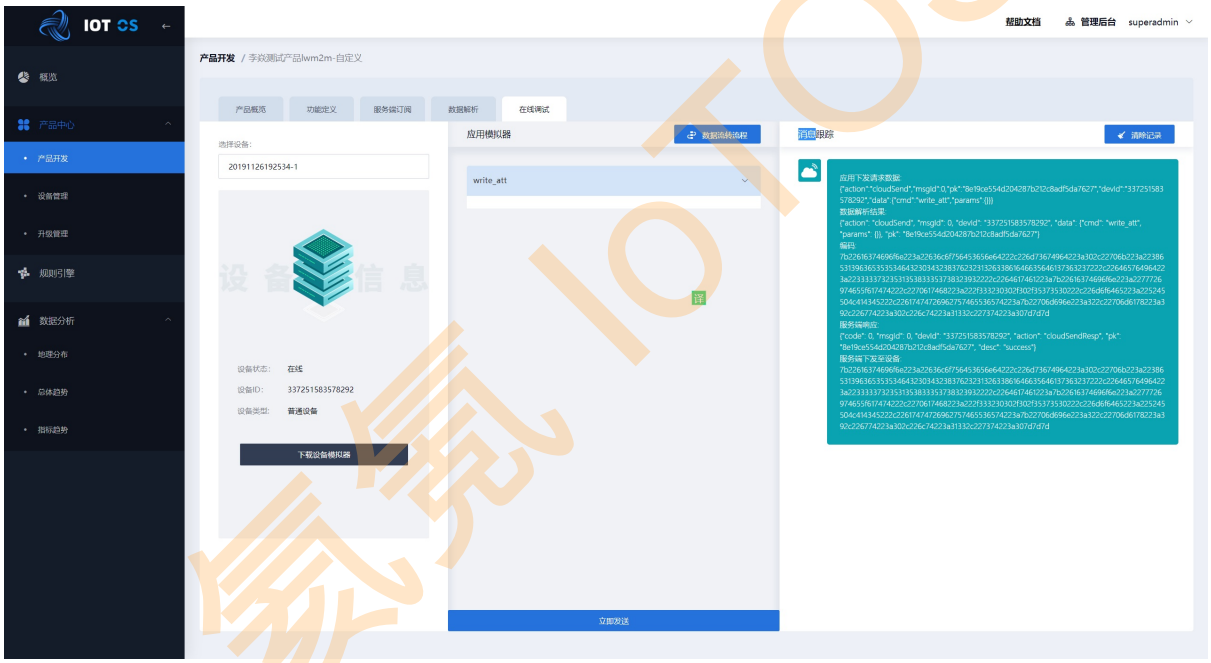
- 在线调试

在线调试

产品、设备都创建好后，可通过平台提供的模拟工具进行设备调试，测试上下行通信数据，确定产品、协议、解析脚本功能是否正确完整。

设备调试步骤：

1. 登录IoT OS，点击要调试的产品卡片，选择“在线调试”标签；
2. 点击“下载设备模拟器”，下载设备模拟器；
3. 根据产品交互协议，在模拟器中选择要模拟的设备类型；
4. 根据模拟器操作说明，填写好设备信息，连接服务；
5. 在平台“在线调试”页面，选择调试设备；
6. 在应用模拟器中选择要下发的命令，填写参数值；
7. 点击“立即发送”，进行下行通信测试；
8. 在消息跟踪中查看下行数据是否正确；
9. 在设备模拟器的上报数据区域，添加上报数据，并点击发送，进行上行通信测试；
10. 在消息跟踪中查看上下行数据是否正确。



【说明】

1. 在线调试，支持设备模拟器模拟的设备调试，也支持真实设备调试。
2. 调试的设备必须在线。

- MQTT协议接入
  - MQTT简介
  - 协议特点
  - 操作前提
  - 获取连接认证参数
  - 参数构造实例
  - 接入详情

MQTT协议接入

MQTT简介

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）协议是即时通信协议，是物联网的重要组成部分。

协议特点

MQTT协议是为大量计算能力有限，且工作在低带宽、不可靠的网络的远程传感器和控制设备通讯而设计的协议，它具有以下主要的几项特性：

```
1.使用发布/订阅消息模式，提供一对多的消息发布，解除应用程序耦合；

2.对负载内容屏蔽的消息传输；

3.使用 TCP/IP 提供网络连接；

4.小型传输，开销很小（固定长度的头部是 2 字节），协议交换最小化，以降低网络流量；

5.使用 Last Will 和 Testament 特性通知有关各方客户端异常中断的机制；

6.有三种消息发布服务质量：

    “至多一次”，消息至多发送一次。

    “至少一次”，确保消息到达，但消息重复可能会发生。

    “只有一次”，确保消息到达一次。
```

操作前提

```
1.已创建好产品和设备（产品管理、设备管理）
2.定义好产品模型（模型管理）
```

产品管理、设备管理、模型管理。

获取连接认证参数

设备通过mqtt协议连接云服务，必须认证通过后才能成功建立连接。

【注意】直连设备（网关设备和普通设备）才允许连接

1)建立连接前先从平台获取认证需要的参数

参数	说明	获取路径
devId	设备ID	用户添加设备时输入的“设备ID”，在“设备管理”中可查看
pk	产品PK	平台“设备管理”，根据设备ID找到设备，点击列表右侧的“查看”，复制“设备信息”中的“产品PK”。
devSecret	设备秘钥	平台“设备管理”，根据设备ID找到设备，点击列表右侧的“查看”，复制“设备信息”中的“devSecret”。

2）构造MQTT连接的参数

参数	说明	构造方式
clientId	客户端ID	dev:{pk}:{devId}
username	用户名	{hashMethod}:{random}
password	密码	hash(pk+devId+devSecret+random)，加密密钥：devSecret。

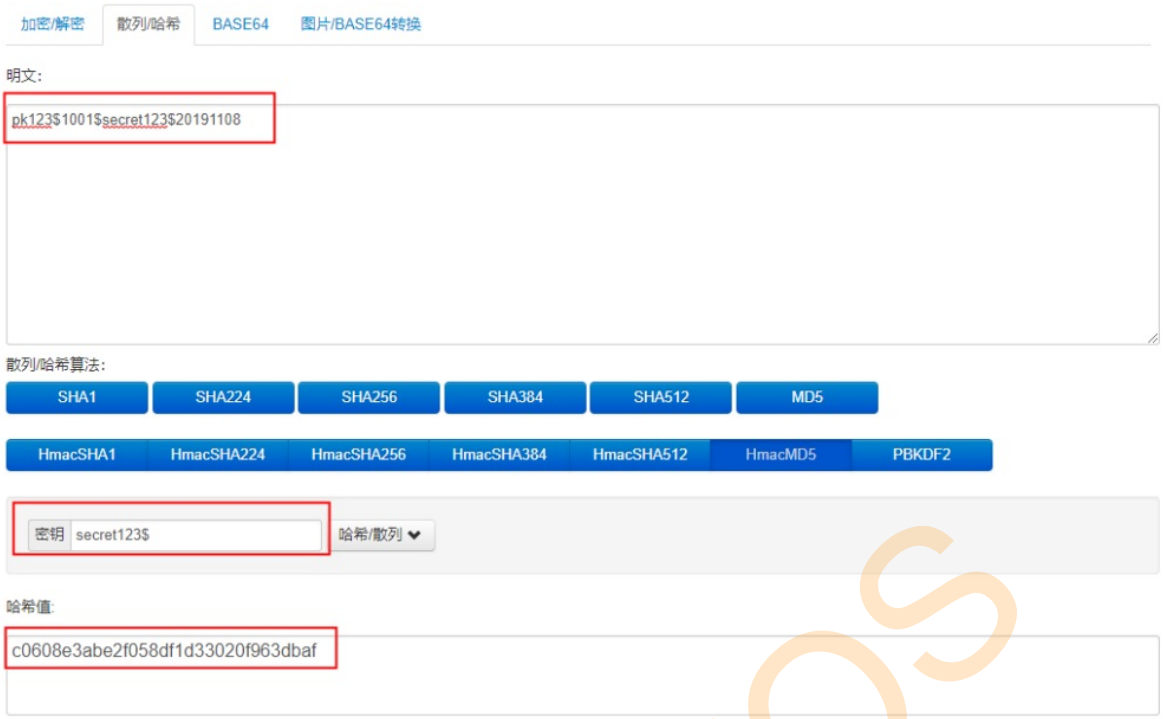
【说明】

（1）参数间使用“:”隔开。
（2）{hashMethod}支持：HmacMD5、HmacSHA1、HmacSHA256 和 HmacSHA512。
（3）password中加密使用的hashMethod、random需跟username中一致。

参数构造实例

Eg：pk=’pk123$’，devId=’1001$’，devSecret=’secret123$’，使用HashMD5方式进行加密，随机字符（random）为’20191108’。
clientId = dev:pk123$:1001$
username = HashMD5:20191108
password = c0608e3abe2f058df1d33020f963dbaf
password是通过HashMD5方法加密后得到。
要加密的字符串：pk123$1001$secret123$20191108，加密密钥：secret123$，加密后得到“c0608e3abe2f058df1d33020f963dbaf”。

（在线工具）



接入详情

使用MQTT.fx接入

使用SDK接入

- 使用MQTT.fx接入
  - 操作步骤：
    - 数据格式

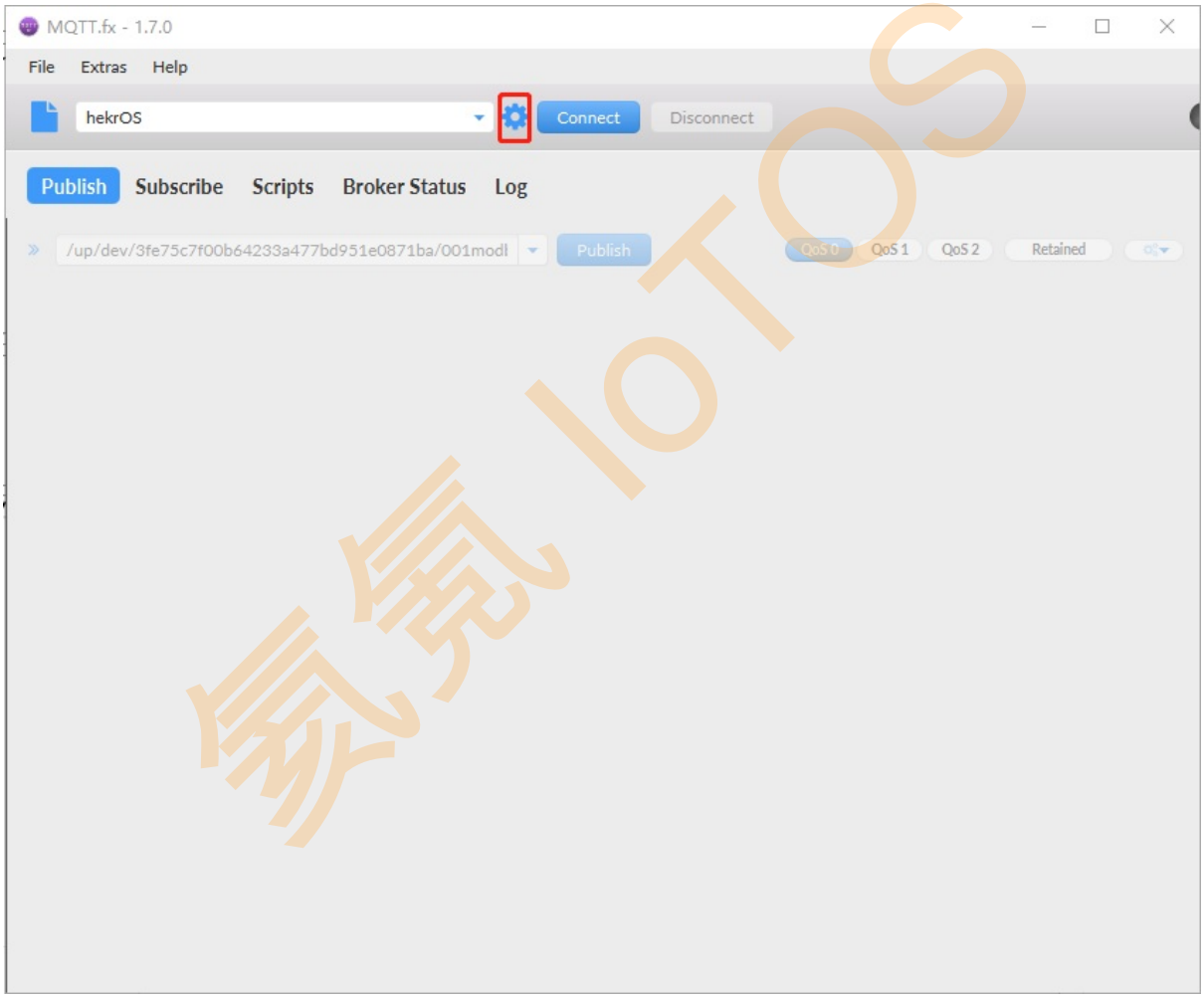
使用MQTT.fx接入

操作步骤：

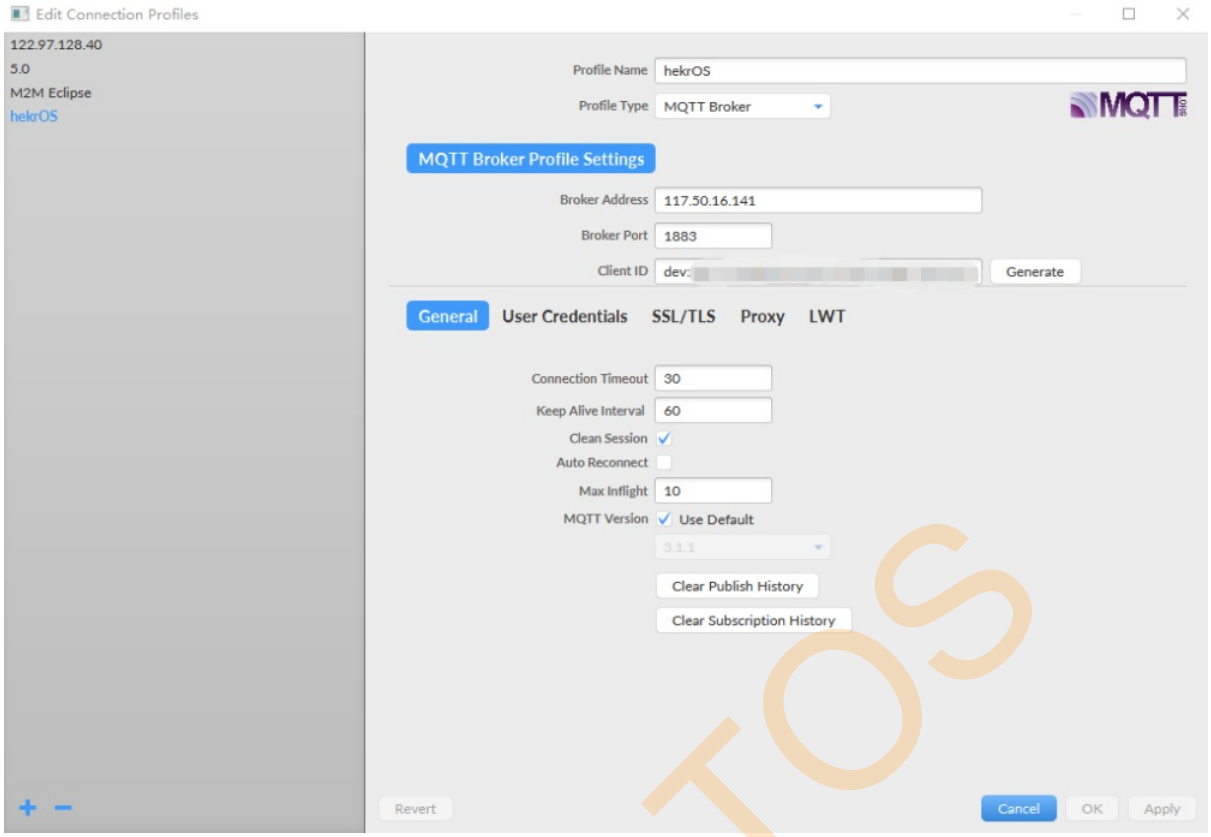
1.下载并安装MQTT.fx软件。

MQTT.fx下载地址

2.打开MQTT.fx软件，单击设置图标

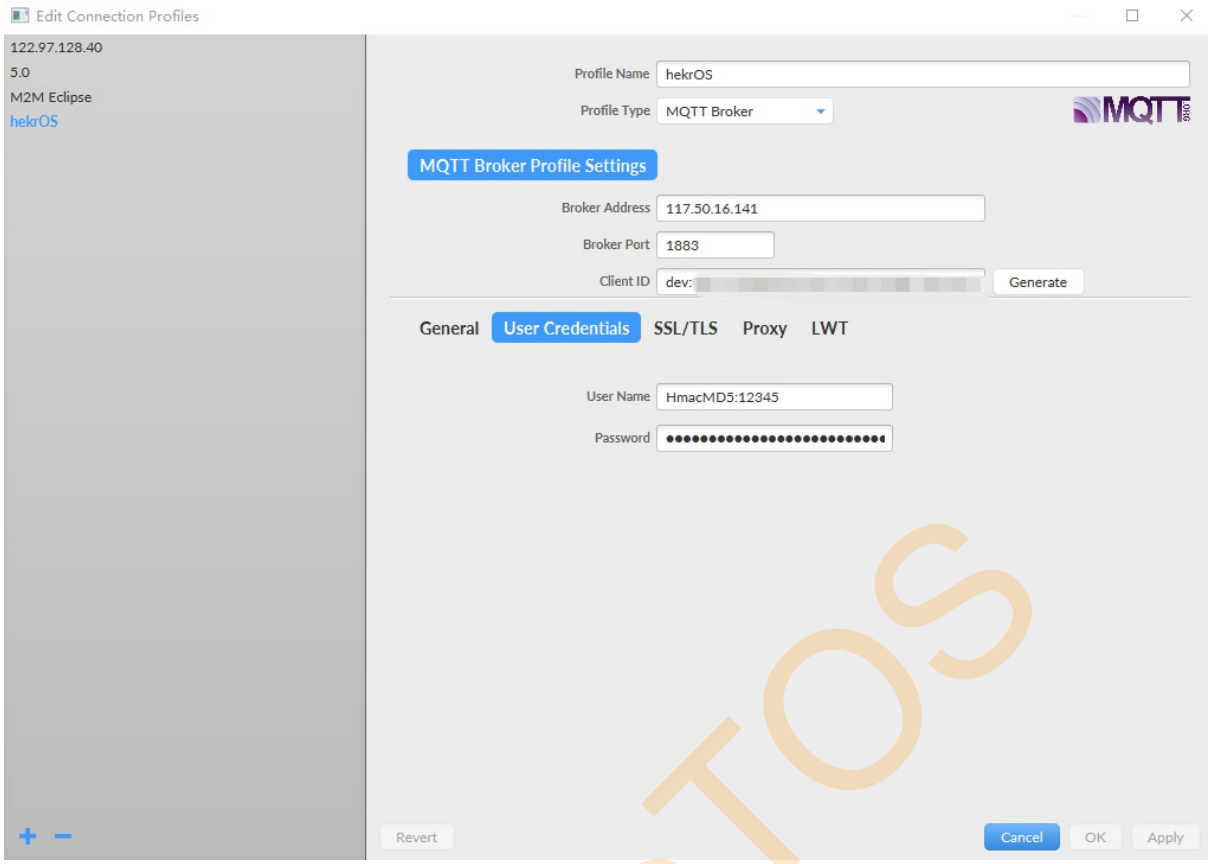


3.设置连接参数



参数	说明
Profile Name	输入您的自定义名称。
Profile Type	选择为MQTT Broker。
Broker Address	接入地址，在平台“产品开发”，点击产品卡片，复制“产品概览”中的接入地址。
Broker Port	设置为1883。
Client ID	填写mqttClientId，用于MQTT的底层协议报文。格式固定：dev:{pk}:{devId} 详情参考。
General	General栏目下的设置项可保持系统默认，也可以根据您的具体需求设置。

4.单击User Credentials，设置Username和Password

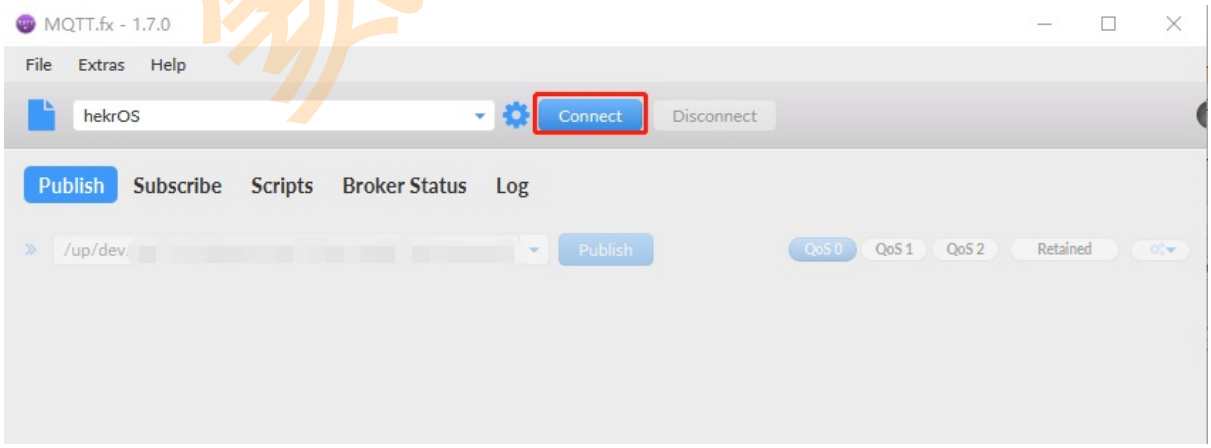


参数	说明
User Name	填写登录用户名，格式{hashMethod}:{random}
Password	填写登录密码，格式hash(pk+devId+devSecret+random)，加密密钥：devSecret。计算方式参考

【注意】：

只有在需要进行登录校验的设备才需要填写用户名和密码信息。

5.设置完成后，单击Connect进行连接



6.发布消息，测试上行通信

1）在MQTT.fx上，单击Publish
2）输入具有发布权限的Topic。设备发送到云端的 topic 有且只有一个，格式为 up/dev/{pk}/{devId}，这里的{pk}和{devId}要替换成设备

对应的产品PK和设备ID。
3）文本框中填写要发布的数据，IoT OS标准数据格式为KLink，格式如下：

```
{
    "action": "devSend",
    "msgId": 1,
    "pk": "xxxxxxxxxxxx",
    "devId": "xxxxxxxxxxx",
    "data": {
        "cmd": "reportPower",
        "params": {
            "power": 0,
            "hum": "one"
        }
    }
}
```

4）发布数据，单击“Publish”，即向这个Topic推送了一条消息。
5）订阅消息，单击“Subscribe”。
6）输入具有订阅权限的Topic，格式为 down/dev/{pk}/{devId}，这里的{pk}和{devId}要替换成设备对应的产品PK和设备ID。
7）填好订阅Topic后，单击Topic右侧的“Subscribe”，完成订阅。设备发布消息后，就可在订阅的页面右侧消息栏中查看云端回复的消息。

数据格式

产品创建时可以将产品格式设置为"自定义格式"或者"KLink"格式。

自定义格式

如果产品的数据格式为自定义格式，则需要用户在"产品开发"，"数据解析"中将符合自己需求的数据解析协议编写完成。操作参考。

数据解析协议编写完成后，用户可以在传输信息时（即如上述发布消息操作）用户发送自定义格式的数据即可。

KLink格式

如果产品选择的数据格式为KLink格式，则在传输信息时（即如上述发布消息操作）用户需要发送符合KLink协议的数据。接入协议简介。

【注意】除了消息格式必须符合KLink协议，使用的指令必须属于MQTT设备支持的指令。

MQTT设备支持指令表：

指令功能	发送形式	详情
设备上报数据	D => C	devSend
云端回复设备上报数据	C => D	devSendResp
云端给设备下发指令	C => D	cloudSend
设备回复云端下发指令	D => C	cloudSendResp
设备上报固件信息	D => C	reportFirmware
云端回应设备上报固件信息	C => D	reportFirmwareResp
云端给设备下发固件升级指令	C => D	devUpgrade
设备回应固件升级指令	D => C	devUpgradeResp
设备上报升级进度	D => C	devUpgradeProgress
动态注册设备	D => C	register

云端回复动态注册设备	C => D	registerResp
网关添加拓扑关系（子设备）	D => C	addTopo
云端回复网关添加拓扑关系（子设备）	C => D	addTopoResp
网关查询拓扑关系（子设备）	C => D	getTopo
云端回复网关查询拓扑关系（子设备）	D => C	getTopoResp
网关删除拓扑关系（子设备）	D => C	delTopo
云端回复网关删除子设备（子设备）	C => D	delTopoResp
子设备上线	D => C	devLogin
云端回复子设备上线	C => D	devLoginResp
子设备下线	D => C	devLogout
云端回复子设备下线	C => D	devLogoutResp

- 使用SDK接入

使用SDK接入

KLink MQTT基于MQTT协议之上进行数据格式定义，设备开发者可以参考KLink与MQTT协议开源实现Eclipse Paho自主开发。

准备工作： 在IoT OS后台创建MQTT协议登录不校验产品并创建设备。

- 1.用户可在github获取paho.mqtt.c开源实现，使用gcc编译客户端，操作如下：

```
git clone https://github.com/eclipse/paho.mqtt.c.git

cd paho.mqtt.c

make
```

- 2.进入samples文件夹编辑修改MQTTClient_publish.c文件（修改连接IP、clientid、topic见注释）

```
/*******************************************************************************
 * Copyright (c) 2012, 2017 IBM Corp.
 *
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * and Eclipse Distribution License v1.0 which accompany this distribution.
 *
 * The Eclipse Public License is available at
 *   http://www.eclipse.org/legal/epl-v10.html
 * and the Eclipse Distribution License is available at
 *   http://www.eclipse.org/org/documents/edl-v10.php.
 *
 * Contributors:
 *Ian Craggs - initial contribution
 *******************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "MQTTClient.h"

#define ADDRESS "test-mqtt.hekr.me:1883"//定义MQTT设备对接域名与端口
#define CLIENTID"dev:{pk}:{devId}"//定义客户端链接识别码，参数之间用：隔开，括号中填入平台上创建的产品PK和设备ID，括号为占位符，参数填入后请删除括号
#define TOPIC   "up/dev/{pk}/{devId}"//定义数据发布主题，括号中填入平台上创建的产品PK和设备ID，括号为占位符，参数填入后请删除括号
#define PAYLOAD "Hello World!"//定义上报内容
#define QOS 1
#define TIMEOUT 10000L

int main(int argc, char* argv[])
{
MQTTClient client;
MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
MQTTClient_message pubmsg = MQTTClient_message_initializer;
MQTTClient_deliveryToken token;
int rc;

MQTTClient_create(&client, ADDRESS, CLIENTID,
MQTTCLIENT_PERSISTENCE_NONE, NULL);
conn_opts.keepAliveInterval = 20;
conn_opts.cleansession = 1;

if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
```

```
{
printf("Failed to connect, return code %d\n", rc);
exit(EXIT_FAILURE);
}
pubmsg.payload = PAYLOAD;
pubmsg.payloadlen = (int)strlen(PAYLOAD);
pubmsg.qos = QOS;
pubmsg.retained = 0;
MQTTClient_publishMessage(client, TOPIC, &pubmsg, &token);
printf("Waiting for up to %d seconds for publication of %s\n"
"on topic %s for client with ClientID: %s\n",
(int)(TIMEOUT/1000), PAYLOAD, TOPIC, CLIENTID);
rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
printf("Message with delivery token %d delivered\n", token);
MQTTClient_disconnect(client, 10000);
MQTTClient_destroy(&client);
return rc;
}
```

- 3.在sample目录用以下命令进行编译：

```
gcc -I../ -L../../../paho.mqtt.c/build/output/ MQTTClient_subscribe.c -lpaho-mqtt3c -o MQTTTest -lrt
```

- 4.编译完成后运行可执行文件：

  ./MQTTTest

- 5.成功运行后可在IoT OS后台上下行数据中查询到相关设备的登录信息和数据上报信息。

注： 若在IoT OS后台创建的是MQTT协议登录需校验产品，则2.2中的代码需要在44行后增加两行：

```
conn_opts.username = &USERNAME;//定义MQTT登录用户名
conn_opts.password = &PASSWORD;//定义MQTT登录用户密码
```

【注】其中的USERNAME与PASSWORD需根据对应校验规则进行拼接计算，计算方式参考。

- CoAP协议接入
  - CoAP简介
  - 协议特点
  - 协议接入
  - 操作前提
  - 操作步骤
  - 数据格式

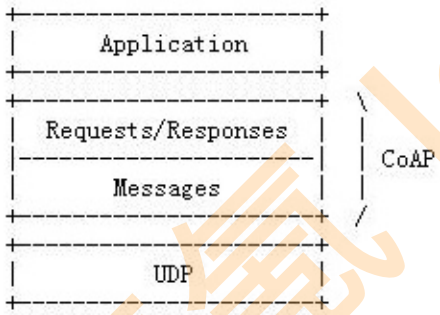
CoAP协议接入

CoAP简介

CoAP（Constrained Application Protocol）是一种在物联网世界的类web协议，它的详细规范定义在 RFC 7252。CoAP名字翻译来就是“受限应用协议”，由于物联网中的很多设备都是资源受限型的，即只有少量的内存空间和有限的计算能力，所以传统的HTTP协议应用在物联网上就显得过于庞大而不适用。 IETF的CoRE工作组提出了一种基于REST架构的CoAP协议。CoAP是6LowPAN协议栈中的应用层协议。

协议特点

- 1.CoAP协议的传输层协议为UDP。



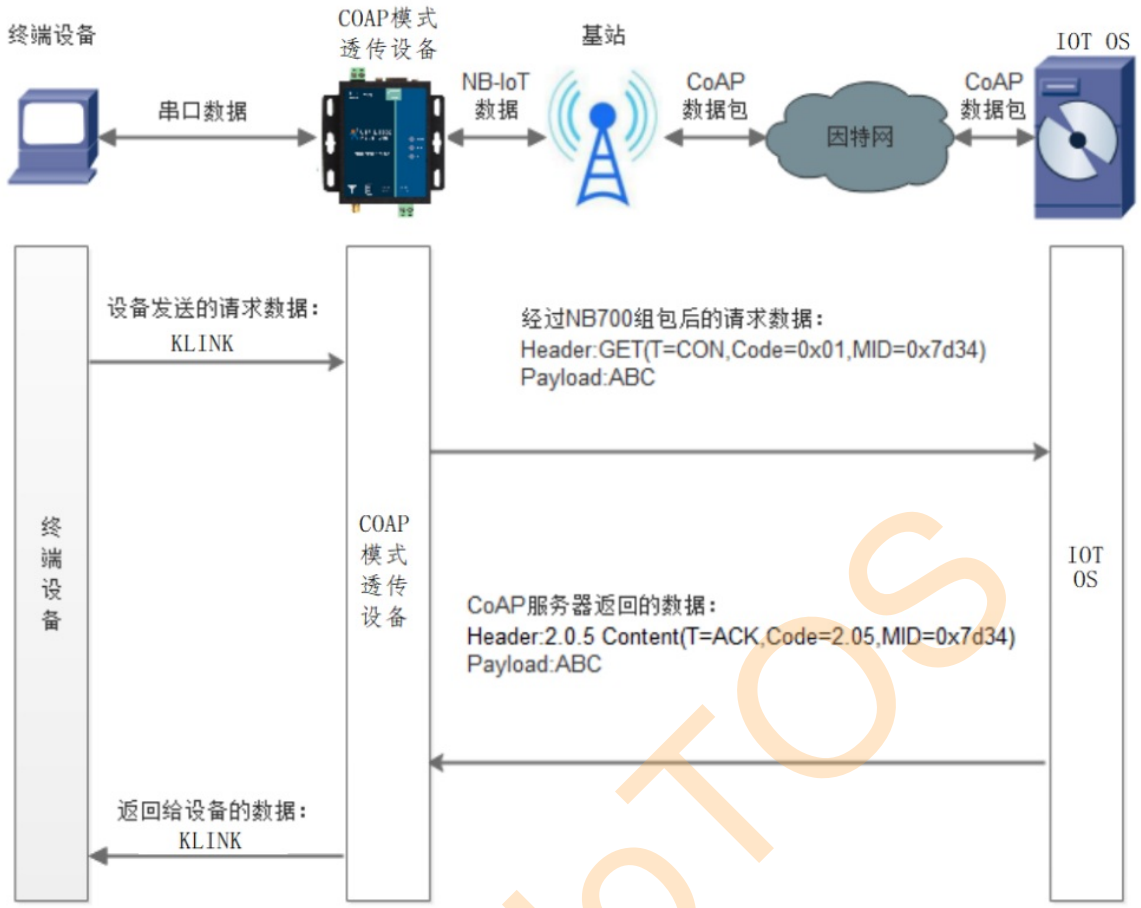
- 2.基于REST，server的资源地址和互联网一样也有类似url的格式，客户端同样有POST，GET,PUT,DELETE方法来访问server，对HTTP做了简化。
- 3.CoAP是二进制格式的，HTTP是文本格式的，CoAP比HTTP更加紧凑。
- 4.轻量化，CoAP最小长度仅仅4B，光HTTP的头已几十个B了。
- 5.支持可靠传输，数据重传，块传输。 确保数据可靠到达。
- 6.支持IP多播, 即可以同时向多个设备发送请求。
- 7.非长连接通信，适用于低功耗物联网场景。

协议接入

采用CoAP协议的终端设备，可以通过集成CoAP模式的DTU设备发送请求数据到指定的 CoAP 服务器，然后设备接收来自CoAP 服务器的数据，对数据进行解析并将结果发至串口设备。

本章使用集成CoAP模式的DTU设备对接IoT OS物联网云平台，可以将数据发送到云平台后，通过云平台提供的接口,用户自己开发自己的应用程序。

用户不需要关注串口数据与网络数据包之间的数据转换过程，只需通过简单的参数设置，即可实现串口设备向IoT OS物联网云平台的数据交互。 CoAP模式链路图：



操作前提

```
1.已创建好产品和设备（细节参考产品管理、设备管理）
2.定义好产品模型（细节参考模型管理）
```

产品管理、设备管理、模型管理

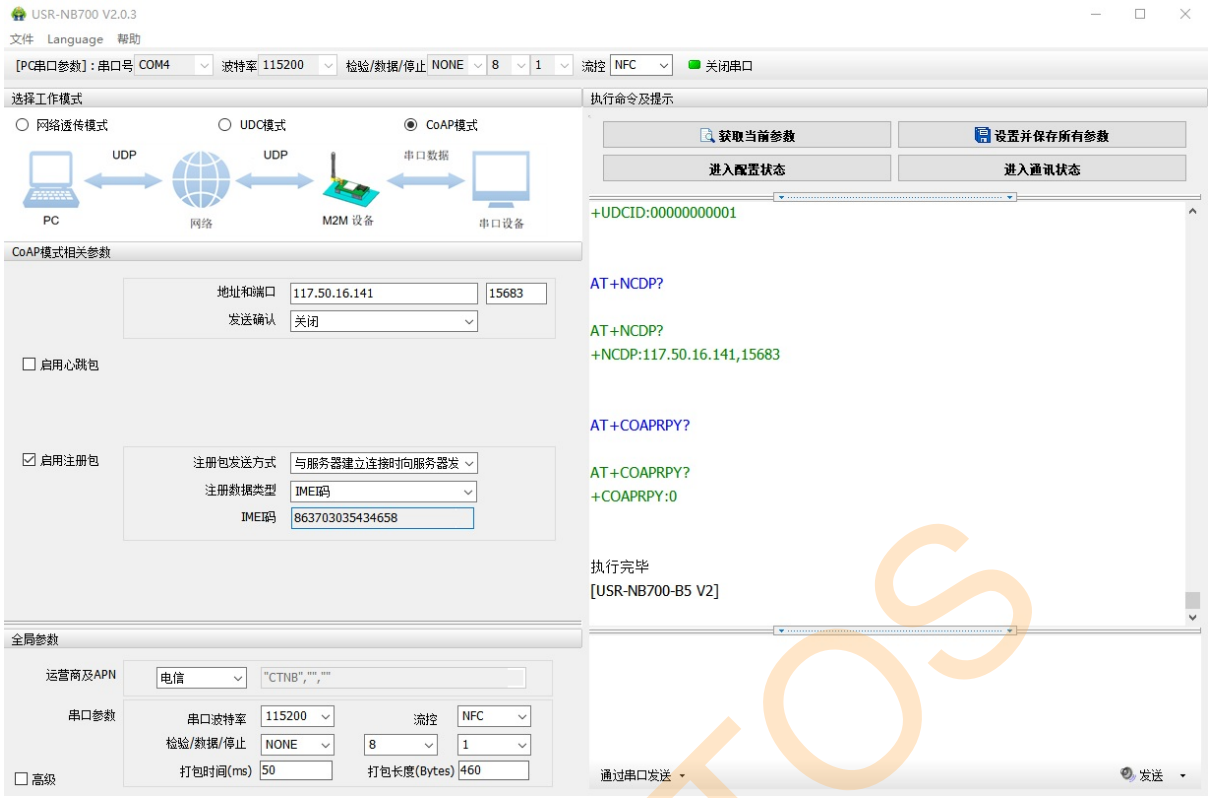
操作步骤

用户可以通过设备与IoT OS进行对接，以下以USR-NB700软件为例，进行接入演示。

设备参数配置：

- （1）将CoAP模式透传设备上电，并通过485-USB转接线将设备连接到PC机。PC机打开配置软件USR-NB700 V2.0.3.exe； （下载地址）
- （2）在软件中打开对应的串口号，波特率配置为115200，点击“打开串口”；
- （3）将工作模式选择为“CoAP模式”，在地址栏中输入IoT OS的CoAP连接IP：117.50.16.141和端口：15683；
- （4）勾选“启用注册包”；

配置参数如下图：



- （5）点击“进入配置状态”-“设置并保存所有参数”-“进入通讯状态”；
- （6）上面操作执行完毕后，在平台查看设备；
- （7）设备状态“在线”后，在串口输入框中填写上报的数据，点击发送（IoT OS标准数据格式KLink Json）：

```
{"action":"devSend","msgId":1,"pk":"xxxxxxxxxxxx","devId":"xxxxxxxxxxxx","data":{"cmd":"reportPower","para
ms":{"power":0,"hum":"one"}}}
```

数据格式

用户在创建产品时可以对产品传输的数据格式进行设置，CoAP设备可以选择的数据格式为“自定义格式”和“KLink格式”，用户在设备接入是要用对应的数据格式进行数据传输。

KLink格式

当设备为使用KLink格式传输数据时，数据格式需要符合KLink协议。详情参考KLink协议接入。 并且使用的指令必须是CoAP设备支持的指令。

CoAP设备支持的指令如下表所示：

指令功能	发送形式	详情
设备上报数据	D => C	devSend
云端回复设备上报数据	C => D	devSendResp
云端给设备下发指令	C => D	cloudSend
设备回复云端下发指令	D => C	cloudSendResp
设备上报固件信息	D => C	reportFirmware
云端回应设备上报固件信息	C => D	reportFirmwareResp

子设备上线	D => C	devLogin
云端回复子设备上线	C => D	devLoginResp
子设备下线	D => C	devLogout
云端回复子设备下线	C => D	devLogoutResp

自定义格式

若为自定义格式的设备，用户需要在IoT OS中“产品开发”功能的“数据解析”中提前写好协议解析的脚本。通过协议解析脚本，可以将用户上报的原始数据解析成KLink格式。操作参考。

当用户完成解析脚本的编写并运行后，仅需在操作步骤第（7）步中上传的数据改为符合解析脚本的数据即可。

【例】假设用户设置的自定义格式为KLink格式，则在"产品开发"的"数据解析"中写好解析KLink的解析脚本。

解析脚本内容可在平台"设备开发"-"解析数据"-"脚本编辑"框右上角的[示例]中获取.

此时按照脚本，设备可以发送hex数据。

按照KLink json格式的数据为：

```
{"action":"devSend","msgId":1,"pk":"953d2282**********8","devId":"53847******","data":{"cmd":"reportPower","
params":{"power":1}}}
```

则符合解析脚本的数据为：

```
7B22616374696F6E223A2264657653656E64222C226D73674964223A312C22706B223A223935336432323832 2A2A2A2A2A2A2A2A2A2A2
A38222C226465764964223A2235333834372A2A2A2A2A2A222C226461746 1223A7B22636D64223A227265706F7274506F776572222C22
706172616D73223A7B22706F776572223A317D7D7D
```

- LwM2M协议接入
  - LwM2M简介
  - 协议特点
  - 操作前提
  - 操作步骤
  - 数据格式

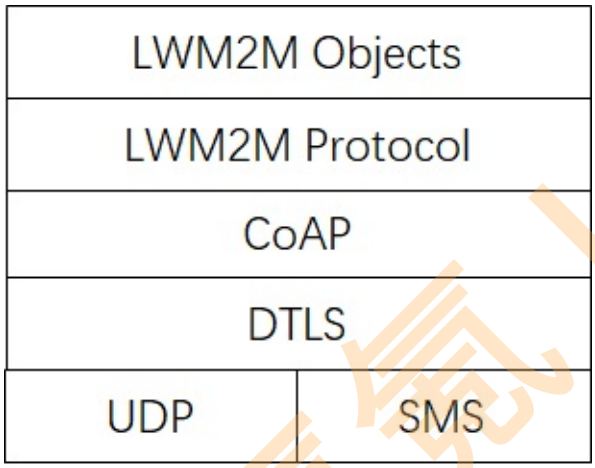
LwM2M协议接入

LwM2M简介

LwM2M（lightweight Machine to Machine），是由OMA（open Mobile Alliance)定义的物联网协议，主要使用在资源受限(包括存储、功耗等)的NB终端。

协议特点

LwM2M协议栈如下图所示：



- LwM2M 把设备上的服务抽象为 Object 和 Resource，并在 XML 文件中定义各种 Object 的属性和功能。
- LwM2M Objects：每个对象对应客户端的某个特定功能实体。LwM2M 规范定义了标准Objects，比如 urn:oma:lwm2m:oma:1; (LwM2M Server Object)、urn:oma:lwm2m:oma:3; (Device Object)，每个object下可以有很多resource。 比如Device Object可以有Manufacturer，Model Number等resource。
- LwM2M Protocol定义了一些逻辑操作，比如Read、Write、Execute等。
- CoAP是IETF定义的Constrained Application Protocol，用来做LwM2M的传输层，下层可以是UDP或者SMS，UDP是必须支持的，SMS可选。
- DTLS用来保证客户端和服务器间的安全性。
- 支持IMEI认证和SM9认证两种设备认证方式。

OMA官方网站

LwM2M协议

LwM2M定义的Object和Resourc

操作前提

```
1.已创建好产品和设备（细节参考产品管理、设备管理）
2.定义好产品模型（细节参考模型管理）
```

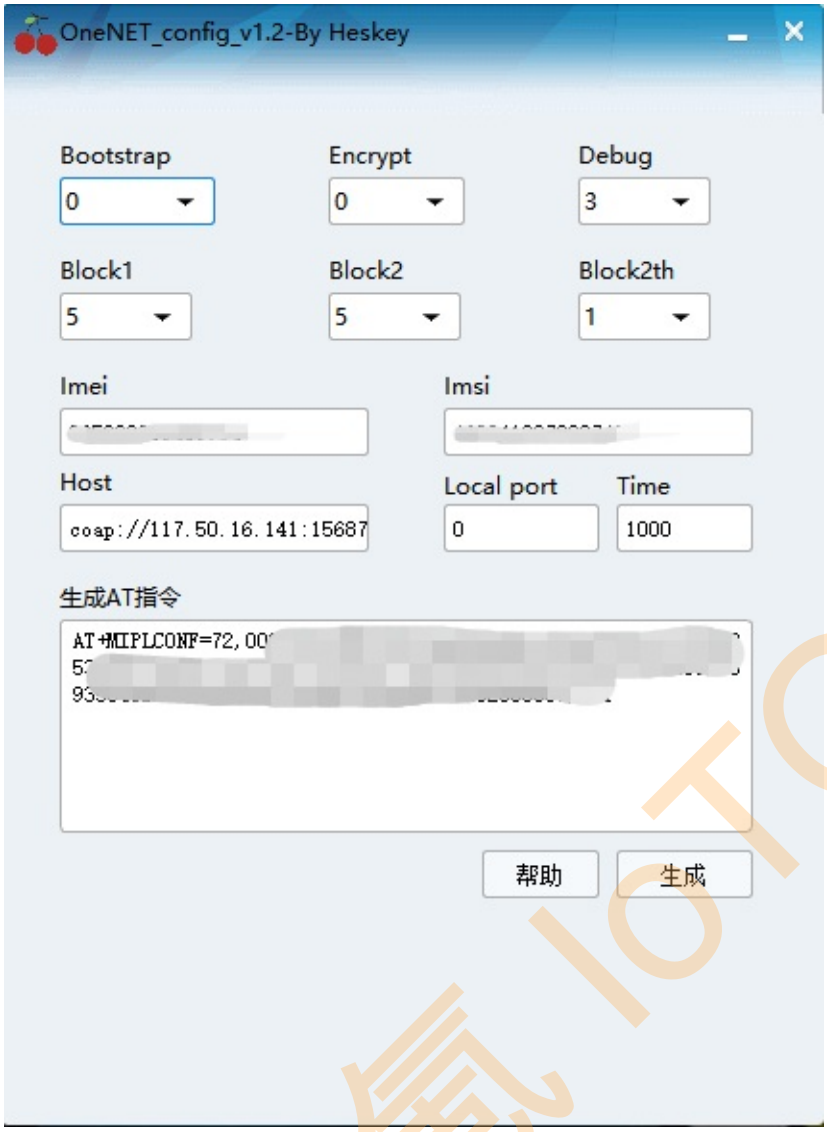
产品管理、设备管理、模型管理

操作步骤

用户可以通过市面上大多数模组与IoT OS进行对接，此文以移动M5310模组为例进行操作示范。

```
1.将移动M5310模组插上物联网卡并上电，并通过TTL-USB转接线将串口连接到PC机，PC机打开串口软件。
2.在软件中打开对应的串口号，波特率配置为9600，点击“打开串口”。
3.通过串口工具依次输入以下命令：
    1)上电检查流程：
        AT            //判断模组是否上电开机成功
        AT+CSQ       //信号质量检查
        AT+CEREG?    //判断 PS 域附着状态，标识位返回 1 或 5 表示附着正常AT+CGATT?  //检查模组 PS 附着状态
    2)模组侧设备创建
    打开命令自动生成软件OneNET_Tool.exe，在Imei配置项中填入模组IMEI码，在Imsi配置项中填入IMSI码，在Host配置项中填入服务器IP和端口，点击“生成”按钮后在“生成AT指令”栏中会生成一系列字符串，将该字符串复制并粘贴到串口数据发送栏并点击发送，返回0  OK
```

```
Eg: AT+MIPLCONF=58,100300002C0100001A00636F61703A2F2F3132332E35392E38312E3130323A3135363839110038363538323030333034353139353433B33050501,1,1
```




```
    3)给模组配置对象 ObjectId 3200
        AT+MIPLADDOBJ=0,3200,0 返回 OK
    4)设备跟服务器建立连接
        AT+MIPLOPEN=0,3600  返回 OK及一系列数据，表示登录成功，此时登录到后台可以看到设备已经上线
    5)数据上报，向ObjectId 3200 instanceId 0 resourceId 5505 发送数据
        AT+MIPLNOTIFY=0,3200,0,5505,6,"74657374",1返回  OK及一系列数据，表示数据上报成功，平台收到上报数据"74657374"转化
的字符串“test”。可修改指令中“”内的内容上报不同数据。
    6)登录注销
        AT+MIPLCLOSE=0返回  OK
        AT+MIPLDELOBJ=0,3200,0   //模组侧订阅资源列表释放
        AT+MIPLDEL=0   //模组侧通信实例删除
```

数据格式

用户在创建产品时可以对产品传输的数据格式进行设置，LwM2M设备可以选择的数据格式为“自定义格式”和“KLink格式”，按照产品格式的不同，设备接入也略有不同。

自定义格式

用户在严格遵守LwM2M协议的情况下，可以进行自定义格式的传输。详情可见LwM2M协议特点。 其设备端操作如上述操作步骤所示。

【注意】

- 用户需要按照自定的格式在产品管理处对产品进行数据解析脚本的编写。数据解析脚本详情。

KLink格式

若用户在建立产品时，选择的数据传输格式为KLink格式，则需要对设备端进行以下改动。

- 在操作步骤设置模组对象时，设置ObjectId 19；instanceId 0；resourceId 0。
- 传输的数据要符合KLink格式，并且功能属于LwM2M设备支持的功能。

*KLink*协议相关可以查看*KLink*协议接入。

*LwM2M*设备支持的指令如下表所示。

指令功能	发送形式	详情
设备上报数据	D => C	devSend
云端回复设备上报数据	C => D	devSendResp
云端给设备下发指令	C => D	cloudSend
设备回复云端下发指令	D => C	cloudSendResp
设备上报固件信息	D => C	reportFirmware
云端回应设备上报固件信息	C => D	reportFirmwareResp
子设备上线	D => C	devLogin
云端回复子设备上线	C => D	devLoginResp
子设备下线	D => C	devLogout
云端回复子设备下线	C => D	devLogoutResp

- HTTP协议接入
  - HTTP简介
  - 协议特点
  - 操作前提
  - 操作步骤
  - 指令详情
    - 指令列表

HTTP协议接入

HTTP简介

HTTP协议（HyperText Transfer Protocol，超文本传输协议）是互联网上应用最为广泛的一种网络协议。它用于传送WWW方式的数据，关于HTTP协议的详细内容请参考RFC2616。平台支持http及https两种接入方式。

HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求，请求头包含请求的方法、URL、协议版本、以及包含请求修饰符、客户信息和内容的类似于MIME的消息结构。服务器以一个状态行作为响应，响应的内容包括消息协议的版本，成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。

协议特点

- 支持HTTP/HTTPS协议，仅支持上行报文和获取缓存的下发报文(最多10条)。
- HTTP为短连接会话，无法实现长连接建链。维持1小时内会话(token有效时间为1小时)。
- 报文类型：接入鉴权报文，上行数据报文，获取下发数据报文。

操作前提

```
1.已创建好产品和设备（细节参考产品管理、设备管理）
2.定义好产品模型（细节参考模型管理）
```

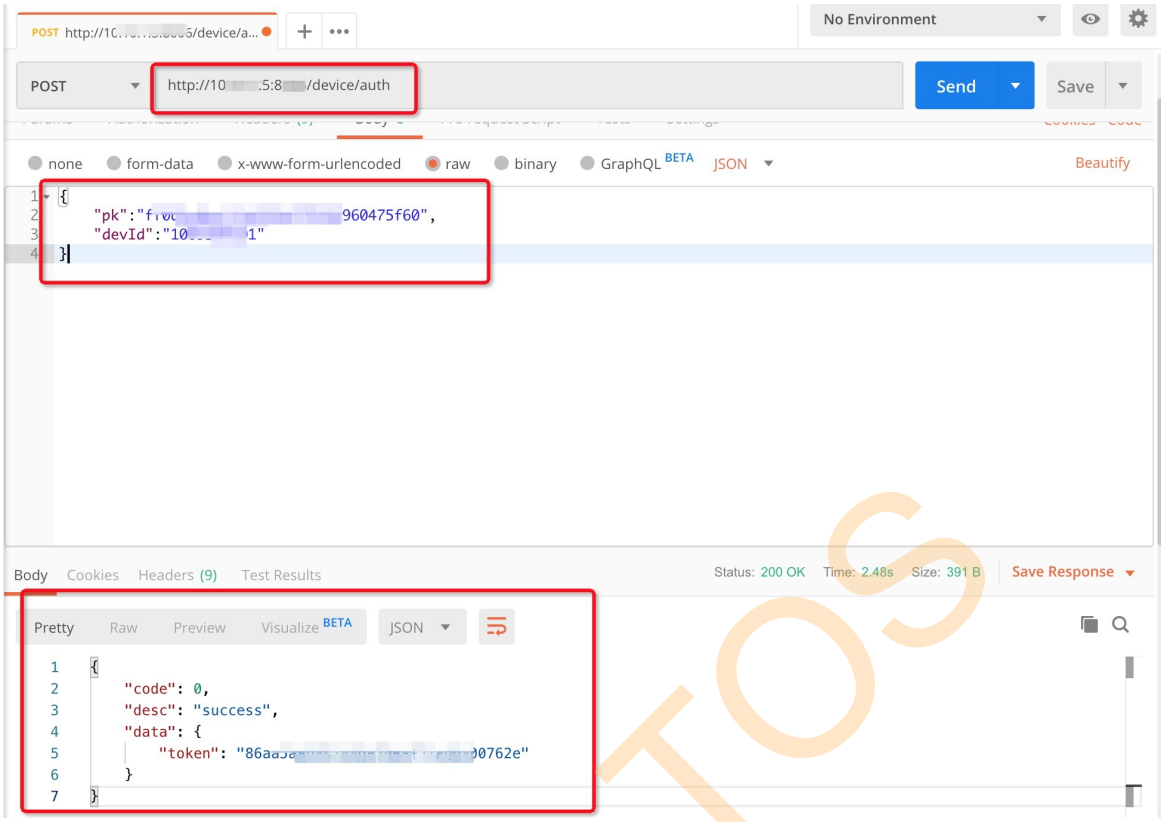
产品管理、设备管理、模型管理

操作步骤

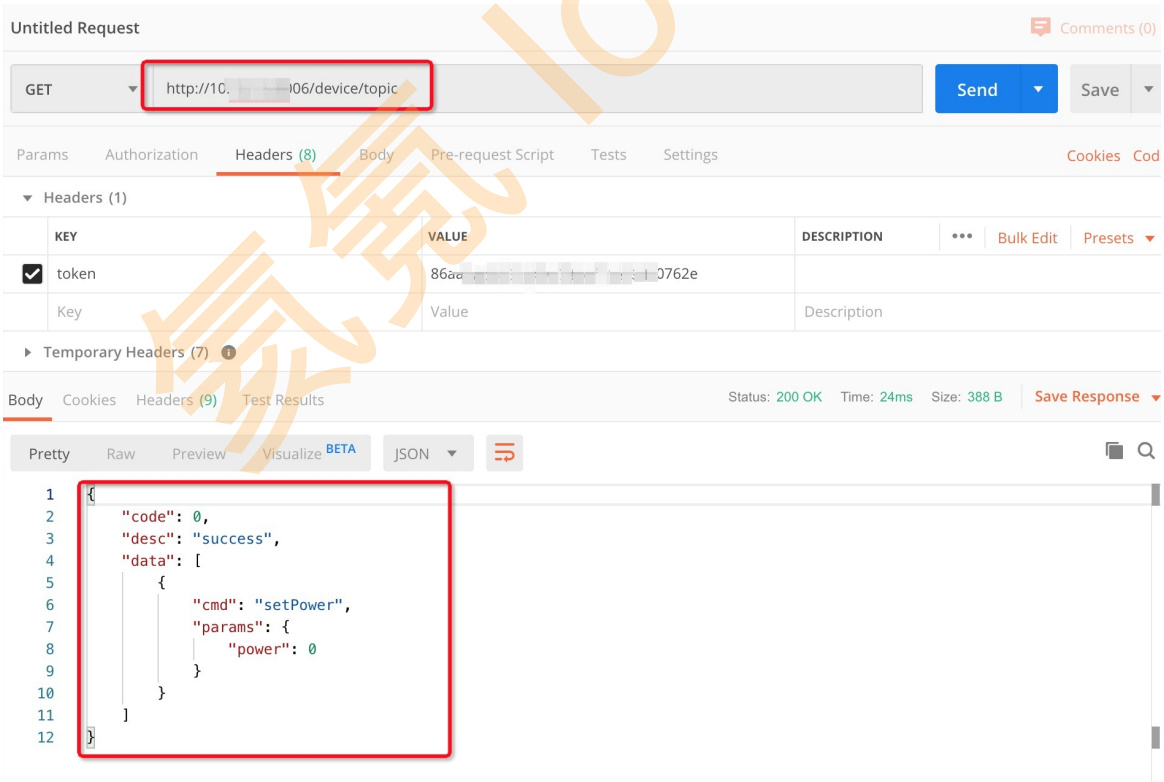
支持HTTP协议传输的设备按照下列支持的指令请求方式进行http访问即可。

下图是使用postman模拟http设备进行设备上线和主动拉取云端命令操作。

- 设备上线



- 主动拉取云端命令



指令详情

指令列表

指令功能	发送形式	详情
设备上线	D => C	/device/auth
设备上报数据	D => C	/device/topic
设备主动拉取云端下发命令	D => C	/device/topic

设备上线

D => C

```
POST /device/auth HTTP/1.1

Content-Type:application/json
Accept:application/json

{
    "pk":"f264aed445b745aea0ea8f2c4880b090",
    "devId":"820842994100009",
    "random":"8",
    "hashMethod":"HmacMD5",
    "sign":"f3bea7ab22f3a9291cb9494ca0951f86"
}
```

参数	必填	类型	说明
pk	是	string	子设备节点 productKey。
devId	是	string	子设备节点 devId。
random	否	string	设备加密校验，随机字符串，可以使用当前时间毫秒数的字符串，也可以使用uuid生成的，也可以是 math.random()等等。
hashMethod	否	string	hash 方法，选填 `HmacMD5` 、 `HmacSHA1` 、 `HmacSHA256` 或者 `HmacSHA512` 。
sign	否	string	要注册的设备的产品认证签名。

其中只有当设备需要登录安全校验时才需要填写random、hashMethod和sign。

sign的计算方式可以参考密码计算方式中对password的计算。

【注意】发起"设备上线"请求后，云端会对请求回复。即下述【云端回复设备上线】。

云端回复设备上线

C => D

```
HTTP/1.1 200 OK

{
    "code": 0,
    "desc": "success",
    "data": {
        "token": "68c75412cdc346178c5fb1fc0cd42237"
    }
}
```

参数	必填	类型	说明

code	是	uint	如果没有错误回复 0。
desc	是	string	错误说明。
token	是	string	设备token，后续所有操作都需要携带，如果token失效后需要再次登录获取(默认有效时间为1个小时)。

设备上报数据

D => C

```
POST /device/topic HTTP/1.1

Content-Type:application/json
Accept:application/json
token:{{token}}

{
  "cmd": "up",
  "params":{
    "data":"updata1"
  }
}
```

参数	必填	类型	说明
token	是	string	登录时云端返回的token值。
cmd	是	string	标识符。
params	是	object	参数，如果只有指令，没有参数允许不填；否则应该填写该指令下的参数标识符。可以少不能多，值做强较验（类型，长度等必须符合协议规定）。

【注意】发起"设备上报数据"请求后，云端会对请求回复。即下述【云端回复设备上报数据】。

云端回复设备上报数据

C => D

```
HTTP/1.1 200 OK

{
    "code": 0,
    "desc": "success"
}
```

参数	必填	类型	说明
code	是	uint	如果没有错误回复 0。
desc	是	string	错误说明。

设备主动拉取云端下发命令

D => C

```
GET /device/topic HTTP/1.1

Content-Type:application/json
```

```
Accept:application/json
token:{{token}}
```

参数	必填	类型	说明
token	是	string	登录时云端返回的token值。

【注意】发起"设备主动拉取云端下发命令"请求后，云端会对请求回复。即下述【云端回复信息】。

云端回复信息

```
HTTP/1.1 200 OK

{
    "code": 0,
    "desc": "success",
    "data": [
        {
            "cmd": "down",
            "params": {
                "data": "1"
            }
        },
        {
            "cmd": "down",
            "params": {
                "data": "1"
            }
        }
    ]
}
```

参数	必填	类型	说明
code	是	uint	如果没有错误回复 0。
desc	是	string	错误说明。
data	否	object	云端下发命令。

- 订阅设备消息

订阅设备消息

IoT OS支持将设备上报到平台的数据通过HTTP或Topic流转至指定的服务地址或Topic中。用户可以通过自定义数据筛选规则和转发内容，将产品下的设备所有类型消息，进行筛选并按指定格式转发到指定服务地址或Topic中。平台提供数据流转服务的有服务端订阅和规则引擎两个模块。

- 服务端订阅
  - 操作说明
  - 订阅类型：
    - 设备上报消息
    - 数据变化通知
    - 控制响应通知
    - 上下线通知

服务端订阅

服务端订阅是通过HTTP通道，将产品下所有设备数据按订阅类型转发到指定的HTTP地址。服务端订阅适用于单纯接收设备数据的场景。可快速地获取设备消息，无消息过滤功能，功能较单一但简单易用且高效。目前支持的订阅类型消息有：数据上报消息、数据变化通知、控制响应通知、上下线通知。

操作说明

```
1．“产品开发”页面，点击单个产品，点击“服务端订阅”；
2．在“服务端订阅”中，点击“添加订阅”按钮；
3．在“创建服务端订阅”的编辑框中，选择要订阅的类型、填写订阅的名称、订阅方URL地址；
4．点击确定。
```

【说明】

订阅方URL地址： 数据流转的目的地

【注意】

平台以HTTP POST请求形式向第三方平台URL地址推送数据，第三方平台接收到数据后需要返回HTTP状态码 2xx，否则IoT OS会认为此次推送无效并重试，最多重试3次。且等待客户端的响应都设有时限（目前是5秒），在规定时限内没有收到响应会认为发送失败，连续失败100次（包括重试次数，并且计数规则是根据URL计算。相同URL下的不同规则都会被计算在内。）则会认为第三方地址不可用，推送服务将停止，对应订阅或者规则状态显示为"未启用"。建议第三方应用接收程序接收到数据时，先做数据缓存，再做业务逻辑处理。 其中请求的 `Content-Type` 为 `application/json` 。

订阅类型：

下述为用户通过HTTP通道获取到的不同订阅类型下的数据。

设备上报消息

指产品下的设备进行上报消息（devSend）时通知给订阅端的消息。

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_params": {
        "power": 0
    },
    "iot_sys_event": "devSend",
    "iot_sys_cmd": "reportPower",
    "iot_sys_raw": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
```

```
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_resp": "{\"action\":\"devSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devI
d\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576467306280
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_params	object	所属产品定义的参数
iot_sys_event	String	事件（devSend\devLogin等事件）
iot_sys_cmd	String	所属产品定义的命令
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

数据变化通知

指产品下的设备参数值发生变化时通知给订阅端的消息


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_event": "dataChanged",
    "iot_sys_raw": "{\"last\":{\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"tim
estamp\":1576482068202,\"data\":{\"cmd\":\"reportPower\",\"params\":{\"hum\":23,\"light\":50,\"power\":1}}},\"c
urrent\":{\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"timestamp\":157648207251
4,\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}},\"changed\":true}",
    "iot_sys_timestamp": 1576482072526,
    "iot_sys_snapshot": "{last=Snapshot(pk=fb4327cdcfe5432e82ad217572bd8ebd, devId=8237948938472348, timestamp=
1576482068202, data=ModelData(cmd=reportPower, params={hum=23, light=50, power=1})), current=Snapshot(pk=fb4327
cdcfe5432e82ad217572bd8ebd, devId=8237948938472348, timestamp=1576482072514, data=ModelData(cmd=reportPower, pa
rams={power=0}))}"
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）
iot_sys_snapshot	object	设备快照

【注意】

设备快照中last字段表示数据改变前一次的快照，current字段表示数据改变时的快照。

如果是第一次上报数据，last 为 null。

控制响应通知

指产品下的设备对下发命令应答时通知给订阅端消息。


```
{
    "iot_sys_devId": "10003",
    "iot_sys_pk": "da47c27d593e4ecfb2e8f031875463d1",
    "iot_sys_event": "cloudSendResp",
    "iot_sys_raw":"{\n\"action\":\"cloudSendResp\",\n\"pk\":\"da47c27d593e4ecfb2e8f031875463d1\",\n\"devId\":\"
10003\",\n\"msgId\":1,\n\"code\":0\n}\n",
    "iot_sys_timestamp": 1573111224907
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_timestamp	number	请求时间（ms）

上下线通知

产品下的设备发生上下线状态变化时通知给订阅端的消息。

【上线通知】：


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devLogin\",\"msgId\":0,\"random\":\"init\",\"hashMethod\":\"HmacSHA1\",\"sign
\":\"4f6a95a54bad27a874132929781ff4fea1748b7c\"}",
    "iot_sys_event": "devLogin",
    "iot_sys_raw": "{\"action\":\"devLogin\",\"msgId\":0,\"random\":\"init\",\"hashMethod\":\"HmacSHA1\",\"sign
\":\"4f6a95a54bad27a874132929781ff4fea1748b7c\"}",
    "iot_sys_resp": "{\"action\":\"devLoginResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237948938472348\",\"code\":0,\"desc\":\"success, \"}",
    "iot_sys_timestamp": 1576468460670
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据

iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

【离线通知】：


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devLogout\",\"msgId\":0,\"reason\":\"device send disconnect\"}",
    "iot_sys_event": "devLogout",
    "iot_sys_raw": "{\"action\":\"devLogout\",\"msgId\":0,\"reason\":\"device send disconnect\"}",
    "iot_sys_resp": "{\"action\":\"devLogoutResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576468459769
}
```


字段	说明
iot_sys_devId	设备ID
iot_sys_pk	设备所属产品pk
iot_sys_req	解析后的KLink数据
iot_sys_event	事件
iot_sys_raw	原始请求数据
iot_sys_resp	云端针对该事件的数据回复
iot_sys_timestamp	请求时间（ms）

- 规则引擎
  - 操作说明
  - SELECT字段说明
  - 过滤条件WHERE
  - 设备操作事件对应推送数据格式详情
    - 设备登录(devLogin)
    - 设备登出(devLogout)
    - 设备上报(devSend)
    - 数据变化(dataChanged)
    - 设备上报固件信息(reportFirmware)
    - 控制下发(cloudSend)
    - 控制下发回复(cloudSendResp)
    - 控制设备升级-包括远程配置(devUpgrade)
    - 设备升级回复(devUpgradeResp)
    - 设备上报升级进度(devUpgradeProgress)
    - 注册设备(register)

规则引擎

相比服务端订阅只单纯流转上行数据，规则引擎功能更强大。可对上下行数据进行操作，支持用户自定义筛选规则以及指定推送内容，对要流转的数据进行过滤，支持HTTP和MQTT两种流转方式。

规则引擎中通过编写SQL来解析和处理数据，SQL表达式为：

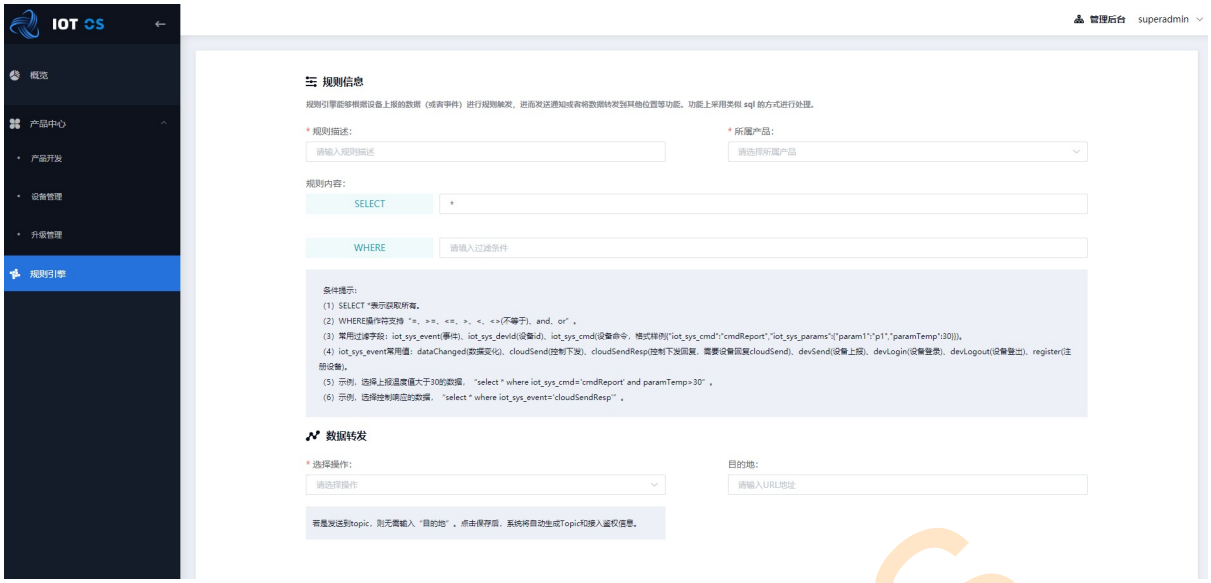
```
SELECT 【转发的字段】 FROM "数据流" WHERE "过滤条件"
```

【注意】

平台以HTTP POST请求形式向第三方平台URL地址推送数据时，第三方平台接收到数据后需要返回HTTP状态码 2xx，否则IoT OS会认为此次推送无效并重试，最多重试3次。且等待客户端的响应都设有时限（目前是5秒），在规定时限内没有收到响应会认为发送失败，连续失败100次（包括重试次数，并且计数规则是根据URL计算。相同URL下的不同规则都会被计算在内。）则会认为第三方地址不可用，推送服务将停止，对应订阅或者规则状态显示为"未启用"。建议第三方应用接收程序接收到数据时，先做数据缓存，再做业务逻辑处理。 其中请求的 `Content-Type` 为 `application/json` 。

操作说明

```
1. 登录平台;
2. 在左侧导航栏中选择“规则引擎”;
3. 单击“创建规则”;
4. 选择产品、填写推送字段以及过滤条件;
5. 选择数据转发操作;
6. 添加目的地址（要求服务端使用POST请求）。
```



【说明】

1. SELECT中“*”表示转发数据中所有字段；
2. WHERE操作符支持“=、>=、<=、>、<、<>(不等于)、and、or”，不支持函数。

SELECT字段说明

可选字段	字段说明
iot_sys_event	事件（devSend\devLogin等事件）
iot_sys_devId	设备ID
iot_sys_pk	设备所属产品pk
iot_sys_req	解析后的KLink数据
iot_sys_resp	云端针对该事件的数据回复
iot_sys_raw	原始请求数据
iot_sys_timestamp	请求时间（ms）
iot_sys_cmd	设备命令
【协议参数】	协议中定义的所有参数标识符

过滤条件WHERE

可选字段	字段说明	可取值
iot_sys_event	事件	dataChanged：数据变化； cloudSend：控制下发； cloudSendResp：控制下发回复； devSend：设备上报； devLogin：设备登录； devLogout：设备登出； reportFirmware：设备上报固件信息； devUpgradeProgress：设备上报升级进度； devUpgrade：控制设备升级（包括远程配置）； devUpgradeResp：设备升级回复；

		register：注册设备。
iot_sys_devId	设备ID	设备ID，例如：`10001` 。
iot_sys_pk	设备所属产品pk	设备pk，例如：`c54b2bc00c164e569ac6666aa6a81e8c` 。
iot_sys_cmd	设备命令	协议中定义的命令，例如： `iot_sys_cmd="reportPower"` 。
【协议参数】	协议中定义的所有参数标识符	例如： `hum>90` 。

用户可以通过规则引擎推送实例中的三个实例来更好的理解规则引擎的用法和功能。

设备操作事件对应推送数据格式详情

下述数据结构表示在一个不进行任何字段筛选（SELECT）或条件过滤（WHERE）的规则引擎下，设备进行相应操作（如设备进行 `devlogin` ）时，用户通过HTTP或Topic获得到的数据。

即推送实例中的对照实例所获取到的推送数据。

【注意】

用户在进行字段筛选或条件过滤时需要保证字段或者条件被包含在下述数据格式中，尤其是对参数的过滤条件需要被参数的数据类型支持。

设备登录(devLogin)

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devLogin\",\"msgId\":0,\"random\":\"init\",\"hashMethod\":\"HmacSHA1\",\"sign
\":\"4f6a95a54bad27a874132929781ff4fea1748b7c\"}",
    "iot_sys_event": "devLogin",
    "iot_sys_raw": "{\"action\":\"devLogin\",\"msgId\":0,\"random\":\"init\",\"hashMethod\":\"HmacSHA1\",\"sign
\":\"4f6a95a54bad27a874132929781ff4fea1748b7c\"}",
    "iot_sys_resp": "{\"action\":\"devLoginResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237948938472348\",\"code\":0,\"desc\":\"success, \"}",
    "iot_sys_timestamp": 1576466870416
}
```

参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

设备登出(devLogout)

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devLogout\",\"msgId\":0,\"reason\":\"device send disconnect\"}",
```

```
    "iot_sys_event": "devLogout",
    "iot_sys_raw": "{\"action\":\"devLogout\",\"msgId\":0,\"reason\":\"device send disconnect\"}",
    "iot_sys_resp": "{\"action\":\"devLogoutResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576467109928
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

设备上报(devSend)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_params": {
        "power": 0
    },
    "iot_sys_event": "devSend",
    "iot_sys_cmd": "reportPower",
    "iot_sys_raw": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_resp": "{\"action\":\"devSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devI
d\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576467306280
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_params	object	所属产品定义的参数
iot_sys_event	String	事件
iot_sys_cmd	String	所属产品定义的命令
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

数据变化(dataChanged)

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}}",
    "iot_sys_event": "dataChanged",
    "iot_sys_raw": "{\"last\":{\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"tim
estamp\":1576482068202,\"data\":{\"cmd\":\"reportPower\",\"params\":{\"hum\":23,\"light\":50,\"power\":1}}},\"c
urrent\":{\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"timestamp\":157648207251
4,\"data\":{\"cmd\":\"reportPower\",\"params\":{\"power\":0}}},\"changed\":true}",
    "iot_sys_timestamp": 1576482072526,
    "iot_sys_snapshot": "{last=Snapshot(pk=fb4327cdcfe5432e82ad217572bd8ebd, devId=8237948938472348, timestamp=
1576482068202, data=ModelData(cmd=reportPower, params={hum=23, light=50, power=1})), current=Snapshot(pk=fb4327
cdcfe5432e82ad217572bd8ebd, devId=8237948938472348, timestamp=1576482072514, data=ModelData(cmd=reportPower, pa
rams={power=0}))}"
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）
iot_sys_snapshot	object	设备快照

【注意】

设备快照中last字段表示数据改变前一次的快照，current字段表示数据改变时的快照。

如果是第一次上报数据，last 为 null。

设备上报固件信息(reportFirmware)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"reportFirmware\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"type\":\"module\",\"version\":\"1.2.1\"}",
    "iot_sys_event": "reportFirmware",
    "iot_sys_raw": "{\"action\":\"reportFirmware\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"type\":\"module\",\"version\":\"1.2.1\"}",
    "iot_sys_resp": "{\"action\":\"reportFirmwareResp\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\"
,\"devId\":\"8237948938472348\",\"code\":0}",
    "iot_sys_timestamp": 1576029149134
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件

iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

控制下发(cloudSend)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"cloudSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\"
:\"8237948938472348\",\"data\":{\"cmd\":\"setPower\",\"params\":{\"power\":0}}}",
    "iot_sys_event": "cloudSend",
    "iot_sys_raw": "{\"action\":\"cloudSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\"
:\"8237948938472348\",\"data\":{\"cmd\":\"setPower\",\"params\":{\"power\":0}}}",
    "iot_sys_resp": "{\"action\":\"cloudSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576467240890
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

控制下发回复(cloudSendResp)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"cloudSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237948938472348\",\"code\":0}",
    "iot_sys_event": "cloudSendResp",
    "iot_sys_raw": "{\"action\":\"cloudSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237948938472348\",\"code\":0}",
    "iot_sys_timestamp": 1576029802710
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_timestamp	number	请求时间（ms）

控制设备升级-包括远程配置(devUpgrade)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devUpgrade\",\"msgId\":0,\"url\":\"http://123.59.81.102:9002/hekr/d40a16a8a6bf4d418aa8bd3b01026fb1.zip\",\"md5\":\"1ecbdec973be231e5a84a8803996d5e5\",\"version\":\"1.2.1\",\"type\":\"module\"}",
    "iot_sys_event": "devUpgrade",
    "iot_sys_raw": "{\"action\":\"devUpgrade\",\"msgId\":0,\"url\":\"http://123.59.81.102:9002/hekr/d40a16a8a6bf4d418aa8bd3b01026fb1.zip\",\"md5\":\"1ecbdec973be231e5a84a8803996d5e5\",\"version\":\"1.2.1\",\"type\":\"module\"}",
    "iot_sys_resp": "{\"action\":\"devUpgradeResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576467452534
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

设备升级回复(devUpgradeResp)


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devUpgradeResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"code\":0}",
    "iot_sys_event": "devUpgradeResp",
    "iot_sys_raw": "{\"action\":\"devUpgradeResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"type\":\"module\",\"code\":0}",
    "iot_sys_timestamp": 1576467892152
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_timestamp	number	请求时间（ms）

设备上报升级进度(devUpgradeProgress)

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"devUpgradeProgress\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",
\"devId\":\"8237948938472348\",\"progress\":19,\"type\":\"module\"}",
    "iot_sys_event": "devUpgradeProgress",
    "iot_sys_raw": "{\"action\":\"devUpgradeProgress\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",
\"devId\":\"8237948938472348\",\"type\":\"module\",\"code\":0,\"progress\":19}",
    "iot_sys_resp": "{\"action\":\"devUpgradeProgressResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8e
bd\",\"devId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576468087217
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

注册设备(register)


```
{
    "iot_sys_devId": "8237942348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_req": "{\"action\":\"register\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":
\"8237942348\",\"random\":\"random\",\"hashMethod\":\"HmacMD5\",\"sign\":\"5e2b3d5b9023761a07dd8ca34c3db9c4\"}"
,
    "iot_sys_event": "register",
    "iot_sys_raw": "{\"action\":\"register\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":
\"8237942348\",\"random\":\"random\",\"hashMethod\":\"HmacMD5\",\"sign\":\"5e2b3d5b9023761a07dd8ca34c3db9c4\"}"
,
    "iot_sys_resp": "{\"action\":\"registerResp\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237942348\",\"code\":0,\"desc\":\"success\",\"devSecret\":\"817cbd2487064dd4a199618f29e12cbb\"}",
    "iot_sys_timestamp": 1576029701743
}
```


参数	类型	说明
iot_sys_devId	String	设备ID
iot_sys_pk	String	设备所属产品pk
iot_sys_req	String	解析后的KLink数据
iot_sys_event	String	事件
iot_sys_raw	String	原始请求数据
iot_sys_resp	String	云端针对该事件的数据回复
iot_sys_timestamp	number	请求时间（ms）

- 规则引擎推送实例
  - 实例0：对照实例
  - 实例1：转发到指定HTTP（S）地址
  - 实例2：转发到一个Topic
  - 说明

规则引擎推送实例

此处将编写三个实例来使用户更好地理解规则引擎的用法及功能。

实例0：对照实例

【说明】：

此实例将创建一个不进行任何字段筛选（SELECT）或条件过滤（WHERE）的规则引擎，即将所有的数据通过HTTP(S)的推送方式推送到指定地址。 所得到的结果可以为后续实例进行对比。

【操作步骤】：

1. 在“规则引擎”页面，点击“新建规则”，填好规则描述，选择产品；
2. 规则内容 `SELECT` 中填写 `*`；
3. 规则内容 `WHERE` 中不填写过滤条件；
4. 选择操作“发送到HTTP(S)接口”；
5. 目的地填写 `http://10.50.80.100:1200` (此地址为无效地址，仅作示例)；
6. 点击“确定”，完成规则创建；
7. 在规则列表中点击“启用”，启用规则；
8. 规则启用后，满足规则条件的数据会通过POST方式发送到所配置的http地址。

【设备操作】：

1. 设备登录；
2. 发送设备所有参数（电源开关信息 `power:1` ； 光照强度 `light: 50` ； 湿度 `hum: 23` ）；
3. 发送设备所有参数（电源开关信息 `power: 1` ； 光照强度 `light: 95` ； 湿度 `hum: 23` ）；
4. 设备登出。

【数据结果】： 共收到推送四组数据：

操作1对应的数据


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_event": "devLogin",
    "iot_sys_raw": "{\"action\":\"devLogin\",\"msgId\":1,\"sign\":\"4f6a95a54bad27a874132929781ff4fea1748b7c\",
\"random\":\"init\",\"hashMethod\":\"HmacSHA1\"}",
    "iot_sys_resp": "{\"action\":\"devLoginResp\",\"msgId\":1,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"dev
Id\":\"8237948938472348\",\"code\":0,\"desc\":\"success, \"}",
    "iot_sys_timestamp": 1576034808492
}
```


操作2对应的数据


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_params": {
        "hum": 23,
        "light": 50,
        "power": 1
    },
    "iot_sys_event": "devSend",
    "iot_sys_cmd": "reportAll",
    "iot_sys_raw": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportAll\",\"params\":{\"power\":1,\"light\":50,\"hum\":23}}}",
    "iot_sys_resp": "{\"action\":\"devSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devI
d\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576033843689
}
```


操作3对应的数据


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_params": {
        "hum": 23,
        "light": 95,
        "power": 1
    },
    "iot_sys_event": "devSend",
    "iot_sys_cmd": "reportAll",
    "iot_sys_raw": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\
"8237948938472348\",\"data\":{\"cmd\":\"reportAll\",\"params\":{\"power\":1,\"light\":95,\"hum\":23}}}",
    "iot_sys_resp": "{\"action\":\"devSendResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devI
```

```
d\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576035409147
}
```


操作4对应的数据


```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_pk": "fb4327cdcfe5432e82ad217572bd8ebd",
    "iot_sys_event": "devLogout",
    "iot_sys_raw": "{\"action\":\"devLogout\",\"msgId\":0,\"reason\":\"normal\"}",
    "iot_sys_resp": "{\"action\":\"devLogoutResp\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"de
vId\":\"8237948938472348\",\"code\":0,\"desc\":\"success\"}",
    "iot_sys_timestamp": 1576034807613
}
```


实例1：转发到指定HTTP（S）地址

【说明】：

只接收devSend请求的推送，且只把部分字段推送到指定地址。

推送地址：`http://10.50.80.100:1200` (此地址为无效地址，仅作示例)。

推送的字段为：设备ID `iot_sys_devId` 、事件 `iot_sys_event` 以及部分协议参数 `iot_sys_params` （光照强度 `light` ，湿度 `hum` ）。

【操作步骤】：

1. 在“规则引擎”页面，点击“新建规则”，填好规则描述，选择产品；
2. 根据需求“推送的字段为：设备ID、PK以及协议参数”，SELECT中填写 `iot_sys_devId,iot_sys_event,light,hum` （light，hum为协议中定义的要上报的参数）；
3. 根据需求“设备上报状态时（devSend），若设备的亮度高于阈值90时推送”，WHERE中填写； `iot_sys_event="devSend" and light > 90` （devSen为设备上报状态的数据、light > 90 表示亮度高于阈值90）；
4. 选择操作“发送到HTTP(S)接口”；
5. 目的地填写 `http://10.50.80.100:1200` （此地址为无效地址，仅作示例）；
6. 点击“确定”，完成规则创建；
7. 在规则列表中点击“启用”，启用规则；
8. 规则启用后，满足规则条件的数据会通过POST方式发送到所配置的http地址。

【设备操作】：

1. 设备登录；
2. 发送设备所有参数（电源开关信息 `power:1` ；光照强度 `light: 50` ；湿度 `hum: 23` ）；
3. 发送设备所有参数（电源开关信息 `power: 1` ；光照强度 `light: 95` ；湿度 `hum: 23` ）；
4. 设备登出。

【数据结果】：

仅收到一组数据：

操作3对应的数据

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_params": {
        "hum": 23,
        "light": 95
    },
    "iot_sys_event": "devSend"
}
```

【对照分析】：

对比对照实例，可知：

1. 规则引擎的条件过滤功能使得设备的 `操作1` 、 `操作2` 和 `操作4` 数据被过滤；
2. 规则引擎的字段筛选功能使得仅获得到设备ID `iot_sys_devId` 、事件 `iot_sys_event` 和参数 `iot_sys_params` ，且参数只转发湿度 `hum` 和光照强度 `light` 。

实例2：转发到一个Topic

【说明】：

获取设备上下线通知

推送的字段为：设备ID `iot_sys_devId` 、事件 `iot_sys_event` 以及操作时间 `iot_sys_timestamp` 。

【操作步骤】：

1. 在“规则引擎”页面，点击“新建规则”，填好规则描述，选择产品；
2. 根据需求，SELECT中填写 `iot_sys_devId,iot_sys_event,iot_sys_timestamp` ；
3. 根据需求“设备上线、离线状态变化时”时推送，WHERE中填写 `iot_sys_event="devLogin" or iot_sys_event="devLogout"` （devLogin设备上线、devLogout设备离线）；
4. 选择操作“发送到一个Topic”；
5. 点击“确定”，完成规则创建；
6. 在规则列表中点击“启用”，启用规则；
7. 规则创建好后，点击“查看”，可获取推送的Topic，以及建立MQTT连接认证需要的AccessKey和AccessSecret。注意，此AccessKey和AccessSecret不同于"应用开发指南"中的AccessKey，仅用于当前Topic数据获取。

接收数据操作步骤：

```
1. 构造clientId：rule:{AccessKey}
2. 构造username：{hashMethod}:{random}（hashMethod支持：HmacMD5、HmacSHA1、HmacSHA256和 HmacSHA512。
3. 计算password，password= hashMethod(AccessKey+AccessSecret+random)，加密秘钥填写AccessSecret。
4. 建立MQTT连接，进行接入认证。
5. 复制规则引擎中生成的Topic，并订阅该Topic。
6. 订阅成功后就可正常接收规则引擎推送来的数据。
```

【注意】

```
暂不支持自定义Topic。
注意区分clientId和Topic。
```

【设备操作】：

1. 设备登录；
2. 发送设备所有参数（电源开关信息 `power:1`；光照强度 `light：50`；湿度 `hum：23`）；
3. 发送设备所有参数（电源开关信息 `power：1`；光照强度 `light：95`；湿度 `hum：23`）；
4. 设备登出。

【数据结果】：

收到二组数据：

操作1对应的数据

```
{
    "iot_sys_devId": "8237948938472348",
    "iot_sys_event": "devLogin",
    "iot_sys_timestamp": 1576053309725
}
```

操作4对应的数据

```
{
    "iot_sys_devId": "8237948938472348",
```

```
    "iot_sys_event": "devLogout",
    "iot_sys_timestamp": 1576053324204
}
```

【对照分析】：

对比对照实例，可知：

1. 规则引擎的条件过滤功能使得设备的 `操作2` 和 `操作3` 数据被过滤；
2. 规则引擎的字段筛选功能使得仅获得到设备ID `iot_sys_devId` 、事件 `iot_sys_event` 和时间 `iot_sys_timestamp` 。

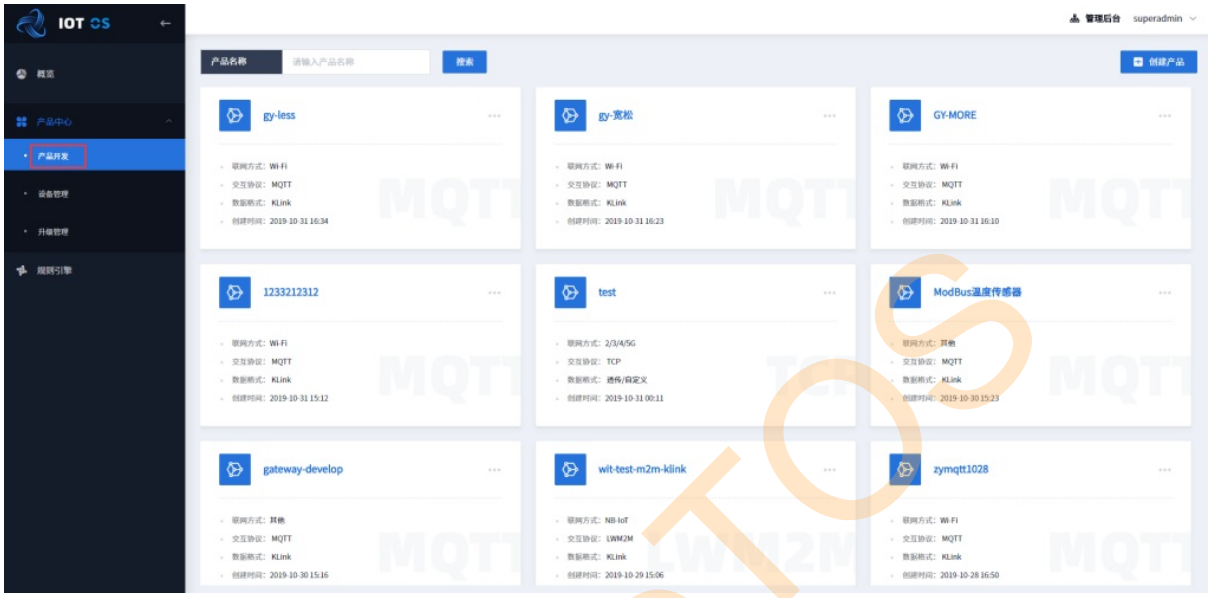
说明

此处实例只涉及到了部分筛选字段和部分过滤条件，可以在规则引擎中查看所有的字段和条件，以及不同设备操作对应推送消息包含的字段说明。

- 产品管理

产品管理

用户登录平台后，点击左侧导航栏的“产品中心”-“产品开发”菜单，可以看到账户下所创建的所有产品列表，在这个页面，用户可以创建新的产品，也可以编辑和删除已有产品。



- 添加协议参数

添加协议参数

“产品开发”页面，点击已创建的产品，进入到产品概览页面，点击“功能定义”标签，进入自定义参数页面，再点击自定义参数中的“创建参数”按钮，可添加自定义参数。

【说明】

1．参数名称：可录入中英文字符、数字，用来标识功能参数；
2．标识符：只能录入英文字符、数字，下划线，必须以字母开头，用于云端协议解析的参数识别标识。
3．数据类型：

数据类型	校验方式	说明
字符串	枚举校验	枚举所有上传的数值及注释，云端根据设置的枚举值校验上报或下发的数据，不在范围内会返回错误。
	长度校验	参数允许传递的最大、最小长度，云端根据设置的最大最小值校验上报或下发的数据，不在范围内会返回错误。
整数	枚举校验	枚举所有上传的数值及注释，云端根据设置的枚举值校验上报或下发的数据，不在范围内会返回错误。
	范围校验	设置参数最大最小值及注释，云端根据设置的最大最小值校验上报或下发的数据，不在范围内会返回错误。
浮点数	枚举校验	枚举所有上传的数值及注释，云端根据设置的枚举值校验上报或下发的数据，不在范围内会返回错误。
	范围校验	设置参数最大最小值及注释，云端根据设置的最大最小值校验上报或下发的数据，不在范围内会返回错误。

4．传输类型：
- 只上报：本参数只能出现在上行数据中，通过设备上传给云端的参数

- 只下发：本参数只能出现在下行数据中，通过云端下发给设备的数据
- 可上报可下发：本参数既可上报也可以下发

参数创建好后，在“自定义参数”页面可以看到所有已创建的参数。点击某个参数右侧的“修改”按钮，可修改参数的信息；点击“复制”按钮，可快速复制参数；点击“删除”按钮，可删除参数。

- 添加协议命令

添加协议命令

用户在“功能定义”页面，点击“命令”标签页，再点击“创建命令”按钮，可添加命令内容。页面选择好命令类型后，待选参数框会显示所有可选择的参数，勾选命令中要传递的参数，点击“>”，可将选择的参数批量移到已选参数中，实现产品协议的快速定义。

【说明】

1．参数名称：可录入中英文字符、数字，用来标识命令功能；
2．标识符：只能录入英文字符、数字，用于云端协议解析的命令识别标识；
3．命令类型：分为上报帧和下发帧，用来区分命令帧的通讯类型；
4．待选参数：根据所选的命令类型，显示可选择的所有参数。命令类型为上报帧时，仅传输类型为“只上报”或“可上报可下发”的参数会显示在待选参数中；
5．选择参数：支持将左侧“待选参数”框中的参数批量移动到“已选参数”框。

- 数据解析

数据解析

针对数据传输格式为自定义格式的产品，在“功能定义”页面定义好产品协议后，需要提供数据解析脚本。主要作用将上行的原数据转换成物联网平台定义的标准格式（KLink JSON），同时将下行的标准格式（KLink JSON）数据解析为设备自定义数据格式。

登录IoT OS后，单击左侧菜单栏的“产品开发”，点击要添加数据解析脚本的产品，点击“数据解析”标签。页面如下所示，提供了编辑脚本的输入框，以及模拟数据输入框。

调试上行数据：

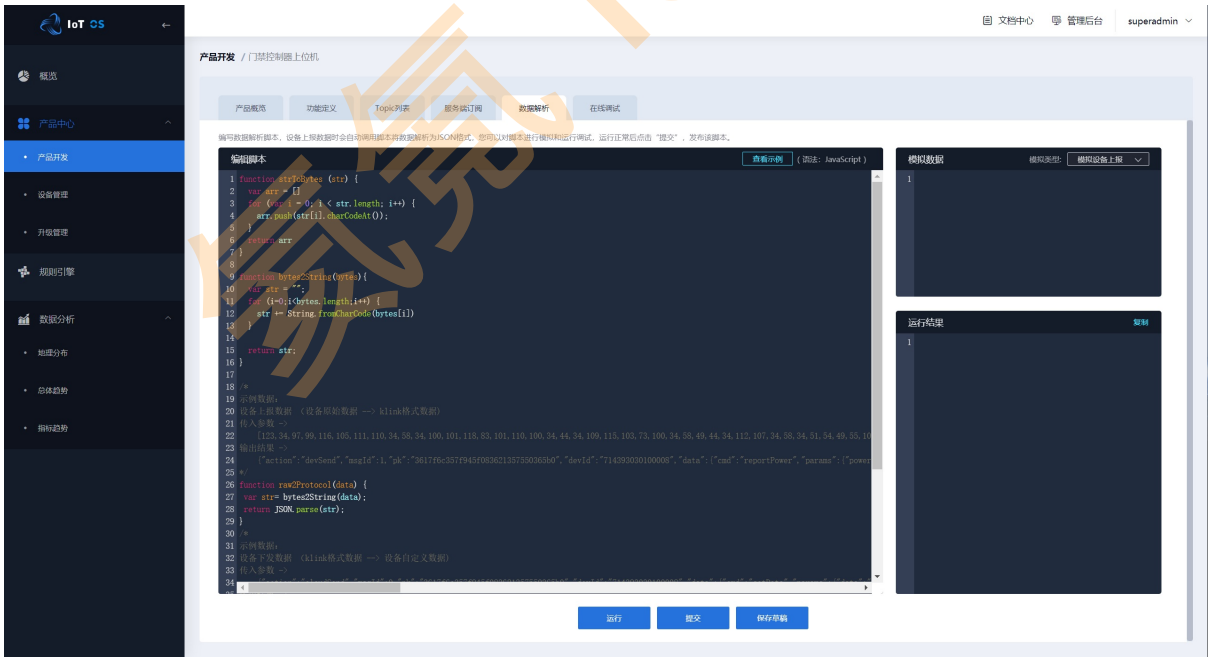
1．点击示例，查看示例代码；
2．在“编辑脚本”框中编写数据解析脚本；
3．在“模拟数据”框中填写设备上报的原始数据，点击“运行”；
4．在“运行结果”中查看解析出来的数据是否是平台定义标准的数据。

调试下行数据：

1．在“编辑脚本”框中编写数据解析脚本；
2．在“模拟数据”框中填写平台定义的标准下行数据，点击“运行”；
3．在“运行结果”中查看解析出来的数据是否是设备端能处理的原始数据格式。

提交解析脚本：

上下行数据解析都调试通过后，点击“提交”按钮，将解析脚本提交。数据解析脚本提交成功后，设备上报数据时会自动调用脚本将数据解析成标准格式；下发数据给设备端时，也会调用脚本将数据解析成设备端自定义数据格式。



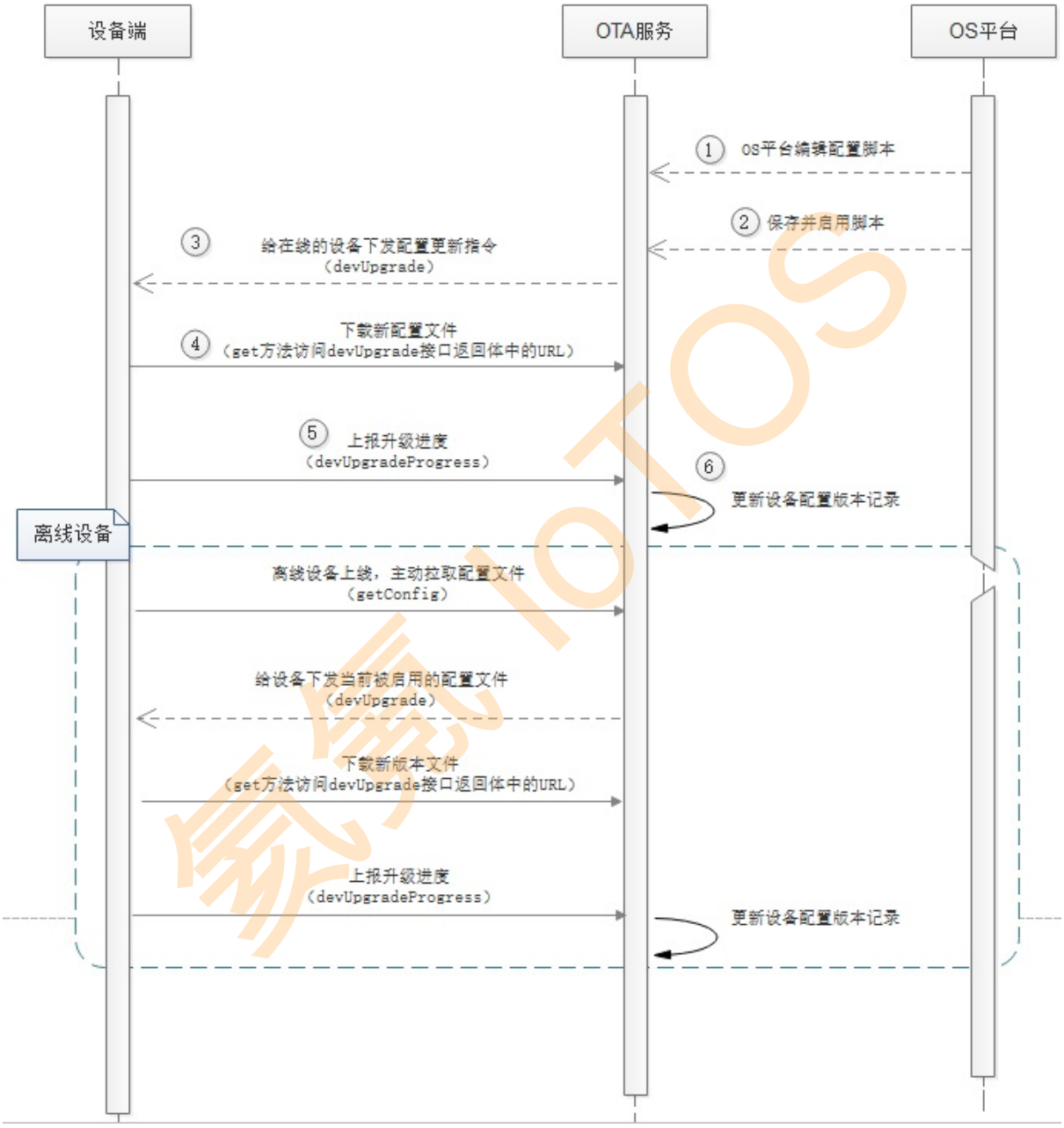
【说明】

1．数据解析脚本中必须同时包含上行，下行数据的解析方法；
2．点击数据解析脚本编写格式示例，查看示例代码。

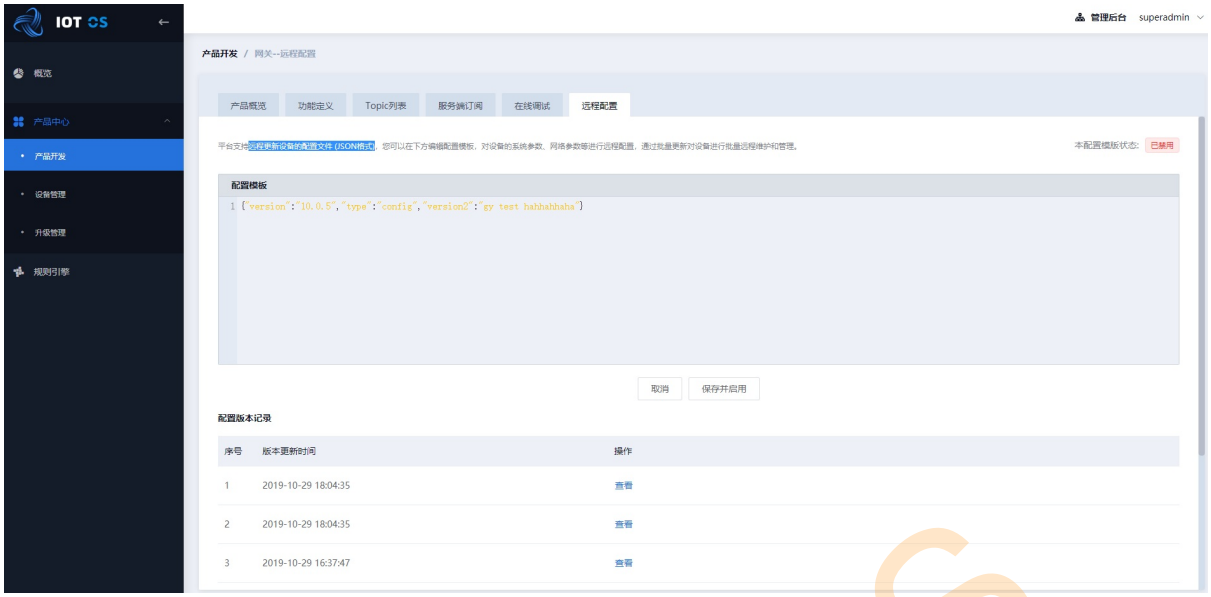
- 远程配置

远程配置

针对网关类型的产品，平台支持远程更新网关设备的配置文件 (JSON格式)，前提是网关所属的产品在创建时选择了允许远程配置。



登录平台后，选择左侧导航栏中的“产品开发”，单击要更新配置文件的设备所属的产品，点击“远程配置”标签。在页面配置模板中填写配置文件的脚本（JSON格式），点击“保存并启用”。远程配置规则将被启用，启用后，已在线的设备会马上收到升级任务，离线设备重新上线后需要通过“getConfig”指令主动获取更新任务。远程配置启用后，在“配置版本记录”会记录该条启动记录，用户可查看已启用过的配置任务，点击“恢复此版本”按钮，可查看并启用历史版本。



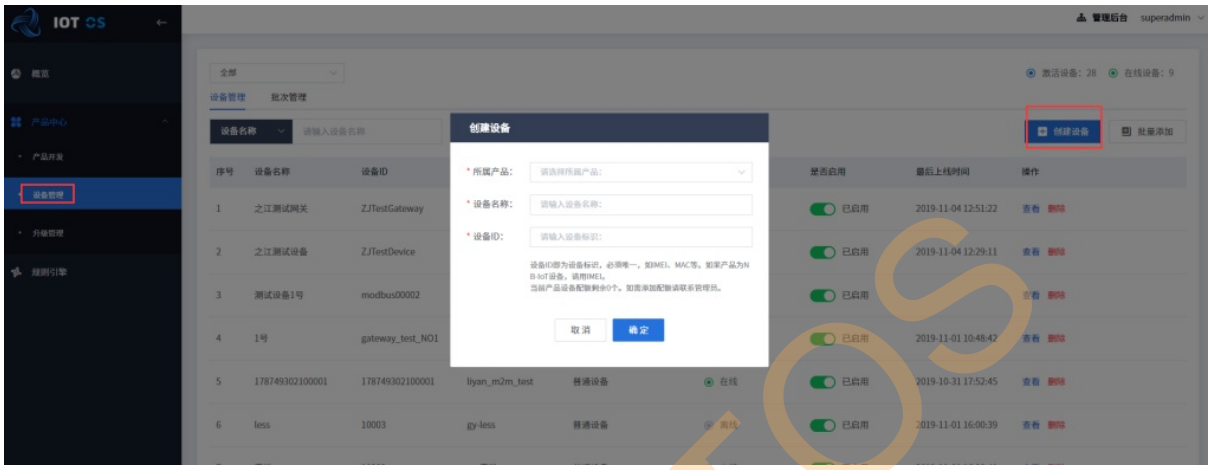
【说明】

一个产品下只允许同时启用一条配置任务，当用户保存并提交了新的配置模板时，原来启用的任务就会被覆盖。

- 创建单个设备

创建单个设备

产品创建成功后，用户需要先将设备信息添加到平台，进行设备注册，设备才能正常连接IoT OS。IoT OS支持创建单个设备，也支持批量创建设备。 登录IoT OS后，在左侧导航栏中选择“设备管理”，在设备管理页面，单击“创建设备”，在创建设备对话框输入设备信息，点击确定即可完成添加。



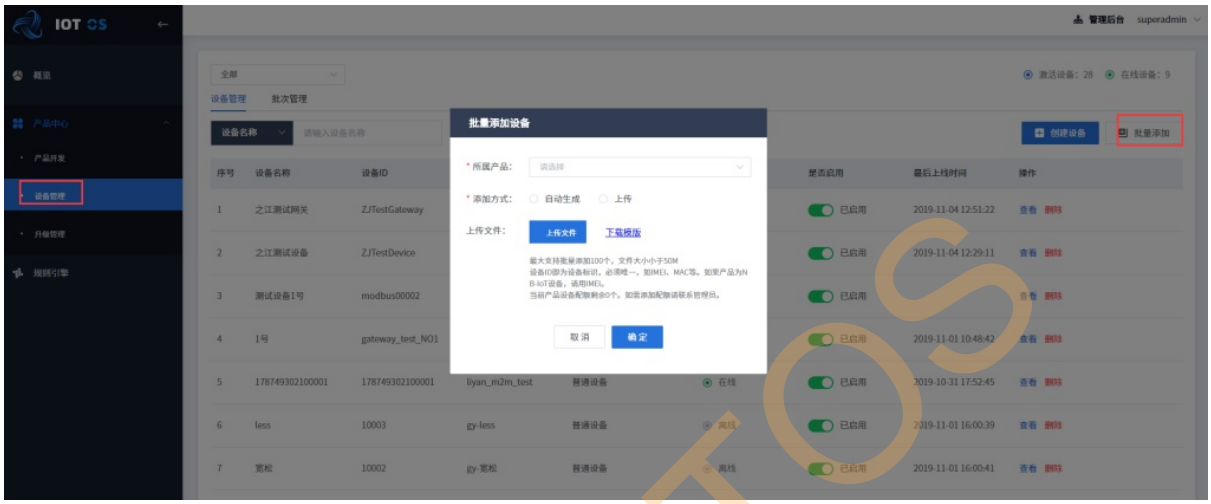
【说明】

1．所属产品：设备所属产品，新创建的设备会继承该产品定义好的功能和特性；
2．设备名称：支持2~32长度的字符；
3．设备ID：设备唯一标识，NB-IoT设备在该处填写设备的IMEI；
4．配额剩余数：所选产品下还可添加的设备个数。新建的产品默认最多允许添加10个设备。选择所属产品后，对话框下面的文案中会提示该产品还剩余的设备配额，若配额为0，该产品将不允许再添加设备，需要向平台管理员申请增加设备额度。

- 批量创建设备

批量创建设备

IoT OS支持通过导入CSV文件或服务端自动生成两种方式批量添加设备。登录IoT OS后，在左侧导航栏中选择“设备管理”，在设备管理页面，单击“批量添加”，在批量添加设备的对话框中，选择批量生成的方式，填入设备信息，点击确定，可批量创建设备。



【说明】

1．所属产品：设备所属产品，新创建的设备会继承该产品定义好的功能和特性；
2．添加方式：
【自动生成】无需指定设备ID，填写设备数量后，系统自动生成设备ID；
【上传】需要指定要添加的所有设备ID。单击“下载模板”，下载表格模板，在模板中填写设备ID，然后将填写好的表格上传到控制台。

3．批量添加额度：单次批量添加，最大允许添加100个设备，超过100个设备，需要分批次添加。

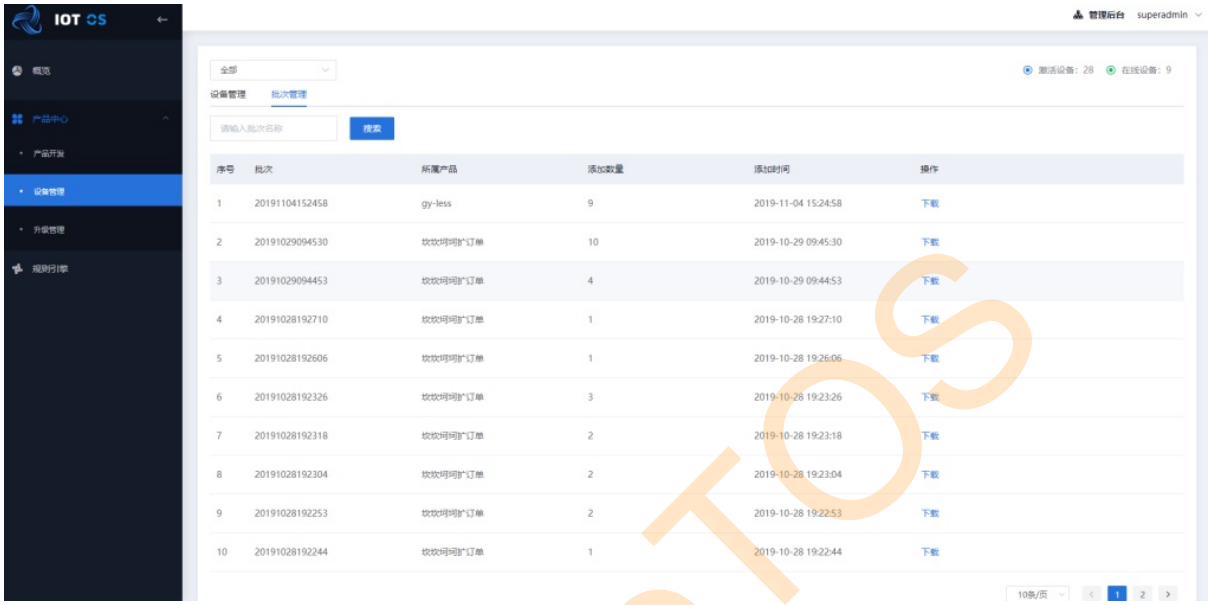
【注意】

若所选产品剩余的设备额度小于要添加的设备数量，只能添加剩余额度数量的设备，超出的部分会添加失败。

- 设备导出

设备导出

对于批量添加的设备，平台支持批量导出。在“设备管理”模块，点击“批次管理”标签，可查看已经生成的所有批次设备，单击某个批次左侧的“下载”按钮，可批量导出该批次的设备信息。



- 禁用启用与删除

禁用启用与删除

用户可以对自己创建的产品进行禁用启用以及删除的操作。

进入平台"产品开发"-"设备管理"，可见如图所示页面。

在用户想要禁用的设备中点击【1】处按钮，并点确认即可对产品进行禁用。重复操作可重新开启设备。

若用户想要删除设备，则点击【2】处删除按钮，并点击确认后即可删除设备。

【注意】

1.删除设备后不能恢复。
2.同一产品下的所有设备删除后才能删除产品。

- 设备详情

设备详情

设备添加成功后，在设备列表中找到添加的设备，单击“查看”，跳转到设备信息页面，在该页面可查看设备所属产品PK、设备devSecret等信息。

参数	说明
设备ID	设备唯一识别码，设备跟云端通信时必须提供。
产品PK	设备所隶属产品的Key，设备跟云端通信时必须提供。
devSecret	物联网平台为设备颁发的设备密钥，用于认证加密。

- 查看设备影子

查看设备影子

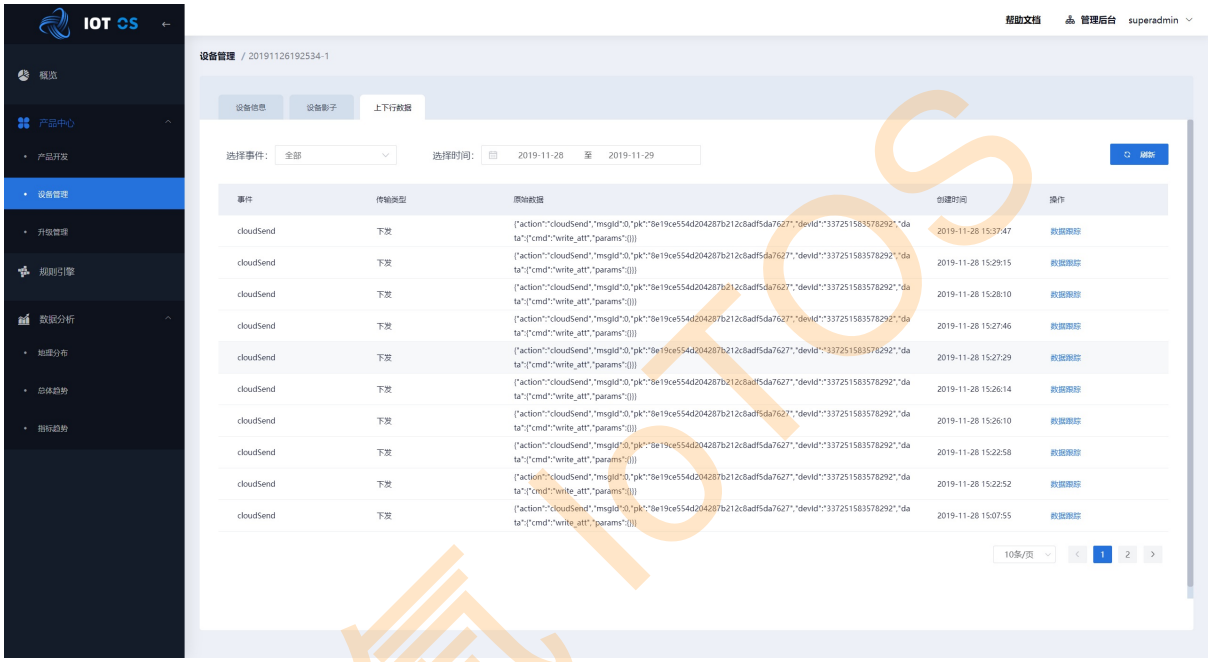
IoT OS提供设备影子功能，用于缓存设备状态，设备影子中记录了设备最新上报给云端的状态。登录IoT OS后，在“设备管理”中找到要查看设备影子的设备，点击“查看”按钮，点击“设备影子”标签，即可查看设备影子。

- 上下行数据
  - 上下行流程
    - 数据跟踪-下发数据流程
    - 数据跟踪-上报流程

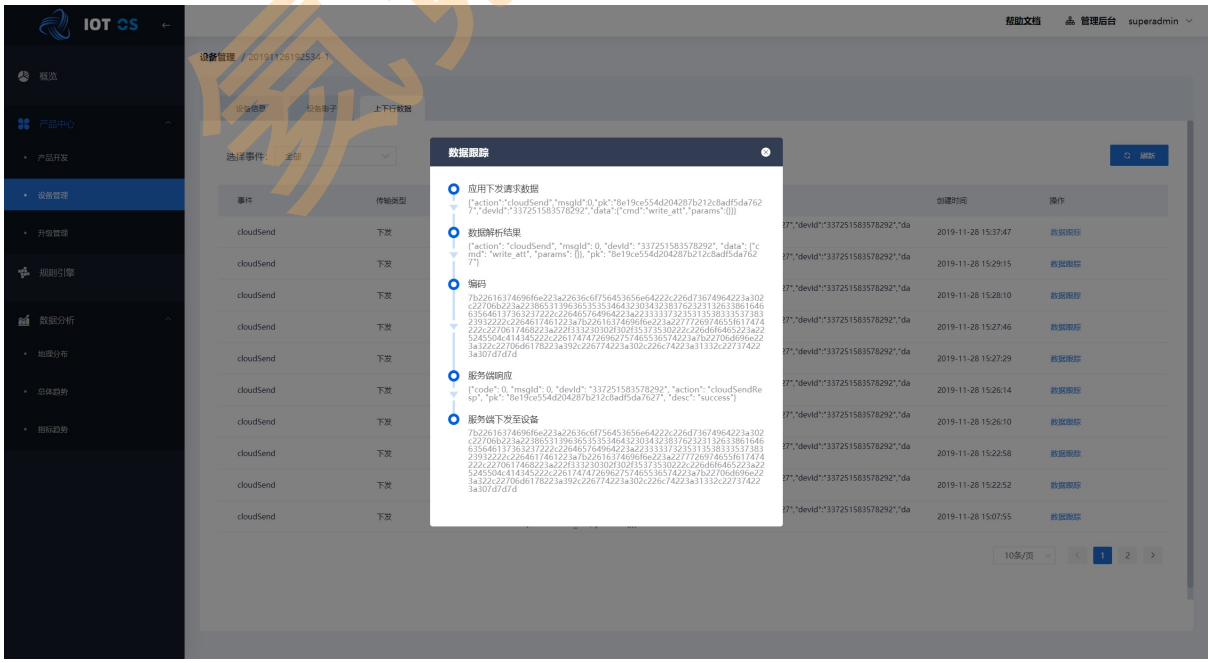
上下行数据

IoT OS缓存了设备的所有上下行数据，方便用户查看设备跟云端通信的历史记录。

登录IoT OS后，在“设备管理”中找到要查看上下行数据的设备，点击“查看”按钮，点击“上下行数据”标签，即可查看设备的所有上下行记录。



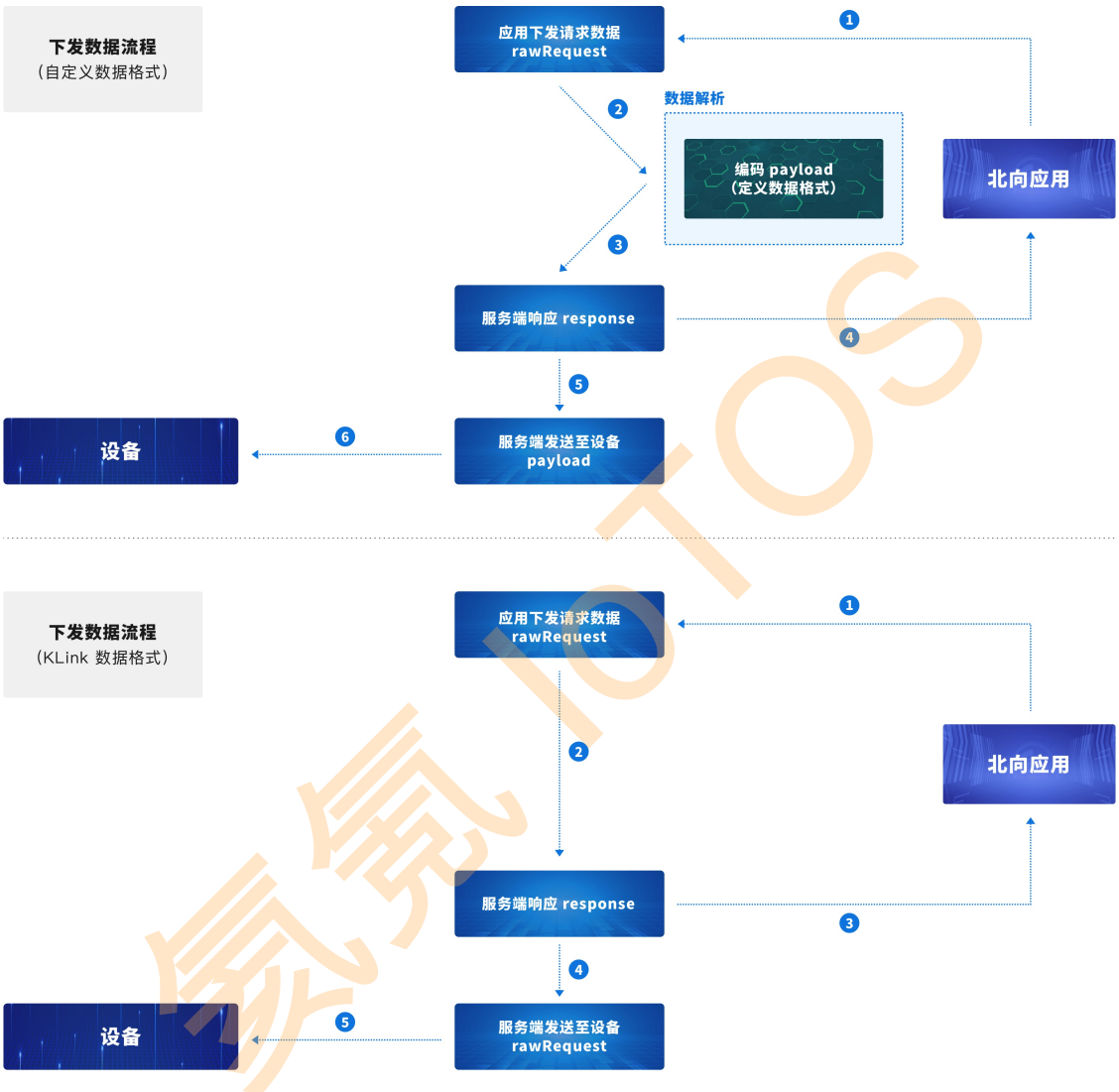
点击操作栏中“数据跟踪”，即可查看数据流转过程



上下行流程

数据跟踪-下发数据流程

(frameType==DEV_DOWN)

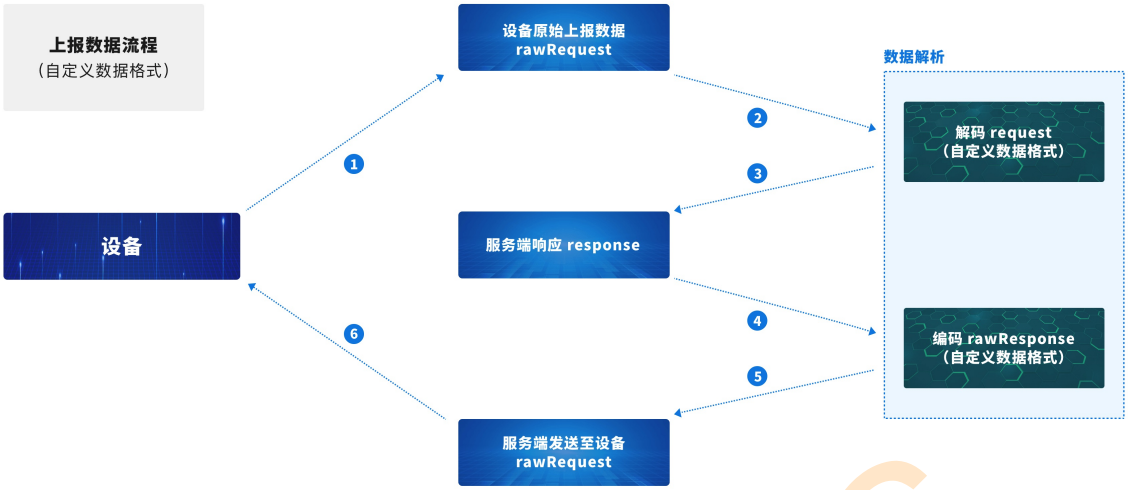


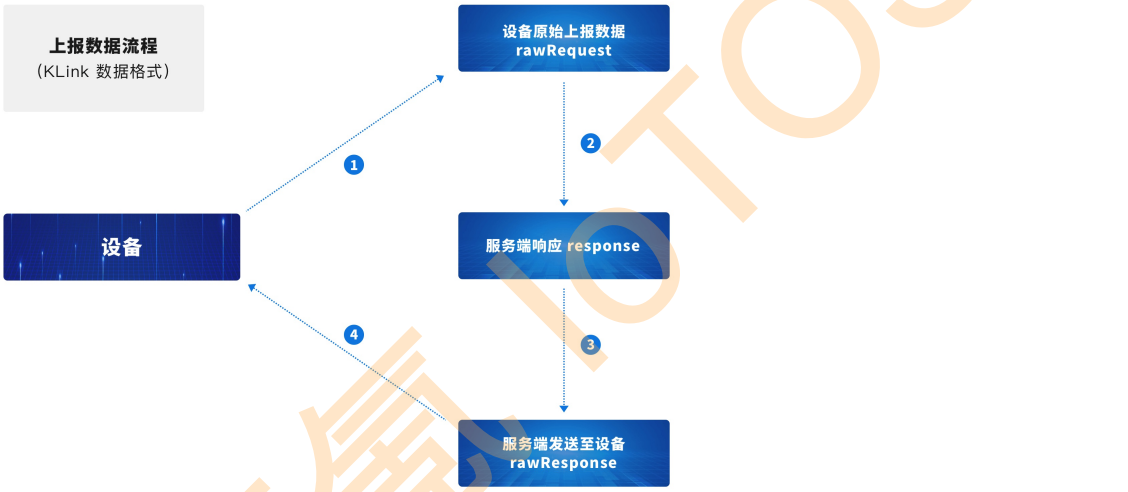
【说明】

```
编码payload：当下发的数据为自定义格式时，需要将KLink编码为自定义格式；
服务端响应responce：判断下发是否成功，失败会返回具体错误码。
```

数据跟踪-上报流程

(frameType==DEV_UP)





【说明】

解码request：自定义格式下，需要将上报的数据进行解码；
服务端响应responce：判断上报是否成功，失败会返回具体错误码；
编码rawResponce：自定义格式下，需要对数据进行编码后再发送给设备。

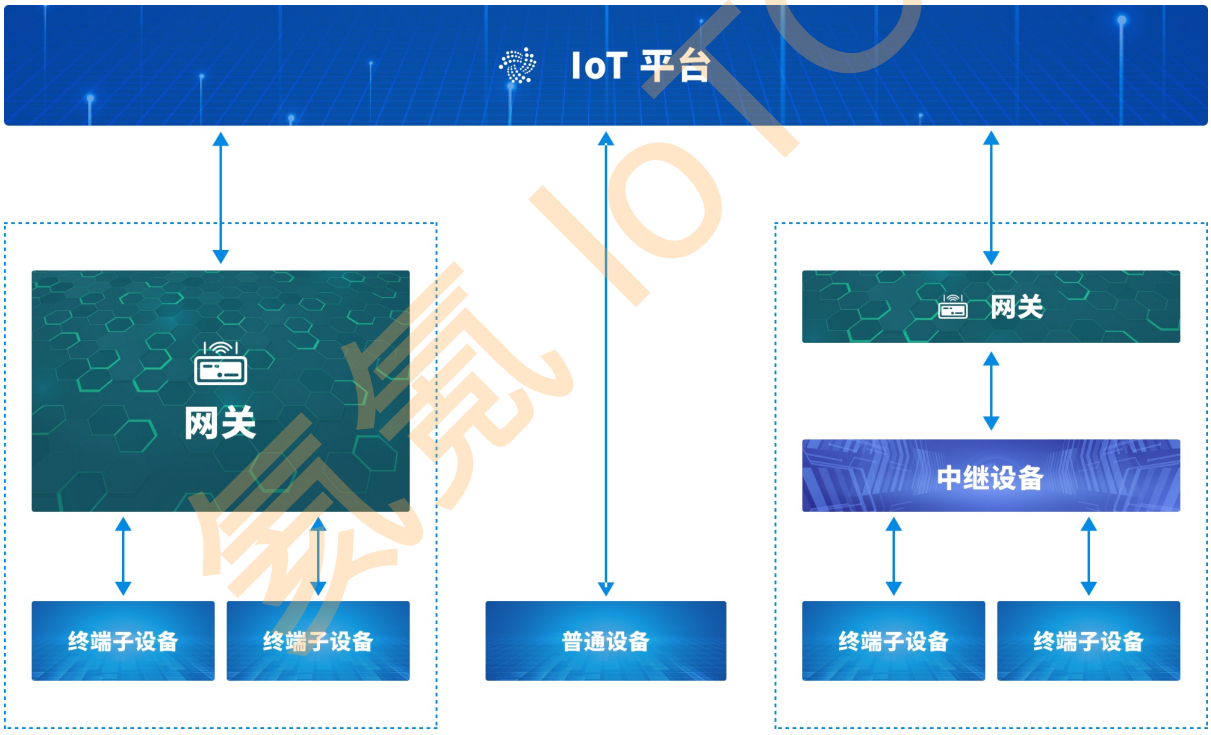
- 网关与设备

网关与设备

创建产品与设备时，需要选择节点类型。平台目前支持三大类节点类型：设备、网关、中继。其中设备细分为接入网关设备（终端子设备）和不接入网关设备（普通设备）。

名称	解释
设备	指不能挂载子设备的设备。设备可以直连物联网平台，也可以作为网关的子设备，由网关代理连接物联网平台。
普通设备	不能挂载子设备，可以直接连接物联网平台。
终端子设备	作为网关的子设备，由网关代理连接物联网平台。
中继	可以挂载子设备，但必须由网关代理连接物联网平台，类似于主机设备。
网关	指可以挂载子设备的直连设备。网关可以管理子设备、可以维持与子设备的拓扑关系，并将该拓扑关系同步到云端。

设备与平台连接拓扑图如下：

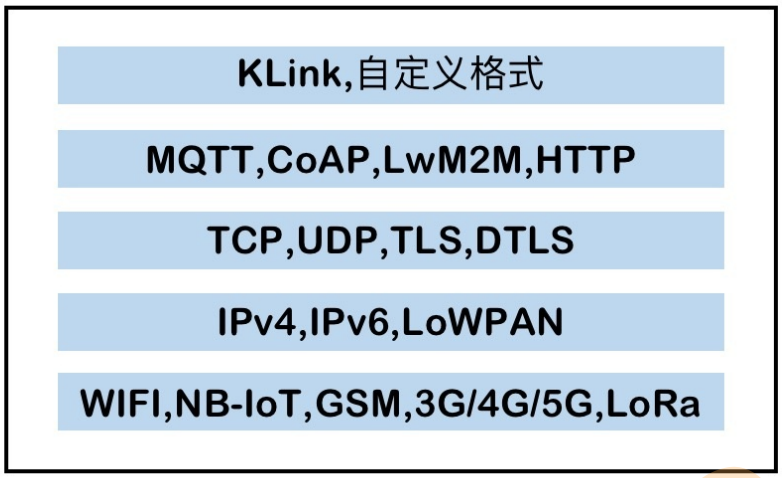


- KLink
  - 简介
  - 协议特点
  - 重要说明
  - 文档格式格式说明
  - KLink指令详解
    - 指令列表
    - 设备上报数据
    - 云端回复设备上报数据
    - 网关批量上报设备数据
    - 云端回复网关批量上报设备数据
    - 云端给设备下发指令
    - 设备回复云端下发指令
    - 设备上报固件信息
    - 云端回应设备上报固件信息
    - 云端给设备下发固件升级指令
    - 设备回应固件升级指令
    - 设备上报升级进度
    - 动态注册设备
    - sign算法说明
    - 云端回复动态注册设备
    - 网关添加拓扑关系（子设备）
    - 添加拓扑子设备 sign 计算方法
    - 云端回复网关添加拓扑关系（子设备）
    - 网关查询拓扑关系（子设备）
    - 云端回复网关查询拓扑关系（子设备）
    - 网关删除拓扑关系（子设备）
    - 云端回复网关删除子设备（子设备）
    - 子设备上线
    - 云端回复子设备上线
    - 子设备下线
    - 云端回复子设备下线

KLink

简介

KLink协议，是IoT OS为开发者提供的设备与云端之间数据交换的标准协议，采用JSON格式。



协议特点

每一帧应都带上msgId，该字段用于设备端和云端做消息请求匹配，如果短时间内msgId一样即可认为是一对请求和回复；如果不填则每条消息都默认为0。云端允许范围是 有符号 int64范围，设备端可以根据自身资源情况设计，比如可以设计成2Byte无符号范围；如果超出int64范围则会造成数值溢出，导致msgId回应不准确。

重要说明

云端对设备发送的数据中一定包含 pk和devId，如果设备不需要这些信息，可以使用自定义解码器进行过滤后再转发。

文档格式格式说明

- 为了文档清晰，下面的 json 指令均进行了格式化输出，在实际发送数据的时候应该将json数据压缩成一行并且没有多余空格的数据，只保留最后一个换行符发送。比如下面的指令只是为了阅读方便

```
{
    "action": "devSend"
}
```

设备实际发送的的数据应该是

```
{"action":"devSend"}
```

KLink指令详解

指令列表

指令功能	发送形式	详情
设备上报数据	D => C	devSend
云端回复设备上报数据	C => D	devSendResp

网关批量上报设备数据	D => C	batchDevSend
云端回复网关批量上报设备数据	C => D	batchDevSendResp
云端给设备下发指令	C => D	cloudSend
设备回复云端下发指令	D => C	cloudSendResp
设备上报固件信息	D => C	reportFirmware
云端回应设备上报固件信息	C => D	reportFirmwareResp
云端给设备下发固件升级指令	C => D	devUpgrade
设备回应固件升级指令	D => C	devUpgradeResp
设备上报升级进度	D => C	devUpgradeProgress
动态注册设备	D => C	register
云端回复动态注册设备	C => D	registerResp
网关添加拓扑关系（子设备）	D => C	addTopo
云端回复网关添加拓扑关系（子设备）	C => D	addTopoResp
网关查询拓扑关系（子设备）	C => D	getTopo
云端回复网关查询拓扑关系（子设备）	D => C	getTopoResp
网关删除拓扑关系（子设备）	D => C	delTopo
云端回复网关删除子设备（子设备）	C => D	delTopoResp
子设备上线	D => C	devLogin
云端回复子设备上线	C => D	devLoginResp
子设备下线	D => C	devLogout
云端回复子设备下线	C => D	devLogoutResp

设备上报数据

D => C

```
{
    "action": "devSend",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "data": {
        "cmd": "标识符",
        "params": {
            "power": 1,
            "light": 99.2
        }
    }
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devSend。
pk	否	string	要发送数据的设备产品PK，如果是子设备必填。

devId	否	string	要发送数据的设备ID，如果是子设备必填。
data	是	object	上报的指令和参数数据。
data.cmd	是	string	标识符
data.params	否	object	参数，如果只有指令，没有参数允许不填；否则应该填写该指令下的参数标识符。 可以少不能多，值做强较验（类型，长度等必须符合协议规定）。

注意： 如果是子设备上报数据，且是透传数据，需要在网关上统一解码成云端要求的格式，否则云端解码失败。

云端回复设备上报数据

C => D

```
{
    "action": "devSendResp",
    "pk": "pk",
    "devId": "devId",
    "msgId": 1,
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devSendResp。
pk	是	string	设备所属产品PK。
devId	是	string	设备ID。
code	是	uint	如果没有错误回复 0。

网关批量上报设备数据

D => C

```
{
    "action": "batchDevSend",
    "msgId": 1,
    "data": [{
        "pk": "pk",
        "devId": "devId",
        "cmd": "标识符",
        "params": {
            "power": 1,
            "light": 99.2
        }
    },{
        "pk": "pk1",
        "devId": "devId2",
        "cmd": "标识符",
        "params": {
            "power": 1,
            "light": 99.2
        }
    }]
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 batchDevSend

data.pk	是	string	要发送数据的设备的 productKey
data.devId	是	string	要发送数据的设备的 devId
data.cmd	是	string	标识符
data.params	否	object	参数，如果只有指令，没有参数允许不填；否则应该填写该指令下的参数标识符。

云端回复网关批量上报设备数据

C => D

```
{
    "action": "batchDevSendResp",
    "pk": "pk",
    "devId": "devId",
    "msgId": 1,
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 batchDevSendResp。
pk	是	string	设备所属产品PK。
devId	是	string	设备ID。
code	是	uint	如果没有错误回复 0。

云端给设备下发指令

云端给设备（包括子设备）下发指令，设备需要处理所接收的数据。

C => D

```
{
    "action": "cloudSend",
    "pk": "pk",
    "devId": "devId",
    "msgId": 1,
    "data": {
        "cmd": "cmdFlag",
        "params": {
            "power": 1,
            "light": 99.2
        }
    }
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 cloudSend
pk	是	string	设备所属产品PK。
devId	是	string	要控制的设备ID。
data	是	object	指令数据。

data.cmd	是	string	命令标识符。
data.params	否	object	参数，如果只有指令，没有参数允许不填，否则应该填写该指令下的参数标识符。

设备回复云端下发指令

D => C

```
{
    "action": "cloudSendResp",
    "pk":"pk",
    "devId":"devId",
    "msgId": 1,
    "code": 0
}
```

设备必须回复，如果没有错误是0；错误码在产品上增加说明。 如果云端没有收到回复则认为是设备没有收到控制指令，控制失败。

参数	必填	类型	说明
action	是	string	动作，固定为 cloudSendResp。
pk	否	string	设备所属产品PK，如果是子设备必填。
devId	否	string	要控制的设备ID，如果是子设备必填。
code	是	uint	如果没有错误回复 0。

设备上报固件信息

D => C

```
{
    "action": "reportFirmware",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "type": "module/mcu/config",
    "version": "1.2.1"
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 reportFirmware。
pk	否	string	设备所属产品PK。
devId	否	string	要控制的设备ID。
type	是	string	module/mcu/config。
version	是	string	版本，格式随意，比如可以是 a.b.c，也可以是 1.0.1。

云端回应设备上报固件信息

C => D

```
{
```

```
    "action": "reportFirmwareResp",
    "pk":"pk",
    "devId":"devId",
    "msgId": 1,
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 reportFirmwareResp。
pk	否	string	设备所属产品PK， 如果是子设备必填。
devId	否	string	要控制的设备ID，如果是子设备必填。
code	是	uint	如果没有错误回复 0。

云端给设备下发固件升级指令

C => D

```
{
    "action": "devUpgrade",
    "msgId": 1,
    "pk":"pk",
    "devId":"devId",
    "url": "http://xx.xx/ss.bin",
    "md5": "md5",
    "type": "module/mcu/config",
    "version":"currentVersion"
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devUpgrade。
pk	是	string	要控制的设备所属产品PK。
devId	是	string	要控制的设备ID。
url	是	string	固件下载地址(http，非https)。
md5	是	string	固件的md5值。
type	是	string	更新类型 module(模组)，mcu(mcu)，config(远程配置)。
version	是	string	目标版本。

设备回应固件升级指令

设备必须回复，否则云端认为下发指令失败，设备未收到控制指令。

D => C

```
{
    "action": "devUpgradeResp",
    "pk":"pk",
    "devId":"devId",
    "type": "module/mcu",
    "msgId": 1,
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devUpgradeResp。
pk	否	string	要控制的设备的所属产品PK， 如果是子设备必填。
devId	否	string	要控制的设备ID，如果是子设备必填。
code	是	uint	如果没有错误回复 0。
type	是	string	升级类型 module，mcu。

设备上报升级进度

注意 只是记录升级进度，具体升级到什么版本还需要设备主动上报，参见上报固件版本

D => C

```
{
    "action": "devUpgradeProgress",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "progress": 19,
    "type": "module/mcu/config",
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devUpgradeProgress。
pk	否	string	要控制的设备的所属产品PK， 如果是子设备必填。
devId	否	string	要控制的设备ID，如果是子设备必填。
type	是	string	升级类型 module，mcu，config。
code	是	uint	如果没有错误回复 0。
progress	是	uint	取值范围 [0,100]。

如果code=0，progress=100 只能代表设备端升级完成，真正升级完成必须是上报的版本和预期的一样。设备升级完成，需要再次发送上报固件信息到云端以便更新云端设备固件版本信息，如果版本号跟预期不一样算作失败处理。

动态注册设备

D => C

```
{
    "action": "register",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "random": "random",
    "hashMethod": "method",
    "sign": "sign"
}
```

参数	必填	类型	说明

action	是	string	动作，固定为 register。
pk	是	string	要注册的设备的所属产品PK。
devId	是	string	要注册的设备ID。
sign	否	string	要注册的设备的产品认证签名。
random	否	string	设备加密校验，随机字符串，可以使用当前时间毫秒数的字符串，也可以使用uuid生成的，也可以是 math.random()等等。

hashMethod | 否 | string | hash 方法，选填 `HmacMD5` 、 `HmacSHA1` 、 `HmacSHA256` 或者 `HmacSHA512` 。

注意 在产品上设置有安全开关：动态注册设备是否需要鉴权，如果开启则 sign，random 和 hashMethod 字段必填，关闭则不需要填写。开启能够增强注册设备的安全性，建议开启。

sign算法说明

sign=hash(pk+productSecrect+random)，加密密钥（密码）填写 productSecrect

其中 `hash` 算法支持： `HmacMD5` 、 `HmacSHA1` 、 `HmacSHA256` 和 `HmacSHA512` 。

比如 pk= `pk123` ，productSecrect= `excvw2cw` ，random= `QWExm` ，则要hash的值应该为 `pk123excvw2cwQWExm` 使用不同 hash 算法得到的值如下，可以使用在线工具测试 进行测试。 使用 hash 算法得到sign后，需要使用对应的 toHex 方法将其编码为 hex string。

- HmacMD5: fea83bd79fb090b1529f1468bb1051f7
- HmacSHA1: 2c32e0bfbdee359eb71afb9efeb7ad922e1321c7
- HmacSHA256: 09e17c16188da122ff5cf6dd384df44260f546c5a38b6764c7e9f4b05f8e2452
- HmacSHA512: 0009dcd96b7f0d6c5667aa0a1b0aa8a354391598d7375152342c37a30fab5ec2ba17f85b9006dded114181447e29bb9ad616f61c24e77a103c99b9f25ccc9f59

云端回复动态注册设备

云端允许多次调用激活指令，注册成功后返回该设备的信息。

C => D

```
{
    "action": "registerResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "devSecret": "devSecret",
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 registerResp。
pk	是	string	设备所属产品PK。
devId	是	string	设备ID。
devSecret	是	string	设备的密钥，添加拓扑关系的时候需要使用。
code	是	uint	如果没有错误回复 0。

网关添加拓扑关系（子设备）

设备如果要构建自身和子设备的拓扑关系，同时，这种拓扑关系也是一种约束，只有在这个网关下的子设备才允许通过该网关上报数据，所以需要向云端发送添加子设备的指令，将子设备添加到网关拓扑关系中。

1. 添加拓扑的时候，如果云端没有注册该子设备，则会自动注册子设备。
2. 添加关系的时候，云端会自动设置子设备状态为在线状态。
3. 支持一次性添加一个
4. 支持多级拓扑关系，如下图
5. 允许重复请求（防止云端添加成功，网关本地没有添加成功）
6. 网关应该等待云端回复网关添加子设备返回信息，code=0的时候才将子设备信息挂载到自身下。

```
        dev1            ---|  在该拓扑关系中，加上网关（顶层dev1）设备
       /   \                |  一共有4层，最多支持5层。
    dev2   dev3             |
           /   \            |  dev1 是 dev2 和 dev3 的网关节点；
        dev4   dev5         |  反过来说，dev2 和 dev3 是 dev1 子设备节点。
        /                   |
       dev6             ----| 其他节点同理；下面的说明会用到 网关节点 和 子设备节点 两个概念。
```

注意：

- 不允许两个设备相互设置为子设备
- 子设备层级最多支持5级
- 整个网关所包含的设备最多支持1000个
- 添加的子设备必的父级设备必须已经挂载到该拓扑中

D => C

```
{
    "action": "addTopo",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "sub": {
        "pk": "pk",
        "devId": "devId",
        "random": "random",
        "sign": "sign"
    }
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 addTopo。
pk	否	string	父级设备节点所属产品PK，如果是子设备必填。
devId	否	string	父级设备节点设备ID，如果是子设备必填。
sub	是	object	子设备节点。
sub.pk	是	string	子设备节点所属产品PK。
sub.devId	是	string	子设备节点设备ID。
sub.random	否	string	同设备登录认证的 random 意义。
sub.sign	否	string	子设备认证签名，算法说明。 MQTT登录认证
sub.hashMethod	否	string	hash 方法。

注意 子设备上有安全开关：添加拓扑关系，是否关闭安全认证，如果关闭，random、sign和hashMethod不需要天，开启则必填。出于安全考虑，建议开启安全认证。

添加拓扑子设备 sign 计算方法

设备链接MQTT认证方式一致

云端回复网关添加拓扑关系（子设备）

C => D

```
{
    "action": "addTopoResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "sub": {
        "pk": "pk",
        "devId": "devId"
    },
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 addTopoResp。
pk	是	string	父设备节点所属产品PK。
devId	是	string	父设备节点设备ID。
sub	是	object	子设备节点信息。
sub.pk	是	string	子设备节点所属产品PK。
sub.devId	是	string	子设备节点设备ID。
code	是	uint	如果没有错误回复 0。

网关查询拓扑关系（子设备）

注意 只支持查询直属下级的设备，不支持查询所有设备。

C => D

```
{
    "action": "getTopo",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId"
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 addTopo。
pk	否	string	父级设备节点所属产品PK，如果是子设备必填。
devId	否	string	父级设备节点设备ID，如果是子设备必填。

云端回复网关查询拓扑关系（子设备）

D => C

```
{
    "action": "getTopoResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "subs": [
        {
            "pk": "pk",
            "subPk": "subPk",
            "devId": "subDevId"
        }
    ]
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 addTopo。
pk	否	string	父级设备节点所属产品PK，如果是子设备必填。
devId	否	string	父级设备节点设备ID，如果是子设备必填。
subs	是	array	直属下级的所有设备，如果没有则数组为空。

网关删除拓扑关系（子设备）

前置条件：网关设备已经登录并保持连接。

网关删除拓扑关系，仅仅是在逻辑上解除了与子设备的关系，并没有删除设备注册信息。

D => C

```
{
    "action": "delTopo",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "sub": {
        "pk": "pk",
        "devId": "devId"
    }
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 delTopo。
pk	否	string	父设备节点所属产品PK，如果是子设备必填。
devId	否	string	父设备节点设备ID，如果是子设备必填。
sub	是	object	子设备节点信息。
sub.pk	是	string	子设备节点所属产品PK。
sub.devId	是	string	子设备节点设备ID。

云端回复网关删除子设备（子设备）

C => D

```
{
    "action": "delTopoResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "sub": {
        "pk": "pk",
        "devId": "devId"
    },
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 delTopoResp。
pk	是	string	父设备节点所属产品PK。
devId	是	string	父设备节点设备ID。
sub	是	object	子设备节点信息。
sub.pk	是	string	子设备节点所属产品PK。
sub.devId	是	string	子设备节点设备ID。
code	是	uint	如果没有错误回复 0。

子设备上线

注意，该指令仅供子设备使用

前置条件：网关设备已经登录并保持连接。

D => C

```
{
    "action": "devLogin",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId"
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devLogin。
pk	是	string	子设备节点所属产品PK。
devId	是	string	子设备节点设备ID。

云端回复子设备上线

C => D

```
{
    "action": "devLoginResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
```

```
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devLoginResp。
pk	是	string	子设备节点所属产品PK。
devId	是	string	子设备节点设备ID。
code	是	uint	如果没有错误回复 0。

子设备下线

注意，该指令仅供子设备使用

前置条件：网关设备已经登录并保持连接。

D => C

```
{
    "action": "devLogout",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId"
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devLogout。
pk	是	string	子设备节点所属产品PK。
devId	是	string	子设备节点设备ID。

云端回复子设备下线

C => D

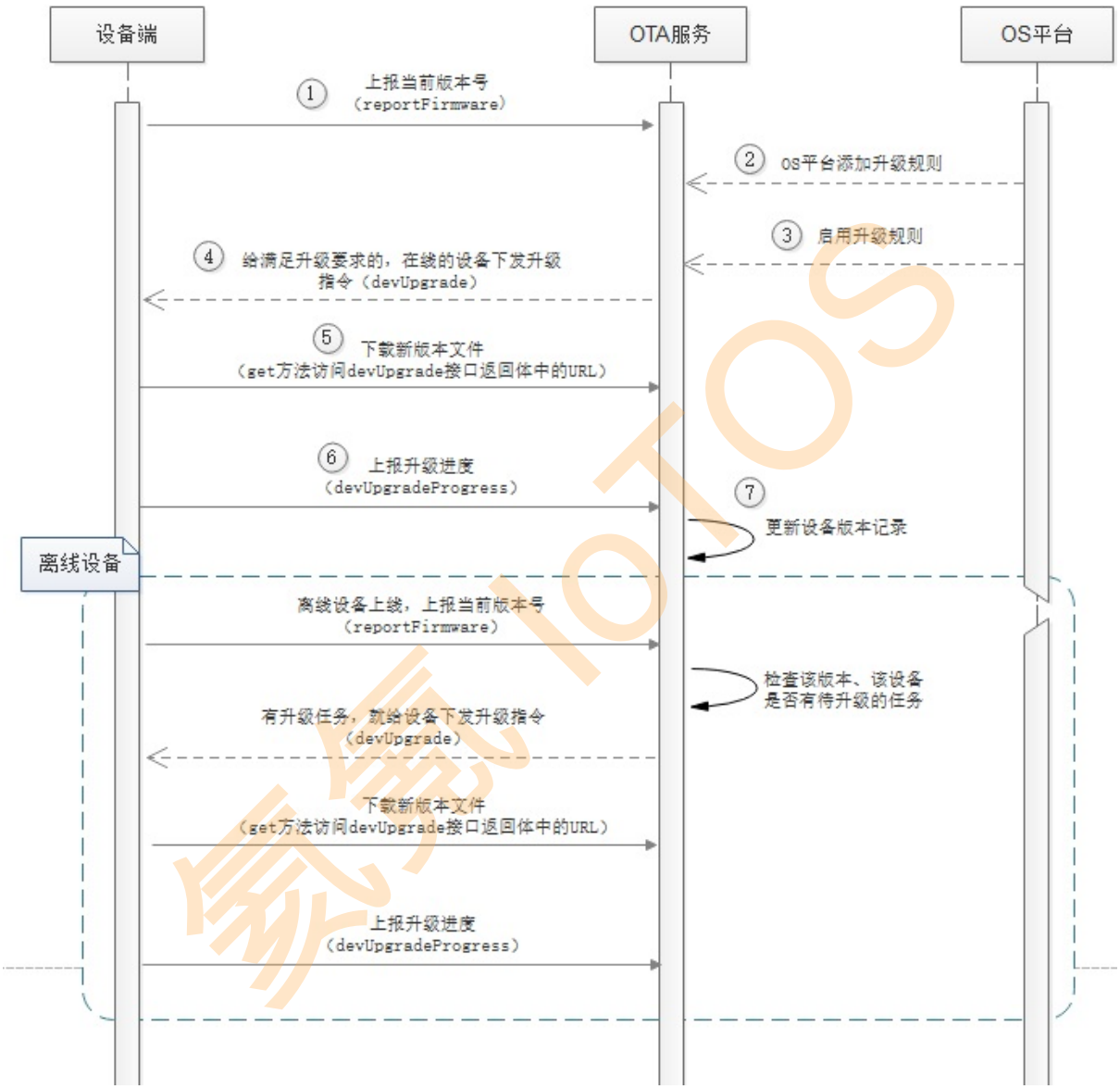
```
{
    "action": "devLogoutResp",
    "msgId": 1,
    "pk": "pk",
    "devId": "devId",
    "code": 0
}
```

参数	必填	类型	说明
action	是	string	动作，固定为 devLogoutResp。
pk	是	string	子设备节点所属产品PK。
devId	是	string	子设备节点设备ID。
code	是	uint	如果没有错误回复 0。

- 远程升级

远程升级

IoT OS提供固件和MCU的远程升级功能，使用该功能时需要先确保设备端支持OTA服务，硬件升级或软件升级，操作流程如下：



- 硬件升级

硬件升级

硬件升级是针对模组等固件版本升级。

IoT OS操作流程：

1．登录IoT OS；
2．选择左侧导航栏“升级管理”；
3．在“硬件升级”页面单击“添加升级包”；
4．在添加升级包对话框中，输入固件信息，点击确定；
5．在硬件升级列表中，单击升级规则的“启用”按钮，启用升级规则。

【说明】

1．所属产品：要升级的设备所隶属的产品。
2．目标版本：要升级到的新版本号。
3．升级包：新版本升级包。
4．待升级版本：
【全部版本】所属产品下所有版本的设备，包括没有版本的设备。
【指定版本】指定的要进行升级的版本，不是该版本的设备不会升级。
5．待升级设备：
【全部设备】产品下符合待升级版本条件的所有设备。
【指定设备】用来指定要升级的设备，设置后，仅添加了tid的设备才能升级到新版本。

【注意】

1．启用的升级规则在系统中一直是可执行的状态，有满足升级规则的设备上线，系统就会给该设备下发升级任务；
2．同一个产品下，可同时启动多条指定版本的升级任务，但同一版本不允许启用多条升级任务（例如：某款产品下，已存在一条A-B的升级任务被启动，再启动一条A-C的升级任务时，会因为A版本已存在升级任务，版本冲突而启动失败，必须先将A-B的升级任务禁用，才能启动A-C）；
3．同一个产品下，若已启用一条针对全部版本的升级任务，则不允许再启用其他任何版本的升级任务；
4．不允许启用闭环的升级任务（例如：某款产品下，已存在一条A-B的升级任务被启动，再启动一条B-A的升级任务时，会因为将产生死循环的升级任务，启用B-A升级任务失败）。

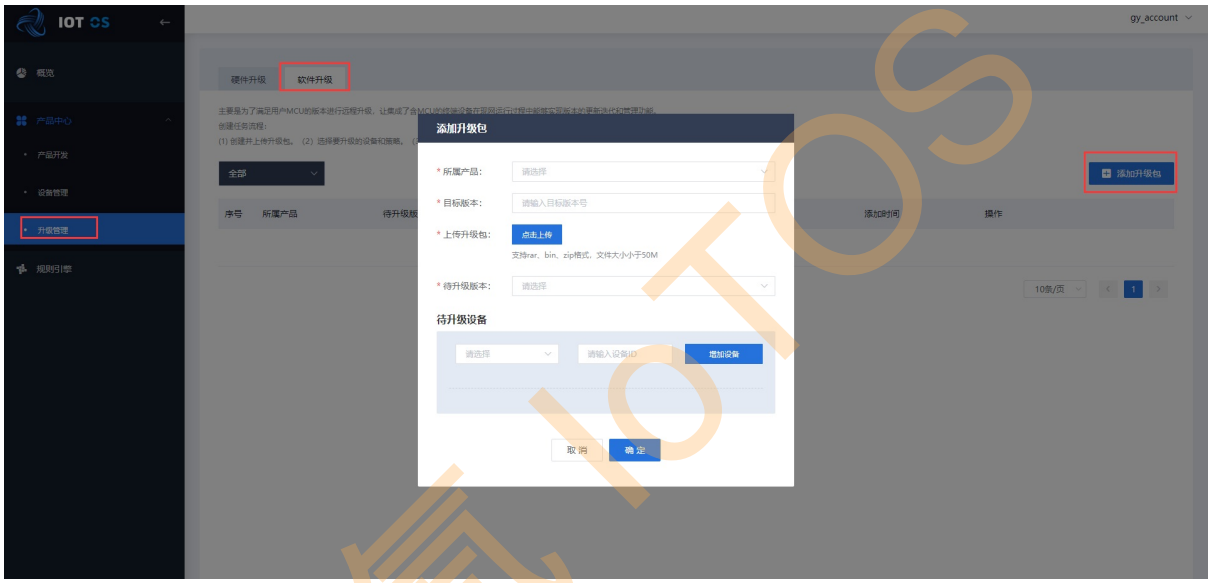
- 软件升级

软件升级

软件升级是针对MCU等应用软件的升级。

IoT OS操作流程：

1．登录IoT OS；
2．选择左侧导航栏“升级管理”；
3．单击“软件升级”标签；
4．单击“添加升级包”；
5．在添加升级包对话框中，输入软件信息，点击确定；
6．在软件升级列表中，单击升级规则的“启用”按钮，启用升级规则。



【说明】

1．所属产品：要升级的设备所隶属的产品。
2．目标版本：要升级到的新版本号。
3．升级包：新版本升级包。
4．待升级版本：
【全部版本】所属产品下所有版本的设备，包括没有版本的设备。
【指定版本】指定的要进行升级的版本，不是该版本的设备不会升级。

5．待升级设备：
【全部设备】产品下符合待升级版本条件的所有设备。
【指定设备】用来指定要升级的设备，设置后，仅添加了tid的设备才能升级到新版本。

【注意】

1．启用的升级规则在系统中一直是可执行的状态，有满足升级规则的设备上线，系统就会给该设备下发升级任务；
2．同一个产品下，可同时启动多条指定版本的升级任务，但同一版本不允许启用多条升级任务（例如：某款产品下，已存在一条A-B的升级任务被启动，再启动一条A-C的升级任务时，会因为A版本已存在升级任务，版本冲突而启动失败，必须先将A-B的升级任务禁用，才能启动A-C）；
3．同一个产品下，若已启用一条针对全部版本的升级任务，则不允许再启用其他任何版本的升级任务；
4．不允许启用闭环的升级任务（例如：某款产品下，已存在一条A-B的升级任务被启动，再启动一条B-A的升级任务时，会因为将产生死循环的升级任务，启用B-A升级任务失败）。

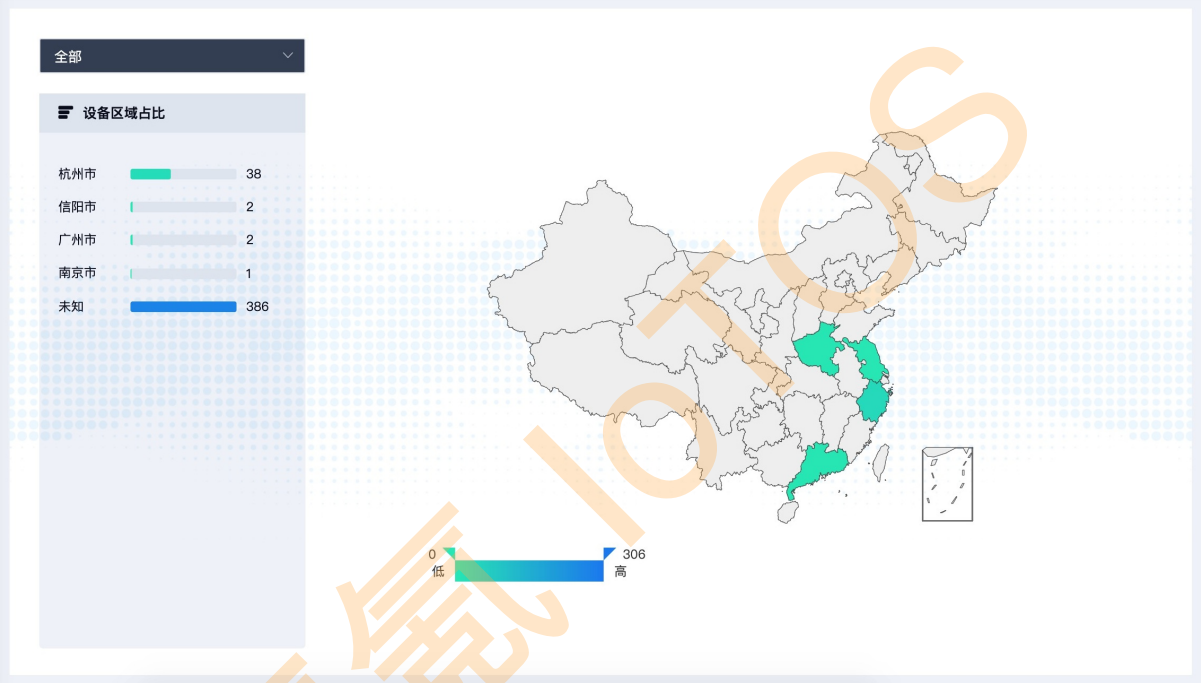
- 地理分析

地理分析

地理分析主要展示设备地理位置分布，通过左上角下拉选择框可以选择对应的产品进行精细统计，默认显示所有产品的设备统计。该页面分为以下两个部分：

【说明】

1．设备区域占比：统计所选产品下所有设备的省市分布，无法识别地理位置信息的设备归入“未知”；
2．地理位置可视分布：在中国省市地图上显示具体省份的设备数量分布，通过不同颜色区分设备数量占比，鼠标移至特定省份，悬浮显示该省份下的设备总数。



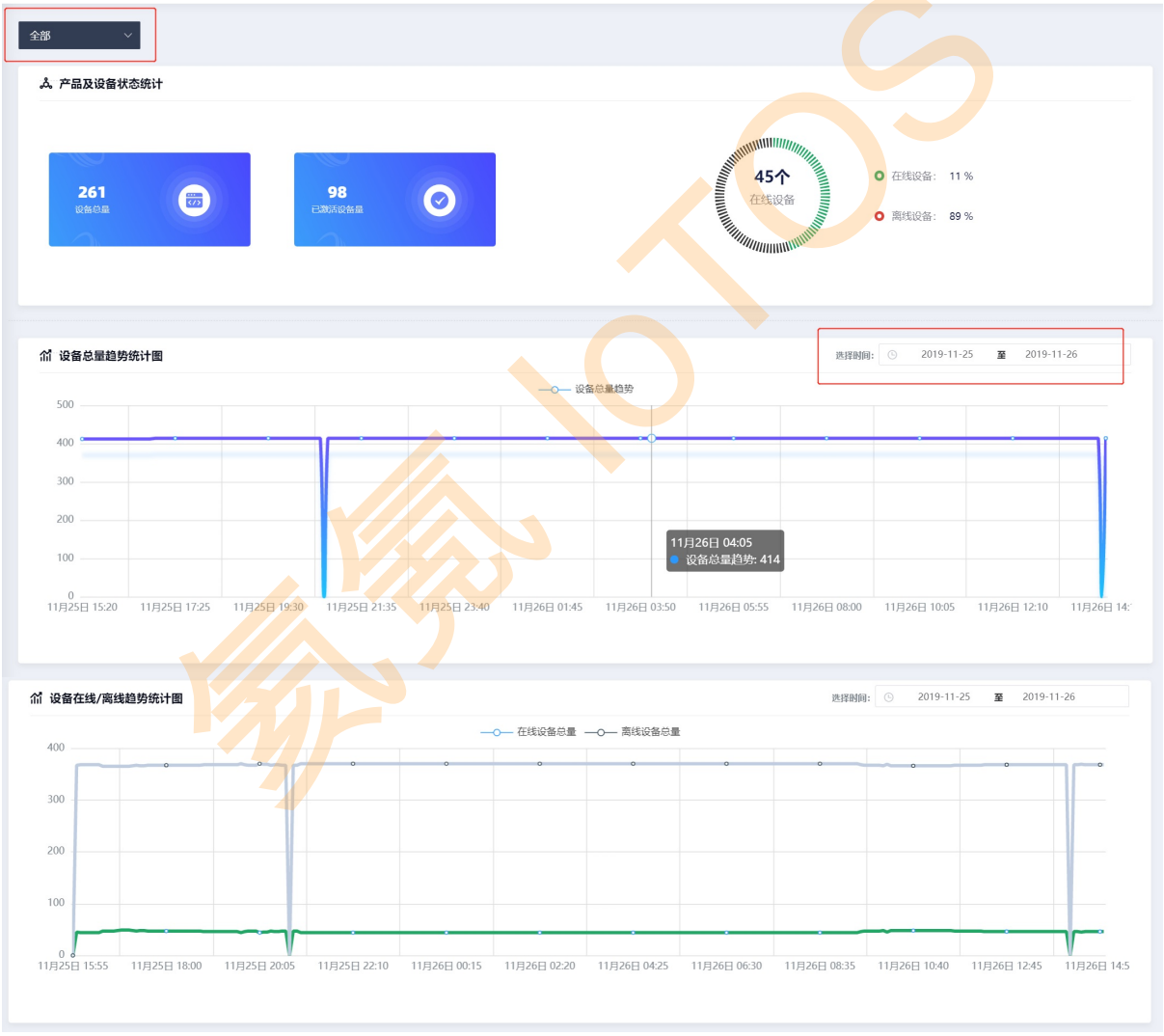
- 总体趋势

总体趋势

总体趋势统计主要统计设备数量随时间的变化趋势，左上角的下拉选择框可以选择不同的设备，默认统计所有设备下的设备数量变化趋势。

注意

1．产品及设备状态统计： 统计产品总量、激活设备数量以及在线离线设备数量占比，实时值；
2．设备总量趋势统计图：统计每日设备总量变化，点击右上角日期选择框可切换统计不同日期范围内的设备变化。时间范围最大可选30天，7天内展示数据点为每5分钟的瞬时情况，7天以上展示数据点为小时整点的瞬时情况；
3．设备在线、离线趋势统计图：统计每日设备在线离线数量变化趋势，点击右上角日期选择框可切换统计不同日期范围内的设备变化。时间范围最大可选7天，统计点为每5分钟的瞬时情况。



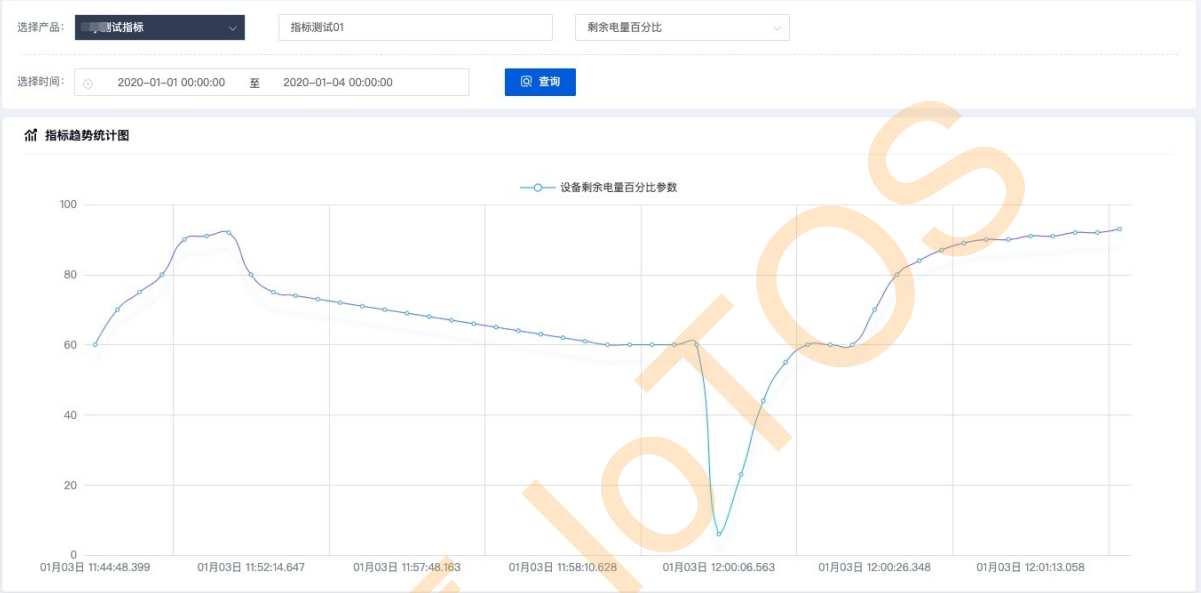
- 指标趋势

指标趋势

指标趋势功能表达的是某一设备的具体参数原始值的变化趋势。进行指标分析时，产品、设备、对应参数和时间范围均为必选值。

注意

当且仅当设备参数的数据类型为Number（包括整数和浮点数）时，可进行指标分析。



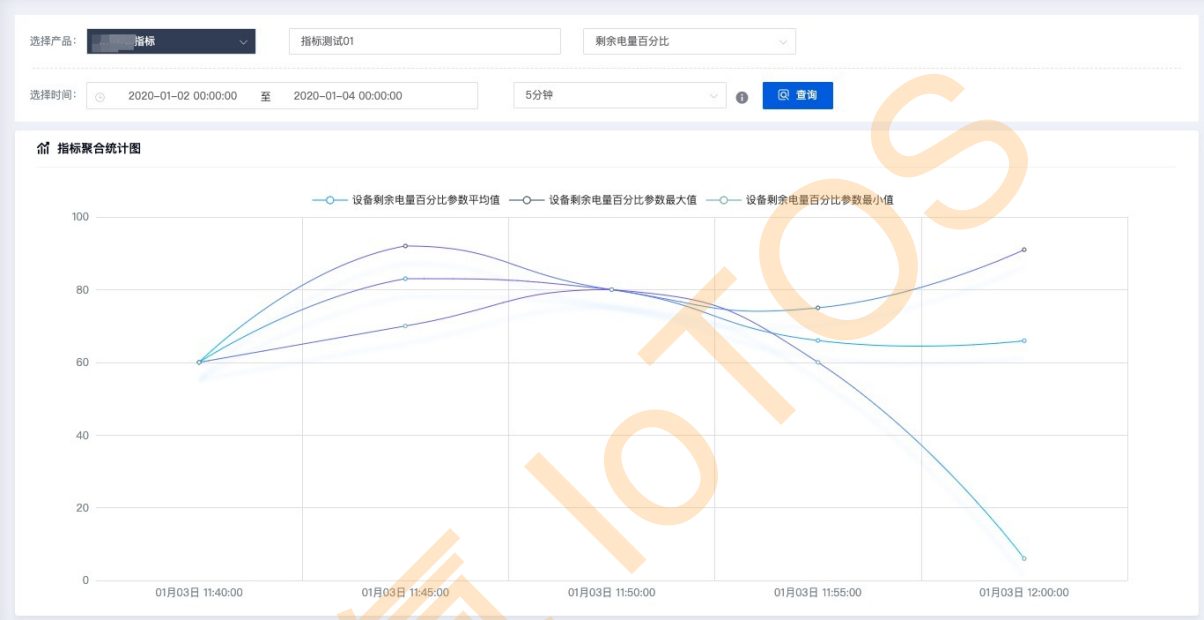
- 指标聚合

指标聚合

指标聚合是对某一设备的具体参数的原始值进行聚合计算，分别得出此参数在某一聚合力度下的最大值、最小值和平均值。进行指标聚合时，产品、设备、对应参数、时间范围和聚合力度均为必选值。

注意

```
1.当且仅当设备参数的数据类型为Number（包括整数和浮点数）时，可进行指标聚合分析；
2.聚合力度可选值为"5分钟"、"1小时"和"1天"，对应的最大时间范围为"一周"、"一个月"和"一年"。
```



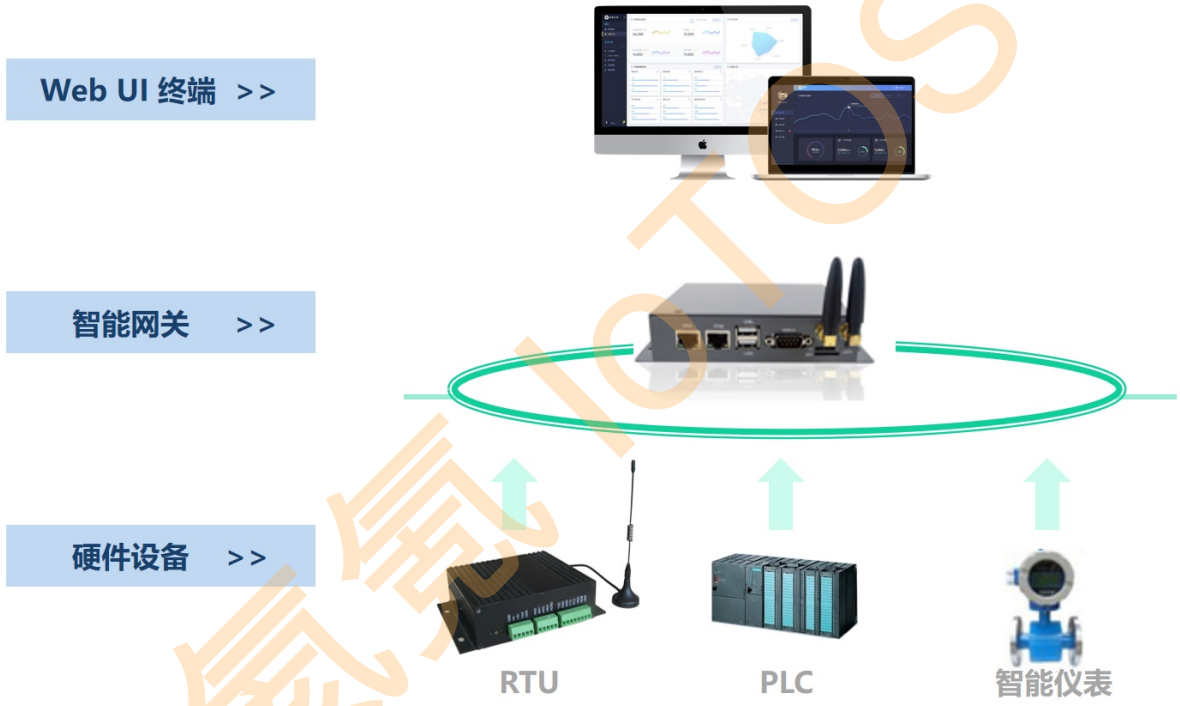
- 概述

概述

IoT OS可接入市面上大多数硬件网关，这里以氦氪工业网关Hekr Gateway204作为对接实例进行介绍。

Hekr Gateway204采用 TI 公司 CortexA8 嵌入式低功耗 CPU（主频600M）为核心，外扩 256MB DDR3,256MB Nandflash （可选8G e.MMC）存储，4路RS485/RS232，2个10M/100M 自适应的以太网，1个4G 模块，2个USB-Host接口，可接TF卡外外扩大容量存储，具有电磁屏蔽性好，美观坚固的铝合金结构机箱。

工业网关是一款基于物联网架构设计的工业级嵌入式软硬件一体化设备。可实现工业现场各种设备的数据采集，存储，转发，系统维护，域名管理等功能，用户可在PC，手机上使用浏览器进行数据监控和参数配置。该网关可广泛应用于光伏电站，风电站，小水电，配电室，抽油机，灌区，管道，环境监测站，消防监控室，农业大棚，空气监测站,实验室，工控中心，无人值守站等场站监控和园区能源管理系统，各种在线监测系统等。 工业网关在智慧工厂中的典型应用场景：



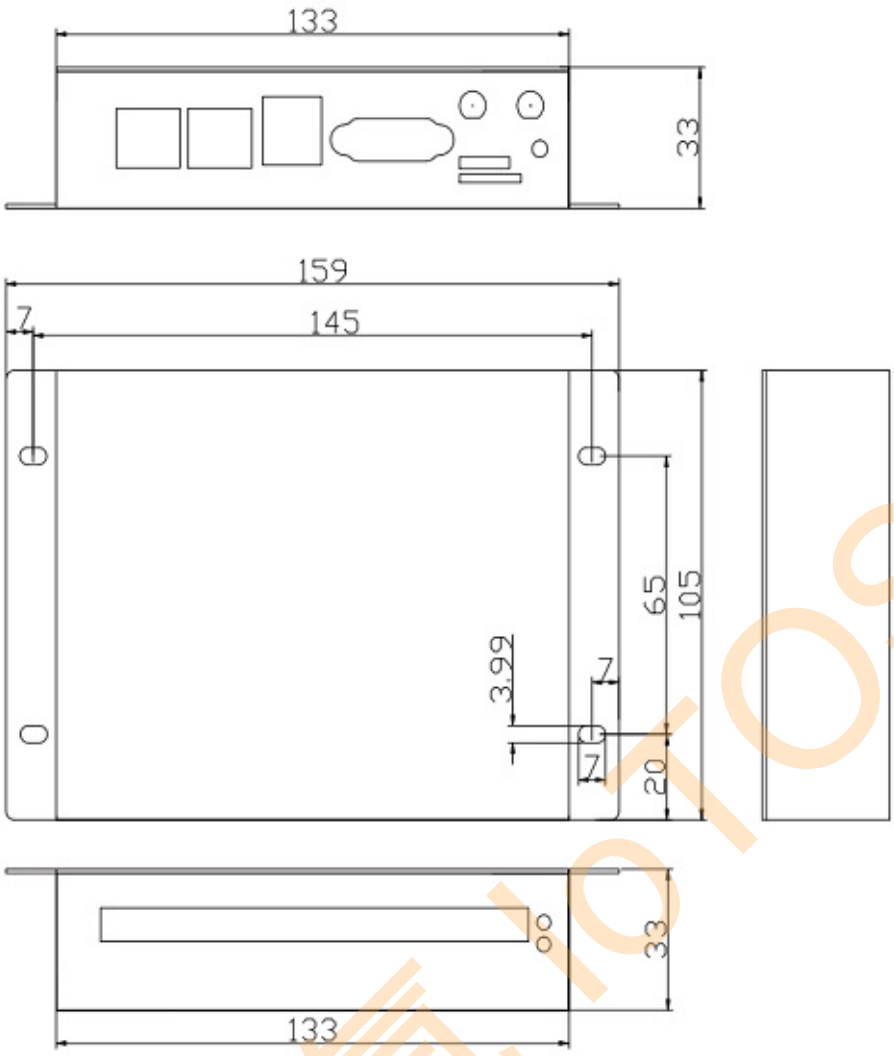
- 硬件介绍
  - Hekr Gateway204硬件参数
  - Hekr Gateway204尺寸图
  - 电源及串口端子定义
  - 接线安装
  - 网络接口图示

硬件介绍

Hekr Gateway204硬件参数

硬件参数
- CPU: Cortex-A8 AM335x
- 2 X LAN：10/100Mbps自适应
- 4 X RS232 / RS485
- 2 X CAN
- 2 X USB-Host
- 支持4G、Wi-Fi、以太网通讯
- 内存：1 X 256MB DDR3
- 电源要求： 9~30C DC，推荐12V DC
- 可扩展存储：32G（TF卡）
- 工作温度：-40 - 85℃
- 工作湿度：5% - 95%无凝露

Hekr Gateway204尺寸图



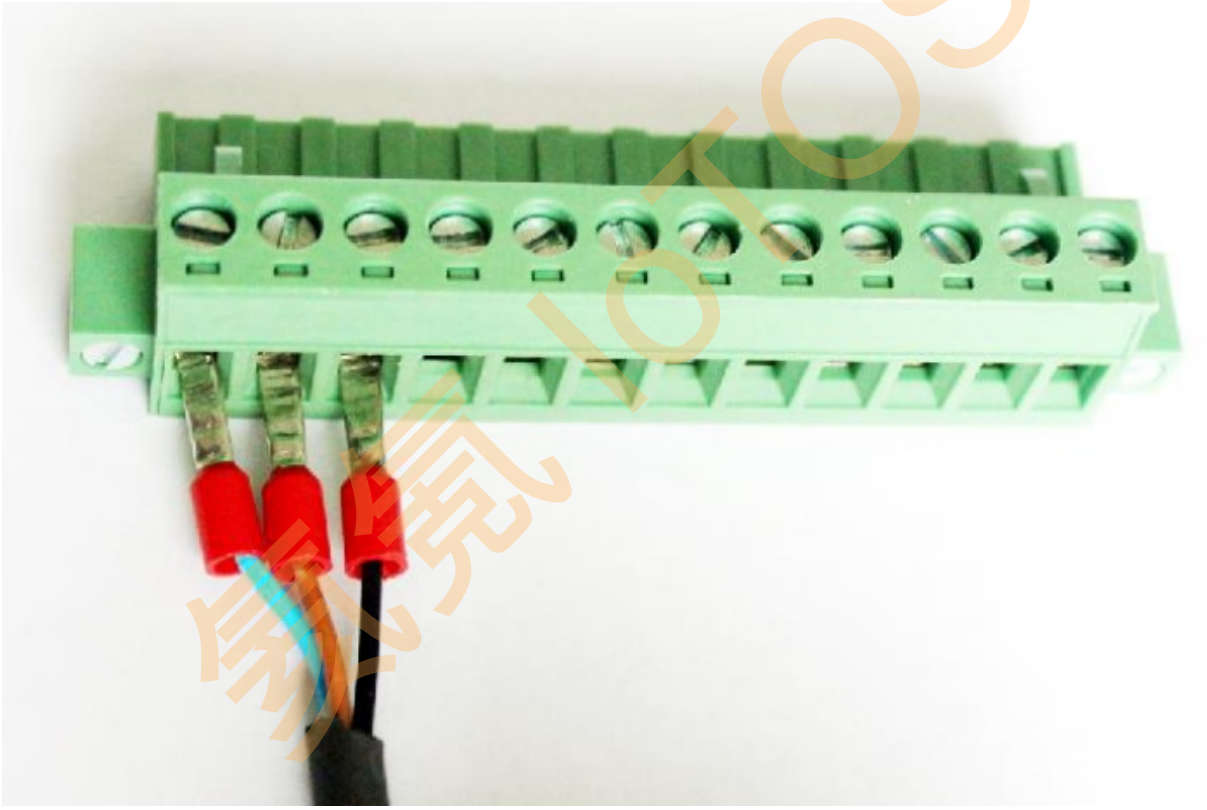
电源及串口端子定义

引脚序号	信号定义	功能说明	备注
1	12V/VCC	电源输入+端	电压范围9V-30V 推荐使用12V供电
2	GND	电源输入-端/电源地	电压范围9V-30V 推荐使用12V供电
3	232TX1	232发送引脚	第一路RS232接口
4	232RX1	232接收引脚	第一路RS232接口
5	232GND	信号地	
6	232TX2	232发送引脚	第二路RS232接口
7	232RX2	232接收引脚	第二路RS232接口
8	232GND	信号地	
9	232TX3	232发送引脚	第三路RS232接口
10	232RX3	232接收引脚	第三路RS232接口
11	232GND	信号地	
12	232TX4	232发送引脚	第四路RS232接口
13	232RX4	232接收引脚	第四路RS232接口
14	232GND	信号地	
15	485A1	485+端引脚	第一路RS485接口
16	485B1	485-端引脚	第一路RS485接口
17	485A2	485+端引脚	第二路RS485接口
18	485B2	485-端引脚	第二路RS485接口
19	485A3	485+端引脚	第三路RS485接口

20	485B3	485-端引脚	第三路RS485接口
21	485A4	485+端引脚	第四路RS485接口
22	485B4	485-端引脚	第四路RS485接口
23	CANL1	CAN总线L端引脚	第一路CAN接口
24	CANH1	CAN总线H端引脚	第一路CAN接口
25	CANGND	信号地	
26	CANL0	CAN总线L端引脚	第零路CAN接口
27	CANH0	CAN总线H端引脚	第零路CAN接口
28	CANGND	信号地	

接线安装

接线时，我们建议您带有管型预绝缘端子的导线。将导线插入接线端子对应的插槽中，并将螺丝拧紧即可。



网络接口图示

Hekr Gateway204型工业网关共有2个网口。ETH1作为外网网口接入路由器或者交换机，ETH0作为内网网口常用于网关配置，首次使用网关建议PC机与ETH0直连进行参数配置。

- 功能介绍
  - 主要接口及协议适用情况

功能介绍

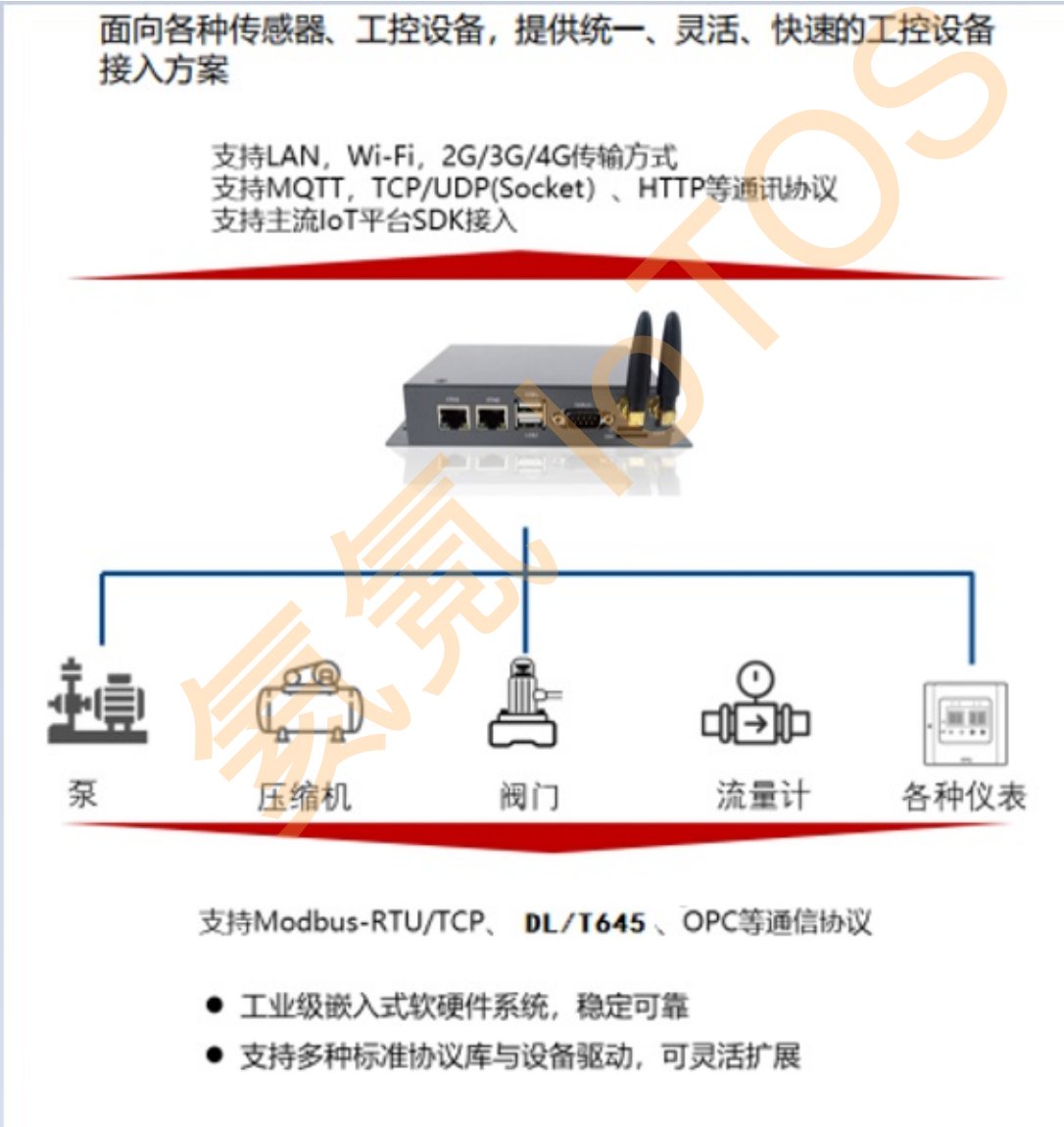
1.具备对下（自动化系统）协议解析能力（通讯协议：Modbus、DL/T645、CJ/T188等）；总线协议：OPC UA、BACNET/IP，KNX等。

2.具备对上（IT系统）的协议对接能力（MQTT、TCP、UDP、HTTP），对上的通讯能力（以太网、Wi-Fi、4G）。

3.具备对下（采集）和对上（转发）的私有协议二次开发功能。

4.具备数据缓存、本地数据存储的功能。

5.支持web登录配置设备参数、通信参数、通讯点位。



主要接口及协议适用情况

接口类型	协议
串口	ModbusRTU、DL/T645-1997、DL/T645-2007、CJ/T188等

网口	ModbusTCP、MQTT、OPC_UA、BACNET/IP、KNX等

- 工程配置
  - 准备工作
  - 建立网络连接
  - 登录智能管理界面
  - 设备主界面
  - 创建并配置实例

工程配置

设备上电后，就可通过web浏览器配置工程了。下面我们将以使用Modbus协议采集，KLink协议转发到IoT OS为例，引导配置一个网关运行实例。

准备工作

初次使用工业网关时，需用网线将上位机与其ETH0网口直连。请保证上位机的网卡正常并且有线网络连接处于开启状态。

建立网络连接

Hekr Gateway204型工业网关共有2个网口。ETH1作为外网网口接入路由器或者交换机，可通过DHCP自动获取上级设备分配的IP地址，ETH0作为内网网口，上位机与ETH0直连后可通过DHCP自动获取内网给上位机分配的IP地址，windows上位机可通过cmd指令ipconfig查询本机有线网络IP地址，通常为192.168.17.X，那么ETH0网关访问地址即为192.168.17.1，可使用Ping命令确认网络连通性。

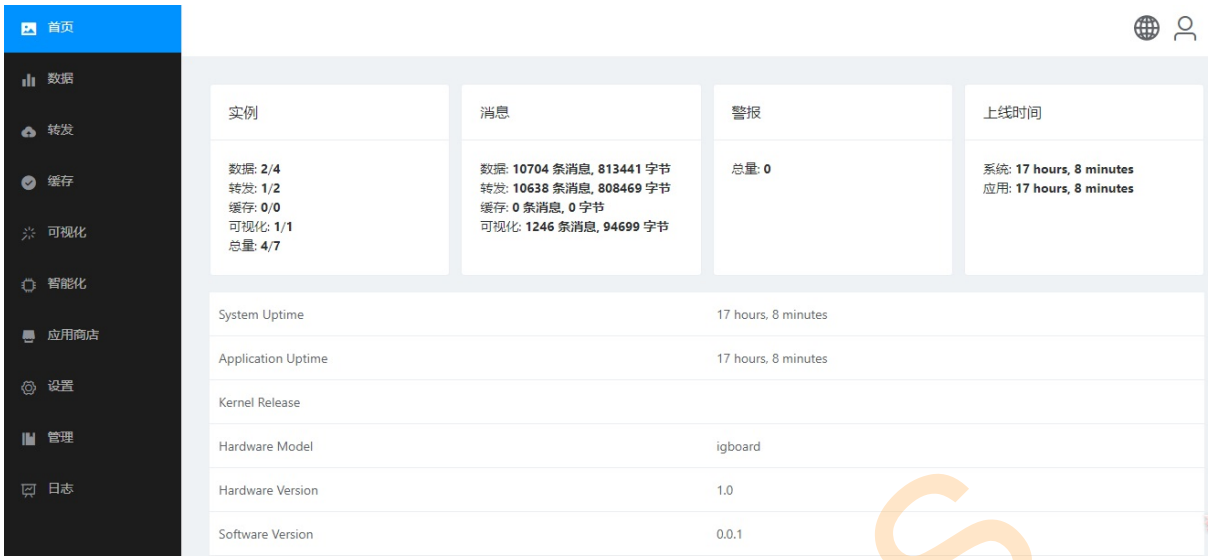
登录智能管理界面

运行Web浏览器（推荐Chrome浏览器版本号78.0.3904.108），在地址栏输入网关访问地址（通常为192.168.17.1，部分场景会有不同），回车后进入如下登录页面。

输入用户名、密码（缺省均为admin，区分大小写），单机“登录”按钮或直接回车即可进入智能管理页面。

设备主界面

登录成功后，主界面如下图



主界面信息栏显示网关设备当前主要的运行参数。

创建并配置实例

下面，我们将以典型的串口Modbus协议采集，网口KLink协议转发到IoT OS为例，引导您一步一步创建并配置一个实例。一般步骤为：

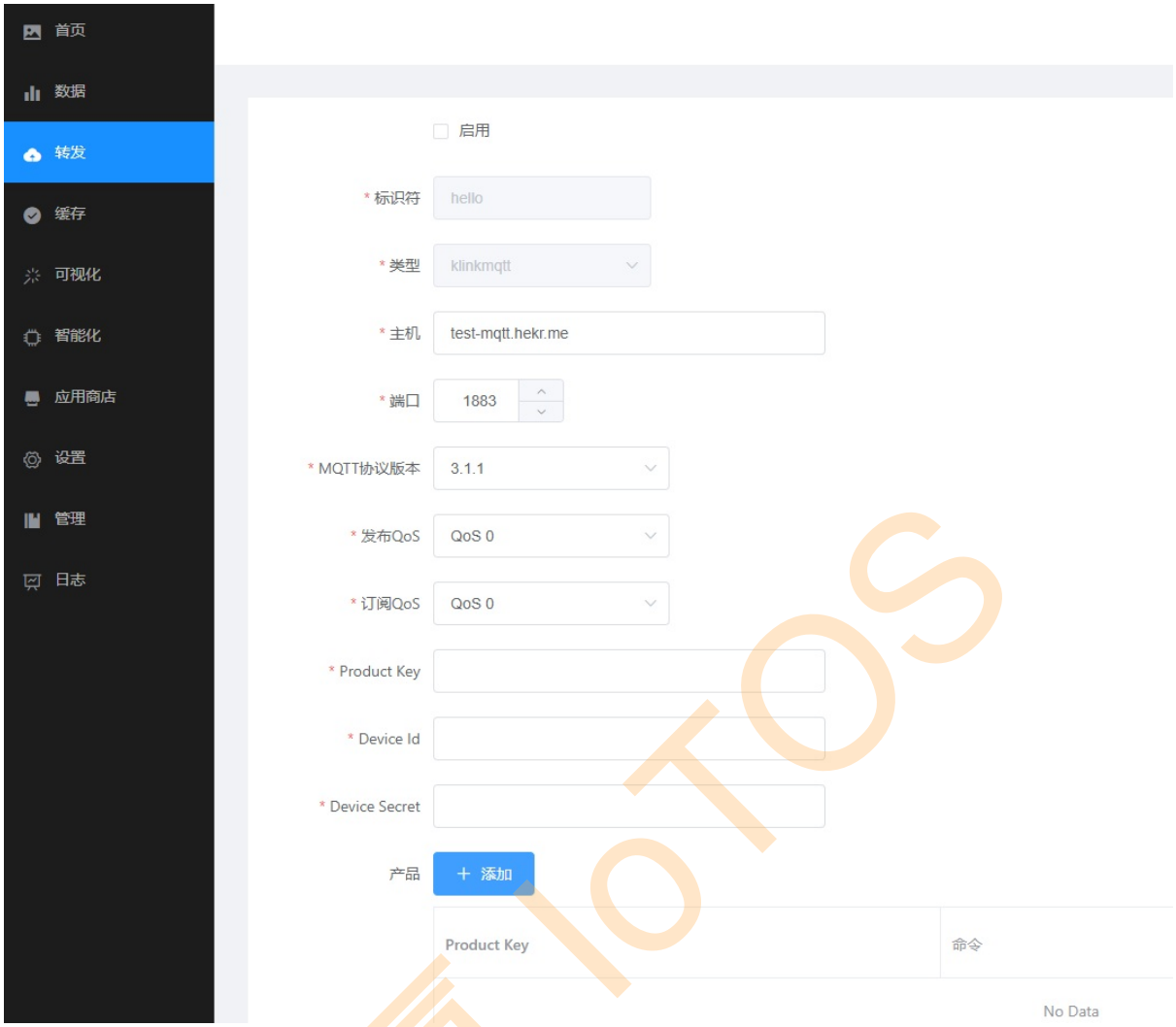
1. 创建转发实例
2. 配置转发实例参数
3. 创建数据采集实例
4. 配置数据采集实例参数并关联到转发实例
5. 保存改动到配置文件并重启设备

创建转发实例

点击侧边栏“转发”按钮并点击“添加”在弹出来的页面中填写转发实例标识符，类型选择“KLinkMQTT”

配置转发实例参数

- 添加实例完成后点击该实例的  按钮进行参数配置，界面如下：



- 在“主机”栏中填写设备连接服务器域名或者IP：IoT OS测试mqtt协议连接地址为test-mqtt.hekr.me
- 在“端口”栏中填写设备连接服务器端口：mqtt协议连接端口默认为1883
- 在“MQTT协议版本”栏中选择3.1.1
- 在“发布QoS”栏中选择QoS0
- 在“订阅QoS”栏中选择QoS0
- 在“Product Key”栏中填写在IoT OS上创建的网关产品PK
- 在“Device Id”栏中填写在IoT OS上创建的网关设备名称
- 在“Device Secret”栏中填写在IoT OS上创建的网关设备秘钥
- 点击“添加”在“添加产品”页面的“Product Key”栏中填写在IoT OS上创建的子设备产品PK，在“命令”栏中填写命令名称，这里上报命令可填写report，填写完成后点击“确定”
- 上一步添加子产品完成后会生成子设备的产品记录，点击该行记录的“编辑”按钮可进行管理子设备，这里点击“添加”按钮在“Device Id”栏中填写在IoT OS上创建的子设备名称
- 在“上报间隔”栏中可填写命令帧的上报时间间隔，这里默认填写5000（毫秒）
- 上述一系列操作完成后在页面最上方的“启用”按钮栏打钩并点击页面最下方的“更新”按钮。

至此，KLink协议连接到IoT OS转发实例配置完成。

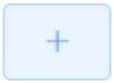
创建数据采集实例

点击侧边栏“数据”按钮并点击“添加”在弹出来的页面中填写数据采集实例标识符，类型选择“Modbus RTU”

配置数据采集实例参数并关联到转发实例

- 添加实例完成后点击该实例的  按钮进行参数配置，界面如下：

- 在“设备节点”栏中填写网关设备串口通道地址，默认通道为/dev/ttyO4
- 在“波特率”栏中选择通信波特率，默认为9600
- 在“数据位”栏中选择数据位，默认为8
- 在“校验位”栏中选择校验位，默认无校验为N
- 在“停止位”栏中选择1
- 在“超时时间”栏中填写Modbus命令应答超时时间，这里填1000（ms）

- 点击  按钮添加设备，在页面中的“设备标识”栏填写在IoT OS上创建的子设备名称（同转发实例中的子设备名称）
- 在“设备采集点表”栏下点击“+”进行Modbus采集点位编辑
- 本实例中的Modbus协议设备为温度传感器，“属性名”填为channelA，“设备地址”填为1，“功能码”选择保持寄存器（0x03，0x16），“寄存器地址”填为40，“数据类型”填为INT16，“交换寄存器内高低字节”填为否，“交换寄存器顺序”填为否，“时间间隔”填为1000（这里的Modbus解析参数可能需要根据实际接入的设备协议不同进行调整）
- 上述一系列操作完成后在页面最上方的“启用”按钮栏打钩并点击页面最下方的“更新”按钮。并将上方菜单“高级”栏中“转发目标”选为上一步建立的转发实例，勾选“启用转发”的启用按钮并点击页面下方的“更新”按钮

至此，通过Modbus RTU协议采集温度传感器数据采集实例配置完成，并成功关联到KLink转发实例。

保存改动到配置文件并重启

点击侧边栏“设置”按钮并点击“保存改动”页面右上角弹出“成功”则配置文件成功保存到网关设备。点击“重新启动硬件”则可重启网关设备。

- 管理后台

管理后台

此章节为面向IoT OS管理员对各种管理操作的说明，普通用户可以忽略此章节内容。

- 用户管理
  - 创建用户
  - 修改用户
  - 切换用户
  - 禁用用户
  - 审核（认证）用户

用户管理

创建用户

超管账户登录后，点击右上角的“管理后台”，切换到管理后台模块中，点击“用户管理”-“创建用户”，即可开始创建平台用户。 创建用户时，需填写的用户信息包括“用户类型”、“用户名”、“密码”、“手机号”、“邮箱”。

【说明】

```
1．用户类型：分个人用户、企业用户两种。用来标识用户类型的标签，功能上两种用户暂无区别；
2．用户名：登录平台时使用的账号；
3．密码：登录平台时使用的密码；
4．手机号/邮箱：跟用户名进行绑定的手机号，邮箱，仅绑定用，不能作为账号登录平台。
```

修改用户

超管账户登录后，点击平台右上角的“管理后台”，切换到管理后台模块中，点击“用户管理”，在用户列表中找到需要修改的用户，点击“编辑”按钮，可重新编辑用户密码、手机号、邮箱等信息。

【说明】

```
1．用户名、用户类型不可修改；
2．编辑用户时，若密码输入框中填写了信息，表示将密码修改为新填写的内容，不填写表示不修改原来的密码。
```

切换用户

超管账户可以切换成用户列表中的任意用户，点击操作栏下的"切换用户"按钮，即可以此用户的身份登录IoT OS。

禁用用户

若要限制已创建的用户登录平台，可通过超级管理员账号登录平台，点击平台右上角的“管理后台”，切换到管理后台模块中，点击“用户管理”，在用户列表中找到需要禁用的用户，点击“禁用”按钮，在弹出的禁用确认框中点击“确定”，即可完成用户禁用操作。被禁用的用户将无法再登录IoT OS，需要超管用户在管理后台解禁，才能恢复登录权限。

【说明】

```
1．用户被禁用后，用户列表中“用户状态”显示为“禁用”，操作下的禁用按钮显示为“解禁”；
2．超管可点击“解禁”，解除用户的登录限制。
```

审核（认证）用户

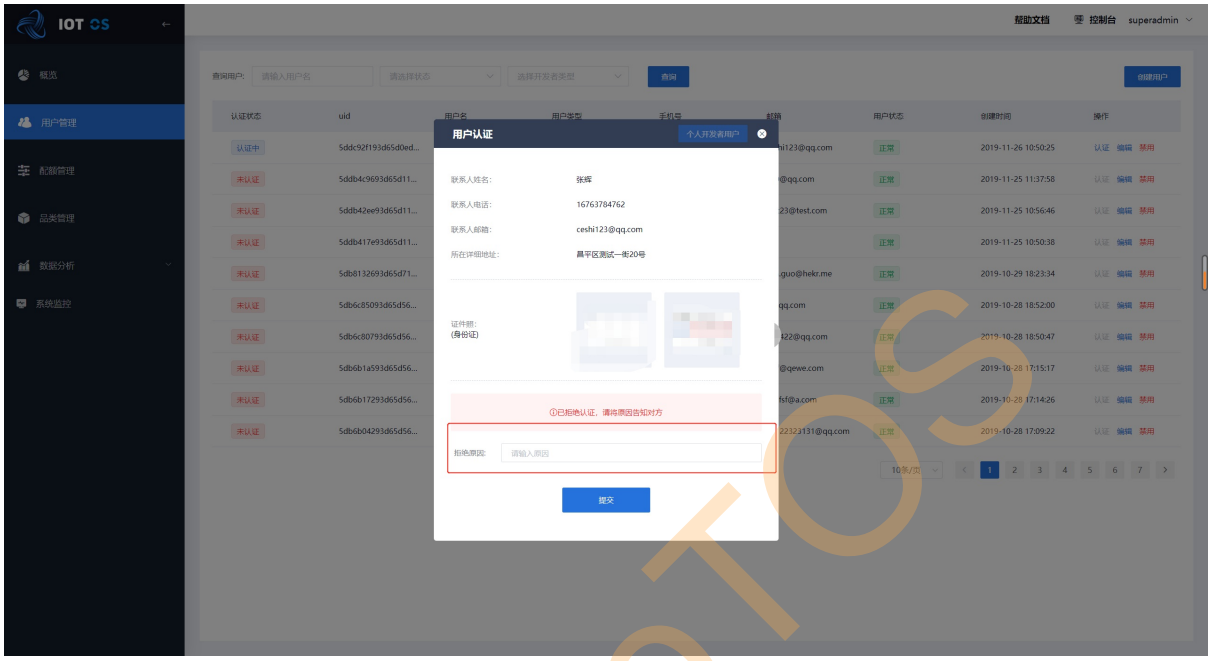
用户认证状态分为四种：

- 未认证： 未提交实名认证申请的用户。
- 认证中： 已提交认证申请，超管未审核。
- 认证成功：已提交认证申请，超管审核通过。

- 认证失败：已提交认证申请，超管审核不通过。

已提交实名认证申请的用户，在超管后台用户列表中认证状态显示为“待认证”，超管登录后进入“管理后台”-“用户管理”，操作栏中点击“认证”，即可进行用户认证。

点击“确认”，即可认证成功，用户认证状态变为认证成功。点击“拒绝认证”，需要填入拒绝理由，提交后用户认证状态变为认证失败，如下图所示：



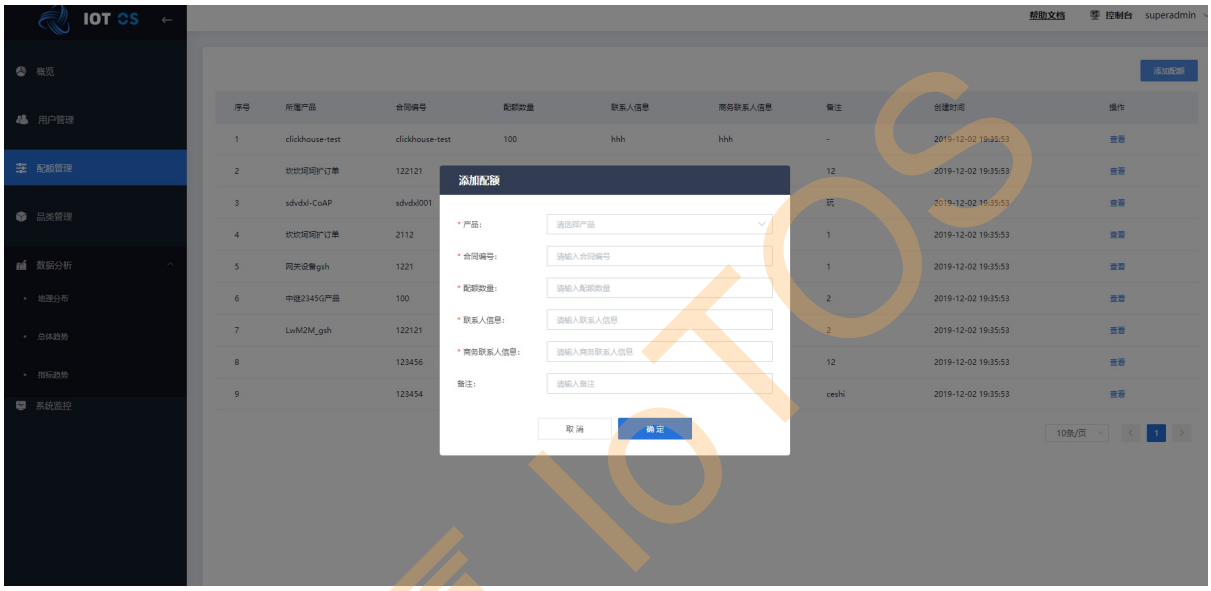
- 配额管理

配额管理

每个产品创建后会默认分配10个设备配额，用户可免费在产品下添加10个设备，每添加一个设备，配额会自动减1。

用户配额用尽后，需向管理员申请增加配额。管理员账号登录平台后点击“管理后台”切换到管理后台模块，在左侧导航栏中点击“配额管理”，点击“新增配额”，选择要增加配额的产品，填写增加的额度以及其他信息，点击“确定”即可为选择的产品增加设备额度。

添加配额时，须填写“所属产品”、“合同编号”、“配额数量”、“联系人信息”、“商务联系人信息”。 管理员在后台新增配额后，用户在“设备管理”中，新增设备时可看到设备新剩余的配额。



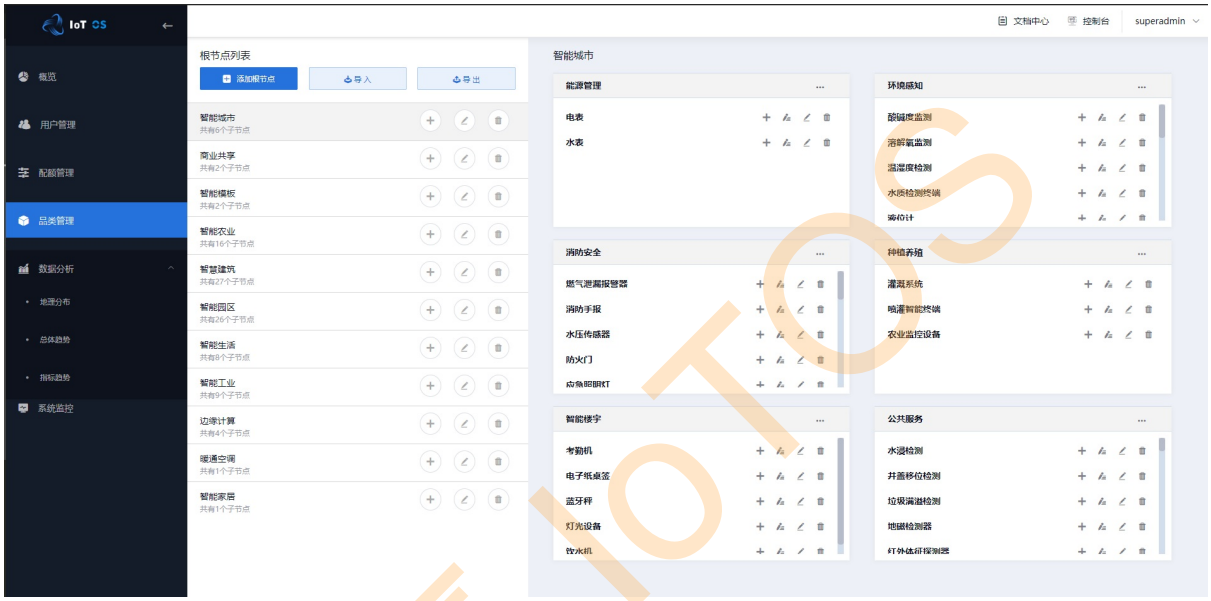
- 品类管理

品类管理

IoT OS的产品品类是您给产品的标识。您可以使用产品品类来灵活管理产品分组。

产品品类通常描述的是对一个产品下所有设备所具有的共性信息。如产品的制造商、所属单位、外观尺寸、操作系统等。在系统录入产品品类后，创建产品时便可灵活选择所属品类。

超管账户登录，点击平台右上角的“管理后台”，点击“品类管理”，可进行品类新增、编辑、删除等操作。目前最多支持添加五级品类，多级品类时，需要一级一级添加。



以新增品类“智能家居/智能插座”为例，操作说明如下：

1. 在品类管理中，点击“增加根节点”，输入一级品类名称“智能家居”，点击确定；
2. 品类管理列表中找到新增的“智慧城市”品类卡片，点击“+增加二级节点”，输入二级品类名称“智能插座”，点击确定；
3. 最后一级品类名称可点击“编辑功能定义”，输入标准功能定义。保存之后创建产品时选择该品类可直接导入标准功能定义。

编辑/删除品类

超管账户登录，点击平台右上角的“管理后台”，点击“品类管理”，找到要编辑的品类名称，点击品类右侧对应的编辑或删除按钮，可对品类进行修改或删除。

【注意】

> 若删除上级节点，会默认将该节点下的所有子节点也删除。

- 数据分析
  - 地理分析
  - 指标趋势
  - 总体趋势

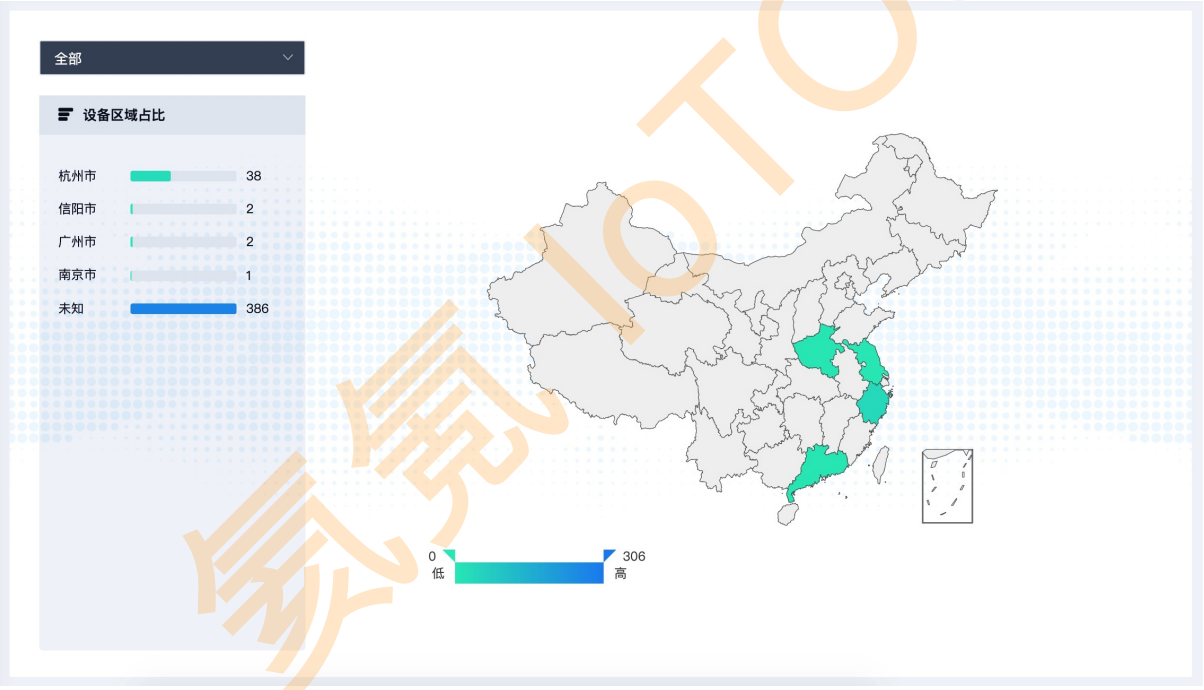
数据分析

地理分析

地理分析主要展示设备地理位置分布，通过左上角下拉选择框可以选择对应的产品进行精细统计。默认显示所有产品下的设备统计。该页面分为以下三个模块:

【说明】

```
1．设备区域占比：统计所选产品下所有设备的省市分布，未设置地理位置信息的设备归入“其他”范畴；
2．设备状态占比：  统计所选产品下所有设备的在线状态，圆环中间显示设备总数；
3．地理位置可视分布：在中国省市地图上显示具体省份的设备数量分布，通过不同颜色区分设备数量占比，鼠标移至特定省份，悬浮显示该省份下的设备总数。
```

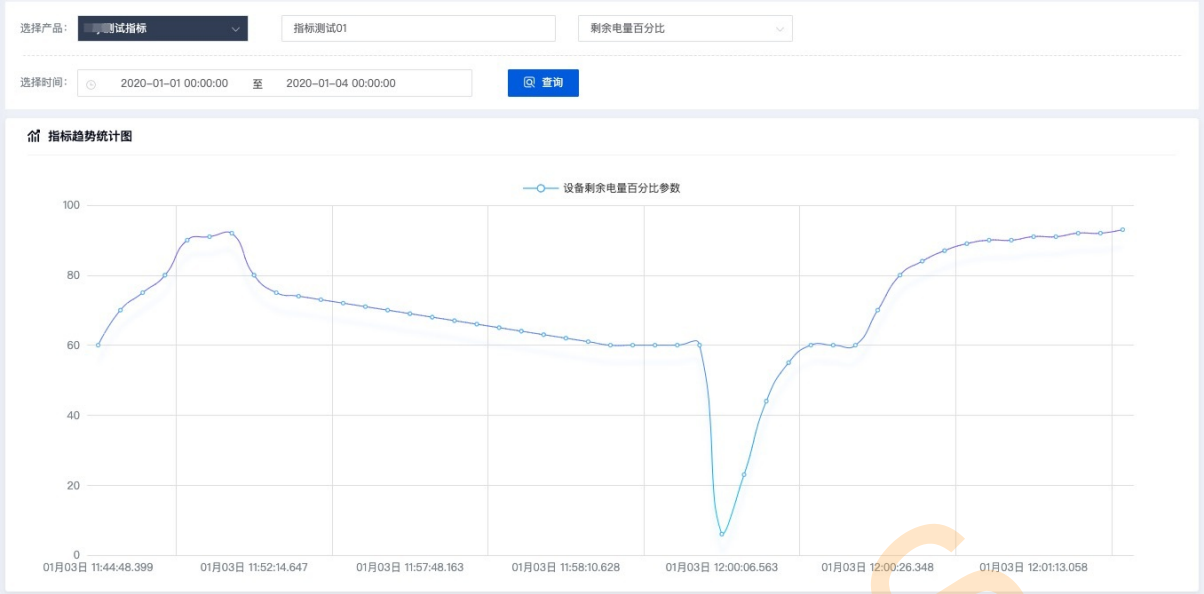


指标趋势

指标趋势功能表达的是某一设备的具体参数原始值的变化趋势。进行指标分析时，产品、设备、对应参数和时间范围均为必要值。

注意

```
当且仅当设备参数的数据类型为Number（包括整数和浮点数）时，可进行指标分析。
```

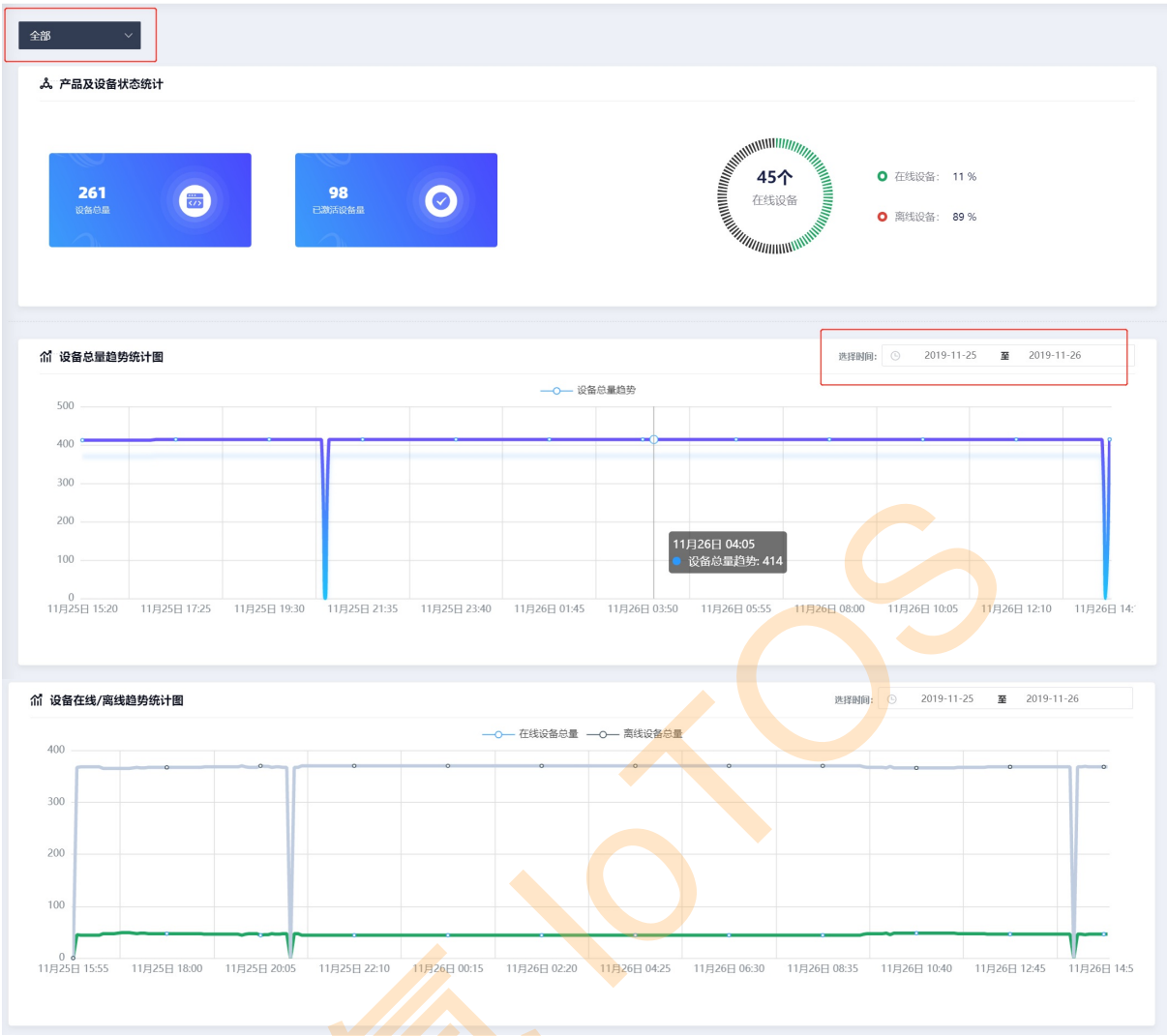


总体趋势

总体趋势统计主要统计设备数量随时间的变化趋势，左上角的下拉选择框可以选择不同的设备，默认统计所有设备下的设备数量变化趋势。

注意

1．产品及设备状态统计： 统计产品总量、激活设备数量以及在线离线设备数量占比，实时值。
2．设备总量趋势统计图：统计每日设备总量变化，点击右上角日期选择框可切换统计不同日期范围内的设备变化。时间范围最大可选30天，7天内展示数据点为每5分钟的瞬时情况，7天以上展示数据点为小时整点的瞬时情况。
3．设备在线、离线趋势统计图：统计每日设备在线离线数量变化趋势，点击右上角日期选择框可切换统计不同日期范围内的设备变化。时间范围最大可选7天，统计点为每5分钟的瞬时情况。


全部
产品及设备状态统计
261
设备总量
98
已激活设备量
45个
在线设备
在线设备： 11 %
离线设备： 89 %
设备总量趋势统计图
选择时间： 2019-11-25 至 2019-11-26
设备总量趋势
500
400
300
200
100
0
11月25日 15:20
11月25日 17:25
11月25日 19:30
11月25日 21:35
11月25日 23:40
11月26日 01:45
11月26日 03:50
11月26日 05:55
11月26日 08:00
11月26日 10:05
11月26日 12:10
11月26日 04:05
设备总量趋势: 414
设备在线/离线趋势统计图
选择时间： 2019-11-25 至 2019-11-26
在线设备总量
离线设备总量
400
300
200
100
0
11月25日 15:55
11月25日 18:00
11月25日 20:05
11月25日 22:10
11月26日 00:15
11月26日 02:20
11月26日 04:25
11月26日 06:30
11月26日 08:35
11月26日 10:40
11月26日 12:45

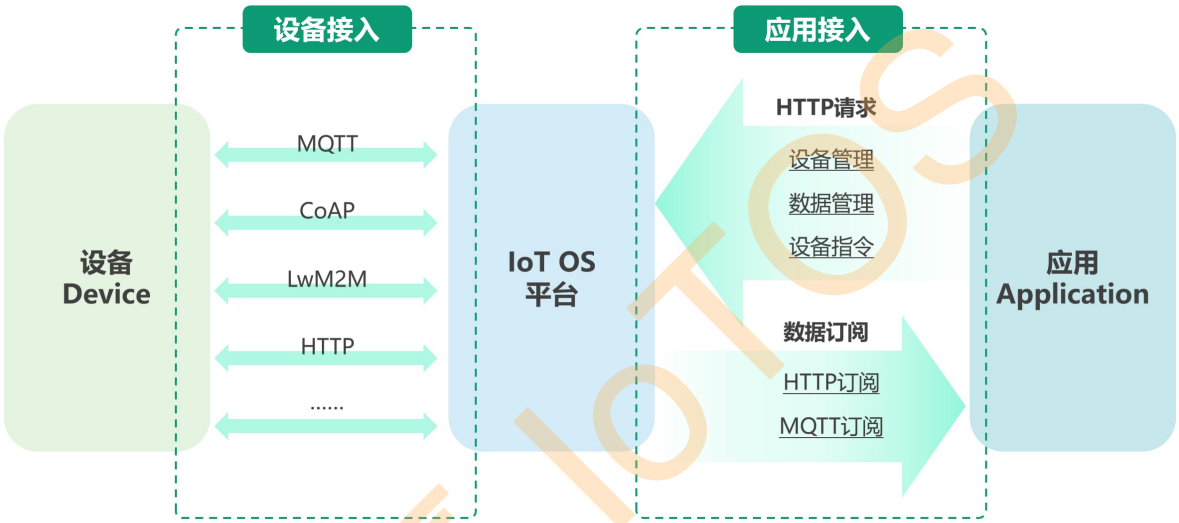
- 系统监控

系统监控

主要用于监控系统指标，包括但不限于消息量大小，设备连接数、主机运行状态等，采用可视化图表（如折线图，仪表盘，柱状图等）展示实时统计数据。

- 应用开发指南及API
  - 使用说明
    - 安全鉴权
    - 获取AccessKey
    - 使用AccessKey计算token
    - 如何使用token
    - 获取请求地址和端口
    - 北向开发demo

应用开发指南及API



此部分的接口提供给北向开发者使用，以进行北向应用的开发。

开发者通过HTTP请求向IoT OS进行设备管理、数据管理和设备指令相关的请求，通过数据订阅获取设备的各种数据。

使用说明

IoT OS为接口访问提供了安全鉴权认证。开发者访问接口时需要在HTTP请求头中包含签名信息。

安全鉴权

IoT OS提供给用户AccessKey。 AccessKey由AccessKey ID和AccessKey Secret组成。AccessKey ID用于标识访问者身份，可以明文的形式传输。AccessKey Secret是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密。

可以避免密钥在传输过程中泄露，且通过包含由非可逆算法生成的签名的token来进行身份认证，即使token被窃取，攻击者也无法通过token反向获得核心密钥。

鉴权参数token具有时间属性，IoT OS会计算token有效时间，可从时间维度降低被攻击/仿冒的风险。

获取AccessKey

登录Iot OS点击右上角"个人中心"，选择"AccessKey"标签。

若未获取过AccessKey则点击右边"获取AccessKey"按钮。

【注意】 每个账号最多添加10个AccessKey。

使用AccessKey计算token

token参数组成

名称	类型	说明	示例
accessKey	String	用户的AccessKey ID，为由24位数字或字母组成。	qzJ2UCE86Fd14hRG1LzrkT7w
path	String	用户发起请求的路径	/addDevice
timestamp	String	数值为当前时间戳的长整型形式的字符串，时间戳单位为毫秒。	1575615530113
method	String	加密方式，IoT OS使用的是HmacSHA1，此处写"SHA1"即可	SHA1
sign	String	通过AccessKey Secret与部分参数进行加密后的签名	具体计算方法见下详解

【注意】：

1.其中参数 `path` 的格式为请求 `端口` 与定界符 `?` 之间的内容。

即 protocol://userInfo@host:port `/path` ?query#fragment 中高亮部分。

若请求有路径参数时也需要包含路径参数。

例：

```
http://localhost:8080/api/device/getDeviceHistoryData/9d7bc79042934535/Modb453543?page=0&size=10&startTime=1575993600000&endTime=1576166399999
```

其中 `path` 值为 `/api/device/getDeviceHistoryData/9d7bc79042934535/Modb453543` 。

2.token有效期为参数 `timestamp` 时刻前后五分钟。

sign计算

首先对path、timestamp和method进行拼接得出data。拼接方式如下：

```
data = path + "\n" + timestamp + "\n" + method
```

使用HmacSHA1加密方式，使用AccessKey Secret作为密钥进行加密。

```
sign = HmacSHA1(data,accessKeySecret)
```

【注意】sign的计算结果要将字母保持为全小写。

token组合

计算出所有的参数后，按照顺序，使用"&"符号将参数连接。

【注意】path参数的数值需要进过URL编码。需要进行编码的特殊符号如下：

符号	编码
+	%2B
空格	%20
/	%2F
?	%3F
%	%25
#	%23
&	%26
=	%3D

token结果：

```
accessKey=qzJ2UCE86Fd14hRG1LzrkT7w&path=%2FaccessKey&timestamp=1575652666325&method=SHA1&sign=58d5e5972e3d69c5d
a1867416726966182e73adb
```

token生成示例

```
public class Test {
    private static final String MAC_NAME = "HmacSHA1";

    public static byte[] HmacSHA1Encrypt(String encryptText, String encryptKey) throws Exception {
        byte[] data = encryptKey.getBytes();
        //根据给定的字节数组构造一个密钥，第二参数指定一个密钥算法的名称。
        SecretKey secretKey = new SecretKeySpec(data, MAC_NAME);
        //生成一个指定 Mac 算法 的 Mac 对象
        Mac mac = Mac.getInstance(MAC_NAME);
        //用给定密钥初始化 Mac 对象
        mac.init(secretKey);
        byte[] text = encryptText.getBytes();
        //完成 Mac 操作
        return mac.doFinal(text);
    }

    public static String parseByte2HexStr(byte[] buf) {
        if (null == buf) {
            return null;
        }
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < buf.length; i++) {
```

```
            String hex = Integer.toHexString(buf[i] & 0xFF);
            if (hex.length() == 1) {
                hex = '0' + hex;
            }
            sb.append(hex.toUpperCase());
        }
        return sb.toString().toLowerCase();
    }

    public static String assembleToken(String path, String timestamp, String method, String accessKey, String a
ccessSecret) throws Exception {
        String encodePath = URLEncoder.encode(path, "UTF-8");
        String data = path + "\n" + timestamp + "\n" + method;
        String signature = parseByte2HexStr(HmacSHA1Encrypt(data, accessSecret));
        return "accessKey=" +
                accessKey +
                "&path=" +
                encodePath +
                "&timestamp=" +
                timestamp +
                "&method=" +
                method +
                "&sign=" +
                signature;
    }

    public static void main(String[] args) throws Exception {
        String path = "/addDevice";
        String timestamp = String.valueOf(new Date().getTime());
        String method = "SHA1";
        String accessKey = "qzJ2UCE86Fd14hRG1LzrkT7w";
        String accessSecret = "yeJEIAwLx0ezct1EK1hrbWOaAhuwAQ";
        String token = assembleToken(path, timestamp, method, accessKey, accessSecret);
        System.out.println("Authorization:" + token);
    }
}
```

如何使用token

由上述计算得到token之后，在此后的API接口的http请求头中添加如下参数即可访问。

Headers:

参数名称	参数值	是否必须
Authorization	{token}	是

获取请求地址和端口

此模块所有的接口请求地址和端口，都可以在平台"产品开发"-选择产品-"产品概览"中的"应用接入信息中获取。如图所示。

北向开发demo

北向应用demo下载

- 新增设备
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

新增设备

调用此接口，新增一个设备。

接口路径

/api/device/addDevice

请求方式

[POST]

请求参数

Headers:

参数名称	参数值	是否必须
Content-Type	application/json	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Body:

参数名称	参数类型	是否必须	说明
pk	String	是	将要添加设备的所属产品pk。
devId	String	是	将要添加的设备ID。设备ID支持英文字母、数字、下划线、中划线、点，4~32个字符。设备ID为设备标识，必须唯一，如IMEI、MAC等。如果产品为NB-IoT设备，请用IMEI号作为设备ID。
name	String	是	将要添加的设备的名称。

【示例】：

```
{
    "pk":"fb432************82bd8ebd",
    "devId":"3*******32j4",
    "name":"test01"
}
```

返回参数

名称	类型	是否必须	说明

name	String	否	设备名称
createdDate	number	否	创建时间（ms）
updatedDate	number	否	更新时间（ms）
enable	boolean	否	是否可用
pk	String	否	所属产品pk
devId	String	否	设备devId
online	boolean	否	在线状态
devSecret	String	否	设备密钥
loginTime	number	否	最近一次登录时间（ms）
logoutTime	number	否	最后一次登出时间（ms）
registerTime	number	否	注册时间（ms）
firstLoginTime	number	否	上电激活时间（ms）
deviceType	String	否	设备类型
ipInfo	object	否	IP信息
moduleVersion	String	否	模组固件版本
mcuVersion	String	否	mcu固件版本
configVersion	String	否	远程配置版本
batchName	String	否	批次名称
imei	String	否	设备imei

【示例】：


```
{
    "name": "test01",
    "createdDate": 1575655647001,
    "updatedDate": 1575655647001,
    "enable": true,
    "pk": "fb432*************82bd8ebd",
    "devId": "3********32j4",
    "online": false,
    "devSecret": "dcdaa**********************cd5",
    "loginTime": null,
    "logoutTime": null,
    "registerTime": 1575655647000,
    "firstLoginTime": null,
    "deviceType": "GENERAL",
    "ipInfo": null,
    "moduleVersion": null,
    "mcuVersion": null,
    "configVersion": null,
    "batchName": null,
    "imei": null
}
```

- 导入设备
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

导入设备

调用此接口，由设备ID组成的数组批量添加设备。

接口路径

/api/device/batchAddDevices

请求方式

[POST]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk

Headers:

参数名称	参数值	是否必须
Content-Type	application/json	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Body:

参数名称	参数类型	是否必须	说明
pk	String	是	将要添加设备的所属产品pk
devIds	Array	是	将要添加的设备ID组成的数组。设备ID支持英文字母、数字、下划线、中划线、点，4~32个字符。设备ID为设备标识，必须唯一，如IMEI、MAC等。如果产品为NB-IoT设备，请用IMEI号作为设备ID。

【注意】数组长度最大为100。

返回参数

名称	类型	是否必须	说明
batchName	String	是	批次名称
batchNumber	number	是	本次调用成功添加的设备数量

【示例】：

```
{
    "batchName": "20191207013747",
    "batchNumber": 2
}
```

- 获取批次下所有设备
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

获取批次下所有设备

调用此接口，根据批次名称获取设备列表。

接口路径

/api/device/getBatchDevices/{pk}/{batchName}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
batchName	String	是	批次名称

Headers:

参数名称	参数值	是否必须
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

返回参数

名称	类型	是否必须	说明
name	String	否	设备名称
createdDate	number	否	创建时间（ms）
updatedDate	number	否	更新时间（ms）
enable	boolean	否	是否可用
pk	String	否	所属产品pk
devId	String	否	设备devId
online	boolean	否	在线状态
devSecret	String	否	设备密钥
loginTime	number	否	最近一次登录时间（ms）
logoutTime	number	否	最后一次登出时间（ms）

registerTime	number	否	注册时间（ms）
firstLoginTime	number	否	上电激活时间（ms）
deviceType	String	否	设备类型
ipInfo	object	否	IP信息
moduleVersion	String	否	模组固件版本
mcuVersion	String	否	mcu固件版本
configVersion	String	否	远程配置版本
batchName	String	否	批次名称
imei	String	否	设备imei

【示例】：


```
[{
    "name": "22********0005",
    "createdDate": 1574672227996,
    "updatedDate": 1574758673843,
    "enable": true,
    "pk": "13fcf****************f4e3e5a",
    "devId": "22*********05",
    "online": false,
    "devSecret": "89ef*************262545",
    "loginTime": 1574672273018,
    "logoutTime": 1574758673842,
    "registerTime": 1574672227950,
    "firstLoginTime": 1574672273018,
    "deviceType": "GENERAL",
    "ipInfo": {
        "ip": "223.104.255.114",
        "country": "中国",
        "province": "广东省",
        "city": "广州市"
    },
    "moduleVersion": null,
    "mcuVersion": null,
    "configVersion": null,
    "batchName": null,
    "imei": "229*********05"
}]
```

- 更改设备名称
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

更改设备名称

调用此接口，更新设备名称。

接口路径

/api/device/updateName

请求方式

[PUT]

请求参数

Headers:

参数名称	参数值	是否必须
Content-Type	application/json	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Body:

参数名称	参数类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID
name	String	是	设备名称

【示例】：

```
http://1*1.*1.*1.*1:8***/api/device/updateName
```

```
{
    "pk": "bfff1c************8b8c6b9f7",
    "devId": "10*****02",
    "name": "test7647"
}
```

返回参数

Status Code: 200 （无返回值）

- 查询设备详情
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询设备详情

调用此接口，查看设备详情。

接口路径

/api/device/deviceInfo/{pk}/{devId}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

【示例】：

```
http://1*3.*9.*1.**2:8***/api/device/deviceInfo/13fcf***********f4e3e5a/22******0005
```

返回参数

名称	类型	是否必须	说明
name	String	否	设备名称
createdDate	number	否	创建时间（ms）
updatedDate	number	否	更新时间（ms）
enable	boolean	否	是否可用
pk	String	否	所属产品pk
devId	String	否	设备devId
online	boolean	否	在线状态

devSecret	String	否	设备密钥
loginTime	number	否	最近一次登录时间（ms）
logoutTime	number	否	最后一次登出时间（ms）
registerTime	number	否	注册时间（ms）
firstLoginTime	number	否	上电激活时间（ms）
deviceType	String	否	设备类型
ipInfo	object	否	IP信息
moduleVersion	String	否	模组固件版本
mcuVersion	String	否	mcu固件版本
configVersion	String	否	远程配置版本
batchName	String	否	批次名称
imei	String	否	设备imei

【示例】：


```
{
    "name": "22********0005",
    "createdDate": 1574672227996,
    "updatedDate": 1574758673843,
    "enable": true,
    "pk": "13fcf****************f4e3e5a",
    "devId": "22*********05",
    "online": false,
    "devSecret": "89ef*************262545",
    "loginTime": 1574672273018,
    "logoutTime": 1574758673842,
    "registerTime": 1574672227950,
    "firstLoginTime": 1574672273018,
    "deviceType": "GENERAL",
    "ipInfo": {
        "ip": "223.104.255.114",
        "country": "中国",
        "province": "广东省",
        "city": "广州市"
    },
    "moduleVersion": null,
    "mcuVersion": null,
    "configVersion": null,
    "batchName": null,
    "imei": "229*********05"
}
```

- 批量查询状态
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

批量查询状态

调用此接口，批量查询设备状态。

接口路径

/api/device/getDeviceStatus

请求方式

[POST]

请求参数

Headers:

参数名称	参数值	是否必须
Content-Type	application/json	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Body:

参数名称	参数类型	是否必须	说明
pk	String	是	产品pk
devIds	Array	是	待查询的devId列表

【示例】：

```
http://10.***.***.**5:8**2/api/device/getDeviceList/byDevIdList
```

```
{
    "pk":"cc0ec66*******b288428f3dc6",
    "devIds":[
        "b92275e***********",
        "101010m***********"
        ]
}
```

返回参数

名称	类型	是否必须	说明
pk	String	是	所属产品pk

devId	String	是	设备devId
online	boolean	是	设备状态
loginTime	number	是	设备登录时间（ms）

【示例】：

```
[
    {
        "pk": "cc0ec66*******b288428f3dc6",
        "devId": "b92275e***********",
        "online": false,
        "loginTime": null
    },
    {
        "pk": "cc0ec66287a9468f8e5eb288428f3dc6",
        "devId": "101010m***********",
        "online": false,
        "loginTime": 1574930066385
    }
]
```

- 删除设备
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

删除设备

调用此接口，删除设备。

接口路径

/api/device/delDevice/{pk}/{devId}

请求方式

[DELETE]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

【示例】:

```
http://10.**.**.**:***62/api/device/delDevice/cc0ec66********88428f3dc6/7bf12eb3e4af4076a*********
```

返回参数

Status Code: 200 （无返回值）

- 查询历史上下行数据
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询历史上下行数据

调用此接口，查询设备的历史上下行数据。

接口路径

/api/device/getDeviceHistoryData/{pk}/{devId}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Query:

参数名称	数据类型	是否必须	说明
startTime	number	是	查询开始时间（ms）
endTime	number	是	查询结束时间（ms）
page	integer	是	页数
size	integer	是	每页元素数量
action	String	否	查询的设备事件（如devLogin、devSend等）

【示例】：

```
http://**3.*9.*1.**2:8***/api/device/getDeviceHistoryData/9d7bc790429*******/Modb*****?page=0&size=10&startTime
=1574784000000&endTime=1574956799999&action=devLogin
```

返回参数

名称	类型	说明
pk	String	所属产品pk
devId	String	设备ID
timestamp	number	请求时间（ms）
frameType	string	帧类型
messageId	string	消息Id
action	string	操作
request	string	解析后的请求
rawRequest	string	原始请求
response	string	请求的结果
rawResponse	string	编码后的结果
payload	string	中转数据
ipInfo	object	IP信息

【示例】：


```
[
    {
            "pk": "fb4327cdcfe5432e82ad217572bd8ebd",
            "devId": "8237948938472348",
            "timestamp": 1576053322490,
            "frameType": "DEV_UP",
            "messageId": "a230011e8922404dba9792afc53acf87",
            "action": "devSend",
            "request": "{\"action\": \"devSend\", \"msgId\": 0, \"devId\": \"8237948938472348\", \"data\": {\"cmd\": \"reportAll\", \"params\": {\"light\": 50, \"power\": 1, \"hum\": 23}}, \"pk\": \"fb4327cdcfe5432e82ad217572bd8ebd\"}",
            "rawRequest": "{\"action\":\"devSend\",\"msgId\":0,\"pk\":\"fb4327cdcfe5432e82ad217572bd8ebd\",\"devId\":\"8237948938472348\",\"data\":{\"cmd\":\"reportAll\",\"params\":{\"power\":1,\"light\":50,\"hum\":23}}}",
            "response": "{\"code\": 0, \"msgId\": 0, \"devId\": \"8237948938472348\", \"action\": \"devSendResp\", \"pk\": \"fb4327cdcfe5432e82ad217572bd8ebd\", \"desc\": \"success\"}",
            "rawResponse": "",
            "payload": "",
            "ipInfo": {
                "ip": "112.17.116.161",
                "country": "中国",
                "province": "浙江省",
                "city": "杭州市"
            }
        }
]
```

- 查询设备影子
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询设备影子

调用此接口，查看设备影子。

接口路径

/api/device/getSnapshot

请求方式

[GET]

请求参数

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Query:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

【示例】：

```
http://**3.*9.*1.**2:8***/api/device/getSnapshot?pk=848f4fc***********85e63e35159783&devId=t*****1
```

返回参数

名称	类型	是否必须	说明
pk	String	是	所属产品pk
devId	String	是	设备ID
timestamp	number	是	时间（ms）
data	object	是	设备数据对象
data.cmd	String	是	最后一次上报命令
data.params	object	是	设备参数

【示例】：


```
{
    "pk": "848f4fc************85e63e35159783",
    "devId": "t*****1",
    "timestamp": 1578303962164,
    "data": {
        "cmd": "reportPower",
        "params": {
            "power": 0
        }
    }
}
```

- 查询设备指标趋势
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询设备指标趋势

调用此接口，查询设备某一参数的指标趋势。

接口路径

/api/deviceStat/kvlog/{pk}/{devId}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Query:

参数名称	数据类型	是否必须	说明
startTime	number	是	查询开始时间（ms）
endTime	number	是	查询结束时间（ms）
key	String	是	所查询设备的参数标识

【注意】：

其中 `key` 必须为number型参数。

【示例】：

```
http://**3.*9.*1.**2:8***/api/deviceStat/kvlog/848f4fc**********85e63e35159783/t*****1?startTime=1578878730279&endTime=1578882330279&key=power
```

返回参数

名称	类型	说明
key	String	设备参数标识
value	number	参数值
timestamp	number	时间节点（ms）

【注意】：

查询结果最多返回 `1000条` ，若查询结果为 `1000条` 时建议修改查询时间范围（如：将总时间段分成若干个时间段查询）。

【示例】：

```
[
    {
        "key": "power",
        "value": 1.0,
        "timestamp": 1578882269208
    },
    {
        "key": "power",
        "value": 1.0,
        "timestamp": 1578882271439
    },
    {
        "key": "power",
        "value": 0.0,
        "timestamp": 1578882274379
    },
    {
        "key": "power",
        "value": 0.0,
        "timestamp": 1578882275028
    }
]
```

- 查询设备指标聚合
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询设备指标聚合

调用此接口，查询设备某一参数的指标聚合。

接口路径

/api/deviceStat/agglog/{pk}/{devId}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Query:

参数名称	数据类型	是否必须	说明
startTime	number	是	查询开始时间（ms）
endTime	number	是	查询结束时间（ms）
key	String	是	所查询设备的参数标识
span	String	是	查询聚合力度

【注意】：

1. 其中 `key` 必须为number型参数；
2. 其中聚合力度 `span` 可选值为 `5min` 、 `1hour` 和 `1day` ，对应的最大时间范围为 `一周` 、 `一个月` 和 `一年` 。即，当 `span` 值选择为 `5min` 时， `startTime` 和 `endTime` 差值不能大于一周。

【示例】：

```
http://**3.*9.*1.**2:8***/api/deviceStat/agglog/848f4fc**********85e63e35159783/t*****1?startTime=1578878730279&endTime=1578882330279&key=power&span=5min
```

返回参数

名称	类型	说明
avgList	array	参数平均值聚合列表
maxList	array	参数最大值聚合列表
minList	array	参数最小值聚合列表
value	number	对应聚合值
timestamp	number	聚合时间（ms）

【示例】：

```
{
    "avgList": [
        {
            "timestamp": 1578882300000,
            "value": 0.3333333432674408
        },
        {
            "timestamp": 1578882000000,
            "value": 0.25
        }
    ],
    "maxList": [
        {
            "timestamp": 1578882300000,
            "value": 1.0
        },
        {
            "timestamp": 1578882000000,
            "value": 1.0
        }
    ],
    "minList": [
        {
            "timestamp": 1578882300000,
            "value": 0.0
        },
        {
            "timestamp": 1578882000000,
            "value": 0.0
        }
    ]
}
```

- 下发控制命令
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

下发控制命令

调用此接口，下发设备控制命令。

接口路径

/api/device/cloudSend/{pk}/{devId}

请求方式

[POST]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
pk	String	是	产品pk
devId	String	是	设备ID

Headers:

参数名称	参数值	是否必须
Content-Type	application/json	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Body:

参数名称	参数类型	是否必须	说明
cmd	string	否	命令标识符
params	object	否	参数值

【示例】:

```
{
    "cmd":"setPower",
    "params":{
        "power":1
    }
}
```

返回参数

名称	类型	是否必须	说明
action	String	是	指令类型
msgId	number	是	消息Id
pk	String	是	所属产品pk
devId	String	是	设备ID
code	number	是	指令码，表示指令下发状态。
desc	String	是	指令下发状态描述。
messageId	String	是	指令Id

【示例】：

```
{
    "action": "cloudSendResp",
    "msgId": 0,
    "pk": "fb4327c***************2bd8ebd",
    "devId": "823**********2348",
    "code": 0,
    "desc": "success",
    "messageId": "8c83aaed5*************599246"
}
```

- 查询命令状态
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询命令状态

调用此接口，查看某条下发命令的状态信息。

接口路径

/api/device/cloudSendMsgInfo/{messageId}

请求方式

[GET]

请求参数

路径参数:

参数名称	数据类型	是否必须	说明
messageId	String	是	所查询命令的ID

Headers:

参数名称	参数值	是否必须
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

【示例】：

```
http://1*3.*9.*1.**2:8***/api/device/cloudSendMsgInfo/7b4ab3ee**************ac8a7d5c9
```

返回参数

名称	类型	是否必须	说明
messageId	String	是	命令ID
code	number	是	错误码
desc	String	是	命令状态描述
createTime	number	是	命令创建时间（ms）

【示例】：

```
{
    "messageId": "7b4ab3ee**************ac8a7d5c9",
    "code": 0,
    "desc": "成功",
```

```
    "createTime": 1576485598335
}
```

- 查询历史控制命令
  - 接口路径
  - 请求方式
  - 请求参数
  - 返回参数

查询历史控制命令

调用此接口，查询设备的历史控制命令。

接口路径

/api/device/cloudSendMsgList

请求方式

[GET]

请求参数

Headers:

参数名称	参数值	是否必须
Content-Type	application/x-www-form-urlencoded	是
Authorization	{token}	是

- 其中token获取方式参考使用AccessKey计算token。

Query:

参数名称	数据类型	是否必须	说明
page	integer	是	页数
size	integer	是	每页元素数量
pk	String	是	所属产品pk
devId	String	是	设备ID
startTime	number	否	查询开始时间（ms）
endTime	number	否	查询结束时间（ms）

【示例】：

```
http://**3.*9.*1.**2:8***/api/device/cloudSendMsgList?page=0&size=10&pk=848f4fc************835159783&devId=t**
*******801
```

返回参数

名称	类型	说明
messageId	String	命令ID

code	number	错误码
desc	String	命令状态描述
createTime	number	命令创建时间（ms）

【示例】：

```
[
    {
        "messageId": "fa8b17*************9c08273eb16",
        "code": 1008,
        "desc": "device not online",
        "createTime": 1577095718969
    }
]
```

- 错误码
  - 使用建议
  - 通用错误码列表
  - 设备通用错误码列表
  - NB-IoT设备错误码列表
  - HTTP设备错误码列表

错误码

文档列举调用api出错时，返回的错误信息。

使用建议

利用本文档左上角的"搜索"功能直接查询返回信息中的 `code` 参数可以较为快捷、精确地找到错误码对应的详细信息。

通用错误码列表

错误描述	错误名称	code	desc
系统内部错误	INTERNAL_ERROR	500	"internal error"
IMEI格式错误	ERROR_IMEI_FORMAT	100004	"imei format error"
用户不存在	ERROR_USER_NOT_EXIST	100103	"user not exist"
没有操作权限	ERROR_NO_PERMISSION	100107	"this user has no permission"
产品不存在	ERROR_PRODUCT_NOT_EXIST	100201	"product not exist"
品类不存在	ERROR_CATEGORY_NOT_EXIST	100301	"category not exist"
devId列表查询数量过多	ERROR_DEVICE_LIMIT_EXIST	100504	"size of devId list too big"
设备id列表为空	ERROR_DEVICE_LIST_IS_NULL	100804	"devId list should not null"

设备通用错误码列表

错误描述	错误名称	code	desc
系统内部错误	INTERNAL_ERROR	500	"internal error"
未知	UNDEFINED	65535	"undefined"
参数错误	PARAM_INVALID	4000	"param invalid"
json 解析错误	JSON_PARSE_ERROR	1001	"json parse error"
指令（动作）不支持	ACTION_NOT_SUPPORT	1002	"action not support"
设备未登录	DEVICE_NOT_LOGIN	1003	"device not login"
			"device not

			exist"
设备登录校验失败	DEVICE_LOGIN_AUTH_FAILED	1005	"device login auth failed"
设备不支持子设备	DEVICE_NOT_SUPPORT_SUB_DEV	1006	"device not support sub device"
设备添加拓扑结构不允许形成环	DEVICE_TOPO_NOT_ALLOW_CIRCLE	1007	"device topo not allow circle"
设备不在线	DEVICE_NOT_ONLINE	1008	"device not online"
设备被禁用	DEVICE_DISABLED	1009	"device is disabled"
设备没有这个子设备	DEVICE_NOT_HAS_SUB_DEVICE	1010	"device not has the sub device"
产品不存在	PRODUCT_NOT_EXIST	1104	"product not exist"
产品不支持远程配置	PRODUCT_NOT_SUPPORT_REMOTE_CONFIG	1105	"product not support remote config"
产品不存在	PRODUCT_DELETED	1106	"product deleted"
imei 不合法	DEVICE_IMEI_INVALID	1107	"device imei invalid"
imei 已经存在	DEVICE_IMEI_EXIST	1108	"device imei exist"
rule select 参数不正确	RULE_SELECT_PARAM_INVALID	1109	"rule select param not valid"
规则表达式不合法	RULE_EXPRESSION_INVALID	1110	"rule expression not valid"
规则不存在	RULE_NOT_FOUND	1111	"rule not found"
规则 目标地址 http 不合法	RULE_DEST_HTTP_NOT_VALID	1112	"rule dest http url not valid"
协议参数不合法（包括 cmd）	MODEL_DATA_PARAM_INVALID	1113	"model params not valid"
配额已经存在	QUOTA_EXIST	1114	"quota exist"
配额不存在	QUOTA_NOT_FOUND	1115	"quota not found"
升级任务启用冲突	UPGRADE_TASK_ENABLE_CONFLICT	1116	"upgrade task enable conflict"
			"upgrade

升级任务启用存在循环	UPGRADE_TASK_ENABLE_CYCLE	1117	"upgrade task has cycle"
升级任务不存在	UPGRADE_TASK_NOT_FOUND	1118	"upgrade task not found"
么有可用的远程配置	AVAILABLE_REMOTE_CONFIG_NOT_FOUND	1119	"available remote config not found"
配额不足	QUOTA_INSUFFICIENT	1120	"quota insufficient"
协议参数名字重复	MODEL_DATA_PARAM_NAME_CONFLICT	1121	"model data param name conflict"
协议参数标识符重复	MODEL_DATA_PARAM_IDENTIFIER_CONFLICT	1122	"model data param identifier conflict"
协议命令名字符重复	MODEL_DATA_CMD_NAME_CONFLICT	1123	"model data cmd name conflict"
协议命令标识符重复	MODEL_DATA_CMD_IDENTIFIER_CONFLICT	1124	"model data cmd identifier conflict"
设备id不合法	DEV_ID_INVALID	1125	"devId invalid"
imei 已经存在	DEV_ID_EXIST	1126	"device id exist"
产品下还存在设备，不允许删除产品	PRODUCT_NOT_ALLOW_DELETE_DEVICE_EXIST	1127	"product has devices not allow to delete"
子设备不允许直连	DEVICE_NOT_ALLOW_TO_DIRECT_CONNECT	1128	"device(sub) not allow to direct connect"
设备所属网关不在线	DEVICE_GATEWAY_NOT_ONLINE	1129	"device's gateway not online"
产品编解码插件（脚本）未运行，可能不存在或者编译失败	PRODUCT_SCRIPT_NOT_RUNNING	1130	"product script not running or not exist"
产品编码插件编码错误	PRODUCT_SCRIPT_ENCODE_ERROR	1131	"product script encode error"
产品编码插件解码码错误	PRODUCT_SCRIPT_DECODE_ERROR	1132	"product script decode error"

产品不支持脚本	PRODUCT_NOT_SUPPORT_SCRIPT	1133	support script"
产品不支持脚本	PRODUCT_SCRIPT_RETURN_VALUE_INVALID	1134	"product script return value invalid"
产品脚本编译失败	PRODUCT_SCRIPT_COMPILE_ERROR	1135	"product script compile error"

NB-IoT设备错误码列表

错误描述	错误名称	code	desc
下发命令等待	CLOUD_SEND_PENDING	2001	"等待"
下发命令失败	CLOUD_SEND_FAIL	2002	"失败"
下发命令超时	CLOUD_SEND_TIME_OUT	2003	"超时"
下发命令参数错误	CLOUD_SEND_ENCODE_FAIL	2004	"参数错误

HTTP设备错误码列表

错误描述	错误名称	code	desc
参数错误	PARAMETER_ERROR	3001	"illegal parameter"
token错误	TOKEN_ERROR	3002	"token is error!"
下发等待	CIOUD_SEND_PENDING	3003	"等待"
下发失败	CLOUD_SEND_FAIL	3004	"失败"
下发超时	CLOUD_SEND_TIME_OUT	3005	"超时"

- 数据订阅

数据订阅

IoT OS支持将设备上报到平台的数据通过HTTP或Topic流转至指定的服务地址或Topic中。用户可以通过自定义数据筛选规则和转发内容，将产品下的设备所有类型消息，进行筛选并按指定格式转发到指定服务地址或Topic中。平台提供数据流转服务详情可见服务端订阅和规则引擎。

- 资料下载

资料下载

点击按钮下载相应资料

技术白皮书

北向应用demo