IoT 边缘 IoTEdge

开发指南

文档版本　1.1

发布日期　2024-10-21

华为云计算技术有限公司

版权所有 © 华为云计算技术有限公司 2024。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

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

注意

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

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

目 录

1 开发前必读......1

2 资源获取......3

3 应用侧开发......4

3.1 调试 API......4

4 插件侧开发......7

4.1 概述......7

4.2 架构介绍......9

4.2.1 数据处理(OT 应用)......9

4.2.2 协议转换(驱动应用)......10

4.2.3 工业子系统接入(IT 应用)......13

4.3 集成 ModuleSDK(Java)......14

4.3.1 内部架构......14

4.3.2 开发指导......14

4.3.2.1 接口函数......14

4.3.2.1.1 客户端通用方法说明......14

4.3.2.1.2 AppClient 方法说明......16

4.3.2.1.3 DriverClient 方法说明......19

4.3.2.1.4 ItClient 方法说明......23

4.3.2.1.5 DcClient 方法说明......27

4.3.2.2 方案概述......28

4.3.2.3 前提条件......29

4.3.2.4 创建工程......29

4.3.2.5 项目打包......32

4.3.2.6 制作镜像包或插件包......33

4.3.2.7 添加应用......35

4.3.2.8 发布应用......40

4.3.2.9 如何使用......41

4.3.3 下载 demo......41

4.3.4 集成 ModuleSDK 进行数据处理......42

4.3.4.1 操作场景......42

4.3.4.2 代码解析......42

4.3.4.3 注册节点......44

 版权所有 © 华为云计算技术有限公司

4.3.4.4 创建产品 44
4.3.4.5 修改代码 46
4.3.4.6 项目打包 46
4.3.4.7 制作镜像包 46
4.3.4.8 创建应用 46
4.3.4.9 部署应用 49
4.3.4.10 添加边缘设备 49
4.3.4.11 设备接入 50
4.3.5 集成 ModuleSDK 进行工业子系统接入 55
4.3.5.1 操作场景 55
4.3.5.2 代码解析 55
4.3.5.3 项目打包 59
4.3.5.4 制作镜像包 59
4.3.5.5 添加应用 59
4.3.5.6 注册节点 59
4.3.5.7 创建 API 61
4.3.5.8 网关应用创建及绑定 66
4.3.5.9 添加数据端点 67
4.3.5.10 部署应用 68
4.3.5.11 使用 69
4.3.6 集成 ModuleSDK 进行协议转换 72
4.3.6.1 操作场景 72
4.3.6.2 代码解析 72
4.3.6.3 注册节点 75
4.3.6.4 设备建模 75
4.3.6.5 项目打包 77
4.3.6.6 制作镜像包 77
4.3.6.7 添加应用 78
4.3.6.8 部署应用 78
4.3.6.9 添加边缘设备 79
4.3.6.10 设备接入 79
4.3.7 集成 ModuleSDK 进行 OT 数采 80
4.3.7.1 操作场景 80
4.3.7.2 代码解析 80
4.3.7.3 注册节点 82
4.3.7.4 设备建模&发放 82
4.3.7.5 项目打包 82
4.3.7.6 制作镜像包 83
4.3.7.7 添加应用 83
4.3.7.8 部署应用 83
4.3.7.9 OT 数采配置 83
4.3.7.10 查看采集结果 83

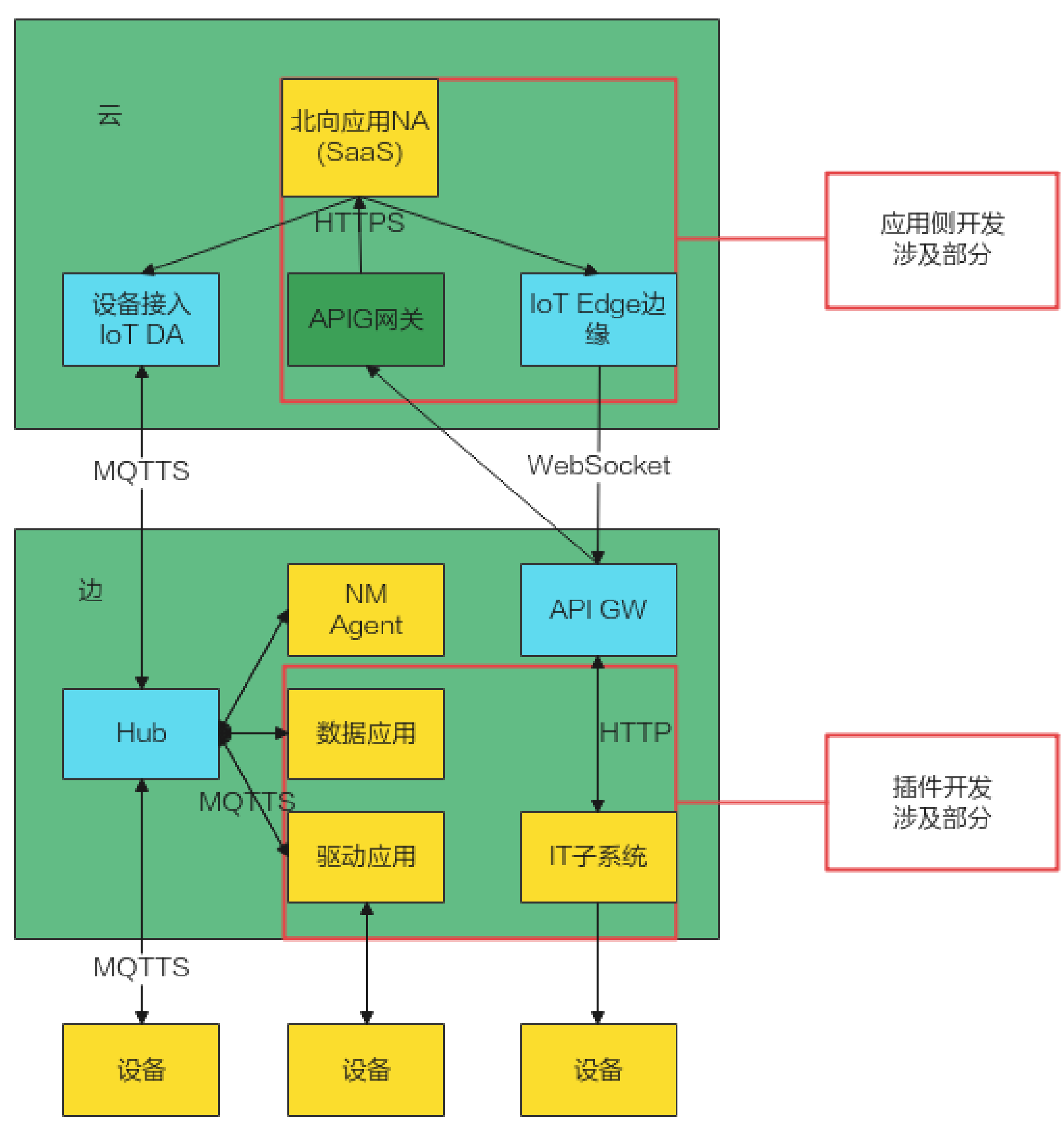
 版权所有 © 华为云计算技术有限公司

4.3.8 集成 ModuleSDK 进行进程应用的开发 83
4.3.8.1 操作场景 83
4.3.8.2 代码解析 83
4.3.8.3 注册节点 84
4.3.8.4 设备建模&发放 84
4.3.8.5 项目打包 84
4.3.8.6 制作插件包 84
4.3.8.7 添加应用 85
4.3.8.8 部署应用 86
4.4 集成 ModuleSDK(C) 86
4.4.1 内部架构 86
4.4.2 开发指导 88
4.4.2.1 接口函数 88
4.4.2.2 前提条件 109
4.4.2.3 创建工程 109
4.4.2.4 生成可执行文件 110
4.4.2.5 制作镜像包或插件包 111
4.4.2.6 添加应用 115
4.4.2.7 发布应用 119
4.4.2.8 如何使用 119
4.4.3 下载 Demo 120
4.4.4 集成 ModuleSDK 进行数据处理 122
4.4.4.1 场景说明 123
4.4.4.2 代码解析 123
4.4.4.3 注册节点 125
4.4.4.4 创建产品 125
4.4.4.5 修改代码 127
4.4.4.6 项目打包 128
4.4.4.7 制作镜像包或插件包 129
4.4.4.8 创建应用 129
4.4.4.9 部署应用 130
4.4.4.10 添加边缘设备 131
4.4.4.11 设备接入 132
4.4.4.12 查看 SDK 运行日志 136
4.4.5 ModuleSDK-C Demo 展示 137
4.4.5.1 Demo1 137
4.4.5.2 Demo2 148
4.4.5.3 Demo3 157
4.5 集成 ModuleSDK(C#) 159
4.5.1 内部架构 159
4.5.2 开发指导 159
4.5.2.1 方案概述 160

4.5.2.2 前提条件 160
4.5.2.3 创建工程 160
4.5.2.4 项目构建发布 161
4.5.2.5 制作镜像包或插件包 164
4.5.2.6 添加应用 167
4.5.2.7 发布应用 173
4.5.2.8 接口方法 173
4.5.3 下载 Demo 180
4.5.4 集成 ModuleSDK 进行数据采集 181
4.5.4.1 场景说明 181
4.5.4.2 代码解析 181
4.5.4.3 注册节点 183
4.5.4.4 设备建模&发放 183
4.5.4.5 项目打包 183
4.5.4.6 制作镜像包或插件包 183
4.5.4.7 添加应用 183
4.5.4.8 部署应用 187
4.5.4.9 OT 数采配置 187
4.5.4.10 查看采集结果 187

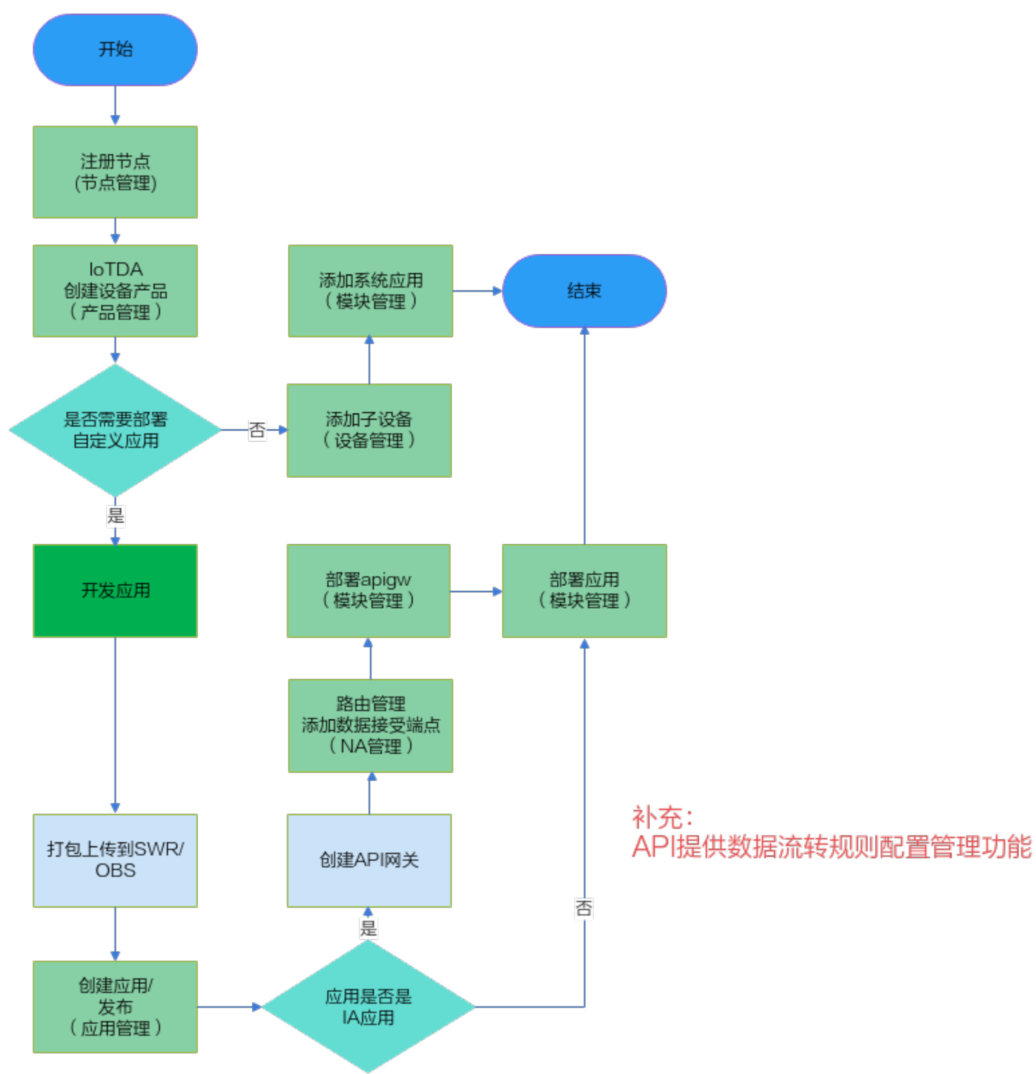
1 开发前必读

基于IoT边缘去实现一个物联网解决方案，涉及使用IoT边缘服务开发的部分在架构上涉及部分为：



在业务流程中（下图只涉及到已提供API的业务，非全部功能）涉及的部分为：

版权所有 © 华为云计算技术有限公司



开发操作	开发说明
应用侧开发	流程图中涉及淡绿色部分（在用户的控制系统、大屏监控系统等应用中调用IoTEdge API，实现如应用的创建修改删除等操作。例如调用创建应用API,可以实现与在云端手动创建同样的效果）。
插件开发	流程图中涉及深绿色部分（开发自定义应用部署到边侧实现数据处理（OT应用）、协议转换（驱动应用）、接入IT子系统（IT应用）。

业务概览

开通IoT边缘服务后，使用设备接入服务的完整流程如下图所示，主要分为应用侧开发、插件开发。

- 应用侧开发：IoTEdge提供了丰富的API，如节点管理、应用管理，开发者可以在应用侧(如管理系统、大屏监控系统)接入API实现在应用侧管理的功能。
- 插件的开发：ModuleSDK开发插件应用在边侧进行协议转换、设备上传的数据处理、接入IT子系统。

版权所有 © 华为云计算技术有限公司

2 资源获取

证书文件

在设备和对接边缘节点场景中，需要在设备侧集成相应证书。

资源包名	描述	下载路径
Module SDK(java)	利用ModuleSDK开发插件在边侧进行协议转换、设备上传的数据处理、接入IT子系统。	**ModuleSDK（Java版）**
Module SDK(C)	Module SDK是开发边缘运行应用（插件）所必须的工具包，提供数据处理、协议转换、IT子系统接入等功能，开发完成后，通过选择打包方式来决定是容器化部署还是进程化部署。	**ModuleSDK_C_latest**(包括x86_64, arm32, arm64版本,下载后解压选择对应版本)
Module SDK(C#)	利用ModuleSDK开发插件在边侧进行数据采集等功能。	**ModuleSDK_CSharp**
MQTT.fx	MQTT.fx是MQTT设备模拟软件。	**MQTT.fx下载地址**
plt-device-ca	设备通过mqtts协议与边缘节点建立连接时，该证书用于校验边缘节点的身份。	**证书文件**
Modbus Slave	Modbus Slave是Modbus Slave设备模拟软件。	**Modbus Slave下载地址**

版权所有 © 华为云计算技术有限公司

3 应用侧开发

3.1 调试 API

使用paw/postman等工具调试API。

前提条件

1. 在调用API之前需要**获取token**。
2. 调用查询应用列表API。

操作步骤

步骤1 查看API说明（关注API地址、请求类型、请求参数、返回参数），**IoTEdge文档**>API参考>API>应用管理>查询应用列表。

其包含两个必选参数

X-Auth-Token，即前面获取的token。

project_id，项目ID。获取方式参照**获取项目ID**

版权所有 © 华为云计算技术有限公司

名称	必选/可选	类型	位置	说明
X-Auth-Token	必选	String(0-20000)	Header	用户Token。通过调用IAM服务 获取IAM用户Token接口获取，接口返回的响应消息头中"X-Subject-Token"就是需要获取的用户Token。简要的获取方法样例请参见 Token认证。
project_id	必选	String(1-64)	Path	项目ID。获取方法请参见 获取项目ID。
edge_app_id	可选	String(4-32)	Query	应用ID搜索关键字
offset	可选	Integer[0,1000000]	Query	查询的起始位置，取值范围为非负整数，默认为0
limit	可选	Integer[0,1000000]	Query	每页记录数，默认值为10，取值区间为1-1000
app_type	可选	String(0-64)	Query	应用id搜索关键字
function_type	可选	String(0-64)	Query	功能类型

在paw/postman中输入以下内容

请求方式：GET

请求地址：https://iotedge-api.cn-north-4.myhuaweicloud.com/v2/{project_id}/edge-apps

需要将{project_id}替换为实际项目ID。

Headers:

Content-Type：application/json

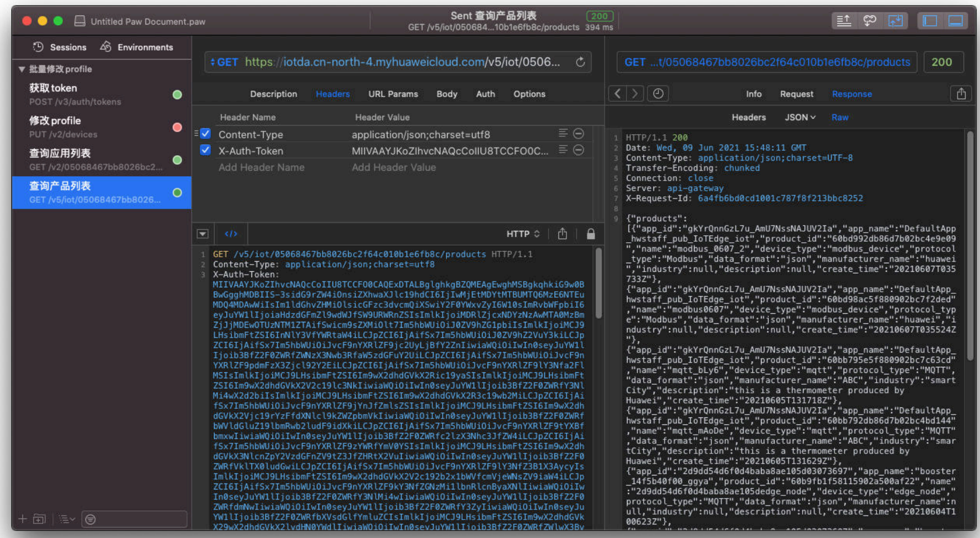
X-Auth-Token：输入前面获取的token。

body:

该API请求示例中未说明，不填写。

步骤2 单击Enter发送请求,得到应用列表。

版权所有 © 华为云计算技术有限公司



将返回数据格式化以便于查看。


```
on_type":"ON_PREMISE_INTEGRATION","deploy_type":"docker"},
{"edge_app_id":"App_47hxl","description":null,"create_time":"2021-04-
16T09:26:01Z","update_time":"2021-04-
16T09:26:01Z","last_published_version":null,"app_type":"USER","function_
type":"DATA_PROCESSING","deploy_type":"docker"},
{"edge_app_id":"iot_edge_process","description":null,"create_time":"2021
-04-15T03:36:13Z","update_time":"2021-04-
15T03:36:13Z","last_published_version":"1.0.2","app_type":"USER","functi
on_type":"DATA_PROCESSING","deploy_type":"process"},
{"edge_app_id":"iot_edge_app","description":null,"create_time":"2021-04-
14T09:30:48Z","update_time":"2021-04-
14T09:30:48Z","last_published_version":"222","app_type":"USER","function
_type":"DATA_PROCESSING","deploy_type":"docker"},
{"edge_app_id":"test","description":null,"create_time":"2021-04-
09T08:04:55Z","update_time":"2021-04-
09T08:04:55Z","last_published_version":null,"app_type":"USER","function_
type":"ON_PREMISE_INTEGRATION","deploy_type":"docker"},
{"edge_app_id":"test-ganwei-x86","description":null,"create_time":"2021-
04-06T08:03:47Z","update_time":"2021-04-
06T08:03:47Z","last_published_version":"1-0-
2","app_type":"USER","function_type":"DATA_PROCESSING","deploy_type":"do
cker"},{"edge_app_id":"test-jj","description":null,"create_time":"2021-
04-06T08:00:26Z","update_time":"2021-04-
06T08:00:26Z","last_published_version":"1-0-
0","app_type":"USER","function_type":"DATA_PROCESSING","deploy_type":"do
cker"},
{"edge_app_id":"sdfdssdfsdfsd","description":null,"create_time":"2021-
03-22T01:48:55Z","update_time":"2021-03-
22T01:48:55Z","last_published_version":null,"app_type":"USER","function_
type":"DATA_PROCESSING","deploy_type":"process"}]}
```


保留转义　可点击key和value值进行编辑


```
{
    "count":42,
    "page_info":{
        "offset":0,
        "limit":10
    },
    "edge_apps":[
        {
            "edge_app_id":"546456",
            "description":null,
            "create_time":"2021-06-07T12:17:41Z",
            "update_time":"2021-06-07T12:17:41Z",
            "last_published_version":null,
            "app_type":"USER",
            "function_type":"DATA_PROCESSING",
            "deploy_type":"docker"
        },
        {
            "edge_app_id":"iot-erp-integration",
            "description":null,
            "create_time":"2021-05-17T02:46:09Z",
            "update_time":"2021-05-17T02:46:09Z",
            "last_published_version":"1.1.7",
            "app_type":"USER",
            "function_type":"ON_PREMISE_INTEGRATION",
            "deploy_type":"docker"
        },
```


----结束

版权所有 © 华为云计算技术有限公司

4 插件侧开发

4.1 概述

简介

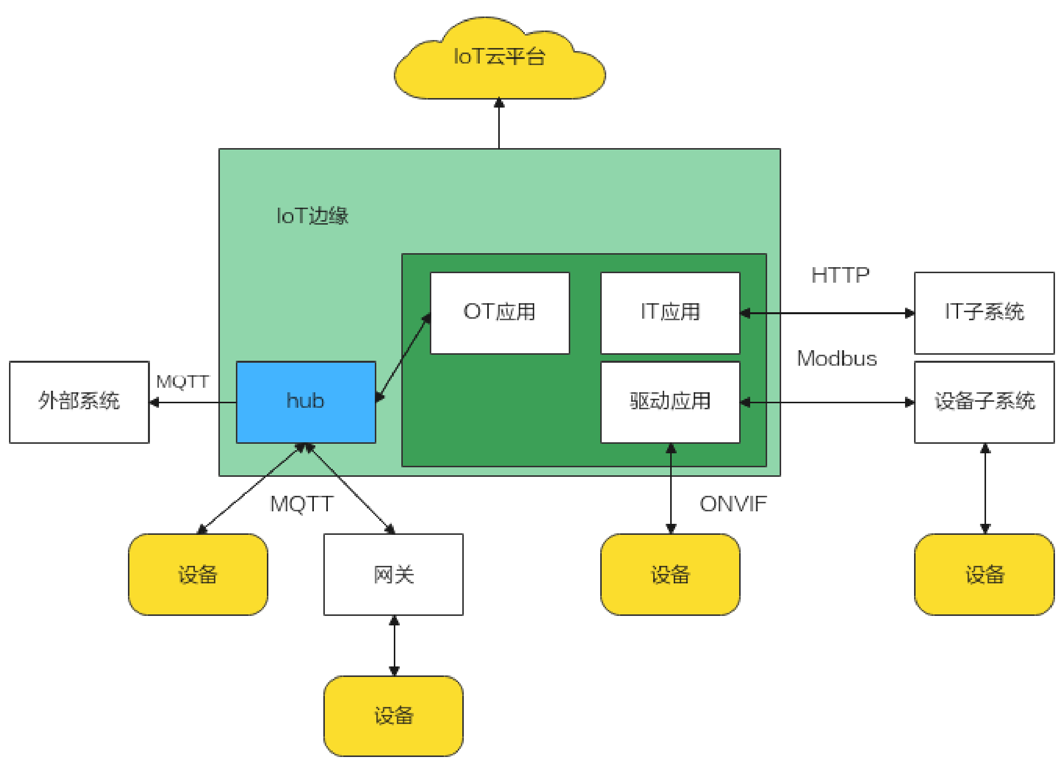
为解决用户自定义处理设备数据以及自定义协议设备快速接入IOT平台的诉求，IoT边缘提供ModuleSDK，用户可通过集成SDK让设备以及设备数据快速上云。IoT边缘平台应用功能有自定义处理设备数据（即数据处理），自定义协议设备快速接入（即协议解析），IT子系统接入(即IT应用)，并且支持容器化部署和安装包部署的方式。

集成ModuleSDK开发的应用被称为插件应用，其最终将被通过云部署到边缘节点。

集成ModuleSDK的应用分为：

- OT应用：数据处理类型的应用，实现了总线回调接口，IoT边缘云服务为OT应用提供总线对接能力、设备反向控制（命令）能力。
- 驱动应用：协议转换类型的应用，可将使用自定义协议的设备接入边缘计算。
- IT应用：实现工业子系统接入的应用，实现了对子系统配置同步、反向调用接口、以及数据采集能力。

版权所有 © 华为云计算技术有限公司



集成了SDK的应用称为插件，华为云市场提供了插件上架功能。您可以将开发的集成了SDK的应用上传到华为云市场来获取收益，也可以在华为云市场下载使用第三方插件。

使用场景

场景挑战

- 设备采集的数据信息量庞大但只需部分有效数据。部署在节点下的设备需要在本地判断情况以及时进行控制。
- 设备使用的协议不能接入边缘计算。
- 节点下的IT子系统众多，接入时不便于管理，配置繁琐。子系统采集的数据需要传输到用户的北向应用。

解决方案

通过开发集成ModuleSDK的应用，您可以：

- 使用ModuleSDK提供的AppClient开发OT应用对需要上传到云端的信息进行过滤上传。OT应用可以在本地对设备进行命令的下发。
- 使用ModuleSDK提供的DriverClient开发驱动应用进行协议转换。
- 使用ModuleSDK提供的ItClient开发IT应用实现子系统与云端配置的自动同步，也可以实现云端对子系统的配置管理。云端可以通过IT应用实现对子系统数据的采集。

运行 SDK 的配置要求

1、机器需要注册成IoT边缘节点，节点注册要求请参考节点安装限制。

2、节点部署IT应用需要购买工业网关资源包下的工业子系统采集服务。

版权所有 © 华为云计算技术有限公司

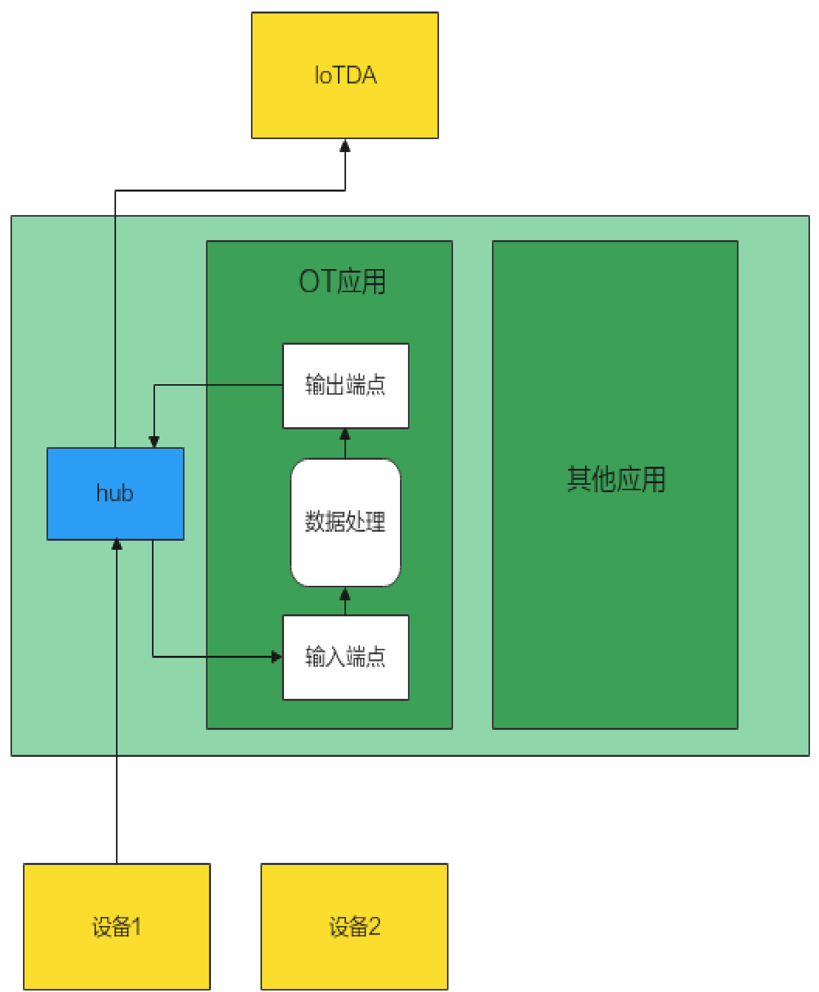
注意

因SDK使用IAM统一身份认证服务时需要校验时间，请在部署边缘节点后同步时区与时间。如出现认证问题，请检查时区与时间是否同步。

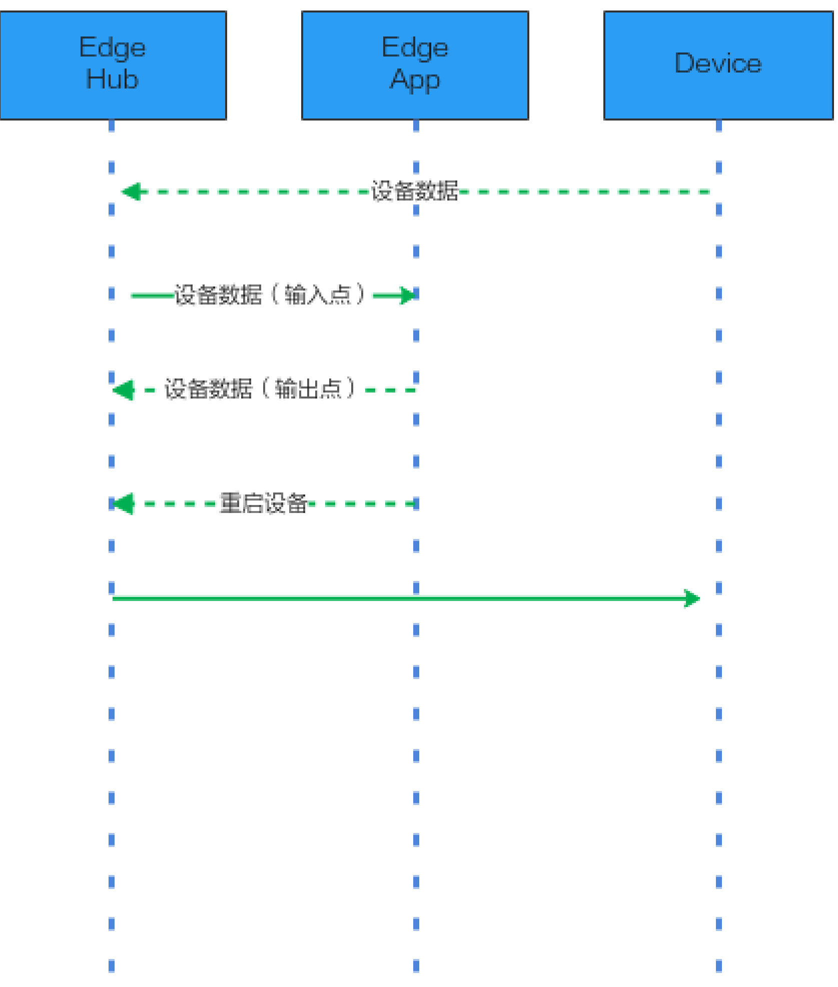
4.2 架构介绍

4.2.1 数据处理(OT 应用)

IoT边缘云服务为应用提供总线对接能力、设备命令下发能力。IoTEdge对应用的日志、数据存储目录进行统一配置，应用相关设置通过环境变量传递给应用。



版权所有 © 华为云计算技术有限公司



1. App从输入点接收来自总线的设备数据上报，对数据进行处理，将处理后的数据通过输出点发送到总线。
2. App也可以根据设备数据对设备进行反向控制，例如通过命令重启设备。

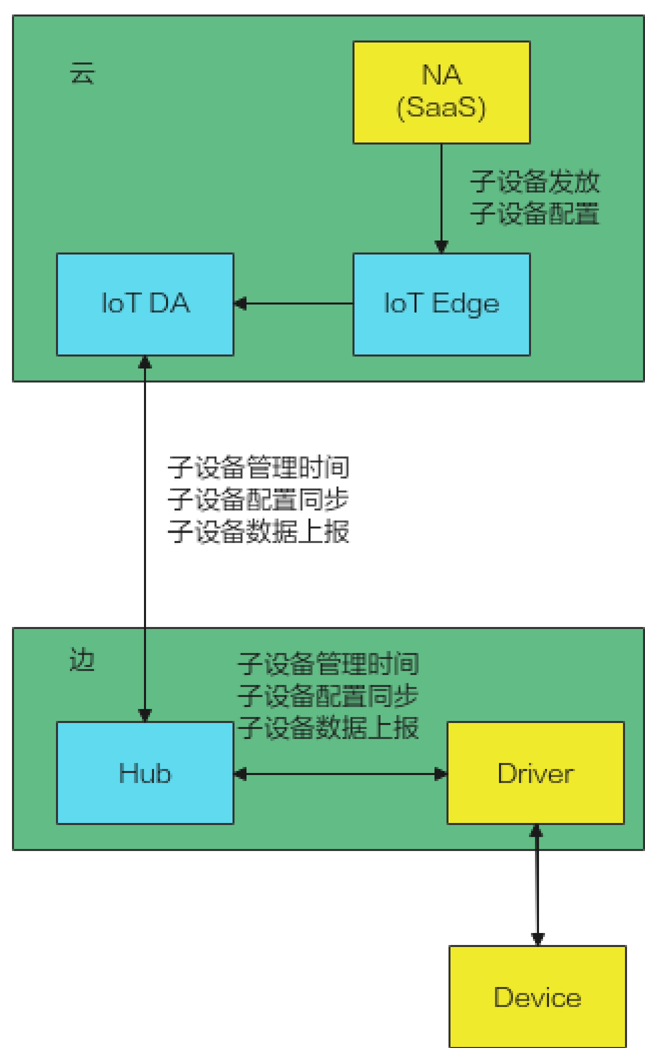
4.2.2 协议转换(驱动应用)

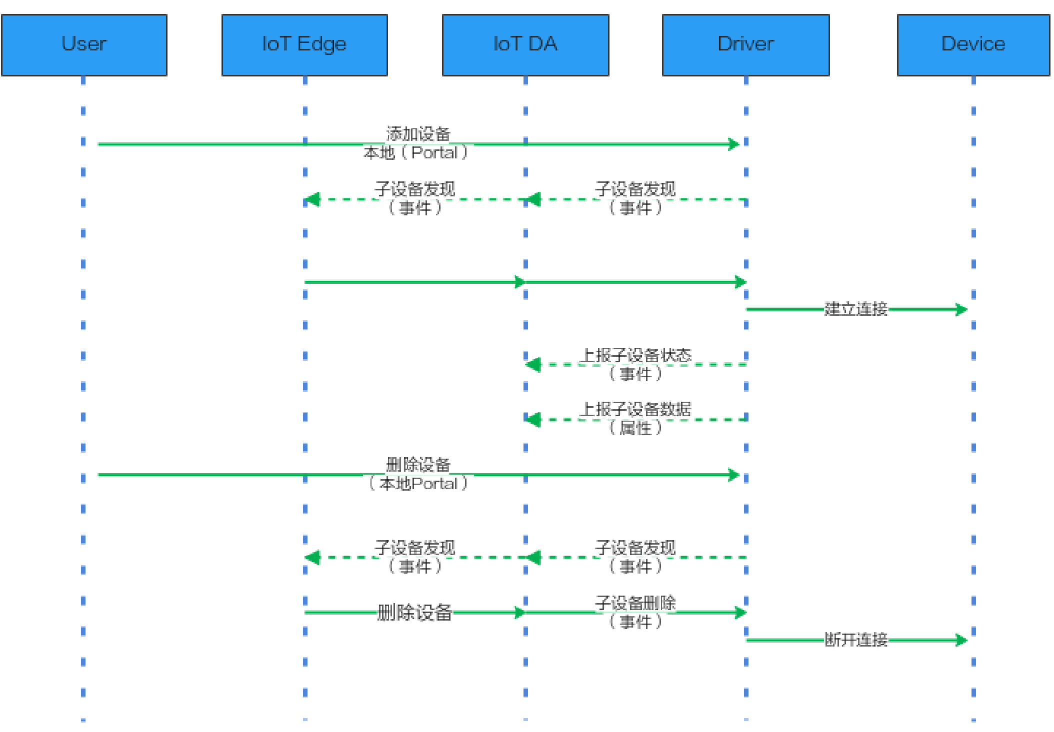
驱动提供子设备管理和数据上报功能。

子设备管理功能包括子设备增删和配置子设备数据采集协议。设备的增删有两种形式，一是在云端增删，通过集成驱动SDK，可以接收到北向应用管理子设备的回调，即北向应用通过IoTEdge创建/删除设备时，自定义驱动应用可以接收到事件回调，在边侧进行设备接入及协议解析。二是在本地增删，自定义驱动应用也提供本地页面进行设备管理，驱动应用通过SDK上报设备添加/删除通知到云端IoTEdge，将边侧的设备管理操作同步到云端。

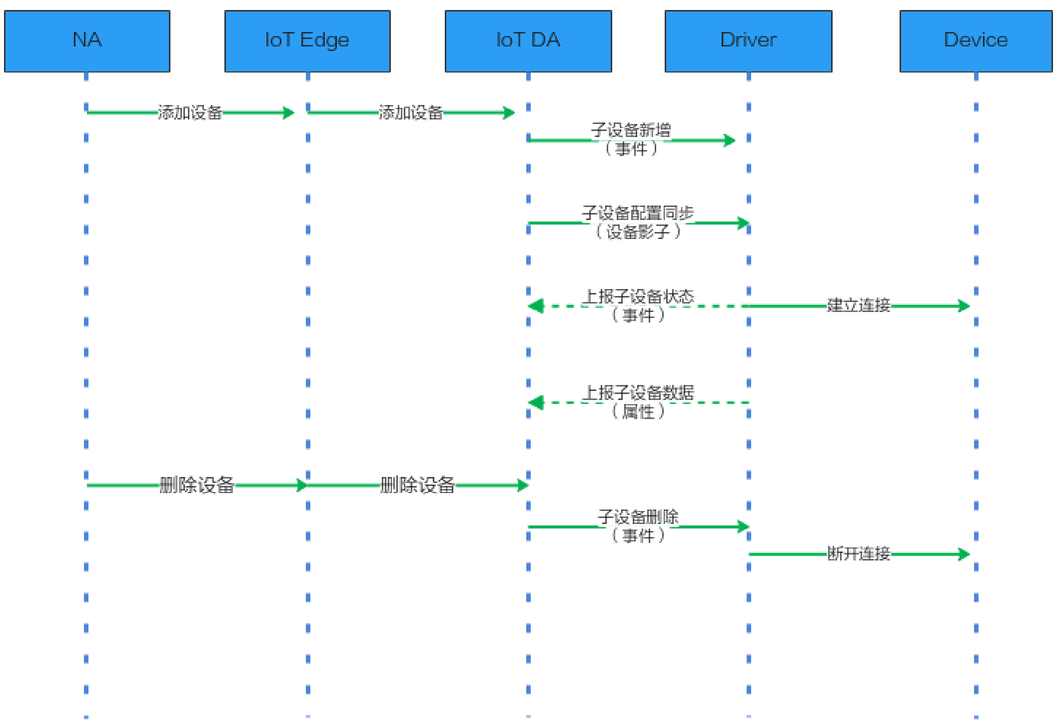
数据上报功能可以将设备状态和设备数据上报到云端。

版权所有 © 华为云计算技术有限公司


云
NA
(SaaS)
子设备发放
子设备配置
IoT DA
IoT Edge
子设备管理时间
子设备配置同步
子设备数据上报
边
子设备管理时间
子设备配置同步
子设备数据上报
Hub
Driver
Device



1. 驱动提供本地Portal对子设备进行管理，包括子设备数据采集协议相关配置；驱动将设备状态和设备数据上报到云端。
2. 本地Portal增删设备时，通过子设备发现事件上报增删的设备，IoT Edge在云侧增删设备，增删的设备信息通过子设备增删事件通知到驱动

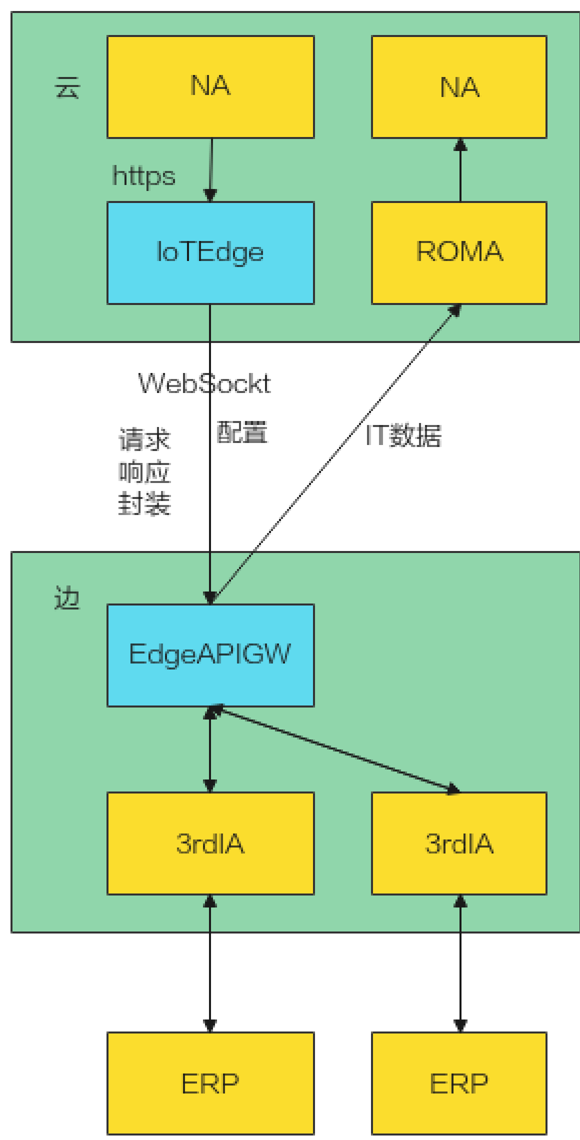


1. 北向应用NA提供设备管理界面，调用IoTEdge接口增删设备，通过子设备增删事件通知到驱动。
2. 北向应用在增加设备时指定设备数据采集协议相关配置。

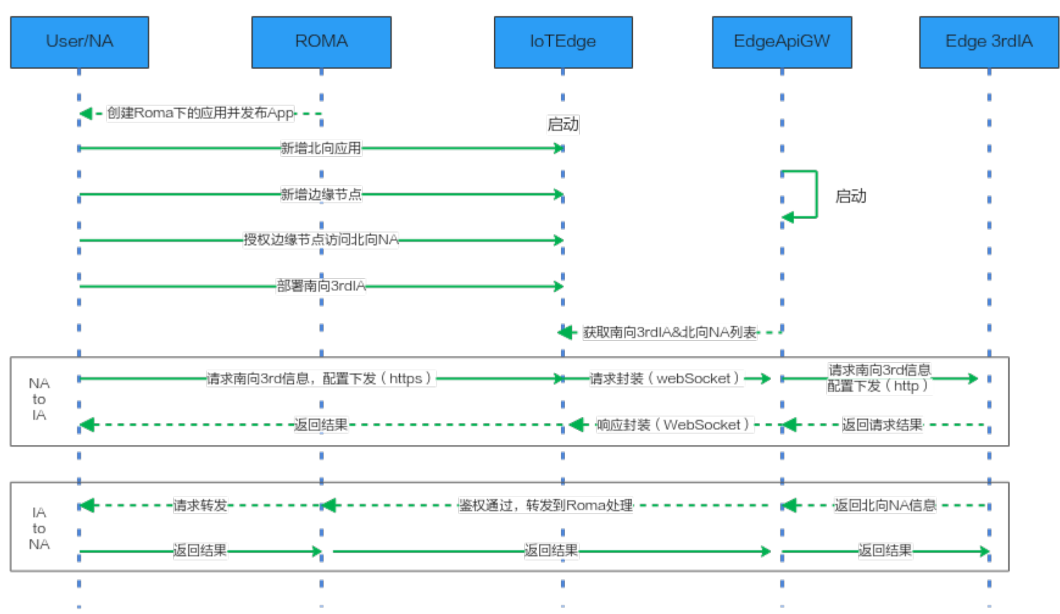
版权所有 © 华为云计算技术有限公司

4.2.3 工业子系统接入(IT 应用)

IT应用利用APIGW与云端协同提供下行通道、上行请求代理（鉴权终结）等能力，实现了云端（NA）与IT应用的相互调用。



版权所有 © 华为云计算技术有限公司



4.3 集成 ModuleSDK(Java)

4.3.1 内部架构

表 4-1 提供的客户端类型

类	说明
AppClient	应用客户端，用于开发设备数据处理应用。
DriverClient	驱动客户端，用于开发驱动接入设备，驱动需实现网关回调接口。
ItClient	IT客户端，用于开发IT集成应用。
DcDriver	OT数采客户端，用于开发点位数采驱动。

4.3.2 开发指导

4.3.2.1 接口函数

4.3.2.1.1 客户端通用方法说明

ModuleSDK中提供多个通用方法供客户开发应用时使用，使用时可参考每个函数的功能定义。

表 4-2 客户端通用方法

接口	说明
open	打开客户端。
startModuleShadow	启动影子，设置收到影子回调并触发获取影子动作。
getModuleShadow	获取影子，触发获取影子动作。
setConnectionStatusChangeCallback	设置连接状态变化回调。

BaseClient::open

打开客户端，客户端打开后才能正常收发消息，该函数适用于所有客户端。

函数描述

```
public void open() throws GeneraException
```

InnerClient::startModuleShadow

启动模块影子，设置收到影子回调并触发获取影子动作，该函数适用于AppClient、DriverClient、DcDriver。

函数描述

```
public void startModuleShadow(ModuleShadowNotificationCallback callback) throws
IllegalArgumentException
```

表 4-3 参数说明

参数名称	类型	参数描述
callback	ModuleShadowNotificationCallback	回调类实例

表 4-4 ModuleShadowNotificationCallback 需要实现的回调接口

回调接口	参数描述
void onModuleShadowReceived(ModuleShadowNotification shadow)	收到模块影子回调，模块影子支持增量同步，根据properties_update_time判断发生改变的属性。

InnerClient::getModuleShadow

获取影子，触发获取影子动作，该函数适用于AppClient、DriverClient、DcDriver。

函数描述

```
private void getModuleShadow()
```


版权所有 © 华为云计算技术有限公司

InnerClient::setConnectionStatusChangeCallback

设置连接状态变化回调，用于MQTT连接状态改变时进行相应的处理，该函数适用于AppClient、DriverClient、DcDriver。

```
public void setConnectionStatusChangeCallback(ConnectionStatusChangeCallback callback)
```

表 4-5 参数说明

参数名称	类型	参数描述
callback	ConnectionStatusChangeCallback	回调类实例

表 4-6 ConnectionStatusChangeCallback 需要实现的回调接口

回调接口	参数描述
`void onConnectionStatusChanged(ConnectionStatus connectionStatus)`	MQTT连接状态变化回调

4.3.2.1.2 AppClient 方法说明

ModuleSDK中，客户端类AppClient提供多个方法供客户开发应用时使用，使用时可参考每个函数的功能定义。

表 4-7 AppClient 方法说明

接口	说明
createFromEnv	根据环境变量创建客户端，相关配置参数从环境变量获取。
setBusMessageCallback	设置消息总线回调，用于对设备上报的数据进行处理。
sendBusMessage	向总线发送设备数据，用于将处理后的设备数据发送到总线。
callDeviceCommand	调用设备命令。
getDevicesStatus	查询设备状态。
getDevicesInfos	查询子设备列表。
getDeviceProperties	获取设备属性。
setCustomizedMessageCallback	设置自定义消息回调，用于对自定义数据进行处理。
sendCustomizedMessage	向总线发送自定义消息。

 版权所有 © 华为云计算技术有限公司

AppClient::createFromEnv

根据环境变量创建客户端，相关配置参数从环境变量获取。

函数描述

```
public static AppClient createFromEnv()throws GeneraException
```

AppClient::setBusMessageCallback

设置总线消息回调，用于对设备上报的数据进行处理。

函数描述

```
public void setBusMessageCallback(String inputName,BusMessageCallback callback)throws
java.lang.IllegalArgumentException
```

表 4-8 参数说明

参数名称	类型	参数描述
inputName	String	模块输入点
callback	BusMessageCallback	回调类实例

表 4-9 BusMessageCallback 需要实现的回调接口

回调接口	参数描述
void onMessageReceived(BusMessage busMessage)	收到来自消息总线的设备上报数据回调

AppClient::sendBusMessage

向总线发送消息，用于将处理后的设备数据发送到总线。

函数描述

```
public void sendBusMessage(String outputName,BusMessage message)throws
java.lang.IllegalArgumentException,JsonException
```

表 4-10 参数说明

参数名称	类型	参数描述
outputName	String	模块输出点
message	BusMessage	设备数据

AppClient::getDevicesStatus

查询设备状态。

版权所有 © 华为云计算技术有限公司

函数描述

```
public List<DeviceStatus> getDevicesStatus(String[] deviceIds,int timeout)throws GeneraException
```

表 4-11 参数说明

参数名称	类型	参数描述
deviceIds	String[]	设备ID数组
timeout	int	等待超时毫秒数

AppClient::getDevicesInfos

查询子设备列表。

函数描述

```
public List<EdgeDeviceInfo> getDevicesInfos()throws GeneraException
```

表 4-12 参数说明

参数名称	类型	参数描述
deviceIds	String[]	设备ID数组。
timeout	int	等待超时毫秒数

AppClient::setCustomizedMessageCallback

设置自定义消息回调，用于从消息总线中接收自定义消息。

函数描述

```
public void setCustomizedMessageCallback(CustomizedMessageCallback callback)
```

表 4-13 参数说明

参数名称	类型	参数描述
callback	CustomizedMessageCallback	回调类实例

表 4-14 CustomizedMessageCallback 需要实现的回调接口

回调接口	参数描述
`void onMessageReceived(BusMessage busMessage)`	收到来自消息总线的设备上报数据回调

 版权所有 © 华为云计算技术有限公司

AppClient::sendCustomizedMessage

向总线发送自定义消息

函数描述

```
public void sendCustomizedMessage(java.lang.String topic,String payload)
```

表 4-15 参数说明

参数名称	类型	参数描述
topic	String	自定义消息的目标topic
payload	String	自定义消息

4.3.2.1.3 DriverClient 方法说明

ModuleSDK中，客户端类DriverClient提供多个方法供客户开发应用时使用，使用时可参考每个函数的功能定义。

表 4-16 DriverClient 方法说明

接口	说明
createFromEnv	根据环境变量创建客户端，相关配置参数从环境变量获取。
setGatewayCallback	设置网关回调，实现对子设备下行请求、网关下行事件的处理。
sendDeviceMessage	发送设备消息。
sendDeviceEvent	发送设备事件。
getDeviceShadow	获取设备影子。
reportSubDevicesProperties	批量上报子设备属性。
reportSubDevicesPropertiesAndDiscovery	上报子设备发现属性。
syncSubDevices	发送子设备同步请求事件。
updateSubDevicesStatus	发送更新子设备状态事件。
getProducts	发送获取产品事件。
reportScanResult	上报子设备扫描结果。

DriverClient::createFromEnv

根据环境变量创建客户端，相关配置参数从环境变量获取。

函数描述

```
public static DriverClient createFromEnv()throws GeneraException
```

DriverClient::setGatewayCallback

设置网关回调，实现对子设备下行请求、网关下行事件的处理。

函数描述

```
public void setGatewayCallback(GatewayCallback gatewayCallback)
```

表 4-17 参数说明

参数名称	类型	参数描述
gatewayCallback	GatewayCallback	网关回调类实例

表 4-18 GatewayCallback 需要实现的回调接口

回调接口	参数描述
void onDeviceMessageReceived(Message message)	收到子设备消息回调
CommandRsp onDeviceCommandCalled(String requestId, Command command)	收到子设备命令回调
IotResult onDevicePropertiesSet(String requestId, PropsSet propsSet)	收到子设备属性设置回调
PropsGetRsp onDevicePropertiesGet(String requestId, PropsGet propsGet)	收到子设备属性获取回调
void onDeviceShadowReceived(String requestId, ShadowGetRsp shadowGetRsp)	收到子设备影子回调
void onDeviceEventReceived(Event event)	收到子设备事件回调
void onSubDevicesAdded(String eventId, AddSubDevicesEvent addSubDevicesEvent)	收到新增子设备事件回调
void onSubDevicesDeleted(String eventId, DeleteSubDevicesEvent deleteSubDevicesEvent)	收到新增子设备事件回调
void onGetProductsResponse(String eventId, GetProductsRspEvent response)	收到获取产品应答回调
void onStartScan(String eventId, StartScanEvent startScanEvent)	收到启动子设备扫描回调

DriverClient::sendDeviceMessage

发送设备消息。

函数描述

```
public void sendDeviceMessage(Message message) throws IllegalArgumentException, JsonException
```


版权所有 © 华为云计算技术有限公司

表 4-19 参数说明

参数名称	类型	参数描述
message	Message	设备消息

DriverClient::sendDeviceEvent

发送设备消息。

函数描述

```
public void sendDeviceEvent(Event event) throws IllegalArgumentException, JsonException
```

表 4-20 参数说明

参数名称	类型	参数描述
event	Event	设备事件

DriverClient::getDeviceShadow

获取设备影子。

函数描述

```
public void getDeviceShadow(String requestId, ShadowGet shadowGet) throws IllegalArgumentException,
JsonException
```

表 4-21 参数说明

参数名称	类型	参数描述
requestId	String	请求ID
shadowGet	ShadowGet	应获取请求

DriverClient::reportSubDevicesProperties

批量上报子设备属性。

函数描述

```
public void reportSubDevicesProperties(SubDevicesPropsReport report)
```

表 4-22 参数说明

参数名称	类型	参数描述
report	SubDevicesPropsReport	子设备属性上报消息体

 版权所有 © 华为云计算技术有限公司

DriverClient::reportSubDevicesPropertiesAndDiscovery

上报子设备发现属性。

函数描述

```
public void reportSubDevicesPropertiesAndDiscovery(SubDevicesDiscoveryPropsReport report)
```

表 4-23 参数说明

参数名称	类型	参数描述
report	SubDevicesDiscoveryPropsReport	子设备发现属性数据消息体

DriverClient::syncSubDevices

发送子设备同步请求事件。

函数描述

```
public void syncSubDevices(String eventId, SyncSubDevicesEvent syncSubDevicesEvent) throws
IllegalArgumentException, JsonException
```

表 4-24 参数说明

参数名称	类型	参数描述
eventId	String	事件ID
syncSubDevicesEvent	SyncSubDevicesEvent	同步子设备请求事件

DriverClient::updateSubDevicesStatus

发送更新子设备状态事件。

函数描述

```
public void updateSubDevicesStatus(String eventId, UpdateSubDevicesStatusEvent statusEvent) throws
JsonException
```

表 4-25 参数说明

参数名称	类型	参数描述
eventId	String	事件ID
statusEvent	UpdateSubDevicesStatusEvent	更新子设备状态事件

DriverClient::getProducts

发送获取产品事件。

 版权所有 © 华为云计算技术有限公司

函数描述

```
public void getProducts(String eventId, GetProductsEvent getProductsEvent) throws
IllegalArgumentException, JsonException
```

表 4-26 参数说明

参数名称	类型	参数描述
eventId	String	事件ID
getProductsEvent	GetProductsEvent	获取产品事件

DriverClient::reportScanResult

上报子设备扫描结果。

函数描述

```
public void reportScanResult(String eventId, ScanResultEvent scanResultEvent) throws JsonException
```

表 4-27 参数说明

参数名称	类型	参数描述
eventId	String	事件ID
scanResultEvent	ScanResultEvent	子设备扫描结果事件

4.3.2.1.4 ItClient 方法说明

ModuleSDK中，客户端类ItClient提供多个方法供客户开发应用时使用，使用时可参考每个函数的功能定义。

表 4-28 ItClient 方法说明

接口	说明
createFromEnv	根据环境变量创建客户端，相关配置参数从环境变量获取。
syncConfigs	从云端同步集成应用的配置。
confirmConfigs	向云端确认已同步的配置。
getJson	发送GET json请求。
postJson	发送POST json请求。
deleteJson	发送DELETE json请求。
putJson	发送PUT json请求。
patchJson	发送PATCH json请求。

 版权所有 © 华为云计算技术有限公司

接口	说明
verifyByDaemon	对来自边缘API GW的请求进行鉴权。
sign	对发送给边缘API GW的请求进行鉴权签名。
getHttpClient	获取httpClient，用于直接使用HttpClient发送请求。

ItClient::createFromEnv

根据环境变量创建客户端，相关配置参数从环境变量获取。

函数描述

```
public static ItClient createFromEnv()throws GeneraException
```

ItClient::syncConfigs

从云端同步集成应用的配置。

函数描述

```
public List<Config> syncConfigs() throws GeneraException
```

ItClient::confirmConfigs

向云端确认已同步的配置。

函数描述

```
public void confirmConfigs(List<Config> configs) throws GeneraException
```

表 4-29 参数说明

参数名称	类型	参数描述
configs	List<Config>	配置项列表

ItClient::getJson

发送GET json请求。

函数描述

```
public String getJson(String uri, Header[] headers) throws HttpException, CryptException, AuthException
```

表 4-30 参数说明

参数名称	类型	参数描述
uri	String	请求uri

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述
headers	Header[]	请求头部

ItClient::postJson

发送POST json请求。

函数描述

```
public String postJson(String uri, String body, Header[] headers) throws HttpException, CryptException,
AuthException
```

表 4-31 参数说明

参数名称	类型	参数描述
uri	String	请求uri
body	String	请求消息体
headers	Header[]	请求头部

ItClient::deleteJson

发送DELETE json请求。

函数描述

```
public String deleteJson(String uri, Header[] headers) throws HttpException, CryptException, AuthException
```

表 4-32 参数说明

参数名称	类型	参数描述
uri	String	请求uri
headers	Header[]	请求头部

ItClient::putJson

发送PUT json请求。

函数描述

```
public String putJson(String uri, String body) throws HttpException, CryptException, AuthException
```


版权所有 © 华为云计算技术有限公司

表 4-33 参数说明

参数名称	类型	参数描述
uri	String	请求uri
body	String	请求消息体
headers	Header[]	请求头部

ItClient::patchJson

发送PATCH json请求。

函数描述

```
public String patchJson(String uri, String body) throws HttpException, CryptException, AuthException
```

表 4-34 参数说明

参数名称	类型	参数描述
uri	String	请求uri
body	String	请求消息体
headers	Header[]	请求头部

ItClient::verifyByDaemon

对来自边缘API GW的请求进行鉴权。

函数描述

```
public void verifyByDaemon(String authorization) throws AuthException
```

表 4-35 参数说明

参数名称	类型	参数描述
authorization	String	鉴权token

ItClient::sign

对发送给边缘API GW的请求进行鉴权签名。

函数描述

```
public void sign(HttpRequestBase request) throws CryptException, AuthException
```

 版权所有 © 华为云计算技术有限公司

表 4-36 参数说明

参数名称	类型	参数描述
request	HttpRequestBase	http请求

ItClient::getHttpClient

获取httpClient，用于直接使用HttpClient发送请求。

函数描述

```
public HttpClient getHttpClient()
```

4.3.2.1.5 DcClient 方法说明

ModuleSDK中，客户端类DcClient提供多个方法供客户开发应用时使用，使用时可参考每个函数的功能定义。

表 4-37 DcClient 方法说明

接口	说明
createFromEnv	根据环境变量创建客户端，相关配置参数从环境变量获取。
setPointsCallback	设置点位相关处理回调方法。
pointReport	点位上报。
notifyDsConnectionState	上报连接状态到云端。

DcClient::createFromEnv

根据环境变量创建客户端，相关配置参数从环境变量获取。

函数描述

```
public static DcClient createFromEnv()throws GeneraException
```

DcClient::setPointsCallback

设置点位相关处理方法回调，实现下行点位获取、点位设置的处理。

函数描述

```
public void setPointsCallback(PointsCallback pointsCallback)
```

表 4-38 参数说明

参数名称	类型	参数描述
pointsCallback	PointsCallback	点位回调类实例

 版权所有 © 华为云计算技术有限公司

表 4-39 PointsCallback 回调接口说明

回调接口	参数描述
PointsSetRsp onPointSet(String requestId, PointsSetReq pointsSetReq)	点位设置回调
PointsGetRsp onPointGet(String requestId, PointsGetReq pointsGetReq)	点位获取回调

DcClient::pointReport

上报数采点位信息。

函数描述

```
public void pointReport(PointsReport pointsReport) throws JsonException, TransportException
```

表 4-40 参数说明

参数名称	类型	参数描述
pointsReport	PointsReport	点位信息

DcClient::notifyDsConnectionState

上报连接状态到云端。

函数描述

```
public void notifyDsConnectionState(DsConnectionState dsConnectionState)
```

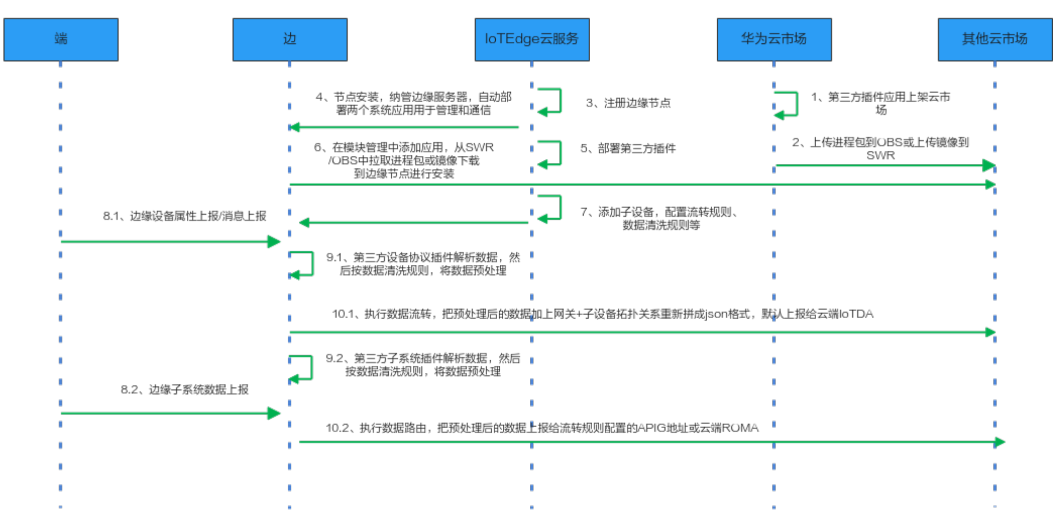
表 4-41 参数说明

参数名称	类型	参数描述
dsConnectionState	DsConnectionState	数据源连接状态

4.3.2.2 方案概述

基于ModuleSDK开发应用实现数据处理或自定义驱动时，分为开发和使用两个部分。

开发操作	开发说明
应用的开发	利用客户端（AppClient、DriverClient、ItClient、DcDriver）进行自定义的业务处理
应用的使用	将应用打包上传至云，部署到节点。连接子设备查看应用工作。



4.3.2.3 前提条件

使用本地 IDE 进行代码开发：

- 开发环境要求：已经安装JDK（版本jdk1.8以上，访问**Java官网**）和maven（**下载**并**安装 Maven**）。
- 开发工具：**IntelliJ IDEA** 或者**eclipse**。
- 下载Java版**Module SDK**。

使用 Visual Studio Code 安装 java SDK 插件进行代码开发：

- 开发环境要求：已经安装JDK（版本jdk1.8以上，访问**Java官网**）和maven（**下载**并**安装 Maven**）。
- 开发工具：Visual Studio Code。

4.3.2.4 创建工程

以下两种方式选其一：

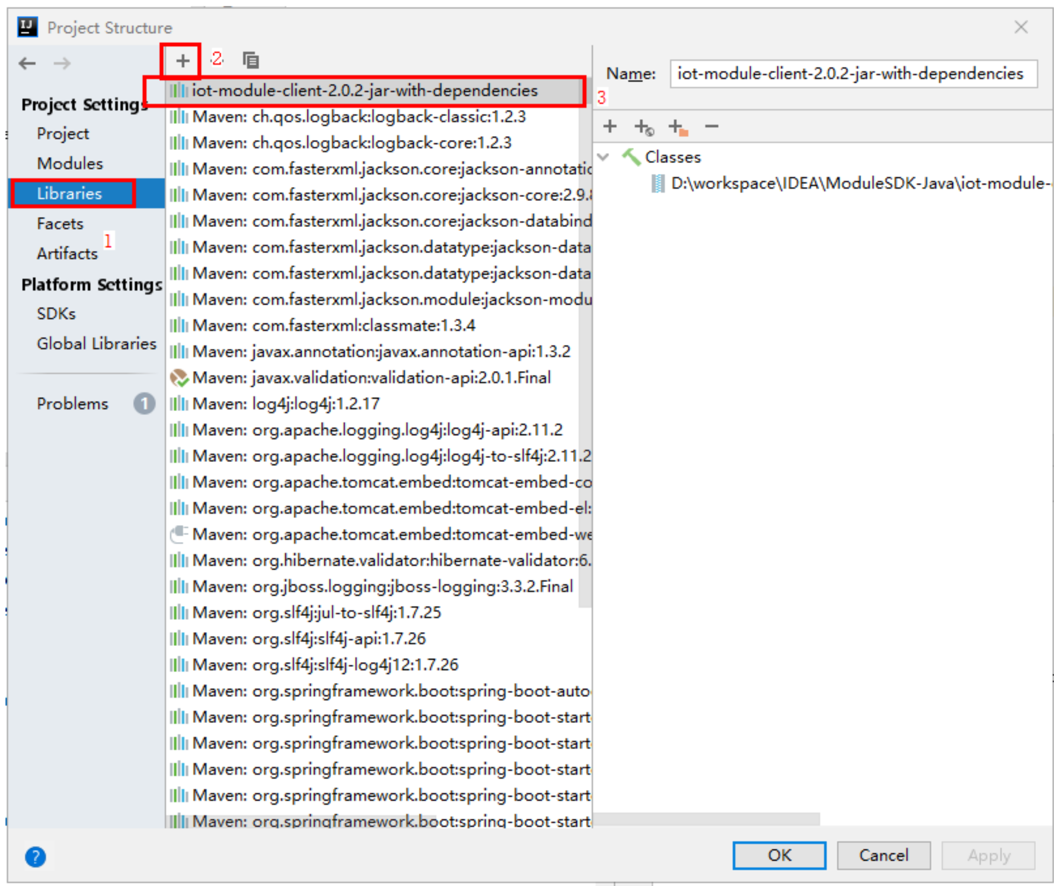
使用本地 IDE 进行开发

步骤1 打开IDEA，选择“file > new > New Project”，选择新建maven工程。

步骤2 导入相关maven依赖。

步骤3 导入ModuleSDK依赖包。右键单击“Open Project Structrure”，选择“Libraries”后，单击中间框上方“+”，选择下载好的ModuleSDK，添加到工程模块。

版权所有 © 华为云计算技术有限公司



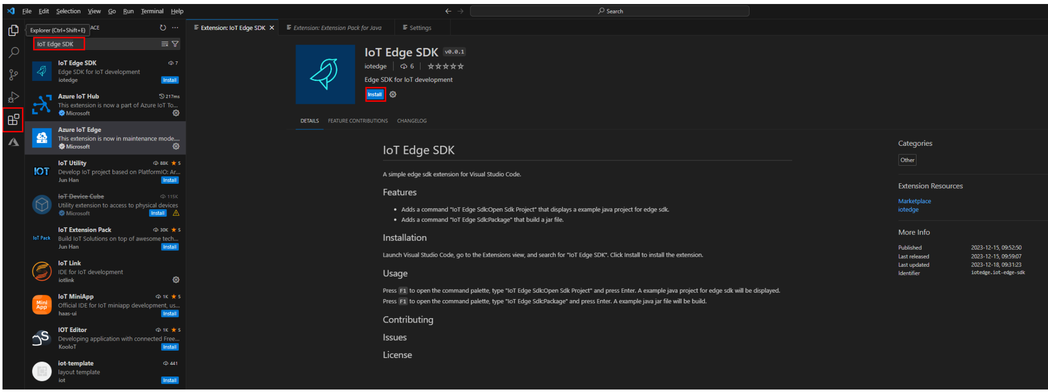
----结束

使用 Visual Studio Code 进行开发

使用Visual Studio Code进行开发，已经完成项目打包，跳过项目打包。

步骤1 打开Visual Studio Code，单击进入插件应用商店页签，搜索IoT Edge SDK，找到后单击“install”。

图 4-1 安装 IoT Edge SDK



步骤2 菜单栏中选择“view > Command Palette...”后，弹出命令窗口，输入如下命令打开SDK的示例工程。

版权所有 © 华为云计算技术有限公司

```
IoT Edge Sdk:Open Sdk Project
```

图 4-2 输入命令

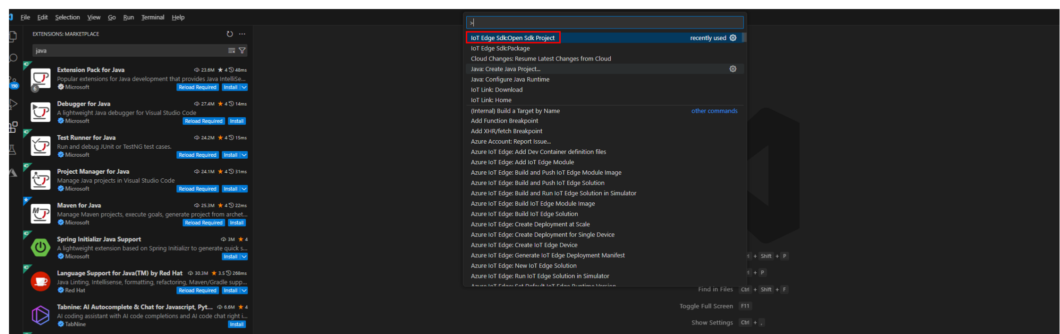
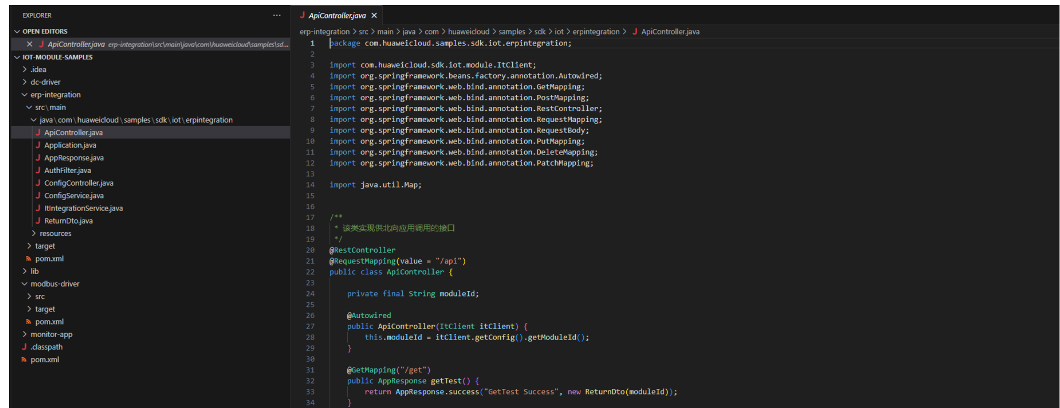
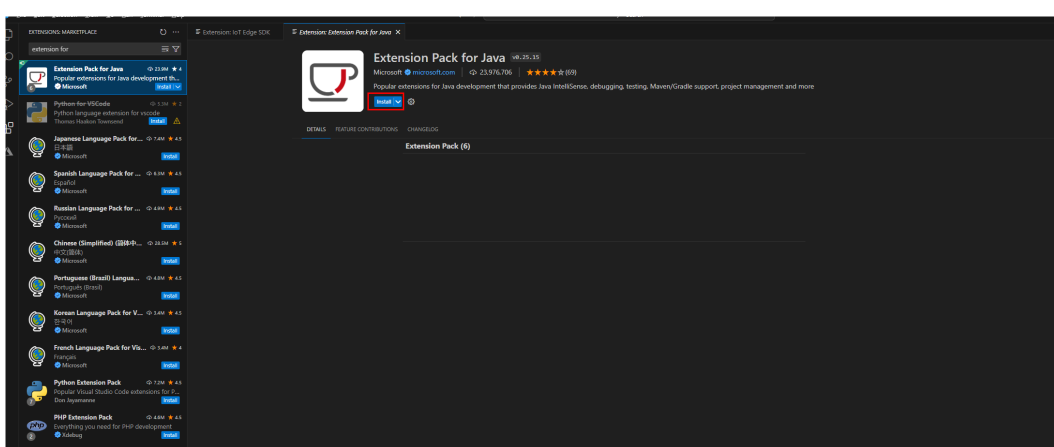


图 4-3 SDK 示例工程页面



步骤3 如果需要调试和运行代码，参考步骤**步骤1**，安装“Extension Pack for Java”” 插件。

图 4-4 安装 Extension Pack for Java



步骤4 参考步骤**步骤2**，输入如下命令，mvn自动构建各个模块生成jar包。

```
IoT Edge Sdk:Package
```


版权所有 © 华为云计算技术有限公司

图 4-5 构建 jar 包

步骤5 如下图，构建成功后jar包会显示在如下目录。

图 4-6 jar 包所在目录

----结束

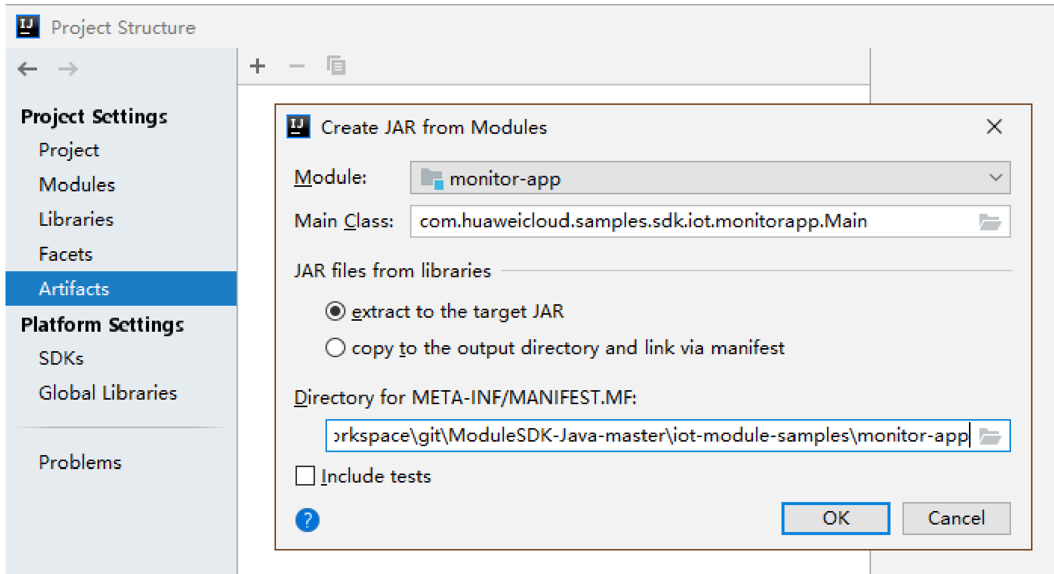
4.3.2.5 项目打包

根据您的需求进行相关代码的开发，并将项目打包，以编辑器IDEA为例：

步骤1 选中项目->右键open Module setting

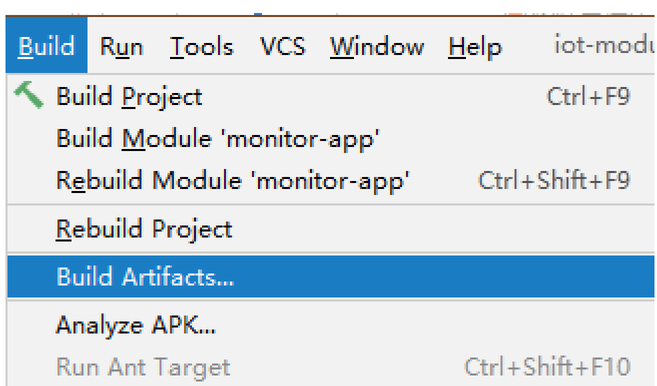
步骤2 Artifacts->单击+号>JAR->From modules with dependencies->模块选择monitor-app,选择monitorapp的Main入口类,注意MANIFEST.MF位置选择模块根目录->单击apply。

版权所有 © 华为云计算技术有限公司



步骤3 单击上方build选项->选择build Artifacts->monitor-app:jar->build

步骤4 打包完成得到monitor-app.jar文件。(如遇到错误(Invalid signature file digest for Manifest main attributes)请使用压缩文件进入metf目录删除.rsa和.sf文件)



----结束

4.3.2.6 制作镜像包或插件包

将jar包转换为镜像包或插件包。

注意

若制作镜像包以容器化方式部署应用，不支持在一个容器内运行多个集成ModuleSDK的软件进程或者重启集成ModuleSDK的软件进程，会导致鉴权失败等问题。

镜像包打包

步骤1 上传打包的项目。

将jar文件上传到联网的linux机器上，如目录（/home/monitor）中

步骤2 安装docker。

请确认使用的系统已经安装docker(docker版本需要高于17.06，推荐18.06),安装参照**docker 安装教程**

步骤3 制作镜像。

1. 搜索基础镜像，基础镜像需要集成jre。

   docker search jre8

2. 选择合适的镜像（镜像需要集成了jre且版本不低于8）

```
NAME              DESCRIPTION   STARS     OFFICIAL   AUTOMATED
livingobjects/jre8   Jre8 image     4          [OK]
```

   说明

   livingobjects/jre8镜像是docker hub第三方提供的镜像，非IoT团队发布，且IoT团队未提供任何官方镜像。该镜像在此仅做示例，IoT团队对该镜像的安全性不作保证。强烈建议用户自己封装镜像！

3. 拉取镜像

   docker pull livingobjects/jre8

4. 编写dockerfile制作镜像

   dockerfile内容参照如下（具体可参考**编写高效的Dockerfile** ）

```
#Version 1.0.0
FROM livingobjects/jre8  #基础镜像来源
RUN mkdir -p /opt/iot/edge/monitor / && chmod -R 777 /opt/ #授权
COPY monitor /opt/iot/edge/monitor   #复制文件到指定目录,此dockerfile文件位置:/home,jar包位置:/
home/monitor
USER root      #用户名
EXPOSE 8080
CMD ["java", "-jar", "/opt/iot/edge/monitor/monitor-app.jar", "run"] #运行命令
```

5. 构建镜像

   docker build -t edge_monitor:1.0.0 /home --no-cache

6. 查看打包完成的镜像

   docker images

```
REPOSITORY      TAG       IMAGE ID        CREATED            SIZE
edge_monitor    1.0.0    93f9d964bcea   12 seconds ago   243MB
```

步骤4 镜像上传。

1. 上传镜像

   开通容器镜像服务SWR

   镜像上传需要使用SWR镜像容器服务，开通及使用请参照**容器镜像服务 SWR_用户指南**

2. 获取SWR登录指令

   请参照**使用容器引擎客户端上传镜像**第四节。

3. tag镜像

   使用tag命令将镜像重命名以确定上传位置和组织，获取上传位置和组织前缀请参考**pull/push命令**

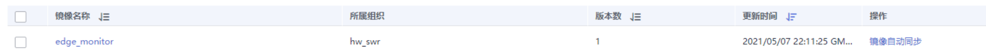
   docker tag edge_monitor:1.0.0 swr.cn-north-4.myhuaweicloud.com/hw_swr/edge_monitor:1.0.0

4. 上传镜像

   docker push swr.cn-north-4.myhuaweicloud.com/hw_swr/edge_monitor:1.0.0

版权所有 © 华为云计算技术有限公司

5. 查看上传结果



6. 上传镜像后请在SWR将镜像设置为公开。

----结束

插件包打包

步骤1 插件包制作。

1. 插件包格式要求如下：

   插件包仅支持.tar.gz 、.tar或者 .zip格式。

   插件包结构如下：

   app.zip
   ├── ****.jar //可执行jar文件,必须
   ├── start.sh //启动脚本 必须文件 当前不提供参数方式启动
   └── stop.sh //停止脚本 非必须

2. 构建插件包。

   以monitor-app为例，在项目打包后得到monitor-app.jar

   在monitor-app.jar文件的同目录下创建start.sh,内容如下:

```
function log(){
    echo "`date "+%Y-%m-%d %T"`: $1"
}
log "[INFO] start execut process."
#调试时可打开，确认sdk需要的环境能被获取
#echo "${device_id}" > test_enviroment.file
pwd
#更新环境变量，防止找不到java命令。
source /etc/profile
#运行文件在/var/IoTEdge/downloaded-job/run下面
java -jar ./monitor-app.jar  > monitor_running.log  2>&1
```

   将monitor-app.jar和start.sh一起压缩得到monitor-app.zip。

步骤2 插件包上传。

1. 开通对象存储服务OBS。

   进程包上传方式需要开通对象存储服务OBS，请参考**对象存储服务 OBS_快速入门**

2. 上传进程包。

   上传方式请参照**对象存储服务(OBS)**。

> **注意**
>
> 请设置桶策略为【公开读】，如未设置请前往OBS首页->单击桶ID->访问权限控制->桶策略中设置。

----结束

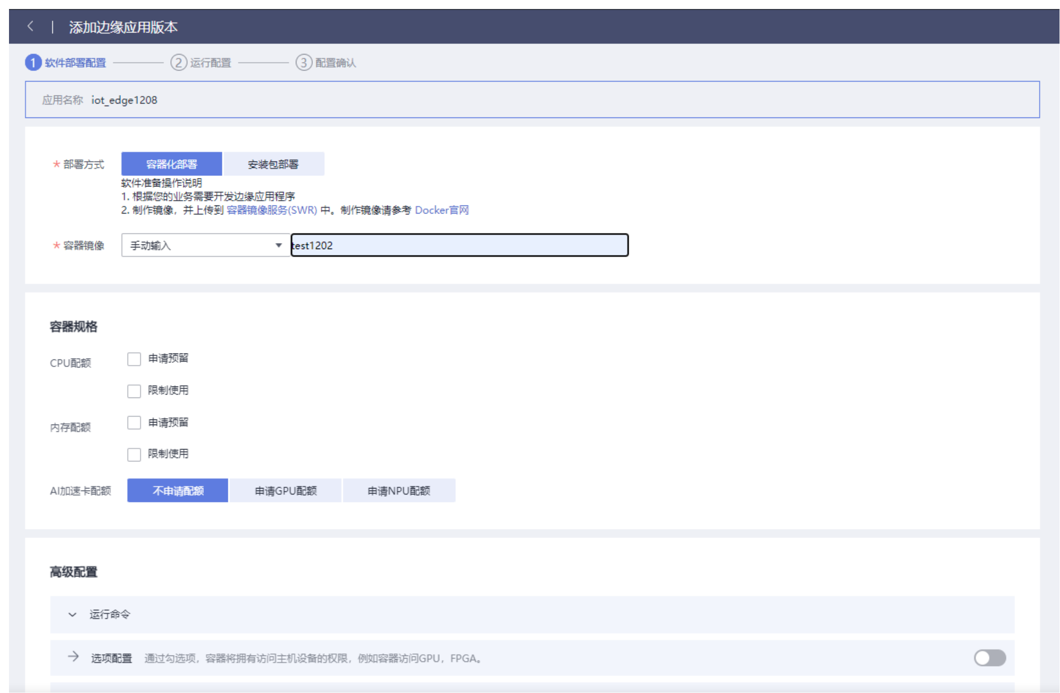
4.3.2.7 添加应用

添加边缘应用具体请参考**添加应用**。

容器化部署

步骤1 软件部署配置，部署方式选择“容器化部署”。

选择上传到SWR服务的镜像，如未发现镜像，请检查镜像是否为公开，设置镜像为公开方式：容器镜像服务SWR->我的镜像->单击镜像ID进入详情->右上方编辑。



步骤2 软件和运行配置。运行配置

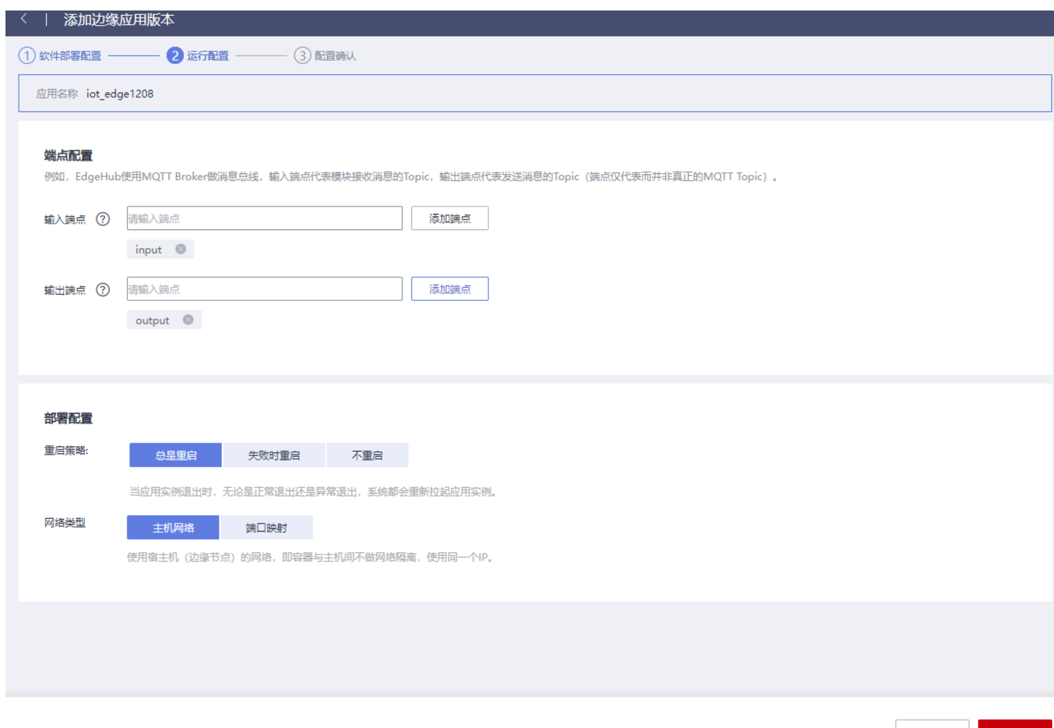
根据需要进行配置。

输入端点输出端点与demo中代码定义的端点对应，如monitor-app中输入与输出端点设置为input和output,则配置为：

输入端点：input

输出端点：output

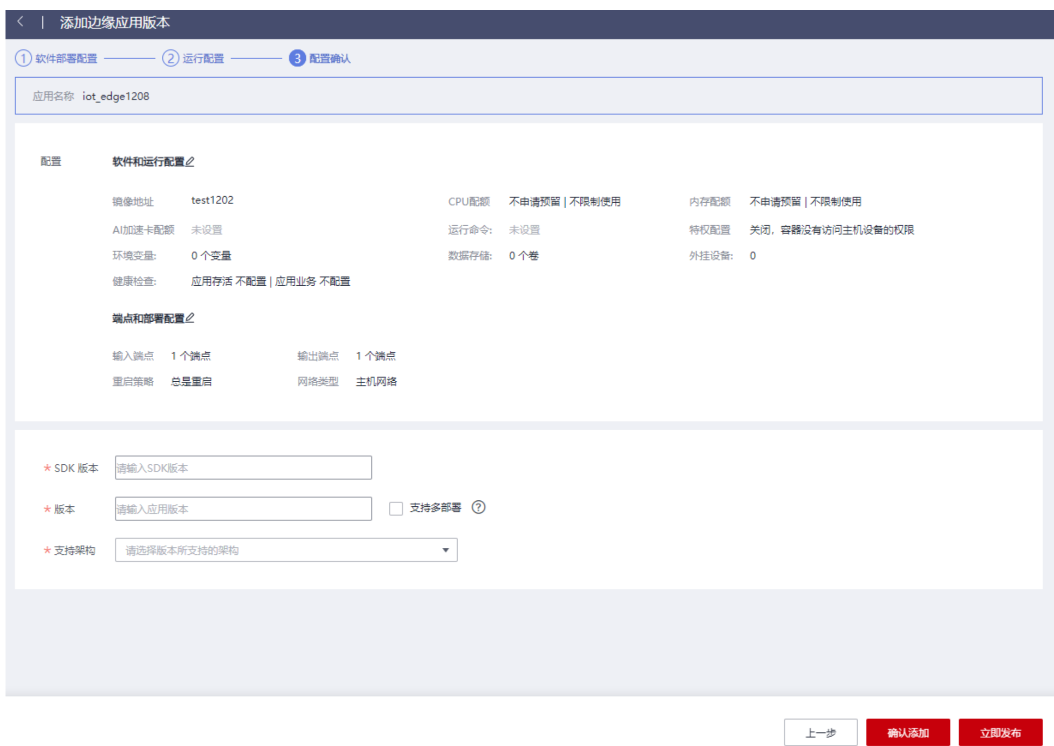
版权所有 © 华为云计算技术有限公司



📖 **说明**

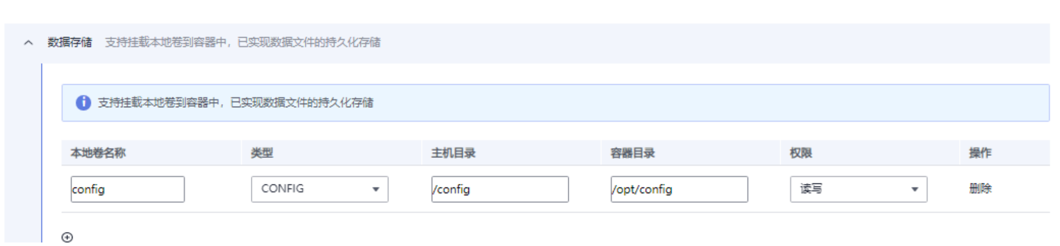
输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

步骤3 确认配置，填写基本信息。



⚠ **注意**

集成ModuleSDK后，在高级配置中必须挂载config卷，主机目录和容器目录均为自定义，可参考下图。模块身份等信息均由SDK存储在config卷中，如果不配置挂载config卷，会导致自定义应用重启后无法与EdgeHub成功建立Mqtt连接。



----结束

安装包部署

步骤1 添加边缘应用-应用配置

部署方式选择安装包部署

步骤2 添加边缘应用-软件和运行配置

安装包地址为{桶名/对象名}。

如桶名为edge-monitor,对象名为monitor-app.zip， 则安装包为edge-monitor/monitor-app.zip。

版权所有 © 华为云计算技术有限公司

步骤3 添加边缘应用-端点和部署配置同容器化部署

根据需要进行配置。

输入端点输出端点与demo中的设置对应，如monitor-app中输入与输出端点设置为input和output,则配置为：

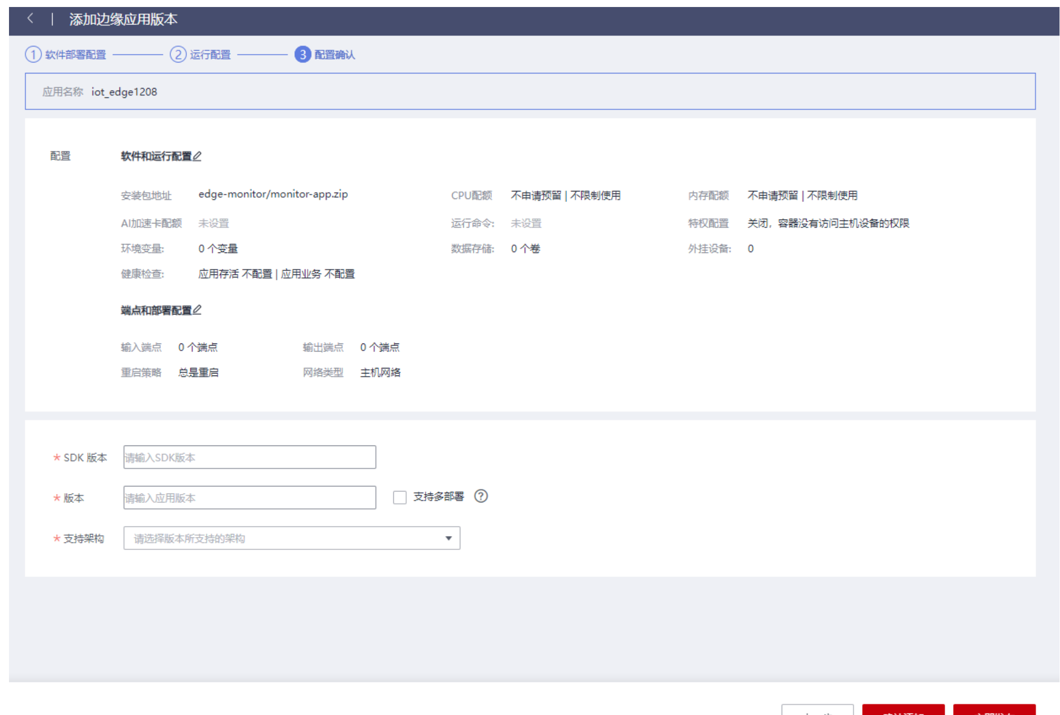
输入端点：input。

输出端点：output。

📖 **说明**

输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

步骤4 配置确认,填写基本信息



步骤5 单击“确认添加”，完成应用的创建；单击"立即发布"，完成应用创建并发布新版本

创建应用管理成功!

您已创建应用管理djdj，和版本1

返回应用管理列表

后续操作说明

- 接下来您可以在边缘节点部署该版本应用
- 已发布版本不允许删除和修改
- 如需调整版本发布状态，请在应用的版本列表中操作

----结束

4.3.2.8 发布应用

应用创建之后需要发布才允许在节点部署。

操作步骤

步骤1 访问**IoT边缘**，单击“**管理控制台**”进入IoT边缘控制台。

步骤2 选择左侧导航栏“边缘节点 > 应用管理”进入页面，选择“应用名称”进入应用详情页。

步骤3 单击右上角“发布”按钮。

版权所有 © 华为云计算技术有限公司

----结束

> **说明**
>
> 可在创建应用时可勾选【立即发布】进行发布。

4.3.2.9 如何使用

OT应用使用步骤：

1. 将创建好的应用部署到节点。部署请参照**应用部署**。
2. 添加设备进行测试（添加设备请参考**设备接入边缘节点**）。

> **说明**
>
> 可以利用MQTT.fx软件模拟设备接入调试。

驱动应用使用步骤：

1. 将创建好的应用部署到节点。部署请参照**应用部署**。
2. 添加网关。
3. 添加设备进行测试（添加设备请参考**设备接入边缘节点**）。

> **说明**
>
> 可以利用Modbus Slave软件模拟设备接入调试。

IT应用使用步骤：

1.注册节点，绑定工业资源包。

2.创建API网关。

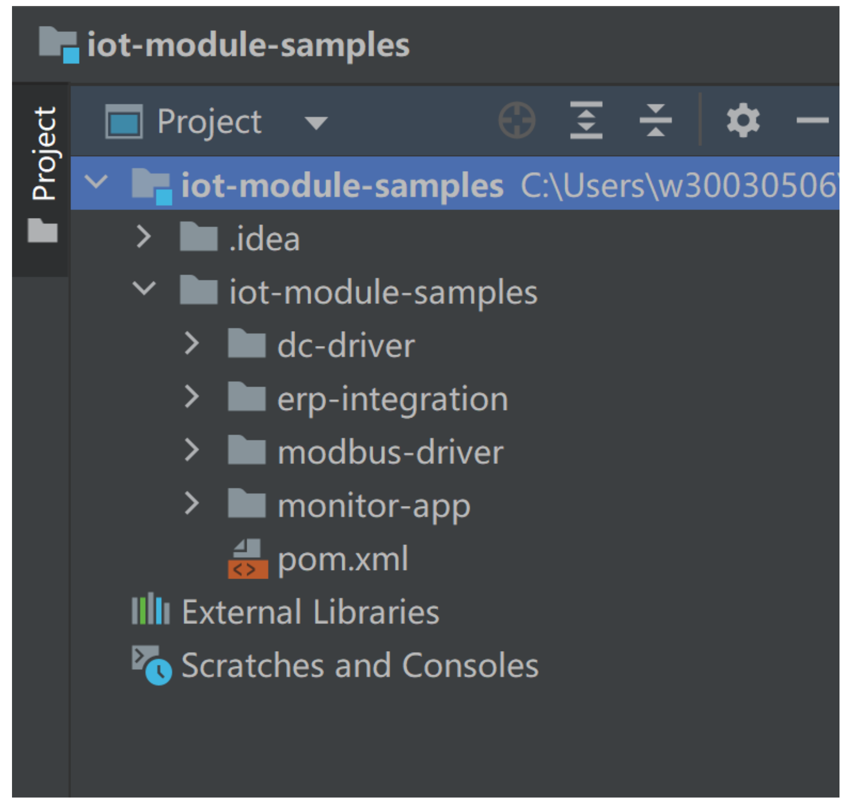
3.将apigw系统组件和创建的应用部署到节点。

4.使用postman进行调试验证。

4.3.3 下载 demo

下载demo，解压并导入示例工程。

版权所有 © 华为云计算技术有限公司



- dc-driver: 此模块主要演示ot数采集成。
- erp-integration：此模块主要演示It集成服务。
- modbus-driver：此模块主要演示协议转换。
- monitor-app：此模块主要演示数据处理。

4.3.4 集成 ModuleSDK 进行数据处理

4.3.4.1 操作场景

开发应用集成ModuleSDK进行数据处理。

在节点接入一个电机设备，设备遇到问题上报信息“error”给节点，节点监听到设备的“error”信息，下发命令让设备进行重启。

示例工程为monitor-app。

4.3.4.2 代码解析

代码解释使用ModuleSDK开发应用集成ModuleSDK进行数据处理。

AppClient类有以下几个关键方法（具体参考JavaDoc）。

- createFromEnv(): AppClient创建时由此方法自动获取环境变量。
- setBusMessageCallback(): 设置总线消息回调，用于对设备上报的数据进行处理
- sendBusMessage(): 向总线发送消息，用于将处理后的设备数据发送到总线
- callDeviceCommand(): 调用设备命令

版权所有 © 华为云计算技术有限公司

- getDevicesInfo(): 查询设备状态

MonitorApp代码解析

片段一

```
privatestaticfinalString INPUT = "input";
public static final String OUTPUT = "output";
```

定义输入和输出的端点，关于取值需要需在创建应用版本的inputs参数中定义,创建应用时输入端点与输出端点以及数据流转规则的配置与此是对应的。例如此处定义了输入端点为“input”，输出端点为“output”，则创建应用时的端点和软件配置输入端点需要配置为input，输出端点需要配置为output。

应用部署后还需要设置数据流转规则后，决定数据的流向。

/** * 电机设备的产品ID */ public static final String *MOTOR_PRODUCT_ID* = "60988d94aa3bcc02c0200667";

单击设备的产品ID,需要在IoTDA设备接入创建产品时获取。

片段二

```
private AppClient appClient;
publicMonitorApp() throws GeneraException {
appClient = AppClient.createFromEnv();
}
```

定义并创建AppClient， AppClient.createFromEnv()创建其配置参数将会自动从边缘节点环境中获取，数据的传输将会依赖AppClient。

片段三

```
publicvoidstart() throws GeneraException {
//设置回调，打开客户端
appClient.setBusMessageCallback(INPUT, this);//设置收到设备数据的回调
appClient.open();
}
```

appClient在接收到数据后的处理动作需要用户定义，具体操作是设置回调。 这里设置回调方法并传入输入端点后，appClient将会开启并启动一个监听器监听输入端点的数据传输，接收到设备经过hub发来的数据后会调用回调进行数据处理。

片段四

```
public void onMessageReceived(BusMessage busMessage) {
try {
    if (busMessage.getProductId().equals(MOTOR_PRODUCT_ID)) {
      //马达设备状态错误时对马达进行重启
       MotorData motorData = JsonUtil.fromJson(
       JsonUtil.toJson(busMessage.getServices().get(0).getProperties()), MotorData.class);
           if (motorData.getStatus().equals("error")) {
               Command command = new Command(busMessage.getDeviceId(),"power", "power_control",
"restart"
               );
       appClient.callDeviceCommand(command, FIVE_SECOND);
      }
    } else {
     //其他设备数据发布到总线
      appClient.sendBusMessage(OUTPUT, busMessage);
}
} catch (Exception e) {
System.out.println(e.getMessage());
}
}
```


版权所有 © 华为云计算技术有限公司

onMessageReceived(BusMessage busMessage)是前面设置回调具体的回调函数，它是BusMessageCallback接口的方法，MonitorApp要实现BusMessageCallback接口并实现此方法，AppClient通过设置的输入端口input监听到设备发送数据时调用此方法进行数据处理，用户关于设备发送的数据的处理逻辑在此方法内实现，处理后的数据通过设置的输出端口output发送经过hub发送到云端。

appClient.callDeviceCommand(command, FIVE_SECOND)是示例演示应用在接收到设备传来的error信号后，向设备发送秒后重启命令。

4.3.4.3 注册节点

注册节点请参照**注册边缘节点**。

4.3.4.4 创建产品

创建产品具体教程参照**创建产品_设备接入 IoTD** ,以下是具体配置中的参照。

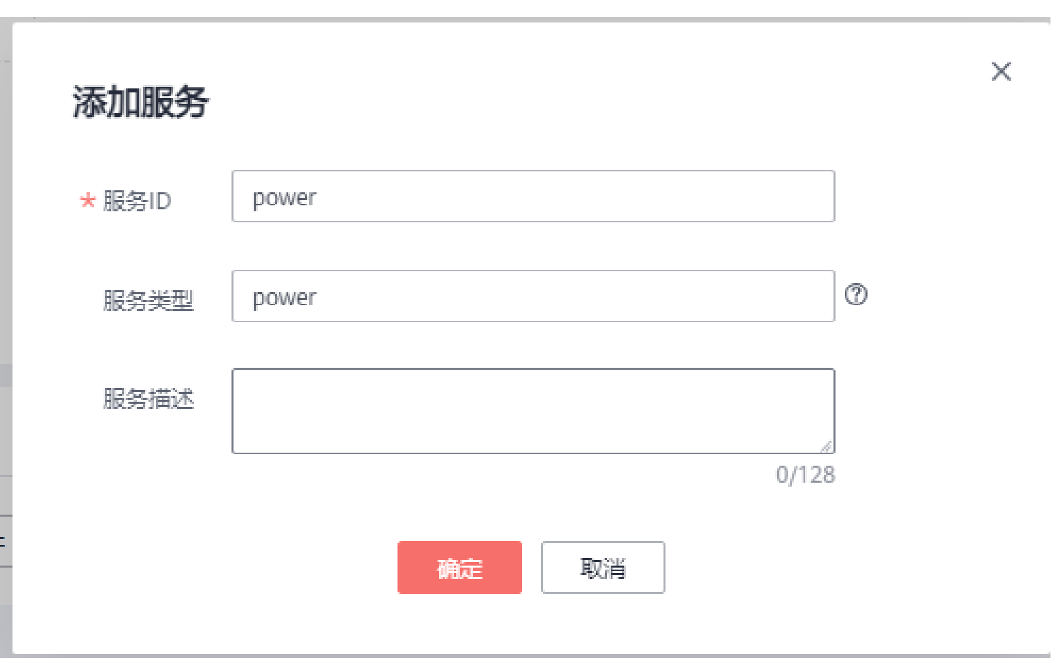
步骤1 创建产品

说明

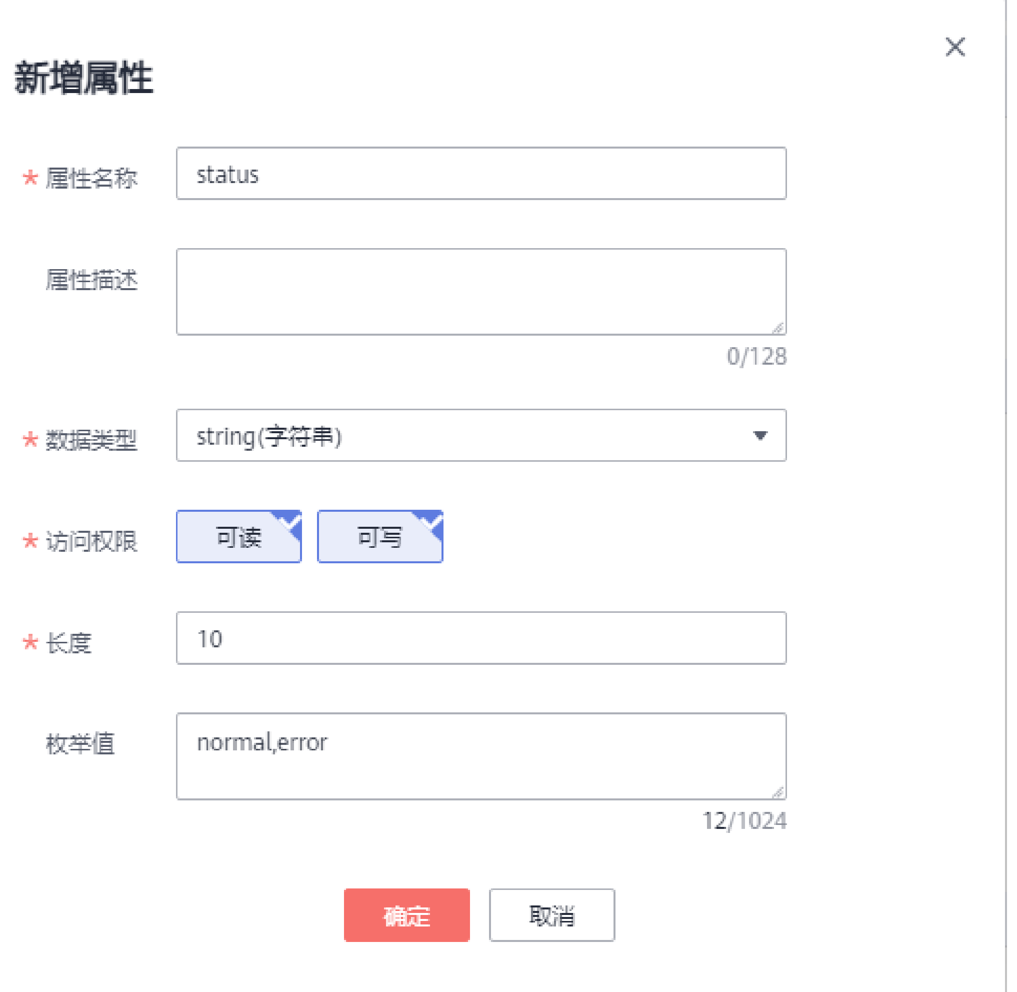
将**修改代码**里代码的产品ID复制到高级设置>定制ProjectID，自定义ID后可省去修改代码步骤。

步骤2 在新建产品后需要在产品页的模型定义中添加服务。

 版权所有 © 华为云计算技术有限公司



步骤3 新增属性



步骤4 新增命令。

步骤5 单击确定完成创建。

----结束

4.3.4.5 修改代码

查看所创建产品的id，查看方式：IoTDA->产品列表。

根据id修改代码。

/** * 电机设备的产品ID */ public static final String *MOTOR_PRODUCT_ID* = "60988d94aa3bcc02c0200667";

4.3.4.6 项目打包

打包参考**项目打包**

将monitor-app进行打包得到monitor-app.jar。

4.3.4.7 制作镜像包

将jar文件打包成镜像文件上，请参照**制作镜像包或插件包**。

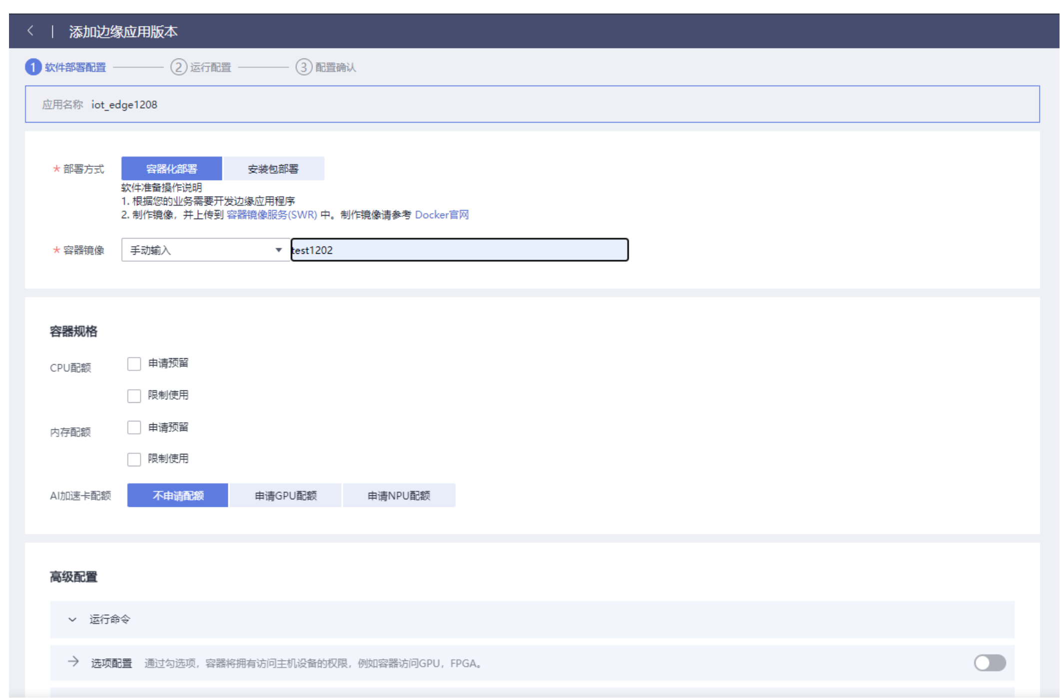
4.3.4.8 创建应用

以容器镜像方式为例，镜像包上传到容器镜像服务SWR后，创建应用。

容器化部署

步骤1 软件部署配置，部署方式选择“容器化部署”。

选择上传到SWR服务的镜像，如未发现镜像，请检查镜像是否为公开，设置镜像为公开方式：容器镜像服务SWR->我的镜像->单击镜像ID进入详情->右上方编辑。



步骤2 运行配置

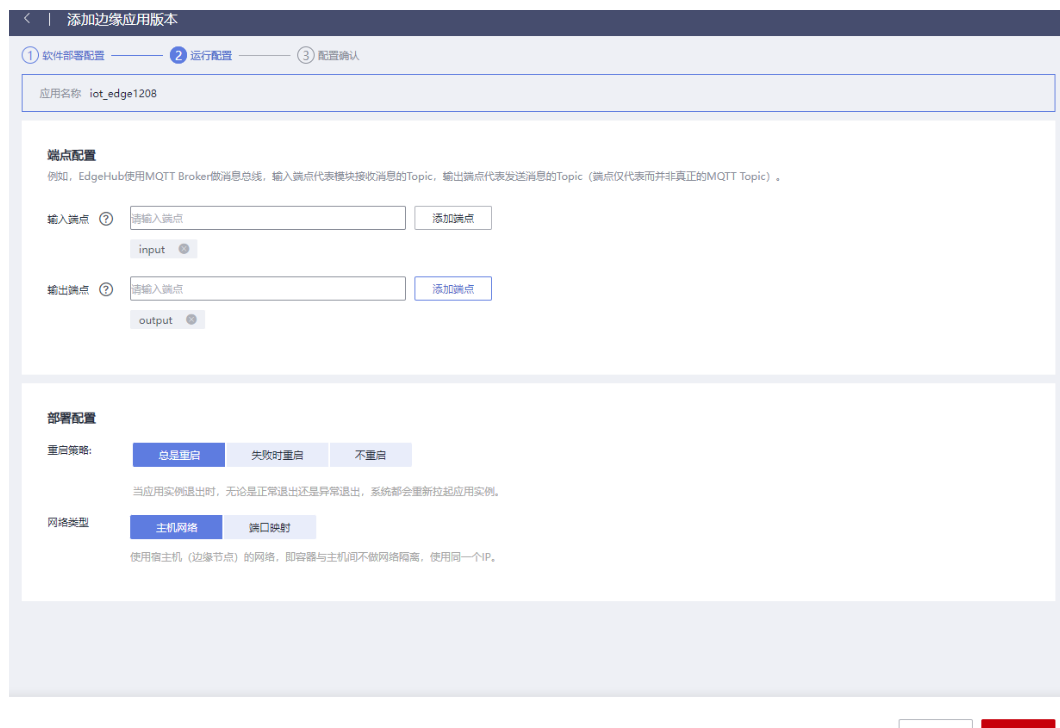
根据需要进行配置。

输入端点输出端点与demo中代码定义的端点对应，如monitor-app中输入与输出端点设置为input和output,则配置为：

输入端点：input

输出端点：output

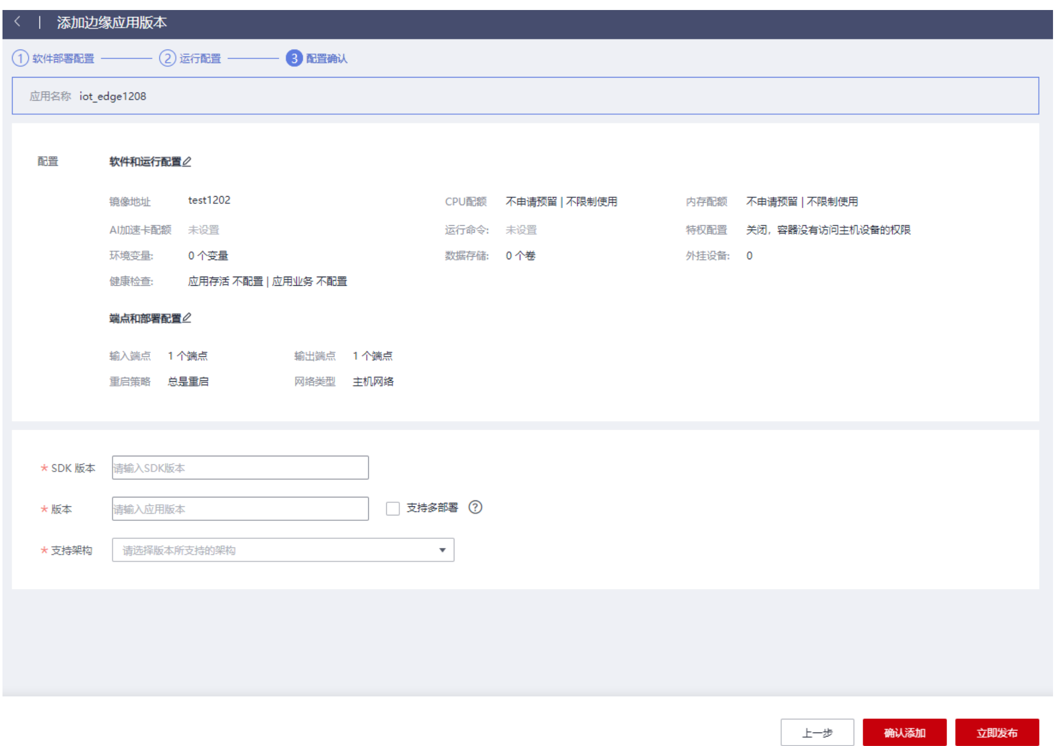
版权所有 © 华为云计算技术有限公司



说明

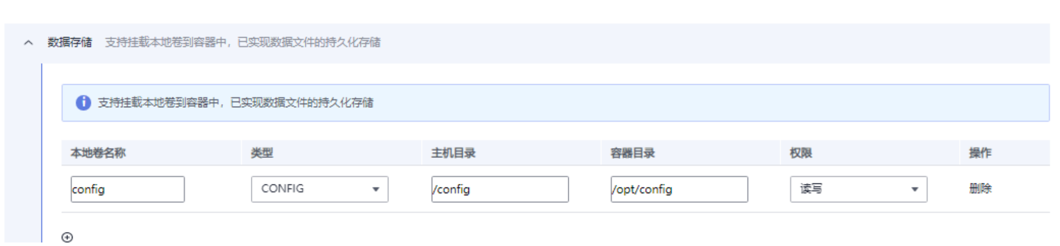
输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

步骤3 确认配置，填写基本信息。



⚠ 注意

集成ModuleSDK后，在高级配置中必须挂载config卷，主机目录和容器目录均为自定义，可参考下图。模块身份等信息均由SDK存储在config卷中，如果不配置挂载config卷，会导致自定义应用重启后无法与EdgeHub成功建立Mqtt连接。



----结束

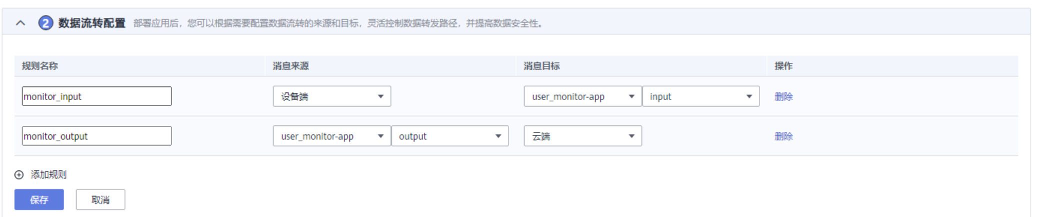
4.3.4.9 部署应用

步骤1 部署应用，具体参考部署应用。

⚠ 注意

IT应用需要依赖APIGW，在部署ITy应用之前，请先部署系统应用$sys_edge_apigw。

步骤2 添加流转规则



说明

流转规则是非必选的，OT应用需要添加数据流转规则。驱动应用和IT应用不用添加。

----结束

4.3.4.10 添加边缘设备

添加子设备请参照设备接入，以下是添加边缘设备（MQTT设备）配置时的参考：

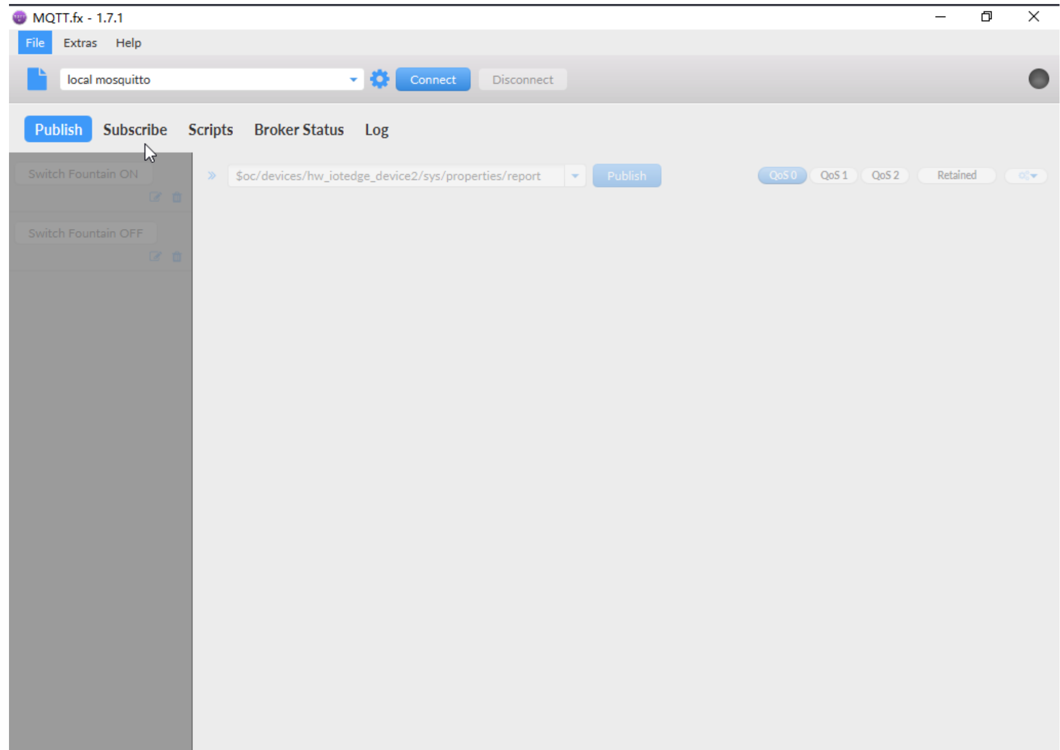
记住设备ID和密码，用于设备接入平台认证 。

4.3.4.11 设备接入

使用MQTT.fx模拟设备接入。

步骤1 下载MQTT.fx及证书，**证书下载地址**。

安装完成后打开，MQTT.fx软件界面如下：

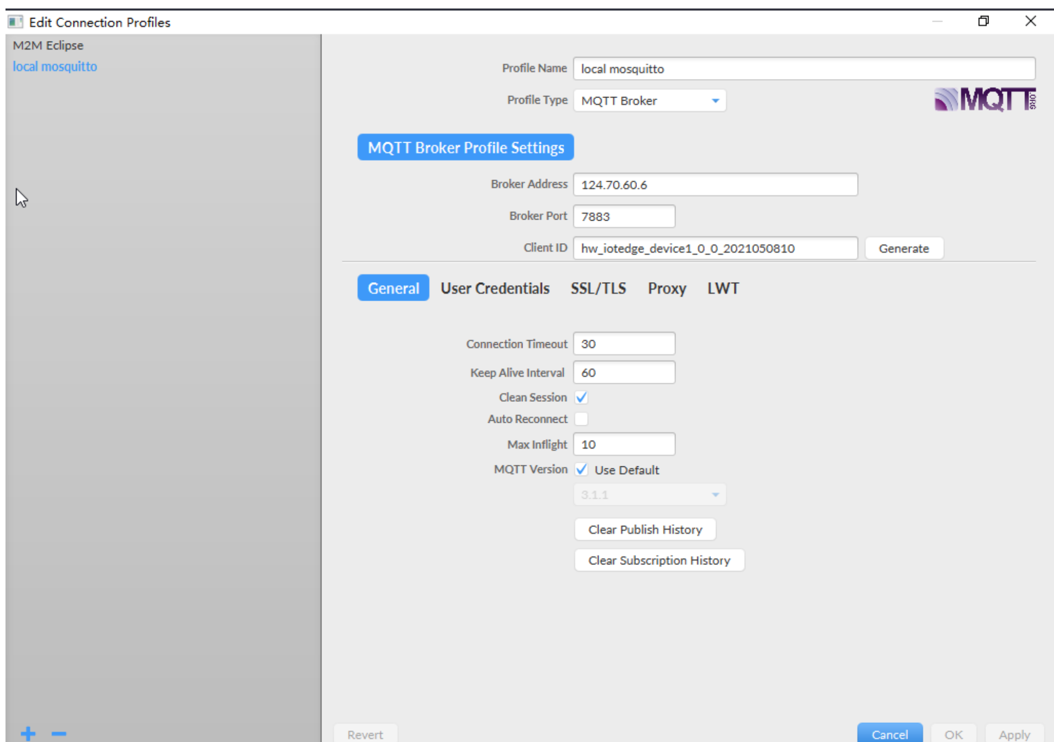


⚠ 注意

Connect左边的蓝色齿轮为设置。

Publish是消息发送，Subscribe为消息接收，Log可查看日志。

步骤2 单击设置-General，输入以下信息



Broker Address：输入节点的公网地址。

Broker Port: MQTTS协议使用的端口，默认为7883。

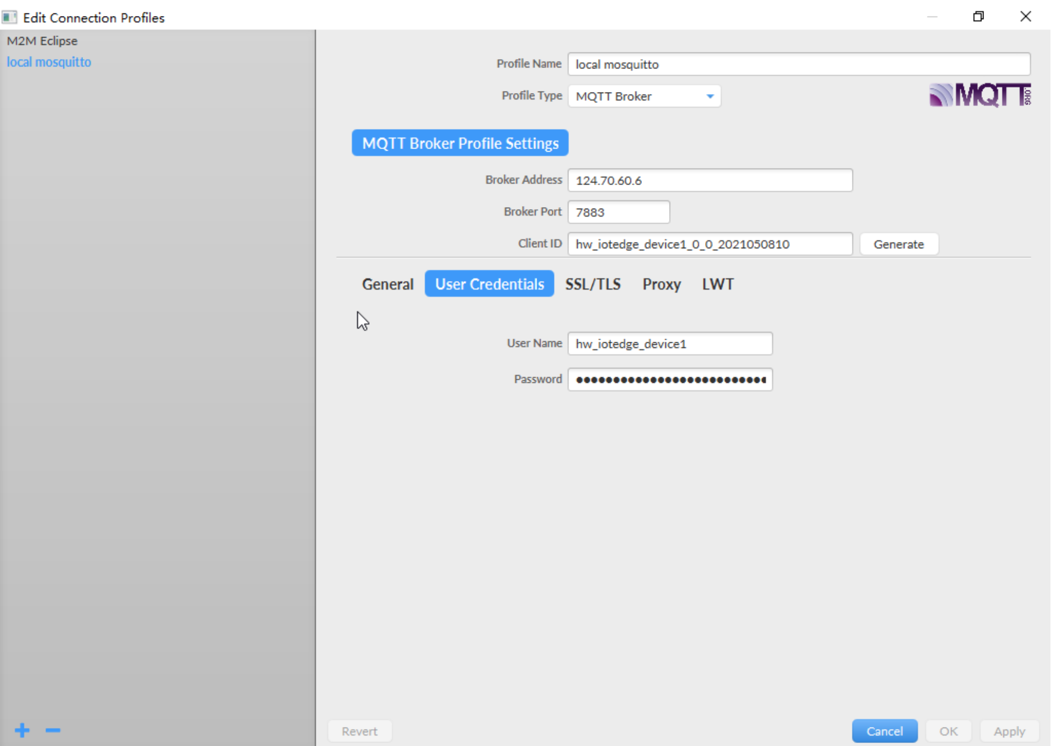
尝试连接时间和保持连接时间等自定义。

步骤3 单击设置-User Credentials

Client ID和密码需要工具进行转换。利用网页转换工具进行转换。

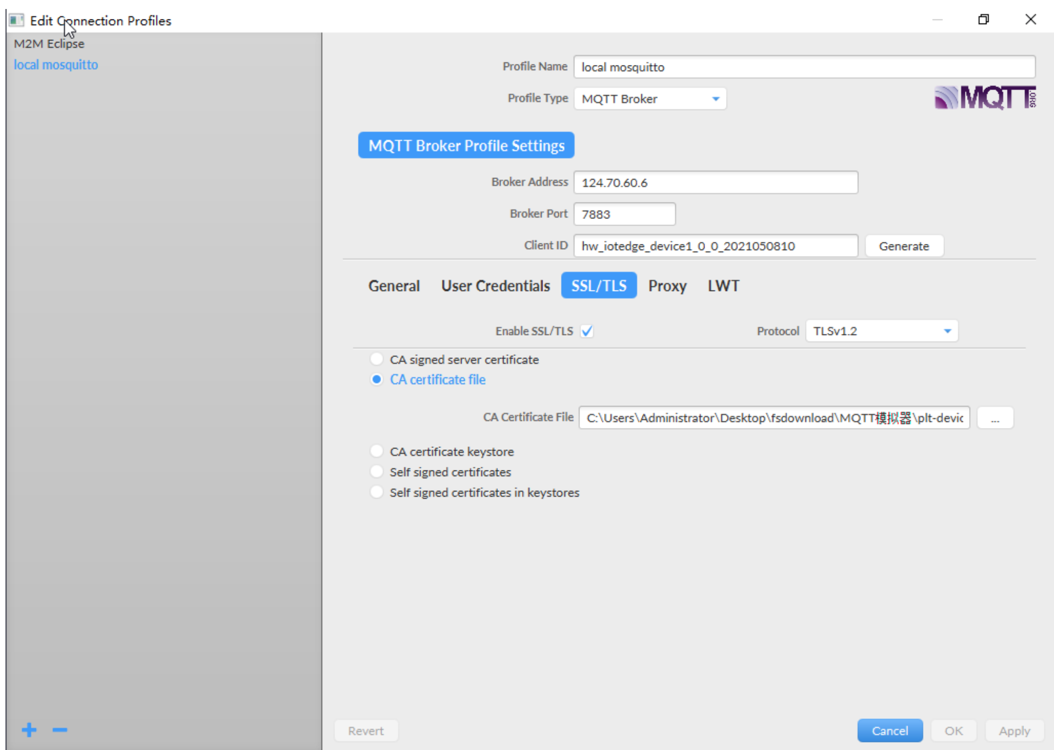
填写添加设备（IoT边缘）后生成的设备ID和设备密钥，生成连接信息（ClientId、Username、Password）。

版权所有 © 华为云计算技术有限公司



步骤4 单击设置-SSL/TLS

勾选Enable SSL/TLS，单击CA certificate，选择下载的证书文件。



单击Apply应用设置后返回。

单击Connect连接,连接成功后右边红点会变成绿色，IoTDA也会显示在线。

步骤5 选择publish输入topic地址。

Topic：$oc/devices/hw*iotedge*device2/sys/properties/report

其中，hw*iotedge*device2为设备ID，请替换为实际值,可在IoTDA->产品管理中查看。消息体输入：

```
{
 "services": [{
        "service_id": "power",
         "properties": {
             "status":"error"  },
         "event_time": "20210508T173342Z"
      }  ]
}
```

步骤6 进入边缘设备查看数据上报情况

进入IoTDA单击设备，进入概览发现并无数据上报，说明设备发送的数据在节点本地被集成SDK的monitor-app应用拦截。

步骤7 输入订阅的topic，可在IoTDA->产品->topic管理中查看。

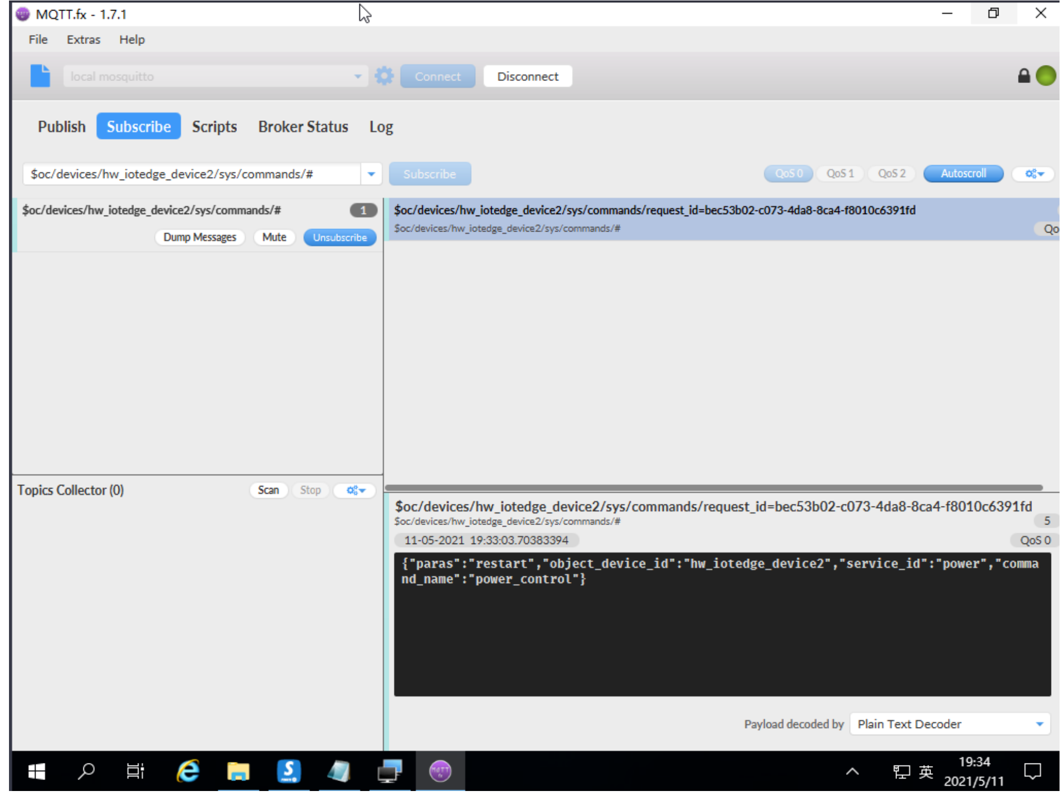
Topic：$oc/devices/hw_iotedge_device2/sys/commands/#

其中，hw_iotedge_device2为设备ID，请替换为实际值,设备ID请进入设备详情查看。

进入Subscribe,可以看到订阅命令收到一条command。这是因为monitor-app应用拦截了设备上报的数据。当数据内容为error时，monitor-app应用会向hub调用设备命令，

版权所有 © 华为云计算技术有限公司

该命令为重启操作，命令消息体如下图所示，具体处理逻辑见4.3.3.2.2。这说明集成appClient的应用可以实现了数据处理和命令下发的功能。



----结束

4.3.5 集成 ModuleSDK 进行工业子系统接入

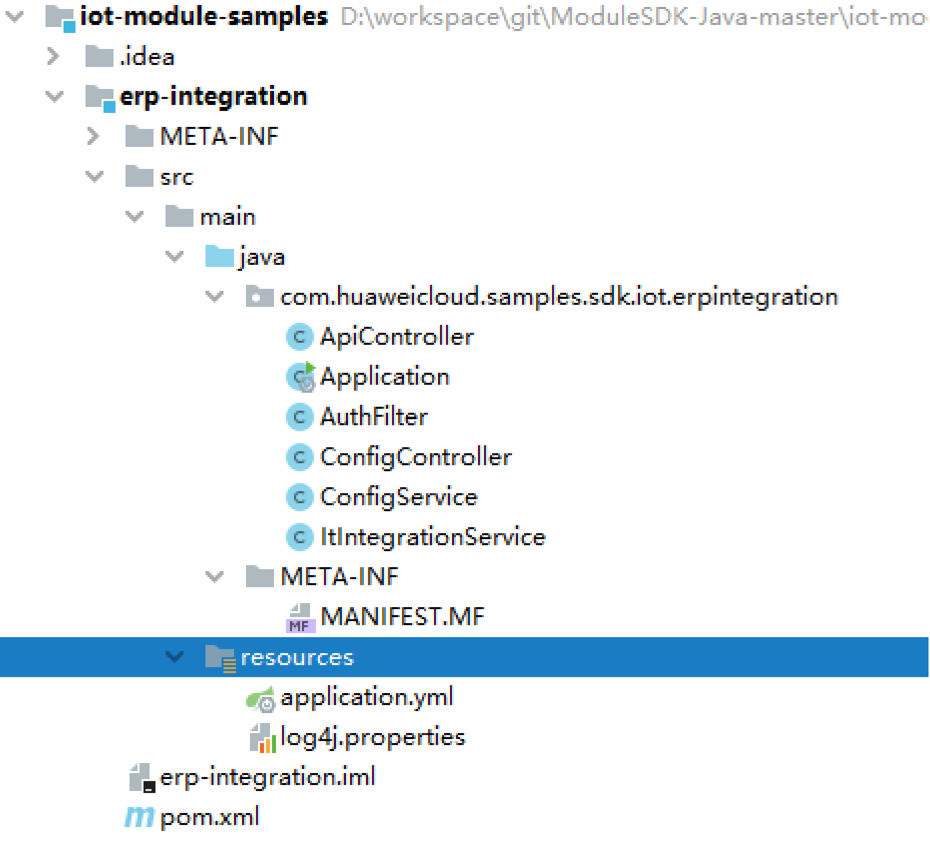
4.3.5.1 操作场景

开发应用集成ModuleSDK进行工业子系统接入。

用户在个节点下部署了多个子系统（如erp），北向应用NA需要调用某个子系统的接口,该子系统需要将order数据上传到用户的北向应用NA上。

4.3.5.2 代码解析

项目结构如下



- ApiController：提供被北向应用NA调用的接口。
- Application：主启动类
- AuthFilter：鉴权过滤器。
- ConfigController：被云端调用进行配置处理。
- ConfigService：配置管理服务 。
- ItIntegrationService：向北向应用NA发送数据。

代码解释使用ModuleSDK开发it子系统集成服务时使用的主要ItClient类。

ItClient类有以下几个关键方法（具体参考JavaDoc）。

- createFromEnv(): ItClient创建时由此方法自动获取环境变量。
- syncConfigs()：IT应用启动时由此方法实现从北向应用NA同步配置。
- confirmConfigs()：向北向应用NA确认已经同步的配置。
- sign()：对于发送到APIGW的请求，需要使用此方法进行签名。
- verify()：对于来自APIGW的请求，由此方法进行鉴权。
- **Json()：根据需求选择不同的方法向NA发送请求。

片段一

ItIntegrationService 类

```
@Scheduled(cron = "0 0/5 * * * ?")
public void collectData() throws HttpException, CryptException {
    //TODO 采集订单数据
    String body = "{\"orders\":\"data of orders\"}";
```

 版权所有 © 华为云计算技术有限公司

```
        itClient.postJson("/nas/erp/orders", body);
    }
```

对于来自子系统erp的数据orders（此处未真正接入erp系统），采用定时任务进行发送。

注意请求的地址（demo为"/nas/erp/orders"）,erp为NA的id(创建路由管理时定义)。

最终请求地址http://sys-edge-apigw:8900/nas/erp/orders

http://sys-edge-apigw:8900为Api GW接收请求的地址，Api GW接收到此请求会查询本地存储的NA信息中的API网关地址向网关发送请求，如API网关分组下的子域名为

068b72f3b75444dda67cc6e********.apic.cn-south-1.huaweicloudapis.com

则Api GW转发地址为https://068b72f3b75444dda67cc6e********.apic.cn-south-1.huaweicloudapis.com/oreders

API网关再将请求转发到定义的NA地址。如如API定义的后端NA为:

请求方式:http

host地址:110.*.*.*

端口号:8080

则API网关会将请求转发至http://110.*.*.*:8080/orders

注意

Api GW保存NA与IA信息的位置为：/var/IoTEdge/db/sys_edge_apigw/db
Api GW无法转发请求请查看该sqlite数据库是否正确保存了NA与IA的信息。

片段二

AuthFilter 类

```
@Order(1)
@WebFilter(filterName = "authFilter", urlPatterns = "/*")
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain
filterChain)
        throws IOException, ServletException {
        HttpServletRequest request = (HttpServletRequest) servletRequest;
        try {
            itClient.verify(request.getHeader("Authorization"));
            filterChain.doFilter(servletRequest, servletResponse);
        } catch (AuthException e) {
            HttpServletResponse response = (HttpServletResponse) servletResponse;
            response.sendError(HttpStatus.SC_FORBIDDEN);
        }
    }
```

对于来自Api GW的请求，此过滤器会进行拦截，并由itClient.verify()方法进行鉴权。

鉴权的方式为在请求头中增加Authorization字段，值的示例如下所示：

Algorithm=HMAC_SHA_256;AK=ia_1;SignedTime=1600763045361;Signature=0A1B0C3D

其中：

版权所有 © 华为云计算技术有限公司

Algorithm：表示签名使用的算法名称。

AK：表示客户端身份。

SignedTime：表示签名时间戳。

Signature：为使用由Algorithm指定的签名算法对以上相应字段进行签名的结果。

须知

对于发送到Api GW的请求，ItClient中提供的**Json()方法已做鉴权相关处理。

片段三

ConfigService 类

```
    //创建时运行一次
    @PostConstruct
    public void syncConfigs() {
        //TODO 先从本地加载配置项
        //从云端同步配置
        try {
            List<Config> configs = itClient.syncConfigs();
            configs.forEach(config -> configMap.put(config.getId(), config));
            //TODO 持久化保存配置
            //确认已同步的配置
            itClient.confirmConfigs(configs);
        } catch (GeneraException e) {
            System.out.println(e.getMessage());
        }
    }
```

创建时ItClient从云端ITIntegration获取配置并确认（IA>Api GW>云端ITIntegration）。

片段四

```
/**
 * 该类实现供北向应用调用的接口
 */
@RestController
@RequestMapping(value = "/api")
public class ApiController {
 private final String moduleId;

    @Autowired
    public ApiController(ItClient itClient) {
        this.moduleId = itClient.getConfig().getModuleId();
    }

    @GetMapping("/get")
    public AppResponse GetTest() {
        return AppResponse.success("GetTest Success", new ReturnDto(moduleId));
    }

    @PostMapping("/post")
    public AppResponse PostTest(@RequestBody Map<String, Object> inputArgs) {
        return AppResponse.success("PostTest Success", new ReturnDto(moduleId, inputArgs));
    }
}
```

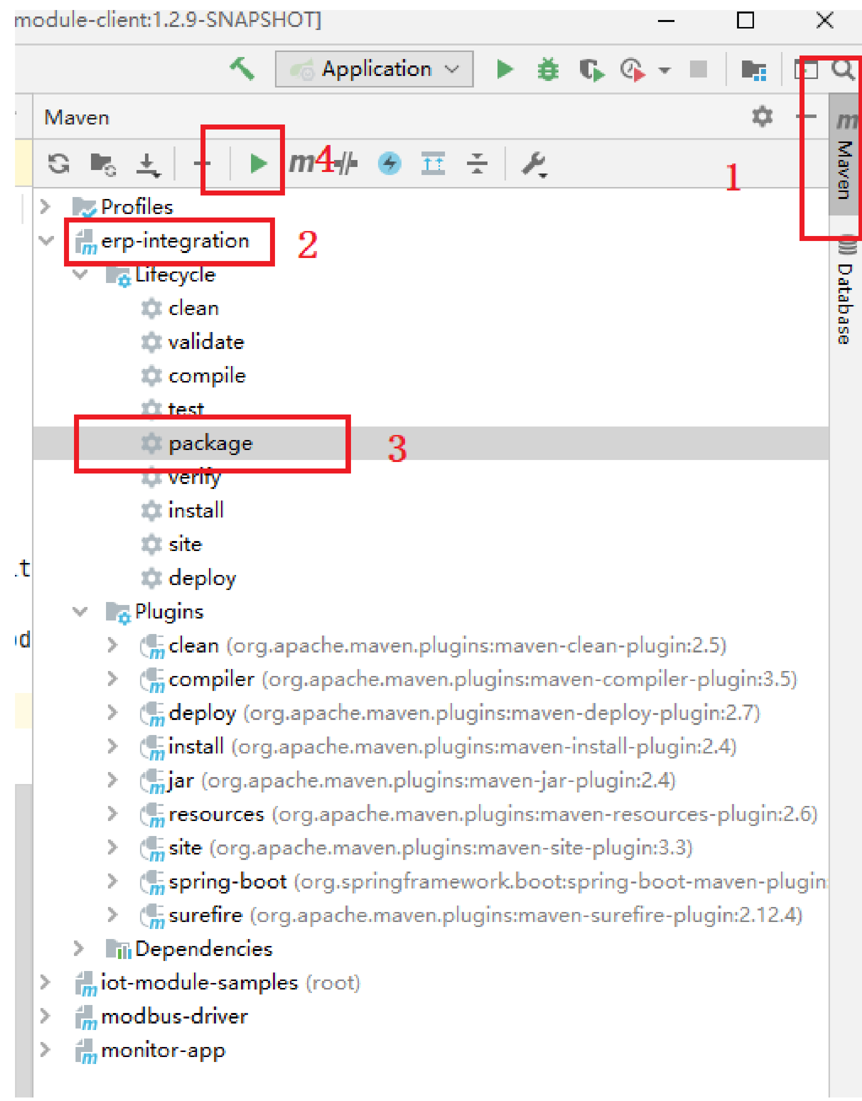
APIGW能将来自北向NA的响应（如配置）自ITIntegration请求转发送到iT应用(NA>ITIntegration>Api GW>IA)，IT应用可以由此实现对本地子系统的控制。

版权所有 © 华为云计算技术有限公司

EdgeApiGW收到南向3rdIA的应答后，需要将应答信息构造成用于进行websocket传输的ResponseDto，并调用sendResponse接口，发送该数据到云端。

/api/** 注：支持GET/POST/PUT/DELETE/PATCH五种方式。

4.3.5.3 项目打包



打包方式idea右上方>Maven>选择ero-integration模块>选择package>单击上方绿色三角按钮。完成打包。

4.3.5.4 制作镜像包

将jar打包成镜像文件上传，请参照制作镜像包或插件包。

4.3.5.5 添加应用

添加边缘应用具体请参考添加应用。

4.3.5.6 注册节点

创建边缘节点请参照注册边缘节点。

版权所有 © 华为云计算技术有限公司

注：对于需要部署IT应用的节点，节点需要绑定工业资源包。

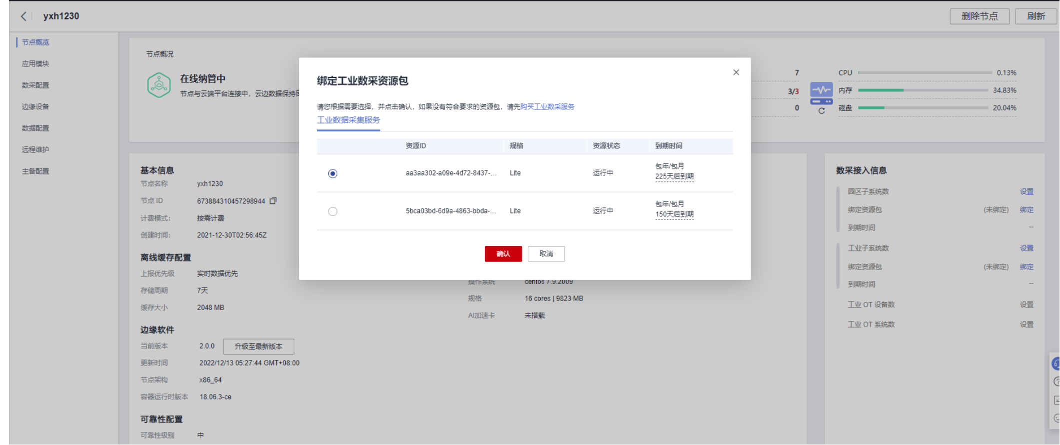
如没有工业网关资源包选项，请先购买。工业网关资源包计费及使用请参考工业资源包。

绑定方法如下：

步骤1 在左侧导航栏选择“IoT边缘 > 边缘节点”“选择之前创建的边缘节点，单击“节点名称”进入节点概览页。

步骤2 先设置“工业子系统数”再“绑定资源包”。

步骤3 绑定工业资源包选择“工业子系统采集服务”，请您根据需要选择绑定的资源包，并单击确认。



 版权所有 © 华为云计算技术有限公司

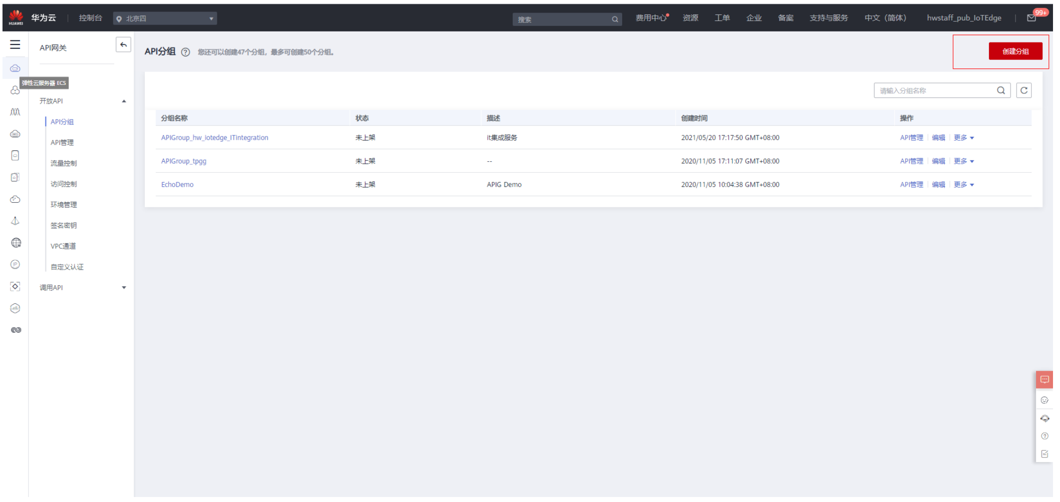
步骤4 在“节点概览”页可以看到“工业子系统数”以及“绑定的资源包”信息。

----结束

4.3.5.7 创建 API

步骤1 创建API分组。

API网关>开放API>右上方创建分组。



编辑分组

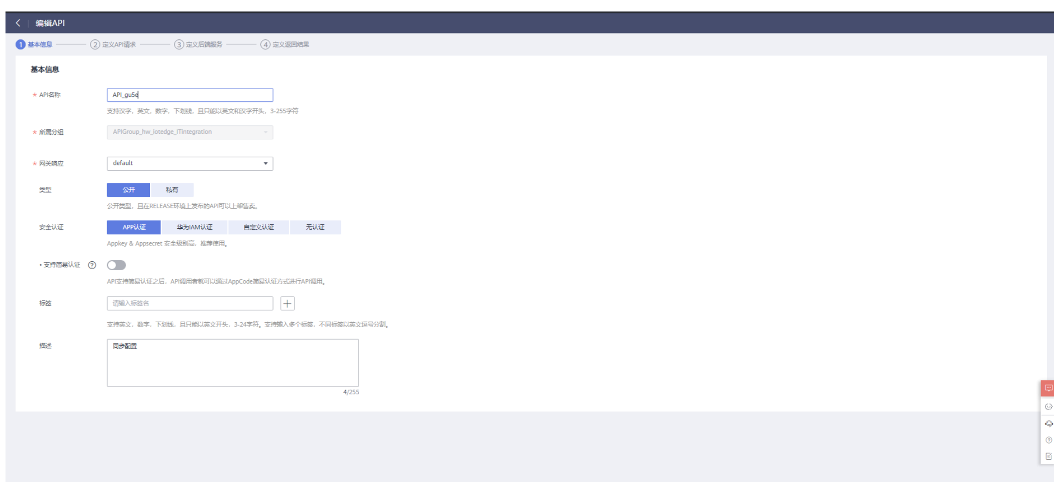
填写分组名称（自定义），带年纪确定完成创建。

步骤2 创建API。

进入API分组，创建API，API提供erp数据上传的请求转发功能。

创建过程说明：

1. 基本信息。



API名称：自定义。

所属分组：默认。

网关响应：默认。

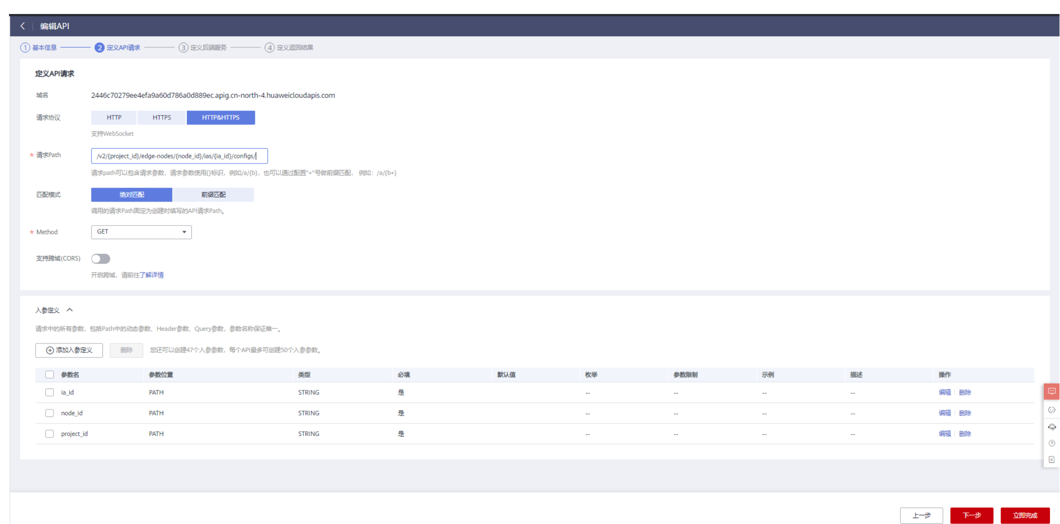
版权所有 © 华为云计算技术有限公司

类型：公开。

安全认证：App认证。

其他根据需要填写，没有则默认。

2. 定义API请求。



域名：默认。

请求协议：根据需要选择。

请求path:

IT应用发送数据的请求地址：/orders

匹配模式：绝对匹配。

Method：POST（根据请求方式选择）。

入参定义：

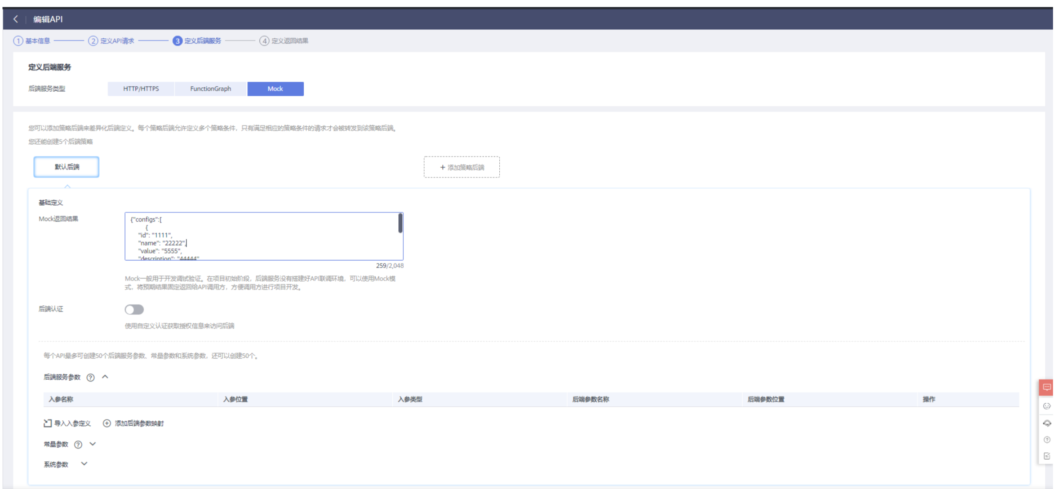
对于带有参数的请求需要声明入参定义，如请求path为"/configs/{ia_id}",则入参定义为：

参数名：ia_id

参数位置：path

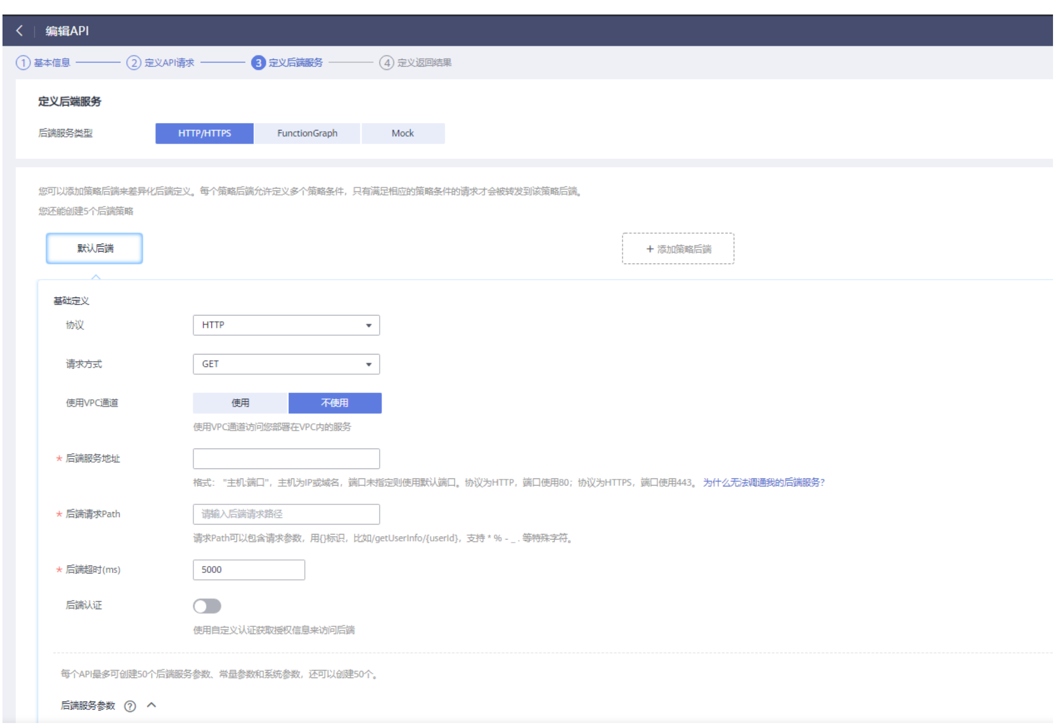
3. 定义后端请求。

 版权所有 © 华为云计算技术有限公司



后端服务类型提供了三种方式。

后端服务类型选择HTTP/HTTPS时的配置：



协议：根据北向应用NA使用的协议填写。

请求方式：根据北向应用NA定义的请求方式填写。

vpc通道：根据需求选择。

Virtual Private Network，虚拟专用网络。在公用网络上建立专用网络，进行加密通讯。在企业网络中有广泛应用。VPN网关通过对数据包的加密和数据包目标地址的转换实现远程访问。VPN可通过服务器、硬件、软件等多种方式实现。

后端服务地址：北向应用NA接收请求的ip或域名。

后端请求Path：北向应用NA接收请求的地址。

后端超时：自定义。

版权所有 © 华为云计算技术有限公司

注意

后端服务地址和后端请求Path很重要，这两项决定NA的请求地址。

后端服务类型选择Mock时的配置：

定义后端请求即定义API网关接收的请求将转发的位置，可通过Mock模拟后端响应。

Mock返回结果：此处定义的内容会被返回到请求端。（非必填）

返回示例：

```
{
  "configs": [
    {
      "id": "config1203",
      "name": "config1202",
      "value": "config1202",
      "description": "config1202",
      "version": "1606878222614",
      "state": "SUCCESS",
      "create_time": "2020-12-02T03:02:42Z",
      "update_time": "2020-12-16T08:44:33Z"
    }
  ]
}
```

须知

使用mock不能显示请求携带的数据，只能接收到请求后返回定义的结果。

4. 定义返回结果

成功响应示例：自定义。

失败响应示例：自定义。

5. 发布API

版权所有 © 华为云计算技术有限公司

将创建的API发布到release环境。

API网关>API分组>选择创建的API分组>API管理>勾选创建API>单击发布。

----结束

4.3.5.8 网关应用创建及绑定

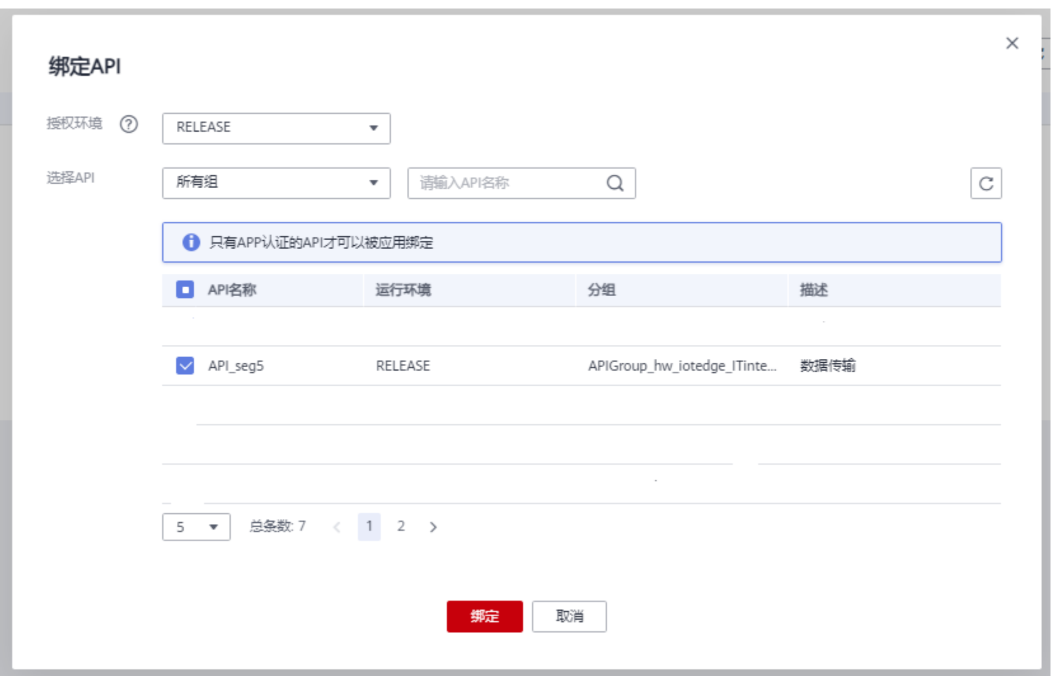
步骤1 创建网关应用

API网关>调用API>应用管理。

步骤2 绑定API

勾选前面创建的API，单击绑定。

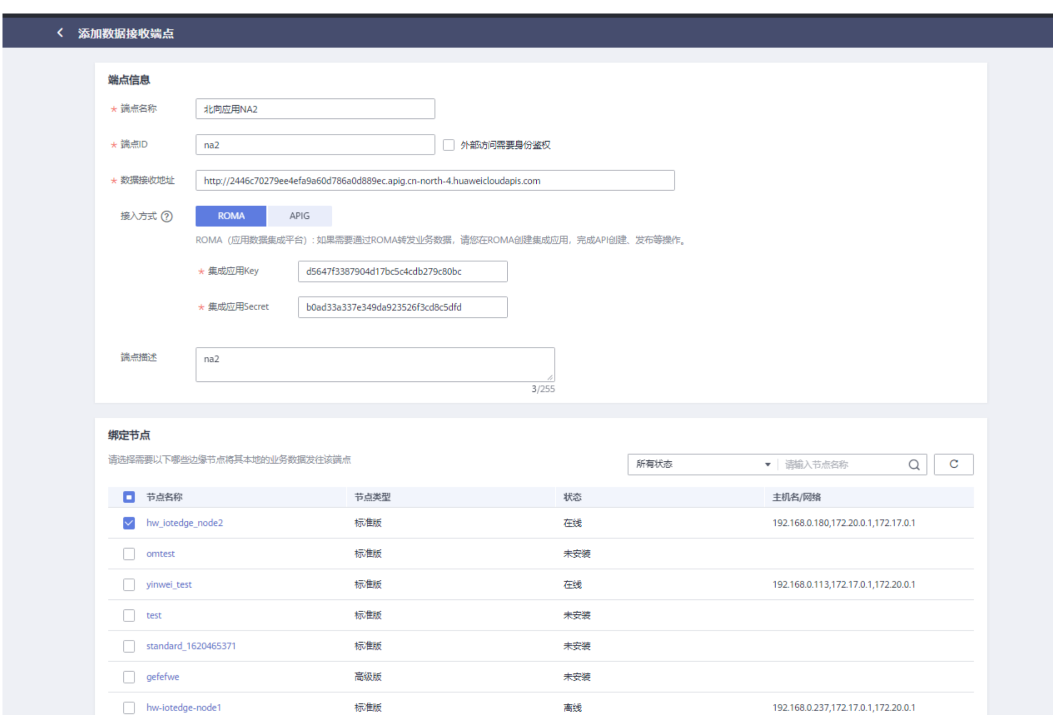
 版权所有 © 华为云计算技术有限公司



----结束

4.3.5.9 添加数据端点

IoT边缘>路由管理>添加数据接收端点。



端点名称：自定义。

端点ID：erp（端点id即为NA的id，代码中IA请求NA地址需要与此对应，如"/nas/erp/orders"）。

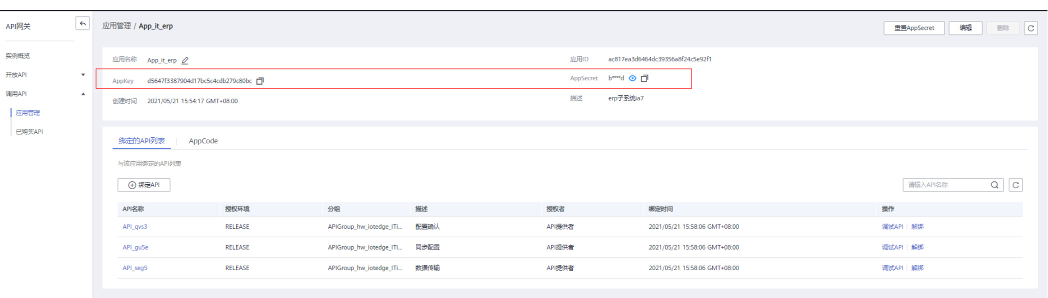
 版权所有 © 华为云计算技术有限公司

数据接收地址：

API分组绑定的子域名。创建API时会自动分配一个访问量和性能都极低的子域名用于开发测试（API分组>选择创建的API>概览），用于生产环境请绑定子域名！

接入方式：选择ROMA。

集成应用key和secret选择API应用分配的key和AppSecret(API网关>调用API>应用管理)

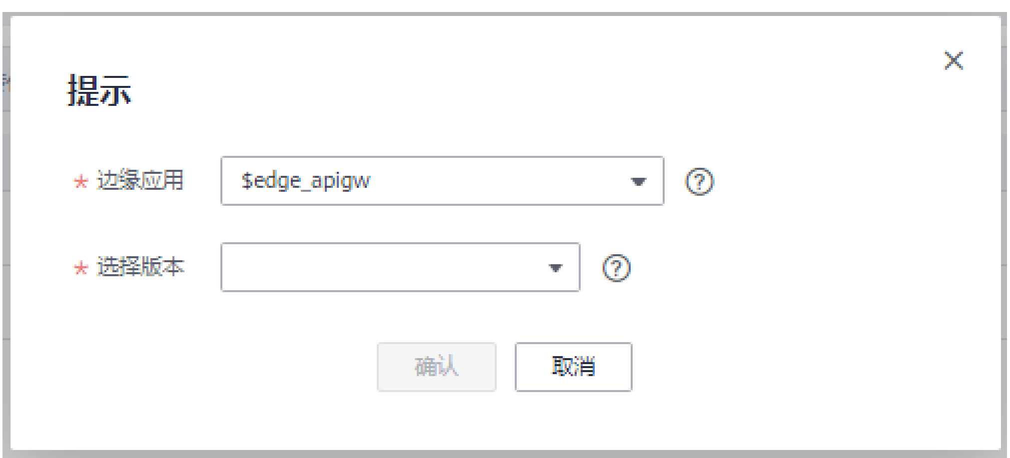


绑定节点：勾选为部署IT应用注册的节点。

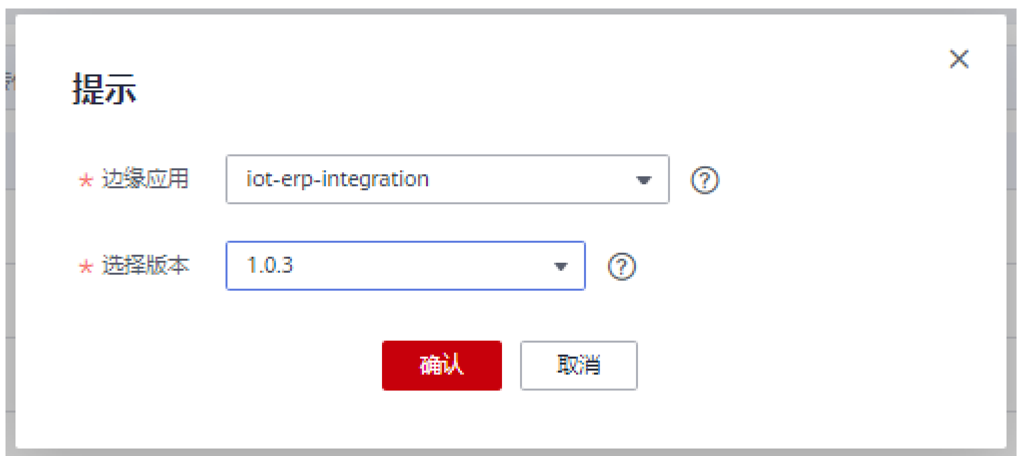
4.3.5.10 部署应用

IoTEdge 边缘>边缘节点>选择创建的节点>应用模块。

1. 部署$edge_apigw



2. 部署It应用



3.最终节点部署了四个应用

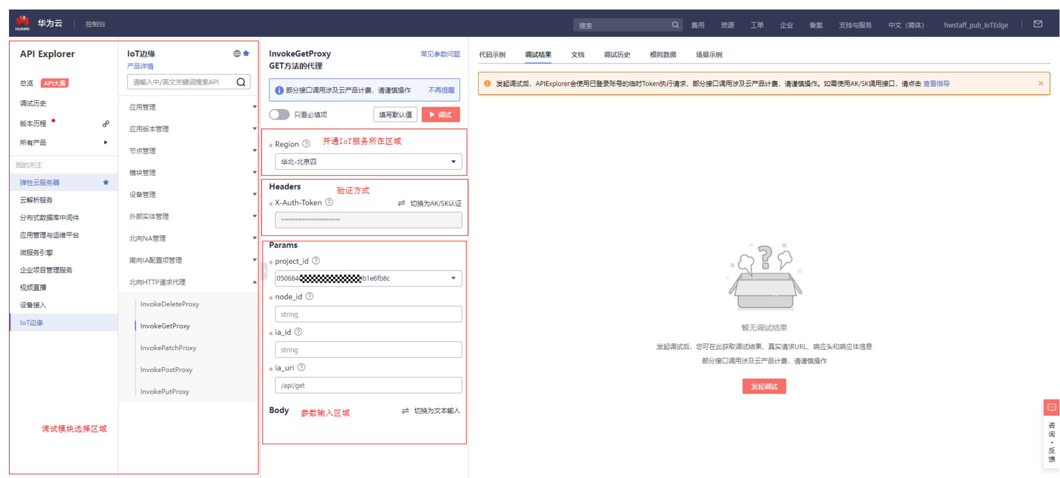
4.3.5.11 使用

1. 验证北向应用NA通过云端ITIntegration调用IT应用（IA）

验证方法可选择API Explorer、Postman等接口测试工具、实际构建南向应用NA三种方式来验证。

如果使用Postman等接口调试工具或者实际构建南向应用NA来测试请参考**API参考**。

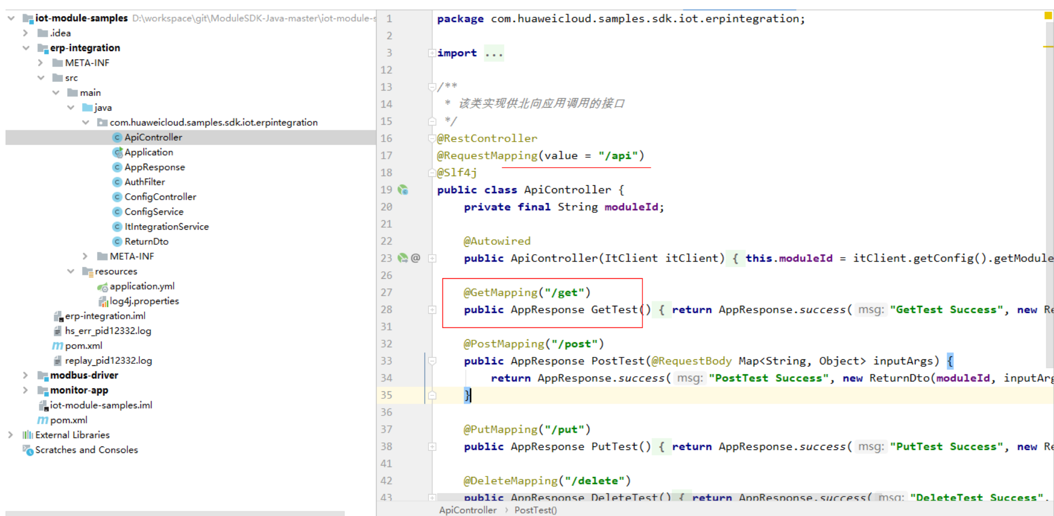
API Explorer界面如下：



如果调用开发的IT应用erp-integration中ApiController下的GetTest()方法，

即对应工程代码中的请求路径/api/get。

 版权所有 © 华为云计算技术有限公司



API Explorer设置如下：

在模块调试区域选择IoT>边缘>北向http请求代理>InvokeGetProxy。

Region：选择所在区域

Headers：选择AK/SK认证，即使用Authorization。

Params：

project_id：选择Region后自动填充（如没有自动填充请开通统一身份认证服务，开通参照**统一身份认证服务IAM文档**，项目id获取参照 **获取项目ID**）。

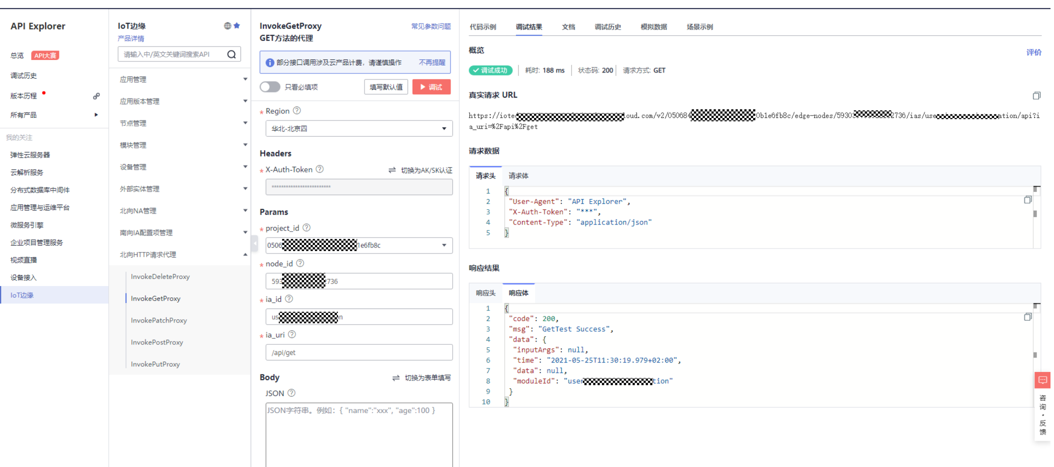
node_id：节点id，获取方式：边缘节点>节点管理>单击节点列表中节点名称>节点概览。

ia_id：IA应用id，获取方式：边缘节点 > 节点管理>单击节点列表中节点名称>应用模块>模块管理。

 版权所有 © 华为云计算技术有限公司

ia_uri：IA应用的接口地址，/api/get

填写完成后单击“调试”



右侧的返回结果即是IT应用（IA）响应的结果。

1. 验证IT应用（IA）通过API网关调用北向应用NA

以下代码是模拟NA接收来自API网关的转发请求。

```
@Controller
@Slf4j
public class ReceiveDataController {

    @RequestMapping(value = "/orders", method = RequestMethod.POST)
    @ResponseBody
    public void receive(HttpServletRequest request) {
        InputStream inputStream = null;
        ByteArrayOutputStream outputStream = null;
        try {
            inputStream = request.getInputStream();
            outputStream = new ByteArrayOutputStream();
            byte[] b = new byte[1024];
            int len;
            while ((len = inputStream.read(b)) != -1) {
                outputStream.write(b, 0, len);
            }
            log.info(new String("receive data:----------->" + new String(outputStream.toByteArray())));
        } catch (Exception e) {
            log.error("get InputStream from request failed");
        } finally {
            if (inputStream != null) {
                try {
                    inputStream.close();
                } catch (IOException e) {
                    log.error("close inputStream failed");
                }
            }
            if (outputStream != null) {
                try {
                    outputStream.close();
                } catch (IOException e) {
                    log.error("close outputStream failed");
                }
            }
        }
    }
}
```

将以上工程打包，部署到创建API时定义的后端服务器上运行(创建API网关时定义的host主机)。可以看到IA定时发送的数据。

版权所有 © 华为云计算技术有限公司

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.3.RELEASE)

2021-05-27 11:29:18.767  INFO 1599 --- [           main] c.huaweicloud.iotsamples.na.Application  : Starting Application on zxl-ecs-iot with PID 1599 (/home/iotedge_na-1.0-SNAPSHOT.jar started by root in /home)
2021-05-27 11:29:18.769  INFO 1599 --- [           main] c.huaweicloud.iotsamples.na.Application  : No active profile set, falling back to default profiles: default
2021-05-27 11:29:19.668  INFO 1599 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8088 (http)
2021-05-27 11:29:19.693  INFO 1599 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-05-27 11:29:19.693  INFO 1599 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.16]
2021-05-27 11:29:19.703  INFO 1599 --- [           main] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: [/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib]
2021-05-27 11:29:19.778  INFO 1599 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-05-27 11:29:19.779  INFO 1599 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 972 ms
2021-05-27 11:29:19.953  INFO 1599 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2021-05-27 11:29:20.193  INFO 1599 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8088 (http) with context path ''
2021-05-27 11:29:20.196  INFO 1599 --- [           main] c.huaweicloud.iotsamples.na.Application  : Started Application in 1.822 seconds (JVM running for 2.114)
2021-05-27 11:29:28.558  INFO 1599 --- [nio-8088-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2021-05-27 11:29:28.558  INFO 1599 --- [nio-8088-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2021-05-27 11:29:28.563  INFO 1599 --- [nio-8088-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
receive data:----------->{"orders":"data of orders"}

receive data:----------->{"orders":"data of orders"}
```

4.3.6 集成 ModuleSDK 进行协议转换

4.3.6.1 操作场景

使用ModuleSDK开发插件应用，接入其他协议设备（如HTTP请求数据），将其他协议的数据转化为MQTT协议JSON数据上报到IoTDA。

4.3.6.2 代码解析

项目结构如下

版权所有 © 华为云计算技术有限公司

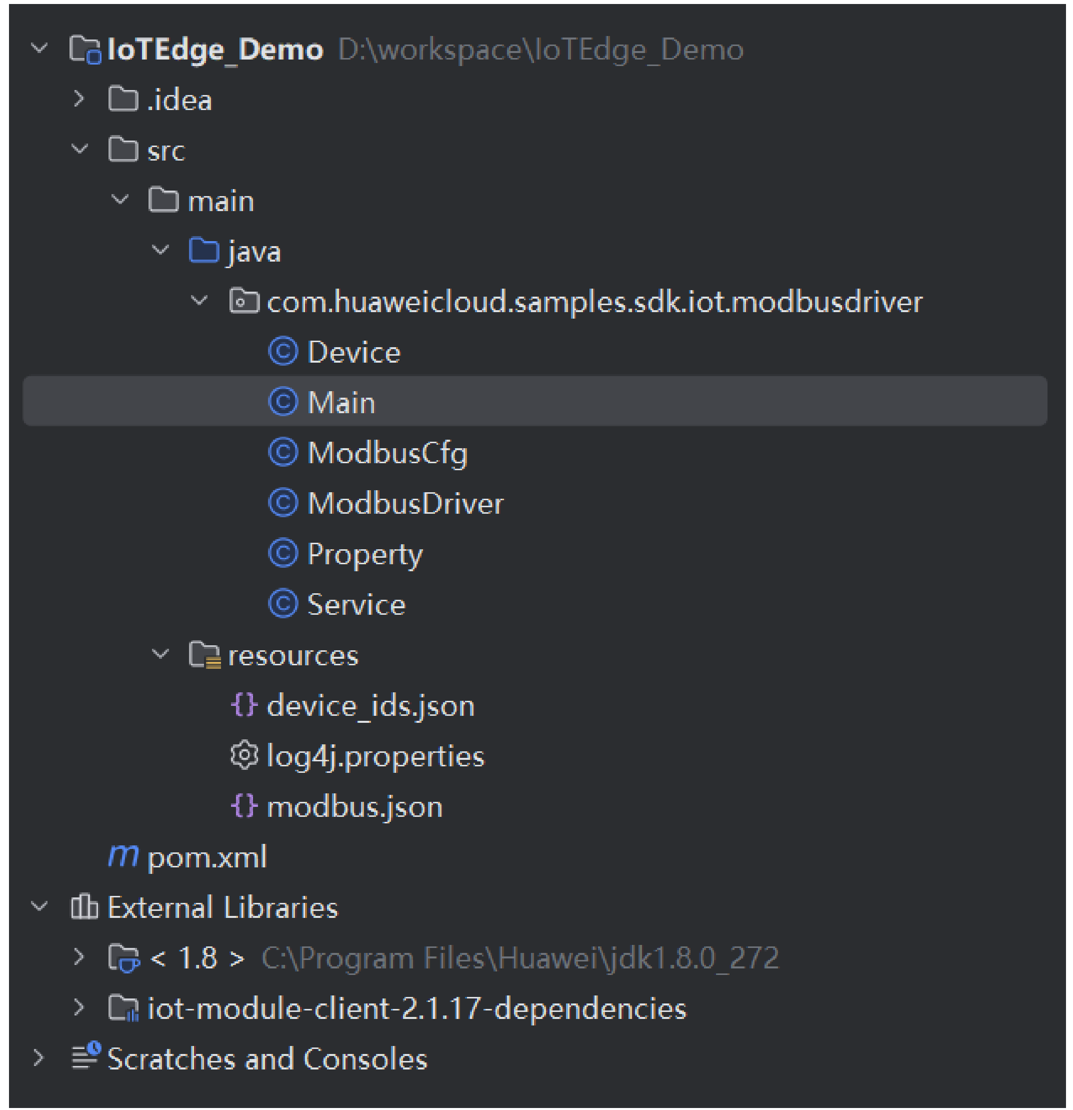


表 4-42 类相关说明

类名称	描述
Device	设备类，包含设备的id，产品id，slaveId及能力定义。
Main	主启动类。
ModbusCfg	modbus配置。
ModbusDriver	业务主体类，该类含边侧设备添加，设备同步，数据收集及上报等演示。
Property	设备属性。
Service	设备能力。

ModbusDriver代码解析

片段一

版权所有 © 华为云计算技术有限公司

通过DriverClient.createFromEnv初始化客户端，调用driverClient来完成数据的发送。

```
public ModbusDriver() throws GeneraException {
    driverClient = DriverClient.createFromEnv();
}

public void start() throws Exception {
    //设置回调，打开客户端
    driverClient.setGatewayCallback(this);
    driverClient.open();
    loadConfig();
    startCollection();
}
```

片段二

此为从变量points中获取设备数采结果，进行设备数据上报。

```
private void startCollection() {
scheduler.scheduleAtFixedRate(new Runnable() {
    @Override
    public void run() {
        //todo 采集modbus点位数据，保存到points

        //上报数据
        List<DeviceService> devices = new LinkedList<>();
        for (Device device : modbusCfg.getDevices()) {
            List<ServiceData> services = new LinkedList<>();
            for (Service service : device.getServices()) {
                Map<String, Object> properties = new HashMap<>();
                for (Property property : service.getProperties()) {
                    String key = device.getId() + "." + service.getServiceId() + "." + property.getPropertyName();
                    properties.put(property.getPropertyName(), points.get(key));
                }
                services.add(new ServiceData(service.getServiceId(), properties, ZonedDateTime.now()));
            }
            String deviceId = deviceIds.get(device.getId());
            if (deviceId != null) {
                devices.add(new DeviceService(deviceId, services));
            }
        }
        try {
            driverClient.reportSubDevicesProperties(new SubDevicesPropsReport(devices));
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}, 0, modbusCfg.getPeriod(), TimeUnit.SECONDS);

}
```

片段三

查看“modbus.json”文件，点位上报数据关系对应：设备id.模型id.服务id.属性名；设备id需要与添加边缘设备时设置的“设备标识码”一致。

```
{
 "period": 10,
 "server": "192.168.1.2:502",
 "devices": [{
  "id": "8616ac9b-1e66-4fbf-8871-1ca4cb7103fc",
  "product_id": "c6b39067b0325db34663d3ef421a42f6",
  "name": "light_1",
  "slave_id": 1,
  "services": [{
   "service_id": "light",
   "properties": [{
    "property_name": "switch",
    "register_address": 1234,
```

```
        "data_type": "bool"
      }]
    }]
  }, {
    "id": "c1ae5e25-23c6-4e01-b64f-7b00b4066667",
    "product_id": "c6b39067b0325db34663d3ef421a42f6",
    "name": "light_2",
    "slave_id": 2,
    "services": [{
      "service_id": "light",
      "properties": [{
        "property_name": "switch",
        "register_address": 5678,
        "data_type": "bool"
      }]
    }]
  }]
}
```

片段四

查看“device_ids.json”文件，设备id需要与添加边缘设备时设置的“设备标识码”一致。

```
{
  "8616ac9b-1e66-4fbf-8871-1ca4cb7103fc": "bf40f0c4-4022-41c6-a201-c5133122054a"
}
```

4.3.6.3 注册节点

注册节点，请参照注册边缘节点。

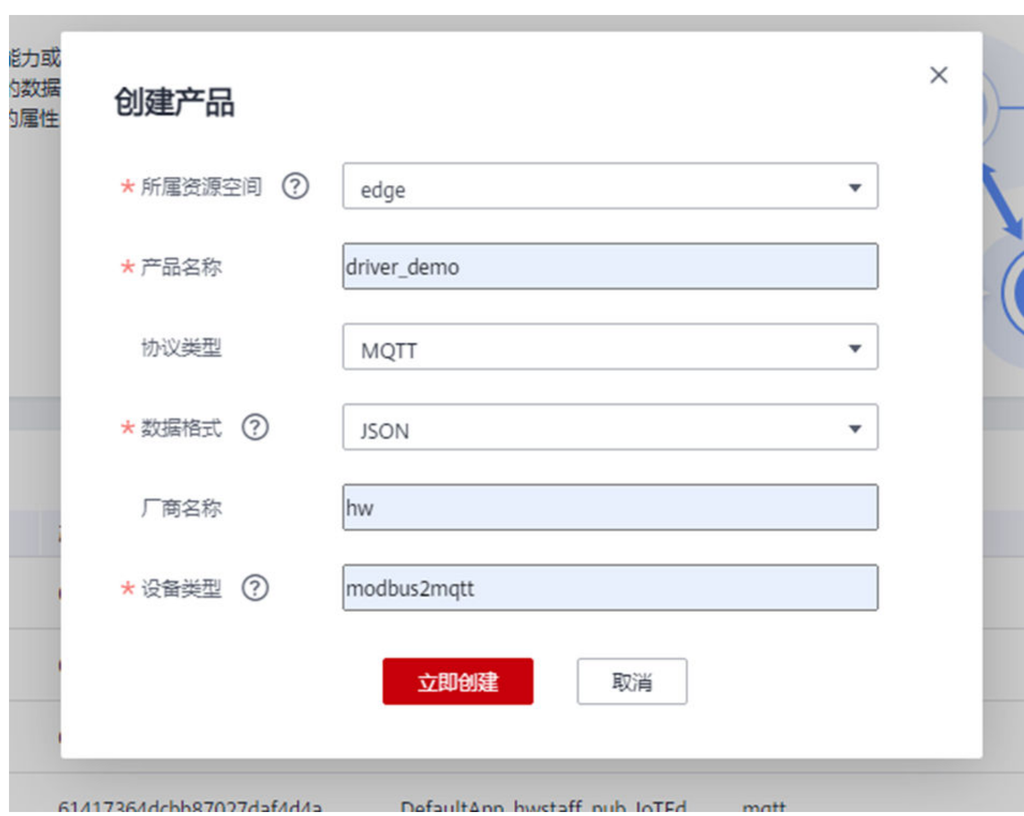
4.3.6.4 设备建模

步骤1 访问**IoT边缘**，单击**“管理控制台”**进入IoT边缘控制台。

步骤2 在左侧导航中选择“设备建模”，单击页面右上角“创建产品”。

步骤3 填写参数信息，如图所示，单击“立即创建”。

 版权所有 © 华为云计算技术有限公司



步骤4 进入产品详情页，单击“自定义模型”，添加“light”服务ID，并“确认”。

步骤5 展开服务列表，添加“switch”属性。

----结束

4.3.6.5 项目打包

打包参考项目打包

将modbusdriver进行打包得到modbusdriver.jar。

4.3.6.6 制作镜像包

将jar文件打包成镜像文件上，请参照制作镜像包或插件包。

dockerfile内容参照如下（具体可参考编写高效的Dockerfile ）。

```
FROM registry-cbu.huawei.com/csopenjdk/openjdk

RUN mkdir -p /opt/iot/edge/monitor / && chmod -R 777 /opt/

COPY monitor /opt/iot/edge/monitor
USER root
EXPOSE 8080
CMD ["java", "-jar", "/opt/iot/edge/monitor/monitor-app.jar", "run"]

#构造镜像
#docker build -t edge_monitor:1.0.0 /home --no-cache
#打标签
#docker tag edge_monitor:1.0.0 swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1
#推送
#docker push swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1
#打成镜像包
#docker save swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1 > ot_test.tar
```


版权所有 © 华为云计算技术有限公司

4.3.6.7 添加应用

以容器镜像方式为例，镜像包上传到容器镜像服务SWR后，创建应用。

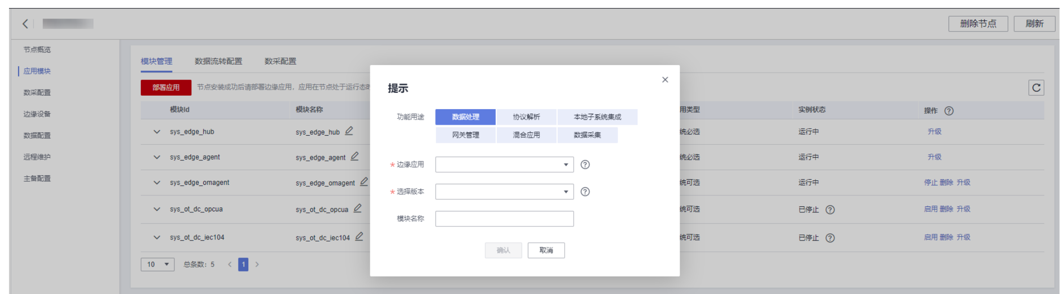
步骤1 在IoT边缘单击创建应用，进入软件部署配置、运行配置，并确认发布。

步骤2 在左侧导航栏，单击“应用管理”，选择“应用名称”进入页面，查看应用为“已发布”状态。

----结束

4.3.6.8 部署应用

部署应用，具体请参考部署应用，进入节点详情页安装应用。



版权所有 © 华为云计算技术有限公司

4.3.6.9 添加边缘设备

步骤1 进入边缘节点概览页，在左侧导航中选择“边缘设备”，单击“添加边缘设备”。

- 所属产品：选择**设备建模**中创建的产品
- 设备标识码：与代码示例保持一致
- 设备名称：与代码示例保持一致
- 模块ID: 与部署应用的模块id保持一致

图 4-7 添加边缘设备

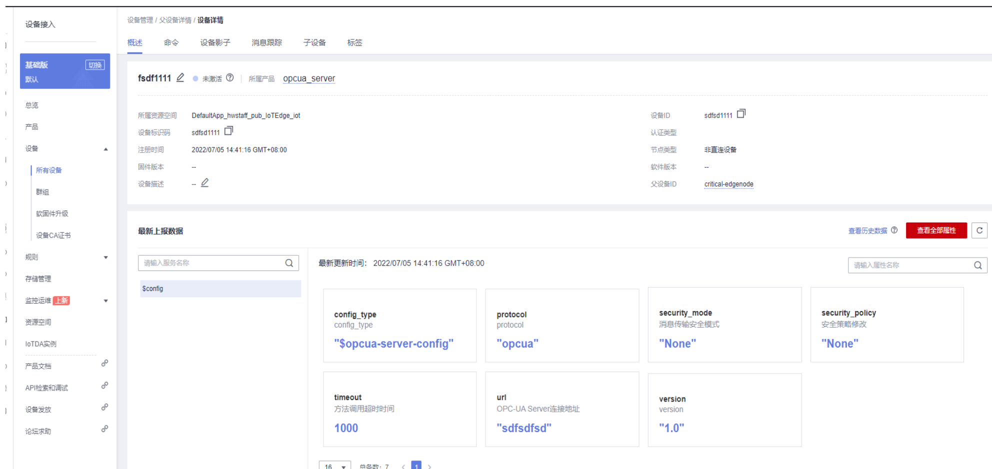
步骤2 单击“确认”，添加设备完成。

----结束

4.3.6.10 设备接入

启动HTTP服务端，进入设备详情页可看到上报的数据。设备状态显示未激活。如需更改可参照**集成ModuleSDK后，上报数据成功后，设备状态显示为未激活，如何上报子设备状态?**

图 4-8 边缘设备列表



4.3.7 集成 ModuleSDK 进行 OT 数采

4.3.7.1 操作场景

开发应用驱动集成ModuleSDK进行OT数采。（此示例以采集OPCUA为示例）

步骤1 在节点部署集成了ModuleSDK应用驱动。

步骤2 配置好数据源模板（可自定义）。

步骤3 在平台进行数据源配置，以及点位配置，下发。

步骤4 集成ModuleSDK应用驱动对平台下发的配置进行处理。

步骤5 通过下发信息获取数据源连接，以及点位信息进行数据周期采集。

步骤6 最后运用ModuleSDK的客户端进行点位数据进行周期上报。

----结束

4.3.7.2 代码解析

开发自定义驱动，进行OT数采。

```
public class DcDriver implements PointsCallback, ModuleShadowNotificationCallback {

    /**
     * 数采应用客户端，与边缘Hub建立MQTT连接
     */
    private DcClient dcClient;

    @PostConstruct
    void init() throws Exception {

        //打开客户端
        dcClient = DcClient.createFromEnv();
        dcClient.open();
        //设置回调，并同步模块影子
        dcClient.setPointsCallback(this);
        dcClient.startModuleShadow(this);
    }

    /**
     * 收到模块下行数采配置，消息需要缓存或持久化
     *进入边缘节点详情-》应用模块-》数采配置-》下发按钮
```


版权所有 © 华为云计算技术有限公司

```
 */
@Override
public void onModuleShadowReceived(ModuleShadowNotification shadow) {
    BriefModuleShadowDTO briefModuleShadowDTO = JacksonUtil.json2Pojo(
        JacksonUtil.pojo2Json(shadow.getProperties()), BriefModuleShadowDTO.class);
    connectDatasource(briefModuleShadowDTO.getConnectionInfo());
    collectData(briefModuleShadowDTO.getPoints());
}

private void collectData(Map<String, Object> points) {

    Map<String, Object> returnValues = new HashMap<>();
    points.forEach((k, v) -> {
        PointConfig pointConfig = JacksonUtil.json2Pojo(JacksonUtil.pojo2Json(points), PointConfig.class);
        Object value = collectPointDataFromOpcuaServer(pointConfig);
        returnValues.put(pointConfig.getPointId(), value);
    });
    //数据采集后上报数据
    pointsReport(returnValues);
}

/**
 * 上报服务器采集点位数据到EdgeHub
 */
private void pointsReport(Map<String, Object> points) {
    PointsReport report = new PointsReport();
    report.setPoints(points);
    try {
        dcClient.pointReport(report);
    } catch (JsonException e) {
        e.printStackTrace();
    }
}

/**
 * 根据根据opcua的点位信息从opcua服务器读取或者订阅数据
 */
private Object collectPointDataFromOpcuaServer(PointConfig pointConfig) {
    //todo 伙伴根据address和周期读取点位数据
    //示例从服务器读取到点位数据为10
    Object returnValue = 10;
    return returnValue;
}

/**
 * 根据数采配置的数据源连接参数完成数据源连接
 */
private void connectDatasource(Map<String, String> connectionInfo) {
    //opcua 服务器为示例,获取服务器连接地址
    String endpoint = connectionInfo.get("endpoint");
    //todo 伙伴根据endpoint实现连接数据源动作
}

/**
 * 收到点位设置的处理
 */
@Override
public PointsSetRsp onPointSet(String requestId, PointsSetReq pointsSetReq) {
    //PointsSetReq结构points为{pointId:value}的键值对，
    // 伙伴需要根据onModuleShadowReceived获取的数采配置实现写opcua服务器
    //正常写数据到opcua服务器响应示例
    return new PointsSetRsp(0,"success");
}

/**
 * 收到点位读取的处理
 */
@Override
public PointsGetRsp onPointGet(String requestId, PointsGetReq pointsGetReq) {
```


版权所有 © 华为云计算技术有限公司

```
        //PointsSetReq结构points为[pointId1,pointId2]的列表，
        // 伙伴需要根据onModuleShadowReceived获取的数采配置实现读取opcua服务器的点位信息
        //正常从opcua服务器读取点位数据响应示例
        PointsGetRsp rsp = new PointsGetRsp();
        Map<String, Object> points = new HashMap<>();
        for (String point : pointsGetReq.getPoints()) {
            points.put(point, 1);
        }
        return rsp;
    }
}
```

下发配置对象

```
public class BriefModuleShadowDTO {

    //数据源id
    @JsonProperty("ds_id")
    private String dsId;

    //数采模板默认参数
    @JsonProperty("default_values")
    private Map<String, String> defaultValues;

    //数据源附加参数
    @JsonProperty("collection_paras")
    private Map<String, Integer> collectionParas;

    //数据源连接信息
    @JsonProperty("connection_info")
    private Map<String, String> connectionInfo;

    //点位信息
    private Map<String, Object> points;
}
```

点位信息对象

```
public class PointConfig {

    //点位id
    @JsonProperty("point_id")
    private String pointId;

    //点位地址，opcua地址：address = "ns=3;i=1002"
    private String address;

    //数据类型int、int32、float、double、bool等
    @JsonProperty("data_type")
    private String dataType;

    //点位采集周期单位毫秒
    private Integer cycle;
}
```

4.3.7.3 注册节点

注册节点请参照**注册边缘节点**。

4.3.7.4 设备建模&发放

请参照**设备建模&发放**。

4.3.7.5 项目打包

打包参考**项目打包**。

版权所有 © 华为云计算技术有限公司

将集成ModuleSDK进行项目打包。

4.3.7.6 制作镜像包

将jar文件打包成镜像文件上，请参照**制作镜像包或插件包**。

dockerfile内容参照如下（具体可参考**编写高效的Dockerfile** ）：

```
FROM registry-cbu.huawei.com/csopenjdk/openjdk

RUN mkdir -p /opt/iot/edge/monitor / && chmod -R 777 /opt/

COPY monitor /opt/iot/edge/monitor
USER root
EXPOSE 8080
CMD ["java", "-jar", "/opt/iot/edge/monitor/monitor-app.jar", "run"]

#构造镜像
#docker build -t edge_monitor:1.0.0 /home --no-cache
#打标签
#docker tag edge_monitor:1.0.0 swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1
#推送
#docker push swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1
#打成镜像包
#docker save swr.cn-north-4.myhuaweicloud.com/iot_edge_test/ot_test:v1 > ot_test.tar
```

4.3.7.7 添加应用

请参考**添加应用**

4.3.7.8 部署应用

部署应用，具体参考**部署应用**，进入节点详情页安装应用。

4.3.7.9 OT 数采配置

此数据源以opcua模板为例。可参考**OT数采配置**自定义数据模板。自行开发集成ModuleSDK驱动应用，定义自己的数据源模板，以及点位信息后，下发配置。

4.3.7.10 查看采集结果

请参考**查看采集结果**。

4.3.8 集成 ModuleSDK 进行进程应用的开发

4.3.8.1 操作场景

使用ModuleSDK开发插件应用，并以进程方式跑在服务器上。

4.3.8.2 代码解析

代码解析样例：

- **数据处理代码解析**
- **工业子系统接入代码解析**
- **协议转换代码解析**

 版权所有 © 华为云计算技术有限公司

- OT数采代码解析

4.3.8.3 注册节点

注册节点，请参照注册边缘节点。

4.3.8.4 设备建模&发放

请参照设备建模&发放。

4.3.8.5 项目打包

打包请参考项目打包。

将集成ModuleSDK进行项目打包。

4.3.8.6 制作插件包

步骤1 插件包制作。

1. 插件包格式要求如下：
   插件包仅支持.tar.gz 、.tar或者 .zip格式。
   插件包结构如下：
   app.zip
   ├── ****.jar //可执行jar文件,必须
   ├── start.sh //启动脚本 必须文件 当前不提供参数方式启动
   └── stop.sh //停止脚本 非必须
2. 构建插件包。
   以monitor-app为例，在项目打包后得到monitor-app.jar
   在monitor-app.jar文件的同目录下创建start.sh,内容如下:

```
function log(){
    echo "`date "+%Y-%m-%d %T"`: $1"
}
log "[INFO] start execut process."
#调试时可打开，确认sdk需要的环境能被获取
#echo "${device_id}" > test_enviroment.file
pwd
#更新环境变量，防止找不到java命令。
source /etc/profile
#运行文件在/var/IoTEdge/downloaded-job/run下面
java -jar ./monitor-app.jar  > monitor_running.log  2>&1
```

   将monitor-app.jar和start.sh一起压缩得到monitor-app.zip。

注意

1. 插件包升级时，会删除运行目录的所有文件，注意持久化文件的存储。插件包的运行路径为{installer_dir}/IoTEdge/downloaded-job/run/{moduleId}/{appVersion}/。
2. 当前插件包的大小限制为最大500M。
3. 程序内对于文件的访问使用相对目录访问（因为程序的安装目录是不确定的）。
4. 程序不允许包含后台运行的程序，可以包含多级进程，所有程序均为start.sh的子进程。
5. 进程压缩包命名规范：英文字母或者数字或者“_”,”.”，长度不超过64，不允许出现空格。

说明

1. 插件包为一层压缩结构，即插件包的压缩包解压之后直接为start.sh脚本所在目录的结构形式，不能多一层目录。
2. start.sh脚本为必须脚本，启动时默认调用该脚本进行启动，当前支持root用户以及非root用户（固定为1000用户,非root需要确定是否能够成功依赖系统库）启动，用户可以在start.sh脚本中自由修改自己的启动方式以及环境变量的修改等。
3. stop.sh为非必须的脚本，但是用户如果需要优雅停止的话，需要在该脚本中书写自己进程的停止方式（文件监控、接口调用等）。如果没有该脚本的话，默认对进程组先发送SIGTERM信号，如果进程组对该信号没有处理，达到最大等待时间则发送SIGKILL信号强制停止。整个停止的最大周期为10s。

步骤2 插件包上传。

1. 开通对象存储服务OBS。

   进程包上传方式需要开通对象存储服务OBS，请参考**对象存储服务 OBS_快速入门**。

2. 上传进程包。

   上传方式，请参照**对象存储服务(OBS)**。

注意

请设置桶策略为【公开读】，如未设置请前往“OBS首页 > 单击桶ID > 访问权限控制 > 桶策略中设置”。

----结束

4.3.8.7 添加应用

以安装包部署方式为例，将应用程序打包成安装包文件，并上传到 **对象存储服务(OBS)**。

步骤1 在IoT边缘单击创建应用，进入软件部署配置、运行配置，并确认发布。

步骤2 在左侧导航栏，单击“应用管理”，选择“应用名称”进入页面，查看应用为“已发布”状态。

----结束

4.3.8.8 部署应用

部署应用，具体请参考**部署应用**，进入节点详情页安装应用。

4.4 集成 ModuleSDK(C)

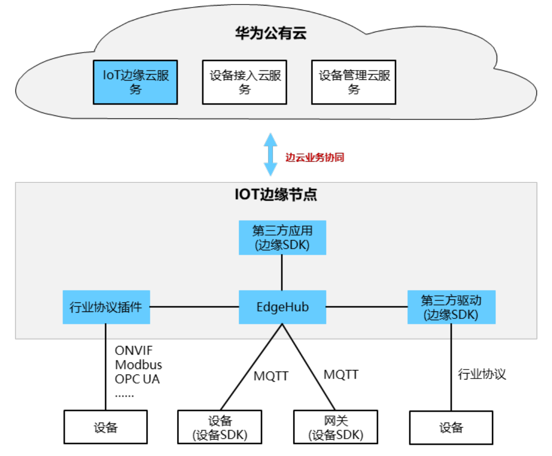
4.4.1 内部架构

模块SDK用于开发运行在边缘节点中的应用，包括数据处理应用（简称应用，例如数据清洗）和协议驱动（简称驱动，例如EdgeAccess）。

版权所有 © 华为云计算技术有限公司

图 4-9 边缘设备接入和应用开发能力架构图

边缘设备接入和应用开发能力



设备接入

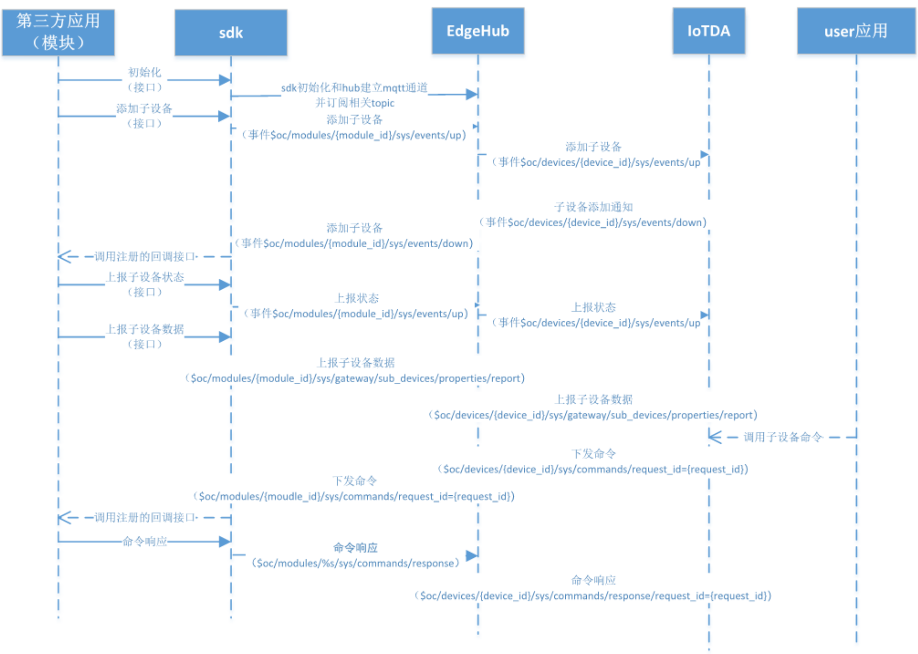
- 设备直接接入（MQTT）
- 通过网关接入（MQTT）
- 设备通过行业协议插件接入（ONVIF、Modbus、OPC UA等）
- 设备通过第三方驱动接入（行业协议）

开发SDK

- 设备和网关，通过设备SDK进行开发
- 第三方驱动、第三方应用，通过边缘SDK进行开发（提供子设备同步、产品同步、子设备命令调用等能力）

部署在边缘节点中的第三方应用和第三方驱动使用模块SDK进行开发。

图 4-10 模块 SDK 调用流程图



基本调用流程如图2所示。

基于ModuleSDK-C开发应用实现数据的云边同步，主要分为开发和使用两个部分。

开发操作	开发说明
SDK应用的开发	按照SDK提供的Demo进行自定义应用开发。

版权所有 © 华为云计算技术有限公司

开发操作	开发说明
SDK应用的使用	将应用打包上传至华为云，部署到节点。连接子设备查看应用工作情况。

ModuleSDK-C提供了以下头文件，用户可根据需求引用相应的头文件：

头文件	说明
edge.h	实现基本的回调函数和接口函数
edge_daemon.h	实现所有证书、鉴权相关接口
edge_driver.h	驱动模块，用于开发驱动接入设备，驱动需实现网关回调函数
edge_error.h	错误码定义
edge_struct.h	所有可能使用到的头文件定义
edge_dc_driver.h	数采驱动模块，用于开发数采驱动接入点位数据，驱动需实现相关回调函数

4.4.2 开发指导

4.4.2.1 接口函数

表 **4-43** edge.h 中支持的接口函数

接口	说明
edge_init	初始化sdk的工作环境，加载证书、读取配置等
edge_set_callbacks	注册回调接口，登录、子设备添加、删除、设备消息、设备命令等都是异步通知的，故需要注册对应的处理函数
edge_login	用于SDK登录边缘，该函数是异步函数，登录成功会调用4.2章节中已注册的pfn_connected回调函数
edge_logout	用于SDK登出边缘
edge_destroy	用于清理SDK资源
edge_send_add_sub_device	网关添加子设备，可批量添加
edge_send_delete_sub_device	删除子设备，可批量删除

 版权所有 © 华为云计算技术有限公司

接口	说明
edge_send_batch_device_data	批量上报子设备数据，可以同时上报多个设备的多个service数据
edge_send_batch_device_data_v2	该接口主要用于上报单个设备单服务的属性数据
edge_send_batch_device_data_v3	上报子设备数据，自行构造json数据
edge_send_batch_device_discovery_data	上报子设备数据，若设备不存在则上报子设备发现事件
edge_send_command_rsp	用于响应平台下发的设备命令
edge_send_get_sub_device_shadow	用于网关发送请求给平台获取子设备影子
edge_send_sub_device_event	用于网关给平台发送子设备事件
edge_send_get_product	用于网关给平台发送获取产品的请求
edge_send_sub_device_property_get_rsp	用于网关响应平台的子设备属性获取请求
edge_send_sub_device_property_set_rsp	用于网关响应平台的子设备属性设置请求
edge_send_customized_message	用于网关发送自定义topic消息
edge_set_bus_message_cb	为消息总线输入点设置回调函数
edge_send_bus_message	向消息总线输出点发送数据
edge_call_device_command	调用产品模型里定义的命令
edge_set_send_mode	设置发送模式（异步|同步）
edge_forbid_send_when_broker_overloaded	设置是否在离线缓存满时禁止发送
edge_get_devices_statuses	查询子设备状态
edge_init_devices_statuses	初始化用于存放子设备状态查询结果的结构体
edge_release_devices_statuses	释放子设备状态结构体

详细接口说明参见代码解析。

 版权所有 © 华为云计算技术有限公司

说明

include目录下的头文件**edge.h**，包括所有C SDK提供的回调函数和接口。以下是具体函数介绍。

edge.h 回调函数说明

1. 模块影子数据回调

   函数描述：

   ```
   typedef  EDGE_RETCODE (FN_SHADOW_ARRIVED)(const char* shadow, unsigned int shadow_len);
   ```

   函数功能：

   当下发模块配置时，通过此函数通知到用户，用户应用的配置也通过此函数通知。

   表 4-44 参数说明

参数名称	类型	参数描述	示例
shadow	char*	模块影子数据（json字符串），第三方应用下发用户的配置数据	{ “config”:”test” }
shadow_len	unsigned int	影子数据的长度	-

2. 命令下发回调

   函数描述：

   ```
   typedef EDGE_RETCODE (FN_COMMAND_ARRIVED)(const char* command_name, const char* device_id, const char* service_id, const char* request_id, const char* body, unsigned int body_len);
   ```

   函数功能：

   此函数声明用户命令下发通知，设备命令下发即使用此函数声明。

   表 4-45 参数说明

参数名称	类型	参数描述
command_name	char*	设备命令名称，在设备关联的产品模型中定义。
device_id	char*	命令对应的目标设备ID，命令下发对应的最终目标设备
service_id	char*	设备的服务ID，在设备关联的产品模型中定义
request_id	char*	{request_id}用于唯一标识这次请求，响应该命令时需要带上。

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述
body	char*	json字符串，设备命令的执行参数，具体字段在设备关联的产品模型中定义
body_len	unsigned int	-

3. 消息下发回调

   函数描述：

```
typedef  EDGE_RETCODE (FN_MESSAGE_ARRIVED)(const char* message_id, const char* message_name, const char* device_id, const char* body, unsigned int body_len);
```

   函数功能：

   此函数声明用于消息下发通知，平台使用此接口承接平台下发给设备的自定义格式的数据。

表 4-46 参数说明

参数名称	类型	参数描述
message_id	char*	消息名称
message_name	char*	消息的唯一标识
device_id	char*	命令对应的目标设备ID，命令下发对应的最终目标设备
body	char*	消息内容。
body_len	unsigned int	消息长度

4. 事件下发回调

   函数描述：

```
typedef  EDGE_RETCODE (FN_DEVICE_EVENT_ARRIVED)(const char* device_id, const char* body, unsigned int body_len);
```

   函数功能：

   此函数声明用于事件下发通知，平台使用此接口承接平台下发给设备的自定义格式的数据。

表 4-47 参数说明

参数名称	类型	参数描述	示例
device_id	char*	命令对应的目标设备ID，命令下发对应的最终目标设备	-

版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述	示例

body	char*	事件内容。	{
			"object_device_id":"deviceId",
			"services":[
			{
			"service_id":"serviceTest",
			"event_id":"eventTest",
			"event_type":"eventTypeTest",
			"event_time":"time",
			"paras":{
			"test":"test"
			}
			}
			]
			}

body_len	unsigned int	消息长度	-

5. 子设备添加回调

   函数描述：

```
typedef  EDGE_RETCODE (FN_SUB_DEVICE_ADD_ARRIVED)(const char* addSubDeviceInfo,
unsigned int body_len);
```

   函数功能：

   此函数声明用于通知子设备添加，使用此接口承接平台添加成功子设备的通知。

表 4-48 参数说明

参数名称	类型	参数描述	示例
addSubDeviceInfo	char*	添加成功的子设备信息，具体格式见示例	{"devices": [{"parent_device_id":"c6b39067b03421a48", "node_id": "subdevice11", "device_id":"2bb77-063ad2f5a6cc", "name": "subDevice11", "description": null, "product_id":"c6b3b34663d3ea42f6", "fw_version": null, "sw_version": null, "status": "ONLINE" }],"version":1}

参数名称	类型	参数描述	示例
body_len	unsigned int	长度	-

6. 子设备删除回调

   函数描述：

```
typedef  EDGE_RETCODE (FN_SUB_DEVICE_DELETED_ARRIVED)(const char*
deleteSubDeviceInfo, unsigned int body_len);
```

   函数功能：

   此函数声明用于通知子设备删除，使用此接口承接平台删除成功子设备的通知。

表 4-49 参数说明

参数名称	类型	参数描述	示例
deleteSubDeviceInfo	char*	删除成功的子设备信息，具体格式见示例	{"devices": [{"parent_device_id":"c6b39067b03421a48", "node_id": "subdevice11", "device_id":"2bb77-063ad2f5a6cc", "name": "subDevice11", "description": null, "product_id":"c6b3b34663d3ea42f6", "fw_version": null, "sw_version": null, "status": "ONLINE" }],"version":1}
body_len	unsigned int	长度	-

7. 子设备扫描回调

   函数描述：

```
typedef  EDGE_RETCODE (FN_START_SCAN_ARRIVED)(const char* protocol, const char*
channel, const char* parentDeviceId,const char* scan_setting, unsigned int body_len);
```

   函数功能：

   此函数声明用于通知网关扫描子设备。

表 4-50 参数说明

参数名称	类型	参数描述
protocol	char*	协议

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述
channel	char*	通道信息
parentDeviceId	char*	父设备ID
scan_setting	char*	扫描设备
body_len	unsigned int	配置长度

8. 子设备属性设置回调

   函数描述：

```
typedef EDGE_RETCODE (FN_DEVICE_PROPERTIES_SET_ARRIVED)(ST_PROPERTY_SET*
sub_device_property_set);
```

   函数功能：

   此函数声明用于接收平台对子设备的属性设置。

   表 4-51 参数说明

参数名称	类型	参数描述	示例
sub_device_property_set	ST_PROPERTY_SET*	属性设置	参见edge.h

9. 子设备属性获取回调

   函数描述：

```
typedef EDGE_RETCODE (FN_DEVICE_PROPERTIES_GET_ARRIVED)(ST_PROPERTY_GET*
sub_device_property_get);
```

   函数功能：

   此函数声明用于接收平台获取子设备的属性。

   表 4-52 参数说明

参数名称	类型	参数描述	示例
sub_device_property_get	ST_PROPERTY_GET*	属性设置	参见edge.h

10. 子设备影子回调

   函数描述：

```
typedef EDGE_RETCODE (FN_DEVICE_SHADOW_ARRIVED)(ST_DEVICE_SHADOW*
sub_device_shadow);
```

   函数功能：

   此函数声明用于接收平台设置子设备的影子。

 版权所有 © 华为云计算技术有限公司

表 4-53 参数说明

参数名称	类型	参数描述	示例
sub_device_property_get	ST_DEVICE_SHADOW*	属性设置	参见edge.h

11. 自定义topic消息通知回调

函数描述：

```
typedef  EDGE_RETCODE (FN_CUSTOMIZED_MESSAGE_ARRIVED)(const char* topic, const char* payload, unsigned int len);
```

函数功能：

此函数声明用于接收平台设置子设备的影子。

表 4-54 参数说明

参数名称	类型	参数描述
topic	char*	自定义topic
payload	char*	消息内容
len	unsigned int	消息长度

edge.h 的接口函数介绍

所有接口函数定义的数据结构均呈现在edge_struct.h头文件中。

1. 初始化

接口描述：

```
int  edge_init(const char* workdir)
```

接口功能：

初始化sdk的工作环境，加载证书、读取配置等

表 4-55 参数说明

参数名称	类型	参数描述	示例
workdir	workdir	初始化文件所在目录，conf目录所在的目录，conf是存放证书文件以及日志配置的目录	conf在/code/api_test/workdir目录下，就填写/code/api_test/workdir

2. 注册回调接口

版权所有 © 华为云计算技术有限公司

接口描述：

```
int edge_set_callbacks(ST_MODULE_CBS* module_cbs, ST_DEVICE_CBS* device_cbs)
```

接口功能：

注册回调接口，登录、子设备添加、删除、设备消息、设备命令等都是异步通知的，故需要注册对应的处理函数。

表 4-56 参数说明

参数名称	类型	参数描述
module_cbs	ST_MODULE_CBS	模块回调函数结构体
device_cbs	ST_DEVICE_CBS	设备回调函数结构体

表 4-57 ST_MODULE_CBS 模块相关的回调函数说明

参数名称	类型	参数描述
pfn_shadow_cb	FN_SHADOW_ARRIVED*	模块影子回调函数类型，模块的配置可以使该接口接收
pfn_command_cb	FN_COMMAND_ARRIVED*	发送到模块的命令的接收函数声明，
pfn_connected	FN_SDK_CONNECTED	SDK连接到边缘hub的回调函数声明
pfn_disconnected	FN_SDK_DISCONNECTED	SDK和边缘hub断链的回调函数声明

表 4-58 ST_DEVICE_CBS 子设备相关的回调函数

参数名称	类型	参数描述
pfn_device_message_cb	FN_MESSAGE_ARRIVED*	子设备消息回调
pfn_device_command_cb	FN_COMMAND_ARRIVED*	子设备命令回调
pfn_device_event_cb	FN_DEVICE_EVENT_ARRIVED*	子设备事件回调
pfn_sub_device_add_cb	FN_SUB_DEVICE_ADD_ARRIVED*	子设备添加通知回调
pfn_sub_device_deleted_cb	FN_SUB_DEVICE_DELETED_ARRIVED*	子设备删除通知回调
pfn_on_start_scan_cb	FN_START_SCAN_ARRIVED*	收到子设备扫描通知回调

参数名称	类型	参数描述
pfn_device_properties_set_cb	FN_DEVICE_PROPERTIES_SET_ARRIVED*	收到子设备属性设置通知回调
pfn_device_properties_get_cb	FN_DEVICE_PROPERTIES_GET_ARRIVED*	收到子设备属性获取通知回调
pfn_device_shadow_cb	FN_DEVICE_SHADOW_ARRIVED*	收到子设备影子

3. 登录

接口描述：

```
int edge_login()
```

接口功能：

用于SDK登录边缘，该函数是异步函数，登录成功会调用4.2章节中已注册的pfn_connected回调函数。

4. 登出

接口描述：

```
void edge_logout()
```

接口功能：

用于SDK登出边缘

5. 清理

接口描述：

```
void edge_destroy()
```

接口功能：

用于清理SDK资源。

6. 网关添加子设备

接口描述：

```
int edge_send_add_sub_device(ST_DEVICE_INFO* device_info, unsigned int size)
```

接口功能：

网关添加子设备，可批量添加，device_info为ST_DEVICE_INFO数组，size为数组的大小（个数）；

表 4-59 参数描述

参数名称	类型	参数描述	示例
device_info	ST_DEVICE_INFO	设备信息	参见edge.h
size	Int	数组个数	-

备注：

该接口的调用成功不代表添加成功，仅仅代表添加子设备的请求发送到hub，设备的添加成功与否需要edge.h中注册的设备回调接口pfn_sub_device_add_cb。

样例：

```
[{
"parent_device_id": "c6b39067b0325db34663d3ef421a42f6_12345678",
"node_id": "subdevice11",
"device_id": "2bb4ddba-fb56-4566-8577-063ad2f5a6cc",
"name": "subDevice11",
"description": null,
"product_id": "c6b39067b0325db34663d3ef421a42f6",
"fw_version": null,
"sw_version": null,
"status": "ONLINE"
}]
```

7. 网关删除子设备

接口描述：

```
int edge_send_delete_sub_device (ST_DEVICE_INFO* device_info, unsigned int size)
```

接口功能：

删除子设备，可批量删除，device_info为ST_DEVICE_INFO数组，size为数组的大小（个数）；

表 4-60 参数描述

参数名称	类型	参数描述	示例
device_info	ST_DEVICE_INFO	设备信息	-
size	Int	数组个数	-

备注：

该接口的调用成功不代表删除成功，仅仅代表删除子设备的请求发送到hub，设备的删除成功与否需要edge.h中注册的设备回调接口pfn_sub_device_deleted_cb。

样例：

```
[{
"parent_device_id": "c6b39067b0325db34663d3ef421a42f6_12345678",
"node_id": "subdevice11",
"device_id": "2bb4ddba-fb56-4566-8577-063ad2f5a6cc",
"name": "subDevice11",
"description": null,
```


版权所有 © 华为云计算技术有限公司

```
"product_id": "c6b39067b0325db34663d3ef421a42f6",
"fw_version": null,
"sw_version": null,
"status": "ONLINE"
}]
```

8. 设备数据批量上报v1

接口描述:

```
int edge_send_batch_device_data(ST_DEVICE_SERVICE* devices,unsigned int size)
```

接口功能:

批量上报子设备数据，可以同时上报多个设备的多个servicce数据，devices为ST_DEVICE_SERVICE数组，size为数组的个数;

表 4-61 参数描述

参数名称	类型	参数描述
devices	ST_DEVICE_SERVICE	设备数据
size	Int	数组个数

样例:

ST_DEVICE_SERVICE样式

```
{
"device_id":"bf40f0c4-4022-41c6-a201-c5133122054a",
"services":[
{
"service_id":"analog",
"properties":{
"PhV_phsA":"1",
"PhV_phsB":"2"
},
"event_time":"20190606T121212Z"
}
]
}
```

9. 设备数据批量上报v2（单设备单serviceId上报）

接口描述:

```
int edge_send_batch_device_data_v2(const char *device_id, const char *service_id, const char
*service_properties, const char *event_time)
```

接口功能:

该接口主要用于上报单个设备单服务的属性数据;

表 4-62 参数描述

参数名称	类型	参数描述	示例
device_id	char*	设备ID	-
service_id	char*	服务Id	-

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述	示例
properties	char*	属性值（Json字符串）	-
event_time	char*	时间戳	"20190606T121212Z"

10. 设备数据批量上报v3（自行构造json体上报）

接口描述：

```
int edge_send_batch_device_data_v3(const char *service_properties, unsigned int size)
```

接口功能：

上报子设备数据，自行构造json数据；

表 4-63 参数描述

参数名称	类型	参数描述	示例
service_properties	Json字符串	设备数据	-
size	Int	长度	-

样例：

service_properties数据格式如下：

```
[{
"device_id":"bf40f0c4-4022-41c6-a201-c5133122054a",
"services":[
{
"service_id":"test",
"properties":{
"PhV_phsA":"1",
"PhV_phsB":"2"
},
"event_time":"20190606T121212Z"
}
]
}]
```

11. 设备命令响应

接口描述：

```
int  edge_send_command_rsp(const char* rsp_name, const char* request_id, int result_code,
const char* rsp_body, unsigned int body_lens);
```

接口功能：

用于响应平台下发的设备命令，该接口和"命令下发回调"搭配使用。

 版权所有 © 华为云计算技术有限公司

表 4-64 参数描述

参数名称	类型	参数描述	示例
request_id	char*	长度	{request_id}用于唯一标识这次请求,匹配下发的命令，
rsp_name	char*	命令响应名称	命令的响应名称，在设备关联的产品模型中定义
result_code	char*	命令响应码	标识命令的执行结果，0表示成功，其他表示失败。不带默认认为成功。
rsp_body	char*	响应参数	命令的响应参数，具体字段在设备关联的产品模型中定义。
body_lens	unsigned int	rsp_body长度	-

12. 获取子设备影子

接口描述：

```
int edge_send_get_sub_device_shadow(const char* request_id, ST_DEVICE_SHADOW_GET* device_shadow_get);
```

接口功能：

用于网关发送请求给平台获取子设备影子，该接口和"子设备影子回调"搭配使用，请求后将通过"子设备影子回调"的回调函数通知到网关。

表 4-65 参数描述

参数名称	类型	参数描述	示例
request_id	char*	长度	{request_id}用于唯一标识这次请求,
device_shadow_get	ST_DEVICE_SHADOW_GET*	获取影子结构	具体示例参见edge.h

13. 发送子设备消息

接口描述：

```
int  edge_send_sub_device_message(ST_DEVICE_MESSAGE* device_message);
```

接口功能：

用于网关给平台发送子设备消息；

表 4-66 参数描述

参数名称	类型	参数描述	示例
device_message	ST_DEVICE_MESSAGE*	设备消息格式	具体示例参见edge.h

14. 发送子设备事件

接口描述：

```
int  edge_send_sub_device_event(ST_DEVICE_EVENT* device_event);
```

接口功能：

用于网关给平台发送子设备事件；

表 4-67 参数描述

参数名称	类型	参数描述	示例
device_event	ST_DEVICE_EVENT*	设备事件格式	具体示例参见edge.h

15. 发送获取产品请求

接口描述：

```
int edge_send_get_product(const char** product_ids, unsigned int product_id_size);
```

接口功能：

用于网关给平台发送获取产品的请求；

表 4-68 参数描述

参数名称	类型	参数描述	示例
product_ids	char**	产品id的数组列表	["productId1", "productId2",]
product_id_size	unsigned int	数组长度	-

16. 发送子设备属性获取响应

接口描述：

```
int edge_send_sub_device_property_get_rsp(const char* request_id,
ST_DEVICE_PROPERTY_GET_RSP* device_property_get_rsp);
```


版权所有 © 华为云计算技术有限公司

接口功能：

用于网关响应平台的子设备属性获取请求,和"子设备属性获取回调"回调请求搭配子设备属性获取回调子设备属性获取回调使用。

表 4-69 参数描述

参数名称	类型	参数描述	示例
request_id	char*	请求id	{request_id}用于唯一标识这次请求,
device_property_get_rsp	ST_DEVICE_PROPERTY_GET_RS子设备属性获取回调P*	属性获取响应	参见edge.h

17. 发送子设备属性设置响应

接口描述：

```
int edge_send_sub_device_property_set_rsp(const char* request_id, ST_IOT_RESULT* iot_result);
```

接口功能：子设备属性设置回调

用于网关响应平台的子设备属性设置请求,和"子设备属性获取回调"回调请求搭配使用。

表 4-70 参数描述

参数名称	类型	参数描述	示例
request_id	char*	请求id	{request_id}用于唯一标识这次请求,
iot_result	ST_IOT_RESULT*	属性设置响应	参见edge.h子设备属性获取回调

18. 发送自定义topic消息

接口描述：

```
int edge_send_customized_message(const char* topic, const char* body, unsigned int body_len);
```

接口功能：

用于网关发送自定义topic消息。

表 4-71 参数描述

参数名称	类型	参数描述
topic	char*	自定义topic
body	char*	消息体

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述
body_len	unsigned int	消息长度

图 4-11 图示

解释：

用户在调用设备接入的自定义消息时，注册的自定义topic需要附加上/modules/{moduleId}/前缀。

19. 设置发送总线消息回调

接口描述：

```
int edge_set_bus_message_cb(const char* input_name);
```

接口功能

为消息总线输入点设置回调函数。

表 4-72 参数描述

参数名称	类型	参数描述	示例
input_name	char*	消息总线输入点	见Demo

20. 发送总线消息

接口描述：

```
edge_send_bus_message(const char* output_name, const char* body, unsigned int body_len);
```

接口功能：

向消息总线输出点发送数据。

表 4-73 参数描述

参数名称	类型	参数描述
output_name	char*	消息总线输出点
body	char*	消息体
body_len	unsigned int	消息长度

21. 调用设备命令

接口描述

```
int   edge_call_device_command(ST_COMMAND* command, unsigned int timeout);
```

接口功能:

调用产品模型里定义的命令。

表 4-74 参数描述

参数名称	类型	参数描述	示例

command	ST_COMMAND*	设备命令	{
			"device_123",
			"power",
			"restart"
			}

timeout	unsigned int	超时参数(以秒为单位)	-

22. 设置发送模式

接口描述

```
int edge_set_send_mode(EN_SEND_MODE mode);
```

接口功能

设置点位上报、属性上报的发送模式，有同步、异步两种可选。

表 4-75 参数描述

参数名称	类型	参数描述	示例
mode	EN_SEND_MODE	上报模式	SEND_ASYNC

 版权所有 © 华为云计算技术有限公司

注意

此设置只影响点位上报和属性上报。默认为异步发送，若您设置为同步后，所有的点位上报、属性上报发送请求都会调整为同步发送（最多阻塞3秒），超时则返回非0错误码。

23. 设置是否在Hub离线缓存满时禁止发送

接口描述

```
int edge_forbid_send_when_broker_overloaded(_BOOL forbidden);
```

接口功能

当SDK感知到Hub的离线缓存已满时，是否继续发送数据。

表 4-76 参数描述

参数名称	类型	参数描述	示例
forbidden	_BOOL	是否禁止	_TRUE

注意

此设置只影响点位上报和属性上报。默认为否（不禁止发送），若您设置为是，所有的点位上报、属性上报发送请求都会在Hub离线缓存满时被禁止，并返回非0错误码。

24. 获取子设备状态

接口描述

```
int edge_get_devices_statuses(ST_DEVICES_STATUSES* devices_statuses, const char** devices_ids, unsigned int device_id_size, unsigned int timeout
```

接口功能

用于获取指定的子设备的状态

表 4-77 参数描述

参数名称	类型	参数描述	示例
devices_statuses	ST_DEVICES_STATUSES*	用于存放查询结果的结构体	见 25. 初始化子设备状态结构体
devices_ids	const char**	存放要查询的子设备id（数组）	const char* devices_ids[2] = {0}; devices_ids[0] = "dev_id1"; devices_ids[1] = "dev_ids2";

 版权所有 © 华为云计算技术有限公司

参数名称	类型	参数描述	示例
device_id_size	unsigned int	上述数组的大小	2
timeout	unsigned int	超时，单位为毫秒，范围为1ms~30000ms	3000

表 4-78 ST_DEVICES_STATUSES 结构体描述

参数名称	类型	参数描述
devices	ST_DEVICE_STATUS*	设备状态
size	Int	数组个数

表 4-79 ST_DEVICE_STATUS 结构体描述

参数名称	类型	参数描述
device_id	char*	设备ID
status	char*	设备状态

注意

ST_DEVICES_STATUSES参数必须使用特定的接口进行初始化和释放操作，具体见25和 26

25. 初始化子设备状态结构体

接口描述

```
int edge_init_devices_statuses(ST_DEVICES_STATUSES* devices_statuses, unsigned int size);
```

接口功能

用于存放子设备状态查询结果的结构体，需要使用该接口进行初始化后，才能传递给edge_get_devices_statuses

参数描述

版权所有 © 华为云计算技术有限公司

表 4-80 参数描述

参数名称	类型	参数描述	示例
devices_statuses	ST_DEVICES_STATUSES*	指向该结构体的指针	ST_DEVICES_STATUSES devices_statuses = {0}; edge_init_devices_statuses(&devices_statuses, 2);
size	unsigned int	要查询的子设备数，与24. 中的devices_id_size一致	2

注意

在调用子设备查询接口前，请务必使用本接口对ST_DEVICES_STATUSES进行初始化操作，查询结束后，必须使用对应的释放接口对资源进行释放。（具体见26.）

26. 释放子设备状态结构体

接口描述

```
int edge_release_devices_statuses(ST_DEVICES_STATUSES* devices_statuses);
```

接口功能

用于释放子设备状态结构体所占有的资源

表 4-81 参数描述

参数名称	类型	参数描述	示例
devices_statuses	ST_DEVICES_STATUSES*	指向该结构体的指针	edge_release_devices_statuses(&devices_statuses);

注意

在调用edge_get_devices_statuses完成相关操作后，请及时使用该接口对资源进行释放，避免出现内存泄漏的问题。

27. 设备数据批量上报（支持子设备发现事件）

接口描述

```
int edge_send_batch_device_discovery_data(ST_DEVICE_DISCOVERY_SERVICE* devices, unsigned int size);
```

接口功能：

上报子设备数据，自行构造json体上报，支持在上报的数据中，携带product_id、device_name，当发现设备不存在时则上报子设备发现事件

表 4-82 参数描述

参数名称	类型	参数描述	示例
devices	ST_DEVICES_DISCOVERY_SERVICE*	设备数据	/
size	unsigned int	长度	/

表 4-83 ST_DEVICES_DISCOVERY_SERVICE 结构体描述

参数名称	类型	参数描述
device_id	char*	设备ID
product_id	char*	产品ID
device_name	char*	设备名
services	ST_SERVICE_DATA*	子设备数据
size	unsigned int	数组长度

表 4-84 ST_SERVICE_DATA 结构体描述

参数名称	类型	参数描述
service_id	char*	服务ID
properties	char*	属性值（Json字符串）
event_time	char*	时间戳

4.4.2.2 前提条件

- 开发环境要求：安装cmake（版本要求为 3.9.5以上）。
- 开发工具：**CLion** 或者 **Visual Studio Code** 。

4.4.2.3 创建工程

下面以 CLion 作为项目开发IDE。

步骤1 打开CLion->File->New Project->选择新建“C Executable”工程。

这里创建一个名为“MyCApp”的工程进行下面的实践。

步骤2 下载C语言版本SDK，将文件解压缩之后，复制到新创建的项目下。

说明

C语言版本SDK一共支持三个版本，分别是

ModuleSDK_C_latest(包括x86_64, arm32, arm64版本,下载后解压选择对应版本)。

步骤3 下载SDK Demo。

参考**下载Demo**

步骤4 修改CMakeLists.txt文件，添加如下行。

注意

最后三行需保持顺序一致，以免编译出错。

```
cmake_minimum_required(VERSION 3.9.5)
project(MyCApp C)

set(CMAKE_C_STANDARD 99)

link_directories(./lib)

add_executable(MyCApp main.c)

target_link_libraries(MyCApp module)
```

----结束

4.4.2.4 生成可执行文件

说明

需要Linux开发环境，可利用CLion远程调试或者将工程打包到Linux服务器上编译生成可执行文件

编译生成可执行文件(MyCApp)，可以通过CLion之间编译生成，也可以通过在Linux服务器上通过命令行生成。

下面提供在Linux服务器使用命令行生成的方法。

1.
```
cmake -DCMAKE_BUILD_TYPE=Debug -G "CodeBlocks - Unix Makefiles" ./
-- The C compiler identification is GNU 4.8.5
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/MyCApp
```

2.
```
make
Consolidate compiler generated dependencies of target MyCApp
[100%] Built target MyCApp
```
可以在目录下找到生成的可执行文件，即MyCApp。

 版权所有 © 华为云计算技术有限公司

4.4.2.5 制作镜像包或插件包

注意

若制作镜像包以容器化方式部署应用，不支持在一个容器内运行多个集成ModuleSDK的软件进程或者重启集成ModuleSDK的软件进程，会导致鉴权失败等问题。

镜像包打包

步骤1 上传需要打包的项目。将项目上传到网络能访问到的Linux机器上，放到目录下（比如 /home/MyCApp）。

步骤2 安装docker。

请确认使用的系统已经安装docker(docker版本需要高于17.06，推荐18.06)，安装方法可参照**docker 安装教程**。

步骤3 制作镜像。

1. 搜索基础镜像。

   说明

   对基础镜像没有要求，可自行选择合适的基础镜像，以下选用带cmake的镜像作为示例

   ```
   docker search cmake
   ```

2. 选择合适的镜像（镜像需要集成了cmake且版本不低于3.9.5）

   ```
   NAME                DESCRIPTION                  STARS     OFFICIAL   AUTOMATED
   lycantropos/cmake   CMake Docker image           3                    [OK]
   ```

3. 拉取镜像

   说明

   lycantropos/cmake镜像是docker hub第三方提供的镜像，非IoT团队发布，且IoT团队未提供任何官方镜像。该镜像在此仅做示例，IoT团队对该镜像的安全性不作保证。强烈建议用户自己封装镜像！

   ```
   docker pull lycantropos/cmake
   ```

4. 添加启动脚本start.sh，放到项目文件下（和main.c在一个目录下）。

   ```
   function log(){
      echo `date "+%Y-%m-%d %T"`: $1
   }

   log "[INFO] start execute process."

   # 这里的路径取决于项目保存的位置
   cd /opt/iot/edge/MyCApp

   ./MyCApp
   ```

   说明

   MyCApp为可执行文件，生成步骤可参考**生成可执行文件**

5. 编写 Dockerfile 制作镜像

   创建 Dockerfile，命名为 myapp-dockerfile 内容参照如下（具体可参考**编写高效的Dockerfile**）。

版权所有 © 华为云计算技术有限公司

须知

下面提供myapp-dockerfile 样例，请根据具体需要修改。
注意myapp-dockerfile需要和项目放到一个目录下。

```
#Version 1.0.0

# 基础镜像来源
# 如果不采用在镜像中编译源文件的方式，可以任意选择基础镜像
FROM lycantropos/cmake

# 创建镜像文件目录，并且授权
RUN mkdir -p /opt/iot/edge/MyCApp/conf && chmod -R 777 /opt

ENV docker_path=/opt/iot/edge/MyCApp
ENV LD_LIBRARY_PATH=${docker_path}:$LD_LIBRARY_PATH

WORKDIR ${docker_path}

# 复制工程或文件到指定目录
COPY MyCApp/MyCApp $docker_path

# 将依赖库和配置文件放到对应目录下，确保编译的时候不会出错
COPY MyCApp/lib /usr/lib
COPY MyCApp/conf $docker_path/conf
COPY MyCApp/start.sh $docker_path

# 指定容器将要监听的端口
USER root
EXPOSE 8082

ENTRYPOINT ["/bin/bash", "/opt/iot/edge/MyCApp/start.sh"]
```

6. 构建镜像
```
docker build -f ./myapp-dockerfile -t my_app_docker:v1.0.0 ./
```
7. 查看打包完成的镜像
```
docker images
```
可以看到**my_app_docker**这个镜像已经制作完成。
```
REPOSITORY      TAG         IMAGE ID       CREATED          SIZE
my_app_docker   v1.0.0      983b4e5aa72a   10 minutes ago   1.51GB
```

步骤4 镜像上传

须知

以上步骤可以通过**体验馆**熟悉流程。

1. 上传镜像

   镜像上传需要使用镜像容器服务(SWR)，首先需要开通容器器镜像服务(SWR)。开通及使用请参照**容器镜像服务(SWR)**。
2. 获取 SWR 登录指令

   获取登录指令请参照**获取指令**。

   说明

   访问密钥即AK/SK（Access Key ID/Secret Access Key），获取的密钥和AK将用于登录。
3. 登录 SWR 仓库
```
docker login -u [区域项目名]@[AK] -p [登录密钥] [镜像仓库地址]
```

可以直接从控制台获取登录命令，如下图。

📖 **说明**

镜像仓库地址 = swr.区域项目名称.myhuaweicloud.com

例如，华北-北京一对应的镜像仓库地址为：swr.cn-north-1.myhuaweicloud.com

4. 修改镜像所属组织

   修改镜像的组织名，以便推送到个人组织内。

   docker tag [OPTIONS] [镜像名:版本号] [镜像仓库地址/所属组织/镜像名:版本号]

   例如，

```
docker tag my_app_docker:v1.0.0 swr.cn-north-4.myhuaweicloud.com/iotedge/
my_app_docker:v1.0.0
```

5. 上传镜像

```
docker push [镜像仓库地址/所属组织/镜像名:版本号]
```

   例如，

```
docker push swr.cn-north-4.myhuaweicloud.com/iotedge/my_app_docker:v1.0.0
```

6. 在我的镜像查看上传结果

7. 上传镜像后请在 SWR 将镜像设置为公开。

   步骤一：

   步骤二：

   步骤三：

⚠ **注意**

这一步很重要，关系到后面能否正常部署应用。

----结束

插件包打包

步骤1 插件包制作。

1. 插件包格式要求如下：

   插件包仅支持.tar.gz 、.tar或者 .zip格式。

   插件包结构如下：

```
monitor-app.zip
├── MyCApp // 可执行文件
└── stop.sh //停止脚本 非必须
│ ├─lib
│ │ ├─libmodule.so //sdk库文件
│ │ └─***
│ ├─conf
│ │ ├─sdk_log.conf //sdk日志配置
│ │ ├─module.dat //sdk需要的密钥
│ │ ├─rootcert.pem //证书
└────start.sh //启动脚本 必须文件 当前不提供参数方式启动
```

   说明

   MyCApp为可执行文件，生成步骤可参考生成可执行文件

   注意

   打包的文件至少需要包括lib和conf两个目录（目录下所有文件），加启动脚本start.sh和可执行文件 MyCApp 。

 版权所有 © 华为云计算技术有限公司

2. 添加启动脚本

在可执行文件MyCApp的同一目录下创建启动脚本start.sh，内容如下

```
log(){
    echo `date "+%Y-%m-%d %T"`: $1
}

log "[INFO] start execute process."

# 指定添加链接库
export LD_LIBRARY_PATH=$PWD/lib/:$LD_LIBRARY_PATH

# 修改执行权限
chmod 755 MyCApp
./MyCApp > myapp_running.log 2>&1
```

说明

通过安装包部署的文件会放在/var/IoTEdge/downloaded-job/run下面，包括生成的日志文件myapp_running.log。

3. 构建插件包。

压缩文件得到monitor-app.zip。

步骤2 插件包上传。

1. 开通对象存储服务OBS。

进程包上传方式需要开通对象存储服务OBS，请参考**对象存储服务 OBS_快速入门**

2. 上传进程包。

上传方式请参照**对象存储服务(OBS)**。

注意

请设置桶策略为【公开读】，如未设置请前往OBS首页->单击桶ID->访问权限控制->桶策略中设置。

----结束

4.4.2.6 添加应用

添加边缘应用具体请参考添加应用。

说明

提供两种不同的部署方式，请根据需要选择合适的部署方式。

容器化部署

步骤1 部署方式选择”**容器化部署**”。

说明

建议直接勾选“立即发布”，方便后面直接部署应用的时候，能够获取到最新版本。

步骤2 软件和运行配置

选择上传到SWR服务的镜像，如未发现镜像，请检查镜像是否为公开，设置镜像为公开方式：容器镜像服务SWR->我的镜像->单击镜像ID进入详情->右上方编辑。

版权所有 © 华为云计算技术有限公司

步骤3 添加边缘应用-端点和部署配置

步骤4 根据需要进行配置。

输入端点、输出端点与demo中代码定义的端点对应，如myapp中输入与输出端点设置为input和output,则配置为：

输入端点：input

 版权所有 © 华为云计算技术有限公司

输出端点：output

----结束

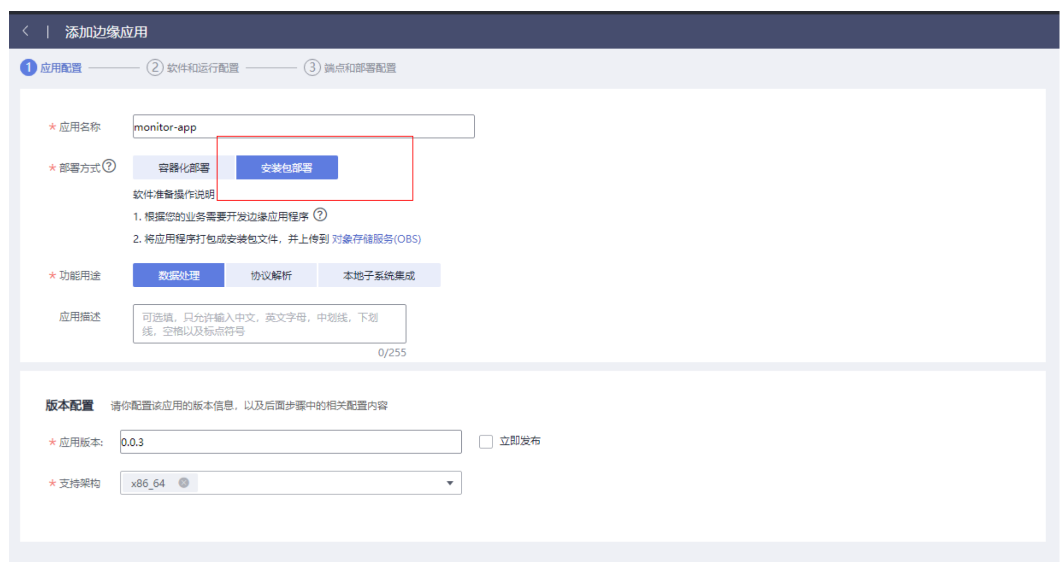
说明

输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

安装包部署

步骤1 添加边缘应用-应用配置

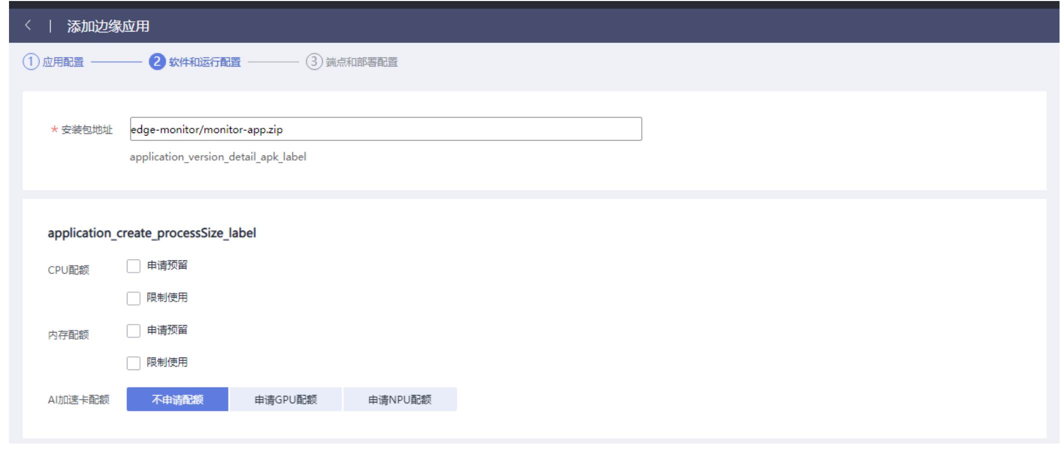
部署方式选择安装包部署



步骤2 添加边缘应用-软件和运行配置

”安装包地址”为{桶名/对象名}。

如桶名为edge-monitor，对象名为monitor-app.zip， 则安装包为edge-monitor/monitor-app.zip。



步骤3 添加边缘应用-端点和部署配置同容器化部署

版权所有 © 华为云计算技术有限公司

根据需要进行配置。

输入端点输出端点与demo中的设置对应，如monitor-app中输入与输出端点设置为input和output,则配置为：

输入端点：input。

输出端点：output。

说明

输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。

驱动类应用和IT应用一般不需要配置。

建议直接勾选“立即发布”，方便后面直接部署应用的时候，能够获取到最新版本。

----结束

4.4.2.7 发布应用

应用创建之后需要发布才允许在节点部署。

操作步骤

步骤1 访问**IoT边缘**，单击**“管理控制台”**进入IoT边缘控制台。

步骤2 选择左侧导航栏“边缘节点 > 应用管理”进入页面，选择“应用名称”进入应用详情页。

步骤3 单击右上角“发布”按钮。

----结束

说明

可在创建应用时可勾选【立即发布】进行发布。

4.4.2.8 如何使用

OT应用使用步骤：

1. 将创建好的应用部署到节点。部署请参照**应用部署**。

 版权所有 © 华为云计算技术有限公司

2. 添加设备进行测试（添加设备请参考设备接入边缘节点）。

> 说明
>
> 可以利用MQTT.fx软件模拟设备接入调试。

驱动应用使用步骤：

1. 将创建好的应用部署到节点。部署请参照应用部署。
2. 添加网关。
3. 添加设备进行测试（添加设备请参考设备接入边缘节点）。

> 说明
>
> 可以利用Modbus Slave软件模拟设备接入调试。

IT应用使用步骤：

1. 注册节点，绑定工业资源包。
2. 创建API网关。
3. 将apigw系统组件和创建的应用部署到节点。
4. 使用postman进行调试验证。

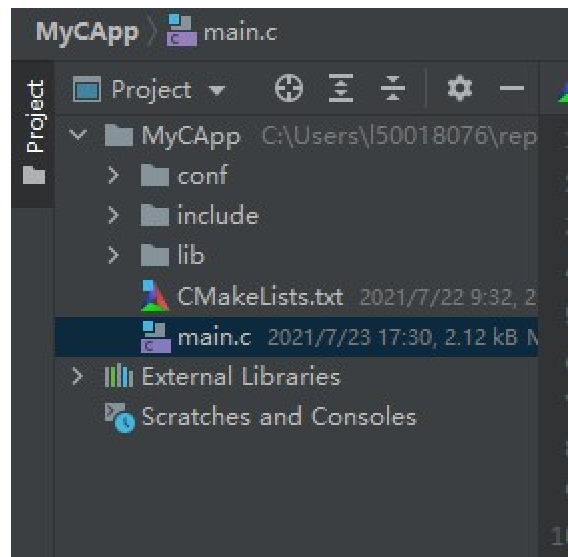
4.4.3 下载 Demo

首先参考创建工程的步骤创建新的工程。

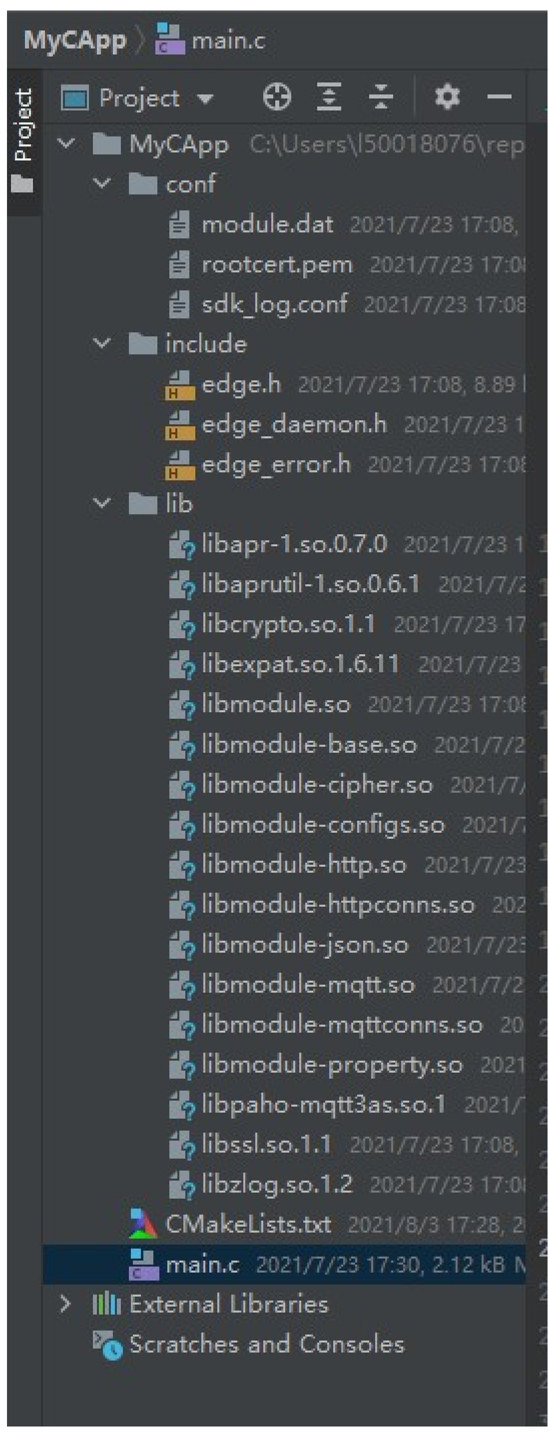
复制**Demo**，解压并覆盖新建项目中的 main.c 文件。

最终的文件树应如下图所示。

```
MyCApp
│ ├─main.c // 主程序
│ ├─lib // sdk库文件
│ │ ├─libmodule.so
│ │ └─***
│ ├─include // 头文件
│ │ ├─edge_error.h
│ │ ├─edge_daemon.h
│ │ └─edge.h
└───conf // 配置文件
│ │ ├─sdk_log.conf //sdk日志配置
│ │ ├─module.dat //sdk需要的密钥
│ │ └─rootcert.pem //证书
└───start.sh // 启动脚本（编写启动脚本可参考制作镜像包或插件包）
```



详细目录树如下图



说明

更多Demo参见**ModuleSDK-C Demo展示**。

4.4.4 集成 ModuleSDK 进行数据处理

版权所有 © 华为云计算技术有限公司

4.4.4.1 场景说明

在节点接入一个电机设备，设备遇到问题上报信息“error”给节点，节点监听到设备的“error”信息，下发命令让设备进行重启。示例工程为MyCApp。

4.4.4.2 代码解析

Demo代码如下，具体实现的是模拟电机设备上报数据，SDK获取上报数据做进一步分析处理。如果遇到状态为error，则调用事先在产品模型定义好的设备命令。对于未指定MOTOR_PRODUCT_ID的产品上报的数据将继续上报给云端。

```
#include "edge.h"

#include <stdio.h>
#include <string.h>
#include <unistd.h>

/*
 * 描述：设置总线消息回调，用于对设备上报的数据进行处理
 * 参数：
 *  input_name：消息总线输入点
 * */
EDGE_RETCODE set_bus_message_cb(const char* input_name)
{
    edge_set_bus_message_cb(input_name);
    printf("set bus message callback with input name: %s\n", input_name);

    return EDGE_SUCCESS;
}

/*
 * 描述：收到设备上报数据的回调处理，样例代码在马达设备状态错误时对马达进行重启
 * 参数：
 *  device_id：设备ID
 *  product_id: 产品ID
 *  body: 上报的数据
 *  body_len: 上报数据的大小
 * */
EDGE_RETCODE on_message_received_cb(const char* device_id, const char* product_id, const char* body,
unsigned int body_len)
{
    // 设置发送设备数据的消息总线输出点，取值需在创建应用版本的outputs参数中定义
    char* output_name = "output";

    printf("start send message to output topic: %s\n", output_name);
    printf("body is: %s\nbody len is: %d\n", body, body_len);
    printf("product_id is: %s\n", product_id);
    printf("start processing device.\n");

    // 设置电机设备产品ID
    char* MOTOR_PRODUCT_ID = "product_123";

    if (strcmp(product_id, MOTOR_PRODUCT_ID) == 0)
    {
        //马达设备状态错误时对马达进行重启
        char* error = "error";
        char* is_error = strstr(body, error);

        // 设置默认超时时间
        unsigned int timeout = 5;
        ST_COMMAND command = {0};
        command.object_device_id = device_id;
        command.service_id = "power";
        command.command_name = "restart";

        // 调用设备命令重启
        if (is_error != NULL) edge_call_device_command(&command, timeout);
```

```
    }
    else {
        //其他设备数据发送到消息总线
        edge_send_bus_message(output_name, body, body_len);
    }

    printf("process ended.\n");

    return EDGE_SUCCESS;
}

/*
 * 监控APP，检视设备上报的数据，并对设备进行相应的控制
 */

void monitor_app()
{
    // 禁用缓冲区
    setvbuf(stdout, NULL, _IONBF, 0);

    printf("start monitor app\n");

    //初始化sdk，工作路径设置（工作路径下需要含有 /conf 目录（该目录下包含证书等信息））
    edge_init("../code/api_test/workdir");

    ST_MODULE_CBS module_cbs = {0};
    ST_DEVICE_CBS device_cbs = {0};

    module_cbs.pfn_on_message_received_cb = on_message_received_cb;

    // 设置回调函数
    edge_set_callbacks(&module_cbs, &device_cbs);

    printf("SDK start running!\n");

    sleep(1);
    edge_login();

    sleep(1);

    // 接收设备数据的消息总线输入点，取值需在创建应用版本的inputs参数中定义
    char* input_name = "input";
    set_bus_message_cb(input_name);

    // 这里是为了使应用能够长时间运行
    while(1)
    {
        sleep(1000);
    }

    edge_logout();
    sleep(1000);
    edge_destroy();
}


int main()
{
    // 监控app demo
    monitor_app();

    return 0;
}
```

Demo实现的流程如下：

步骤1 通过edge_init初始化工作目录。

 版权所有 © 华为云计算技术有限公司

步骤2 通过edge_set_callbacks设置回调函数。

Demo中只使用到on_message_received_cb回调函数，只需修改on_message_received_cb即可。

步骤3 通过edge_login初始化SDK，包括连接环境变量，连接Hub，订阅Topic，设置回调。

步骤4 通过set_bus_message_cb调用edge_set_bus_message_cb，SDK会根据input_name订阅Topic(比如/modules/user_monitor_app/messages/inputs/input，这里user_monitor_app是SDK应用对应的模块id，最后的“input“就是Demo代码里的input_name)，这个函数会将on_message_received_cb作为回调函数。

步骤5 回调函数on_message_received_cb里调用edge_send_bus_message，将未处理的数据发送回消息总线，设置该函数里的output_name，边缘Hub会订阅类似/modules/user_monitor_app/messages/outputs/output的Topic（这里user_monitor_app是SDK应用对应的模块id，最后的“output“就是Demo代码里的output_name）。

步骤6 调用设备命令，只有当设置的MOTOR_PRODUCT_ID的当前上报数据的设备的产品ID吻合，并且显示状态为error时，通过edge_call_device_command调用设备命令将设备重启。

步骤7 处理过程结束。

----结束

修改Demo里的参数可参考**修改代码**。

4.4.4.3 注册节点

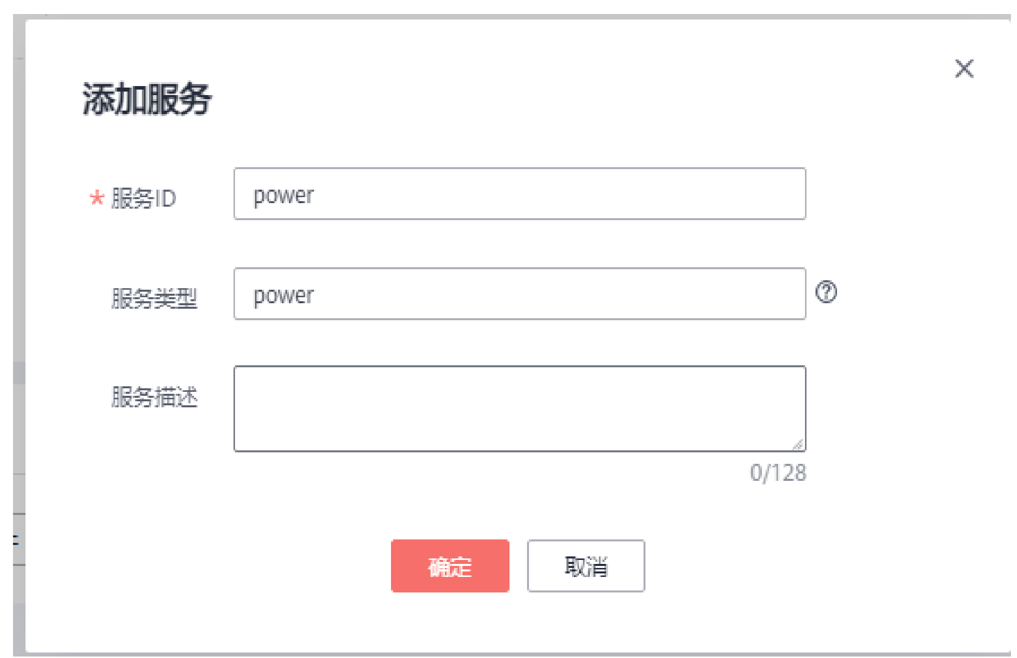
注册节点请参照**注册边缘节点**。

4.4.4.4 创建产品

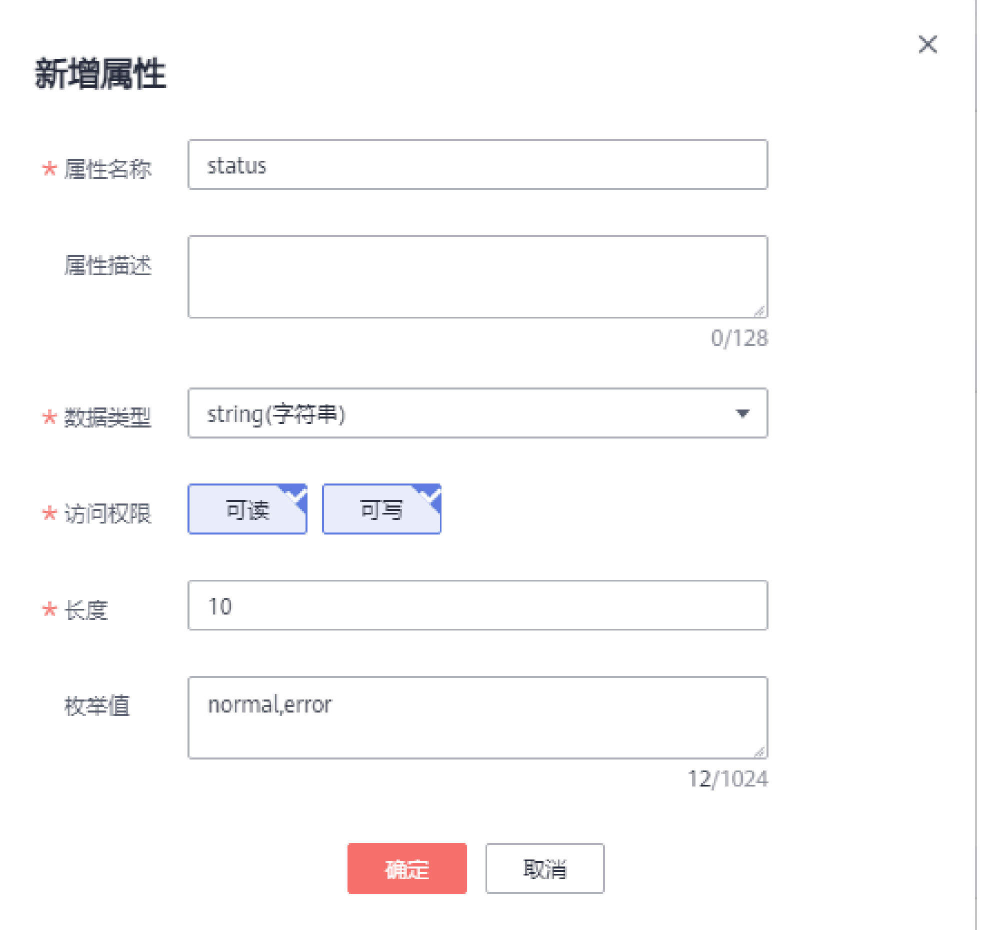
创建产品具体教程参照**创建产品_设备接入 IoTD** ,以下是具体配置中的参照。

1. 创建产品

2. 在新建产品后需要在产品页的”自定义模型”中添加服务。



3. 新增属性



4. 新增命令。

单击确定完成创建。

4.4.4.5 修改代码

1. 如果在编译之前要保证目录树和上一专题里提到的一致
2. 修改头文件引用。

版权所有 © 华为云计算技术有限公司

```
#include "edge.h"
```

改为

```
#include "include/edge.h"
```

也可以根据include头文件所在的路径做相应修改。

3. 修改初始化工作路径

```
edge_init("../code/api_test/workdir");
```

改为

```
edge_init("./");
```

将当前目录指定为工作目录。

4. 修改输入点

这里的input_name必须和**3.添加边缘应用-端点和部署配置**里的“输入端点“保持一致。

```
// 接收设备数据的消息总线输入点，取值需在创建应用版本的inputs参数中定义
char* input_name = "input";
```

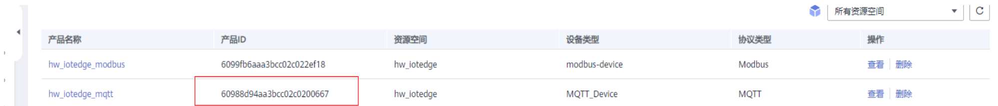
5. 修改输出点

这里的output_name必须和**3.添加边缘应用-端点和部署配置**里的“输出端点“保持一致。

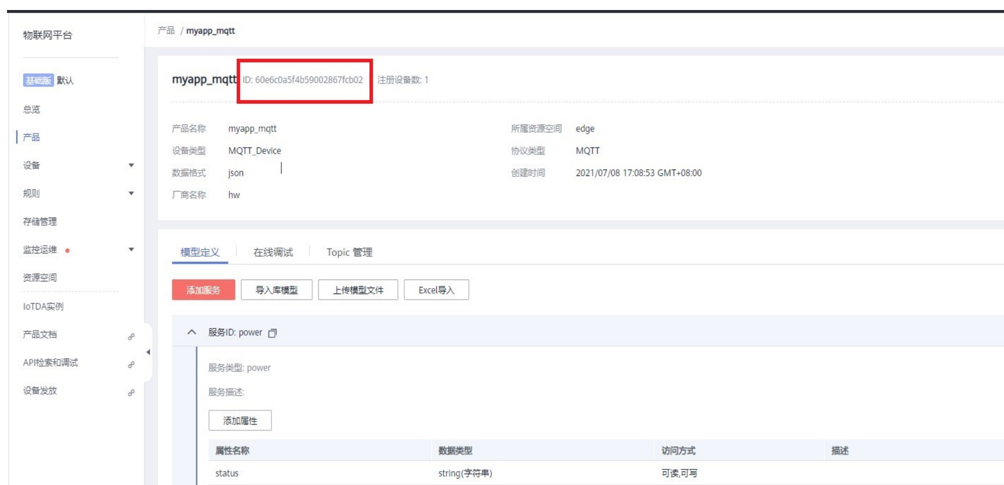
```
// 设置发送设备数据的消息总线输出点，取值需在创建应用版本的outputs参数中定义
char* output_name = "output";
```

6. 修改电机设备的产品ID

查看所创建产品的id，查看方式：IoTDA->产品列表。



或者在产品详情页查看。



根据产品ID修改默认的MOTOR_PRODUCT_ID。

```
// 设置电机设备产品ID
char* MOTOR_PRODUCT_ID = "product_123";
```

修改代码之后，根据**生成可执行文件**进行编译，生成可执行文件。

4.4.4.6 项目打包

按照**下载Demo**里展示的目录树准备好所需文件，并且将SDK应用编译成可执行文件之后，接下来按照下一章节制作镜像包或插件包。

 版权所有 © 华为云计算技术有限公司

4.4.4.7 制作镜像包或插件包

请参照**制作镜像包或插件包**。

4.4.4.8 创建应用

以容器镜像方式为例，镜像包上传到容器镜像服务SWR后。

1. 在IoT边缘单击创建应用

须知

建议直接勾选“立即发布”，方便后面直接部署应用的时候，能够获取到最新版本。

2. 添加边缘应用-软件和运行配置

3. 添加边缘应用-端点和部署配置

4. 单击确定完成创建。

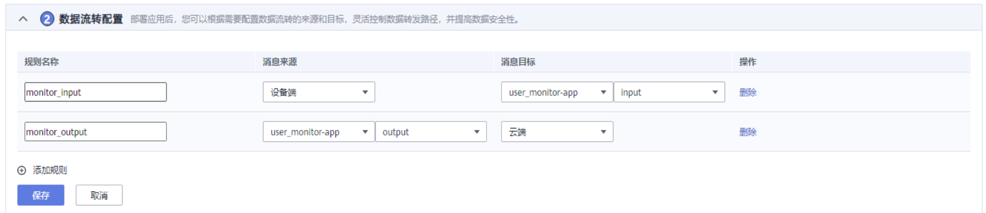
4.4.4.9 部署应用

1. IoT边缘>节点>模块>部署应用，具体参考部署应用。

注意

IT应用需要依赖APIGW，在部署IT应用之前，请先部署系统应用$sys_edge_apigw。

2. 添加流转规则



说明

流转规则是非必选的，OT应用需要添加数据流转规则。驱动应用和IT应用不用添加。

4.4.4.10 添加边缘设备

添加子设备请参考设备接入。

以下是添加边缘设备（MQTT设备）配置时的参考：

提示

接入设备需统一纳入物联网平台管理，并和当前边缘节点归属同一个设备接入服务实例/资源空间。

归属服务实例 IoTDA默认基础版

归属资源空间 edge

* 所属产品 myapp_mqtt

没有可选产品？请前往设备接入服务增加自定义产品，并定义设备功能 前往添加产品

* 设备标识码 myapp_device

* 设备名称 myapp_device

* password ••••••••••

确认 取消

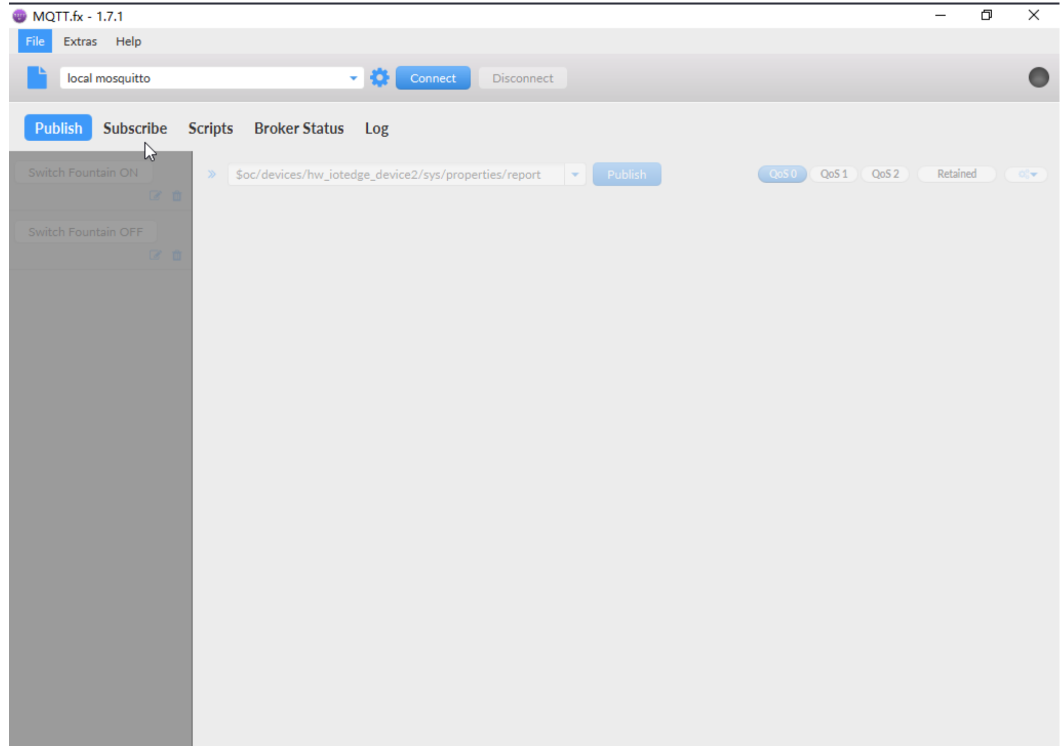
版权所有 © 华为云计算技术有限公司

记住设备ID和密码，用于设备接入平台认证 。

4.4.4.11 设备接入

下载MQTT.fx及证书，**证书下载地址**。

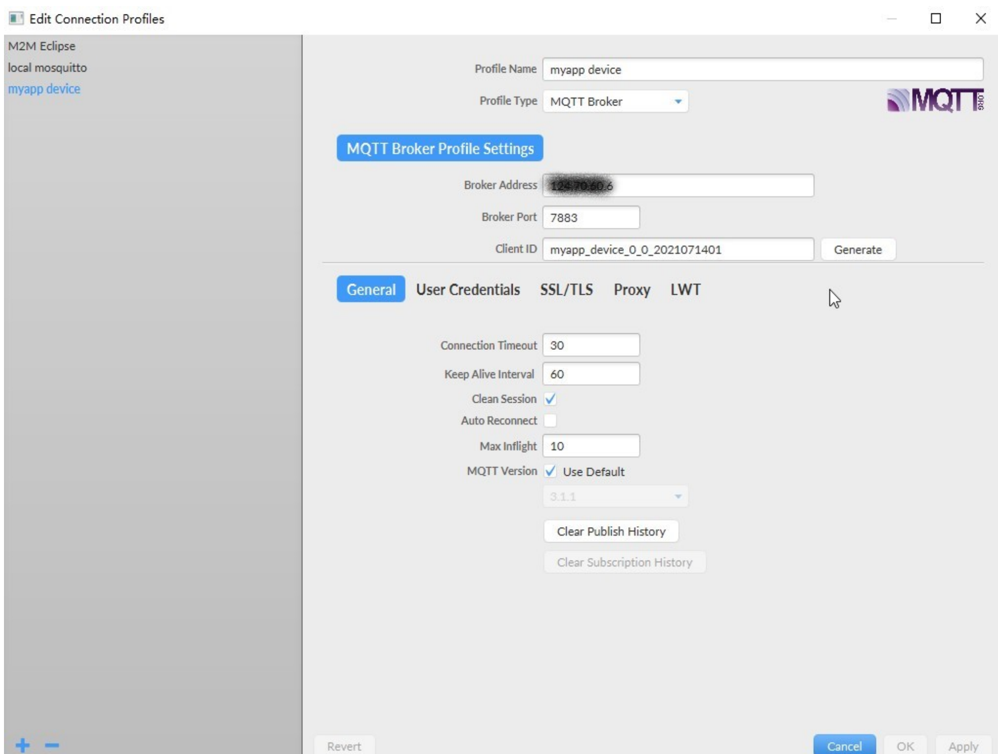
安装完成后打开，MQTT.fx软件界面如下



注：Connect左边的蓝色齿轮为设置。

Publish是消息发送，Subscribe为消息接收，Log可查看日志。

1. 单击设置-General，输入以下信息



Broker Address：输入节点的公网地址。

Broker Port: MQTTS协议使用的端口，默认为7883。

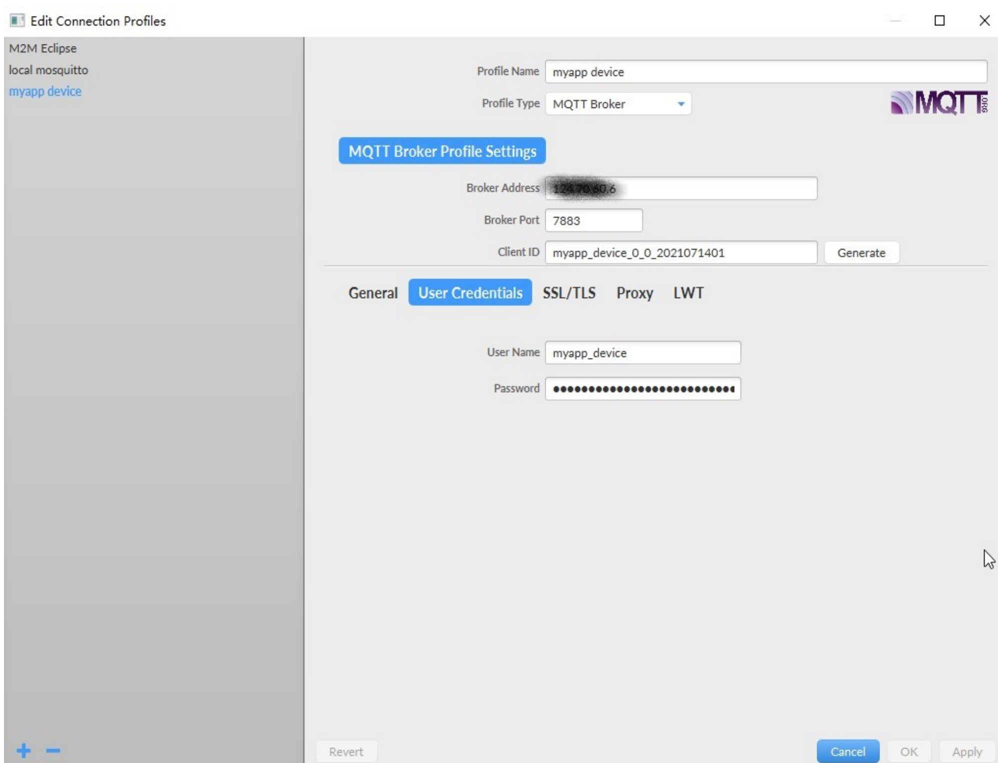
尝试连接时间和保持连接时间等自定义。

2. 单击设置-User Credentials

Client ID和密码需要工具进行转换。利用网页转换工具进行转换。

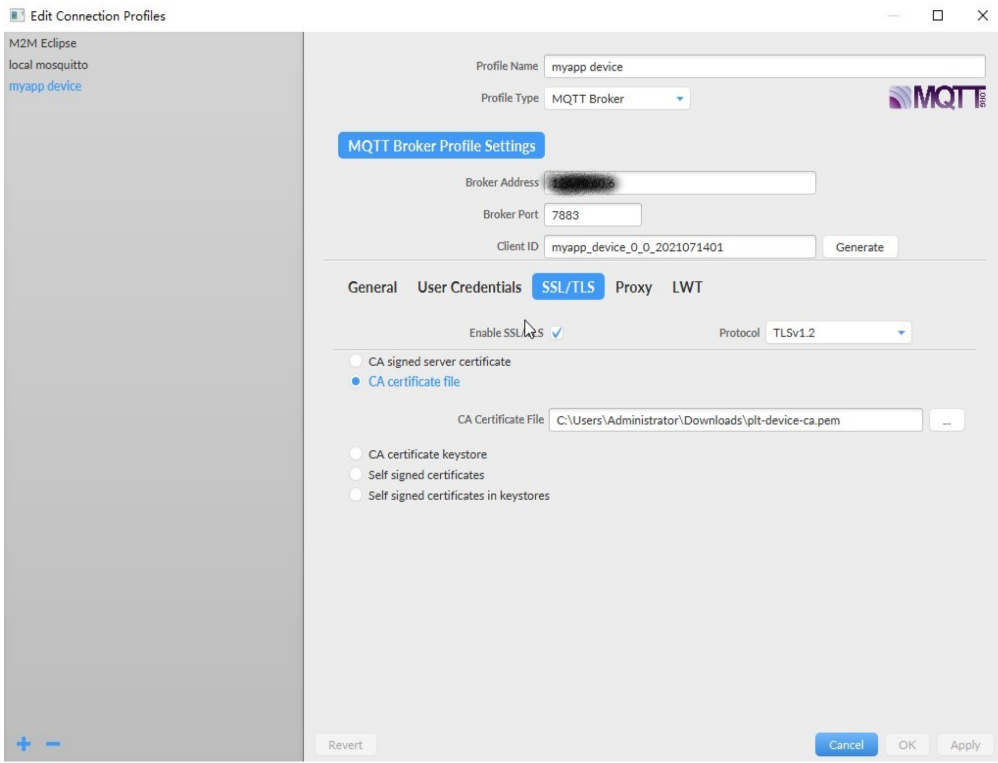
填写添加设备（IoT边缘）后生成的设备ID和设备密钥，生成连接信息（ClientId、Username、Password）。

 版权所有 © 华为云计算技术有限公司



3. 单击设置-SSL/TLS

勾选Enable SSL/TLS，单击CA certificate，选择下载的证书文件。



单击Apply应用设置后返回。

单击Connect连接,连接成功后右边红点会变成绿色，IoTDA也会显示在线。

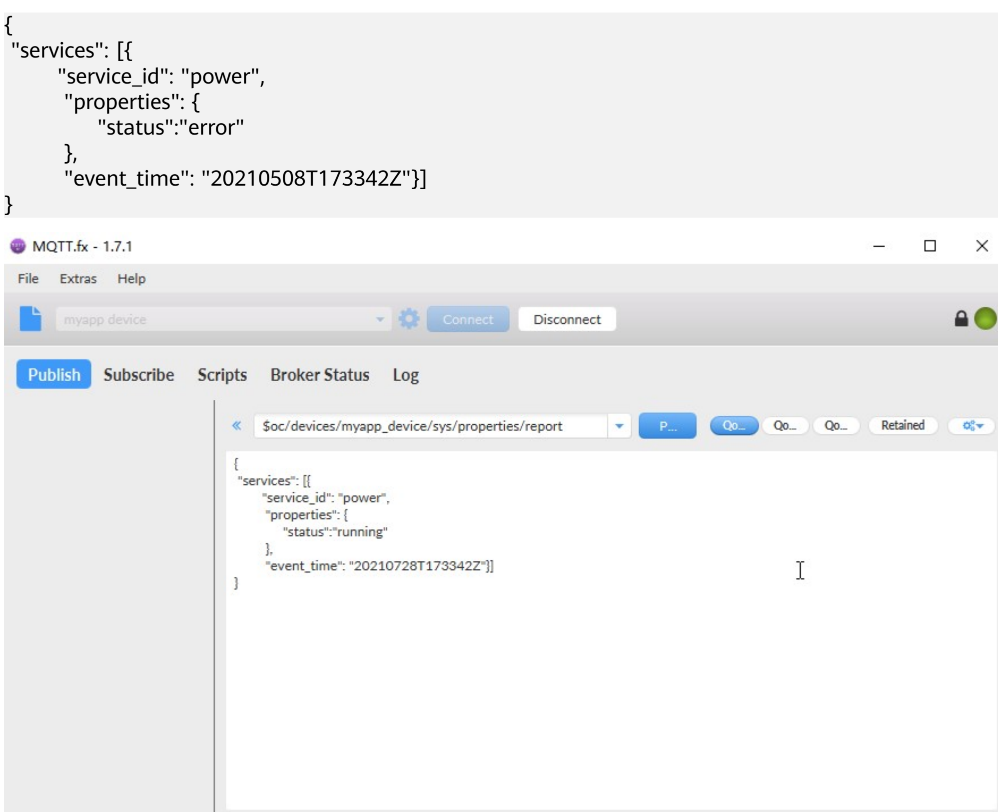
4. 选择publish输入topic地址。

 版权所有 © 华为云计算技术有限公司

Topic：$oc/devices/myapp_device/sys/properties/report

其中，**myapp_device**为设备ID，请替换为实际值,可在IoTDA->产品管理中查看。消息体输入：

```
{
 "services": [{
       "service_id": "power",
        "properties": {
            "status":"error"
        },
        "event_time": "20210508T173342Z"}]
}
```

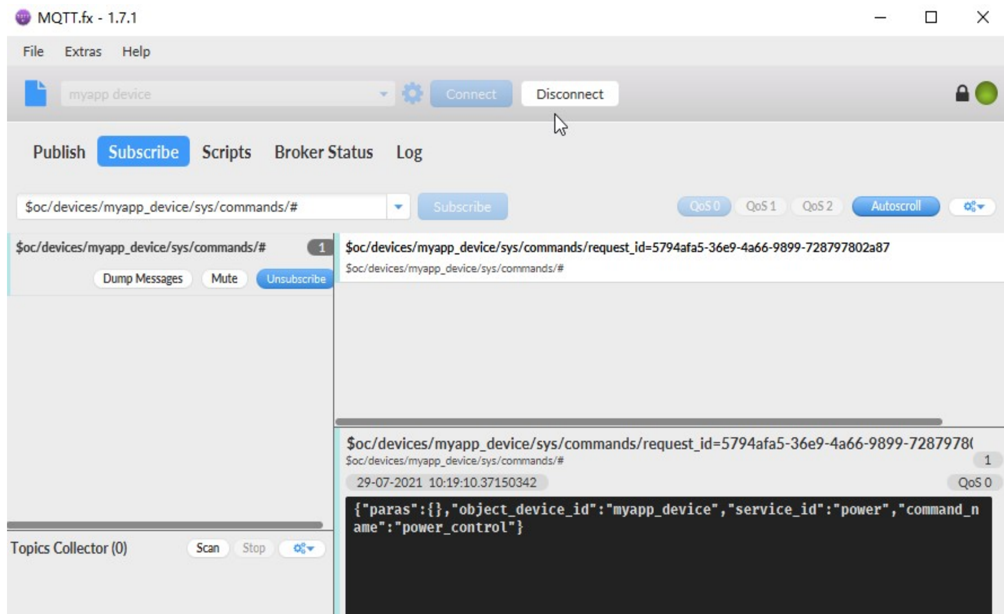


5. 输入订阅的topic，可在IoTDA->产品->topic管理中查看。

Topic：$oc/devices/myapp_device/sys/commands/#

其中，**myapp_device**为设备ID，请替换为实际值，设备ID请进入设备详情查看。

返回到publish，单击publish按钮后进入Subscribe,可以看到订阅命令收到一条command。

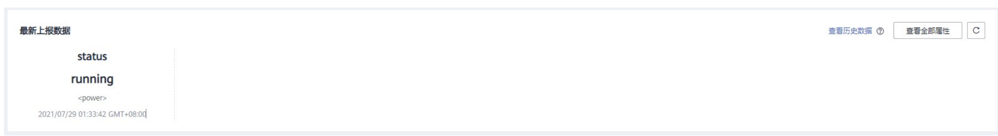


 版权所有 © 华为云计算技术有限公司

6. 进入边缘设备查看数据上报情况

进入IoTDA单击设备，进入概览发现并无数据上报，说明设备发送的数据在节点本地被集成SDK的monitor-app应用拦截，当数据内容为error时，在本地向hub调用重启命令。应用实现了数据处理和命令下发的功能。

如果该设备不属于之前代码修改里的电机设备（按产品ID区分），仍然可以看到上报的数据。



4.4.4.12 查看 SDK 运行日志

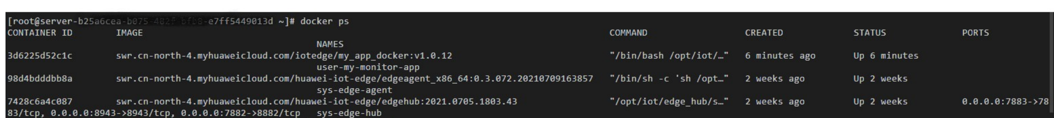
容器化部署-SDK 应用日志

通过”容器化部署”的SDK应用可以通过下述方式查看日志。

步骤1 登录节点

步骤2 查看SDK容器ID

通过**docker ps**查看



```
[root@server-b25a6cea-b075-482f-bfb8-e7ff5449013d ~]# docker ps
CONTAINER ID   IMAGE                                                                              COMMAND                  CREATED          STATUS         PORTS
                                                        NAMES
3d6225d52c1c   swr.cn-north-4.myhuaweicloud.com/iotedge/my_app_docker:v1.0.12                     "/bin/bash /opt/iot/…"   6 minutes ago    Up 6 minutes
                                                        user-my-monitor-app
98d4bdddbb8a   swr.cn-north-4.myhuaweicloud.com/huawei-iot-edge/edgeagent_x86_64:0.3.072.20210709163857   "/bin/sh -c 'sh /opt…"   2 weeks ago   Up 2 weeks
                                                        sys-edge-agent
7428c6a4c087   swr.cn-north-4.myhuaweicloud.com/huawei-iot-edge/edgehub:2021.0705.1803.43         "/opt/iot/edge_hub/s…"   2 weeks ago      Up 2 weeks     0.0.0.0:7883->78
83/tcp, 0.0.0.0:8943->8943/tcp, 0.0.0.0:7882->8882/tcp   sys-edge-hub
```

步骤3 查看SDK容器日志

通过**docker logs -f 容器ID**查看实时日志。

通过运行SDK的Demo应用，可以看到如下打印信息（仅供参考）。

图 **4-12** SDK 开始运行的日志

```
[root@server-b25a6cea-b075-482f-bfb8-e7ff5449013d ~]# docker logs -f 3d6225d52c1c
2021-07-29 03:37:13: [INFO] start execute process.
start monitor app
2021-07-29T03:37:14,206 | INFO | ec9839c0 | edge_base_init(edge_base.c:40) | sdk compile time: Jul 29 2021 11:17:29
2021-07-29T03:37:14,206 | INFO | ec9839c0 | edge_base_init(edge_base.c:41) | sdk work dir: ./
2021-07-29T03:37:14,206 | INFO | ec9839c0 | edge_base_init(edge_base.c:42) | version: 1.0.0
2021-07-29T03:37:14,206 | INFO | ec9839c0 | edge_base_init(edge_base.c:43) | base init success.
2021-07-29T03:37:14,206 | INFO | ec9839c0 | edge_json_init(edge_json.c:18) | json init success.
2021-07-29T03:37:14,239 | INFO | ec9839c0 | edge_http_init(edge_http.c:37) | env:daemon_verify_code is not empty, so is new daemon.
2021-07-29T03:37:14,239 | INFO | ec9839c0 | edge_http_init(edge_http.c:41) | env:daemon_uri is not empty, daemon_uri:unix:/opt/iot/edgeDaemon/socket/edgedaemon.sock.
2021-07-29T03:37:14,239 | INFO | ec9839c0 | edge_http_init(edge_http.c:54) | http init success, version: libcurl/7.64.0 OpenSSL/1.1.1c zlib/1.2.11 libidn2/2.0.5 libpsl/0.20.2 (+libidn2/2.0.5)
libssh2/1.8.0 nghttp2/1.36.0 librtmp/2.3
2021-07-29T03:37:14,239 | INFO | ec9839c0 | edge_mqtt_init(edge_mqtt.c:21) | mqtt init success.
2021-07-29T03:37:14,240 | INFO | ec9839c0 | edge_cipher_init(edge_cipher.c:26) | cipher init success.
2021-07-29T03:37:14,245 | INFO | ec9839c0 | edge_cipher_load_keystore_from_file(edge_cipher.c:503) | load domain key, name:0, lens:16, pointer:0x24f79a0
2021-07-29T03:37:14,245 | ERROR | ec9839c0 | edge_cipher_load_keystore_from_file_param_check(edge_cipher.c:429) | keystore file not found, file_name:/opt/config/secret/4e8a86a7502404a0
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_cipher_save_key(edge_cipher.c:66) | save key, domain_name:configs, enduring:0
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_configs_init(edge_configs.c:50) | edge configs init success.
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_base_schedule_job_create(edge_base_schedule_job.c:25) | schedule job create, name:edge_daemon_http_token
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_http_conns_init(edge_http_connections.c:40) | edge http connections init success.
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_base_retry_create(edge_base_retry.c:26) | retry worker create, name:edge_mqtt_connections
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_mqtt_conns_init(edge_mqtt_connections.c:26) | edge mqtt connections init success.
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_base_retry_create(edge_base_retry.c:26) | retry worker create, name:edge_app
2021-07-29T03:37:14,246 | INFO | ec9839c0 | edge_app_init(edge_app.c:52) | edge app init success.
SDK start running!
2021-07-29T03:37:15,246 | INFO | ec9839c0 | edge_app_login(edge_app.c:926) | login.
```

 版权所有 © 华为云计算技术有限公司

图 **4-13** SDK 结束运行的日志

```
2021-07-29T03:43:53,752 | DEBUG | e9158700 | edge_mqtt_message_arrived_callback(edge_mqtt.c:280) | mqtt recv message, topic:/modules/user_my_monitor_app/messages/inputs/input, len:197.
2021-07-29T03:43:53,752 | INFO | e9158700 | edge_mqtt_conns_recv_msg(edge_mqtt_connections.c:175) | recv msg, topic:***, payload_len:197
2021-07-29T03:43:53,752 | INFO | e9158700 | edge_app_recv_msg(edge_app.c:693) | set handler for topic: /modules/user_my_monitor_app/messages/inputs/input
2021-07-29T03:43:53,752 | DEBUG | e9158700 | edge_app_on_message_received_cb(edge_app.c:1204) | bus message product_id is: 60e6c0a5f4b59002867fcb02
2021-07-29T03:43:53,752 | DEBUG | e9158700 | edge_app_on_message_received_cb(edge_app.c:1205) | bus message device_id is: myapp_device
set bus message callback with input name: input
start send message to output topic: output
body is: {"services":[{"properties":{"status":"error"},"service_id":"power","event_time":"20210728T173342Z"}],"device_id":"myapp_device","product_id":"60e6c0a5f4b59002867fcb02","receive_time":
1627530233335}
body len is: 197
product_id is: 60e6c0a5f4b59002867fcb02
start processing device.
2021-07-29T03:43:53,752 | INFO | e9158700 | edge_app_send_bus_message(edge_app.c:1238) | start to send bus message.
2021-07-29T03:43:53,752 | INFO | e9158700 | edge_mqtt_publish(edge_mqtt.c:194) | mqtt: 0x7ff4e4010ab0 publish topic: /modules/user_my_monitor_app/messages/outputs/output lens: 197 qos: 1
2021-07-29T03:43:53,752 | INFO | e9158700 | edge_app_send_bus_message(edge_app.c:1246) | succeed to send bus message.
process ended.
2021-07-29T03:43:54,853 | DEBUG | e9158700 | edge_mqtt_delivery_complete_callback(edge_mqtt.c:301) | mqtt message delive complete, token:10.
```

----结束

安装包部署-SDK 应用日志

通过”安装包”部署的SDK应用可以下述方式查看日志。

步骤1 进入/var/IoTEdge/downloaded-job/run目录.

```
cd /var/IoTEdge/downloaded-job/run
```

步骤2 找到相应的安装包目录，找到myapp_running.log查看日志。

步骤3 可以看到SDK应用启动的日志如下

```
start monitor app
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_base_init(edge_base.c:40) | sdk compile time: Jul 29 2021 11:17:29
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_base_init(edge_base.c:41) | sdk work dir: ./
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_base_init(edge_base.c:42) | version: 1.0.0
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_base_init(edge_base.c:43) | base init success.
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_json_init(edge_json.c:18) | json init success.
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_http_init(edge_http.c:37) | env:daemon_verify_code is not empty, so is new daemon.
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_http_init(edge_http.c:41) | env:daemon_uri is not empty, daemon_uri:unix:/opt/IoTEdge/edgeDaemon/socket/edgedaemon.sock.
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_http_init(edge_http.c:54) | http init success, version: libcurl/7.29.0 NSS/3.53.1 zlib/1.2.7 libidn/1.28 libssh2/1.8.0
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_mqtt_init(edge_mqtt.c:21) | mqtt init success.
2021-08-02T17:15:03,588 | INFO | f696c840 | edge_cipher_init(edge_cipher.c:26) | cipher init success.
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_cipher_load_keystore_from_file(edge_cipher.c:503) | load domain key, name:0, lens:16, pointer:0x231bf30
2021-08-02T17:15:03,590 | ERROR | f696c840 | edge_cipher_load_keystore_from_file_param_check(edge_cipher.c:429) | keystore file not found, file_name:.//secret/4e8a86a7502404a0
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_cipher_save_key(edge_cipher.c:66) | save key, domain_name:configs, enduring:0
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_configs_init(edge_configs.c:50) | edge configs init success.
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_base_schedule_job_create(edge_base_schedule_job.c:25) | schedule job create, name:edge_daemon_http_token
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_http_conns_init(edge_http_connections.c:40) | edge http connections init success.
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_base_retry_create(edge_base_retry.c:26) | retry worker create, name:edge_mqtt_connections
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_mqtt_conns_init(edge_mqtt_connections.c:26) | edge mqtt connections init success.
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_base_retry_create(edge_base_retry.c:26) | retry worker create, name:edge_app
2021-08-02T17:15:03,590 | INFO | f696c840 | edge_app_init(edge_app.c:52) | edge app init success.
SDK start running!
2021-08-02T17:15:04,590 | INFO | f696c840 | edge_app_login(edge_app.c:926) | login.
2021-08-02T17:15:04,590 | INFO | f696c840 | edge_app_get_env(edge_app.c:738) | get env:device_id
```

步骤4 通过用**MQTT设备模拟器**发送数据，可以看到SDK处理相应的消息日志。

```
[root@server-b25a6cea-b075-482f-bfb8-e7ff5449013d 1.0.4]# tail -f myapp_running.log
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: /modules/user_my-c-sdk/messages/inputs/+ qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/events/down qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/commands/+ qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/messages/down qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/properties/get/+ qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/properties/set/+ qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: $oc/modules/user_my-c-sdk/sys/shadow/get/response/+ qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_mqtt_subscribe(edge_mqtt.c:156) | mqtt: 0x7f7be800e7c0 subscribe topic: /modules/user_my-c-sdk/customized/inputs/# qos: 1
2021-08-02T17:15:35,537 | INFO | dffff700 | edge_base_retry_success_notify(edge_base_retry.c:61) | retry worker:edge_mqtt_connections notify success.
2021-08-02T17:16:07,154 | INFO | dffff700 | edge_base_retry_stop(edge_base_retry.c:79) | retry worker edge_mqtt_connections exited, ret:0
2021-08-03T11:29:36,718 | DEBUG | dffff700 | edge_mqtt_message_arrived_callback(edge_mqtt.c:280) | mqtt recv message, topic:/modules/user_my-c-sdk/messages/inputs/input, len:199.
2021-08-03T11:29:36,718 | INFO | dffff700 | edge_mqtt_conns_recv_msg(edge_mqtt_connections.c:175) | recv msg, topic:***, payload_len:199
2021-08-03T11:29:36,718 | INFO | dffff700 | edge_app_recv_msg(edge_app.c:693) | set handler for topic: /modules/user_my-c-sdk/messages/inputs/input
2021-08-03T11:29:36,718 | DEBUG | dffff700 | edge_app_on_message_received_cb(edge_app.c:1204) | bus message product_id is: 60e6c0a5f4b59002867fcb02
2021-08-03T11:29:36,718 | DEBUG | dffff700 | edge_app_on_message_received_cb(edge_app.c:1205) | bus message device_id is: myapp_device
set bus message callback with input name: input
start send message to output topic: output
body is: {"services":[{"properties":{"status":"running"},"service_id":"power","event_time":"20210508T173342Z"}],"device_id":"myapp_device","product_id":"60e6c0a5f4b59002867fcb02","receive_time
":1627961376294}
body len is: 199
product_id is: 60e6c0a5f4b59002867fcb02
start processing device.
2021-08-03T11:29:36,718 | INFO | dffff700 | edge_app_send_bus_message(edge_app.c:1238) | start to send bus message.
2021-08-03T11:29:36,718 | INFO | dffff700 | edge_mqtt_publish(edge_mqtt.c:194) | mqtt: 0x7f7be800e7c0 publish topic: /modules/user_my-c-sdk/messages/outputs/output lens: 199 qos: 1
2021-08-03T11:29:36,718 | INFO | dffff700 | edge_app_send_bus_message(edge_app.c:1246) | succeed to send bus message.
process ended.
plete, token:10.
```

----结束

4.4.5 ModuleSDK-C Demo 展示

4.4.5.1 Demo1

说明

该Demo主要展示edge.h和edge_daemon.h里的回调函数和接口函数使用。

```
#include "edge.h"
#include "edge_daemon.h"
```

 版权所有 © 华为云计算技术有限公司

```
#include <stdio.h>
#include <unistd.h>

/*
 * 描述：针对模块的命令的回调函数
 * 参数：
 *  command_name：命令名称
 *  device_id：设备Id
 *  service_id：服务Id
 *  request_id：请求Id（响应命令）
 *  body：命令体
 *  body_len：命令体长度
 * */
EDGE_RETCODE moduleCmdCb(const char* command_name, const char* device_id,
                                  const char* service_id, const char* request_id, const char* body, unsigned int
body_len)
{
    printf("command_name:%s, device_id:%s, service_id:%s, request_id:%s, body:%s, body_len:%d",
            command_name, device_id, service_id, request_id, body, body_len);

    char* rsp = "{\"error_desc\":\"ok\"}";

    //命令响应
    edge_send_command_rsp(NULL, request_id, 0, rsp, strlen(rsp));

    return EDGE_SUCCESS;
}

/*
 * 描述：针对子设备的命令的回调函数
 * 参数：
 *  command_name：命令名称
 *  device_id：子设备Id
 *  service_id：服务Id
 *  request_id：请求Id（响应命令）
 *  body：命令体
 *  body_len：命令体长度
 * */
EDGE_RETCODE deviceCmdCb(const char* command_name, const char* device_id,
                 const char* service_id, const char* request_id, const char* body, unsigned int body_len)
{
    printf("command_name:%s, device_id:%s, service_id:%s, request_id:%s, body:%s, body_len:%d",
           command_name, device_id, service_id, request_id, body, body_len);

    char* rsp = "{\"error_desc\":\"ok\"}";

    edge_send_command_rsp(NULL, request_id, 0, rsp, strlen(rsp));

    return EDGE_SUCCESS;
}

/*
 * 描述：影子回调
 * 参数：
 *  shadow：影子数据
 *  body_len：长度
 * */
EDGE_RETCODE shadowCb(const char* shadow, unsigned int shadow_len)
{
    printf("shadowCb:%s\n", shadow);

    return EDGE_SUCCESS;
}

/*
 * 描述：自定义topic消息
 * 参数：
 *  shadow：影子数据
 *  body_len：长度
```

```
 * */
EDGE_RETCODE customizedMessageCb(const char* topic, const char* payload, unsigned int len)
{
    printf("topic:%s\n", topic);
    printf("payload:%s\n", payload);

    return EDGE_SUCCESS;
}

/*
 * 描述：收到设备上报数据的回调处理
 * 参数：
 *  device_id：设备ID
 *  product_id: 产品ID
 *  body: 上报的数据
 *  body_len: 上报数据的大小
 * */
EDGE_RETCODE messageReceivedCb(const char* device_id, const char* product_id, const char* body,
unsigned int body_len)
{
    printf("body is: %s\nbody len is: %d\n", body, body_len);
    printf("product_id is: %s, device_id: %s\n", product_id, device_id);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备添加的回调
 * 参数：
 *  add_device_info：添加的子设备信息
 *  len：长度
 * */
EDGE_RETCODE sub_device_add_cb(const char* add_device_info, unsigned int len)
{
    printf("sub_device_add_cb:%s\n", add_device_info);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备删除的回调
 * 参数：
 *  delete_device_info：删除的子设备信息
 *  len：长度
 * */
EDGE_RETCODE sub_device_delete_cb(const char* delete_device_info, unsigned int len)
{
    printf("sub_device_delete_cb:%s\n", delete_device_info);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备请求响应的回调
 * 参数：
 *  getProductsRspEvent：获取到的子设备请求响应信息
 * */
EDGE_RETCODE get_products_response_cb(ST_GET_PRODUCTS_RSP_EVENT* getProductsRspEvent)
{
    int i, j, k, l, m;
    for (i = 0; i < getProductsRspEvent->product_len; i++)
    {
        printf("%d-th product's info shows below:\n", i+1);
        printf("product_id:%s\n", getProductsRspEvent->products[i].product_id);
        printf("name:%s\n", getProductsRspEvent->products[i].name);
        printf("device_type:%s\n", getProductsRspEvent->products[i].device_type);
        printf("protocol_type:%s\n", getProductsRspEvent->products[i].protocol_type);
        printf("data_format:%s\n", getProductsRspEvent->products[i].data_format);
        printf("industry:%s\n", getProductsRspEvent->products[i].industry);
```

```
        printf("name:%s\n", getProductsRspEvent->products[i].name);
        printf("service_capabilities:\n");
        for (j = 0; j < getProductsRspEvent->products[i].service_capability_len; j++)
        {
            printf(" service_id:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].service_id);
            printf("service_type:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].service_type);
            printf("description:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].description);
            printf("option:%s\n", getProductsRspEvent->products[i].serviceCapabilities[j].option);
            printf(" properties:\n");
            for(k = 0; k < getProductsRspEvent->products[i].serviceCapabilities[j].property_len; k++)
            {
                printf("  property_name:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].property_name);
                printf("required:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].required ? "true" : "false");
                printf("data_type:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].data_type);
                printf("enum_list:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].enum_list);
                printf("min:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].min);
                printf("max:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].max);
                printf("max_length:%d, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].max_length);
                printf("step:%lf, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].step);
                printf("unit:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].unit);
                printf("method:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].method);
                printf("description:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].description);
                printf("default_value:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].default_value);
            }

            printf(" commands:\n");
            for(k = 0; k < getProductsRspEvent->products[i].serviceCapabilities[j].command_len; k++)
            {
                printf("  command_name:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].command_name);
                printf("  paras:\n");
                for(l = 0; l < getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].para_len; l++)
                {
                    printf("    para_name:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].para_name);
                    printf("required:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].required ? "true" : "false");
                    printf("data_type:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].data_type);
                    printf("enum_list:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].enum_list);
                    printf("min:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].min);
                    printf("max:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].max);
                    printf("max_length:%d, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].max_length);
                    printf("step:%lf, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].step);
                    printf("unit:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].unit);
                    printf("description:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].description);
                }

                printf("  responses:\n");
                for(l = 0; l < getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].response_len; l
++)
                {
                    printf("   response_name:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].response_name);
```


版权所有 © 华为云计算技术有限公司

```
                    printf("    paras:\n");
                    for(m = 0; m < getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].service_command_para_len; m++)
                    {
                        printf("      para_name:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].para_name);
                        printf("required:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].required ? "true" : "false");
                        printf("data_type:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].data_type);
                        printf("enum_list:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].enum_list);
                        printf("min:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].min);
                        printf("max:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].max);
                        printf("max_length:%d, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].max_length);
                        printf("step:%lf, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].step);
                        printf("unit:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].unit);
                        printf("description:%s\n", getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].description);
                    }
                }
            }
        }
        printf("\n");
    }

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备事件的回调
 * 参数：
 *  device_id：子设备Id
 *  body：事件体
 *  len：长度
 * */
EDGE_RETCODE sub_device_event_cb(const char* device_id, const char* body, unsigned int body_len)
{
    printf("%s\n", device_id);
    printf("%s", body);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备start_scan事件的回调
 * 参数：
 *  protocol：协议
 *  channel：通道
 *  parentDeviceId：父设备Id
 *  scan_setting：扫描设置
 *  body_len：长度
 * */
EDGE_RETCODE sub_device_start_scan_cb(const char* protocol, const char* channel, const char* parentDeviceId,const char* scan_setting, unsigned int body_len)
{
    printf("protocol = %s, channel = %s, parentDeviceId = %s, scan_setting = %s", protocol, channel, parentDeviceId, scan_setting);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备消息的回调
```

```
 * 参数:
 *  message_id：消息Id
 *  channel：通道
 *  device_id：设备Id
 *  body：消息体
 *  body_len：长度
 * */
EDGE_RETCODE sub_device_messages_down_cb(const char* message_id, const char* message_name, const
char* device_id, const char* body, unsigned int body_len)
{
    printf("message_id = %s, message_name = %s, device_id = %s, body = %s", message_id, message_name,
device_id, body);

    return EDGE_SUCCESS;
}

/*
 * 描述：子设备收到属性设置的回调
 * 参数:
 *  sub_device_property_set：属性设置的具体数据
 * */
EDGE_RETCODE sub_device_properties_set_cb(ST_PROPERTY_SET* sub_device_property_set)
{
    printf("object_device_id = %s, request_id = %s\n", sub_device_property_set->object_device_id,
sub_device_property_set->request_id);
    printf("services_size = %d\n", sub_device_property_set->services_size);
    int i;
    for(i = 0; i< sub_device_property_set->services_size; i++)
    {
        printf("service_id = %s, properties = %s\n", sub_device_property_set->services[i].service_id,
sub_device_property_set->services[i].properties);
    }
    ST_IOT_RESULT result;
    result.result_desc = "ok";
    result.result_code = "0";
    edge_send_sub_device_property_set_rsp(sub_device_property_set->request_id, &result);
    return EDGE_SUCCESS;
}

EDGE_RETCODE sub_device_properties_get_cb(ST_PROPERTY_GET* sub_device_property_get)
{
    printf("object_device_id = %s, request_id = %s, service_id = %s", sub_device_property_get-
>object_device_id, sub_device_property_get->request_id, sub_device_property_get->service_id);
    char* data_body1 = "{\n"
                      "\t\"PhV_phsA\":1,\n"
                      "\t\"PhV_phsB\":2\n"
                      "}";
    ST_DEVICE_PROPERTY_GET_RSP device_property_get_rsp;
    ST_SERVICE_DATA service_data_1 = {0};
    service_data_1.service_id = "service_id_1";
    service_data_1.properties = data_body1;
    service_data_1.event_time = "20200520T115630Z";
    device_property_get_rsp.services = &service_data_1;
    device_property_get_rsp.services_size = 1;

    edge_send_sub_device_property_get_rsp(sub_device_property_get->request_id, &device_property_get_rsp);
    return EDGE_SUCCESS;
}

EDGE_RETCODE sub_device_shadow_cb(ST_DEVICE_SHADOW* sub_device_shadow)
{
    printf("object_device_id = %s, request_id = %s\n", sub_device_shadow->object_device_id,
sub_device_shadow->request_id);
    printf("services_size = %d\n", sub_device_shadow->shadow_size);
    int i;
    for(i = 0; i< sub_device_shadow->shadow_size; i++) {
        printf("service_id = %s, properties = %s\n", sub_device_shadow->shadow[i].service_id,
sub_device_shadow->shadow[i].desired_properties);
    }
```


版权所有 © 华为云计算技术有限公司

```
    return EDGE_SUCCESS;
}

/*
 * 描述：添加子设备调用方式示例
 * */
void send_add_sub_device(){
    //构造设备数据
    ST_DEVICE_INFO device_info[2];
    ST_DEVICE_INFO device_info_1 = {0};
    device_info_1.device_id = "device_id-1";
    device_info_1.parent_device_id = "parent_device_id_1";
    device_info_1.node_id = "node_id_1";
    device_info_1.name = "name_1";
    device_info_1.product_id = "product_id_1";
    device_info_1.status = "RUNNING";
    ST_DEVICE_INFO device_info_2 = {0};
    device_info_2.device_id = "device_id-2";
    device_info_2.parent_device_id = "parent_device_id_2";
    device_info_2.node_id = "node_id_2";
    device_info_2.name = "name_2";
    device_info_2.product_id = "product_id_2";
    device_info_2.status = "RUNNING";
    device_info[0] = device_info_1;
    device_info[1] = device_info_2;

    edge_send_add_sub_device(device_info, 2);
}

/*
 * 描述：添加子设备调用方式示例
 * */
void send_delete_sub_device(){
    //构造设备数据
    ST_DEVICE_INFO device_info[2];
    ST_DEVICE_INFO device_info_1 = {0};
    device_info_1.device_id = "device_id-1";
    device_info_1.parent_device_id = "parent_device_id_1";
    device_info_1.node_id = "node_id_1";
    device_info_1.name = "name_1";
    device_info_1.product_id = "product_id_1";
    device_info_1.status = "RUNNING";
    ST_DEVICE_INFO device_info_2 = {0};
    device_info_2.device_id = "device_id-2";
    device_info_2.parent_device_id = "parent_device_id_2";
    device_info_2.node_id = "node_id_2";
    device_info_2.name = "name_2";
    device_info_2.product_id = "product_id_2";
    device_info_2.status = "RUNNING";
    device_info[0] = device_info_1;
    device_info[1] = device_info_2;

    edge_send_delete_sub_device(device_info, 2);
}

/*
 * 描述：同步子设备调用方式示例
 * 参数;
 * version:版本（添加子设备成功后回调接口有此字段）
 * */
void send_sync_sub_device(long long version){
    edge_send_sync_sub_device(version);
}

/*
 * 描述：发送子设备状态方式示例
 * */
void send_sub_device_status(){
    /*
```


版权所有 © 华为云计算技术有限公司

```
     * 构造 {"device_statuses":[{
     *          "device_id":"deviceId",
     *          "status":"status"
     *  }]}
     * */
    ST_DEVICE_STATUS device_status[2];
    ST_DEVICE_STATUS device_status_1 = {0};
    device_status_1.device_id = "device_id_1";
    device_status_1.status = "RUNNING";
    device_status[0] = device_status_1;
    ST_DEVICE_STATUS device_status_2 = {0};
    device_status_2.device_id = "device_id_2";
    device_status_2.status = "RUNNING";
    device_status[1] = device_status_2;

    edge_send_sub_device_status(device_status, 2);
}

/*
 * 发送子设备数据
 * */
void send_batch_device_data(){
    /*
     * 具体上报的子设备数据和设备的产品模型有关系，和属性对应
     * */
    char* data_body1 = "{\n"
                     "\t\"PhV_phsA\":1,\n"
                     "\t\"PhV_phsB\":2\n"
                     "}";
    char* data_body2 =  "{\"PhV_phsA\":\"1\",\"PhV_phsB\":\"2\"}";
    printf("data_body1:%s\n", data_body1);
    printf("data_body2:%s\n", data_body2);
    ST_DEVICE_SERVICE device_data[2];
    ST_DEVICE_SERVICE devcie_data_1 = {0};
    ST_DEVICE_SERVICE devcie_data_2 = {0};

    ST_SERVICE_DATA service_data[2];
    ST_SERVICE_DATA service_data_1 = {0};
    service_data_1.service_id = "service_id_1";
    service_data_1.properties = data_body1;
    service_data_1.event_time = "20200520T115630Z";
    ST_SERVICE_DATA service_data_2 = {0};
    service_data_2.service_id = "service_id_2";
    service_data_2.properties = data_body2;
    service_data_2.event_time = "20200520T115630Z";
    service_data[0] = service_data_1;
    service_data[1] = service_data_2;

    devcie_data_1.services = service_data;
    devcie_data_1.device_id = "device_id_1";
    devcie_data_1.size = 2;

    devcie_data_2.services = service_data;
    devcie_data_2.device_id = "device_id_2";
    devcie_data_2.size = 2;
    device_data[0] = devcie_data_1;
    device_data[1] = devcie_data_2;

    edge_send_batch_device_data(device_data, 2);
}

/*
 * 描述：连接到hub的回调
 * */
void connected()
{
    edge_get_shadow();
```

 版权所有 © 华为云计算技术有限公司

```
    ST_CLOUD_TOKEN token = {0};

    edge_get_cloud_token(&token);

    printf("ak:%s sk:%s region:%s expires_at:%s\n", token.ak, token.sk, token.region, token.expire_time);

    char* body = "{\n"
                "\t\"module_id\": \"module_id\",\n"
                "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
                "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
                "}";

    edge_send_service_event("module_management", "module_status_change", body, strlen(body));
    sleep(10);
    send_add_sub_device();
    sleep(10);
    send_delete_sub_device();
    sleep(10);
    send_sync_sub_device(1234);
    sleep(10);
    send_sub_device_status();
    sleep(10);
    send_batch_device_data();
    sleep(10);
}

/*
 * 描述：与hub断链的回调
 * */
void disconnected()
{
    printf("disconnected.");
}

/*
 * 描述：获取模块的token
 * */
void send_get_daemon_token()
{
    ST_DAEMON_TOKEN daemon_token;
    int ret = edge_daemon_get_token(&daemon_token);
    if (ret != EDGE_SUCCESS) {
        printf("get daemon token fail.");
        return;
    }
    printf("get daemon token success.");
    printf("daemon token:%s\n", daemon_token.token);
    printf("daemon token expires_at:%lld\n", daemon_token.expires_at);
}

/*
 * 描述：获取云端授权token信息，仅系统模块可用
 * */
void send_get_daemon_cloud_token()
{
    ST_DAEMON_CLOUD_TOKEN daemon_token;
    int ret = edge_daemon_cloud_token(&daemon_token);
    if (ret != EDGE_SUCCESS) {
        printf("get daemon cloud token fail.");
        return;
    }
    printf("get daemon cloud token success.");
    printf("daemon cloud token:%s\n", daemon_token.token);
    printf("daemon cloud token expires_at:%s\n", daemon_token.access_key);
}

/*
 * 描述：A模块校验B的访问模块Token
 * */
```


版权所有 © 华为云计算技术有限公司

```
void send_check_daemon_token()
{
    ST_VERIFY_RESPONSE token;
    int ret =
edge_daemon_verify("Algorithm=HMAC_SHA_256;NodeId=076aeddff400d2472f60c012d120bc7f;ModuleId=u
ser_edge_test;SignedTime=1617759175394;Signature=6af3c8306e4f370a2939803d06719737da1b03cf0d7a04
ccdadcc8830460af6c", &token);
    if (ret != EDGE_SUCCESS) {
        printf("check daemon token fail.");
        return;
    }
    printf("check daemon token success.");
    printf("verify moduleId:%s\n", token.module_id);
    printf("verify token expires_at:%lld\n", token.expires_at);
}

/*
 * 描述：获取节点证书，系统模块可用
 * */
void send_get_node_certs()
{
    ST_NODE_CERT node_cert;
    int ret = edge_daemon_node_certs(&node_cert);
    if (ret != EDGE_SUCCESS) {
        printf("get node cert fail.");
        return;
    }
    printf("get node cert success.");
    printf("node cert certificate:%s\n", node_cert.certificate);
    printf("node cert expires_at:%lld\n", node_cert.expires_at);
}

/*
 * 描述：获取模块信任的证书
 * */
void send_get_module_trust_certs()
{
    ST_MODULE_TRUST_CERTS module_trust_certs;
    int ret = edge_daemon_trust_certs(&module_trust_certs);
    if (ret != EDGE_SUCCESS) {
        printf("get module trust cert fail.");
        return;
    }
    printf("get module trust cert success.");
    printf("module trust cert certificate:%s\n", module_trust_certs.certificate);
    printf("module trust cert expires_at:%lld\n", module_trust_certs.expires_at);
}

void send_send_get_sub_device_shadow()
{
    ST_DEVICE_SHADOW_GET device_shadow_get = {0};
    device_shadow_get.service_id = "service1";
    device_shadow_get.object_device_id = "deviceId1";
    char* request_id = "123";
    edge_send_get_sub_device_shadow(request_id, &device_shadow_get);
}

void send_sub_device_message()
{
    ST_DEVICE_MESSAGE device_message = {0};
    device_message.object_device_id = "deviceId1";
    device_message.name = "message_name";
    device_message.id = "message_id";
    device_message.content =  "{\n"
                              "\t\"module_id\": \"module_id\",\n"
                              "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
                              "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
                              "}";
    device_message.content_len = (int)strlen(device_message.content);
```

 版权所有 © 华为云计算技术有限公司

```
    edge_send_sub_device_message(&device_message);
}

void send_sub_device_event()
{
    ST_DEVICE_EVENT device_event = {0};
    device_event.object_device_id = "deviceId1";
    ST_SERVICE_EVENT service_event = {0};
    service_event.service_id = "message_name";
    service_event.event_id = "message_id";
    service_event.event_type = "message_name";
    service_event.paras =  "{\n"
                             "\t\"module_id\": \"module_id\",\n"
                             "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
                             "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
                             "}";
    service_event.paras_len = (int)strlen(service_event.paras);
    device_event.services = &service_event;
    device_event.services_size = 1;
    edge_send_sub_device_event(&device_event);
}

void send_get_sub_device_product()
{
    char* product_ids[10] = {0};
    product_ids[0] = "1234";
    product_ids[1] = "1234";
    edge_send_get_product(product_ids, 2);
}


void send_customized_message()
{
    char* topic = "hello";
    char* body = "{\n"
                 "\t\"PhV_phsA\":1,\n"
                 "\t\"PhV_phsB\":2\n"
                 "}";;
    edge_send_customized_message(topic, body, strlen(body));
}

void send_customized_message2()
{
    char* topic = "/hello/test2";
    char* body = "{\n"
                 "\t\"PhV_phsA\":1,\n"
                 "\t\"PhV_phsB\":2\n"
                 "}";;
    edge_send_customized_message(topic, body, strlen(body));
}
```

说明

下面是主函数demo，里面包括调用上面所有的demo函数，用不到的地方注释掉即可。

```
int main()
{
    // 禁用缓冲区
    setvbuf(stdout, NULL, _IONBF, 0);

    //初始化sdk，工作路径设置（工作路径下需要含有 /conf  目录（该目录下包含证书等信息））
    edge_init("../code/api_test/workdir");

    printf("demo start.\n");

    ST_MODULE_CBS cbs = {0};
    cbs.pfn_command_cb = moduleCmdCb;
    cbs.pfn_shadow_cb = shadowCb;
    cbs.pfn_customized_message_cb = customizedMessageCb;
    cbs.pfn_connected = connected;
```

```
    cbs.pfn_disconnected = disconnected;
    cbs.pfn_on_message_received_cb = messageReceivedCb;

    ST_DEVICE_CBS device_cbs = {0};
    device_cbs.pfn_sub_device_add_cb = sub_device_add_cb;
    device_cbs.pfn_device_command_cb = deviceCmdCb;
    device_cbs.pfn_sub_device_deleted_cb = sub_device_delete_cb;
    device_cbs.pfn_device_event_cb = sub_device_event_cb;
    device_cbs.pfn_on_start_scan_cb = sub_device_start_scan_cb;
    device_cbs.pfn_device_message_cb = sub_device_messages_down_cb;
    device_cbs.pfn_on_get_products_rsp_cb = get_products_response_cb;

    device_cbs.pfn_device_properties_set_cb = sub_device_properties_set_cb;
    device_cbs.pfn_device_properties_get_cb = sub_device_properties_get_cb;
    device_cbs.pfn_device_shadow_cb = sub_device_shadow_cb;
    //设置回调函数（无需全部设置，按需设置需要接收的回调）
    edge_set_callbacks(&cbs, &device_cbs);

    //连接hub
    edge_login();

    sleep(10);
    send_send_get_sub_device_shadow();
    sleep(1);
    send_sub_device_message();
    sleep(1);
    send_sub_device_event();
    sleep(1);
    send_get_sub_device_product();
    sleep(1);
    send_customized_message();
    sleep(1);
    send_customized_message2();

    while(1) {
        sleep(1000);
    }

    //登出
    edge_logout();
    sleep(10000);
    //清理
    edge_destroy();

    return 0;
}
```

4.4.5.2 Demo2

说明

该Demo主要展示edge_driver.h里的回调函数和接口函数使用。

```
#include "edge_driver.h"

#include <stdio.h>
#include <string.h>
#include <unistd.h>

/**
 * 驱动Demo
 * Demo内容涉及函数函数，接口函数，Modbus Demo和other Demo
 * Modbus Demo和other Demo主要是展示调用有关接口
 * ****************************************/

/****************************** 回调函数Demo ****************************************/

/**
 * 描述：子设备添加的回调
```

```
 * 参数：
 *  event_id: null
 *  add_sub_devices_event：添加的子设备信息
 */
EDGE_RETCODE sub_device_add_cb_1(const char* event_id, ST_ADD_SUB_DEVICES_EVENT*
add_sub_devices_event)
{
    int i;

    printf("add_sub_device_event payload shows below:\n");
    printf("device's version:%lld\n", (long long)add_sub_devices_event->version);
    printf("devices:\n");
    for(i = 0; i < add_sub_devices_event->device_len; i++)
    {
        printf(" {parent_device_id:%s, ", add_sub_devices_event->devices[i].parent_device_id);
        printf("node_id:%s, ", add_sub_devices_event->devices[i].node_id);
        printf("device_id:%s, ", add_sub_devices_event->devices[i].device_id);
        printf("name:%s, ", add_sub_devices_event->devices[i].name);
        printf("description:%s, ", add_sub_devices_event->devices[i].description);

        printf("product_id:%s, ", add_sub_devices_event->devices[i].product_id);
        printf("fw_version:%s, ", add_sub_devices_event->devices[i].fw_version);
        printf("sw_version:%s, ", add_sub_devices_event->devices[i].sw_version);
        printf("status:%s}\n", add_sub_devices_event->devices[i].status);
    }

    return EDGE_SUCCESS;
}

/**
 * 描述：子设备删除的回调
 * 参数：
 *  event_id: null
 *  delete_sub_devices_event：添加的子设备信息
 */
EDGE_RETCODE sub_device_delete_cb_1(const char* event_id, ST_DELETE_SUB_DEVICES_EVENT*
delete_sub_devices_event)
{
    int i;

    printf("delete_sub_device_event payload shows below:\n");
    printf("device's version:%lld\n", delete_sub_devices_event->version);
    printf("devices:\n");
    for(i = 0; i < delete_sub_devices_event->device_len; i++)
    {
        printf(" {parent_device_id:%s, ", delete_sub_devices_event->devices[i].parent_device_id);
        printf("node_id:%s, ", delete_sub_devices_event->devices[i].node_id);
        printf("device_id:%s, ", delete_sub_devices_event->devices[i].device_id);
        printf("name:%s, ", delete_sub_devices_event->devices[i].name);
        printf("description:%s, ", delete_sub_devices_event->devices[i].description);

        printf("product_id:%s, ", delete_sub_devices_event->devices[i].product_id);
        printf("fw_version:%s, ", delete_sub_devices_event->devices[i].fw_version);
        printf("sw_version:%s, ", delete_sub_devices_event->devices[i].sw_version);
        printf("status:%s}\n", delete_sub_devices_event->devices[i].status);
    }

    return EDGE_SUCCESS;
}

/**
 * 描述：针对子设备的命令的回调函数
 * 参数：
 *  command_name：命令名称
 *  device_id：子设备Id
 *  service_id：服务Id
 *  request_id：请求Id（响应命令）
 *  body：命令体
 *  body_len：命令体长度
```


版权所有 © 华为云计算技术有限公司

```
 */
EDGE_RETCODE deviceCmdCb(const char* command_name, const char* device_id,
                        const char* service_id, const char* request_id, const char* body, unsigned int body_len)
{
    printf("command_name:%s, device_id:%s, service_id:%s, request_id:%s, body:%s, body_len:%d",
          command_name, device_id, service_id, request_id, body, body_len);

    char* rsp = "{\"error_desc\":\"ok\"}";

    edge_send_command_rsp(NULL, request_id, 0, rsp, strlen(rsp));

    return EDGE_SUCCESS;
}

/**
 * 描述：子设备事件的回调
 * 参数：
 *  device_id：子设备Id
 *  body：事件体
 *  len：长度
 */
EDGE_RETCODE sub_device_event_cb(const char* device_id, const char* body, unsigned int body_len)
{
    printf("%s\n", device_id);
    printf("%s", body);

    return EDGE_SUCCESS;
}

/**
 * 描述：子设备start_scan事件的回调
 * 参数：
 *  protocol：协议
 *  channel：通道
 *  parentDeviceId：父设备Id
 *  scan_setting：扫描设置
 *  body_len：长度
 */
EDGE_RETCODE sub_device_start_scan_cb(const char* protocol, const char* channel, const char*
parentDeviceId,const char* scan_setting, unsigned int body_len)
{
    printf("protocol = %s, channel = %s, parentDeviceId = %s, scan_setting = %s", protocol, channel,
parentDeviceId, scan_setting);

    return EDGE_SUCCESS;
}

/**
 * 描述：子设备消息的回调
 * 参数：
 *  message_id：消息Id
 *  channel：通道
 *  device_id：设备Id
 *  body：消息体
 *  body_len：长度
 */
EDGE_RETCODE sub_device_messages_down_cb(const char* message_id, const char* message_name, const
char* device_id, const char* body, unsigned int body_len)
{
    printf("message_id = %s, message_name = %s, device_id = %s, body = %s", message_id, message_name,
device_id, body);

    return EDGE_SUCCESS;
}

/**
 * 描述：子设备收到属性设置的回调
 * 参数：
 *  sub_device_property_set：属性设置的具体数据
```

```
 */
EDGE_RETCODE sub_device_properties_set_cb(ST_PROPERTY_SET* sub_device_property_set)
{
    printf("object_device_id = %s, request_id = %s\n", sub_device_property_set->object_device_id,
sub_device_property_set->request_id);
    printf("services_size = %d\n", sub_device_property_set->services_size);
    int i;
    for(i = 0; i< sub_device_property_set->services_size; i++)
    {
        printf("service_id = %s, properties = %s\n", sub_device_property_set->services[i].service_id,
sub_device_property_set->services[i].properties);
    }
    ST_IOT_RESULT result;
    result.result_desc = "ok";
    result.result_code = "0";
    edge_send_sub_device_property_set_rsp(sub_device_property_set->request_id, &result);
    return EDGE_SUCCESS;
}

/**
 * 描述：子设备属性获取的回调
 * 参数：
 *  sub_device_property_get：获取到的子设备属性
 */
EDGE_RETCODE sub_device_properties_get_cb(ST_PROPERTY_GET* sub_device_property_get)
{
    printf("object_device_id = %s, request_id = %s, service_id = %s", sub_device_property_get-
>object_device_id, sub_device_property_get->request_id, sub_device_property_get->service_id);
    char* data_body1 = "{\n"
                       "\t\"PhV_phsA\":1,\n"
                       "\t\"PhV_phsB\":2\n"
                       "}";
    ST_DEVICE_PROPERTY_GET_RSP device_property_get_rsp;
    ST_SERVICE_DATA service_data_1 = {0};
    service_data_1.service_id = "service_id_1";
    service_data_1.properties = data_body1;
    service_data_1.event_time = "20200520T115630Z";
    device_property_get_rsp.services = &service_data_1;
    device_property_get_rsp.services_size = 1;

    edge_send_sub_device_property_get_rsp(sub_device_property_get->request_id, &device_property_get_rsp);
    return EDGE_SUCCESS;
}

/**
 * 描述：获取子设备影子的回调
 * 参数：
 *  sub_device_shadow：子设备影子
 */
EDGE_RETCODE sub_device_shadow_cb(ST_DEVICE_SHADOW* sub_device_shadow)
{
    printf("object_device_id = %s, request_id = %s\n", sub_device_shadow->object_device_id,
sub_device_shadow->request_id);
    printf("services_size = %d\n", sub_device_shadow->shadow_size);
    int i;
    for(i = 0; i< sub_device_shadow->shadow_size; i++) {
        printf("service_id = %s, properties = %s\n", sub_device_shadow->shadow[i].service_id,
sub_device_shadow->shadow[i].desired_properties);
    }
    return EDGE_SUCCESS;
}

/**
 * 描述：子设备请求响应的回调
 * 参数：
 *  getProductsRspEvent：获取到的子设备请求响应信息
 */
EDGE_RETCODE get_products_response_cb(ST_GET_PRODUCTS_RSP_EVENT* getProductsRspEvent)
{
```

```
    int i, j, k, l, m;
    for (i = 0; i < getProductsRspEvent->product_len; i++)
    {
        printf("%d-th product's info shows below:\n", i+1);
        printf("product_id:%s\n", getProductsRspEvent->products[i].product_id);
        printf("name:%s\n", getProductsRspEvent->products[i].name);
        printf("device_type:%s\n", getProductsRspEvent->products[i].device_type);
        printf("protocol_type:%s\n", getProductsRspEvent->products[i].protocol_type);
        printf("data_format:%s\n", getProductsRspEvent->products[i].data_format);
        printf("industry:%s\n", getProductsRspEvent->products[i].industry);
        printf("name:%s\n", getProductsRspEvent->products[i].name);
        printf("service_capabilities:\n");
        for (j = 0; j < getProductsRspEvent->products[i].service_capability_len; j++)
        {
            printf(" service_id:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].service_id);
            printf("service_type:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].service_type);
            printf("description:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].description);
            printf("option:%s\n", getProductsRspEvent->products[i].serviceCapabilities[j].option);
            printf(" properties:\n");
            for(k = 0; k < getProductsRspEvent->products[i].serviceCapabilities[j].property_len; k++)
            {
                printf("  property_name:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].property_name);
                printf("required:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].required ? "true" : "false");
                printf("data_type:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].data_type);
                printf("enum_list:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].enum_list);
                printf("min:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].min);
                printf("max:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].max);
                printf("max_length:%d, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].max_length);
                printf("step:%lf, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].step);
                printf("unit:%s, ", getProductsRspEvent->products[i].serviceCapabilities[j].properties[k].unit);
                printf("method:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].method);
                printf("description:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].description);
                printf("default_value:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].properties[k].default_value);
            }

            printf(" commands:\n");
            for(k = 0; k < getProductsRspEvent->products[i].serviceCapabilities[j].command_len; k++)
            {
                printf("  command_name:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].command_name);
                printf("  paras:\n");
                for(l = 0; l < getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].para_len; l++)
                {
                    printf("    para_name:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].para_name);
                    printf("required:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].required ? "true" : "false");
                    printf("data_type:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].data_type);
                    printf("enum_list:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].enum_list);
                    printf("min:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].min);
                    printf("max:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].max);
                    printf("max_length:%d, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].max_length);
                    printf("step:%lf, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].step);
                    printf("unit:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].unit);
```

```
                    printf("description:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].description);
                }

                printf("  responses:\n");
                for(l = 0; l < getProductsRspEvent->products[i].serviceCapabilities[j].commands[k].response_len; l
++)
                {
                    printf("   response_name:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].response_name);
                    printf("   paras:\n");
                    for(m = 0; m < getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].service_command_para_len; m++)
                    {
                        printf("    para_name:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].para_name);
                        printf("required:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].required ? "true" : "false");
                        printf("data_type:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].data_type);
                        printf("enum_list:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].enum_list);
                        printf("min:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].min);
                        printf("max:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].max);
                        printf("max_length:%d, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].max_length);
                        printf("step:%lf, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].step);
                        printf("unit:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].unit);
                        printf("description:%s\n", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].responses[l].paras[m].description);
                    }
                }
            }
        }
        printf("\n");
    }

    return EDGE_SUCCESS;
}
```

说明

以下是接口函数部分

```
/******************************  接口函数Demo  ***************************************/

/**
 * 上报子设备属性数据
 */
EDGE_RETCODE send_batch_device_data()
{
    /*
     * 具体上报的子设备数据和设备的产品模型有关系，和属性对应
     * */
    char* data_body1 = "{\n"
                       "\t\"PhV_phsA\":1,\n"
                       "\t\"PhV_phsB\":2\n"
                       "}";
    char* data_body2 =  "{\"PhV_phsA\":\"1\",\"PhV_phsB\":\"2\"}";
    printf("data_body1:%s\n", data_body1);
    printf("data_body2:%s\n", data_body2);

    ST_DEVICE_SERVICE device_data[2];
    ST_DEVICE_SERVICE devcie_data_1 = {0};
    ST_DEVICE_SERVICE devcie_data_2 = {0};
```

```
    ST_SERVICE_DATA service_data[2];
    ST_SERVICE_DATA service_data_1 = {0};
    service_data_1.service_id = "service_id_1";
    service_data_1.properties = data_body1;
    service_data_1.event_time = "20200520T115630Z";
    ST_SERVICE_DATA service_data_2 = {0};
    service_data_2.service_id = "service_id_2";
    service_data_2.properties = data_body2;
    service_data_2.event_time = "20200520T115630Z";
    service_data[0] = service_data_1;
    service_data[1] = service_data_2;

    devcie_data_1.services = service_data;
    devcie_data_1.device_id = "device_id_1";
    devcie_data_1.size = 2;

    devcie_data_2.services = service_data;
    devcie_data_2.device_id = "device_id_2";
    devcie_data_2.size = 2;
    device_data[0] = devcie_data_1;
    device_data[1] = devcie_data_2;

    ST_SUB_DEVICES_PROPERTIES_REPORT subDevicesPropertiesReport = {0};
    subDevicesPropertiesReport.device_size = 2;
    subDevicesPropertiesReport.devices = device_data;

    return edge_driver_report_sub_device_properties(&subDevicesPropertiesReport);
}

/**
 * 描述：获取设备影子事件示例
 */
void get_device_shadow()
{
    ST_DEVICE_SHADOW_GET device_shadow_get = {0};
    device_shadow_get.service_id = "service1";
    device_shadow_get.object_device_id = "deviceId1";
    char* request_id = "123";
    edge_driver_get_device_shadow(request_id, &device_shadow_get);
}

/**
 * 描述：获取产品示例
 * */
void get_products()
{
    char* product_ids[10] = {0};
    product_ids[0] = "1234";
    product_ids[1] = "1234";
    edge_driver_get_products(product_ids, 2);
}

/**
 * 描述：发送设备事件示例
 */
void send_sub_device_event()
{
    ST_DEVICE_EVENT device_event = {0};
    device_event.object_device_id = "deviceId1";
    ST_SERVICE_EVENT service_event = {0};
    service_event.service_id = "message_name";
    service_event.event_id = "message_id";
    service_event.event_type = "message_name";
    service_event.paras =  "{\n"
                        "\t\"module_id\": \"module_id\",\n"
                        "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
                        "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
                        "}";
    service_event.paras_len = (int)strlen(service_event.paras);
```

```
    device_event.services = &service_event;
    device_event.services_size = 1;

    edge_driver_send_device_event(&device_event);
}

/**
 * 描述：发送设备消息示例
 */
void send_sub_device_message()
{
    ST_DEVICE_MESSAGE device_message = {0};
    device_message.object_device_id = "deviceId1";
    device_message.name = "message_name";
    device_message.id = "message_id";
    device_message.content =  "{\n"
                          "\t\"module_id\": \"module_id\",\n"
                          "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
                          "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
                          "}";
    device_message.content_len = (int)strlen(device_message.content);
    edge_driver_send_device_message(&device_message);
}

/**
 * 描述：更新子设备状态方式示例
 * */
void update_sub_device_status(){
    /*
     * 构造 {"device_statuses":[{
     *          "device_id":"deviceId",
     *          "status":"status"
     *  }]}
     * */
    ST_DEVICE_STATUS device_status[2];
    ST_DEVICE_STATUS device_status_1 = {0};
    device_status_1.device_id = "device_id_1";
    device_status_1.status = "RUNNING";
    device_status[0] = device_status_1;
    ST_DEVICE_STATUS device_status_2 = {0};
    device_status_2.device_id = "device_id_2";
    device_status_2.status = "RUNNING";
    device_status[1] = device_status_2;

    const char* event_id = "event_id1";
    edge_driver_update_sub_devices_status(event_id, device_status, 2);
}

/**
 * 描述：同步子设备方式示例
 */
void sync_sub_devices()
{
    const char* event_id = "event_id1";
    long long version = 123;

    edge_driver_sync_sub_devices(event_id, version);
}

/**
 * 描述：发送服务事件方式示例
 */
void send_service_event()
{
    char* body = "{\n"
              "\t\"module_id\": \"module_id\",\n"
              "\t\"old_status\": \"STOPPED|RUNNING|UNHEALTHY\",\n"
              "\t\"new_status\": \"STOPPED|RUNNING|UNHEALTHY\"\n"
              "}";
```

 版权所有 © 华为云计算技术有限公司

```
    edge_driver_send_service_event("module_management", "module_status_change", body, strlen(body));
}
```

说明

下面是Modbus Demo和其他Demo，里面包括调用上面所有的回调函数和接口函数，用不到的地方注释释掉即可。

```
/******************************  Modbus Demo和其他Demo ***************************************/
/**
 * modbus demo
 */

void modbus_driver_demo()
{
    EDGE_RETCODE ret;

    // 设置定时任务
    while(1)
    {
        ret = send_batch_device_data();
        if (EDGE_SUCCESS != ret)
        {
            printf("failed to report sub device data.\n");
        }
        // 每1000秒重复上报数据
        sleep(1000);
    }
}

/**
 * 其他demo，可根据需要自行选择
 */
void other_demo()
{
    edge_driver_get_shadow();

    send_service_event();
    sleep(5);

    get_device_shadow();
    sleep(1);
    get_products();
    sleep(1);
    send_sub_device_event();
    sleep(1);
    send_sub_device_message();
    sleep(1);
    update_sub_device_status();
    sleep(1);
    sync_sub_devices();

    // 这里是为了使应用能够长时间运行
    while (1)
    {
        sleep(1000);
    }
}

void driver_demo() {
    // 禁用缓冲区
    setvbuf(stdout, NULL, _IONBF, 0);

    //初始化sdk，工作路径设置（工作路径下需要含有 /conf  目录（该目录下包含证书等信息））
    edge_driver_init("../code/api_test/workdir");
    printf("driver demo start.\n");

    // 请根据需要选择相应的回调处理函数
    ST_GATEWAY_CBS gateway_cbs = {0};
```


版权所有 © 华为云计算技术有限公司

```
    gateway_cbs.pfn_sub_device_add_cb_1 = sub_device_add_cb_1;
    gateway_cbs.pfn_sub_device_deleted_cb_1 = sub_device_delete_cb_1;
    gateway_cbs.pfn_device_command_cb = deviceCmdCb;
    gateway_cbs.pfn_device_event_cb = sub_device_event_cb;
    gateway_cbs.pfn_on_start_scan_cb = sub_device_start_scan_cb;
    gateway_cbs.pfn_device_message_cb = sub_device_messages_down_cb;
    gateway_cbs.pfn_on_get_products_rsp_cb = get_products_response_cb;

    gateway_cbs.pfn_device_properties_set_cb = sub_device_properties_set_cb;
    gateway_cbs.pfn_device_properties_get_cb = sub_device_properties_get_cb;
    gateway_cbs.pfn_device_shadow_cb = sub_device_shadow_cb;

    //设置网关回调函数（无需全部设置，按需设置需要接收的回调）
    edge_driver_set_gateway_callback(&gateway_cbs);

    //连接hub
    edge_driver_login();
    sleep(10);

    // 自定义demo, 请根据需要替换或修改
    // 如果不需要，请注释掉
    modbus_driver_demo();

    // 如果不需要，请注释掉
    other_demo();

    edge_driver_logout();
    sleep(1000);
    edge_driver_destroy();
}

int main()
{
    driver_demo();
    return 0;
}
```

4.4.5.3 Demo3

Demo代码中具体实现的是模拟电机设备上报数据，SDK获取上报数据做进一步分析处理。

如果遇到状态为error，则调用事先在产品模型定义好的设备命令。对于未指定MOTOR_PRODUCT_ID的产品上报的数据将继续上报给云端。

说明

该Demo的应用可参考**集成ModuleSDK进行数据处理**。

```
#include "edge.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>

/*
 * 描述：设置总线消息回调，用于对设备上报的数据进行处理
 * 参数：
 *   input_name：消息总线输入点
 * */
EDGE_RETCODE set_bus_message_cb(const char* input_name)
{
    edge_set_bus_message_cb(input_name);
    printf("set bus message callback with input name: %s\n", input_name);

    return EDGE_SUCCESS;
}
```

```
/*
 * 描述：收到设备上报数据的回调处理，样例代码在马达设备状态错误时对马达进行重启
 * 参数：
 *  device_id：设备ID
 *  product_id: 产品ID
 *  body: 上报的数据
 *  body_len: 上报数据的大小
 * */
EDGE_RETCODE on_message_received_cb(const char* device_id, const char* product_id, const char* body,
unsigned int body_len)
{
    // 设置发送设备数据的消息总线输出点，取值需在创建应用版本的outputs参数中定义
    char* output_name = "output";

    printf("start send message to output topic: %s\n", output_name);
    printf("body is: %s\nbody len is: %d\n", body, body_len);
    printf("product_id is: %s\n", product_id);
    printf("start processing device.\n");

    // 设置电机设备产品ID
    char* MOTOR_PRODUCT_ID = "product_123";

    if (strcmp(product_id, MOTOR_PRODUCT_ID) == 0)
    {
        //马达设备状态错误时对马达进行重启
        char* error = "error";
        char* is_error = strstr(body, error);

        // 设置默认超时时间
        unsigned int timeout = 5;
        ST_COMMAND command = {0};
        command.object_device_id = device_id;
        command.service_id = "power";
        command.command_name = "restart";

        // 调用设备命令重启
        if (is_error != NULL) edge_call_device_command(&command, timeout);
    }
    else {
        //其他设备数据发送到消息总线
        edge_send_bus_message(output_name, body, body_len);
    }

    printf("process ended.\n");

    return EDGE_SUCCESS;
}

/*
 * 监控APP，检视设备上报的数据，并对设备进行相应的控制
 */

void monitor_app()
{
    // 禁用缓冲区
    setvbuf(stdout, NULL, _IONBF, 0);

    printf("start monitor app\n");

    //初始化sdk，工作路径设置（工作路径下需要含有 /conf  目录（该目录下包含证书等信息））
    edge_init("../code/api_test/workdir");

    ST_MODULE_CBS module_cbs = {0};
    ST_DEVICE_CBS device_cbs = {0};

    module_cbs.pfn_on_message_received_cb = on_message_received_cb;

    // 设置回调函数
    edge_set_callbacks(&module_cbs, &device_cbs);
```

 版权所有 © 华为云计算技术有限公司

```
    printf("SDK start running!\n");

    sleep(1);
    edge_login();

    sleep(1);

    // 接收设备数据的消息总线输入点，取值需在创建应用版本的inputs参数中定义
    char* input_name = "input";
    set_bus_message_cb(input_name);

    // 这里是为了使应用能够长时间运行
    while(1)
    {
        sleep(1000);
    }

    edge_logout();
    sleep(1000);
    edge_destroy();
}

int main()
{
    // 监控app demo
    monitor_app();

    return 0;
}
```

4.5 集成 ModuleSDK(C#)

4.5.1 内部架构

表 4-85 通用接口

接口	说明
IModuleShadowCallback	模块影子回调，实现对影子通知的处理
IPointsCallback	点位处理回调，实现点位的读写操作
IConnectionStatusChangeCallback	连接状态变化回调接口，用于MQTT连接状态改变时的处理

表 4-86 提供的类

类	说明
DcClient	数采客户端，用于开发数采驱动接入点位数据，驱动需实现相关回调函数

4.5.2 开发指导

4.5.2.1 方案概述

基于ModuleSDK开发应用实现数据处理或自定义驱动时，分为开发和使用两个部分。

开发操作	开发说明
App应用的开发	利用DcClient进行自定义的业务处理
App应用的使用	将应用打包上传至云，部署到节点，连接子设备查看应用工作状态

4.5.2.2 前提条件

- 开发环境要求：安装**.Net**开发环境（仅支持.NET and .NET Core 2.0及以上版本，建议使用.Net 6.0版本）。
- 开发工具：**Visual Studio** 或者 **Rider** 。

4.5.2.3 创建工程

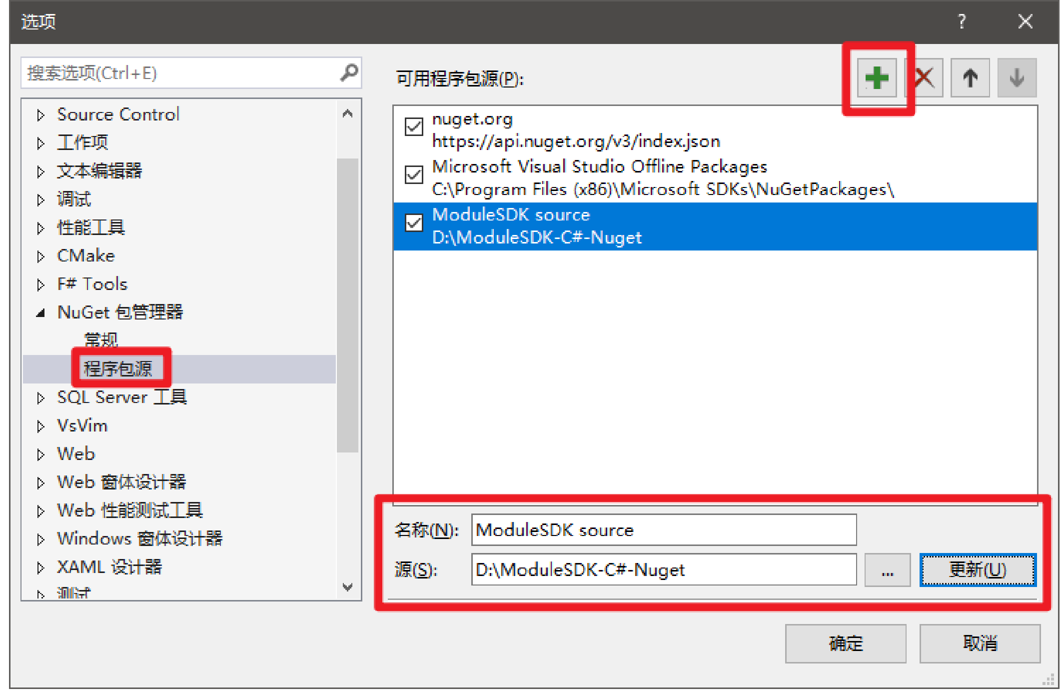
本文以 Visual Studio 2017 作为项目开发IDE。

步骤1 **新建工程**

打开“Visual Studio > 文件 > 新建 > 项目 > 选择新建.Net Core控制台应用”，这里创建一个名为**ModuleSDK-Demo**的工程进行下面的实践。

步骤2 **添加本地Nuget包源**

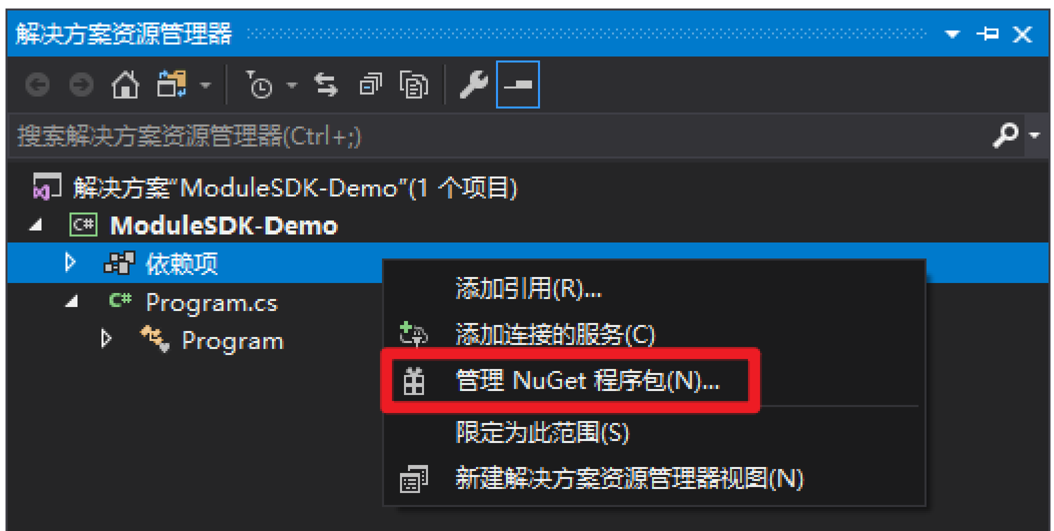
打开“Visual Studio > 工具 > Nuget包管理器 > 程序包管理器设置”，在弹出的窗口中新增一个可用程序包源，"源"为ModuleSDK对应的“NuGet包”所在的本地路径。



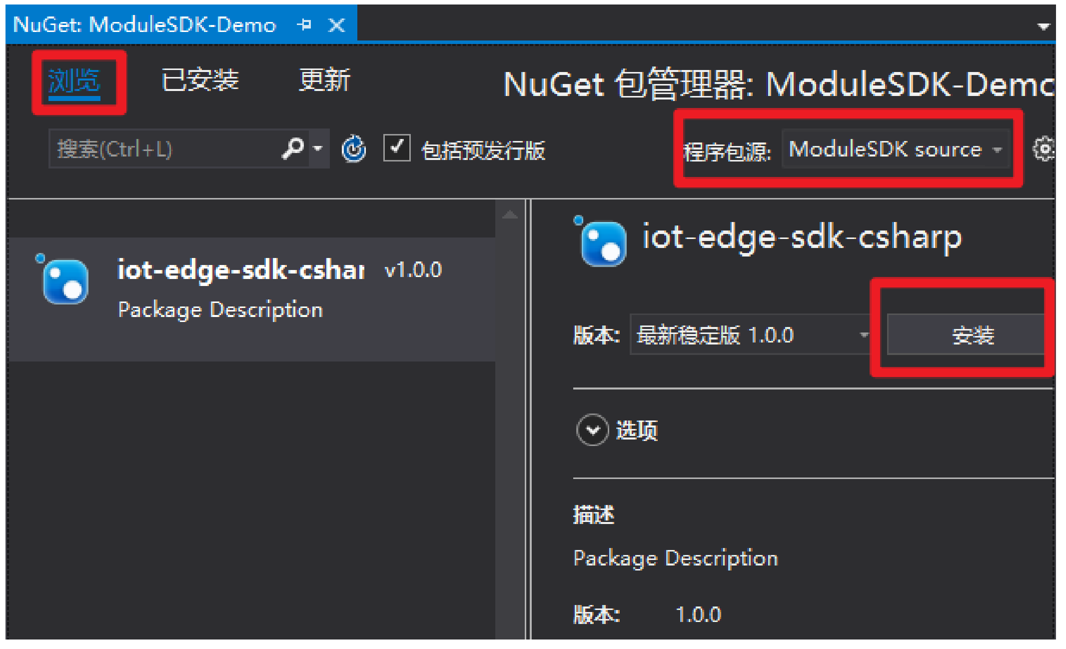
步骤3 **为工程导入Nuget包**

版权所有 © 华为云计算技术有限公司

在Visual Studio的解决方案资源中，右键单击"依赖项"，选择"管理Nuget程序包"。



选择上一步添加的本地包源作为"程序包源"，然后选择"浏览"选项卡，在列表中选择该包，然后单击安装。



注意

如果需要升级SDK，可参考**步骤2**将新版本ModuleSDKd的NuGet包放入同一目录下，待NuGet包管理器自动识别后，根据界面提示进行SDK更新操作即可。

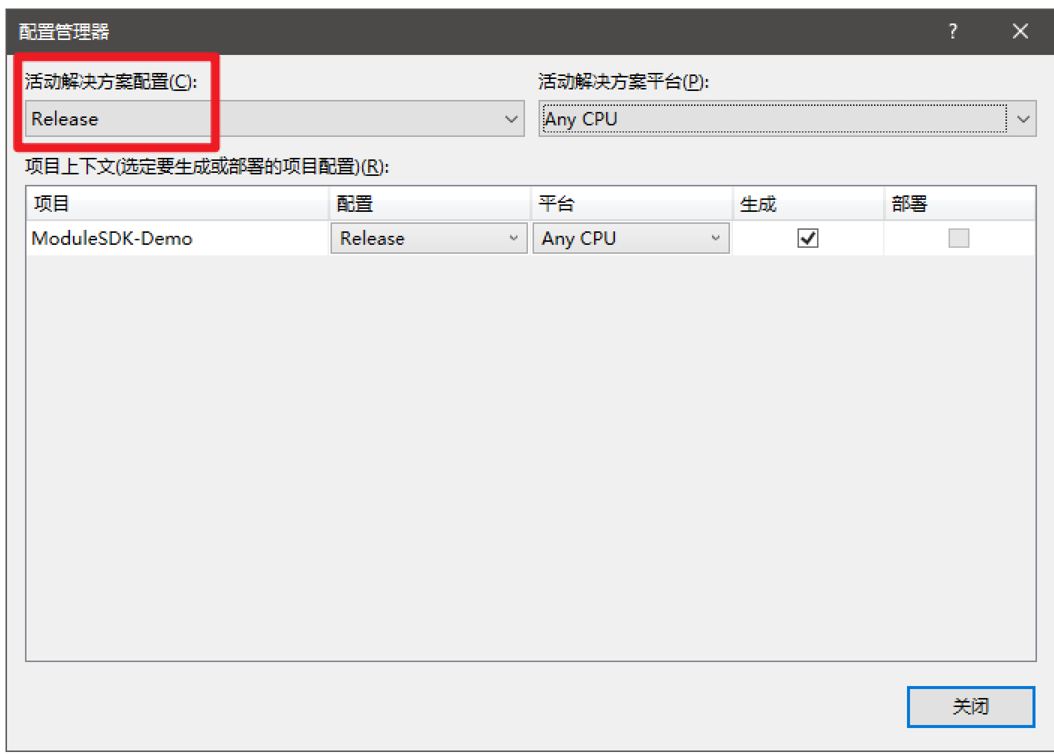
----结束

4.5.2.4 项目构建发布

步骤1 修改构建类型

版权所有 © 华为云计算技术有限公司

“Visual Studio > 生成 > 配置管理器”，将"活动解决方案配置"选择为"Release"。



步骤2 **项目发布**

“Visual Studio > 生成 > 发布 ModuleSDK-Demo”。

单击配置按钮，将弹出"配置文件设置"窗口，可根据您实际使用的开发环境进行配置。

发布

将应用发布到 Azure 或另一台主机。 了解更多

FolderProfile 发布(U)

新建配置文件... 操作

目标位置 bin\Release\netcoreapp... 配置...

配置 Release

目标框架 netcoreapp2.1

目标运行时 可移植

配置文件设置

配置文件名称: FolderProfile

配置: Release | Any CPU

目标框架: netcoreapp2.1

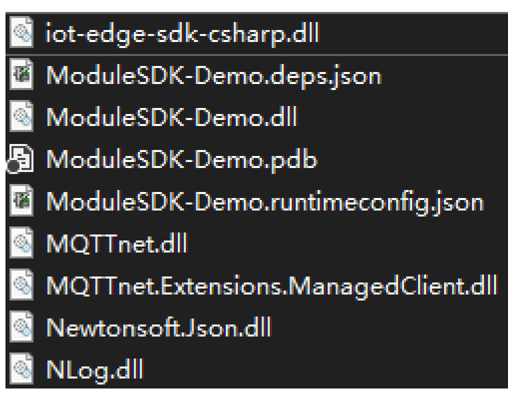
部署模式: 框架依赖

目标运行时: 可移植

目标位置: bin\Release\netcoreapp2.1\publish\ ...

保存 取消

配置完成后单击"发布"按钮，即可在您设定的目标位置看到类似下图的发布文件



----结束

版权所有 © 华为云计算技术有限公司

4.5.2.5 制作镜像包或插件包

注意

若制作镜像包以容器化方式部署应用，不支持在一个容器内运行多个集成ModuleSDK的软件进程或者重启集成ModuleSDK的软件进程，会导致鉴权失败等问题。

镜像包打包

步骤1　构建需要打包的项目

构建项目，此处示例，构建好的文件存放在目录 /home/ModuleSDK-Demo 下

步骤2　安装docker

请确认您使用的系统已经安装Docker(Docker版本需要高于17.06，推荐18.06)，安装方法可参照**docker 安装教程**。

步骤3　制作镜像

1. 搜索基础镜像。

```
docker search dotnet
```

2. 根据您的工程配置，选择合适的镜像。

注意

镜像需要集成与您的工程配置兼容的.Net运行环境，下文使用的.Net版本仅作演示。

3. 拉取镜像。

```
docker pull mcr.microsoft.com/dotnet/runtime:2.1
```

说明

mcr.microsoft.com/dotnet/runtime:2.1 镜像是微软提供的镜像，非IoT团队发布，且IoT团队未提供任何官方镜像。该镜像在此仅做示例，IoT团队对该镜像的安全性不作保证，强烈建议用户自己封装镜像！

4. 编写 Dockerfile 制作镜像。

创建 Dockerfile，内容示例如下（具体可参考**编写高效的Dockerfile** ）。

须知

下面提供了ModuleSDK-Demo镜像构建样例，仅作示例展示，请按需修改。

```
# 基础镜像来源
FROM mcr.microsoft.com/dotnet/runtime:2.1

# 指定工作目录
WORKDIR /app

# 复制工程二进制文件和相关文件（即项目构建发布的产物）
COPY ModuleSDK-Demo/ /app

ENTRYPOINT ["dotnet", "ModuleSDK-Demo.dll"]
```


版权所有 © 华为云计算技术有限公司

5. 构建镜像
```
docker build -t modulesdk-demo:1.0.0 -f Dockerfile .
```
6. 查看打包完成的镜像
```
docker images
```
可以看到**modulesdk-demo**这个镜像已经制作完成。
```
REPOSITORY       TAG        IMAGE ID        CREATED         SIZE
modulesdk-demo   1.0.0    85ed3c3dc738   8 minutes ago    182MB
```

说明

上述步骤演示的是直接复制已编译好的工程文件来构建镜像，您也可以采取在构建镜像时编译的方式，具体可参照**.Net 官方文档**的指引。

步骤4 镜像上传

1. 上传镜像

   镜像上传需要使用镜像容器服务(SWR)，首先需要开通容器镜像服务(SWR)。开通及使用请参照**容器镜像服务(SWR)**。

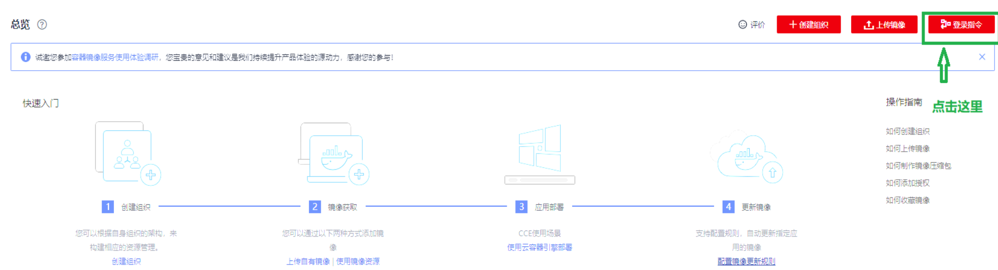
2. 获取 SWR 登录指令

   获取登录指令请参照**获取指令**。

   说明

   访问密钥即AK/SK（Access Key ID/Secret Access Key），获取的AK/SK将用于登录。

3. 登录 SWR 仓库
```
docker login -u [区域项目名]@[AK] -p [登录密钥] [镜像仓库地址]
```
   可以直接从**控制台**获取登录命令，如下图。



   说明

   镜像仓库地址 = swr.区域项目名称.myhuaweicloud.com

   例如，华北-北京一对应的镜像仓库地址为：swr.cn-north-1.myhuaweicloud.com

4. 修改镜像所属组织

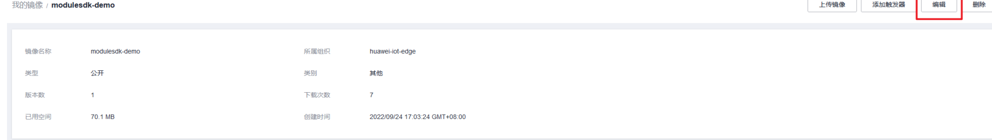
   修改镜像的组织名，以便推送到个人组织内。
```
docker tag [OPTIONS] [镜像名:版本号] [镜像仓库地址/所属组织/镜像名:版本号]
```
   例如：
```
docker tag modulesdk-demo:1.0.0 swr.cn-north-4.myhuaweicloud.com/iotedge/modulesdk-demo:1.0.0
```
5. 上传镜像
```
docker push [镜像仓库地址/所属组织/镜像名:版本号]
```
   例如：
```
docker push swr.cn-north-4.myhuaweicloud.com/iotedge/modulesdk-demo:1.0.0
```
6. 在**我的镜像**查看上传结果

版权所有 © 华为云计算技术有限公司

7. 上传镜像后请在 SWR 将镜像设置为公开。

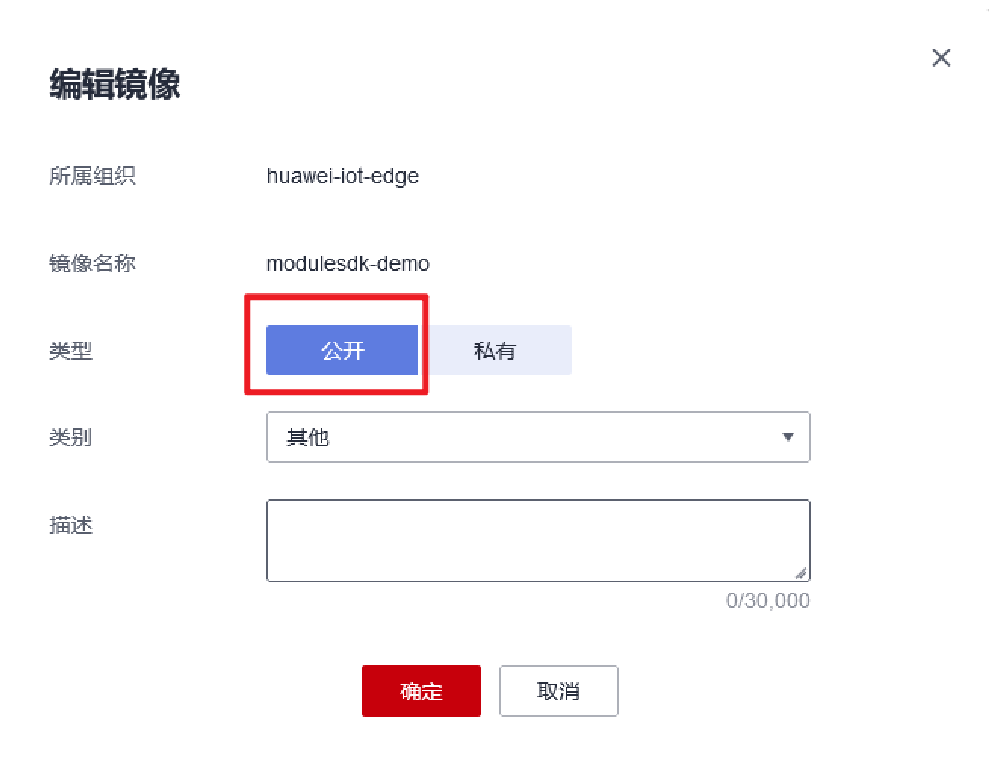
   查看镜像详情：

   编辑镜像：



   设置为公开：

> **注意**
>
> 这一步很重要，关系到后面能否正常部署应用。



----结束

版权所有 © 华为云计算技术有限公司

插件包打包

步骤1 插件包制作

1. **插件包格式要求**

   插件包仅支持.tar.gz 、.tar或者 .zip格式。

   插件包结构如下：

   ModuleSDK-Demo.zip
   ├── ModuleSDK-Demo.*** // 可执行文件
   ├── ***.dll // 外部库文件，在构建发布时产生，可能会有多个
   ├── start.sh // 启动脚本 必须文件 当前不提供参数方式启动
   └── stop.sh // 停止脚本 非必须

   说明

   ModuleSDK-Demo.*** 为可执行文件，类型可能为.dll或.exe，取决于您的工程发布配置。下文将以ModuleSDK-Demo.dll为例。

2. **添加启动脚本**

   在可执行文件ModuleSDK-Demo.dll的同一目录下创建启动脚本start.sh，内容示例如下，可根据您的需要进行修改。

```
function log(){
    echo "`date "+%Y-%m-%d %T"`: $1"
}
log "[INFO] start execut process."
pwd
dotnet ModuleSDK-Demo.dll > ModuleSDK_demo_running.log 2>&1
```

3. **打包插件包**

   压缩文件得到ModuleSDK-Demo.zip。

步骤2 插件包上传

1. **开通对象存储服务OBS**

   进程包上传方式需要开通对象存储服务OBS，请参考**对象存储服务 OBS_快速入门**

2. **上传进程包**

   上传方式请参照**对象存储服务(OBS)**。

注意

请设置桶策略为【公开读】，如未设置请前往“OBS首页 > 单击桶ID > 访问权限控制 > 桶策略中设置”。

----结束

4.5.2.6 添加应用

添加边缘应用具体请参考**添加应用**。

说明

提供两种不同的部署方式，请根据需要选择合适的部署方式。

容器化部署

步骤1　软件部署配置

部署方式选择容器化部署，选择上传到SWR服务的镜像，如未发现镜像，请检查镜像是否为公开，设置镜像为公开方式：容器镜像服务SWR->我的镜像->单击镜像ID进入详情->右上方编辑。

容器规格与高级配置请根据您的需要进行配置。

图 4-14 容器化部署配置

 版权所有 © 华为云计算技术有限公司

表 4-87 部署配置

参数名称	说明
部署方式	容器化部署：在边缘节点上应用通过Docker容器运行；添加容器化部署的应用需先确保容器镜像已上传到**容器镜像服务(SWR)**，且设置其类型为“公开”，例如**图4-14**所示。 安装包部署：在边缘节点是以进程形式运行；添加安装包部署的应用需先确保安装包已按照打包规范进行压缩打包，并上传到**对象存储服务(OBS)**。
安装包部署	部署方式为安装包部署时，填写您此前上传到对象存储服务OBS中的安装包地址。
容器镜像	部署方式为容器化部署时，参考**表4-88**填写。

表 4-88 选择镜像输入方式

参数名称	说明
手动输入	输入镜像地址：统一在**容器镜像服务(SWR)**中管理。
单击选择	选择镜像：选择需要部署的镜像，单击“确认”。 ● 我的镜像：展示了您在容器镜像服务中创建的所有镜像。 ● 他人共享：展示了其他用户共享的镜像。 版本：选择需要部署的镜像版本。

步骤2 **运行配置**

输入端点、输出端点与demo中代码定义的端点一一对应，由于本例演示的是数采模块，因此不对端点进行配置。

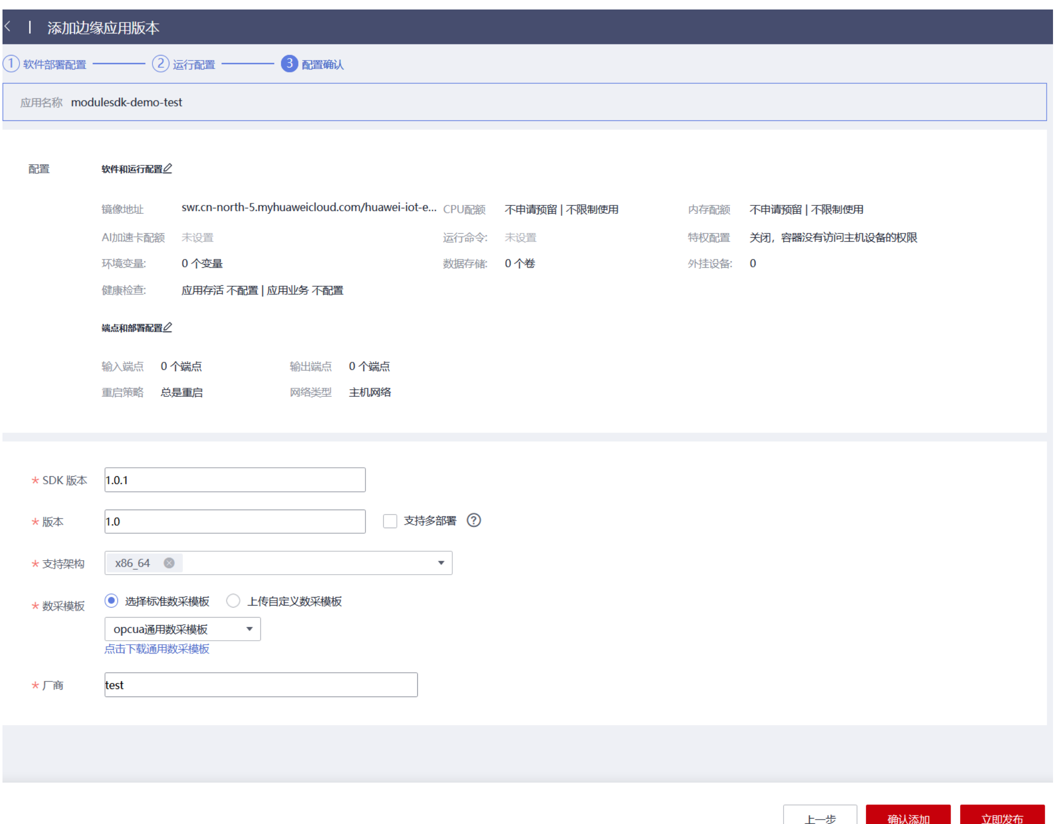
部署配置请根据您的需要进行选择。

 版权所有 © 华为云计算技术有限公司

说明

输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

步骤3 配置确认



📖 **说明**

建议直接单击“立即发布”，方便后面直接部署应用的时候，能够获取到最新版本。

----结束

安装包部署

步骤1 **软件部署配置**

部署方式选择安装包部署，"安装包地址"为{桶名/对象名}。如桶名为:“Demo”，对象名为:“ModuleSDK-Demo.zip”， 则安装包为obs://Demo/ModuleSDK-Demo.zip。

容器规格与高级配置请根据您的需要进行配置。

步骤2 **运行配置**

输入端点、输出端点与demo中代码定义的端点一一对应，由于本例演示的是数采模块，因此不对端点进行配置。

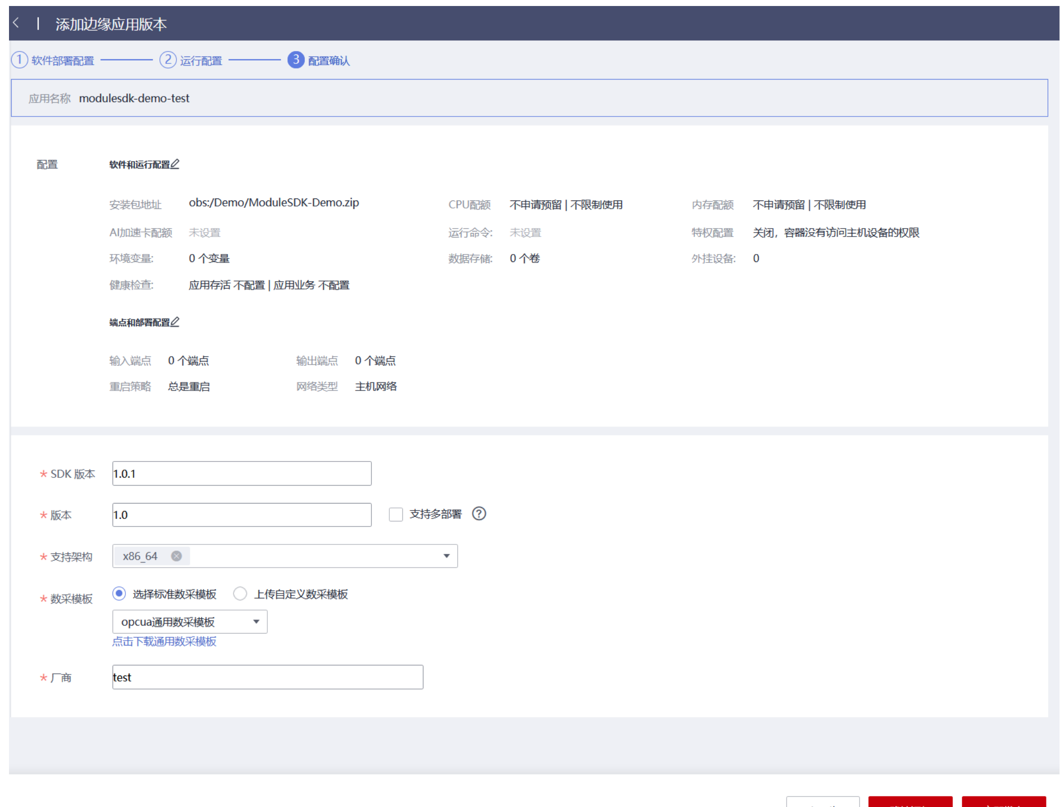
部署配置请根据您的需要进行选择。

版权所有 © 华为云计算技术有限公司

说明

输入输出端点是非必需配置的，当有数据流转时需要配置，如OT应用（数据处理）。驱动类应用和IT应用一般不需要配置。

步骤3 配置确认



> 📖 **说明**
>
> 建议直接单击“立即发布”，方便后面直接部署应用的时候，能够获取到最新版本。

----结束

4.5.2.7 发布应用

应用创建之后需要发布才允许在节点部署。若您在上一节已经单击"立即发布"，则可以跳过本节的内容。

操作步骤

步骤1 访问**IoT边缘**，单击“**管理控制台**”进入IoT边缘控制台。

步骤2 选择左侧导航栏，选择“应用名称”进入应用详情页。

步骤3 单击右上角“发布”按钮。

> 📖 **说明**
>
> 可在创建应用时可勾选【立即发布】进行发布。

----结束

4.5.2.8 接口方法

表1 DcClient支持的接口方法

接口	说明
CreateFromEnv	创建客户端 ，执行初始化操作
Open	打开客户端，开始消息收发
Close	关闭客户端，清理资源
ForbidSendWhenBrokerOverLoaded	设置是否在离线缓存达到阈值时禁止发送数据
SendMode	设置上报模式(同步|异步)

版权所有 © 华为云计算技术有限公司

接口	说明
OnPointGet	用于网关处理点位读取请求
OnPointSet	用于网关处理点位设置请求
OnModuleShadowReceived	用于网关处理模块影子，接收下行数采配置
StartModuleShadow	启动模块影子，设置模块影子回调，并触发获取影子的动作
GetModuleShadow	主动获取一次模块影子
GetDevicesStatuses	获取子设备状态
PointReport	点位上报
DataSourceConnStateReport	连接状态上报

1. 创建客户端

方法描述：

```
static DcClient CreateFromEnv();
```

方法功能：

根据环境变量创建客户端，返回客户端实例。

2. 打开客户端

方法描述：

```
void Open();
```

方法功能：

打开客户端，使用客户端其他功能前必须先调用该方法，才能正常收发消息。

3. 打开客户端

方法描述：

```
void Close();
```

方法功能：

关闭客户端，清理资源，程序将退出时调用该方法。

4. 点位设置请求处理

方法描述：

```
PointsSetRsp OnPointsSet(string requestId, PointsSetReq pointsSetReq);
```

方法功能：

点位设置回调，用于网关处理点位设置请求。

表 4-89 参数说明

名称	类型	描述
requestId	string	用于唯一标识某次请求的请求ID
pointsSetReq	PointsSetReq	点位设置请求的对象，该对象结构见下表
@return	PointsSetRsp	点位设置响应，网关处理完请求以后，需要返回该响应，该对象结构见下表

表 4-90 PointsSetReq 结构说明

名称	类型	描述
Points	Dictionary<string, object>	按<点位ID，点位对象>存储点位数据的字典

表 4-91 PointsSetRsp 结构说明

名称	类型	描述
ResultCode	int	点位设置请求结果码
ResultDesc	string	点位设置请求结果描述

5. 点位读取请求处理

方法描述：

```
PointsGetRsp OnPointsGet(string requestId, PointsGetReq pointsGetReq);
```

方法功能：

点位读取回调，用于网关处理点位读取请求。

版权所有 © 华为云计算技术有限公司

表 **4-92** 参数说明

名称	类型	描述
requestId	string	用于唯一标识某次请求的请求ID
pointsGetReq	PointsGetReq	点位读取请求的对象，该对象结构见下表
@return	PointsGetRsp	点位读取响应，网关处理完请求以后，需要返回该响应，该对象结构见下表

表 **4-93** PointsGetReq 结构说明

名称	类型	描述
Points	List<string>	按[点位ID]存储需要读的点位列表

表 **4-94** PointsGetRsp 结构说明

名称	类型	描述
Points	Dictionary<string, object>	按<点位ID，点位对象>存储点位读取结果的字典

6. 模块影子下行处理

方法描述：

```
void OnModuleShadowReceived(ModuleShadow shadow);
```

方法功能：

模块影子下行回调，用于网关接收模块数采配置信息，可根据需要对模块影子进行缓存或持久化。

版权所有 © 华为云计算技术有限公司

表 4-95 参数说明

名称	类型	描述
shadow	ModuleShadow	模块影子

表 4-96 ModuleShadow 结构说明

名称	类型	描述
Properties	Dictionary<string, object>	按<影子属性名，影子属性对象>存储的字典
PropertiesUpdateTime	Dictionary<string, long>	按<影子属性名，更新时间戳>存储的字典，用户可根据更新时间进行增量同步

7. StartModuleShadow

方法描述：

```
void StartModuleShadow(IModuleShadowCallback callback);
```

方法功能：

启动模块影子，设置模块影子用户回调，并主动触发一次模块影子的获取。SDK在收到下行的模块影子后将调用用户回调。

表 4-97 参数说明

名称	类型	描述
callback	IModuleShadowCallback	用户回调

表 4-98 IModuleShadowCallback 说明

方法签名	描述
void OnModuleShadowReceived(ModuleShadow shadow)	收到模块影子回调

8. GetModuleShadow

方法描述：

```
void GetModuleShadow();
```

方法功能：

主动获取一次模块影子，获取成功后将触发用户回调。

9. GetDevicesStatuses

方法描述：

```
DevicesStatuses GetDevicesStatuses(List<string> deviceIds, int timeout);
```

方法功能：

获取子设备状态。

表 4-99 参数说明

名称	类型	描述
deviceIds	List<string>	要查询的子设备ID列表
timeout	int	超时，范围1~30000(单位ms)
@return	DevicesStatuses	查询到的子设备状态列表

表 4-100 DevicesStatuses 结构说明

名称	类型	描述
Devices	List<DeviceStatus>	存储设备状态信息的列表

 版权所有 © 华为云计算技术有限公司

表 4-101 DevicesStatus 结构说明

名称	类型	描述
DeviceId	string	设备ID
Status	string	设备状态

10. PointReport

方法描述：

```
void PointReport(PointsReport pointsReport);
```

方法功能：

点位上报。

表 4-102 参数说明

名称	类型	描述
pointsReport	PointsReport	要上报的点位数据

表 4-103 PointsReport 结构说明

名称	类型	描述
Points	Dictionary<string,object>	以<点位ID，点位数据对象>存储的字典

11. DataSourceConnStateReport

方法描述：

```
void DataSourceConnStateReport(DataSourceConnState dataSourceConnState);
```

方法功能：

上报连接状态。

版权所有 © 华为云计算技术有限公司

表 4-104 参数说明

名称	类型	描述
dataSourceConnState	DataSourceConnState	存储连接状态的对象

表 4-105 DataSourceConnState 结构说明

名称	类型	描述
State	string	连接状态
Info	string	状态描述

12. ForbidSendWhenBrokerOverLoaded

属性描述：

```
bool ForbidSendWhenBrokerOverLoaded { get; set; };
```

属性功能：

设置是否在离线缓存满时禁止发送数据，默认值为false。当EdgeHub离线缓存达到阈值时，若设定为true，在属性上报和点位上报发送数据时，会抛出HubOverloadedException异常。

13. SendMode

属性描述：

```
ClientConfig.SendMode SendMode { get; set; };
```

属性功能：

设置上报模式，默认为异步发送，若设定为同步发送，则上报数据时将同步等待3秒，超时则抛出TransportException异常。

表 4-106 SendMode 说明

名称	描述
SendMode.Async	异步发送
SendMode.Sync	同步发送

4.5.3 下载 Demo

可参考创建工程的步骤创建新的工程，也可以下载**Demo**，解压并导入示例代码。

版权所有 © 华为云计算技术有限公司

4.5.4 集成 ModuleSDK 进行数据采集

4.5.4.1 场景说明

开发应用驱动集成ModuleSDK进行OT数采。（此示例以采集OPCUA为示例）

1. 在节点部署集成了ModuleSDK应用驱动。
2. 配置好数据源模板（可自定义）。
3. 在平台进行数据源配置，以及点位配置下发。
4. 集成ModuleSDK应用驱动对平台下发的配置进行处理。
5. 通过下发信息获取数据源连接，以及点位信息进行数据周期采集。
6. 最后运用ModuleSDK的客户端进行点位数据周期上报。

4.5.4.2 代码解析

开发自定义驱动，进行OT数采。

```
using IoT.SDK.Edge.Interface;
using IoT.SDK.Edge.Client;
using IoT.SDK.Edge.Utils;
using IoT.SDK.Edge.Dto;
using System.Collections.Generic;

namespace ModuleSDK_Demo
{
    public class DcDriver : IPointsCallback, IModuleShadowCallback
    {
        private DcClient dcClient;
        public DcDriver()
        {
            // 创建数采客户端
            dcClient = DcClient.CreateFromEnv();
        }
        public void Start()
        {
            // 打开数采客户端
            dcClient.Open();
            // 设置点位读写回调
            dcClient.IPointsCallback = this;
            // 同步模块影子
            dcClient.StartModuleShadow(this);
        }
        /*
         * 收到点位读取请求的处理
         */
        public PointsGetRsp OnPointGet(string requestId, PointsGetReq pointsGetReq)
        {
            // TODO 伙伴需要根据OnModuleShadowReceived获取的数采配置实现读取opcua服务器的点位信息
            // PointsGetReq的Points属性结构为[pointId1,pointId2,...]的列表
            // 此处示例，读取到的点位数据均为1
            PointsGetRsp response = new PointsGetRsp();
            foreach(string pointId in pointsGetReq.Points)
            {
                response.Points.TryAdd(pointId, 1);
            }
            return response;
        }
        /*
         * 收到点位设置请求的处理
         */
        public PointsSetRsp OnPointSet(string requestId, PointsSetReq pointsSetReq)
        {
```

```
            // TODO 伙伴需要根据OnModuleShadowReceived获取的数采配置实现写opcua服务器的点位信息
            // PointsSetReq的Points属性结构为[pointId:value]的键值对
            // 此处示例，直接返回成功响应
            return new PointsSetRsp(0, "success");
        }
        /*
         * 模块影子回调，收到模块下行数采配置，消息需要缓存或持久化
         * 进入边缘节点详细->数采配置->下发配置
         */
        public void OnModuleShadowReceived(ModuleShadow shadow)
        {
            // 伙伴需要对影子进行缓存或持久化，可根据影子属性的更新时间进行增量同步
            var briefModuleShadow = JsonUtil.FromJson<BriefModuleShadowDto>(
                JsonUtil.ToJson(shadow.Properties));
            // 此处示例，只要收到模块影子，则重新连接数据源，再收集数据并主动上报
            ConnectDatasource(briefModuleShadow.ConnectionInfo);
            CollectAndReportData(briefModuleShadow.Points);
        }
        /*
         * 收集数据并上报
         */
        public void CollectAndReportData(Dictionary<string, object> points)
        {
            var reportData = new Dictionary<string, object>();
            foreach (string key in points.Keys)
            {
                PointConfig pointConfig = JsonUtil.FromJson<PointConfig>(
                    JsonUtil.ToJson(points.GetValueOrDefault(key)));
                // 伙伴可根据pointConfig中的信息读取点位数据
                // 此处示例，读取到的点位数据为10
                object value = 10;
                reportData.TryAdd(key, value);
            }
            // 调用数采应用端的接口上报数据
            PointsReport pointsReport = new PointsReport(reportData);
            dcClient.PointReport(pointsReport);
        }
        /*
         * 根据数采配置的数据源连接参数完成数据源连接
         */
        private void ConnectDatasource(Dictionary<string, string> connectionInfo)
        {
            // 以Opcua服务器为例，获取服务器连接地址
            string endPoint = connectionInfo.GetValueOrDefault("endpoint");
            // 伙伴根据endpoint实现连接数据源动作
        }
    }
}
```

下发配置对象

```
    public class BriefModuleShadowDto
    {
        // 数据源id
        [JsonProperty("ds_id")]
        public string DsId { get; set; }
        // 数采模板默认参数
        [JsonProperty("default_values")]
        public Dictionary<string, string> DefaultValues { get; set; }
        // 数据源附加参数
        [JsonProperty("collection_paras")]
        public Dictionary<string, int> CollectionParas { get; set; }
        // 数据源连接信息
        [JsonProperty("connection_info")]
        public Dictionary<string, string> ConnectionInfo { get; set; }
        // 点位信息
        [JsonProperty("points")]
        public Dictionary<string, object> Points { get; set; }
    }
```


版权所有 © 华为云计算技术有限公司

点位信息对象

```
public class PointConfig
{
    // 点位地址，opcua地址：address = "ns=3;i=1002"
    [JsonProperty("address")]
    public string Address { get; set; }
    // 数据类型，int、int32、float、double、bool等
    [JsonProperty("data_type")]
    public string DataType { get; set; }
    // 点位采集周期单位毫秒
    [JsonProperty("cycle")]
    public int Cycle { get; set; }
    // 点位更新时间
    [JsonProperty("update_time")]
    public long UpdateTime { get; set; }
}
```

4.5.4.3 注册节点

注册节点请参照注册边缘节点。

4.5.4.4 设备建模&发放

请参照设备建模&发放。

4.5.4.5 项目打包

参考项目构建发布，将集成ModuleSDK进行项目打包。

4.5.4.6 制作镜像包或插件包

请参照制作镜像包或插件包。

4.5.4.7 添加应用

以进程部署方式为例，镜像包上传到对象存储服务OBS后。

1. 访问**IoT边缘**，单击“**管理控制台**”进入IoT边缘控制台。
2. 选择左侧导航栏“边缘节点 > 应用管理”进入页面，选择驱动应用，单击“添加应用”。

 版权所有 © 华为云计算技术有限公司

3. 在“添加驱动应用”弹窗中填写相关信息，然后单击“确认”。

4. 前往应用版本配置界面

5. 添加边缘应用-端点和部署配置

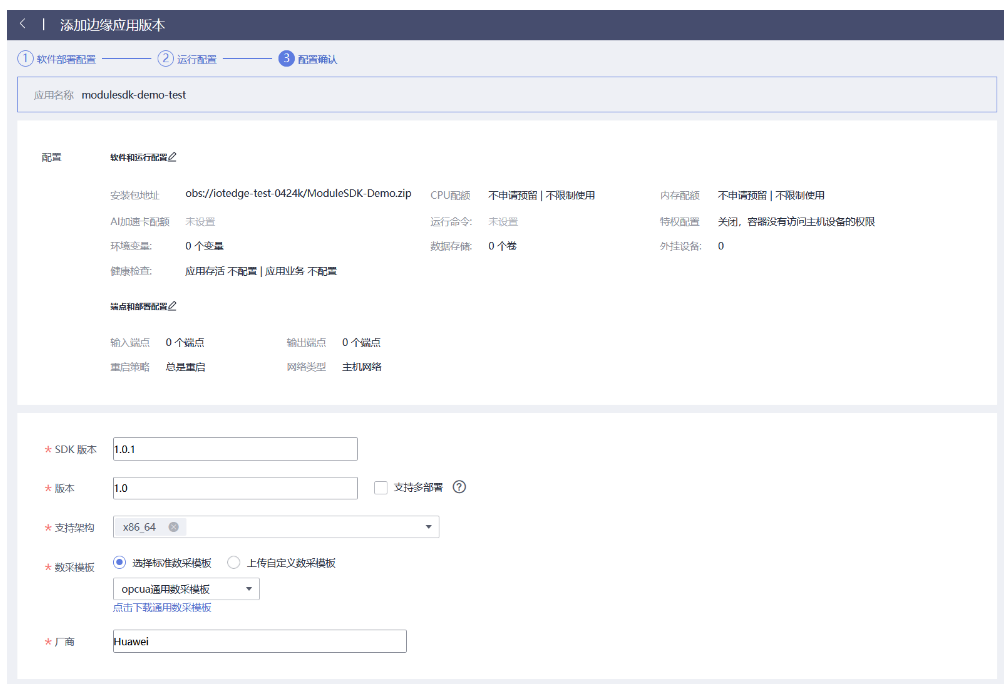
说明

输入输出端点是非必需配置的，当有数据流转时才需要配置。

驱动类应用和IT应用一般不需要配置。

6. 配置确认，单击"立即发布"

 版权所有 © 华为云计算技术有限公司



4.5.4.8 部署应用

部署应用，具体参考**部署应用**，进入节点详情页安装应用。

4.5.4.9 OT 数采配置

此数据源以opcua模板为例。可参考**OT数采配置**自定义数据模板。自行开发集成ModuleSDK驱动应用，定义自己的数据源模板，以及点位信息后，下发配置。

4.5.4.10 查看采集结果

1. 访问**IoT边缘**，单击“管理控制台”进入IoT边缘控制台。
2. 在左侧导航栏选择“边缘节点 > 节点管理”，选择之前创建的边缘节点，单击节点进入详情页。
3. 选择"数采配置"页签，单击采集值下方按钮查看采集的数据。

版权所有 © 华为云计算技术有限公司

采集值详情

链路节点	时间	类型	采集值
EdgeCloud	2022-09-26 11:31:45:...	下行	--
EdgeHub	2022-09-26 11:31:45:...	下行	--
user_modul...	2022-09-26 11:31:45:...	下行	--
user_modul...	2022-09-26 11:31:45:...	上行	1
EdgeHub	2022-09-26 11:31:45:...	上行	1
EdgeCloud	2022-09-26 11:31:45:...	上行	1

版权所有 © 华为云计算技术有限公司