IoT 边缘 IoTEdge

开发指南

文档版本1.1发布日期2024-10-21





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

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。

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	
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

4.3.4.4 创建产品	
4.3.4.5 修改代码	
4.3.4.6 项目打包	
4.3.4.7 制作镜像包	
4.3.4.8 创建应用	
4.3.4.9 部署应用	
4.3.4.10 添加边缘设备	
4.3.4.11 设备接入	
4.3.5 集成 ModuleSDK 进行工业子系统接入	
4.3.5.1 操作场景	
4.3.5.2 代码解析	
4.3.5.3 项目打包	
4.3.5.4 制作镜像包	
4.3.5.5 添加应用	
4.3.5.6 注册节点	
4.3.5.7 创建 API	
4.3.5.8 网关应用创建及绑定	
4.3.5.9 添加数据端点	67
4.3.5.10 部署应用	
4.3.5.11 使用	69
4.3.6 集成 ModuleSDK 进行协议转换	
4.3.6.1 操作场景	
4.3.6.2 代码解析	
4.3.6.3 注册节点	
4.3.6.4 设备建模	
4.3.6.5 项目打包	
4.3.6.6 制作镜像包	
4.3.6.7 添加应用	
4.3.6.8 部署应用	
4.3.6.9 添加边缘设备	
4.3.6.10 设备接入	
4.3.7 集成 ModuleSDK 进行 OT 数采	
4.3.7.1 操作场景	
4.3.7.2 代码解析	
4.3.7.3 注册节点	
4.3.7.4 设备建模&发放	82
4.3.7.5 项目打包	
4.3.7.6 制作镜像包	
4.3.7.7 添加应用	
4.3.7.8 部署应用	
4.3.7.9 OT 数采配置	
4.3.7.10 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	
4.3.8.2 代码解析	
4.3.8.3 注册节点	
4.3.8.4 设备建模&发放	84
4.3.8.5 项目打包	
4.3.8.6 制作插件包	
4.3.8.7 添加应用	85
4.3.8.8 部署应用	
4.4 集成 ModuleSDK(C)	
4.4.1 内部架构	
4.4.2 开发指导	
4.4.2.1 接口函数	
4.4.2.2 前提条件	
4.4.2.3 创建工程	
4.4.2.4 生成可执行文件	
4.4.2.5 制作镜像包或插件包	111
4.4.2.6 添加应用	
4.4.2.7 发布应用	
4.4.2.8 如何使用	
4.4.3 下载 Demo	120
4.4.4 集成 ModuleSDK 进行数据处理	
4.4.4.1 场景说明	
4.4.4.2 代码解析	
4.4.4.3 注册节点	
4.4.4.4 创建产品	
4.4.4.5 修改代码	
4.4.4.6 项目打包	
4.4.4.7 制作镜像包或插件包	129
4.4.4.8 创建应用	
4.4.4.9 部署应用	
4.4.4.10 添加边缘设备	
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	
4.4.5.2 Demo2	
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	

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

目录



굸 北向应用NA (SaaS) HTTPS 应用侧开发 涉及部分 IoT Edge边 设备接入 APIG网关 缘 IoT DA WebSocket MQTTS 边 NM API GW Agent Hub 数据应用 HTTP 插件开发 MQ1 ςΤ. 涉及部分 驱动应用 IT子系统 MQTTS 设备 设备 设备

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:



开发 操作	开发说明
应用	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
侧开	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
发	API,可以实现与在云端手动创建同样的效果)。
插件	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
开发	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loT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。



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(jav a)	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.f x	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

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

名称	必选/可选	类型	位置	说明
X-Auth- Token	必选	String(0-20000)	Hea der	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_i d	可选	String(4-32)	Quer y	应用ID搜索关键字
offset	可选	Integer[0,100000 0]	Quer y	查询的起始位置,取值范围为非负整数,默认为0
limit	可选	Integer[0,100000 0]	Quer y	每页记录数,默认值为10,取值区间为1-1000
app_type	可选	String(0-64)	Quer y	应用id搜索关键字
function_ty pe	可选	String(0-64)	Quer y	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。

可点击key和value值进行编辑

<u></u>						
Sessions 49 Environments		-north-4.myhuaweicloud.com/	√5/iot/0506 Ċ			ts 200
批量傳改 profile						
获取 token POST /v3/auth/tokens	Description He	aders URL Params Body Aut	h Options		Info Request Response	
修改 profile				He	aders JSON ~ Raw	
PUT /v2/devices	EV Content-Type	application/json;charset=utf8				
查询应用列表	X-Auth-Token	MIIVAAYJKoZIhvcNAQcCollU8	STCCFOOC = 💬	2 Date: Wed, 09 Jun 2021 3 Content-Type: applicati	15:48:11 GNT op/ison:charset=UTE-8	
SET /v2/05068467bb8026bc2 查询产品列表 SET /v5/iot/05068467bb8026				4 Transfer-Encoding: chun 5 Connection: close 6 Server: api-gateway 7 X-Request-Id: 6a4fb6bd0 8		
			нттр 🌣 🟦 🔒	<pre>9 {"products": [{"app_id":"gkYrQnnGzL7 bwstaff pub ToTEdge io</pre>	u_AmU7NssNAJUV2Ia","app_name": t"."product_id":"60bd992db86d7	"DefaultApp
	3 X-Auth-Token: HTINAY/NCHL-NAROCCI Boggintee IT-3-1660-70 Auth-Token-State State State State State Comparison of the State State State State Comparison of the State State State State Non-State State State State State Non-State State State State State Non-State State State State State State Non-State State State State State Non-State State State State State Non-State State State State State Non-State State State State State State State Non-State State State State State State State Non-State State State State State State State State Non-State State Sta	$\label{eq:constraint} under Constraint} (C) (C) (C) (C) (C) (C) (C) (C) (C) (C)$	EvghtSbychi.169v48 rvmTuRright.EphtB rvmTuRright.		raption":mull, "creat_lime"'" "product_id": "debd@acsf800 vcc_type:"nodus_device", "debd@acsf800 vcc_type:"nodus_device", "debd@acsf800 vcc_type:"nodus_tid": "debd@acsf800 vcc_type:"nodus_tid": "debd@acsf800 vcc_type:"nodus_tid": "debd?acsf800 vcc_type:"nodus_tid": "debd?acsf800 vcc_type:"nodus_tid	W21000/1055 DefaultApp_ 02bc772de0 Protocol_typ "humave","i 607T0355242 DefaultApp_ 02bc763c20 DefaultApp_ 02bc763c20 Sub

将返回数据格式化以便于查看。

cker"), ("edge_app_id1":"sdfdssdfsdfsd","description":null,"create_time":"2021-03-22701:48:552","upgate_time":"2021-03-22701:48:552","inst_published version":null,"app_type":"USER","function_ type":"DATA_PROCESSING","deploy_type":"process"}]}







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。



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loT边缘节点,节点注册要求请参考<mark>节点安装限制</mark>。
- 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。

⚠ 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)

loT边缘云服务为应用提供总线对接能力、设备命令下发能力。loTEdge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。





- 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。





- 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。

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。





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

 $\label{eq:public void start} ModuleShadow(ModuleShadowNotificationCallback callback) \ throws \ lllegalArgumentException$

表 4-3 参数说明

参数名称	类型	参数描述
callback	ModuleShadowNotificati onCallback	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(Modul eShadowNotification 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()

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	ConnectionStatusChange Callback	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(Con nectionStatus 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。

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.lllegalArgumentException

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。

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	CustomizedMessageCallb ack	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

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。
reportSubDevicesPropertiesAndDiscove ry	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
lotResult 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

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

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	SubDevicesDiscoveryProp sReport	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	UpdateSubDevicesStatus Event	更新子设备状态事件

DriverClient::getProducts

发送获取产品事件。

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ltClient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。

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></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

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

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

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

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。

Project Structur	e		×
$\leftarrow \rightarrow$	+ 2 6	Name	iot-module-client-202-jar-with-dependencies
 Project Structur ← → Project Settings Project Modules Libraries Facets Artifacts 1 Platform Settings SDKs Global Libraries Problems 1 		Name: 3 + +5 Cla	iot-module-client-2.0.2-jar-with-dependencies asses D:\workspace\IDEA\ModuleSDK-Java\iot-module
	III Maven: org.apache.logging.log4j:log4j:to-slf4j:2.11.2 IIII Maven: org.apache.tomcat.embed:tomcat-embed-co IIII Maven: org.apache.tomcat.embed:tomcat-embed-co IIII Maven: org.apache.tomcat.embed:tomcat-embed-el: Maven: org.apache.tomcat.embed:tomcat-embed-wei III Maven: org.apache.tomcat.embed:tomcat-embed-wei III Maven: org.jboss.logging:jboss-logging:3.3.2.Final IIII Maven: org.slf4j:glu-to-slf4j:1.7.25 IIII Maven: org.slf4j:slf4j-api:1.7.26 IIII Maven: org.springframework.boot:spring-boot-auto IIII Maven: org.springframework.boot:spring-boot-start IIII Maven: org.springframework.boot:spring-boot-start IIII Maven: org.springframework.boot:spring-boot-start		
0	IIII Maven: org.springframework.boot.spring-boot-start		OK Cancel Apply

----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<mark>项目打包</mark>。

- 步骤1 打开Visual Studio Code,单击出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

\sim	ne for	Second New 70 Pru leuman	30h			
۵			F Extension: IoT Edge SDK ×			
Q & & B	ыты 2 2 Х	pe SDK Föge SDK Föge SDK for 10 development izeröge Azure IoT Hub S2 Microsoft Microsoft	ସ 	oT Edge SDX (##83) moto: I ###### #2010 for bil dendempenent ∰ @ ■ 0 1000 0 00000000		
Δ		Azure IoT Edge This extension is now in maintenance mor Microsoft		loT Edge SDK	Categories	
		IoT Utility c4 800 Develop IoT project based on PlatformiO: Jun Hen	₩ 5 d		Other Extension Resources	
		IoT Device Cube		Features • Adds a command "to! Edge SaktOpen Sak Project" that deplays a example joon project for edge sakt. • Adds a command "to! Edge SaktOpenSage" that build a jar Tie.	Marketplace iotedge	
		IoT Extension Pack OP som Build IoT Solutions on top of avvesome ter Jun Hen	* 5 L. 	Installation	More Info Published	2023-12-15, 09:52:50
	٢	IoT Link IDE for IoT development iotlink	0	Usage	Lant updated Identifier	2023-12-13, 093907 2023-12-18, 0931-23 Satedge.iot.edge.sok
	No	IoT MiniApp do to Official IDE for IoT miniapp development, hass-ul	• 5 5. •	'nos 1½ το sepan the command pakette, type "to" Edge Seldcipan Sidi Project" and pross Enter. A example java project for edge selk will be displayed. Press ½ (is open the command pakette, type "to"Edge Seldvipskage" and press Enter. A example java jar (file will be build. C'antrihin lutinon		
	⊃ ^s	IOT Editor I to 10 Developing application with connected Pr KooloT	**	Issues		
	۲	iot-template of layout template iot	41	ucense		

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。
IoT Edge Sdk:Open Sdk Project

图 4-2 输入命令

9 BOS	Edit Selection View Go Bun Terminal Help				
¢.			IoT Edge Sdk:Open Sdk Project	recently used 🕲	
	iava	= 7	IoT Edge SdkPackage		
ρ_					
P 1	Extension Pack for Java 0-21.6				
20	Popular extensions for Java development that provides Java	a IntelliSe			
~	Microsoft Reload Require				
\triangleright	Debugger for Java 4-27.4				
	A lightweight Java debugger for Visual Studio Code				
0	Microsoft Reload Reg	ind Initial	Add Function Breakpoint		
- 🖻	Test Russeer for Java 0.242		Add XHR/hetch Ereakpoint		
п	Run and debug JUnit or TestNG test cases.		Azure Account: Report Issue		
=	Microsoft Reput		Azure IoT Edge: Add Dev Container definition files		
- P	Project Manager for Jun (5.11)		Abure to Ledge Add to Ledge Module		
-	Manage Java projects in Visual Studio Code		Acure for Edge, edite and Position Edge Module Image		
	Microsoft Record Record	e Install 🖂	Annual of Column and Point of Columns in Constants		
1	<u> </u>		Acure for Edge, build and Karrior Edge Solution in Simulator		
	Mayen for Java Organization and Constraints and Constraints	rom archal	Arure to't Edge, balls fo't Edge module intege		
	Microsoft Record Projects	ied listal	Anure IoT Edge: Create Deployment at Scale		
			Azure IoT Edge: Create Deployment for Single Device		
	Spring Initialize Java Support	Q 1M 🔮 4	Azure IoT Edge: Create IoT Edge Device		
	Microsoft	Indial India	Azure IoT Edge: Generate IoT Edge Deployment Manifest		
	Language Support for Java(TM) by Red Hat @ 20204	* 2.5 3 266ms			
	Ana Unting, Intellisense, formatting, relactoring, Malvery Ana Unting, Intellisense, formatting, relactoring, Malvery Ana Unting, Intellisense, formatting, relactoring, Malvery	d Redail V			
	Tabnine: Al Autocomplete & Chat for Javascript, Pyt				
	Al coding assistant with Al code completions and Al code	chal right L_		Show Settings	



EXPLORER ····	J ApiCantrollerjava X
V OPEN EDITORS	erp-integration > src > main > java > com > huaweicloud > samples > sdk > iot > erpintegration > 🤳 ApiControllerjava
× J ApiController.java erp-integration\src\main\java\com\huaweicloud\samples\sd	1 package com.huaweicloud.samples.sdk.iot.erpintegration;
V IOT-MODULE-SAMPLES	
> .idea	3 import com.huaweicloud.sdk.iot.module.ItClient;
> dc-driver	4 import org.springtranework.opeans.tatCory.annotation.AutoMired;
v erp-integration	5 import org, springermanwark, web bind anotation. Berthapping; 6 import org, springformanwark, web bind anotation. Derthapping;
✓ src\main	7 import ore springframework.web.bind.annotation.RestController:
viava\com\huaweicloud\samples\sdk\iot\erpintegration	8 import org.springframework.web.bind.annotation.RequestMapping;
J ApiController.iava	9 import org.springframework.web.bind.annotation.RequestBody;
J Application java	10 import org.springframework.web.bind.annotation.PutMapping;
J AppResponse java	11 import org.springframework.web.bind.annotation.DeleteMapping;
1 AuthFilter iava	<pre>12 import org.springframework.web.bind.annotation.PatchWapping;</pre>
1 ConfigController inva	
ConfigService java	14 import java.utii.map; 15
1 HintegrationService inva	
PoturoDto inus	
a recompanya	
> texat	
> target	20 @RestController
· pontonii	21 @RequestHapping(value = "/api")
210	22 public class apicontroller
 moabus-ariver 	23 and usto final States modulate
> src	
> target	26 @Autowired
» pom.xml	27 public ApiController(ItClient itClient) {
> monitor-app	28 this.moduleId = itClient.getConfig().getModuleId();
J .classpath	
» pom.xml	
	31 geothaping(//get)
	22 public Appressions generation and an analysis and a public and a
	34)

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

р ектема			Extension: Extension Pack for Java >	
	sion for Extension Pack for Java Popular extensions for Java deve Microsoft Pythen for VSCele Python language extension for v Thorma Hasken Townsend	Constant the second sec	Exte Microsof Popular Intel 1	nsion Pack for Java 96:51:35 • Intercention • 0 235%076 • + + + + + + + + + + + + + + + + + +
s 🧕	Japanese Language Pack for 日本語 尊 Microsoft	Ф 74M 🛫 45		Extension Pack (6)
٩	Spanish Language Pack for Español Ø Nicrosoft	Ф 6.1М 🛨 4.5 [1916]		
٩	Russian Language Pack for Pyccoriñ @ Microsoft	क रस्म 🛠 रड स्विध्य		
	Chinese (Simplified) (颜修中 中文(語年) ● Microsoft	Ф 20.5М 🛫 5 [HSLAT]		
9	Portuguese (Brazil) Langua Portugués (Itrisil) 🌻 Microsoft	Ф 48М 🛠 45		
9	Korean Language Pack for V 한국이 ♥ Microsoft	Ф 34М 🕊 43		
	French Language Pack for Vis Français Microsoft	ф з.4м 🌪 4 [18434]		
,	Python Extension Pack Popular Visual Studio Code exter Don Jayamanne	Ф 72M 🛨 48 nsions for P		
() () ()	PHP Extension Pack Everything you need for PHP der Ø Xdebug	a 444 🔆 45 thempolev Refer		

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

图 4-5 构建 jar 包

the fore Searchen Thus To Day Names Dath		>	
DPLORDR ····	J ApiControllerjava X	ToT Edge SdkPackage	recently used @
~ OPEN EDITORS		* ToT Edge Sidk:Open Sidk Project	
X J ApiControllerjava exp-integration/univ/jousicon/huoveiclous/sampler/sat	1 package com.huaweicloud.samples.sdk.iot.erpinteg		
V 10T-MODULE-SAMPLES			
	3 import com.huaweicloud.sdk.iot.module.itclient;	IoT Link: Home	
> dc-driver	Inport org.springramwork.eeais.ractory.amotation	(Internal) Build a Target by Name	
v erp-integration	import org.springframework.web.bind.annotation.P		
✓ src\main	7 import org.springframework.web.bind.annotation.R		
v java\com\huaweidoud\samples\sdk\iot\erpintegration	8 import org.springframework.web.bind.annotation.R		
J ApiController.java	9 import org.springframework.web.bind.annotation.R		
J Application.java	10 import org.springframework.web.bind.annotation.P	Add Cursors To Top	
J AppResponse.jiwa	11 Import org.springframework.web.bind.annotation.U	Add Function Breakpoint	
J AuthFilterjava	13 Input orgaspringtranewski, web. official and called		
J ConfigController.java	14 import lava.util.Map:		
J ConfigService.java			
J tintegrationService java		Add XHR/fetch Breakpoint	
J ReturnOto java		Azure Account: Report Issue	
> resources		Azure IoT Edge: Add Dev Container definition files	
> target	19 -7 10 - Alextrosteollan	Azure IoT Edge: Add IoT Edge Module	
pomumi	21 RequestBansing(value = "/ani")	Azure IoT Edge: Build and Push IoT Edge Module Image	
> Ib	22 public class ApiController {	Azure IoT Edge: Build and Push IoT Edge Solution	
✓ modbus-driver		Azure IoT Edge: Build and Run IoT Edge Solution in Simulator	
> src		Azure IoT Edge: Build IoT Edge Module Image	
> target		Arure IoT Edge: Build IoT Edge Solution	
> pom ani	20 Mutowined		
> monitor-app	27 public Apicontroller(Itclient Itclient) {	man 1-7475	
J classrath	20 CHIS.MODULEIG = ICCILENC.getConvig().get		
a nomami			
	<pre>32 public AppResponse getTest() {</pre>		
	17 rejuste UtateersticoService itTeteersticoSe	mire	



V IOT-MODULE-SAMPLES	
	3 import com.huaweicloud.sdk.iot.module.ItClient;
✓ dc-driver	4 import org.springramework.beans.factory.annotation.Autowired;
> src	5 import org.springrammeork.web.bind.annotation.BectManding.
✓ target	7 import org.scripeframework.web.bind.anotation.BestController:
> dasses	<pre>import org.springframework.web.bind.annotation.RequestMapping;</pre>
> generated-sources	9 import org.springframework.web.bind.annotation.RequestBody;
> maven-archiver	10 import org.springframework.web.bind.annotation.PutMapping;
maunactatur	11 import org.springframework.web.bind.annotation
de-driver-100 jar	12 .DeleteMapping;
a nom yml	13 import org.springtramework.web.bind.annotation.PatchNapping; 34
× em-integration	15 import java util Map:
str main	
) inc	
) resources	
A broat	
- unger	
> negrated sources	21 grastcontroller 23 grastcontroller
> generates sources	2.2. guidet schapping (sause = 740)] 23. guidet class Addicated land
> maven-archiver	
7 mayen-status	25 private final String moduleId;
erp-integration-1.0.0jar	
pom.xmi	27 @Autowired
> HD	28 public ApiController(ItClient itClient) {
✓ modbus-driver	29 this.moduleId = itClient.getModuleId();
> src	
∨ target	21 9754Manning("/ast")
> classes	33 public Andersonne getTest() {
> generated-sources	<pre>34 return AgnResponse success("GetTest Success", new ReturnDto(moduleId));</pre>
> maven-archiver	
> maven-status	
modbus-driver-1.0.0.jar	37 @Autowired
pom.xml	38 private ItIntegrationService;
✓ monitor-app	
> src	40 ////////////////////////////////////
✓ target	SOCIETARS CULTERIT DIRECCONDUCT TRANSMA SOCIES
> classes	
> generated-sources	[INU] compiling 3 source files to C: User's Unneded /29/lot-edge-sol(lot-module-samples/worldo-app/target/classes [Diro]
> maven-archiver	[INFO] maven-resources-plugini2.6itestResources (default-testResources) @ monitor-aco
> maven-status	(WARNING) Using platform encoding (GRK actually) to copy filtered resources, i.e. build is platform dependent!
monitor-app-1.0.0.jar	[1NFO] skip non existing resourceDirectory C:\Users\N000000729\iot-edge-sdk\iot-module-samples\monitor-app\src\test\resources
s pom.xmi	[INFO] never-concile-niurin:3.5:testConcile (default-testConcile) @ monitor-ann
J .classpath	[INFO] No surves to compile
	[INFO] maven-surefire-plugin:2.12.4:test (default-test) @ monitor-app
	Lineoj no testis to run, TINEO]
	[INFO] maven-jar-plugin:2.4:jar (default-jar) @ monitor-app
	[INFO] Building jar: C:WSers\M00000729\iot-edge-sdk\iot-module-samples\monitor-app\tanget\monitor-app-1.0.0.jar
	[IN60]
	[IN 0] Reaction Journaly for too module samples 1.010.
	[INFO] jot-module-samples
	[DNF0] de-driver
	[1000] models-druper
	[1040] monitor-app
	[INFO]

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。

Project Structure	
$\leftarrow \rightarrow$	+ 一 恒
Project Settings Project	Create JAR from Modules ×
Modules	Module: monitor-app ~
Libraries	Main <u>C</u> lass: com.huaweicloud.samples.sdk.iot.monitorapp.Main 📄
Facets	IAR files from libraries
Artifacts	
Platform Settings SDKs	 extract to the target JAR copy to the output directory and link via manifest
Global Libraries	Directory for META-INF/MANIFEST.MF:
Problems	>rkspace\git\ModuleSDK-Java-master\iot-module-samples\monitor-app
	OK Cancel

- 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)

<u>B</u> uild	R <u>u</u> n	<u>T</u> ools	VCS	<u>W</u> indow	<u>H</u> elp	iot-modu
🔨 Bu	ild <u>P</u> ro	oject				Ctrl+F9
Bu	ild <u>M</u> o	dule 'm	onitor	-app'		
R <u>e</u>	build I	Module	'monit	tor-app'	Ctrl	+Shift+F9
<u>R</u> e	build I	Project				
Bu	ild Art	ifacts				
An	alyze /	APK				
Ru	n Ant '	Target			Ctrl+	Shift+F10

----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]

门 说明

livingobjects/jre8镜像是docker hub第三方提供的镜像,非loT团队发布,且loT团队未提供 任何官方镜像。该镜像在此仅做示例,loT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<mark>容器镜像服务 SWR_用</mark> <mark>户指南</mark>

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 5. 查看上传结果

镜像名称 ↓Ξ	所属组织	版本数 ↓Ξ	更新时间 15	操作
edge_monitor	hw_swr	1	2021/05/07 22:11:25 GM	镜像自动同步

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—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**
- 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。

〈 添加边缘应用版本	
 文件部署配置 —— ② 运行配置 —— ③ 配置确认 	
应用名称 iot_edge1208	
* 部语方式 容器化结晶 安装包部器 软件集督操作时明 1. 根据您的业务需要开发边染应用程序 2. 制作镜像,并上传到 容器镜像感觉(SWR)中、制作镜像语参考 Docker首网	
* 容器領像 手动输入 ▼ est1202	
容器规格 CPU 配额 申请预留 一段创使用 内存配额 申请预留 月均使用 日 申请项目 日 申请项目 日 申请项目 日 申请项目 日 申请项目 日 申请の目の記録	
高级配置 ◇ 运行命令	
→ 透爽配置 通过勾运项,容器将拥有访问主机设备的权限,例如容器访问GPU,FPGA。	

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

〈 添加边缘	应用版本	
① 软件部署配置 —		
应用名称 iot_edg	ge 1208	
端点配置		
例如, EdgeHub	使用MQTT Broker做消息总线,输入疏点代表概块接收消息的Topic,输出疏点代表发送消息的Topic(疏点仅代表而并非真正的MQTT Topic)。	
输入端点 🕐	谐输入端点 汤加满点	
	input 💿	
输出端点 ⑦	请输入通点 通加胰点	
	output 🚳	
部署配置		
重启策略:	总是里台 先欧时堂台 不堂台	
网络米田	indutyonatio, toartalatoeffalat, amercemitaloutyo,	
州如关室		
	(现用描述》(过速力点) 即阿森,即除露马王的时代都阿南福冀,使用同一个PP。	
	上—步 下—	步

🛄 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。

应用名称 iot_	edge1208				
配置	软件和运行配置之				
	锦缨地址 test1202	CPU配额	不申请预留 不限制使用	内存配额	不申请预留 不限制使用
	Al加速卡配额 未设置	运行命令:	未设置	特权配置	关闭,容器没有访问主机设备的权限
	环境变量: 0 个 变量	数据存储:	0 个卷	外挂设备:	0
	健康检查: 应用存活 不配置 应用业务 不配置				
	端点和部署配置《				
	输入端点 1个端点 输出端点 1个端点				
	重启策略 总 是重启 网络类型 主机网络				
* SDK 版本	请输入SDK版本				
* 版本	请输入应用版本 🗌 対	寺多部署 ⑦			
* 支持架构	请选择版本所支持的架构	*			

⚠ 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。

数据存储 支持挂载本地卷到	J容器中, 已实现数据文件的持	久化存储			
支持挂载本地卷到	容器中, 已实现数据文件的持久	化存储			
本地卷名称	类型	主机目录	容器目录	权限	操作
config	CONFIG	▼ /config	/opt/config	读写	▼ 删除

----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

〈 添加边缘	应用版本			
1 软件部署配置 —		③配置	确认	
应用名称 iot_edg	ge1208			
* 部署方式	容器化部署 软件准备操作说明 1. 根据您的业务需要开 2. 将应用程序打包成安	安装包部署 发边缘应用程序 装包文件,并上传到 %	İ象存储服务(OBS)	

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。

〈 添加边缘应用版本	
 軟件部署配置 ——— ② 运行配置 ——— ③ 配置确认 	
应用名称 iot_edge1208	
* 部署方式	
* 安装包地址 edge-monitor/monitor-app.zip	
进程规格	
CPU配版 申请预留	
□ 限制使用	
内存配版 申请预留	
□ 限制使用 	
AI加速卡配線 不由请配線 申请GPU配線 申请GPU配線	
高级配置	
→ 违项配置 通过勾造项, 容器将拥有访问主机设备的权限,例如容器访问GPU, FPGA。	
✓ 环境变量 容器运行环境中设定的变量。可以在应用部署后传改,为应用提供极大的党活性。设置容器运行环境中的系统环境变量。	

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。

〈 添加边缘	应用版本
① 软件部署配置 —	
应用名称 iot_edg	ye 1208
端点配置 例如,EdgeHub 输入端点 ⑦ 输出满点 ⑦	使用MQTT Broker做消息急线,输入就点代表模块接收消息的Topic,输出端点代表发送消息的Topic(端点仅代表而并非真正的MQTT Topic)。
部署配置	
重启策略:	总是重启 失败对重启 不重启
	当应用实例通出时,无论是正常退出还是异常退出,系统都会重新拉起应用实例。
网络类型	主机网络 第二种时
	使用宿主机。(边壕节点) 的网络,即容器与主机间不做网络隔离,使用同一个IP。

🛄 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

< 添加边线	家应用版本				
1 软件部署配置	(2) 运行配置 —— 3 配置确认				
应用名称 iot_e	dge1208				
配置	软件和运行配置之				
	安装包地址 edge-monitor/monitor-app.zip	CPU配额	不申请预留 不限制使用	内存配额	不申请预留 不限制使用
	Al加速卡配额 未设置	运行命令:	未设置	特权配置	关闭,容器没有访问主机设备的权限
	环境变量: 0 个变量	数据存储:	0 个卷	外挂设备:	0
	健康检查: 应用存活 不配置 应用业务 不配置				
	端点和部署配置之				
	输入诱点 0个满点 输出满点 0个满点				
	重启策略 总是重启 网络类型 主机网络				
* SDK 版本	请输入SDK版本				
* 版本	请输入应用版本 支持多	部署 ?			
* 支持架构	请选择版本所支持的架构	•			

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



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loT边缘,单击"管理控制台"进入loT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。

00007538x0807694 0000010102022/04/0010101000	3MT+08:00 / 編述					
版本列表 ⊙ ♀ Q	版本信息					
3 x86_64,arm32 0	3 ● 米文作 ② 不定的多部署 新在 第編 第五 書称 新在 第二 第二 日本 10 月 月 日 日本 10 月 月 日本 10 月 日本					
1111 x86_64 0						
1.0.0 x86_64 0	容器配置 姚中配置 影響配置 实动例类(0)					
	基本值班	海拔配置				
	能验账径 swr.cn-north-4.myhuaweicloud.com/lot_edge_test/ot_test/v2	這项配置:关闭,審課没有访问主机设备的权限				
	客器機格	环境空量:未设置				
	CPU起源:不申请预算 不限制使用	数据存储 1个卷 查看				
	内容配版 不申请预留1不限制使用	外挂设备: 未设置				
	Allfies于配额: 不現射	健康检查: 应用存活 不配置 应用业务 不配量				
		這行命令: 未设置				

----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<mark>应用部署</mark>。
- 2. 添加设备进行测试(添加设备请参考<mark>设备接入边缘节点</mark>)。

🛄 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<mark>应用部署</mark>。
- 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。



- 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

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(); l

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());
```

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<mark>注册边缘节点</mark>。

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	÷.
----------	----

创建产品					×
* 所属资源空间	hw_iotedge 如需创建新的资源空间,您可前往当前实例详情创建	•	0		
★ 产品名称	hw_iotedge_mqtt]		
协议类型	MQTT	•	0		
★ 数据格式	JSON	•	0		
★ 厂商名称	hw]		
所属行业	无	•]		
* 设备类型	MQTT_Device		0		
高级配置 ▼	定制ProductID 备注信息				
				确定	取消

🗀 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。

添加服务	5	×
★ 服务ID	power	
服务类型	power	0
服务描述]
	0/1:	28
	确定取消	



新增属性		×
★ 属性名称	status	
属性描述		
	0/128	
* 数据类型	string(字符串) ▼	
* 访问权限	可读	
★长度	10	
枚举值	normal,error	
	12/1024	
	确定取消	

步骤4	新增命令。							
	新增命令							×
	★ 命令名称	power_control]				
	下发参数	新増輸入参数						
		参数名称	数据类型		描述	操作		
		restart	string(字符串)			修改	删除	
		5 🔻 总条数:1 < 1						
	响应参数	新增响应参数						
		参数名称	数据类型		描述	操作		
				暂无表	格数据			
			确定	取消				

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: loTDA->产品列表。

					4	所有资源空间	• C
	产品名称	产品ID	资源空间	设备类型	协议类型	操作	
1	hw_lotedge_modbus	6099fb6aaa3bcc02c022ef18	hw_lotedge	modbus-device	Modbus	查看 删除	
	hw_iotedge_mqtt	60988d94aa3bcc02c0200667	hw_iotedge	MQTT_Device	MQTT	查看 删除	

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<mark>项目打包</mark>

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<mark>制作镜像包或插件包</mark>。

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。

〈 添加边缘应用版本	
① 软件部署配置 ——— ② 运行配置 ———— ③ 配置端从	
应用答称 iot_edge1208	
* 部署方式	
* 容器視像 手动输入 ▼ test1202	
容器规格	
CPU配版 申请预篇	
□ 限制使用	
内存配版 日申请预留	
AI加速卡配额 不申请AI線 申请GPU配额 申请NPU配额	
高级配置	
✓ 遺行命令	
→ 這项配置 通过均连项,容器构拥有访问主机设备的权限,例如容器访问GPU, FPGA,	

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

< 添加边缘	应用版本	
① 软件部署配置 —		
应用名称 iot_edg	ge 1208	
端点配置		
例如, EdgeHub	使用MQTT Broker做消息总线,输入就点代表模块接收消息的Topic,输出就点代表发送消息的Topic(端点仅代表而并非真正的MQTT Topic)。	
输入端点 ⑦	请输入端点 添加端点	
	input 🕲	
输出端点 ⑦	请输入跳点 添加满点	
	output 🚳	
部署配置		
重启策略:		
177.4 6 .346.330	当应用头颅虚凹时,尤论是止率遮凹还是异率遮凹,系统命管里所包起应用头例。	
內暗突型	主机网络 第日映射	
	使用做主机(这場节点)的网络,即警諜与主机间不做网络隔离,使用同一个P。	
	Ŀ	-步 下步

🛄 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。

環像地址 test1202 CPU配额 A加速卡配板 未设置 运行命令 环境变量: 0 个变量 数据符储 健康检查: 应用存活 不配置 应用业务 不配置	 不申请预留 不限制使用 未设置 0 个卷 	内存配额 特权配置 外挂设备:	不申请预留 不限制使用 关闭,容器设有访问主机设备的权限
A加速卡配額 未设置 运行命令 环境交易: 0 个变量 数据存储 健康检查: 应用存活 不配置 应用业务 不配置	:: 未设置 :: 0个卷	特权配置 外挂设备:	关闭,容器没有访问主机设备的权限 -
环境变量: 0 个变量 数据存储 健康检查: 应用存活 不配置 应用业务 不配置	:: 0 个卷	外挂设备:	
健康检查: 应用存活 不配置 应用业务 不配置			0
端点和部署配置∠			
输入端点 1个端点 输出端点 1个端点			
重启策略 总是重启 网络类型 主机网络			
★ SDK 版本 请输入SDK版本			
★ 版本 法输入应用版本 □ 支持多部署 ⑦			
- Wet- alders are proved			
★支持架构 请选择版本所支持的架构 ▼			

⚠ 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。

数据存储 支持挂载本地	卷到容器中,已实现数据文件的持法	久化存储				
1 支持挂載本地卷]]容器中, 已实现数据文件的持久	化存储				
本地卷名称	类型	主机目录	容器目录	权限		操作
config	CONFIG	▼ /config	/opt/config	读写	•	删除

----结束

4.3.4.9 部署应用

步骤1 部署应用,具体参考<mark>部署应用</mark>。

<u>∧ 注意</u>

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

> 2 数据流转配置 部署应用后,您可以根据需要配置	國数据流转的来源和目标,灵活控制数据转发路径,并提高数据:	安全性。	
规则名称	消息来源	消息目标	操作
monitor_input	设备端 ▼	user_monitor-app v input v	御除
monitor_output	user_monitor-app 💌 output 💌	云满	制除
 添加規則 保存<取消 			

🛄 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<mark>设备接入</mark>,以下是添加边缘设备(MQTT设备)配置时的参考:

接入设备需统· 例/资源空间。	统入物联网平台管理,并和当前边缘节点归属同一个诊	2 會接入服务实
归属服务实例 归属资源空间	loTDA默认基础版 hw_lotedge	
所屬产品	hw_iotedge_mqtt v (沒有可选产品? 请前往设备接入服务增加且定义产品, 往添加产品	C 并定义设备功能
设备标识码	hw_iotedge_device1	
设备名称	hw_iotedge_device1	

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:

	TT fv - 171			_	n	×
File	Extras Help					~
	local mosquitto 🔹 🔅 Connect	Disconnect				•
Pub	blish Subscribe Scripts Broker Status Log					
Switc	ch Fountain ON Soc/devices/hw_jotedge_device2/sys/prop	erties/report Publish	QoS0 QoS1 QoS2	Retained		0;▼
Switc	ch Fountain OFF					

▲ 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

Edit Connection Profiles			– 0 ×
M2M Eclipse			
local mosquitto	Profile Name	local mosquitto	
	Profile Type	MQTT Broker	
	MQTT Broker Profile Settings		
N	Broker Address	124.70.60.6	
~0	Broker Port	7883	
	Client ID	hw_iotedge_device1_0_0_2021050810	Generate
	General User Credentials	SSL/TLS Proxy LWT	
	Connection Timeout	30	
	Keep Alive Interval	60	
	Clean Session	\checkmark	
	Auto Reconnect		
	Max Inflight	10	
	MQTT Version	✓ Use Default	
		3.1.1	
		Clear Publish History	
		Clear Subscription History	
+ -	Revert	(Cancel OK Apply

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)。

Edit Connection Profiles			– 0 ×
M2M Eclipse			
local mosquitto	Profile Name	local mosquitto	
	Profile Type	MQTT Broker	MOT
	MQTT Broker Profile Settings		
	Broker Address	124.70.60.6	
	Broker Port	7883	
	Client ID	hw_iotedge_device1_0_0_2021050810	Generate
	General User Credentials	SSL/TLS Proxy LWT	
	User Name	hw_iotedge_device1	
	Pacaword		
	1 333000		
+ -	Revert		Cancel OK Apply

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。

Edit Connection Profiles		– 0 ×
M2M Eclipse		
local mosquitto	Profile Name local mosquitto	
	Profile Type MQTT Broker	
	MQTT Broker Profile Settings	
	Broker Address 124.70.60.6	
	Broker Port 7883	
	Client ID hw_iotedge_device1_0_0_2021050810	Generate
	General User Credentials SSL/TLS Proxy LWT	
	Enable SSL/TLS V Protocol TLSv1.2	•
	CA signed server certificate CA certificate file	
	CA Certificate File C:\Users\Administrator\Desktop\fsdownload\MQTT	莫拟器\plt-devic
	CA certificate keystore Self signed certificates Self signed certificates in keystores	
+ -	Revert	ancel OK Apply

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,loTDA也会显示在线。

步骤5 选择publish输入topic地址。

Topic: Soc/devices/hwiotedgedevice2/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": "properties": "status":' "event_time }] }	"power", : { "error" }, ": "20210508T173342Z"	
WQTT.fx - 1.7.1		– 0 ×
File Extras Help		
local mosquitto	Connect Disconnect	۵ 🔴
Switch Fountain ON Switch Fountain OFF	Surpts Droker Status Log Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/report Publish Soc/devices/hw_jotedge_device1/sys/properties/report Soc/devices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/properties/lipedavices/hw_jotedge_device1/sys/prop	QoSO QoS1 QoS2 Retained O(*

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。

最新上级数据		查看历史数据 ⑦	查看全部屬性	С
	[]			
	智无政策			

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, 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。

A MOTT fx - 1.7.1	Ν			– n	X
File Extras Help	142				
local mosquitto	Connect	Disconnect			A O
Publish Subscribe Scripts Broker S	tatus Log				
\$oc/devices/hw_iotedge_device2/sys/commands/#	▼ Subscribe		50 Qo51 Qo52	Autoscroll	0°*
\$oc/devices/hw_iotedge_device2/sys/commands/# Dump Messages Mute	(1) \$oc/devices/ Soc/devices/hy	w_iotedge_device2/sys/commands/request_id=bec53b02-c073- jotedge_device2/sys/commands/#	4da8-8ca4-f8010c6391fd		Qo
Topics Collector (0) Scan Sto	p 00 Soc/device Soc/devices/hy	:/hw_iotedge_device2/sys/commands/request_id=be iotedge_device2/sys/commands/#	c53b02-c073-4da8-8	ca4-f8010c63	91fd 5
	11-05-202	19:33:03.70383394			QoS 0
	{"paras" nd_name"	"restart","object_device_id":"hw_iotedge_de "power_control"}	vice2","service_id	":"power","	comma
		Payload	decoded by Plain Text I	Decoder	•
🖪 A 🛱 🥭 関 🚨	4 📮 💿		~ P		

----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

```
iot-module-samples D:\workspace\git\ModuleSDK-Java-master\iot-mo.
  > 📄 .idea
  erp-integration
     META-INF
     src
        main
           🗸 📄 java
             com.huaweicloud.samples.sdk.iot.erpintegration
                   C ApiController
                   C Application
                   C AuthFilter
                   ConfigController
                   ConfigService
                   ItIntegrationService
             META-INF
                   MANIFEST.MF
             resources
                application.yml
                log4j.properties
        erp-integration.iml
        m pom.xml
```

- 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\"}";
```

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

<u>∕</u>∧ 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=0A1B 0C3D

其中:

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lTIntegration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。 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<mark>制作镜像包或插件包</mark>。

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。

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<mark>工业资源</mark> <mark>包</mark>。

绑定方法如下:

步骤1 在左侧导航栏选择"loT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。

10 T32398	边缘节点 ⊕ 引导流程					ধ্ব ল	品約5 🍞 使用消除 注册订成 我的资源也
总定	引导流程						
边缘节点	-1		-2		3		4)
应用管理	創建して边缘节点		添加边缘设备 第520年には10月間間 第550日第第520日には	11 / T + 60.04			
设新建模	 在边缘节点上部署系统软件 		要将收留的7.42家后点。 再将物理设备接入边缘节点。	R LOVER PAREND NO ME V	上报的清洗规则管理。	ACTIVATION OF THE CONTROL OF THE ACTIVE	9.0.8820012288251308884。2.588286392825,20193 直过这級行师的本地控制台,对边缘设备进行本地管理。
数描述法							
路由转发							
						在线	 ・ 請給入市点名称 Q C
	节点名称	节点类型	资源包类型	资源包ID	秘念	主机名网络	操作
		经重版	-	-	在线		意 時
		经量板	-	-	在战		89
		标准版	-	-	在城		
		58E	工业子系统数采 Lite 版	66b412c2-5e05-4e12-ad	01-8364d6a 在副		
		标准版			在城		教師
		经最短	-	-	石线		#19:
	yxh1230	标准版		-	在城		89

步骤2 先设置"工业子系统数"再"绑定资源包"。

< yxh1230						删除节点 刷新
可点風波 可点風波 应用模块 可点風波 如用模块 () 如用模块 () 如用模块 () 如用模块 () 如用模块 () 如用模块 () 如用機块 () 如用機块 () 如用機块 () 如用機块 () 如用機块 () 如用機块 () () ()	52 - 在総物首中 - 市成の芝麻平台論語中、三位的感染物明合		 法定法律の条 111 血月環境(法行中/所常) 実際対策条件所向) 	7 3/3 0	CPU 内存 C 截盘	0.13% 34.83% 20.04%
2000年7 三部配道		規格配置 込まれの発展 約 当該方式 第 当前方式 第 王月高 bc 月前の予約 41 日本市式 64 人加速子 第	制任 円 Androit locationum Dplart1 10 12 9 100; docker0: 172.17.8 1; br-687769419404 172.20.8 1 Hoto 7 5.2004 (中国 1922) 281 現実		取果接入信息 間で予約数2 何定思想を 列時的月 工业予約数2 可定思想を 目前回到 工业ので総数数 工业ので系統数	12 28 (204) 90 90 90 12 90 12 12 12 12 12 12 12 12 12 12 12 12 12

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。

< yxh1230											删除节点	刷新
日田原道	节点概况							_				
应用模块 数采配置	高盘		绑定工业数采资	源包				×	7 CPU		0.13%	6
边鄉设备		-5247TH1238++, 242263814393	请您根据需要选择,并点 <u>工业</u> 数据采集服务	r击确认,如果没有符合要求的资源包	,请先购买工业数采服	895			0 C 22		20.04%	6
数据配置				资源ID	规临	资源状态	FiteBbii					
主筆記畫	基本信息 节点名称	yoth1230	۲	aa3aa302-a09a-4d72-8437	Lite	运行中	包年/包月 225天后到期		数采接)	(信息		
	节点 ID 计原模式:	673884310457298944 🗇 按照计算		5bca03bd-6d9a-4863-bbda	Lite	运行中	包年/包月 150天后到期		柳定日	-96,6530 19941	(未绑定)	役置 御定
	क्षेत्रियोग्याः	2021-12-30702:56:45Z		_	NI BRAN				ENDR T.W.F	1间 - 系统数		-
	高线缓存配置 上报优先级	实时数据优先		terf to Re Co.		09		_	1912E	19940	(朱郎定)	鄉走
	存值周期 重存大小	7天 2048 MB		現格	16 cores 983	23 MB			1 39494 TAR	1回)T 设飾数		(2) 11
	边缘软件	200 升级至最新新大		Mauer	77.28.36				TW	DT 系统数		0.00
	更新时间	2022/12/13 05:27:44 GMT+08:00										0
	¹⁰ 应采购 容器运行时版本	x86_64 18.06.3-ce										6
	可靠性配置 可靠性取到	Φ										

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。

() 五成時五時千台進第中, 五位数編集時間的		田敷之後 20 年 田敷 20 代表 (25 行年/月第1) 学家 2 新 2 新 2 新 2 新 2 新 2 新 2 新 2 新 2 新 2	1 3/2 0	CPU 内存 C 磁盘	3.3 27.0
基本信息 亚成名称	規格配置 边缘节点网型	经删版		数采接入信息	
石点ID 注意構成: 包閣購 計畫構成: 包閣購 台間時1回, 2022-11-21707-30-367	200万式 节点设备	案明		绑定资源包 至期时间	(未練定
1998月9日	主机名 网络(网卡:IP) 强作系统	zw-łotedge obumtu 18.04		工业子系统数 術定資源包 至期时间	Lit 2023/07/26 23:59:59 GM
存值回明 7天 派存大小 2048 MB	规核 AltDEE卡	4 cores 15563 MB 未搭配		工业 OT 设备数 工业 OT 返给数	
2019年4月7 当前版本 2.0.0 <u></u> 升坂至麋新版本 更新时间 2022/12/13 08:15:28 GMT+08:00					
17点原約 x86_64 等構造行町版本 20.10.7 771集社 57日間					
可继性级别 中					

----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。

	华为云 拉制台	Q 北建	Zi V		邀家 Q 勝用4	中心 潮	19 IA	企业 智	窟 支持与服务	中文 (簡体)	hwstaf	f_pub_loTE	idge 🗹	a
Ξ	API网关	*	API分组 ③ 即还可以创建47个分组,最多可创建50个分组。										estoria	
ය ^ණ	元献劳善 KCS									清辅入分组名称			QC	
.00.	APISHE		分组名称	状态	描述	100	動利用			操作				
۲	API管理		APIGroup_hw_lotedge_ITIntegration	朱上端	注集成报告	202	21/05/20 17:17:	50 GMT+08:00		API管理	49	<u>9</u> 8 +		
0	沈重控制		APIGroup_tpgg	未上規		2020	20/11/05 17:11:	:07 GMT+08:00		API管理	940	更多 •		
ß	访问控制		EchoDemo	未上業	APIG Demo	2020	20/11/05 10:04	38 GMT+08:00		API管理	99	<u>8</u> 8 +		
٢	环境管理													
4	签名密钥													
Θ	VPC通道													
۲	自定文认证													
0	请用API	*												
۲														
60														
														e
													r	0
														9
														0

编辑分组

编辑分组		×
* 分组名称	APIGroup_hw_iotedge_ITintegration	
	支持汉字、英文、数字、中划线、下划线、点、斜杠、中英文格式下的小括号和冒号、中文格式下的顿 号,且只能以英文、汉字和数字开头,3-255个字符。	
描述	it集成服务	
	6/1,000	
	确定取消	
	6/1,000 确定 取消	

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。

API管理 ⑦ 您还可以创建191个API,最多可创	醴200个API。							REEDAM	問入API	發出API
107 【日常任祭】 前往 金贵中心 領取日常住	1条、直向豆焼好礼									
授权 双布 下线 副	* 0					APIGroup_hw_lotedg ▼ 解釋运行环境	▼ API ID	▼ 請輸入API ID	Q	₩¥ > C
名称/ID	运行环境	类型	安全认证	所属分组	描述	标签	更新时间	操作		
									更多,	
APL:seg5 205989712a214e48864d22b73f0b2c	RELEASE	公开	APPIAE	APIGroup_hw_lotedge_lTintegrati	(2018)195前		2021/05/21 14:38:22 GMT+08:0	0 接权 发布	更多 •	
									贾多 •	

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。 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。

< SRSHAPI										
() ##### ——	— 🕗 EXAMER ——— (3) EXEMBS ———	- (4) ZOMBIN								
定义API请求										
城市	2446c70279ee4efa9a60d786a0d889ec.apig.cn-north	1-4.huaweicloudapis.com								
请求协议	HTTP HTTPS HTTP&HTTPS									
	支持WebSocket									
★ 请求Path	/v2/(project_id)/edge-nodes/(node_id)/ias/(ia_id)/configs/									
	请求oath可以包含请求参数,请求参数使用D198日,例如0a/b	1、也可以通过配置"+"号体前端正配。 例	Ω: /a/{b+}							
匹配模式										
* Method	GET +									
WINDOW CONT.										
SEPERAL(COVS)	开启路域,请能往了解详情									
入参加文 ヘ										
请求中的所有参数	、包括Path中的动物意教、Header意教、Query意教、意教名称6	\$1还睡—。								
④ 添加入参5	型文 影除 您还可以出建47个入参参数,每个API信	多可创建50个入参参数。								
● 参数名	参数位置	英臣	必填	默认值	牧単	参数限制	示例	腦述	操作	14
biel -	PATH	STRING	景		-		-		997E 1979	- 6
node_id	PATH	STRING	是						1951 B59	- 4
project_	d PATH	STRING	是						961 BM	- 0
										E
									1-3 7-3 200	adî.

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。

K SRSEAPI						
) ####	(2) EXAMBA —— (3) EXCHARA —— (4) EXCERNA					
定义后端服务						
后满服务类型	HTTP/HTTPS FunctionGraph Mock					
か可以現れ新聞日期	来能导化后跳空之,每个振动后跳光开空之多个振动条件,只有满空把空的振动条件的清末	tommuzzaizzmecza,				
加速的创建5个后期算	241					
		+ 1010000000500				
基础定义						
ModdZEWER		259(12)48 現石現在, 町に(18)1594646頃				
后期以注	のの					
MAN BOTT	ante-o-filletan melandrikan iziti latitoo					
Reases						
入參名取	入創作業	入会部型	后接参数名称	后端参数位置	80	۰
21.833.857	·					0
						0
TIERS ()	-					ß

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:

く 编辑API	
① 基本信息	
定义后端服务	i de la construcción de la constru
后端服务类型	HTTP/HTTPS FunctionGraph Mock
您可以添加策	后族来差异化后族主义。每个黄嘴后族允许主义多个策略条件,只有满定相应的策略条件的语求才会被按定到论策略后满。
您还能创建5个	当時第
默认	Sije + 78.005898/5538
基础定义	
协议	4TTH •
请求方式	GET •
使用VP	通道 使用 不使用
	使用VPC直道访问您鄉層在VPC内的服务
* 后講服:	把社
* 后端清	Wath 海输入后端请求路径
	请求Path可以包含请求参数,用[]初记,比如[getUserInfo/(userId),支持*%等特殊字符。
* 后端超8	t(ms) 5000
后满认	
	使用自定义认正控取进风信意来访问后属
	金川山路40小ビ絵類なみ後、 営用会長的1万倍会長、 GTT11山路40小
og ("APIE	ריין ארגעינייג איז אוגעיזע איז
后端服务	

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。

▲ 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

く 编辑API			
1 基本信息	— (2) 定义API请求 —	③ 定义后端服务	4 定义返回结果
返回结果基础定	2义		
★ 成功响应示例	200		
			3/20,480
失败响应示例	400		
	L		3/20,480
成功响应示例: 失败响应示例:	自定义。 自定义。		
5. 发布API			

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。

API管理	② 憲还可以創建191个API,最多可創建20	00°fapi.							REZAPI	₩λарі	특H:API
нот	1000 (日本伝教) 新注 会然中心 夜田日本伝教、北美田的外心										
18	反 发布 下线 删除	Ø					APIGroup_hw_lotedg ▼	• API ID	▼ 谐输入API ID	Q	52 × C
	名称/ID	运行环境	英型	安全认证	所属分组	描述	标签	更新时间	操作		
	APL_seg5 205989712a214e48864d22b73f0b2cec	RELEASE	公开	APPILIE	APIGroup_hw_iotedge_ITintegrati	数据传输		2021/05/21 14:38:22 GMT+08:00	通収	发布 更多 +	
										5│更多 ▼	

----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。

创建应用		×
* 应用名称	App_it_erp	
	支持汉字,英文,数字,下划线,且只能以英文和汉字开头,3-64字符	
描述	erp子系统la7	
	9/255	
	确定 取消	

步骤2 绑定API

这用管理 ⑦ 想还可以创建47个应用	8. 最多可创建50个应用。			क्षर्यप्रद्यम
			应用各称	• 湖航入应用名称 Q C
应用名称	应用ID	绑定API数量 描述	合国建計问问	操作
App_it_erp	ac817ea3d6464dc39356a8f24c5e92f1	0 erp子系统la7	2021/05/21 15:54:17 GMT+08:00	SIDEAPI WRIE BURK
App_95ml	0eaa585e10044eaca8b75b28b0892dab	0	2021/05/21 10:08:17 GMT+08:00	部5至API 網羅 新時
App_iotedge	1e450e5ac463477dbc2b8c375cd56b1b	4	2021/01/13 17:25:37 GMT+08:00	銀冠API 編編 曲時

勾选前面创建的API,单击绑定。
绑定API					×
授权环境 ⑦	RELEASE	•			
选择API	所有组	▼ 请输入API名称	Q	C	2
	1 只有APP认证的API才可以	被应用绑定			
	API名称	运行环境	分组	描述	
	. 401		ADICIONIS husianadas ITana		
	API_seg5	RELEASE	APIGroup_nw_loteage_llinte	刻(山) (立相)	
	5 ▼ 总条数:7 <	1 2 >			
		绑定 取得	3 4		

----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。

く 添加数据接收端点						
端点信息						
* 読点名称	北向应用NA2					
* 蔬નD	na2	□ 外部	方问需要身份鉴权			
* 数据接收地址	http://2446c70279ee4efa9a60d78	6a0d889ec.apig.cn-north-4.huawei	cloudapis.com			
接入方式 ⑦	ROMA APIG					
	ROMA (应用数据集成平台):如果需	要通过ROMA转发业务数据,请您还	EROMA创建集成应用,完成API创建	8、发布等操作。		
	* 集成应用Key d5647f3:	87904d17bc5c4cdb279c80bc				
	★ 集成应用Secret b0ad33a	337e349da923526f3cd8c5dfd				
鏡点描述	na2					
继定节占						
请选择需要以下哪些	边瀑节点将其本地的业务数据发往该满	ā		所有状态	▼ 請輸入节点名称 Q	С
□ 节点名称		节点类型	状态		主机名/网络	
Mw_ioted	je_node2	标准版	在线		192.168.0.180,172.20.0.1,172.17.0.1	
omtest		标准版	未安装			
yinwei_tes	st	标准版	在线		192.168.0.113,172.17.0.1,172.20.0.1	
test		标准版	未安装			
standard_	1620465371	标准版	未安装			
gefefwe		高级版	未安装			
hw-lotedg	e-node1	标准版	高线		192.168.0.237,172.17.0.1,172.20.0.1	

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")。

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!

APG语(APRGroup,Inne, Jonedge, (Trintegration <u>戦変</u> APRGroup, Inne, Jonedge, (Trintegration) 戦な変更 APRGroup, Inne, Jonedge, Inne, Inne		
1988年 - AFOnes Jou straft, Filmspring 企 個数100 20176/20111139 04F-0600 〒1985 - 3440-02079e44-04000756000050-0410-0-05-0-5-0400400400400-00 年7年年の日本学校のの中学校のない、年生年の日、0000、美行いば彼らないたちゃから出来。	9日0 2446/02/78ee4d44466/786466886ee 15点 中上当 第二者 mindladik	

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)

API网关	<u>6</u>	应用管理 / App_it_erp						意思AppSecret 编辑 图印 C				
实例概念		应用名称 App_R_erp 🖉				应用D at	c817ea3d6464dc39356a8f24c5e92f1					
开放API	- [AppKey d5647f3387904d17bc5c4	iedo279e80be 🗇			AppSecret b*	d 💿 🗇					
通用AP1	<u> </u>	(部計時) 2021/05/21 15:54:17 GM	17+08:00			描述 er	rp子系统a7					
应用管理												
已购买API		御念的API列表 AppCode										
		and the District And Dist										
		© açılaraı Demokrationala						(BR) APISIS Q				
		API名称	授权环境	分组	描述	授权者	網定时间	操作				
		API_qvs3	RELEASE	APIGroup_hw_lotedge_ITI	配置時以	API提供書	2021/05/21 15:58:06 GMT+08:00	illidari Milli				
		AP1_gude	RELEASE	APIGroup_hw_iotedge_ITI	网络配置	API提供會	2021/05/21 15:58:06 GMT+08:00	RENCAPI MER				
		API_seg5	RELEASE	APIGroup_hw_iotedge_ITI	\$2 第 件输	API提供者	2021/05/21 15:58:06 GMT+08:00	UTUCAPI NUM				

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。

提示		×
★ 边缘应用	\$edge_apigw •	
* 选择版本	• ?	
	确认取消	

1. 部署\$edge_apigw

2. 部署It应用

提示		×
★ 边缘应用	iot-erp-integration •	
★ 选择版本	1.0.3 🔹 🕜	
	确认取消	

3.最终节点部署了四个应用

▲ ① 都要边境应用 电点实验运动运动网络边缘运用。应用在电池电子运行参加加44亿分型资物缺端							
模块Id	所属应用	版本	应用类型	实例状态	操作 ③		
sys_edge_agent	Sedge_agent	1-0-9-x86	繁统必法	运行中	升级		
sys_edge_apigw	Sedge_apigw	1-0-10-x86	系统可选	运行中	翻除 升级		
sys_edge_hub	Sedge_hub	1-0-22-x86	系统公选	运行中	升级		
user_iot-erp-integration	iot-erp-integration	1.0.3	用户定义	运行中	勤除 升级		

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:

##### 単为云 控制台		建立 Q 義用 共源 工学 企业 製業 支持与服务 中交 (現体) heatalf pole)。FEdge 日	a
API Explorer	loT边绿 ●★ 产品详情	InvokeG4Proxy 第30手指列機 代码系列 構成品種 文档 和成历史 相利法語 法最示例 GET方法装件代理	_
总流 APt大赛	请输入中/英文关键词搜索API Q	● 那份接口適用涉及五产品计表、素種類操作 不再成長 ● 型品環境活用、APE-phones金使用已整要指导的超对Tokenty行像本、部分接口调用涉及五产品计量、重要值加修、如果使用Arcys项用接口、通用金 主要指导	×
调试历史	应用管理		_
版本历程 • 。	应用版本管理		
所有严品	节点整理	a Region ⑦ 开通IoT服务所在区域	
我的关注		单心北府五	
弹性云服鼻器 🔹	(0)(E)#	Headers ware	
云解析服务	设备管理	a X Auth-Token ⑦ ≓ DB#JAK/SKUE	
分布式数据库中间件	外部與体管理		
应用管理与远维平台	北向NA管理	Params	
就服用引擎	南向IA配置项管理		
1214 H B-2800	北向HTTP请求代理	a posses 40000000000000000000000000000000000	
设备接入	InvokeDeleteProxy	* node jd 🗇	
loTizzi錄	InvokeGetProxy	<u>まだの</u> 	
	In solution Distantion Diseases		
	involceratorieroky	tring	
	InvokePostProxy	+ Ng Un (2)	
	InvokePutProxy	1997	-
		Body 参数输入区域 学 印刷功文字输入	-
调试模块选择区域			1
			反携

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。



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。

巴介属或 应用爆快 数示起服 边缘设备 数据配置 边接电量	 セル協同 在 化	3.時公時公開 (11) 日本時間(注)(今(万年)) 本時の建築(水和日)	1 CPU 3/2 シー 内方 - 0 C 注意	0.5% 3.36% 25.66%
±80.0		地球で和型 地球で和型 取りたて 取りた で 取りた 和 和 和 和 和 和 和 和 和	数字接入信息 服行学系統改 研究課題も 調助計測 工造って新統改 工造って新統改 工造って新統改	20 20 20 20 20 20 20 20 20 20 20 20 20 2

ia_id:lA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。

节点概道	横块管理 数据流转配置 3	数采配置						
应用機块	20国成用 节点安装成功后请部署》	边缘应用,应用在节点处于运行志时即转化	为"业务模块"					С
SUHRCE D	模块id	模块名称	所属应用	版本	应用类型	实例状态	操作 ⑦	-
数据配置	✓ sys_edge_hub	sys_edge_hub 🖉	Sedge_hub	1-1-29-lite-x86_64	系统必适	运行中	升级	
运程维护	v sys_edge_agent	sys_edge_agent 🖉	Sedge_agent	1-1-14-lite-x86_64	系统必遗	运行中	升级	
主旨配置	v sys_edge_omagent	sys_edge_omagent 🖉	Sedge_omagent	1-1-14-itte-x86_64	系统可透	运行中	停止 勤除 升级	
	✓ sys_ot_dc_opcua	sys_ot_dc_opcua 🖉	<pre>Sot_dc_opcua</pre>	1-0-6-standard-x86	系统可远	日停止 ②	启用 删除 升级	
	✓ sys_ot_dc_iec104	sys_ol_dc_jec104 🖉	Sot_dc_lec104	1-0-0-standard-x86	系統可透	Efi	麻用 勤除 升级	

ia_uri: IA应用的接口地址,/api/get

填写完成后单击"调试"

API Explorer 忠流 APTX3 頃武历史 合* 所有作品 > 約約2952 日	IoT边場 产品序構 請能入中/完文关键词使求API 应用影理 应用版本管理 可必要理	•* Q	Involved NC0 p.0.01 GET/Jahr/12 NC0 p.0.01 Ø HOUSCARP/02/P.Mir.R. RENKO 749500 Ø ROSK NERDUE RADIO RESUE RADIO NESTURE	代田示明 概章 一概章 直实請求 https:// a_uri=%2	取扱業業 支払 J <thj< th=""> <thj< th=""> <thj< th=""></thj<></thj<></thj<>	评价 〇 Japi?i
弹性云服务器	機块管理	Ť	Headers	请求数据		
云解析服务	设备管理	*	• X-Auth-Token ⑦	200-mail	Jane 1	
分布式数据库中间件	外部实体管理	*	**************	1	alacu.	
应用管理与运进平台	北向NA管理	*	Params	2	"User-Agent": "API Exploren", "X-Auth-Tokan": "***",	0
微报务引擎	撤向IA配置项性理		∗ project_id ⑦	4	"Content-Type": "application/json"	
企业项目管理服务			0500		8	
税或直播	北向HTTP请求代理	^	∗ node_id ⑦	脑齿线图		
设督接入	InvokeDeleteProxy		595			
IoTs238	InvokeGetProxy		∗ ia_id ⑦	明应头	戦団体	
	InvokePatchProxy		us 🗱 🗱 🗰 n	2	"code": 200,	σ
	InvokePostProxy		∗ia_uri ⑦	4	"mog: "Getlest Success", "data": ("fourbace", oull	
	InvokePutProxy		/whi/dur	6	"time": "2021-05-25T11:30:19.979+02:00",	
			Body 全社 12時22原準編項 JSON ⑦	7 8 9 10	"tara": "nolla": "uses	日本の

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	
ØSITAJ	
<pre>@RequestMapping(value = "/orders", method = RequestMethod.POST)</pre>	
@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, ten);	
}	
log.ino(new sting) receive data	
f catch (LACEPhole 7)	
linally (
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");	
}	
}	
3	

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。

:: Spring Boot :: (v2.1.3.RELEASE)	
2024 0F 07 44-20-40 767 THEO 4500 F	
2021-05-27 11:29:10:707 INFO 1599 [main] C.nuawercloud.iotsamples.na.Application : starting Application on ZXI-ecs-10t with Pib 1599 (Momertoled	ge_na-
1.8-SWAPSHOL, Jar Started by root in / nome)	
2021-05-27 11:29:18:/09 INFO 1599 [main] c.nuaweicloud.lotSamples.na.Application : No active profile set, failing back to default profiles: defau	Iτ
2021-05-27 11:29:19.668 INFO 1599 [main] 0.5.D.W.embedded.tomcat.fomcatWebServer : fomcat 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 optima	l perf
ormance 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

IoTEdge_Demo D:\workspace\IoTEdge_Demo
> 🗅 .idea
✓ □ src
🗸 🗀 main
🗸 🗀 java
Com.huaweicloud.samples.sdk.iot.modbusdriver
© Device
© Main
© ModbusCfg
© ModbusDriver
© Property
© Service
✓ C⊇ resources
{} device_ids.json
© log4j.properties
{} modbus.json
<i>m</i> pom.xml

> 🕞 < 1.8 > C:\Program Files\Huawei\jdk1.8.0_272
> 🖙 iot-module-client-2.1.17-dependencies
>

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,slaveld及能力定义。
Main	主启动类。
ModbusCfg	modbus配置。
ModbusDrive r	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

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。

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<mark>注册边缘节点</mark>。

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"。

loT边缘		682733
总派 边缘节点 应用管理 设备建设 和 政疾清洗 路由转没	地域介绍 世報時天中中、第一般具有目間が方前中辺の信頼台集時方ンーが不高。 定時時天中中に第一部は東京市時代記録が出来。 「大学会書所では第半空を上記なかな道見、「大学会を出行者面積後」、名誉時子及一品構成」Profee、豆文Profee、低子台港等点から使うため道 全、命令場合見、特別一部に加入した。 文書作が知られて、「大学会」 (金融版入版51 ・ 小田子文成51 ・ 公報版入版51	

步骤3 填写参数信息,如图所示,单击"立即创建"。

创建产品			×
* 所属资源空间 ?	edge	•	
* 产品名称	driver_demo		
协议类型	MQTT	•	
* 数据格式 ②	JSON	•	
厂商名称	hw		
*设备类型 ⑦	modbus2mqtt		
	立即创建 取消		
			_

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"。

新增服务		×
★ 服务ID	light	
* 服务类型	light	
服务描述		
	确认取消	

步骤5 展开服务列表,添加"switch"属性。

10

修改属性		×
* 属性名称	switch ////////////////////////////////////	
属性描述		
* 数据类型	int(<u>輕型</u>)	
* 访问权限	可读	
取值范围	0 - 65535	
步长	0	
单位		
	确认 取消	

----结束

4.3.6.5 项目打包

打包参考<mark>项目打包</mark>

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<mark>制作镜像包或插件包</mark>。

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

4.3.6.7 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。

₩ 华为云 控制台 ♀ 乌兰家布・二字三	▼	林中文 CDNN_I
=	〈 添加边缘应用	
۵	O KNESTRER	
ම 0 & ය	* 部版の式	
© ⊘ 	23時期時 CFU 記録	
	英級左置 ・ 道行命令 ・ 道時の単 ・ 道時の単 ・ 道時の単 ・ 道時の単 ・ 道時の単 ・ ・ 道時の単 ・ ・ 	
	> 环境交量 容器运行环境中设定的交量,可以在应用邮客后传点、为应用感供吸力的观点性、设置容器运行环境中的系统环境变量。	

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。

<		制除应用
sdfsdfsd675 用种型义 功能用途本地子系统集成 台議时间: 2022/07/26 16	11.95 GMT+00.00) 編述 5475 1503規範:	
版本列表 ③ ⑦ 〇	版本信息	
● 5075 ×86_64 0	5675 ● 已決存 ② 不交均余部署 部語が式: 容器化振客 王功楽期後 x68_64 上穴薬用能引用 2022/07/26161115 OMT+08:00	85201 (1820) (182) (184)
	容祿能置 純常能置 部層能置 实例列表(0)	
	基本侦察	高级配置
	凯泰指径: 	运用配账 先说,容器没有访问主机设备的权限
	容器规格	环境变量 未设置
	CPU記録: 不由講習篇 不裝制使用	>款請得確: 1 个響 並署
	內容能統 不由續號篇 不限射使用	外挂设备: 未设置
	Attrast-mass: 不現制	健康检查: 広用存活 不配量 広用业务 不配置
		還行命令: 未设置

----结束

4.3.6.8 部署应用

部署应用,具体请参考<mark>部署应用</mark>,进入节点详情页安装应用。

<									删除节点 刷新
节点概况	10141-00130 No.45117-44-30199	Marri 21109							
应用模块	保快管理 第3日前元时间2日	家C++用C <u>巴</u>							
数字配置	部西应用 节点安装成功后请部间	副边缘应用,应用在节点处于运行态象	提示				^		C
边缘设备	根iRid	模块名称	功能用途	Reference	协议解析	本地子系统集成	用类型	实例状态	操作 ⑦
救援配置	✓ sys_edge_hub	sys_edge_hub 🖉		同关管理	混合应用	数据采集	统必遭	运行中	升级
近程地的	✓ sys_edge_agent	sys_edge_agent 🖉	★ 边缘应用			* ⑦	统公选	遗行中	升级
主筆配置	v sys_edge_omagent	sys_edge_omagent 🖉	* 选择版本			* ⑦	统可透	运行中	停止 翻除 升级
	v sys_ot_dc_opcua	sys_ot_dc_opcua 🖉	機块名称				她阿远	已停止 ⑦	启用 翻译 升级
	✓ sys_ot_dc_iec104	sys_ot_dc_lec104 🖉			2021		统可适	日停止 ③	启用 翻除 升级
	10 • 思惑歌: 6 < 1 >				909				

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

节点概览			
应用模块	边缘设备列表 批量注册 批量删除		v
数采配置	添加边缘设备 批量删除		提示
边缘设备	设备名称	设备ID (点击进入设备详情)	
数据配置			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(你愿意点同
主备配量			归属服务实例 IoTDA默认墨础版
远程维护			旧属资源空间 43424
			* 新聞产品 int edge MOTT T C
			没有可能不是2 请师往说备接入服务增加自定义产品,并定义设备功能前 往激加产品
	10 ▼ 总条数:0 < 1 >		* 设备标识码 清输入设备标识码 与代码内容保持一致
			设备名称 清给入设备名称 ⑦
			縱块 ID ⑦
			password mqtt设备接入平台密码

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<mark>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</mark> 设备状态?

图 4-8 边缘设备列表

应用模块	边缘设备列表 批量注册 批量删除		
数采配置	添加边缘设备 批型删除		
边缘设备	设备名称	设备ID (点击进入设备详情)	所属产品
数据配置 主备配置	test11	test11	iot_edge
远程维护	10 🔻 总条数: 1 < 1 >		

设备接入	设留管理 / 父设新详情 / 设备详情	從發展課題: / 安徽新建構: / 梁徽和建					
i	概述 命令 设备影子 消息跟踪	■ 命令 设备参子 消息期間 子设备 彩然					
Marking Units Marki	fsdf1111 🖉 • 未敢活 🕲 所服产品 op	cua_server					
总成 产品	所服的原型间 DefautVpp_hvstaff_pub_loTEdge_ 说明标记码 sdtsd1111 〇	និងកូនភ្លេម សម្តេចសម្តេចសម្តេចសម្តែល ខ្លួន សម្តេច សមត្ថកត្តិ សមត្ថការ ប្រឹងស្រុកសម្តែកសម្តេច សមត្ថការ ប្រឹងស្រុកសម្តែកសម្តែកសម្តែកសម្តែកសម្តែកសម្តែកសម្តែកសម្តែកស					
28 1	▲ 注册时间 2022/07/05 14:41:16 GMT+08:00 即注形士			节点类型 非重速设备 約00万大			
) 所有设备) 87组	设新版达 - 之	mmer − Britis − - Remes − L Scillo office+spende					
蚊围伸升级 设备CAIE书	織新上級政盟	NATION BEFORE O STORAGE C					
规则	派遣人服的名称 《政治 (1) (1) (20220705 14.41.16 GMT+08.00 《政治 (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)				请输入履性名称 C		
存储管理	Sconfig						
应控运输 上新	T	config_type	protocol	security_mode	security_policy		
「東線空目」		config_type	protocol	消息传输安全模式 "Nono"	安全策略修改 "Nono"		
28738	P	opcua-server-coming	opcua	None	None		
ADIORNIA	P						
设备发放	P	timeout 方法调用超时时间	url OPC-UA Server/驻接地址	version			
论坛求助	P	1000	"sdfsdfsd"	"1.0"			

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
```

```
@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) {
```

```
//PointsSetReq结构points为[pointld1,pointld2]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 dsld;

//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 pointld;

//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<mark>项目打包</mark>。

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<mark>制作镜像包或插件包</mark>。

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<mark>部署应用</mark>,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<mark>查看采集结果</mark>。

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

• OT数采代码解析

4.3.8.3 注册节点

注册节点,请参照<mark>注册边缘节点</mark>。

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<mark>项目打包</mark>。

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。
- 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。
- 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<mark>对象存储服务 OBS_快速入</mark> <mark>门</mark>。

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。

华为云 前付台 ♥ 北京四	· Rink Q 最同的成本 設置 全业 开发工具 新業 支持地裁長 工芸 際は中文 hostait_pab_lbTEdge LD
≡	く 添加边缘而用
۲	S KYH MARANAR —— (2) ISTANIR —— (3) ARBANA
0	
8	 (新語方式) 容優化影響 (公式) 安秋紀影響 (公式) 安秋紀影響<!--</td-->
Ð	1. 原用空红运动模型计发动模型物理师 2. 将应用图例打构成安排物文件,并上作到 订象种情描绘(OBS)
2	* 安装给地址
4	
J	送照规格
3	CP-128 中海営業
8	同時後期
9	内府起却 🗌 中國的國
۵	
Ð	
G	
9	高级形置
۵	→ 15次配置 通过105元 日報用売用の均差式企業的105元, 005日報が0500, 005日報が0500, 005人
\$	✓ 环境变量 容器运行环境中段地的皮量,可以在应用器器研磨法,为应用器件既大的现金性,设置容器运行环境中的系统环境皮量,
9	< 書寫調確 导始任罰主始時用回答中。目回回於時令後約9月0月日
	6-7

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。

dfsdfsd675 用户定义 邮用读 本统子至56年4 1 创建村前:2022/07/26	16 11 05 GMT+00 00 1655 5675 1010/2000;			删除应用
(本列表 ④ マ Q	版本信息			
● 5675 x86_64 0	5675 ● 已沈布 ④ 不支持多部署 部署方式: 容器化振客 □ 支持保険: x80_64 □ 上次編輯対応: 20220720	1 16: 11: 15 GMT+08:00	部署副的	开级实例 拷贝 下除
	古機配置 進点配置 部署配置 实份列表(0)			
	基本信息	海级配置		
	映参路径 :	,m 2002	关闭,容器没有访问主机设备的权限	
	容器规格	环境空量	未设置	
	CPU記録:不申请预留 不限制使用	数据存储	1个卷 查看	
	内容配統: 不由講習篇 不限制使用	外挂设备:	未设置	
	All0进卡配版: 不現制	使原位查	应用存活 不配置 应用业务 不配置	
		运行命令:	未设置	

----结束

4.3.8.8 部署应用

部署应用,具体请参考<mark>部署应用</mark>,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)。

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。

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_devic e	网关添加子设备,可批量添加	
edge_send_delete_sub_de vice	删除子设备,可批量删除	

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_ev ent	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_pro perty_get_rsp	用于网关响应平台的子设备属性获取请求
edge_send_sub_device_pro perty_set_rsp	用于网关响应平台的子设备属性设置请求
edge_send_customized_m essage	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_comman d	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_b roker_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_stat uses	释放子设备状态结构体

详细接口说明参见代码解析。

🗀 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_l en	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_na me	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。

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_nam e	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	-

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	unsigne d 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	
addSubDevicel	char*	添加成	{"devices":	
nfo		功的子设备信	功的子设备信	[{"parent_device_id":"c6b39067b03421 a48",
		息,兵 体格式	"node_id": "subdevice11",	
		见示例	"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	unsign ed 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。

参数名称	类型	参数描 述	示例
deleteSubDevice Info	char*	删功设息体不同的。一个是一个一个一个一个一个一个一个一个一个一个一个一个一个一个一个一个一个一个	<pre>{"devices": [{"parent_device_id":"c6b39067b03421 a48", "node_id": "subdevice11", "device_id":"2bb77-063ad2f5a6cc", "name": "subDevice11", "description": null, "product_id":"c6b3b34663d3ea42f6", "fw_version": null, "sw_version": null, "status": "ONLINE" }],"version":1}</pre>
body_len	unsign ed int	长度	-

表 4-49 参数说明

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

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_proper ty_set	ST_PROPERTY_SE T*	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_proper	ST_PROPERTY_GE	属性设	参见edge.h
ty_get	T*	置	

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。

表 4-53 参数说明

参数名称	类型	参数描 述	示例
sub_device_proper	ST_DEVICE_SHADO	属性设	参见edge.h
ty_get	W*	置	

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

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_c b	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_c b	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_delete d_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_ARRIVE D*	收到子设备属 性设置通知回 调
pfn_device_properties_ get_cb	FN_DEVICE_PROPERTIES_GET_ARRIVE D*	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,

```
"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	-

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。

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	["productld1", "productld2",]
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);
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_g et_rsp	ST_DEVICE_PROPERTY_GET_R S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	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

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

图 4-11 图示

模块自定义消息

京 NA 自定文演巻 Dot DA 自定文演巻 記定文演巻 記述文演巻 記述文演巻	 以IoT DA的自定义测息实现,对消息内容 IoT DA<->Hub主题以/modules/(moul 别为/modules/(moduleid)/customized /modules/(moduleid)/customized/out 下行自定义消息处理: 设备服人>边像Hub matt/modules/3rdapp/hi helio 边像Hub~服三方模块 matt/modules/3rdapp/customized/imputs/hi helio 上行自定义消息处理: 第三方模块→边像Hub matt/modules/3rdapp/customized/outputs/hi helio 近線Hub~设备接入 matt/modules/3rdapp/hi 	与敏透传,Hub按模块ld进行 leld)/开头,Hub<->模块∃ d/inputs/**和 itputs/**。	う转发 。 目题区分上下行,分
IÇ A BETTER CONNECTED WORLD	Huawei Confidential	45	🚧 HUAWE

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", "rostart"
			}
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

⚠ 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_STATU SES*	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";

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

表 4-80 参数描述

参数名称	类型	参数描述	示例
devices_statuses	ST_DEVICES_STATU SES*	指向该结构体的 指针	ST_DEVICES_STATUS ES devices_statuses = {0};
			edge_init_devices_sta tuses(&devices_statu ses, 2);
size	unsigned int	要查询的子设备 数,与24. 中的 devices_id_size 一致	2

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_STATU SES*	指向该结构体的 指针	edge_release_devices _statuses(&devices_s tatuses);

⚠ 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_DISCO VERY_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

门 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。

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。

门 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

- 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

门 说明

MyCApp为可执行文件,生成步骤可参考<mark>生成可执行文件</mark>

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<mark>编写高效</mark> <mark>的Dockerfile</mark>)。

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"]

- 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<mark>体验馆</mark>熟悉流程。

1. 上传镜像

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

 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。

戦 ②			(平): + (SEE): (1): (1): (1): (1): (1): (1): (1): (1	1. 上传销像
(1) 试验您参加容器模像服务使用体验调研,您里要的意见和建计	《最我们持续提升产品体验的漂动力,感谢您的参与!			A ×
快速入门				□ ^{操作指南} 点击这里
				如何创建组织 如何上传模像 如何制作模像正确包 如何表加研究
1 <1 <1 <1 <1 <1 <1 <1 <1 <1 <1 <1 <1 <1	2 猿像获取	3 应用邮署	4 更新镜象	如何收藏環像
您可以根据最身组织的解构,来 构建相应的法律管理。 会 建 编织	忽可以通过以下两种方式添加網 像 上传由肖编像 使用编像资源	CCE使用结要 使用云容器引擎邮署	受给配置规则,自动更新强定应 用的 机 像 配置确像更新规则	

🛄 说明

ŝ

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

7.

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

· · · · · · · · · · · · · · · · · · ·	↓⊒ ocker	所課组织 iotedge			版本数 J三	更新的间 [F 證作 2021/05/18 10:01:45 GMT+08 限身前的更步
上传镜 步骤一	像后请在 SWR 将镜 :	像设置为公开	Ŧ。			
□ ^{ģ∰38} □ ^{my_app_d} 步骤二	』E occhar ·	所展组织 iotedge			版本数 JΞ 1	開新時间 IF 操作 2021/06/18 100145 GMT+08 戦争音助同步
我的現象 / my_app_d	ocker				化 違言	
所屬組织/镜像名称	iotedge/my_app_docker		分開	其他		
日用版本数	1 2021/05/18 10:01:45 GMT+08:00		下戰次数	0		
9951	私有		0.6204	50%0 MB		
步骤三	:					

▲ 注意

这一步很重要,关系到后面能否正常部署应用。

编辑镜像

组织	iotedge			
名称	my_app_docker		③选择"公开"	
类型	公开	私有		
分类	其他			•
描述	清輸入镜像仓库推	苗述(0~3000		
			确定 取消	0/30,000

----结束

插件包打包

步骤1 插件包制作。

 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

L_***

⊢conf

│ ├──module.dat //sdk需要的密钥

——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 。

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。

构建插件包。
 压缩文件得到monitor-app.zip。

步骤2 插件包上传。

- 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)**。

<u>∕</u>∧ 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<mark>添加应用</mark>。

🛄 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"。

< 添加边缘)	应用
★ 应用名称	туарр
* 部署方式 ?	容器化部署安装包部署
	软件准备操作说明
	1. 根据您的业务需要开发边缘应用程序 ⑦
	2. 制作镜像,并上传到 容器镜像服务(SWR) 中。制作镜像请参考 Docker官网
* 功能用途	数据处理 协议解析 本地子系统集成
应用描述	可洗掉,只会许输入中文,盖文字母,中划线,下划
	0/255
版本配置 请	你配置该应用的版本信息,以及后面步骤中的相关配置内容
★ 应用版本:	1.0.0 立即发布
★ 支持架构	x86_64 🔇 🔻

🛄 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。

〈 │ 添加边缘应用						
① 应用配置 ——						
选择镜像输入方式:	点击选择 •					
★ 选择镜像	my_app_docker 透择镜像 边缘应用程序需要制作为容器镜像,并统一在容器镜像服务(SWR)中管理。请您先准备好镜像文件,然后在这里添加边缘应用					
<mark>*</mark> 版本	v1.0.0 v					
容器规格						
CPU配额	申请预留					
内存配额	限制使用 申请预留 限制使用					
AI加速卡配额	不申请配線 申请GPU配额 申请NPU配额					

步骤3 添加边缘应用-端点和部署配置

	使用MOTT Dealea做迷自らに、 给)送点伊主媒体投诉这自めTable、 给出送点伊主受送送自めTable(送点仍伊主志社北古正的MOTT Table)
אנש, במקפרות.	астрики і вокенциявалася, звіллянті секвентациявалоріс, звідряті сеседлявали оріс (делтіх) секцит-нандарики і оріс) "
輸入端点 🕐	请输入端点 添加满点
	input 💿
	Tanka A da 45
制工)通用 ①	
	output 🚳
部署配置	
重启策略:	
	总是里后 天败的重启 个里后
	当应用实例很出时,于没具正常很出还具具常很出,系统都会重新拉起应用实例

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

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

〈 添加边缘应用						
* 应用名称 * 節署方式(* 功期用途 应用版述	monitor-app ·					
	V/255					
版本配置	请你配置该应用的版本信息,以及后面步覆中的相关配置内容					
* 应用版本:	0.0.3 □ 立即发布					
* 支持架构	x86_64 •					

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。

< 添加边缘	应用				
① 应用配置 ——	— 2 软件和运行配置 —	③ 端点和	部署配置		
* 安装包地址	edge-monitor/monitor-app application_version_detail_s	.zip pk_label]	
application_c	:reate_processSize_label				
CPU配额	□ 申请预留				
	限制使用				
内存配额	□ 申请预留				
	限制使用				
AI加速卡配额	不申请配额 申	请GPU配额	申请NPU配额		

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。

门 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。

ot_test 用户定义 功能用油: 数据采集 创建时间: 2022/04/06 19	510.00 GMT+08.00 HELE							
反本列表 ④ ℃	Q. 版本信息							
3 x86_64.arm32 0 1111 x86_64 0	3 ◎ 米型帝 ②不因为全部署 部署方式: 国際化振客 〕 艾玛架町 x8L_64.xm32 上穴病藥財用 2022/05/05 10:52.07 GMT-00:00	668 (75) (560 (600)						
1.0.0 x86_64 0	古職配置 姚完配置 部署配置 实份列表(0)	お振む置 共同和定 が得む度 会所列的(0)						
	基本信题	游级配置						
	識態課程: swr.cn-north-4.myhuaweicloud.com/lot_edge_test/ot_test.v2	泡河配置:关闭, 容器设有访问主机设备的权限						
	客器规格	环境空量:未设置						
	CPUE想:不申请预留1不限制使用	数据存储: 1个 增 查看						
	内容能振 不申請預備 不限制使用	外挂设备: 未设置						
	Alterer-HEEEE 不現制	健康检查 应用存活 不配置 应用业务 不配置						
		這行命令: 未设置						

----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<mark>应用部署</mark>。

2. 添加设备进行测试(添加设备请参考<mark>设备接入边缘节点</mark>)。

门 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<mark>应用部署</mark>。
- 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

- | |--libmodule.so
- | L__***
- ├-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
- │ │ └─rootcert.pem //证书
- └────start.sh // 启动脚本(编写启动脚本可参考<mark>制作镜像包或插件包</mark>)



详细目录树如下图

MyCApp 〉 📇 main.c
법 🔲 Project 👻 😳 🗄 🌣 —
🖉 🗸 🖿 MyCApp C:\Users\l50018076\rep
🖿 🗸 🖿 conf
d module.dat 2021/7/23 17:08,
🚦 rootcert.pem 2021/7/23 17:0
🖆 sdk_log.conf 2021/7/23 17:08
✓ ■ include
edge.h 2021/7/23 17:08, 8.89 l
edge_daemon.h 2021/7/231
edge_error.h 2021/7/23 17:08
ibapr-1.so.0.7.0 2021/7/23 1 1
p liber/rto so 1 1 2021/7/2 1
libernat so 1.6.11 2021/7/23
libmodule.so 2021/7/23 17:08
libmodule-base.so 2021/7/2
libmodule-cipher.so 2021/7/
bibmodule-configs.so 2021/5
🕌 libmodule-http.so 2021/7/23
🙀 libmodule-httpconns.so 202 ¹
🛃 libmodule-json.so 2021/7/23
🐈 libmodule-mqtt.so 2021/7/2 2
🛃 libmodule-mqttconns.so 20 2
ibmodule-property.so 2021 2
ibpaho-mqtt3as.so.1 2021/
ibssl.so.1.1 2021/7/23 17:08,
ibzlog.so.1.2 2021/7/23 17:01
CMakeLists.txt 2021/8/3 17:28, 2
main.c 2021/7/23 17:30, 2.12 kB N
Scratches and Consoles

🛄 说明

更多Demo参见ModuleSDK-C Demo展示。

4.4.4 集成 ModuleSDK 进行数据处理

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。
```

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 loTD ,以下是具体配置中的参照。

1. 创建产品

创建产品						×
★ 所属资源空间	edge	•	0			
	如需创建新的资源空间,您可前往当前实例详情创建					
* 产品名称	myapp_mqtt					
协议类型	MQTT	•	0			
+ 浙居格式	JSON	•	(?)			
A BYNEIDTA						
★ 厂商名称	hw					
所属行业	无	•				
★ 设备类型	MQTT_Device		0			
高级配置 🔻	定制ProductID 备注信息					
				确定	取消	

2. 在新建产品后需要在产品页的"自定义模型"中添加服务。

		×
添加服务		
★ 服务ID	power	
服务类型	power	0
服务描述		
	0/128	
	确定取消	

3. 新增属性

新增属	生				×	
<mark>★</mark> 属性名称	status					
属性描述	<u><u>R</u></u>					
★ 数据类型	켈 string(字符串)		0/128		
* 访问权附	可读	可写				
★长度	10					
枚举值	normal,error			ĥ		
新增命令。		确定	取消	12/1024		×
新垣叩文 ★命令名称	power_control]			
下发参数	新増輸入参数 参数名称	数据类型	描述	操作		
	restart	string(字符串)		修改	删除	
(5 ▼ 总条数:1 < 1					

4. 亲

* 節令名称	power_control			
下发参数	新增输入参数			
	参数名称	数据类型	描述	操作
	restart	string(字符串)		修改删除
	5 🔻 总条数:1 < 1			
响应参数	新増响应参数			
	参数名称	数据类型	描述	操作
		暂无题	長格数据	
		确定取消		

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。

#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。

					所有资源空间	• C
产品名称	产品ID	资源空间	设备类型	协议类型	操作	
hw_iotedge_modbus	6099fb6aaa3bcc02c022ef18	hw_iotedge	modbus-device	Modbus	查看 删除	
hw_iotedge_mqtt	60988d94aa3bcc02c0200667	hw_lotedge	MQTT_Device	MQTT	查看 删除	

或者在产品详情页查看。

物联网平台		产品 / myapp_mqtt					
基础版 默认		myapp_mqtt ID: 60e6c0a5f4b59002867fcb02 注	册设备数: 1				
总统							
产品		产品名称 myapp_mqtt	所属资源空间	edge			
		设备类型 MQTT_Device	协议类型	MQTT			
定期		数据格式 json	创建时间	2021/07/08 17:08:53 GMT+08:00			
規则	*	厂商名称 hw					
存储管理							
监控运维 •	*	模型定义 在线调试 Topic 管理					
資源空间 IoTDA实例		添加服务 导入库模型 上侍模型文件	Excel导入				
产品文档	æ	へ 服务ID: power □					
API检索和调试	e .	服务类型: power					
设备发放	æ	服务描述					
		添加屬性					
		属性名称	数据类型	访问方式	描述		
		status	string(字符串)	可读,可写			

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。

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

〈 添加边缘应用			
1 应用配置			
* 应用名称	monitor-app		
*部署方式 ⑦	容器化部署 安装包部署		
	软件准备操作说明		
	 根据您的业务需要开发边缘应用程序(2) 制作镜像: 并上传到 容器镜像服务(SWR) 中、制作镜像语参考 Docker 宣网 		
* 切能用途	数据处理 协议解析 本地子系统集成		
应用描述	可选填,只允许输入中文,英文字母,中划线,下划		
	20, IIRVX:10/mtg 5 0/255		
版本配置 请你	尔配置该应用的版本信息, 以及后面步骤中的相关配置内容		
* 应用版本: 1	.0.0 □ 立即发布		
* 支持架构	v86.64 🚳		
* XI3*19	<u>x00_04</u>		
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

< 添加边线	〈 添加边缘应用				
① 应用配置 ——	2 软件和运行配置 (3) 端点和部署配置				
选择镜像输入方式:	点击选择 ▼				
* 选择镜像	edge_monitor 选择镜像 边缘应用程序需要制作为容器镜像,并统一在容器镜像服务(SWR)中管理。请您先准备好镜像文件,然后在这里添加边缘应用				
* 版本	1.0.0				
容器规格					
CPU配额	□ 申请预留				
	□ 限制使用				
内存配额	申请预留				
	□ 限制使用				
AI加速卡配额	不申请配额 申请GPU配额 申请NPU配额				

3. 添加边缘应用-端点和部署配置

添加边缘	
☆用配置 ────	— ② 软件和运行配置 ——— 3 第点和部署配置
端点配置	
例如, EdgeHub	使用MQTT Broker傲消息总线,输入蔬点代表模块接收消息的Topic,输出蔬点代表发送消息的Topic(蔬点仅代表而并非真正的MQTT Topic) 。
输入端点 🕜	请输入端点 添加满点
	input 💿
輸出端点 (?)	靖軍人第点
	output 🔘
部署配置	
重启策略:	总是更合 失败时重启 不重启
	コロカ大切と山口,んと定止市区山之定开市区山,お坊即左主州加起山内大切。
网络类型	主机网络 辨口缺时
	使用摘主机(边缘节点)的网络,即容器与主机间不做网络隔离,使用同一个IP。

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<mark>部署应用</mark>。

⚠ 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

久 2 数据流转配置 部署应用后,您可以根据需要配	显数据流转的来源和目标,灵活控制数据转发路径,并提高数据	安全性。	
规则名称	消息来源	消息目标	操作
monitor_input	· 设备请 •	user_monitor-app v Input v	到除
monitor_output	user_monitor-app 🔻 output 👻		删除
 添加規則 			
保存 取消			

🛄 说明

提示

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:

 \times

接入设备需统- 例/资源空间。	—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
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 • C
	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

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

Edit Connection Profiles			– 🗆 X
M2M Eclipse			
local mosquitto	Profile Name	myapp device	
myapp device	Profile Type	MQTT Broker	MQT
	MQTT Broker Profile Settings		
	Broker Address	128.70.00.6	
	Broker Port	7883	
	Client ID	myapp_device_0_0_2021071401	Generate
	General User Credentials	SSL/TLS Proxy LWT	\searrow
	Connection Timeout	30	
	Keep Alive Interval	60	
	Clean Session	\checkmark	
	Auto Reconnect		
	Max Inflight	10	
	MQTT Version	✓ Use Default	
		3.1.1	
		Clear Publish History	
		Clear Subscription History	
4.2	Dente		
	Kevert		Cancer OK Apply

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(loT边缘)后生成的设备lD和设备密钥,生成连接信息 (ClientId、Username、Password)。

Edit Connection Profiles			– 🗆 ×
M2M Eclipse local mosquitto myapp device	Profile Name Profile Type	myapp device	
	MQTT Broker Profile Settings		
	Broker Address Broker Port Client ID	7883 myapp device 0 0 2021071401	Generate
	General User Credentials	SSL/TLS Proxy LWT	
	User Name Password	myapp_device	
			8
	Douart		Carrol OK Analy
			Cancer On Apply

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。

Edit Connection Profiles		— 🗆 X
M2M Eclipse local mosquitto	Profile Name myapp device	
ттуарр өемсе	Profile Type MQTT Broker	
	MQ11 Broker Profile Settings	
	Broker Port 7883	
	Client ID myapp_device_0_0_2021071401 Gene	rate
	General User Credentials SSL/TLS Proxy LWT	
	Enable SSL S V Protocol TLSv1.2	*
	CA signed server certificate CA certificate file	
	CA Certificate File C:\Users\Administrator\Downloads\plt-device-ca.pem	
	CA certificate keystore	
	Self signed certificates Self signed certificates in keystores	
+ -	Revert Cancel	OK Apply

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,loTDA也会显示在线。

4. 选择publish输入topic地址。

Topic: Soc/devices/myapp_device/sys/properties/report

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



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。



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。

最新上纲数据	查習历史数据 ⑦ 查看全部屬性 C
皆无政策	
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), 上报的数据。	仍然可以看到

最新上报数据
status
running
<power></power>
2021/07/29 01:33:42 GMT+08:00

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



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: [IN	F0] star	t execute p	inocess.	
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 lib	rtmp/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.	
2021-07-29103:37:14,245	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	
2021-07-29103:37:14,246	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.	
2021-07-29103:37:14,246	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.	
2021-07-29103:37:14,246	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!				
2021-07-29103:37:15.246	INFO	ec9839c0	edge app login(edge app.c:926) login.	

图 4-13 SDK 结束运行的日志



----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 00 02117-15-05,588	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!			
2021-08-02117-15:04,590	INFO	f696c840	edge_app_login(edge_app.c:926) login.

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。

[rootgserver-b25a6cea-b075-482f-bfb8-e7+f5449013d 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 mgtt subscribe(edge mgtt.c:156) mgtt: 0x7f7be800e7c0 subscribe topic: \$oc/modules/user my-c-sdk/sys/properties/set/+ gos: 1				
2021-08-02T17:15:35,537 INFO dffff700 edge mgtt subscribe(edge mgtt.c:156) mgtt: 0x7f7be800e7c0 subscribe topic: \$oc/modules/user my-c-sdk/sys/shadow/get/response/+ gos: 1				
2021-08-02T17:15:35,537 INFO dffff700 edge matt subscribe(edge matt.c:156) matt: 0x7f7be800e7c0 subscribe topic: /modules/user mv-c-sdk/customized/inputs/# gos: 1				
2821-08-02T17:15:35,537 INFO dffff700 edge base retry success notify(edge base retry.c:61) retry worker:edge matt connections notify success.				
2021-08-02117:16:07,154 INFO dffff700 edge base retry stop/edge base retry.c:79) retry worker edge mott connections exited not:0				
2821-08-03T11:29:36.718 DEBUG dffff700 edge matt message arrived callback(edge matt.c:280) matt recy message, topic:/modules/user my-c-sdk/messages/inputs/input. en:199.				
2821-88-83T11:29:36.718 TNEO dffff288 edge matt conns recy msg/edge matt connections.c:175 recy msg. tonic:				
2021-08-03T11-29-36 718 TNED dfff720 edge ann pery mag(edge ann c-603) set bandler for tonic: /modules/user my-c-sdk/messages/innuts/innut				
2021_08_03711:20:36_718 DEBUG dffff700 deg_approt_stage_prot_stage_prot_stage_prot_stage_product idits: 6866608564b500826576b50				
2011 09 02111:05:07.06 710 DDB/G dffff700 ddg_apage.an argenan argenan argenan dylag ddgaada daylag daylag daylag				
est hus message callback with input				
set ous wessage caliback with input make. Input				
start sein message to varpat tojt, "("status": "augnian") "sonuise id": "noven" "novent time": "201455051322437"]] "deuten id" "muone deuten" ("status": "seaschasticheousgesticheous") "sonuise id": "muone deuten" ("status": "seaschasticheousgesticheous") "sonuise id": "muone deutent id": "seaschasticheousgest				
budy is, { services .[propercies .[scatus . running], service_tu . power , evenc_time . 2021050617/35422]], device_tu myapp_device , product_tu . obeccoss40550020071002 , receive_time				
. 102/7013/0294j				
body len 15: 199				
product_1d 15: 06060c0a5r405900280/rcm02				
start processing device.				
2021-08-03111:29:36,718 INFO dffff/00 edge_app_send_bus_message(edge_app.c:1238) start to send bus message.				
2021-08-03T11:29:36,718 INFO d++++700 edge_mqtt_publish(edge_mqtt.c:194) mqtt: 0x7+7be800e7c0 publish topic: /modul_s/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"

```
#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ld(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;
}
14
* 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ld(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)
ł
  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:%, ", 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++)
              Ł
                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);
>products[i].serviceCapabilities[j].commands[k].responses[i].paras[m].min);
>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-
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: 诵道
  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); }

```
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);
3
* 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(){
/*
```

```
* 构造 {"device_statuses":[{
   *
           "device_id":"deviceId",
   *
          "status":"status"
   * }]}
   ر
/* *
  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(){
  1
   *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();
```

```
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.");
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
```

```
void send_check_daemon_token()
ł
  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);
```

```
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_shadow_cb = shadowCb;
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

#include "edge_driver.h"

该Demo主要展示edge_driver.h里的回调函数和接口函数使用。

```
* 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;
3
* 描述: 子设备刪除的回调
* 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++)
   ł
      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ld(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) ł 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; 3 * 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)
ł
  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-
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-
printf("max:%s, ", getProductsRspEvent-
>products[i].serviceCapabilities[j].commands[k].paras[l].max);
                printf("max_length:%d, ", getProductsRspEvent-
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].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
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};

3

```
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"
            "}";
```

```
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
   地方注释掉即可。
/**
* 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};
```

```
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<mark>集成ModuleSDK进行数据处理</mark>。

```
#include "edge.h"
#include <stdio.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)
ł
  // 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;
}
```

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
IConnectionStatusChange Callback	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。

选项		? ×
搜索选项(Ctrl+E)	٩	可用程序包源(P):
 ▷ Source Control ▷ 工作项 ▷ 文本编辑器 ▷ 调试 ▷ 性能工具 ▷ CMake ▷ F# Tools △ NuGet 包管理器 常规 程序包源 ▷ SQL Server 工具 ▷ VsVim ▷ Web ▷ Web ▷ Web 窗体设计器 		 nuget.org https://api.nuget.org/v3/index.json Microsoft Visual Studio Offline Packages C:\Program Files (x86)\Microsoft SDKs\NuGetPackages\ ModuleSDK source D:\ModuleSDK-C#-Nuget
 > Web 性能测试工具 > Windows 窗体设计器 > XAML 设计器 ▶ 测试 	~	各称(N): ModuleSDK source 源(S): D:\ModuleSDK-C#-Nuget 更新(U) 确定 取消

步骤3 为工程导入Nuget包

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。

NuGet: ModuleSDK-Demo 🕒 🗙	•
辺覧 已安装 更新	NuGet 包管理器: ModuleSDK-Demc
_{捜索(Ctrl+L)}	2 包括预发行版 程序包源: ModuleSDK source - @
iot-edge-sdk-cshar v	/1.0.0
Package Description	版本:最新稳定版 1.0.0 安装
	 ◇ 选项 描述 Package Description 版本: 1.0.0

▲ 注意

如果需要升级SDK,可参考<mark>步骤2</mark>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

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

i	配置管理器					?	×
	活动解决方案配置(C):		活动解决方案平台(P):				
	Release	~	Any CPU				\sim
	项目上下文(选定要生成或部署的项目的	记置)(<u>R</u>):					
	项目	配置	平台	生成	部署		
	ModuleSDK-Demo	Release ~	Any CPU v	\checkmark			
						关闭	

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				
🗋 Folder	Profile		*	发杯	节(U)	
新建配置文件			操作▼			
目标位置		bin\Release\net	coreapp	🗇	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	J				
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\netco	oreapp2.1\publisł	n\			
				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

- 🗟 iot-edge-sdk-csharp.dll
- ModuleSDK-Demo.deps.json
- 🚳 ModuleSDK-Demo.dll
- 周 ModuleSDK-Demo.pdb
- ModuleSDK-Demo.runtimeconfig.json
- 🚳 MQTTnet.dll
- MQTTnet.Extensions.ManagedClient.dll
- 🚳 Newtonsoft.Json.dll
- 🗟 NLog.dll

----结束

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<mark>编写高效的Dockerfile</mark>)。

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"]

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 <mark>官方文档</mark>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<mark>获取指令</mark>。

🛄 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

á

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 <mark>控制台</mark> 获取登录命令,如下	<u> 왕</u> 。
----------------------------------	-------------

熱斑 ⑦				1 上传論章 🗘 上传論章
(1) 试验您参加容器模像服务使用体验调研, 您里果的意见和建议	量我们伸供搞开产器体验的源助力,感谢您的参与;			
快速入门				操作指南点击这里
				10回台建组织 10回上传镜像 10回号作镜像压缩包 10回感如便权
1 创建组织	2 镜像获取	3 应用部署	4 更新限制	如何收藏總總
您可以根据自身组织的味ね。来 构建但您的附逐管理。 会建模块	您可以通过以下两种方式添加编 參 上後自有機像 使用機像面原	CCE使用场展 使用云 省器引擎邮幕	支持配置规则,自动更新强度应 用的機像 配置機象更新规则	

🛄 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1:

docker tag modulesdk-demo:1.0.0 swr.cn-north-4.myhuaweicloud.com/iotedge/modulesdkdemo:1.0.0

 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

镜像名称↓Ξ	所属组织	版本数 ↓Ξ	更新时间 ↓Ξ
modulesdk-demo	huawei-iot-edge	1	2022/09/24 17:26:37 GMT+08:00

7. 上传镜像后请在 SWR 将镜像设置为公开。

镜像名称↓Ξ	所属组织	版本数 ↓=	更新时间 ↓=
modulesdk-demo	huawei-iot-edge	1	2022/09/24 17:26:37 GMT+08:00

编辑镜像:

我的编像 / modules	dk-demo			上传输着	添加数2番	924E	2020
99.958	modulessk-demo	所属组织	Nuzirei-tol edge				
供型	公开	类别	其他				
版本数	1	下载次数	7				
已用空间	70.1 MB	创建时间	2022/08/24 17 03 24 GMT+08 00				

设置为公开:



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

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)。

<u>∕</u> 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<mark>添加应用</mark>。

🗀 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

软件部署配置一	② 运行配置 ③	3) 配置确认	
应用名称 modu	lesdk-demo-test		
* 部署方式	容器化部署 安装包部署 30.17.症筋球中防明 11.根据您的业务需要开发边缘应用程序 2.制作镜像,并上传到容器镜像服务(SW	/R) 中。制作镜像请参考 Docker 官网	
* 容器镜像	手动输入 ▼ swr.	r. yhuaweicloud.com/* dulesdk-der	
容器规格			
CPU配额			
内存配额	 限制使用 		
AI加速卡配额	不申请配额 申请GPU配	额 申请NPU配额	
高级配置			
→ 运行命	\$		
→ 选项	置 通过勾选项,容器将拥有访问主机设	备的权限,例如容器访问GPU,FPGA。	
~ 环境到	量 容器运行环境中设定的变量。可以在历	应用部署后修改,为应用提供极大的灵活性。设置容器运行环境中的系统环境变量。	
~ 数据有	储 支持挂载本地卷到容器中,以实现数	据文件的持久化存储	

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 <mark>容器镜像服务(SWR)</mark> ,且设 置其类型为"公开",例如 <mark>图</mark> 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 <mark>对象存</mark> <mark>储服务(OBS)</mark> 。
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 <mark>表</mark> 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 <mark>容器镜像服务(SWR)</mark> 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。

〈 丨 添加边缘	应用版本
① 软件部署配置 -	
应用名称 modu	lesdk-demo-test
端点配置 例如,EdgeHul 输入端点 ⑦ 输出端点 ⑦	b使用MQTT Broker做消息总线,输入端点代表模块接收消息的Topic,输出端点代表发送消息的Topic(端点仅代表而并非真正的MQTT Topic)。 清输入端点 添加端点 添加端点 添加端点 添加端点
部署配置	
重启策略:	总是重启 失败时重启 不重启
	当应用实例退出时,无论是正常退出还是异常退出,系统都会重新拉起应用实例。
网络类型	主机网络 端口映射
	使用有主机 (必家节点) 的网络,即答腊与王机即个颇网络隔离,使用同一个PP。

🛄 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
-------	------

用名称 mod	ulesdk-demo-te	est					
配置	软件和运行配置。						
	镜像地址	swr.cn-north-5.my	huaweicloud.	com/huawei-iot-e CPU配额	不申请预留 不限制使用	内存配额	不申请预留 不限制使用
	AI加速卡配额	未设置		运行命令:	未设置	特权配置	关闭,容器没有访问主机设备的权限
	环境变量:	0 个变量		数据存储:	0 个卷	外挂设备:	0
	健康检查:	应用存活 不配置 凡	如用业务 不配置				
	端点和部署配置 <u>。</u>						
	输入端点 0.	个端点	输出端点	0 个端点			
	重启策略 总	是重启	网络类型	主机网络			
★ SDK 版本	1.0.1			□ 支持多部署 ⑦			
* 版本 * 支持架构 * 数采模板	x86_64 《 ● 选择标准数	▶ ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	自定义数采模板	¥			
* 版本 * 支持架构 * 数采模板	x86_64 《 选择标准 opcua通用表 点击下载通用 	处采模板 ↓ 上传的 处采模板 ↓ 处采模板	自定义数采模板	•			
* 版本 * 支持架构 * 数采模板 * 厂商	x86_64 《 ● 选择标准部 のpcua通用表 点击下载通用部 test	效采模板 ↓ 上传目 文采模板 ◆ 文采模板	自定义数采模板	•			

🗀 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。

〈 添加边缘	应用版本	
1 软件部署配置 一		
应用名称 modu	esdk-demo-test	
* 部署方式	容器化部署 安装包部署 软件准备操作说明 1. 根据您购业务需要开发边缘应用程序 2. 将应用程序打包或安装包文件,并上传到 对象存储服务(OBS)	
* 安装包地址		
进程规格		
CPU配额	申请预留	
	限制使用	
内存配额	申请预留	
	限制使用	
AI加速卡配额	不申请配额 申请GPU配额 申请NPU配额	
高级配置		
→ 选项配	置 通过勾选项,容器将拥有访问主机设备的权限,例如容器访问GPU, FPGA。	9
~ 环境变	客器运行环境中设定的变量。可以在应用部署后修改,为应用提供极大的灵活性。设置容器运行环境中的系统环境变量。	E
~ 数据存	诸 支持挂载本地卷到容器中,以实现数据文件的持久化存储	©

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。

下一步

〈 丨 添加边缘	应用版本
① 软件部署配置 -	
应用名称 modu	lesdk-demo-test
端点配置 例如,EdgeHul 输入端点 ⑦ 输出端点 ⑦	b使用MQTT Broker做消息总线,输入端点代表模块接收消息的Topic,输出端点代表发送消息的Topic(端点仅代表而并非真正的MQTT Topic)。 清输入端点 添加端点 添加端点 添加端点 添加端点
部署配置	
重启策略:	总是重启 失败时重启 不重启
	当应用实例退出时,无论是正常退出还是异常退出,系统都会重新拉起应用实例。
网络类型	主机网络 端口映射
	使用有主机 (必家节点) 的网络,即答腊与王机即个颇网络隔离,使用同一个PP。

🗀 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
-------	-----

配置	软件和运行配置。					
	安装包地址	obs:/Demo/ModuleSDK-Demo.zip	CPU配额	不申请预留 不限制使用	内存配额	不申请预留 不限制使用
	AI加速卡配额	未设置	运行命令:	未设置	特权配置	关闭,容器没有访问主机设备的权限
	环境变量:	0 个变量	数据存储:	0 个卷	外挂设备:	0
	健康检查:	应用存活 不配置 应用业务 不配置				
	端点和部署配置之					
	输入端点 0.	个端点 输出端点 0个端点				
	重启策略 总	是重启 网络类型 主机网络				
★ SDK 版本	1.0.1					
			+++			
* 版本	1.0		支持多部署 ?			
* 版本 * 支持架构	1.0 x86_64 @		支持多部署 ⑦			
* 版本 * 支持架构	1.0 x86_64 《 ● 选择标准器	▶ 文架機板 ○ 上传自主义数采模板	支持多部署 ⑦			
* 版本 * 支持架构 * 数采模板	 1.0 x86_64 《 选择标准 opcua通用表) 文平機板 ↓ 上传自定义数采模板 2采模板 ▼	支持多部署 ⑦ ▼			
* 版本 * 支持架构 * 数采模板	 1.0 x86_64 《 选择标键 opcua通用表 点击下载通用器 	▶ 效采模板 上传自定义数采模板 ¥2条模板 ▼ 文采模板	文持多部署 ⑦ ▼			
* 版本 * 支持架构 * 数采模板 * 丁商	1.0 x86_64 《 • 选择标准器 opcua通用器 点击下载递用器 test	タ	文持多部署 ?			

🗀 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loT边缘,单击"管理控制台"进入loT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。

ひて_test / minute 功能用途:数据采集 创建时间: 2022/04/06 15:10:00 C			
版本列表 ② ⑦ Q	版本值息		
3 x86_64,arm32 0	3 ● 未设存 ⑧不变持多能器	发布	编辑 拷贝 删除
• 1111 ×86_64 0	部署方式: 留器化部署 支持装饰, x85_64.arm32 上次编辑时间: 2022/05/26 10.52:07 GMT+08:00		
1.0.0 ×86_54 0	容益配置 純点配置 部署配置 实例列表(0)		
	基本伯应	高级配置	
	服务部任 swt.ch-north-4.myhuaweicloud.com/id_edge_test/ol_test/v2	运攻配置 关闭,容器设有访问主机设备的权限 环境变量 主设置	
	************************************	影振存能: 1个響 查查	
	内存配版 不由遗预副 不限制使用	外狂设新: 未设置 健康检查 应用存活 不配置 应用业务 不配置	
	PAILAR THANK - 179809	运行命令: 未设置	

🗀 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)

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	Dict ion ary <str ing, obj ect ></str 	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	stri ng	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。

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 <str ing ></str 	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	Dict ion ary <str ing, obj ect ></str 	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。

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	Dict ion ary <str ing, obj ect ></str 	按<影子属性名,影子属性对象>存储 的字典
PropertiesUpdateTime	Dict ion ary <str ing, lon g></str 	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(ModuleSh adow 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></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 <d evic eSt atu s></d 	存储设备状态信息的列表

表 4-101 DevicesStatus 结构说明

名称	类 型	描述
DeviceId	stri ng	设备ID
Status	stri ng	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	Dict ion ary <str ing, obj ect ></str 	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。

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	stri ng	连接状态
Info	stri ng	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<mark>创建工程</mark>的步骤创建新的工程,也可以下载Demo,解压并导入示例代码。

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
  ł
    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)
       ł
         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)
  ł
    // 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);
  ł
  /*
  * 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);
  }
  1
  * 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; }
}
```

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<mark>注册边缘节点</mark>。

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。
- 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"。

 \times



3. 在"添加驱动应用"弹窗中填写相关信息,然后单击"确认"。

★ 驱动 ID	modulesdk-demo	
* 驱动名称	modulesdk-demo	
* 协议类型	OPCUA 🔻	
★ 功能用途	数据采集 混合应用	
应用描述	可选填	
		0/255
	确认取消	

4. 前往应用版本配置界面

添加驱动应用

< 添加边缘	应用版本			
1 软件部署配置 -	2 运行配置	3 配置	角认	
应用名称 modu	lesdk-demo-test			
* 部署方式	容器化部署	安装包部署		
	软件准备操作说明 1. 根据您的业务需要开 2. 将应用程序打包成多	F发边缘应用程序 安装包文件,并上传到 对	象存储服务(OBS)	
* 安装包地址	obs://ic	?:/ModuleSDK-Dem	no.zip	
进程规格				
CPU配额	申请预留			
	限制使用			
内存配额	申请预留			
	限制使用			
AI加速卡配额	不申请配额	申请GPU配额	申请NPU配额	

5. 添加边缘应用-端点和部署配置

< 添加边缘/	立用版本	
① 软件部署配置 —	2 运行配置 —— ③ 配置确认	
应用名称 module	esdk-demo-test	
端点配置 例如,EdgeHub 输入端点 ?	使用MQTT Broker做消息总线,输入端点代表模块接收消息的 请输入端点)Topic,輸出端点代表发送洋 添加端点
输出端点 ⑦	请输入端点	添加端点
部署配置		
重启策略:	总是重启 失败时重启 不重启	
	当应用实例退出时,无论是正常退出还是异常退出,系统都会	会重新拉起应用实例。
网络类型	主机网络端口映射	
	使用宿主机 (边缘节点) 的网络, 即容器与主机间不做网络网	閹离,使用同一个IP。

🛄 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"

① 软件部署配置 ·	a ——— ② 运行配置 ———— ③ 配置输 从						
应用名称 mod	odulesdk-demo-test						
配置	软件和运行配置 <i>之</i>						
	安装包地址 obs://iotedge-test-0424k/ModuleSDK-Demo.zip CPU直接	不申请预留 不限制使用	内存配额	不申请预留 不限制使用			
	Al加速卡配额 未设置 运行命令	未设置	特权配置	关闭,容器没有访问主机设备的权限			
	环境变量: 0 个变量 数据存储	0 个卷	外挂设备:	0			
	健康检查: 应用存活 不配置 应用业务 不配置						
	蟻 互和部署程置 <u>/</u>						
	输入端点 0个端点 输出端点 0个端点						
	重启策略 总是重启 网络类型 主机网络						
★ SDK 版本	5 1.0.1						
* 版本	1.0 支持多部署 ⑦						
* 支持架构	x86_64 🔘 👻						
* 数采模板	* 数采模板 💿 选择标准数采模板 🔷 上传自定义数采模板						
	opcua通用数采模板 🔻						
	二山下城通用数米機板						
* 厂商	Huawei						

4.5.4.8 部署应用

部署应用,具体参考<mark>部署应用</mark>,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loT边缘,单击"管理控制台"进入loT边缘控制台。
- 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。

17099								
建数采连接 批量删除								
连接通道名称	连接状态 ⑦	数量	驱动模块	最近修改时间	启用状态	操作		
▲ fsgsdfgsdfgs	未连接	1	user_modulesdk-demo	2022/09/24 17:07:15 GMT+0	$\checkmark \bigcirc$	下发配置 编	闺 删除	
(1) 请确保已注册边缘设备,前往边 点位映射列表 导入点位	隊设备							
添加点位 导出点位	批量删除		默认按照点位标识搜索					Q
点位标识	点位名称	设备 ID	设备属性	更新时间		采集值	操作	

 \times

采集值详情 ⑦

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