设备接入

SDK 参考

文档版本	1.0
发布日期	2025-09-09

华为云计算技术有限公司

商标声明


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

注意

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

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

目 录

1 SDK 概述

物联网平台提供应用侧SDK和设备侧SDK，方便设备通过集成SDK接入到平台，应用通过调用物联网平台的API，实现安全接入、设备管理、数据采集、命令下发等业务场景。

说明

如果无法正常打开GitHub仓库，请检查您所使用的网络是否可以正常访问公网。

资源包名	描述	下载路径
应用侧 Java SDK	应用侧 Java SDK提供Java方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧Java SDK使用指南**。	**应用侧 Java SDK**
应用侧 .NET SDK	应用侧 .NET SDK提供.NET方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧.NET SDK使用指南**。	**应用侧 .NET SDK**
应用侧 Python SDK	应用侧 Python SDK提供Python方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧Python SDK使用指南**。	**应用侧 Python SDK**
应用侧 Go SDK	应用侧 Go SDK提供Go方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧Go SDK使用指南**。	**应用侧 Go SDK**
应用侧 Node.js SDK	应用侧 Node.js SDK提供Node.js方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧Node.js SDK使用指南**。	**应用侧 Node.js SDK**
应用侧 PHP SDK	应用侧 PHP SDK提供PHP方法调用**应用侧API**与平台通信。使用指南可以参考**应用侧PHP SDK使用指南**。	**应用侧 PHP SDK**

资源包名	描述	下载路径
设备侧 IoT Device Java SDK	设备可以通过集成设备侧 IoT Device Java SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **IoT Device SDK使用指南（Java）**。	**设备侧 IoT Device Java SDK**
设备侧 IoT Device C for Linux/ Windows SDK	设备可以通过集成设备侧 IoT Device C for Linux/Windows SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device C for Linux/ Windows SDK使用指南**。	**设备侧 IoT Device C for Linux/Windows SDK**
设备侧 IoT Device C# SDK	设备可以通过集成设备侧 IoT Device C# SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device C# SDK使用指南**。	**设备侧 IoT Device C# SDK**
设备侧 IoT Device Android SDK	设备可以通过集成设备侧 IoT Device Android SDK 接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device Android SDK使用指南**。	**设备侧 IoT Device Android SDK**
设备侧 IoT Device Go SDK(社区版)	设备可以通过集成设备侧 IoT Device Go SDK(社区版)接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device Go SDK(社区版)使用指南**。	**设备侧 IoT Device Go SDK(社区版)**
设备侧 IoT Device C for Linux/ Windows SDK Tiny	设备可以通过集成设备侧 IoT Device C for Linux/Windows SDK Tiny接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device C for Linux/ Windows SDK Tiny使用指南**。	**设备侧 IoT Device C for Linux/Windows SDK Tiny**

资源包名	描述	下载路径
设备侧 IoT Device Arkts(OpenHarmony) SDK	设备可以通过集成设备侧 IoT Device Arkts(OpenHarmony) SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device Arkts(OpenHarmony) SDK使用指南**。	**设备侧 IoT Device Arkts(OpenHarmony) SDK**
设备侧 IoT Device Python SDK	设备可以通过集成设备侧 IoT Device Python SDK接入物联网平台, Demo提供了调用SDK接口的样例代码。使用指导请参考 **设备侧 IoT Device Python SDK使用指南**。	**设备侧 IoT Device Python SDK**

2 应用侧 SDK

2.1 应用侧 Java SDK 使用指南

物联网平台提供Java语言的应用侧SDK供开发者使用。本文介绍应用侧 Java SDK的安装和配置，及使用Java SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装Java开发环境。

访问**Java官网**，下载并说明安装Java开发环境。

说明

华为云Java SDK支持Java JDK 1.8 及其以上版本。

步骤2 安装Maven软件

通过 Maven 安装项目依赖是使用 Java SDK 的推荐方法，首先您需要**下载**并**安装Maven** ，安装完成后您只需在 Java 项目的 pom.xml 文件加入相应的依赖项即可。

步骤3 安装Java SDK

添加Maven依赖：

```
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-core</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-iotda</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

----结束

代码示例

> **⚠ 注意**
>
> Maven依赖版本请使用版本区间，如您使用具体版本号，请使用3.0.60及以上。

以调用**查询设备列表**接口为例，以下代码示例向您展示使用应用侧 Java SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
package com.huaweicloud.sdk.test;

import com.huaweicloud.sdk.core.auth.ICredential;
import com.huaweicloud.sdk.core.exception.ConnectionException;
import com.huaweicloud.sdk.core.exception.RequestTimeoutException;
import com.huaweicloud.sdk.core.exception.ServiceResponseException;
import com.huaweicloud.sdk.core.region.Region;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.iotda.v5.*;
import com.huaweicloud.sdk.iotda.v5.model.*;

public class ListDevicesSolution {

    // REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-1"
    private static final String REGION_ID = "<YOUR REGION ID>";
    // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
    private static final String ENDPOINT = "<YOUR ENDPOINT>";

    public static void main(String[] args) {
        // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
        String ak = System.getenv("HUAWEICLOUD_SDK_AK");
        String sk = System.getenv("HUAWEICLOUD_SDK_SK");
        String projectId = "<YOUR PROJECTID>";

        // 创建认证
        ICredential auth = new BasicCredentials()
                .withAk(ak)
                .withSk(sk)
                // 标准版/企业版需要使用衍生算法，基础版请删除配置"withDerivedPredicate"
                .withDerivedPredicate(BasicCredentials.DEFAULT_DERIVED_PREDICATE)
                .withProjectId(projectId);

        // 创建IoTDAClient实例并初始化
        IoTDAClient client = IoTDAClient.newBuilder()
                .withCredential(auth)
                // 标准版/企业版：需自行创建Region对象，基础版：请使用IoTDARegion的region对象，如"withRegion(IoTDARegion.CN_NORTH_4)"
                .withRegion(new Region(REGION_ID, ENDPOINT))
                // .withRegion(IoTDARegion.CN_NORTH_4)
                // 配置是否忽略SSL证书校验， 默认不忽略
                // .withHttpConfig(new HttpConfig().withIgnoreSSLVerification(true))
                .build();

        // 实例化请求对象
```

```
        ListDevicesRequest request = new ListDevicesRequest();
        try {
            // 调用查询设备列表接口
            ListDevicesResponse response = client.listDevices(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.printStackTrace();
        } catch (RequestTimeoutException e) {
            e.printStackTrace();
        } catch (ServiceResponseException e) {
            e.printStackTrace();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

----结束

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
projectId	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.java**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。
REGION_ID	如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"
ENDPOINT	请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。

说明

项目源码及更多详细的使用指导请参考**华为云Java软件开发工具包（Java SDK）**。

2.2 应用侧 Python SDK 使用指南

物联网平台提供Python语言的应用侧SDK供开发者使用。本文介绍应用侧 Python SDK的安装和配置，及使用Python SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装Python开发环境。

访问**Python官网**，下载并按说明安装Python开发环境。

说明

华为云应用侧 Python SDK 支持 **Python3** 及以上版本。

步骤2 安装pip工具

访问**pip官网**，下载并按说明安装pip工具。

步骤3 安装Python SDK

执行如下命令安装华为云Python SDK核心库以及相关服务库

```
# 安装核心库
pip install huaweicloudsdkcore

# 安装IoTDA服务库
pip install huaweicloudsdkiotda
```

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用Python SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
import os

from huaweicloudsdkcore.exceptions import exceptions
from huaweicloudsdkcore.region.region import Region
from huaweicloudsdkiotda.v5 import *
from huaweicloudsdkcore.auth.credentials import BasicCredentials
from huaweicloudsdkcore.auth.credentials import DerivedCredentials

if __name__ == "__main__":
    # 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    #本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak = os.environ["HUAWEICLOUD_SDK_AK"]
    sk = os.environ["HUAWEICLOUD_SDK_SK"]
    project_id = "<YOUR PROJECTID>"
    # region_id：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4"；如果是华南广州，请填写"cn-south-1"
    region_id = "<YOUR REGION ID>"
    # endpoint：请在控制台的"总览"界面的"平台接入地址"中查看"应用侧"的https接入地址
    endpoint = "<YOUR ENDPOINT>"

    # 标准版/企业版：需自行创建Region对象
    REGION = Region(region_id, endpoint)

    # 创建认证
    # 创建BasicCredentials实例并初始化
    credentials = BasicCredentials(ak, sk, project_id)

    # 标准版/企业版需要使用衍生算法，基础版请删除配置"with_derived_predicate"
    credentials.with_derived_predicate(DerivedCredentials.get_default_derived_predicate())

    # 基础版：请选择IoTDAClient中的Region对象 如： .with_region(IoTDARegion.CN_NORTH_4)
    # 标准版/企业版：需要使用自行创建的Region对象
    # 配置是否忽略SSL证书校验， 默认不忽略： .with_http_config(HttpConfig(ignore_ssl_verification=True)) \
```

```
client = IoTDAClient.new_builder() \
    .with_credentials(credentials) \
    .with_region(REGION) \
    .build()

try:
    # 实例化请求对象
    request = ListDevicesRequest()
    # 调用查询设备列表接口
    response = client.list_devices(request)
    print(response)
except exceptions.ClientRequestException as e:
    print(e.status_code)
    print(e.request_id)
    print(e.error_code)
    print(e.error_msg)
```

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
project_id	项目ID。获取方法请参见 **获取项目ID**。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**iotda_region.py**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。
region_id	如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4"；如果是华南广州，请填写"cn-south-4"
endpoint	请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。

----结束

更多

项目源码及更多详细的使用指导请参考**华为云开发者 Python 软件开发工具包（Python SDK）**。

2.3 应用侧.NET SDK 使用指南

物联网平台提供C#语言的应用侧SDK供开发者使用。本文介绍应用侧 .NET SDK的安装和配置，及使用.NET SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装.NET开发环境。

访问**.NET官网**，下载并按说明安装.NET开发环境。

说明

华为云.NET SDK适用于：

- **.NET Framework 4.5** 及其以上版本。
- **.NET Standard 2.0** 及其以上版本。
- **C# 4.0** 及其以上版本。

步骤2 使用 .NET CLI 工具安装SDK

```
dotnet add package HuaweiCloud.SDK.Core
dotnet add package HuaweiCloud.SDK.IoTDA
```

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用.NET SDK的主要步骤：

步骤1 创建认证。

步骤2 创建BasicCredentials实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
using System;
using System.Collections.Generic;
using HuaweiCloud.SDK.Core;
using HuaweiCloud.SDK.Core.Auth;
using HuaweiCloud.SDK.IoTDA;
using HuaweiCloud.SDK.IoTDA.V5;
using HuaweiCloud.SDK.IoTDA.V5.Model;

namespace ListDevicesSolution
{
    class Program
    {
        static void Main(string[] args)
        {
            var listDevicesRequest = ListDevices();
            var res = JsonUtils.Serialize(listDevicesRequest.Result);
            Console.WriteLine(res);
        }

        private static async Task<ListDevicesResponse> ListDevices()
        {
            // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
            // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
            var ak = Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_AK", EnvironmentVariableTarget.Machine);
            var sk = Environment.GetEnvironmentVariable("HUAWEICLOUD_SDK_SK", EnvironmentVariableTarget.Machine);
            const string projectId = "<YOUR PROJECTID>";
            // region_id：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4"；如果是华南广州，请填写"cn-south-1"
            const string regionId = "<YOUR REGION ID>";
            // endpoint：请在控制台的"总览"界面的"平台接入地址"中查看"应用侧"的https接入地址
            const string endpoint = "<YOUR ENDPOINT>";

            // 创建认证
            var auth = new BasicCredentials(ak, sk, projectId);

            // 标准版/企业版需要使用衍生算法，基础版请删除该配置
```

```
            auth.WithDerivedPredicate(BasicCredentials.DefaultDerivedPredicate);

            var config = HttpConfig.GetDefaultConfig();
            config.IgnoreSslVerification = false;
            // 创建IoTDAClient实例并初始化
            var client = IoTDAAsyncClient.NewBuilder()
                  .WithCredential(auth)
                  .WithHttpConfig(config)
                  // 标准版/企业版需要自行创建region
                  .WithRegion(new Region(regionId, endpoint))
                  // 基础版使用默认 IoTDARegion中的region对象
                  // .WithRegion(IoTDARegion.CN_NORTH_4)
                  // net framework不支持在get请求头中有content-type
                // 配置是否忽略SSL证书校验， 默认不忽略
                  // .WithHttpConfig(new
HttpConfig().WithIgnoreBodyForGetRequest(true).WithIgnoreSslVerification(true))
                  .Build();

            // 实例化请求对象
            var req = new ListDevicesRequest
            {
            };

            try
            {
                // 调用查询设备列表接口
                var resp =await client.ListDevicesAsync(req);
                Console.WriteLine(resp.GetHttpStatusCode());
                return resp;
            }
            catch (RequestTimeoutException requestTimeoutException)
            {
                Console.WriteLine(requestTimeoutException.ErrorMessage);
            }
            catch (ServiceResponseException clientRequestException)
            {
                Console.WriteLine(clientRequestException.HttpStatusCode);
                Console.WriteLine(clientRequestException.ErrorCode);
                Console.WriteLine(clientRequestException.ErrorMsg);
            }
            catch (ConnectionException connectionException)
            {
                Console.WriteLine(connectionException.ErrorMessage);
            }
            return new ListDevicesResponse();
        }
    }
}
```

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**IoTDARegion.cs**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。

----结束

更多

项目源码及更多详细的使用指导请参考**华为云 .Net 软件开发工具包（.NET SDK）**。

2.4 应用侧 Go SDK 使用指南

物联网平台提供Go语言的应用侧SDK供开发者使用。本文介绍应用侧 Go SDK的安装和配置，及使用Go SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装Go开发环境。

访问**Go官网**，下载并按说明安装Go开发环境。

说明

华为云 Go SDK 支持 **Go 1.14** 及以上版本。

步骤2 安装华为云Go库

```
go get github.com/huaweicloud/huaweicloud-sdk-go-v3
```

步骤3 安装依赖

```
go get github.com/json-iterator/go
```

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用Go SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
package main

import (
    "fmt"
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/core/auth"
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/core/auth/basic"
    // 标准版/企业版请使用    "github.com/huaweicloud/huaweicloud-sdk-go-v3/core/region"
    // 基础版请使用"github.com/huaweicloud/huaweicloud-sdk-go-v3/services/iotda/v5/region"
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/core/region"
    //"github.com/huaweicloud/huaweicloud-sdk-go-v3/services/iotda/v5/region"
    iotda "github.com/huaweicloud/huaweicloud-sdk-go-v3/services/iotda/v5"
    "github.com/huaweicloud/huaweicloud-sdk-go-v3/services/iotda/v5/model"
)

func main() {
        // 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
        // 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    ak := os.Getenv("HUAWEICLOUD_SDK_AK")
    sk := os.Getenv("HUAWEICLOUD_SDK_SK")
    projectId := "<YOUR PROJECTID>"
```

```
    // regionID和endpoint用于标准版企业版自行创建region，基础版可删除
    // region_id：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4"；如果是华南广州，请填写"cn-south-1"
    regionId := "<YOUR REGION ID>"
    // endpoint：请在控制台的"总览"界面的"平台接入地址"中查看"应用侧"的https接入地址
    endpoint := "<YOUR ENDPOINT>"

    // 创建认证
    auth := basic.NewCredentialsBuilder().
        WithAk(ak).
        WithSk(sk).
        WithProjectId(projectId).
        // 企业版/标准版需要使用衍生算法，基础版请删除该配置"WithDerivedPredicate"
        WithDerivedPredicate(auth.GetDefaultDerivedPredicate()).
        Build()

    // 创建IoTDAClient实例并初始化
    client := iotda.NewIoTDAClient(
        iotda.IoTDAClientBuilder().
            // 标准版/企业版需要自行创建region，基础版使用IoTDARegion中的region对象
            WithRegion(region.NewRegion(regionId, endpoint)).
            // WithRegion(region.CN_NORTH_4).
            WithCredential(auth).
                    // 配置是否忽略SSL证书校验， 默认不忽略
                    // WithHttpConfig(config.DefaultHttpConfig().WithIgnoreSSLVerification(true)).
            Build())
    // 实例化请求对象
    request := &model.ListDevicesRequest{}
    // 调用查询设备列表接口
    response, err := client.ListDevices(request)
    if err == nil {
        fmt.Printf("%+v\n", response)
    } else {
        fmt.Println(err)
    }
}
```

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
IoTDARegion.CN_NORTH_4	请替换为您要访问的物联网平台的区域，当前物联网平台可以访问的区域，在SDK代码**region.go**中已经定义。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。

----结束

更多

项目源码及更多详细的使用指导请参考**华为云开发者 Go 软件开发工具包（Go SDK）**。

2.5 应用侧 Node.js SDK 使用指南

物联网平台提供Node.js语言的应用侧SDK供开发者使用。本文介绍应用侧 Node.js SDK的安装和配置，及使用Node.js SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装Node.js开发环境。

访问**Node.js官网**，下载并按说明安装Node.js开发环境。

说明

华为云 Node.js SDK 支持 **Node 10.16.1** 及以上版本。

步骤2 安装依赖

```
npm install @huaweicloud/huaweicloud-sdk-core
npm install @huaweicloud/huaweicloud-sdk-iotda
```

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用Node.js SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
const core = require('@huaweicloud/huaweicloud-sdk-core');
const iotda = require("@huaweicloud/huaweicloud-sdk-iotda");
// 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
// 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
const ak = process.env.HUAWEICLOUD_SDK_AK;
const sk = process.env.HUAWEICLOUD_SDK_SK;
// endpoint：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
// const endpoint = "https://iotda.cn-north-4.myhuaweicloud.com";
const endpoint = "<YOUR ENDPOINT>";
const project_id = "<YOUR PROJECT_ID>";
// region_id: 如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-1"
const region_id = "<YOUR REGION_ID>";
// 配置跳过服务端证书验证（可选）
process.env.NODE_TLS_REJECT_UNAUTHORIZED = "0"
// 创建认证
const credentials = new core.BasicCredentials()
                    .withAk(ak)
                    .withSk(sk)
                    // 标准版/企业版需要使用衍生算法，基础版请删除配置"withDerivedPredicate"
                    .withDerivedPredicate(core.BasicCredentials.getDefaultDerivedPredicate)
                    .withProjectId(project_id)
// 创建IoTDAClient实例并初始化
const client = iotda.IoTDAClient.newBuilder()
                        .withCredential(credentials)
                        .withEndpoint(endpoint)
                        .withRegion(new core.Region(region_id, endpoint))
```

```
                        .build();
// 实例化请求对象
const request = new iotda.ListDevicesRequest();
// 调用查询设备列表接口
const result = client.listDevices(request);
result.then(result => {
    console.log("JSON.stringify(result)::" + JSON.stringify(result));
}).catch(ex => {
    console.log("exception:" + JSON.stringify(ex));
});
```

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
endpoint	请替换为您要访问的华为云服务所在区域的终端节点。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤请参考**地区和终端节点**。
project_id	您要访问的华为云服务所在项目 ID ，根据您想操作的项目所属区域选择对应的项目 ID
region_id	如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-4"

----结束

更多

项目源码及更多详细的使用指导请参考**华为云开发者 Node.js 软件开发工具包（Node.js SDK）**。

2.6 应用侧 PHP SDK 使用指南

物联网平台提供PHP语言的应用侧SDK供开发者使用。本文介绍应用侧 PHP SDK的安装和配置，及使用PHP SDK调用**应用侧API**的示例。

SDK 获取和安装

步骤1 安装PHP开发环境。

访问**PHP官网**，下载并按说明安装PHP开发环境。

说明

华为云PHP SDK支持PHP 5.6、PHP6、PHP7、PHP8版本，在运行前可执行 php --version 检查当前 Php 的版本信息，如果安装了PHP其他语言的版本，运行PHP SDK可能会报错。

步骤2 安装composer

```
curl -sS https://getcomposer.org/installer | php
```

步骤3 安装PHP SDK

```
composer require huaweicloud/huaweicloud-sdk-php
```

步骤4 引入 Composer 的自动加载文件

```
require 'path/to/vendor/autoload.php';
```

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用PHP SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
<?php
namespace HuaweiCloud\SDK\IoTDA\V5\Model;
require_once "vendor/autoload.php";
use HuaweiCloud\SDK\Core\Auth\BasicCredentials;
use HuaweiCloud\SDK\Core\Http\HttpConfig;
use HuaweiCloud\SDK\Core\Auth\Credentials;
use HuaweiCloud\SDK\Core\Exceptions\ConnectionException;
use HuaweiCloud\SDK\Core\Exceptions\RequestTimeoutException;
use HuaweiCloud\SDK\Core\Exceptions\ServiceResponseException;
use HuaweiCloud\SDK\Core\Region\Region;
use HuaweiCloud\SDK\IoTDA\V5\IoTDAClient;
// 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
// 本示例以ak和sk保存在环境变量中为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
$ak = getenv('HUAWEICLOUD_SDK_AK');
$sk = getenv('HUAWEICLOUD_SDK_SK');
// endpoint：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
// $endpoint = "https://iotda.cn-north-4.myhuaweicloud.com";
$endpoint = "<YOUR ENDPOINT>";
$projectId = "<YOUR PROJECT_ID>";
// REGION_ID：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-1"
$regionId = "<YOUR REGION ID>";
// 创建认证
$credential = new BasicCredentials($ak,$sk,$projectId);
// 标准版/企业版需要使用衍生算法，基础版请删除配置"withDerivedPredicate"
$credential->withDerivedPredicate(Credentials::getDefaultDerivedPredicate());
// 修改默认配置，跳过服务端证书验证
$config = HttpConfig::getDefaultConfig();
$config->setIgnoreSslVerification(true);
// 创建IoTDAClient实例并初始化（若默认配置无修改config可不添加）
$client = IoTDAClient::newBuilder(new IoTDAClient)
  ->withHttpConfig($config)
  ->withEndpoint($endpoint)
  ->withCredentials($credential)
  ->withRegion(new Region($regionId, $endpoint))
  ->build();
// 实例化请求对象
$request = new ListDevicesRequest();
try {
  // 调用查询设备列表接口
  $response = $client->ListDevices($request);
} catch (ConnectionException $e) {
  $msg = $e->getMessage();
  echo "\n". $msg ."\n";
} catch (RequestTimeoutException $e) {
  $msg = $e->getMessage();
  echo "\n". $msg ."\n";
```

```
} catch (ServiceResponseException $e) {
  echo "\n";
  echo $e->getHttpStatusCode(). "\n";
  echo $e->getErrorCode() . "\n";
  echo $e->getErrorMsg() . "\n";
}
echo "\n";
echo $response;
```

参数	说明
ak	您的华为云账号访问密钥ID（Access Key ID）。请在华为云控制台“我的凭证 > 访问密钥”页面上创建和查看您的 AK/SK。更多信息请查看**访问密钥**。
sk	您的华为云账号秘密访问密钥（Secret Access Key）。
endpoint	请替换为您要访问的华为云服务所在区域的终端节点。 您可以在控制台上查看当前服务所在区域名称，区域名称、区域和终端节点的对应关系，具体步骤**地区和终端节点**。
projectId	您要访问的华为云服务所在项目 ID ，根据您想操作的项目所属区域选择对应的项目 ID

----结束

更多

项目源码及更多详细的使用指导请参考**华为云开发者 PHP 软件开发工具包（PHP SDK）**。

2.7 应用侧 C++ SDK 使用指南

物联网平台提供CPP语言的应用侧SDK供开发者使用。本文介绍应用侧 C++SDK的安装和配置，以及使用C++ SDK调用**应用侧API**的示例。

在 Linux 系统上安装 SDK

步骤1 获取依赖包。

1. 所需的这些第三方软件包在大部分系统的包管理工具中都有提供，例如基于Debian/Ubuntu的系统。
   ```
   sudo apt-get install libcurl4-openssl-dev libboost-all-dev libssl-dev libcpprest-dev
   ```
2. spdlog需要从源码进行安装。
   ```
   git clone https://github.com/gabime/spdlog.git
   cd spdlog
   mkdir build
   cd build
   cmake -DCMAKE_POSITION_INDEPENDENT_CODE=ON ..  // 用以生成动态库 make sudo make install
   ```

步骤2 编译安装。

```
git clone https://github.com/huaweicloud/huaweicloud-sdk-cpp-v3.git
cd huaweicloud-sdk-cpp-v3
mkdir build
cd build
cmake .. make
sudo make install
```

完成上述操作后，C++ SDK安装目录为“ /usr/local”。

----结束

在 Windows 系统上安装 SDK

步骤1 安装vcpkg并使用vcpkg安装所需软件包。

```
vcpkg install curl cpprestsdk boost openssl spdlog
```

步骤2 使用CLion进行编译。

1. 使用CLion打开huaweicloud-sdk-cpp-v3目录。
2. 选择“File > Settings”。
3. 选择“Build > Execution > Deployment > CMake”。
4. 在CMake options中加入以下配置。
   ```
   -DCMAKE_TOOLCHAIN_FILE={your vcpkg install dir}/scripts/buildsystems/vcpkg.cmake
   ```
5. 右键CMakeLists.txt选择“Load CMake Project”。
6. 配置clion的编译工具链为MSVC: 在**步骤2.3**的CMake配置页面选择Toolchain为Visual Studio，不能选择mingw等其他编译器(windows平台下依赖msvc编译器，选择mingw等其他编译器编译会报错)。另外，用户还可以选择编译出来的二进制文件是Debug模式还是Release模式, 选择Build Type进行下拉选择即可。
7. 配置目标文件的架构和平台: windows平台支持编译不同CPU架构（x64, x86)的sdk链接库文件，用户可以根据实际需要进行配置，单击“Build > Execution, Deployment > Toolchains”，在Architecture选项可以下拉选择支持的CPU架构。
8. 选择“Build”开始编译，编译结果会在clion控制台进行打印。

步骤3 安装 C++ SDK。编译完成后选择“Build > Install”，完成后，C++ SDK安装目录为“C:\Program File (x86)\huaweicloud-sdk-cpp-v3”。

----结束

代码示例

以调用**查询设备列表**接口为例，以下代码示例向您展示使用C++ SDK的主要步骤：

步骤1 创建认证。

步骤2 创建IoTDAClient实例并初始化。

步骤3 实例化请求对象。

步骤4 调用查询设备列表接口。

```
#include <cstdio>
#include <iostream>
#include <huaweicloud/core/exception/Exceptions.h>
#include <huaweicloud/core/Client.h>
#include <huaweicloud/iotda/v5/IoTDAClient.h>

using namespace HuaweiCloud::Sdk;
using namespace HuaweiCloud::Sdk::Core;
using namespace HuaweiCloud::Sdk::Core::Utils;
using namespace HuaweiCloud::Sdk::Core::Exception;
using namespace HuaweiCloud::Sdk::Iotda::V5;
using namespace HuaweiCloud::Sdk::Iotda::V5::Model;
using namespace std;
int main(void)
```

```
{
    // region_id：如果是上海一，请填写"cn-east-3"；如果是北京四，请填写"cn-north-4";如果是华南广州，请填写"cn-south-1"
    std::string regionId = "<YOUR REGION ID>";
    std::string projectId = "<YOUR PROJECTID>";
    // ENDPOINT：请在控制台的"总览"界面的"平台接入地址"中查看“应用侧”的https接入地址。
    std::string endpoint = "<YOUR ENDPOINT>";
    std::string ak;
    std::string sk;
#if defined(WIN32) || defined(__WIN32__) || defined(_WIN32) || defined(_MSC_VER)
    ak = getenv("HUAWEICLOUD_SDK_AK");
    sk = getenv("HUAWEICLOUD_SDK_SK");
#elif defined(linux) || defined(__linux) || defined(__linux__)
    char* envVar;
    #define INIT_ENV_VAR(ID, NAME)                \
    do {                                          \
        if (envVar = secure_getenv(#NAME)) { \
            ID = std::string(envVar);         \
        }                                         \
    } while (0)
    INIT_ENV_VAR(ak, HUAWEICLOUD_SDK_AK);
    INIT_ENV_VAR(sk, HUAWEICLOUD_SDK_SK);
    #undef INIT_ENV_VAR
#endif

    auto basicCredentials = std::make_unique<BasicCredentials>();
    // 标准版企业版请使用衍生算法，基础版删除该代码
    basicCredentials->setDerivedPredicate(Constants::DEFAULT_DERIVED_PREDICATE);
    basicCredentials->withAk(ak)
            .withSk(sk)
            .withProjectId(projectId);
    auto client = HuaweiCloud::Sdk::Iotda::V5::IoTDAClient::newBuilder()
            .withCredentials(std::unique_ptr<Credentials>(basicCredentials.release()))
            .withRegion(Region(regionId, endpoint))
            .build();
    // Initialize request parameters
    HuaweiCloud::Sdk::Iotda::V5::Model::ListDevicesRequest request =
HuaweiCloud::Sdk::Iotda::V5::Model::ListDevicesRequest();
    try {
        std::string stringValue;
        std::cout << "***********ListDevice**********" << std::endl;
        // 调用查询设备列表接口
        std::shared_ptr<HuaweiCloud::Sdk::Iotda::V5::Model::ListDevicesResponse> listRes =
                client->listDevices(request);
        std::cout << listRes->getHttpBody() << std::endl;
    } catch (HostUnreachableException& e) { // handle exception
        std::cout << e.what() << std::endl;
    } catch (SslHandShakeException& e) {
        std::cout << e.what() << std::endl;
    } catch (RetryOutageException& e) {
        std::cout << e.what() << std::endl;
    } catch (CallTimeoutException& e) {
        std::cout << e.what() << std::endl;
    } catch (ServiceResponseException& e) {
        std::cout << "StatusCode: " << e.getStatusCode() << std::endl;
        std::cout << "ErrorCode: " << e.getErrorCode() << std::endl;
        std::cout << "ErrorMsg: " << e.getErrorMsg() << std::endl;
        std::cout << "RequestId: " << e.getRequestId() << std::endl;
    }
    return 0;
}
```

步骤5 如果您是在Linux系统中运行该代码，请复制上述文件到iotda_test.cpp。然后执行如下命令：

```
$ g++ -o iotda_test iotda_test.cpp --std=c++14 -liotda_v5 -lcore -lcrypto -lboost_system -lcpprest
$ ./iotda_test
# 下方会显示实际运行结果$
```

如果您是在Windows系统下使用cmake来管理工程，则需要在CMakeLists.txt 中引入sdk core包和服务包的相关依赖。可以参考下面的CMakeLists.txt 文件:

```
cmake_minimum_required(VERSION 3.16)
project(demo)
find_package(CURL REQUIRED)
set(CMAKE_CXX_STANDARD 14)

set(LINK_DIR "C:/Program Files (x86)/huaweicloud_cpp_sdk_v3/bin;")
set(BIN_DIR "C:/Program Files (x86)/huaweicloud_cpp_sdk_v3/lib;")
set(SERVICE_DIR "C:/Program Files (x86)/huaweicloud_cpp_sdk_v3/include;")
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DBOOST_UUID_FORCE_AUTO_LINK")

link_directories(${BIN_DIR})
include_directories(${SERVICE_DIR})
add_compile_options("$<$<C_COMPILER_ID:MSVC>:/utf-8>")
add_compile_options("$<$<CXX_COMPILER_ID:MSVC>:/utf-8>")

add_executable(demo main.cpp)
target_link_libraries(demo PUBLIC core iotda_v5)
```

----结束

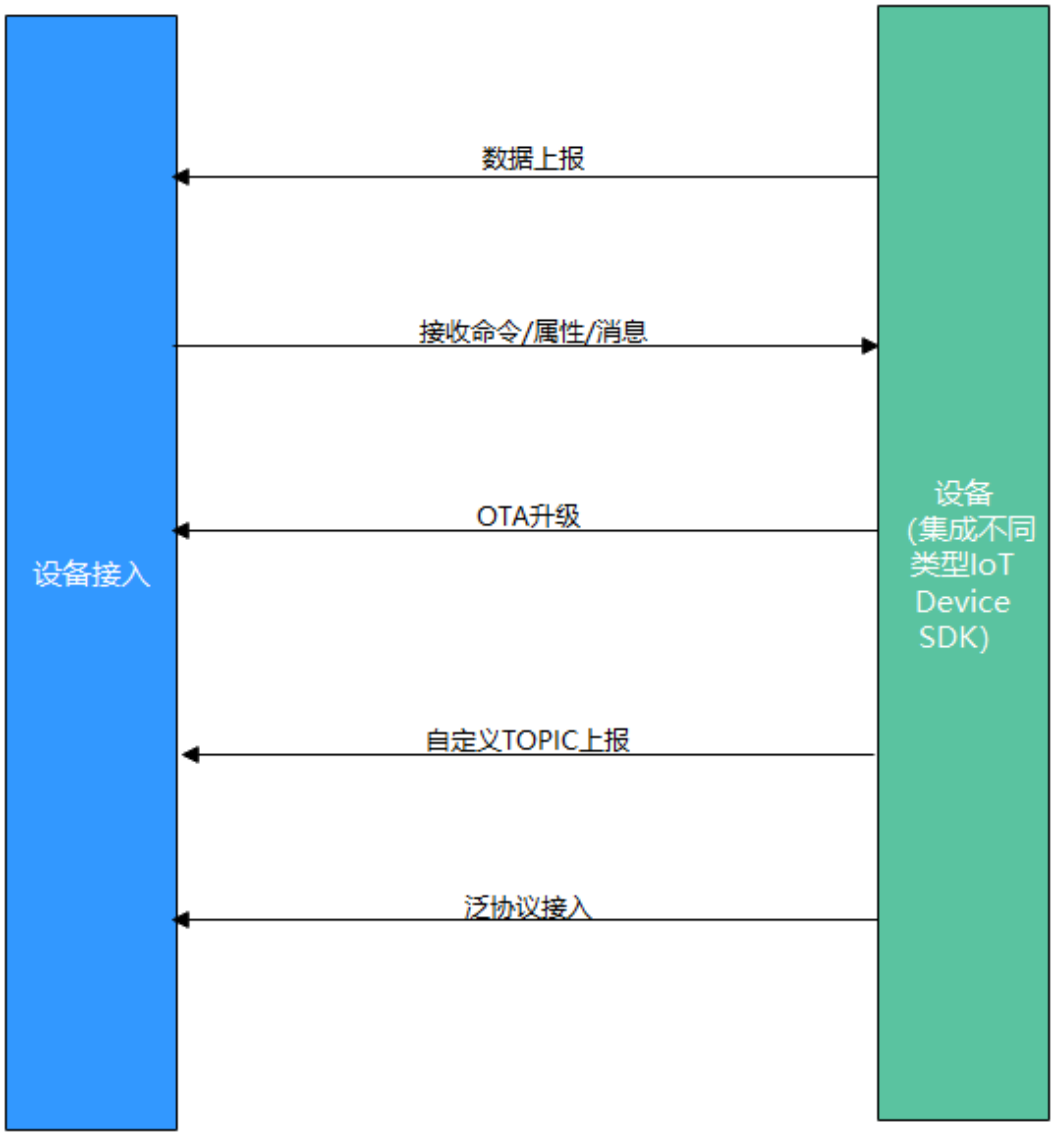
更多

项目源码及更多详细的使用指导请参考**华为云开发者 C++ 软件开发工具包（C++ SDK）**。

3 设备侧 SDK

3.1 设备侧 IoT Device SDK 介绍

为了帮助设备快速连接到物联网平台，华为提供了设备侧IoT Device SDK。支持TCP/IP协议栈的设备集成设备侧IoT Device SDK后，可以直接与物联网平台通信。不支持TCP/IP协议栈的设备，例如蓝牙设备、ZigBee设备等需要利用网关将设备数据转发给物联网平台，此时网关需要事先集成设备侧IoT Device SDK。



1. 设备接入前，需创建产品（可通过控制台创建或者调用应用侧API接口**创建产品**）。
2. 产品创建完毕后，需注册设备（可通过控制台注册单个设备或者使用应用侧API接口**注册设备**）。
3. 设备注册完毕后，按照图中流程实现消息/属性上报、接收命令/属性/消息、OTA升级、自定义TOPIC、泛协议接入（相关**Demo**）等功能。

平台提供了两种SDK，它们之间的区别如下表：

SDK种类	SDK集成场景	SDK支持的物联网通信协议
IoT Device SDK	面向运算、存储能力较强的嵌入式设备，例如网关、采集器等。	MQTT
IoT Device SDK Tiny	面向对功耗、存储、计算资源有苛刻限制的终端设备，例如单片机、模组。	LWM2M over CoAP、MQTT

对接入设备的硬件要求：

SDK名称	RAM容量	FLASH容量	CPU频率	操作系统类型	开发语言
IoT Device SDK	> 4MB	> 2MB	> 200MHZ	C版（Linux）、Java版（Linux/Windows）、C#版（Windows）、Android版（Android）、Go社区版（Linux/Windows/类unix）、OpenHarmony版（OpenHarmony）	C、Java、C#、Android、Go
IoT Device SDK Tiny	> 32KB	> 128KB	> 100MHZ	适配了LiteOS/LINUX/MACOS/freertos，可以通过修改SDK来**适配其他环境**。	C

详细SDK使用指南，请参考：

- **设备侧 IoT Device C使用指南**
- **设备侧 IoT Device Java SDK使用指南**

- **设备侧 IoT Device C# SDK使用指南**
- **设备侧 IoT Device Android SDK使用指南**
- **设备侧 IoT Device Go SDK使用指南**
- **设备侧 IoT Device C for SDK Tiny使用指南**
- **设备侧 IoT Device Python SDK使用指南**

SDK主要功能矩阵，请参考：

表 3-1 SDK 主要功能矩阵

主要功能	C	Java	C#	Android	GO	python	C Tiny	ArkTS
属性上报	√	√	√	√	√	√	√	√
消息上报、下发	√	√	√	√	√	√	√	√
事件上报、下发	√	√	√	√	√	√	√	×
命令下发、响应	√	√	√	√	√	√	√	√
设备影子	√	√	√	√	√	√	√	√
OTA升级	√	√	√	√	√	√	√	×
bootstrap	√	√	√	√	√	√	√	×
时间同步	√	√	√	√	√	√	√	×
网关与子设备管理	√	√	√	√	√	√	√	×
端侧规则引擎	√	×	√	×	×	×	√	×
远程SSH	√	×	√	×	×	×	×	×
异常检测	√	×	√	×	×	×	×	×
端云安全通信（软总线）	√	×	√	×	×	×	×	×
M2M功能	√	×	√	×	×	×	×	×
泛协议接入	√	√	√	√	×	√	×	×

相关 FAQ

设备侧SDK相关问题

端侧设备集成相关问题

3.2 设备侧 IoT Device Java SDK 使用指南

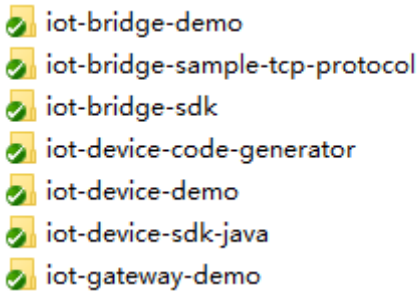
3.2.1 快速引用

mvn 引用

```
<dependencies>
   <dependency>
      <groupId>com.huaweicloud</groupId>
      <artifactId>iot-device-sdk-java</artifactId>
      <version>1.2.2</version>
   </dependency>
</dependencies>
```

源码获取及编译

1. 开发环境要求：已经安装JDK（版本1.8以上）和maven。
2. 访问**SDK下载页面**，下载SDK，整个工程包含以下子工程：

   - iot-device-sdk-java：sdk代码。
   - iot-device-demo：普通直连设备的demo代码。
   - iot-gateway-demo：网关设备的demo代码。
   - iot-bridge-sdk：网桥的sdk代码。
   - iot-bridge-demo：网桥的demo代码，用来演示如何将tcp设备桥接到平台。
   - iot-bridge-sample-tcp-protocol：子设备使用tcp协议连接网桥的样例。
   - iot-device-code-generator：设备代码生成器，可以根据产品模型自动生成设备代码。
3. 编译安装：进入到SDK根目录，执行mvn install。

3.2.2 使用说明

前言

huaweicloud-iot-device-sdk-java提供设备接入华为云IoT物联网平台的Java版本的SDK，提供设备和平台之间通讯能力，以及设备服务、网关服务、OTA等高级服务，并且针对各种场景提供了丰富的demo代码。IoT设备开发者使用SDK可以大大简化开发复杂度，快速的接入平台。

版本更新说明

表 3-2 javaSDK 版本更新说明

版本号	变更类型	说明
1.2.2	优化功能	修改日志框架，使用SLF4J实现日志框架选择
1.2.1	新增功能	OTA升级支持网关模式，端侧规则
1.2.0	新增功能	新增泛协议、国密算法、OBS升级包功能
	功能增强	1、BootstrapClient构造方法传入平台根CA证书方式优化，原有构造方法标为已废弃； 2、更新Samples中的ca.jks为包含平台各区域实例设备侧证书的所有权威根CA证书的证书文件； 3、修复部分拼写错误; 4、paho升级; 5、修复退避重连长时间后不再重试问题
1.1.2	功能增强	修改发放功能问题、兼容多region不同证书场景等
1.0.1	新功能	增加隐式订阅接口、数据压缩上报接口等等
1.0.0	功能增强	1、修改兼容V3旧接口逻辑 2、网关刷新子设备状态 3、修改默认订阅topic的qos、修改重连新链路及老链路、修改重连时间
0.8.0	功能增强	更换新的接入域名（iot-mqtts.cn-north-4.myhuaweicloud.com）和根证书。 如果设备使用老域名（iot-acc.cn-north-4.myhuaweicloud.com）接入，请使用 v0.6.0及以下版本的SDK
0.6.0	功能增强	调整OTA服务使用方式；完善md
0.5.0	新增功能	提供对接华为云物联网平台能力，方便用户实现接入、设备管理、命令下发等业务场景

3.2.3 快速接入

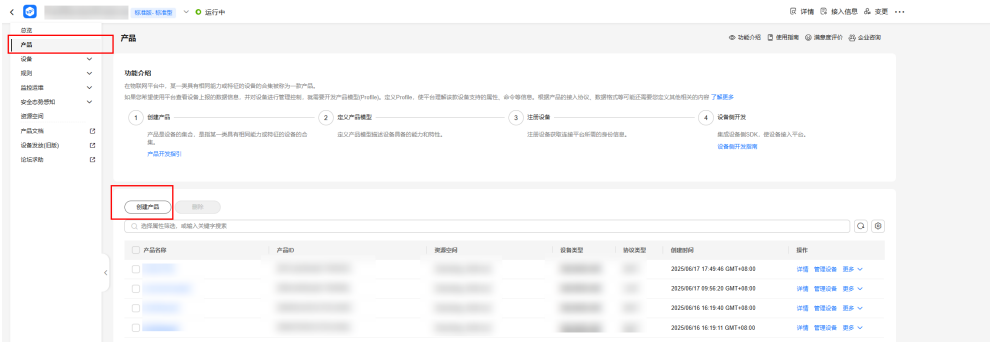
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-1 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-2 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-3 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

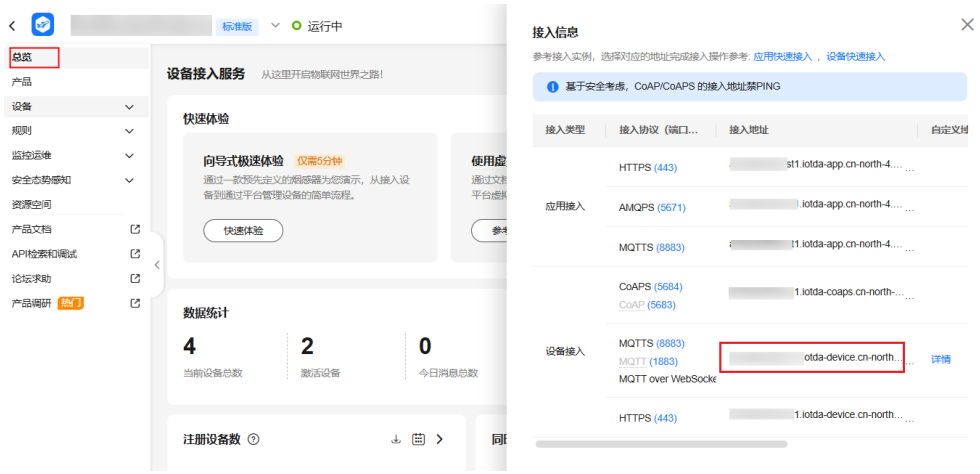
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备初始化

1. 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-4 接入信息-设备侧 MQTT 接入地址



2. 输入设备对接信息。创建设备时，需要写入在**注册设备**时获取的设备ID、密钥，以及**1**中获取的设备对接信息，注意格式为***ssl://域名信息:端口号*** 或 ***ssl://IP地址:端口号***

```
// 获取证书路径：加载iot平台的ca证书，进行服务端校验，使用sdk默认的ca.jks即可。
URL resource = MessageSample.class.getClassLoader().getResource("ca.jks");
File file = new File(resource.getPath());
//例如在iot-device-demo文件 MessageSample.java中修改以下参数
IoTDevice device = new IoTDevice("ssl://域名信息:8883",
        "5e06bfee334dd4f33759f5b3_demo", "mysecret", file);
```

说明

所有涉及设备ID和密码的文件均需要修改成对应的信息。

3. 建立连接。调用init接口，该接口是阻塞调用，如果建立连接成功会返回0。

```
if (device.init() != 0) {
    return;
}
```

如果连接成功就会打印：

```
2023-07-17 17:22:59  INFO MqttConnection:105 - Mqtt client connected. address :ssl://域名信息:8883
```

4. 创建设备并连接成功后，可以开始使用设备进行通信。调用IoT Device 的getClient接口获取设备客户端，客户端提供了消息、属性、命令等通讯接口。

须知

SDK默认提供断线重连功能，当调用device.init()后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。

消息上报

消息上报是指设备向平台上报消息。

1. 从device中获取客户端，调用IoTDevice的getClient接口即可获取到客户端。
2. 调用客户端的reportDeviceMessage接口来上报设备消息。在MessageSample这个例子中周期性上报消息：

```
while (true) {
    device.getClient().reportDeviceMessage(new DeviceMessage("hello"), new ActionListener() {
        @Override
        public void onSuccess(Object context) {
            log.info("reportDeviceMessage ok");
```

```
            }

            @Override
            public void onFailure(Object context, Throwable var2) {
                log.error("reportDeviceMessage fail: " + var2);
            }
        });

        //上报自定义topic消息，注意需要先在平台配置自定义topic
        String topic = "$oc/devices/" + device.getDeviceId() + "/user/wpy";
        device.getClient().publishRawMessage(new RawMessage(topic, "hello raw message "),
            new ActionListener() {
                @Override
                public void onSuccess(Object context) {
                    log.info("publishRawMessage ok: ");
                }

                @Override
                public void onFailure(Object context, Throwable var2) {
                    log.error("publishRawMessage fail: " + var2);
                }
            });

        Thread.sleep(5000);
    }
```

3. 修改MessageSample类的main函数，替换自己的设备参数后运行MessageSample类，查看日志打印包含以下内容则表示消息上报成功。

```
2024-04-16 16:43:11  INFO MessageSample:98 - reportDeviceMessage ok
```

4. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

图 3-5 设备列表-设备在线

5. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

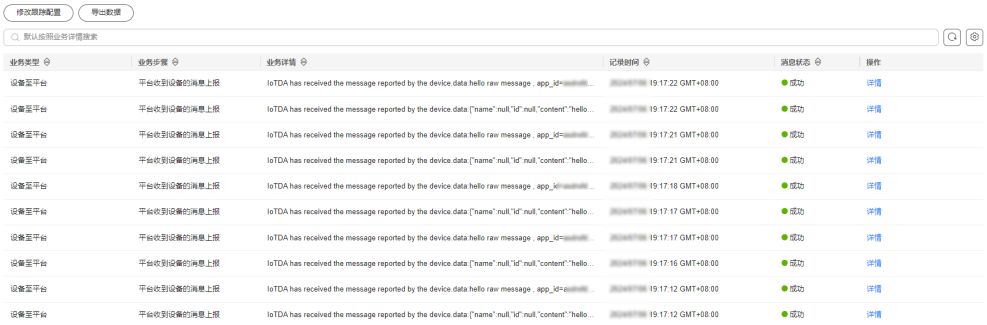
图 3-6 消息跟踪-启动消息跟踪

6. 查看消息跟踪，确认平台是否收到设备消息。

须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

图 3-7 消息跟踪-查看 device_sdk_java 消息跟踪



属性上报

1. 打开PropertySample类，这个例子中会定时的上报alarm、temperature、humidity、smokeConcentration这四个属性。

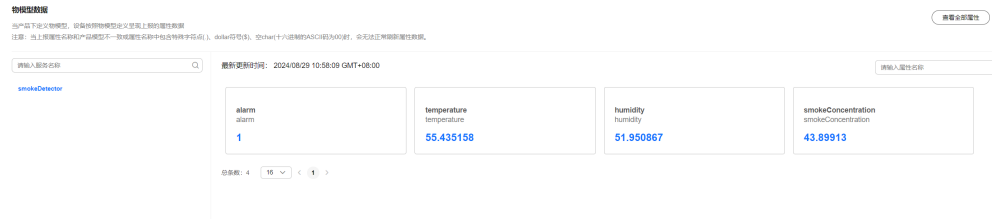
```
    //定时上报属性
      while (true) {

          Map<String ,Object> json = new HashMap<>();
          Random rand = new Random();

          //按照物模型设置属性
          json.put("alarm", 1);
          json.put("temperature", rand.nextFloat()*100.0f);
          json.put("humidity", rand.nextFloat()*100.0f);
          json.put("smokeConcentration", rand.nextFloat() * 100.0f);

          ServiceProperty serviceProperty = new ServiceProperty();
          serviceProperty.setProperties(json);
          serviceProperty.setServiceId("smokeDetector");//serviceId要和物模型一致


            device.getClient().reportProperties(Arrays.asList(serviceProperty), new ActionListener() {
             @Override
             public void onSuccess(Object context) {
                log.info("pubMessage success" );
             }

             @Override
             public void onFailure(Object context, Throwable var2) {
                log.error("reportProperties failed" + var2.toString());
             }
          });

          Thread.sleep(10000);
      }
  }
```

2. 修改PropertySample的main函数后直接运行PropertySample类，查看日志打印包含以下内容则表示属性上报成功。

```
2024-04-17 15:38:38  INFO PropertySample:144 - pubMessage success
```

3. 在平台设备详情页面可以看到最新上报的属性值。

图 3-8 物模型-属性上报



属性读写

1. 调用客户端的setPropertyListener方法来设置属性回调接口。在PropertySample这个例子中，实现了属性读写接口。
   - 写属性处理：实现了alarm属性的写操作，其他属性不支持写操作。
   - 读属性处理：将本地属性值按照接口格式进行拼装。

```
        device.getClient().setPropertyListener(new PropertyListener() {

        //处理写属性
        @Override
        public void onPropertiesSet(String requestId, List<ServiceProperty> services) {
            // 遍历service
            for (ServiceProperty serviceProperty : services) {

                log.info("OnPropertiesSet, serviceId is {}", serviceProperty.getServiceId());

                // 遍历属性
                for (String name : serviceProperty.getProperties().keySet()) {
                    log.info("property name is {}", name);
                    log.info("set property value is {}", serviceProperty.getProperties().get(name));
                }

            }
            // 修改本地的属性值
            device.getClient().respondPropsSet(requestId, IotResult.SUCCESS);
        }

        /**
         * 处理读属性。多数场景下，用户可以直接从平台读设备影子，此接口不用实现。
         * 但如果需要支持从设备实时读属性，则需要实现此接口。
         */
        @Override
        public void onPropertiesGet(String requestId, String serviceId) {
            log.info("OnPropertiesGet, the serviceId is {}", serviceId);
            Map<String, Object> json = new HashMap<>();
            Random rand = new SecureRandom();
            json.put("alarm", 1);
            json.put("temperature", rand.nextFloat() * 100.0f);
            json.put("humidity", rand.nextFloat() * 100.0f);
            json.put("smokeConcentration", rand.nextFloat() * 100.0f);

            ServiceProperty serviceProperty = new ServiceProperty();
            serviceProperty.setProperties(json);
            serviceProperty.setServiceId("smokeDetector");

            device.getClient().respondPropsGet(requestId, Arrays.asList(serviceProperty));
        }
    });
```

说明

1. 属性读写接口需要调用respondPropsGet和respondPropsSet接口来上报操作结果。
2. 如果设备不支持平台主动到设备读，onPropertiesGet接口可以空实现

2. 运行PropertySample类，然后在平台上设备影子页面查看当前alarm属性值为1。

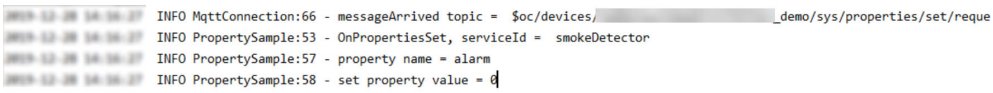
图 3-9 设备影子-查看 alarm 属性

3. alarm属性修改为0。

图 3-10 设备影子-属性配置 alarm

4. 查看设备侧日志，看到设备收到属性设置，alarm被修改为0。

图 3-11 查看属性设置



命令下发

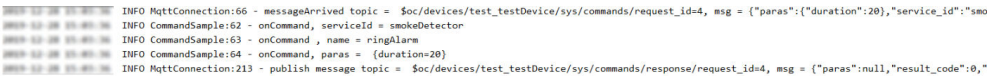
设置命令监听器用来接收平台下发的命令，在回调接口里，需要对命令进行处理，并上报响应。

1. 在CommandSample例子中实现了命令的处理，收到命令后仅进行打印，然后调用respondCommand上报响应。

```
    device.getClient().setCommandListener(new CommandListener() {
            @Override
            public void onCommand(String requestId, String serviceId, String commandName,
Map<String, Object> paras) {
                log.info("onCommand, serviceId = {}", serviceId);
                log.info("onCommand , name = {}", commandName);
                log.info("onCommand, paras =  {}", paras.toString());

                //处理命令

                //发送命令响应
                device.getClient().respondCommand(requestId, new CommandRsp(0));
            }

        });
```

2. 直接运行CommandSample类，然后在平台上下发命令，命令的serviceId填smokeDetector、命令名填ringAlarm、参数携带duration为整数20。
3. 查看日志，看到设备收到命令并上报了响应。

图 3-12 查看日志



3.2.4 使用代码生成器

概述

SDK提供了设备代码生成器，需要下载**SDK源码**。用户只需要提供产品模型文件，就能自动生成设备代码框架。代码生成器可以解析设备模型文件，然后对模型里定义的每个服务，生成对应的service类，然后生成一个设备主类，在main函数中创建设备并注册设备服务实例。

操作步骤

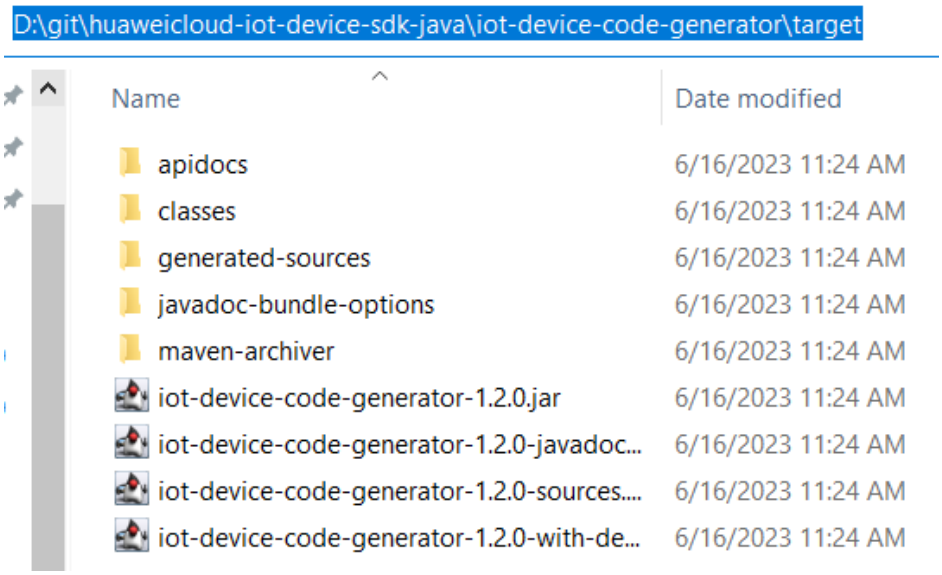
1. 生成物模型代码。
   a. 下载huaweicloud-iot-device-sdk-java工程，解压缩后进入huaweicloud-iot-device-sdk-java目录执行“mvn install”。

   图 3-13 执行编译命令

```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for huaweicloud iot device sdk project for java 1.2.0:
[INFO]
[INFO] huaweicloud iot device sdk project for java ........ SUCCESS [  0.802 s]
[INFO] iot-device-sdk-java ................................ SUCCESS [  3.976 s]
[INFO] iot-device-demo .................................... SUCCESS [  4.112 s]
[INFO] iot-bridge-sdk ..................................... SUCCESS [ 17.187 s]
[INFO] iot-bridge-demo .................................... SUCCESS [  4.168 s]
[INFO] iot-gateway-demo ................................... SUCCESS [  2.852 s]
[INFO] iot-device-code-generator .......................... SUCCESS [  2.658 s]
[INFO] iot-bridge-sample-tcp-protocol ..................... SUCCESS [  4.122 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  39.978 s
[INFO] Finished at: 2023-06-16T11:25:00+08:00
[INFO] ------------------------------------------------------------------------
```

   b. 执行完成会在iot-device-code-generator的target下生成可执行jar包。

图 3-14 生成可执行 jar 包



c. 将产品模型文件保存到本地，比如模型文件“smokeDetector.zip”放到D盘。

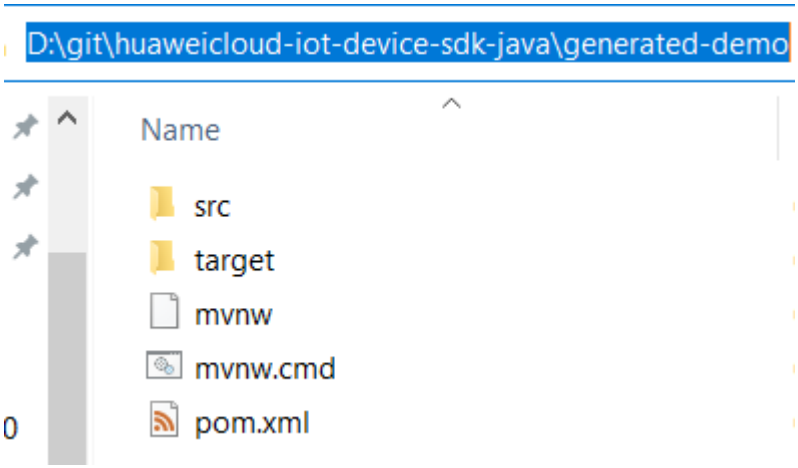
d. 访问SDK根目录，执行“java -jar .\iot-device-code-generator\target\iot-device-code-generator-1.2.0-with-deps.jar D:\smokeDetector.zip”。

图 3-15 执行代码生成命令

```
PS D:\git\huaweicloud-iot-device-sdk-java> java -jar .\iot-device-code-generator\target\iot-device-code-generator-1.2.0-with-deps.jar D:\smokeDetector.zip
2023-06-16 11:30:47  INFO DeviceCodeGenerator:147 - the file generation path is :D:\git\huaweicloud-iot-device-sdk-java\generated-demo\src\main\java\com\huaweicloud\sdk\iot\device\demo\smokeDetectorService.java
2023-06-16 11:30:47  INFO DeviceCodeGenerator:73 - demo code generated to: D:\git\huaweicloud-iot-device-sdk-java\generated-demo
```

e. 查看huaweicloud-iot-device-sdk-java目录下已生成generated-demo包则设备代码成功生成。

图 3-16 代码生成成功



2. 编译及运行物模型代码。

a. 访问“huaweicloud-iot-device-sdk-java\generated-demo”，执行“mvn install”。

```
PS D:\git\huaweicloud-iot-device-sdk-java> cd .\generated-demo\
PS D:\git\huaweicloud-iot-device-sdk-java\generated-demo> mvn install
```

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.386 s
[INFO] Finished at: 2023-06-16T11:31:47+08:00
[INFO] ------------------------------------------------------------------------
PS D:\git\huaweicloud-iot-device-sdk-java\generated-demo> dir target


    Directory: D:\git\huaweicloud-iot-device-sdk-java\generated-demo\target


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         6/16/2023  11:31 AM                apidocs
d-----         6/16/2023  11:31 AM                classes
d-----         6/16/2023  11:31 AM                generated-sources
d-----         6/16/2023  11:31 AM                javadoc-bundle-options
d-----         6/16/2023  11:31 AM                maven-archiver
-a----         6/16/2023  11:31 AM          29924 iot-device-demo-ganerated-1.2.0-javado
                                                  c.jar
-a----         6/16/2023  11:31 AM           6728 iot-device-demo-ganerated-1.2.0-source
                                                  s.jar
-a----         6/16/2023  11:31 AM       11530020 iot-device-demo-ganerated-1.2.0-with-d
                                                  eps.jar
-a----         6/16/2023  11:31 AM           8031 iot-device-demo-ganerated-1.2.0.jar
```

b. 执行java -jar .\target\iot-device-demo-ganerated-1.2.0-with-deps.jar ssl://***域名信息***:8883 ***device_id secret***，三个参数分别为设备接入地址、设备id和密码，运行生成的demo。

```
D:\git\huaweicloud-iot-device-sdk-java\generated-demo>java -jar .\target\iot-device-demo-ganerated-1.2.0-with-deps.jar ssl://域名信息:8883 5e06bfee334dd4f33759f5b3_demo secret
2024-04-17 15:50:53  INFO AbstractService:73 - create device, the deviceId is 5e06bfee334dd4f33759f5b3_demo
2024-04-17 15:50:54  INFO MqttConnection:204 - try to connect to ssl://域名信息:8883
2024-04-17 15:50:55  INFO MqttConnection:228 - connect success, the uri is ssl://域名信息:8883
```

3. 修改物模型代码。生成的代码已经完成了服务的定义和注册，用户只需要进行少量的修改即可。
   - 命令接口，需要添加具体的实现逻辑。

```
/********** commands ***********/
@DeviceCommand
public CommandRsp ringAlarm (Map<String, Object> paras) {
  //todo 请在这里添加命令处理代码
  return new CommandRsp(0);
}
```

   - getter方法，生成的代码是返回随机值，需要改为从传感器读取数据。
   - setter方法，生成的代码只完成了属性的修改保存，还需要添加真实的逻辑处理，比如向传感器下发指令。

3.2.5 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与物联网平台的上下行通讯。SDK当前支持的功能有：

表 3-3 功能支持列表

功能	描述说明
设备初始化	作为客户端使用MQTT协议接入到华为云平台。分为密钥认证和证书认证两种认证方式。
断线重连	设备连接失败或网络不稳定等原因导致被动断开连接时，会进行一个重连操作。
消息上报、下发	消息上报用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理，消息下发用于平台向设备下发消息，设备对收到的消息进行处理。
属性上报、设置、查询	属性上报用于设备按产品模型中定义的格式将属性数据上报给平台。属性设置用于平台设置设备属性。设备收到属性设置请求后，需要将执行结果返回给平台。属性查询用于平台查询设备的属性，设备收到属性查询请求后，需要将设备的属性数据返回给平台
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
获取设备影子	用于设备向平台获取设备影子数据。
软固件（OTA）升级	用于与平台配合下载OTA升级包。
时间同步	设备向平台发起时间同步请求。
网关与子设备	网关设备：通过平台支持的协议，直接连接到平台的设备。子设备：针对未实现TCP/IP协议栈的设备，由于无法直接同物联网平台通信，它需要通过网关进行数据转发。当前仅支持通过mqtt协议直连到平台的设备作为网关设备。
文件上传、下载	华为物联网平台支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。当设备采用HTTPS方式将文件上传到OBS服务进行备份时，您可以在OBS服务管理已上传的设备文件。
端侧规则	通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前物联网平台支持两种联动规则：云端规则和端侧规则。
设备发放	分为证书认证、密钥认证。主要用于分发到不同局点、实例的场景，动态完成不同批次设备初始化配置。
面向物模型编程	面向物模型编程指的是，基于SDK提供的物模型抽象能力，设备代码只需要按照物模型定义设备服务，SDK就能自动的和平台通讯，完成属性的同步和命令的调用。 相比直接调用客户端接口和平台进行通讯，面向物模型编程简化了设备侧代码的复杂度，让设备代码只需要关注业务，而不用关注和平台的通讯过程。

功能	描述说明
泛协议接入	当非HTTP、MQTT、LWM2M等第三方协议接入时，需要在平台外部完成协议转换。推荐使用网关来完成协议转换，将第三方协议转成MQTT协议。

3.3 设备侧 IoT Device C 使用指南

3.3.1 使用说明

前言

设备侧 IoT Device C for Linux SDK提供设备接入华为云IoT物联网平台的C版本的SDK，提供设备和平台之间通讯能力，以及设备服务、网关服务、OTA等高级服务，并且针对各种场景提供了丰富的demo代码。相关集成指导请参考**设备侧 IoT Device C DK使用指南**。

使用说明

- SDK需运行在Linux操作系统上。
- SDK依赖openssl库和paho库，如果开发者有自己的编译链，需要自行编译openssl/paho/zlib/华为安全函数库等库文件。
- 对于使用MCU+模组形式接入的部分设备，请使用**C Tiny SDK**进行开发。

版本更新说明

表 3-4 C 语言版本更新说明

版本号	变更类型	功能描述说明
1.2.0	功能增强	新增SDK测试代码及Demo，优化代码使用。
1.1.5	功能增强	新增使用示例
1.1.4	功能增强	更新OTA升级传输格式
1.1.3	功能增强	更新conf\rootcert.pem证书，增加日志上传
1.1.2	新功能	增加规则引擎、M2M、gn编译文件、异常检测、日志打印时间戳、MQTT_DEBUG、国密算法、远程配置、端云安全通信（软总线）功能
1.1.1	新功能	新增SSH远程运维功能
1.1.0	新功能	增加MQTT5.0功能，优化代码，修复内存溢出问题
1.0.1	功能增强	增加mqtts不校验平台公钥场景、TLS版本为V1.2、增加消息存储样例等场景
0.9.0	新功能	增加网关更新子设备状态接口

版本号	变更类型	功能描述说明
0.8.0	功能增强	更换新的接入域名（iot-mqtts.cn-north-4.myhuaweicloud.com）和根证书。 如果设备使用老域名（iot-acc.cn-north-4.myhuaweicloud.com）接入，请使用 v0.5.0版本的SDK
0.5.0	功能增强	sdk预置了设备接入地址及华为物联网平台配套的CA证书，支持对接华为云物联网平台。

3.3.2 代码下载与编译

环境信息

SDK需运行在Linux操作系统上，并安装好gcc（建议4.8及以上版本）。SDK依赖openssl库和paho库，如果开发者有自己的编译链，需要自行编译openssl/paho库文件。

安装依赖

表 3-5 SDK 依赖组件

库名	描述	具体编译方式
openssl	OpenSSL 是一个开源的密码学工具库，提供了广泛的加密、解密、证书管理以及安全通信功能，广泛应用于网络通信、数据保护和系统安全等领域。	**编译openssl库**
paho.mqtt.c	paho.mqtt.c 是 Eclipse Paho 项目提供的一个 轻量级 MQTT 客户端 C 语言库，专门用于实现 MQTT（Message Queuing Telemetry Transport）协议通信。MQTT 是一种基于发布/订阅模型的物联网（IoT）通信协议，以低带宽、低功耗和高效率著称。	**编译paho库**
zlib	zlib 是一个广泛使用的开源数据压缩库，主要用于无损数据压缩和解压。它实现了 DEFLATE 压缩算法（结合了 LZ77 和 Huffman 编码），被许多软件和协议作为标准压缩方式。	**编译zlib库**
boundscheck	对涉及内存边界检查或数组越界检测，用于提高代码安全性，防止缓冲区溢出等漏洞。	**编译华为安全函数库**
ssh	用于在应用程序中实现安全的远程登录、文件传输和加密通信。它通过加密通道保护数据传输，防止窃听、篡改和中间人攻击。	**编译libssh库**
nopoll	一个轻量级的开源 WebSocket 库，基于 C 语言实现，主要用于在嵌入式系统或资源受限环境中实现高效的 WebSocket 通信。	**编译libnopoll库**

库名	描述	具体编译方式
curl	用于访问网页、调用 REST API（如 GET/POST/PUT/DELETE），文件上传、下载等。	编译curl库

3.3.3 快速接入

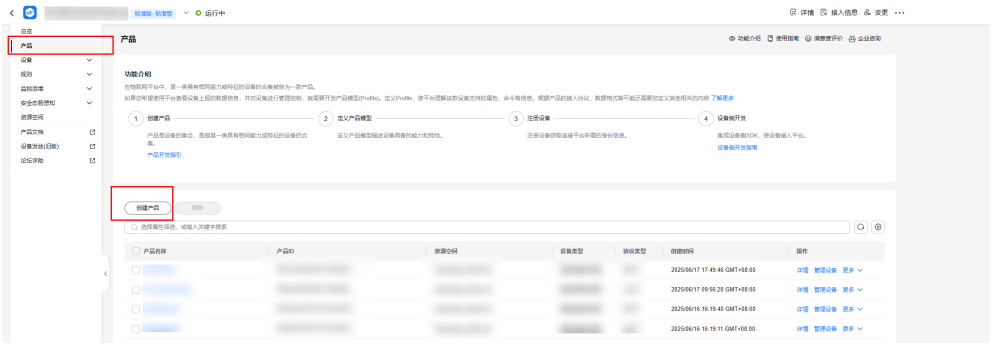
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问设备接入服务，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-17 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建资源空间。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-18 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传1获取的产品模型文件。

图 3-19 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

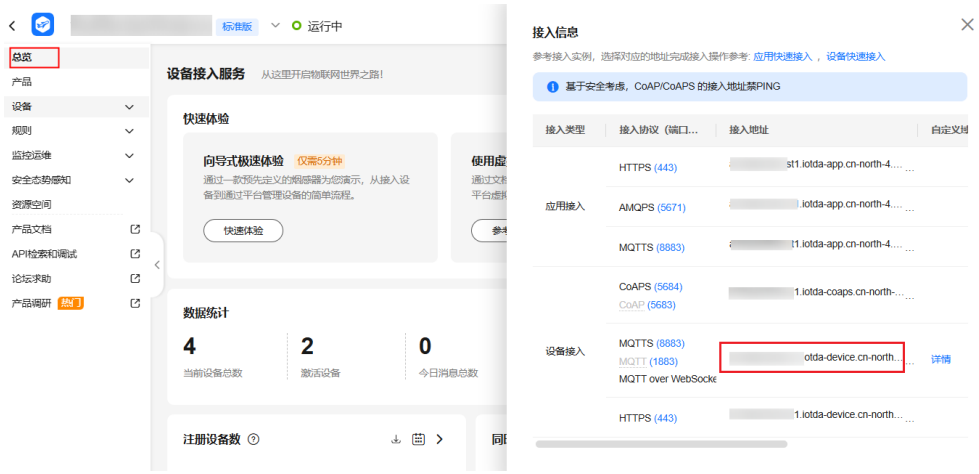
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备初始化

步骤1 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-20 接入信息-设备侧 MQTT 接入地址



步骤2 打开文件demos/device_demo/basic_test.c，将之前获取到的接入地址、设备ID以及设备密钥填入对应位置。

```
// 创建一个设备并初始化。在basic_test.c中修改连接参数:
// You can get the access address from IoT Console "Overview" -> "Access Information"
char *g_address = "域名";
char *g_port = "8883";
char *g_deviceId = "设备id";
char *g_secret = "设备密钥";

static void mqttDeviceSecretInit(char *address, char *port, char *deviceId, char *password) {

    IOTA_Init("."); // The certificate address is set to: ./conf/rootcert.pem
    IOTA_SetPrintLogCallback(MyPrintLog); // Set log printing method

    // MQTT protocol when the connection parameter is 1883; MQTTS protocol when the connection parameter is 8883
    IOTA_ConnectConfigSet(address, port, deviceId, password);
    int ret = IOTA_Connect();
    if (ret != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "basic_test: IOTA_Connect() error, Auth failed, result %d\n", ret);
        return -1;
    }
}
```

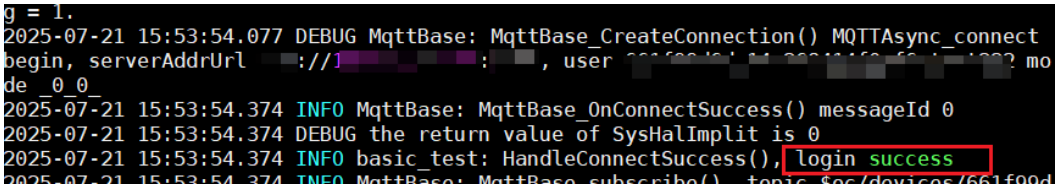
注意

平台的证书可以从源码中**conf/rootcert.pem**下载。

步骤3 编译及运行。在根目录运行以下命令，日志中打印“login success”表示设备鉴权成功。

```
export LD_LIBRARY_PATH=./lib/  #加载库文件
make clean
make device_demo
./basic_test
```

图 3-21 连接成功



步骤4 查看设备运行情况。在控制台的设备详情页面查看设备已在线。

图 3-22 设备列表-设备在线

----结束

须知

SDK默认提供断线重连功能，当调用设备初始化后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。若是客户不需要该机制，请见：**断线重连**文档。

消息上报

消息上报是指设备向平台上报消息。API接口如下：

表 3-6 C 语言消息上报接口

功能	接口
自定义Topic消息上报	HW_API_FUNC HW_INT IOTA_RawTopicMessageReport(HW_CHAR *topic, HW_CHAR *payload, int qos, void *context);
系统Topic消息上报	HW_API_FUNC HW_VOID IOTA_SetUndefinedMessageCallback(PFN_MESSAGE_CALLBACK_HANDLER callbackHandler);

1. ./demos/device_demo/message_test.c是一个上报消息例子，可以通过IOTA_MessageDataReport函数将消息上报给平台。

```
void basicMassage()
{
    // 订阅系统topic
    SubscribeAll();
    // 向连接设备上报一个id为123的消息，消息内容为"hello!"
    ST_IOTA_MESS_REP_INFO mass = {NULL, "name", "id", "hello!", "test"};
    int messageId = IOTA_MessageDataReport(mass, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "basicMassage() failed, messageId %d\n", messageId);
    }
    TimeSleep(3000);
}
```

2. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

图 3-23 设备列表-设备在线

3. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

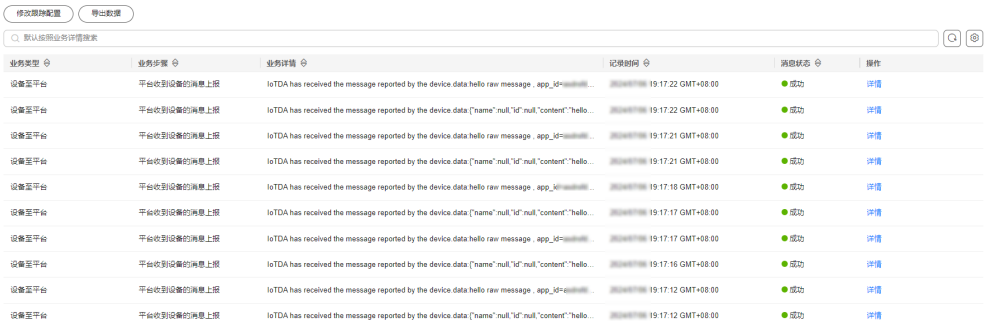
图 3-24 消息跟踪-启动消息跟踪

4. 查看消息跟踪，确认平台是否收到设备消息。

须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

图 3-25 消息跟踪-查看 device_sdk_java 消息跟踪



属性上报

属性上报指的是设备将当前属性值上报给平台。属性设置指的是平台设置设备的属性值，API接口如下。

表 3-7 C 语言属性上报接口

功能	接口
设备属性上报接口	HW_INT IOTA_PropertiesReport(ST_IOTA_SERVICE_DATA_INFO pServiceData[], HW_INT serviceNum, HW_INT compressFlag, void *context)

1. 源代码中**demos/device_demo/properties_test.c**是一个属性上报的例子，可以通过IOTA_PropertiesReport函数将属性上报给平台。

```
void Test_propertiesReport() {
  int serviceNum = 1;//上报的service个数
  ST_IOTA_SERVICE_DATA_INFO services[serviceNum];

  //---------------the data of service-------------------------------
  char *service = "{\"temperature\": 28}";
  services[0].event_time = NULL;
  services[0].service_id = "smokeDetector";
  services[0].properties = service;

  int messageId = IOTA_PropertiesReport(services, serviceNum, 0, NULL);
  if(messageId != 0) {
    PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_batchPropertiesReport() failed, messageId
%d\n", messageId);
  }
}
```

2. 在左侧导航栏中选择“设备 > 所有设备”，选择对应设备单击“详情”，在“设备影子”处可以看到上报的属性值。

图 3-26 设备影子-temperature

须知

若是没有查看到属性上报值，确认是否已上传产品模型：**上传产品模型**。

命令下发

设置命令监听器用来接收平台下发的命令，在回调接口里，将对命令进行处理，并上报响应。./demos/device_demo/command_test.c是一个处理平台命令下发的例子。下面代码的HandleCommandRequest函数作为命令监听器，即：

- 注册回调函数，设备接收命令下发（profile中定义的命令）。HW_API_FUNC HW_VOID IOTA_SetCmdCallback(PFN_CMD_CALLBACK_HANDLER callbackHandler) 通过该接口设置命令回调函数

```
IOTA_SetCmdCallback(HandleCommandRequest);
```

- 当云端下发命令或端侧规则触发执行命令时，定义的函数会被调用。

```
int beepState = 0; // 物模型，蜂鸣器状态
// 响应命令下发
static void Test_CommandResponse(char *requestId)
{
  /* 请客户自行实现命令下发响应参数，以下为示例 */
  char pcCommandRespense[100] = {0}; // in service accumulator
  (void)sprintf_s(pcCommandRespense, sizeof(pcCommandRespense), "{\"Beep_State\": %d}",
beepState);

  int result_code = 0;
  char *response_name = "Smoke_Control_Beep";

  int messageId = IOTA_CommandResponse(requestId, result_code, response_name,
pcCommandRespense, NULL);
  if (messageId != 0) {
      PrintfLog(EN_LOG_LEVEL_ERROR, "Test_CommandResponse() failed, messageId %d\n",
messageId);
  }
}

void HandleCommandRequest(EN_IOTA_COMMAND *command)
{
    if (command == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "command_test: HandleCommandRequest(), messageId %d,
service_id %s, command_name %s, request_id %s\n",
        command->mqtt_msg_info->messageId, command->service_id, command->command_name,
command->request_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "command_test: HandleCommandRequest(), request_id %s\n",
command->request_id);

    /* 请客户自行实现命令下发处理逻辑 */

      Test_CommandResponse(command->request_id); //response command
}
```

3.3.4 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与物联网平台的上下行通讯。SDK当前支持的功能有：

表 3-8 C 语言功能支持列表

功能	描述说明
设备接入	作为客户端使用MQTT协议接入到华为云平台。分为证书认证与密钥认证两种认证方式。
断线重连	当设备由于网络不稳定或其他原因，导致连接断开，设备将每隔一段时间进行重新连接，直到连接成功。
消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。

功能	描述说明
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
设备影子	用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。
软固件（OTA）升级	用于与平台配合下载OTA升级包。
时间同步	设备向平台发起时间同步请求。
网关与子设备	网关设备：通过平台支持的协议，直接连接到平台的设备。子设备：针对未实现TCP/IP协议栈的设备，由于无法直接同物联网平台通信，它需要通过网关进行数据转发。当前仅支持通过mqtt协议直连到平台的设备作为网关设备。
文件上传/下载	支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。
异常检测	提供安全检测能力，可持续检测设备的安全威胁。包括：1、内存泄漏检测 2、异常端口检测3、CPU使用率检测 4、磁盘空间检测 5、电池电量检测
规则引擎	通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前物联网平台支持两种联动规则：云端规则和端侧规则。
MQTT5.0	MQTT5.0版本协议，新增了MQTT5.0新特性：对比数据、Clean Start 与 Session Expiry Interval、有效载荷标识与内容类型、主题别名、用户属性。
国密算法	一种TLS加密算法。
远程配置	提供远程配置功能，用户可用于在不中断设备运行的情况下，远程更新设备的系统参数、运行参数等配置信息。
远程登录	支持通过控制台远程SSH登录设备，可在控制台输入设备支持的命令，进行功能调试及问题定位，从而方便地实现设备管理及远程运维
泛协议接入	当非HTTP、MQTT、LWM2M等第三方协议接入时，需要在平台外部完成协议转换。推荐使用网关来完成协议转换，将第三方协议转成MQTT协议。
软总线	当使用鸿蒙系统时。通过平台下发设备组，设备可通过软总线实现物物互联。IoTDA可以进行安全群组管理以及下发群成员之间通信的授信标识。
设备发放	分为证书认证、密钥认证。主要用于分发到不同局点、实例，动态完成不同批次设备初始化配置。发放完成的数据可以通过设备接入进行数据传输。

3.4 设备侧 IoT Device C# SDK 使用指南

3.4.1 使用说明

前言

设备侧 IoT Device C# SDK提供设备接入华为云IoT物联网平台的C#版本的SDK，提供设备和平台之间通讯能力，以及设备服务、OTA等高级服务，并且针对各种场景提供了丰富的demo代码。相关集成指导请参考**设备侧 IoT Device C# SDK使用指南**。

使用说明

- 已安装 DotNet SDK 8.0。
  - **点此查看 .NET 安装指导**
  - **点此下载 .NET 8.0**
- 已安装对应IDE（Visual Studio Code 2017+, Rider 17.0.6+）。理论上本SDK不依赖IDE，开发者可根据喜好选择IDE或者直接使用CLI。

说明

具体使用方式请看gitHub上的**README**。

目录结构

- iot-device-sdk-csharp：sdk代码。
- iot-device-demo：功能演示的demo代码。
- iot-tcp-device：用于模拟网关、网桥场景下通过tcp连接的设备demo。

版本更新说明

表 3-9 C# SDK 版本更新说明

版本号	变更类型	说明
1.3.4	功能增强	1.优化日志打印 2.oc开头SubscribeTopic 返回 topic 3.demo优化 4.网关接口bug修复 5. 升级目标框架。 以及其它优化
1.3.3	新增功能	OTA升级支持网关模式
1.3.2	功能增强	更新服务器ca证书
1.3.1	修复	修复空指针异常，MQTT对象未释放等问题
1.3.0	新功能	支持通过OBS升级软固件包
1.2.0	新功能	增加泛协议功能
1.1.1	功能增强	添加网关删除子设备功能，完善中英文描述
1.1.0	新功能	新增网关与物模型功能

版本号	变更类型	说明
1.0.0	第一次发布	提供基础的设备接入能力，sdk预置了设备接入地址及华为IoTDA平台配套的CA证书
0.5.0	新增功能	提供对接华为云物联网平台能力，方便用户实现接入、设备管理、命令下发等业务场景

3.4.2 快速接入

本章将使用SmokeDetector Demo 指导您快速体验产品模型的创建、Demo的配置与启动、以及SDK基本功能的使用。Demo源码可见：**SmokeDetector.cs**。为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感例，体验消息上报、属性上报等功能。

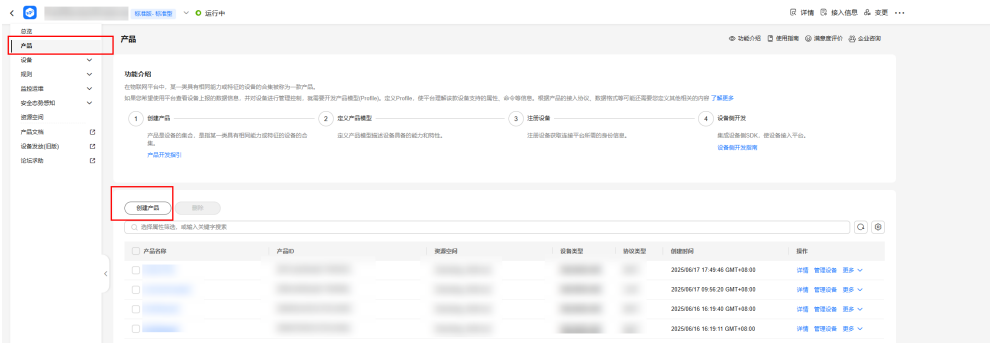
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-27 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。

设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-28 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-29 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

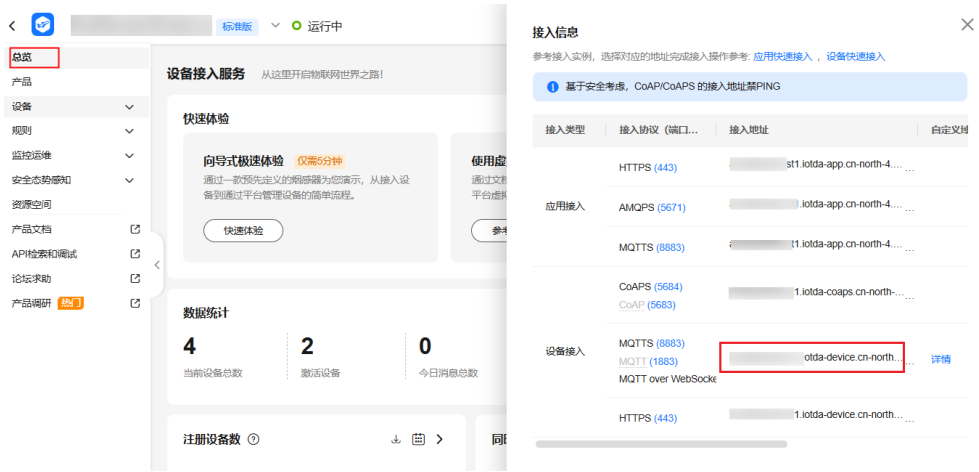
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备接入

步骤1 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-30 接入信息-设备侧 MQTT 接入地址



步骤2 打开文件iot-device-demo/config/DemoConfigDefault.json，将之前获取到的接入地址、设备ID以及设备密钥填入对应位置，或者将配置填入下面模板后再覆盖iot-device-demo/config/DemoConfigDefault.json。

```
{
  "DemoName": "SmokeDetector",
  "AuthConfig": {
    "AuthUseCert": false,
    "ServerAddress": "填入接入地址",
    "ServerPort": 8883,
    "DeviceId": "填入设备ID",
    "DeviceSecret": "填入设备密钥"
  }
}
```

其中DemoName用于指定运行的Demo，此处为SmokeDetector指定基于烟雾报警器产品模型的物模型编程Demo。可以进入SDK目录下运行以下命令查看所有Demo名称列表：

```
cd iot-device-demo
dotnet run --project .\iot-device-demo.csproj
```

图 3-31 查看 Demo 名称列表

步骤3 编译及运行。

1. 进入SDK目录下，执行命令运行demo，其中.\config\DemoConfigDefault.json为配置文件路径。

   ```
   cd iot-device-demo
   dotnet run --project .\iot-device-demo.csproj  .\config\DemoConfigDefault.json
   ```

   说明

   如果您使用旧版本5.0+的SDK并且无法升级到.NET 8.0，可以修改以下文件的TargetFramework。

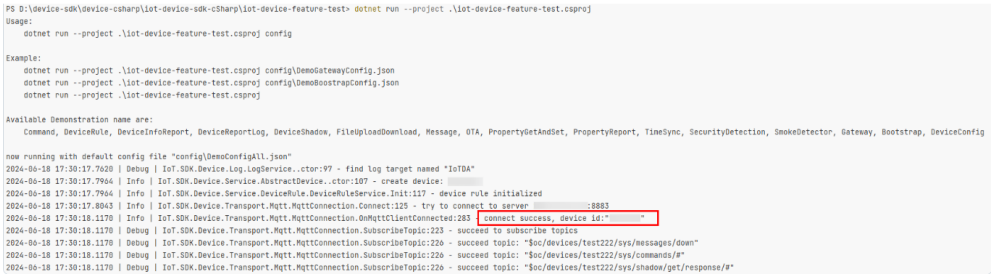
   图 3-32 .NET 版本

   ```
   <TargetFramework>net8.0</TargetFramework>          iot-device-demo.csproj 5
   <TargetFramework>net8.0</TargetFramework>          iot-tcp-device.csproj 5
   ```

2. 执行命令编译project，输出示例如下，connect success则表示连接成功。

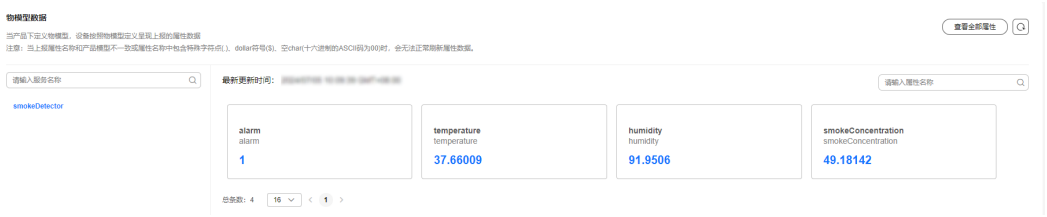
```
dotnet clean
dotnet build
```

图 3-33 连接成功



步骤4 查看设备运行情况。在控制台的设备详情页面查看设备已在线以及设备上报的物模型数据。

图 3-34 查看上报数据-smokeDetector



----结束

须知

SDK默认提供断线重连功能，当调用设备初始化后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。若是客户不需要该机制，请见：**断线重连**文档。

3.4.3 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与IoTDA服务（后续简出现的“服务”、“平台“均指IoTDA服务）双向通讯。SDK当前支持的功能有：

表 3-10 C# 功能支持

功能	描述说明
设备连接鉴权	作为客户端使用MQTT协议接入到平台。分为**证书鉴权**与**密钥鉴权**两种认证方式。
断线重连	当设备由于网络不稳定导致连接失败或中断时，会自动进行重新连接。
消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。

功能	描述说明
属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
设备影子	用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。
软固件（OTA）升级	用于与平台配合下载OTA升级包。
时间同步	设备向平台发起时间同步请求。
网关与子设备	网关设备：通过平台支持的协议，直接连接到平台的设备。子设备：针对未实现TCP/IP协议栈的设备，由于无法直接同平台通信，它需要通过网关进行数据转发。当前仅支持通过mqtt协议直连到平台的设备作为网关设备。
文件上传/下载	支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。
异常检测	提供安全检测能力，可持续检测设备的安全威胁。包括：1、内存泄漏检测 2、异常端口检测3、CPU使用率检测 4、磁盘空间检测 5、电池电量检测
规则引擎	通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前平台支持两种联动规则：云端规则和端侧规则。
远程配置	提供远程配置功能，用户可用于在不中断设备运行的情况下，远程更新设备的系统参数、运行参数等配置信息。
远程登录	支持通过控制台远程SSH登录设备，可在控制台输入设备支持的命令，进行功能调试及问题定位，从而方便地实现设备管理及远程运维
泛协议接入	当非HTTP、MQTT、LWM2M等第三方协议接入时，需要在平台外部完成协议转换。推荐使用网关来完成协议转换，将第三方协议转成MQTT协议。
设备发放	分为证书认证、密钥认证。主要用于分发到不同局点、实例的场景，动态完成不同批次设备初始化配置。

3.5 设备侧 IoT Device Android SDK 使用指南

3.5.1 使用说明

前言

设备侧 IoT Device Android SDK提供设备接入华为云IoT物联网平台的Android版本的SDK，提供设备和平台之间通讯能力，以及设备服务、OTA等高级服务，并且针对各种

场景提供了丰富的demo代码。相关集成指导请参考**设备侧 IoT Device Android SDK 使用指南**。

使用说明

- 已安装Android Studio

目录结构

1. SDK目录结构
   - huaweicloud-iot-device-sdk-android：sdk代码
2. 第三方类库使用版本
   - org.eclipse.paho.client.mqttv3：v1.2.5
   - gson：v2.8.6

版本更新说明

表 3-11 Android SDK 版本更新说明

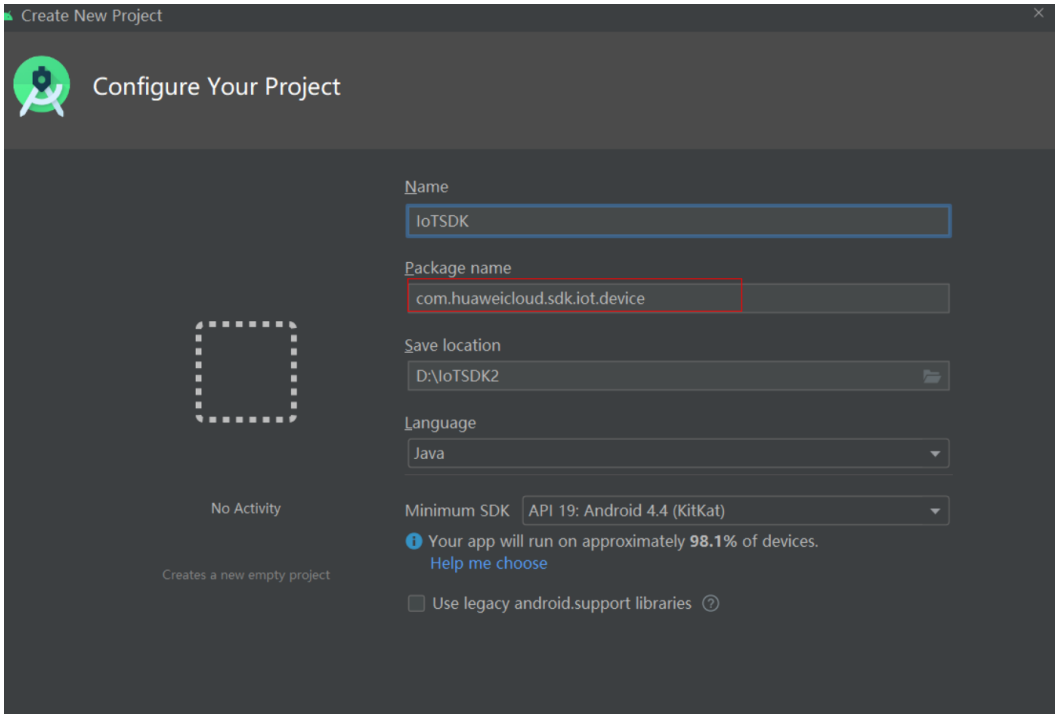
版本号	变更类型	说明
1.0.0	第一次发布	提供基础的设备接入能力

3.5.2 编译及代码工程设置

SDK 编译

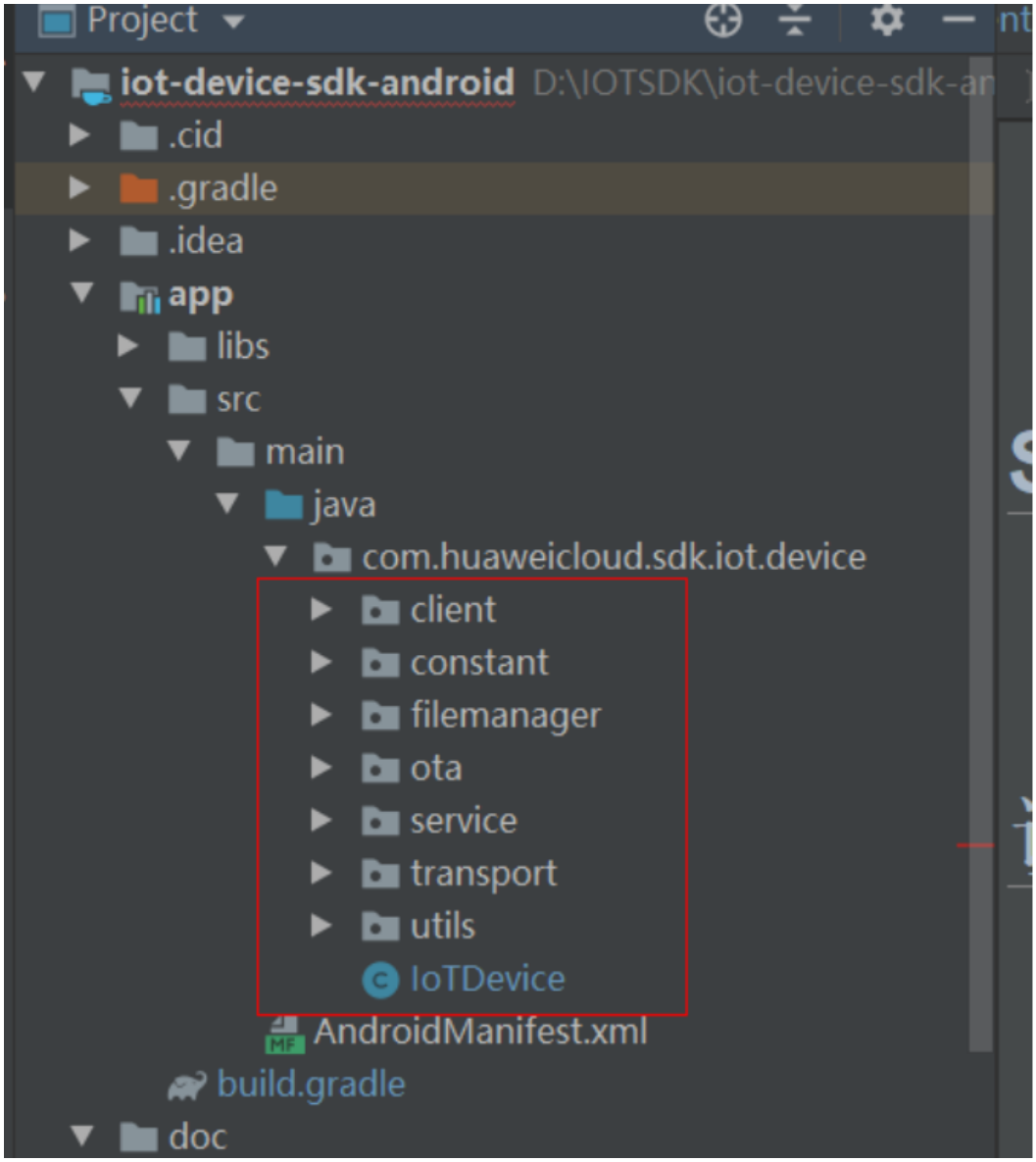
步骤1 使用Android studio创建Android工程，并设置包名为com.huaweicloud.sdk.iot.device。

图 3-35 Android studio 创建 Android 工程



步骤2 下载**iot-device-sdk-android**工程中java源码到com.huaweicloud.sdk.iot.device包下面。

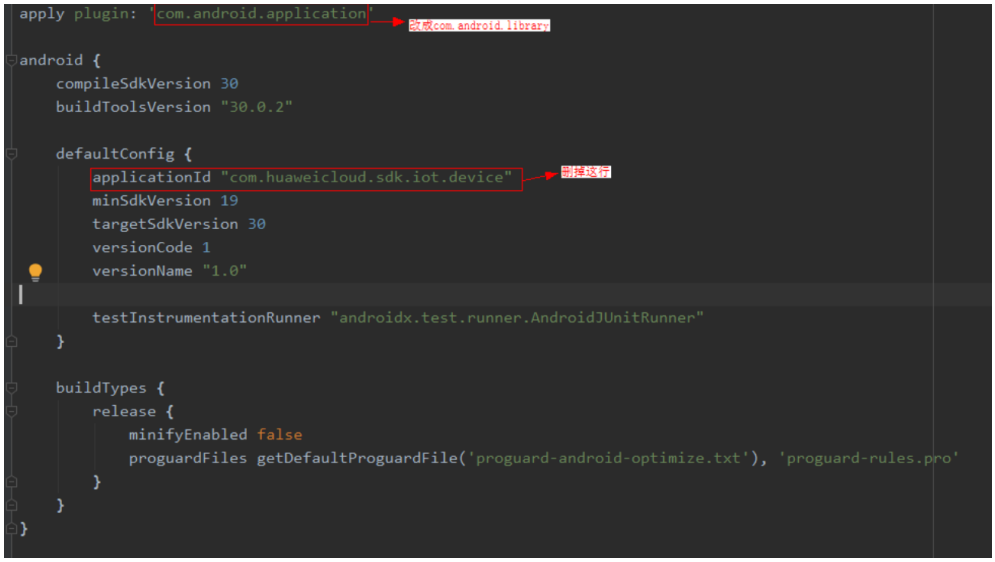
图 3-36 java 源码到复制到包下面



步骤3 配置app目录下build.gradle。

1. 在build.gradle中修改部分代码，具体修改如下。

图 3-37 Android SDK 删除配置数据



2. 在build.gradle中添加以下编译脚本。

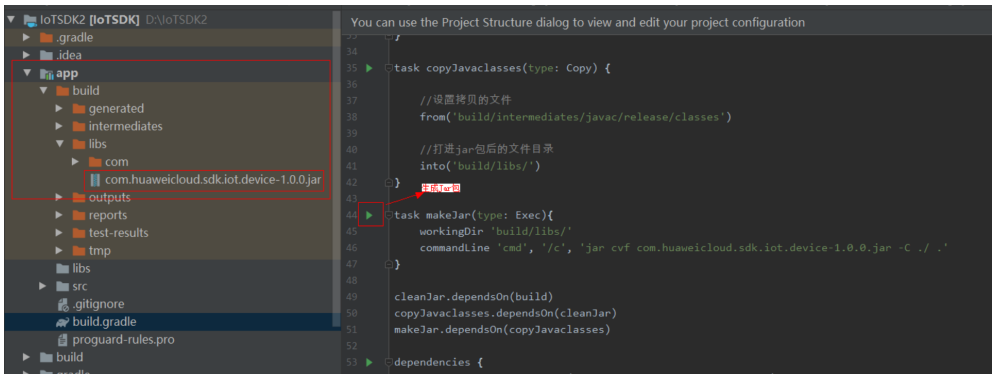
```
task cleanJar(type: Delete){
    //删除存在的
    delete 'build/libs/com.huaweicloud.sdk.iot.device-1.0.0.jar'
    delete 'build/libs/classes/'
}

task copyJavaclasses(type: Copy) {

    //设置拷贝的文件
    from('build/intermediates/javac/release/classes')

    //打进jar包后的文件目录
    into('build/libs/')
}

task makeJar(type: Exec){
    workingDir 'build/libs/'
    commandLine 'cmd', '/c', 'jar cvf com.huaweicloud.sdk.iot.device-1.0.0.jar -C ./ .'
}

cleanJar.dependsOn(build)
copyJavaclasses.dependsOn(cleanJar)
makeJar.dependsOn(copyJavaclasses)
```

3. 在build.gradle中dependencies增加以下三个依赖。

```
implementation 'com.google.code.gson:gson:2.8.6'
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.5'
implementation 'androidx.localbroadcastmanager:localbroadcastmanager:1.0.0'
```

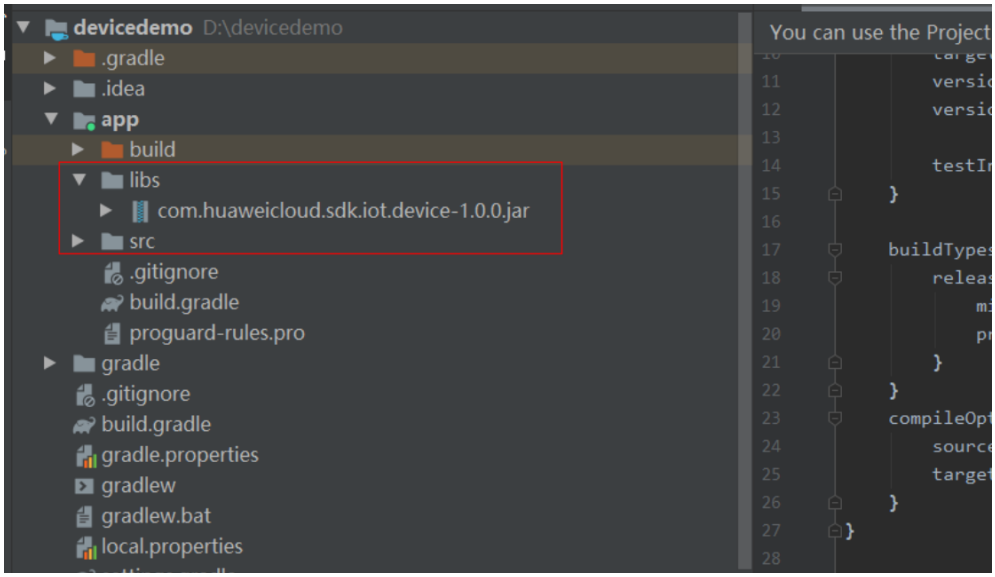
4. Android工程配置完成后，执行build.gradle中task makeJar(请确保java已经添加到环境变量)生成jar包。

图 3-38 Android SDK 生成 jar 包



5. 查看jar包位于app/build/libs目录下。

图 3-39 获取 jar 包



----结束

代码工程配置

步骤1 在工程app/libs下添加生成的Jar包。

步骤2 在build.gradle中添加以下依赖。

```
implementation fileTree(dir: "libs", include: ["*.jar"])
implementation 'androidx.localbroadcastmanager:localbroadcastmanager:1.0.0'
implementation 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.2.5'
implementation 'com.google.code.gson:gson:2.8.6'
implementation group: 'org.bouncycastle', name: 'bcpkix-jdk15to18', version: '1.68'
implementation group: 'org.bouncycastle', name: 'bcprov-jdk15to18', version: '1.68'
implementation group: 'com.squareup.okhttp3', name: 'okhttp', version: '4.9.1'
```

步骤3 在AndroidManifest.xml文件中添加以下网络权限。

```
<uses-permission android:name="android.permission.INTERNET" />
```

----结束

3.5.3 快速接入

本章将使用SmokeDetector Demo指导您快速体验产品模型的创建、Demo的配置与启动、以及SDK基本功能的使用。Demo源码下载SmokeDetector.cs。为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警以及支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

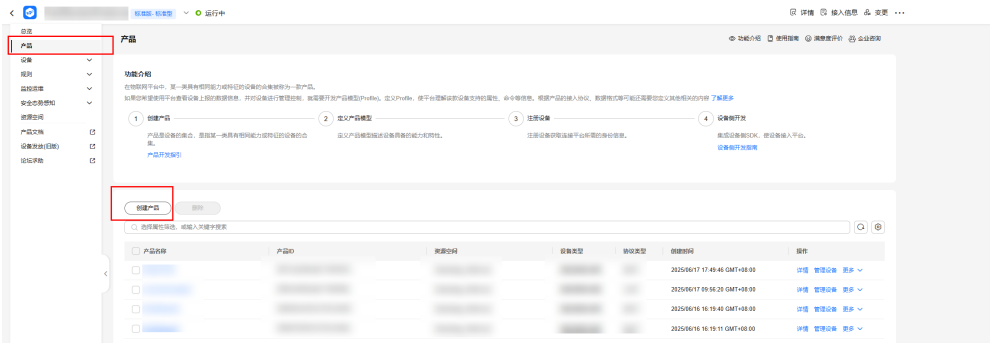
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-40 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-41 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-42 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

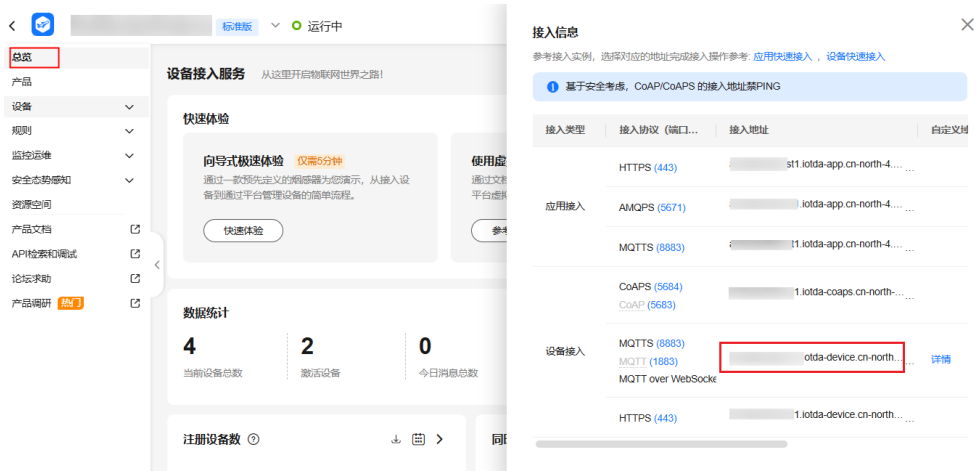
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备接入

步骤1 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-43 接入信息-设备侧 MQTT 接入地址



步骤2 将设备侧MQTTS接入地址、设备ID以及设备密钥填入对应位置。

```
// 创建一个设备并初始化
IoTDevice device = new IoTDevice(mContext, "ssl://iot-mqtts.cn-north-4.myhuaweicloud.com:8883",
"5eb4cd4049a5ab087d7d4861_demo", "secret");
device.init();
```

说明

设备接入平台时，物联网平台提供密钥和证书两种鉴权方式，如果使用MQTTS，请把下载的**bks证书**(证书重命名为DigiCertGlobalRootCA.bks)放置到src/main/assets下。

步骤3 查看设备运行情况。在控制台的设备详情页面可以查看设备已在线以及设备上报的物模型数据。

图 3-44 设备列表-设备在线

----结束

须知

SDK默认提供断线重连功能，当调用设备初始化后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。

消息上报

消息上报是指设备向平台上报消息。

1. iot-device-demo/java/com/huaweicloud/sdk/iot/device/demo/MainActivity.java是一个上报消息例子，可以通过reportDeviceMessage方法将消息上报给平台。

```
DeviceMessage deviceMessage = new DeviceMessage();
deviceMessage.setContent("content");
```

```
        deviceMessage.setId("id");
        deviceMessage.setName("name");

        Device.getDevice().getClient().reportDeviceMessage(deviceMessage);
```

2. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

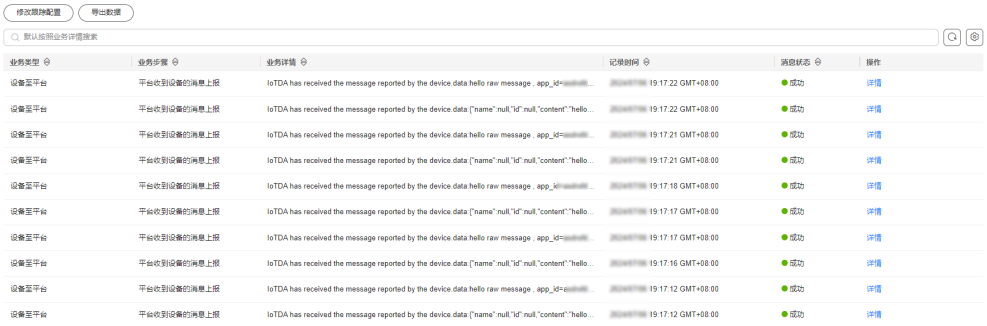
图 3-45 设备列表-设备在线

3. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

图 3-46 消息跟踪-启动消息跟踪

4. 查看消息跟踪，确认平台是否收到设备消息。

图 3-47 消息跟踪-查看 device_sdk_java 消息跟踪



须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

属性上报

1. 属性上报指的是设备将当前属性值上报给平台。代码示例可见iot-device-demo/java/com/huaweicloud/sdk/iot/device/demo/PropertyActivity.java。

```
Map<String, Object> properties = new HashMap<String, Object>();
properties.put("temperature", "28");
```

```
ServiceProperty serviceProperty = new ServiceProperty();
serviceProperty.setServiceId("smokeDetector");
serviceProperty.setProperties(properties);

//创建属性
List<ServiceProperty> serviceProperties = new ArrayList<ServiceProperty>();
serviceProperties.add(serviceProperty);
//上报属性
device.getClient().reportProperties(serviceProperties);
```

2. 运行PropertyActivity.java，查看上报结果。在左侧导航栏中选择“设备 > 所有设备”，选择对应设备在“设备影子”处可以看到上报的属性值。

图 3-48 设备影子-temperature

须知

若是没有查看到属性上报值，确认是否已上传产品模型：**上传产品模型**。

命令下发

设置命令监听器用来接收平台下发的命令，在回调接口里，将对命令进行处理，并上报响应。

1. 下面代码中messageBroadcastReceiver函数作为命令监听器，当命令下发时，会调用该函数，且Action值为IotDeviceIntent.ACTION_IOT_DEVICE_SYS_COMMANDS。

```
MessageBroadcastReceiver messageBroadcastReceiver = new MessageBroadcastReceiver();
LocalBroadcastManager.getInstance(this).registerReceiver(messageBroadcastReceiver, new
IntentFilter(IotDeviceIntent.ACTION_IOT_DEVICE_SYS_COMMANDS));
```

2. 当device收到命令时将自动调用messageBroadcastReceiver函数，messageBroadcastReceiver函数编写可见iot-device-demo/java/com/huaweicloud/sdk/iot/device/demo/MessageActivity.java。

```
private class MessageBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "onReceive: " + intent.getAction());
        if (IotDeviceIntent.ACTION_IOT_DEVICE_SYS_COMMANDS.equals(intent.getAction())) {
            requestId = intent.getStringExtra(BaseConstant.REQUEST_ID);
            Command command = intent.getParcelableExtra(BaseConstant.SYS_COMMANDS);
            edtLog.append("平台下发的命令为:" + JsonUtil.convertObject2String(command));
        }
    }
}
```

3.5.4 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与IoTDA服务（后续简出现的“服务”、“平台“均指IoTDA服务）双向通讯。SDK当前支持的功能有：

表 3-12 Android 功能支持

功能	描述说明
设备连接鉴权	作为客户端使用MQTT协议接入到平台。分为证书鉴权与密钥鉴权两种认证方式。
消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
设备影子	用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。
软固件（OTA）升级	用于与平台配合下载OTA升级包。
文件上传/下载	支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。

3.6 设备侧 IoT Device Go SDK 使用指南

3.6.1 快速引用

GO 语言安装

```
go get github.com/huaweicloud/huaweicloud-iot-device-sdk-go
```

源码获取

- 开发环境要求：Go 1.18+。
- 访问**SDK下载页面**，下载工程，工程包含以下子工程：

表 3-13 GO 语言 SDK 目录结构

目录结构	目录	说明
iot	callback	客户端回调函数
	client	设备客户端
	config	客户端配置
	constants	常量包
	device	直连设备客户端
	file	文件上传下载
	gateway	网关设备客户端
	model	结构体包
	rule	端侧规则包
sample	bs	设备发放demo
	command	命令demo
	file	文件上传下载demo
	device	直连设备客户端
	file	文件上传下载
	gateway	网关设备客户端
	log	设备日志demo
	message	消息上报下发demo
	ota	ota升级demo
	properties	设备属性demo
	rule	端侧规则demo
	test_model	测试结构体
	test_util	测试工具类
	test_sync	时间同步demo

3.6.2 使用说明

前言

huaweicloud-iot-device-sdk-go提供设备接入华为云IoT物联网平台的Go版本的SDK，提供设备和平台之间通讯能力，以及设备服务、网关服务、OTA等高级服务，并且针对各种场景提供了丰富的样例代码。IoT设备开发者使用SDK可以大大简化开发复杂

度，快速的接入平台。该SDK用于帮助设备通过MQTT协议快速连接到华为物联网平台。

版本更新说明

表 3-14 Go 语言版本更新说明

版本	变更类型	说明
v1.0.0	新增功能	提供对接华为云IoT物联网平台能力，方便用户实现安全接入、设备管理、数据采集、命令下发、设备发放、端侧规则等业务场景。

3.6.3 快速接入

本章将使用SmokeDetector样例指导您快速体验产品模型的创建、样例的配置与启动、以及SDK基本功能的使用。为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警，还支持响铃报警命令。 以烟感为例，体验消息上报、属性上报、命令响应等功能。

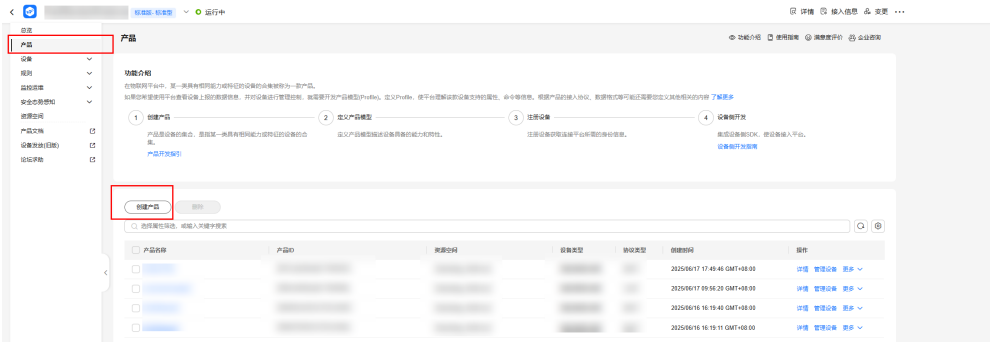
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-49 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。

产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-50 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-51 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

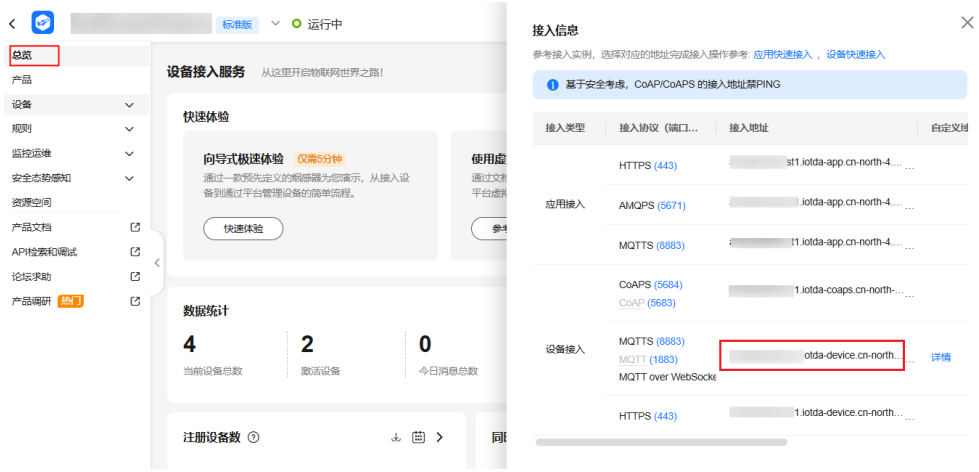
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备接入

步骤1 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-52 接入信息-设备侧 MQTT 接入地址



步骤2 打开文件samples/rule /rule_demo.go，将获取的接入地址、设备ID以及设备密钥填入对应位置。

```
// 创建一个设备并初始化
authConfig := &config2.ConnectAuthConfig{
    Id:           "your device id",
    Servers:      "mqtts://{MQTT_ACCESS_ADDRESS}:8883",
    Password:     "your password",
    ServerCaPath: "iotda server ca path",
}
authConfig.RuleEnable = true
device := device2.NewMqttDevice(authConfig)
if device == nil {
    glog.Warningf("create mqtt device failed.")
    return
}
```

说明

平台的证书可以从源码中/samples/resources/root.pem下载。

步骤3 编译及运行。运行demo后输出示例如下，如果存在connect result is : true则表示连接成功。

图 3-53 连接成功

```
I0819 10:19:13.294995   62732 connect_demo.go:25] connect result is : true
I0819 10:19:13.294995   62732 mqtt_device_client.go:352] connect from server.
```

步骤4 查看设备运行情况。在控制台的设备详情页面可以查看到设备已在线以及设备上报的物模型数据。

图 3-54 设备列表-设备在线

----结束

须知

SDK默认提供断线重连功能，当调用设备初始化后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。若是客户不需要该机制，请见：**断线重连**文档。

消息上报

消息上报是指设备向平台上报消息。

1. ./samples/message/message_demo.go是一个上报消息例子，通过SendMessage方法将消息上报给自定义topic，如果message中未填写topic，消息将会上报给平台的默认topic。

```
""" create device code here """
//向平台发送消息
message := model.Message{
    Topic:   "{custom topic}",
    Payload: "first message",
}

for {
    sendMsgResult := device.SendMessage(message)
    glog.Infof("send message %v", sendMsgResult)
    time.Sleep(1 * time.Second)
}
```

2. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

图 3-55 设备列表-设备在线

3. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

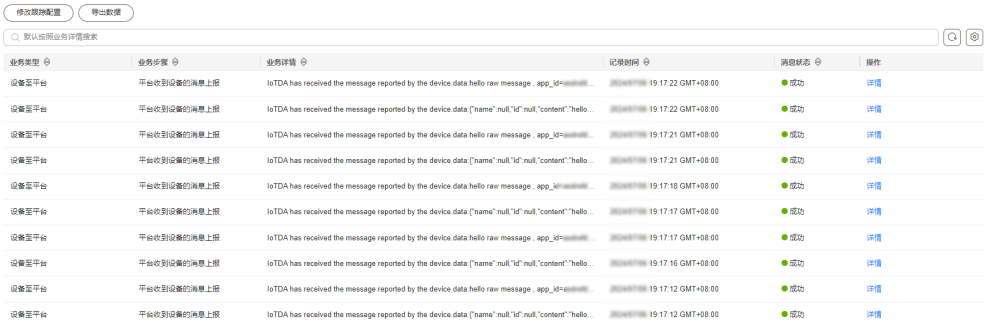
图 3-56 消息跟踪-启动消息跟踪

4. 查看消息跟踪，确认平台是否收到设备消息。

须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

图 3-57 消息跟踪-查看 device_sdk_java 消息跟踪



属性上报

属性上报指的是设备将当前属性值上报给平台。

1. ./samples/properties/device_properties.go是一个属性上报/设置的例子，可以通过ReportProperties方法将属性上报给平台。

```
< create device code here ... >
   // 设备上报属性
props := model.DevicePropertyEntry{
   ServiceId: "smokeDetector",
   EventTime: iot.GetEventTimeStamp(),
   Properties: test_model.DemoProperties{
      Temperature: 28,
   },
}

var content []model.DevicePropertyEntry
content = append(content, props)
services := model.DeviceProperties{
   Services: content,
}
device.ReportProperties(services)
```

2. 在左侧导航栏中选择“设备 > 所有设备”，选择注册的设备进行查看，在“设备影子”处可以看到刚刚上报的属性值。

图 3-58 设备影子-temperature

说明

若是没有查看到属性上报值，请查看是否进行了：**上传产品模型**。

命令下发

设置命令监听器用来接收平台下发的命令，在回调接口里，将对命令进行处理，并上报响应。

1. ./samples/command/platform_command.go是一个处理平台命令下发的例子。使用CommandHandler函数作为命令监听器，即：

```
mqttDevice.Client.AddCommandHandler(func(command model.Command) (bool, interface{}) {
    glog.Infof("I get command from platform")
    glog.Infof("command device id is %s", command.ObjectDeviceId)
    glog.Infof("command name is %s", command.CommandName)
    glog.Infof("command serviceId is %s", command.ServiceId)
    glog.Infof("command params is %v", command.Paras)
    return true, map[string]interface{}{
        "cost_time": 12,
    }
})
```

2. 当device收到命令时将自动调用CommandHandler函数，并将函数响应结果返回给平台。
3. 执行main方法，在平台给设备下发命令，查看代码日志如下。

图 3-59 go 语言命令下发

3.6.4 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与IoTDA服务（后续简出现的“服务”、“平台“均指IoTDA服务）双向通讯。SDK当前支持的功能有：

表 3-15 go 功能支持

功能	描述说明
设备连接鉴权	作为客户端使用MQTT协议接入到平台。分为证书鉴权与密钥鉴权两种认证方式。
断线重连	当设备由于网络不稳定导致连接失败或中断时，会自动进行重新连接。
消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
设备影子	用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。
软固件（OTA）升级	用于与平台配合下载OTA升级包。

功能	描述说明
时间同步	设备向平台发起时间同步请求。
网关与子设备	网关设备：通过平台支持的协议，直接连接到平台的设备。子设备：针对未实现TCP/IP协议栈的设备，由于无法直接同平台通信，它需要通过网关进行数据转发。当前仅支持通过mqtt协议直连到平台的设备作为网关设备。
文件上传/下载	支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。
规则引擎	通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前平台支持两种联动规则：云端规则和端侧规则。
设备发放	分为证书认证、密钥认证。主要用于分发到不同局点、实例的场景，动态完成不同批次设备初始化配置。

3.7 设备侧 IoT Device C for SDK Tiny 使用指南

3.7.1 使用说明

前言

- IoT Device SDK Tiny是部署在具备广域网能力、对功耗、存储、计算资源有苛刻限制的终端设备上的轻量级的互联互通中间件，您只需调用API接口，便可实现设备快速接入到物联网平台以及数据上报和命令接收等功能。
- IoT Device SDK Tiny提供端云协同能力，集成了MQTT、LWM2M、CoAP、mbedtls、LwIP全套IoT互联互通协议栈，且在这些协议栈的基础之上，提供了开放API，用户只需关注自身的应用，而不必关注协议内部实现细节，直接使用SDK封装的API，简单快速的实现与华为云IoT平台的安全可靠连接。使用SDK可以大大减少开发周期，聚焦业务开发，快速构建产品。
- 同时该SDK还具有可裁剪特性，在移植过程中可以根据需求进行定制化组件，节省内存空间，减小移植难度。
- 相关集成指导请参见**端云互通组件开发指南**。

说明

IoT Device SDK Tiny可以运行于无linux操作系统的设备，也可以被模组集成，但是不提供网关服务。

源码目录说明

表 3-16

一级目录	二级目录	三级目录	说明
LiteOS_Lab	at	-	AT指令框架实现
	cJSON	-	cJSON
	crc	-	crc校验
	demos	-	示例
	driver	-	驱动框架
	docs	-	存放使用文档及API说明等文档
	fs	-	文件系统，含VFS、SPIffs、RAMfs、KIfs、DEVfs、FATfs
	inc	-	存放内核内部使用头文件
	link_log	-	日志
	link_misc	-	杂项
	link_ota	-	OTA升级代码实现
	network	coap	CoAP的适配及协议实现
		dtls	mbedtls的适配及协议实现
		lwm2m	LwM2M的开源协议栈wakaama的适配
		mqtt	MQTT的适配及协议实现
		tcpip	TCPIP适配及协议栈实现、lwIP驱动、OS适配及协议栈实现、MacOS_socket适配及协议栈实现
	oc	oc_coap	CoAP协议适配华为云物联网平台
		oc_lwm2m	LwM2M协议适配华为华为云物联网平台
		oc_mqtt	MQTT协议适配华为华为云物联网平台
	os	osal	IoT Device SDK Tiny的OS适配

一级目录	二级目录	三级目录	说明
		xxos	包括freertos、linux、liteos、macos、novaos、ucos_ii等操作系统的适配
	queue	-	queue组件的代码实现
	secure_c	-	C安全函数库
	shell	-	shell组件代码实现
	stimer	-	stimer组件代码实现
	storage	-	存储分区
	usip	-	usip协议
	iot.mk	-	Makefile
	iot_config.h	-	宏定义
	iot_link_config.h	-	-
	link_main.c	-	SDK初始化入口
	README.md	-	SDK 简介

3.7.2 通信协议选择与适配

Tiny中支持MQTT、LWM2M、CoAP、mbedtls、LwIP全套IoT互联互通协议栈，以下主要介绍MQTT、LWM2M、CoAP这三种协议。

表 3-17 支持协议

术语名称	描述
MQTT	MQTT（Message Queue Telemetry Transport）是一个物联网传输协议，被设计用于轻量级的发布/订阅式消息传输，旨在为低带宽和不稳定的网络环境中的物联网设备提供可靠的网络服务。MQTTS指MQTT+SSL/TLS，在MQTTS中使用SSL/TLS协议进行加密传输。
CoAP	受约束的应用协议CoAP（Constrained Application Protocol）是一种软件协议，旨在使非常简单的电子设备能够在互联网上进行交互式通信。CoAPS指CoAP over DTLS，在CoAPS中使用DTLS协议进行加密传输。
LwM2M	LwM2M（lightweight Machine to Machine）是由OMA（Open Mobile Alliance)定义的物联网协议，主要使用在资源受限(包括存储、功耗等)的NB-IoT终端。
mbedtls	mbedTLS（前身PolarSSL）是面向嵌入式系统，实现的一套易用的加解密算法和SSL/TLS库。mbedTLS系统开销极小，对于系统资源要求不高。mbedTLS是开源项目，并且使用Apache 2.0许可证，使得用户既可以将mbedTLS使用在开源项目中，也可以应用于商业项目。目前使用mbedTLS的项目很多，例如Monkey HTTP Daemon，LinkSYS路由器。

云服务抽象层 API

端云互通组件是IoT Device SDK Tiny中的重要组件，提供端云协同能力和基于MQTT、LwM2M、CoAP协议对接华为云的开放API。

支持MQTT/ LwM2M/CoAP协议、MQTT/ LwM2M协议模组或内置对接华为云模组来对接华为云的开发者，无需关注对接华为云物联网协议的实现细节，仅仅需要将所选择的物联网协议注册到云服务抽象层（OC AL），再调用OC AL提供的抽象层接口即可。

- OC-MQTT

  OC AL提供的基于MQTT协议抽象层接口代码目录为：…\iot_link\oc\oc_mqtt，OC AL提供的MQTT协议端云互通API请参考最新的oc_mqtt_al.h文件，API接口具体如下：

目录	描述
…/oc_mqtt_al	适配华为OC服务的MQTT抽象层。通过注册机制，最终调用到用户注册的函数
…/oc_mqtt_tiny_v5	SDK内置的soft类设备适配华为OC服务的MQTT的具体实现(针对云端V5版本接口)。
…/oc_mqtt_profile_v5	SDK内置的基于物模型的接口，解析了topic，定义了相关的数据格式

- OC-LwM2M

  OC AL提供的基于LwM2M协议抽象层接口代码目录为：…\iot_link\oc\oc_lwm2m， OC AL提供的基于LwM2M协议端云互通API参考oc_lwm2m_al.h文件，API接口具体如下：

目录	描述
…/oc_lwm2m_al	华为云服务OC LwM2M的抽象层接口及实现。通过注册，最终调用底层的适配函数
…/atiny_lwm2m	SDK内置的soft类设备适配华为OC服务的LwM2M的具体实现。通过抽象层提供的oc_lwm2m_register注册函数，将相关功能注册到SDK中
…/oc_lwm2m_imp/boudica150	SDK内置的boudica150适配华为OC服务的LwM2M的具体实现。通过抽象层提供的oc_lwm2m_register注册函数，将相关功能注册到SDK中

- OC-CoAP

  OC AL提供的基于CoAP协议抽象层接口代码目录为：…\iot_link\oc\oc_coap，OC AL提供的基于CoAP协议端云互通API参考oc_coap_al.h文件，API接口具体如下：

目录	描述
…/oc_coap_al	华为云服务OC LwM2M的抽象层接口及实现。通过注册，最终调用底层的适配函数
…/atiny_coap	SDK内置的soft类设备适配华为OC服务的CoAP的具体实现。通过抽象层提供的oc_coap_register注册函数，将相关功能注册到SDK中

物联组件协议层

IoT Device SDK Tiny集成了LwM2M、CoAP、MQTT等物联网标准协议，您可以直接调用已实现协议，或添加自定义协议，将其注册进SDK中。其协议层目录为：…\network\。

1. MQTT AL

   根据MQTT标准协议，IoT Device SDK Tiny提供的MQTT服务都是建立在标准的MQTT协议基础上，并提供MQTT协议抽象层接口。适配MQTT的软件结构示意图如下：

```
typedef struct
{
  ///< connect to the server
  void* (* connect)    (mqtt_al_conpara_t *param);
  ///< disconnect from the server
  int     (* disconnect)(void *handle );
  ///< publish a message to the server
  int       (* publish)     (void *handle, mqtt_al_pubpara_t *msg);
  ///< subscribe a topic to the server
```

```
  int    (* subscribe)   (void *handle, mqtt_al_subpara_t *subpara);
  ///< unsubscribe a topic to the server
  int    (* unsubscribe) (void *handle, mqtt_al_unsubpara_t *unsubpara);
  ///< check the mqtt engine status
  en_mqtt_al_connect_state (* check_status) (void *handle);
}mqtt_al_op_t;
int mqtt_al_install(mqtt_al_op_t *op);
```

表 3-18 MQTT 抽象层接口

接口分类	接口名	说明
MQTT相关接口	void * mqtt_al_connect(mqtt_al_conpara_t *conparam)	请求连接
	int mqtt_al_disconnect(void *handle)	断开连接
	int mqtt_al_publish(void *handle, mqtt_al_pubpara_t *pubpara)	发布消息
	int mqtt_al_subscribe(void *handle, mqtt_al_subpara_t *subpara)	订阅请求
	int mqtt_al_unsubscribe(void *handle, mqtt_al_unsubpara_t *unsubpara)	取消订阅
	en_mqtt_al_connect_state mqtt_al_check_status(void *handle)	检查连接状态

2. LWM2M AL

   根据LwM2M标准协议，IoT Device SDK Tiny提供的LwM2M服务都是建立在标准的LwM2M协议基础上，并提供LwM2M协议抽象层接口。适配LwM2M的软件结构示意图如下：

```
typedef struct
{
    /* lwm2m config, prepare endpoint name and message deal callback */
    int (*config)(void **handle, lwm2m_al_init_param_t *init_param);

    /* lwm2m deinit */
    int (*deconfig)(void *handle);

    /* lwm2m add object */
    int (*add_object)(void *handle, int object_id, int object_instance_id, int resource_id, void *param);

    /* lwm2m delete object */
    int (*delete_object)(void *handle, int object_id);

    /* lwm2m connect */
    int (*connect)(void *handle);

    /* lwm2m disconnect */
    int (*disconnect)(void *handle);
```

```
    /* lwm2m send */
    int (*send)(void *handle, lwm2m_al_send_param_t *send_param);
} lwm2m_al_op_t;
```

表 3-19 LWM2M 抽象层接口

接口分类	接口名	说明
LWM2M相关接口	int (*config)(void **handle, lwm2m_al_init_param_t *init_param);	配置初始化
	int (*deconfig)(void *handle);	取消初始化
	int (*add_object)(void *handle, int object_id, int object_instance_id, int resource_id, void *param);	添加对象
	int (*delete_object)(void *handle, int object_id);	删除对象
	int (*connect)(void *handle);	请求连接
	int (*disconnect)(void *handle);	断开连接
	int (*send)(void *handle, lwm2m_al_send_param_t *send_param);	发送

3. COAP AL

   根据COAP标准协议，IoT Device SDK Tiny提供的COAP服务都是建立在标准的COAP协议基础上，并提供COAP协议抽象层接口。适配CoAP的软件结构示意图如下：

```
typedef struct
{
    ///< coap init, prepare context, session, etc
    int     (* init)      (coap_al_initpara_t *initparam);
    ///< coap deinit
    int     (* deinit)    (void *handle);
    ///< coap add option
    void*   (* add_opt)   (coap_al_optpara_t *optparam);
    ///< new coap request
    void*   (* request)   (coap_al_reqpara_t *reqparam);
    ///< send a request to server
    int       (* send)      (coap_al_sndpara_t *sndparam);
    ///< recv and handle response from server
    int     (* recv)      (coap_al_rcvpara_t *rcvparam);
}coap_al_op_t;
int coap_al_install(coap_al_op_t *op);
```

   相关适配接口介绍：

接口分类	接口名	说明
CoAP相关接口	int coap_al_init(coap_al_initpara_t *initparam);	初始化

接口分类	接口名	说明
	int coap_al_deinit(void *handle);	取消初始化
	void *coap_al_add_option(coap_al_optpara_t *optparam);	添加CoAP选项
	void *coap_al_new_request(coap_al_reqpara_t *reqparam);	创建新的消息请求
	int coap_al_send(coap_al_sndpara_t *sndparam);	发送消息给服务器
	int coap_al_recv(coap_al_rcvpara_t *rcvparam);	接收并处理服务器下发的数据
	int coap_al_install(coap_al_op_t *op);	将协议栈注册到系统中
	int coap_al_uninstall();	取消注册

3.7.3 OS 系统选择与适配

Tiny中提供了LiteOS、NovaOS、linux、ucos_ii、macOS、FreeRTOS等多操作系统的适配，您也可以根据需要去适配自己的操作系统。

os 适配

IoT Device SDK Tiny本身是不局限于某一款具体的物联网操作系统的，但是其运行是依赖于操作系统的，因此需要进行OS的相关适配。用户可以根据模组、MCU厂商的需求使用其内置OS，适配IoT Device SDK Tiny实现对接华为云等进行一系列的云服务的使用。目前IoT Device SDK Tiny已经适配了多款OS,包括LiteOS、FreeRTOS、MacOS、Linux、NovaOS、ucos_ii等。

IoT Device SDK Tiny提供了osal（操作系统抽象层），实现了操作系统与SDK组件的隔离。因此厂商可以直接将想要使用的操作系统注册到osal中，即可实现对SDK组件的使用。其中包括与操作系统内核强相关的任务操作、信号量、互斥锁、内存管理接口等内容。在适配过程中可以参考已经适配好的操作系统imp文件。由于本SDK的队列queue采用互斥锁和信号量加以实现，因此无需对队列进行适配。最终调用顺序如下osal_init() ---> os_imp_init() ---> osal_install()，其中os_imp_init是弱符号函数，需要用户根据自身需要进行实现，在其中调用osal_install()函数实现系统服务的注册。

TCP/IP 适配

对于IoT Device SDK Tiny而言，其网络协议栈决定于底层操作系统，在使用三方操作系统时，可以将相关的网络传输接口进行抽象，提供相关的协议栈接口即可。IoT Device SDK Tiny中调用的所有网络传输的接口，通过注册机制，最终都会调用到用户注册的函数。

IoT Device SDK Tiny也提供了抽象层sal，用户可以调用link_sal_install函数进行TCP/IP功能注册，同时用户必须实现link_tcpip_imp_init函数，该函数是一个弱符号函数。在该SDK初始化的过程中，调用顺序为：link_tcp_ip_init > link_tcpip_imp_init >

link_sal_install，在初始化完毕之后，才可以使用TCP/IP相关的功能。详细参考sal_imp.h头文件，同时已经适配了LWIP、ESP8266\L716等软件或者模组提供的TCP/IP功能，如果您有第三方的组件或者模组，可以参考实现。

适配中的细节问题

- 不同的操作系统对于任务优先级的规定是不同的，因此IoT Device SDK Tiny中用到的创建任务的函数调用中赋予的优先级要进行相应的改变，目前本SDK默认优先级顺序为优先级数在0-31，0为最高优先级，31为最低优先级。
- 有些编译架构对于虚引用函数的支持以及编译方式可能略有不同，需要根据需求进行自行修改。
- 对于上述组件，如果本身的SDK中已经具备相应的组件可以根据需求进行适配、裁剪等操作。
- 编译架构的不同，可以保留源文件以及库文件，根据自己的需求进行移植适配以及使能，下面仅仅给个iot_config.h文件的参考示例，示例中以对接接口以V5版本接口为例。这里可以重点关注SDK的入口函数link_main.c，其中包含了如何通过宏定义控制哪些文件参与编译。

```
//enable cJSON
#define CONFIG_CJSON_ENABLE 1

//enable LinkLog
#define CONFIG_LINKLOG_ENABLE 1

//enable queue
#define CONFIG_LINKQUEUE_ENABLE 1

//enable DTLS_AL link_main
#define CONFIG_DTLS_AL_ENABLE 1
#define CONFIG_MBEDTLS_ENABLE 1
#define CONFIG_MBEDTLS_PSK 1


// enable MQTT link_main
#define CONFIG_PAHO_MQTT 1
#define CONFIG_MQTT_AL_ENABLE 1

// enable tcpip
#define CONFIG_TCPIP_AL_ENABLE 1

//choose paho_mqtt or lite_mqtt
//#define CONFIG_PAHO_MQTT 1


//paho_mqtt_port.c use them
#define CONFIG_PAHO_CONNECT_TIMEOUT 10000
#define CONFIG_PAHO_CMD_TIMEOUT 10000
#define CONFIG_PAHO_LOOPTIMEOUT 10
#define CONFIG_PAHO_SNDBUF_SIZE 2048
#define CONFIG_PAHO_RCVBUF_SIZE 2048

//enable OC_MQTT link_main
#define CONFIG_OCMQTT_ENABLE 1

//mqtt the profile function tools and implement component oc_mqtt.mk
#define CONFIG_OC_TINYMQTTV5_ENABLE 1
#define CONFIG_OC_MQTTV5_PROFILE 1

#define CONFIG_IOT_LINK_CONFIGFILE "iot_config.h"
```

3.8 设备侧 IoT Device Python SDK 使用指南

3.8.1 使用说明

前言

iot-device-sdk-python（以下简称SDK）提供设备接入华为云IoT物联网平台的Python版本的SDK，提供设备和平台之间通讯能力， 以及设备服务、网关服务、OTA等高级服务，并且针对各种场景提供了丰富的demo代码。 IoT设备开发者使用SDK可以大大简化开发复杂度，快速的接入平台。本文通过示例讲述SDK帮助设备用MQTT协议快速连接到华为物联网平台。相关集成指导请参考**设备侧 IoT Device Python SDK使用指南**。

准备工作

- 已安装Python 3.11.4。
- 已安装第三方类库paho-mqtt：2.0.0 。
- 已安装第三方类库schedule: 1.2.2 。
- 已安装第三方类库apscheduler: 3.10.4 。
- 已安装第三方类库requests: 2.32.2 （可选，在网关与子设备管理demo演示中使用）。
- 已安装第三方类库tornado: 6.3.3 （可选，在网关与子设备管理demo演示中使用）。

说明

可以直接运行requirement/install_requirements.py进行依赖安装。

版本更新说明

表 3-20 python 语言版本更新说明

版本号	变更类型	功能描述说明
1.2.0	新增功能	增加规则引擎、设备发放功能、自定义断线重连功能、升级组件版本。
1.1.4	新增功能	OTA升级支持网关模式。
1.1.3	功能增强	更新服务端ca证书。
1.1.2	新增功能	增加micropython支持和对应demo，从OBS下载OTA，以及说明文档。
1.1.1	新增功能	提供对接华为云IoT物联网平台能力，方便用户实现安全接入、设备管理、数据采集、命令下发等业务场景。

3.8.2 快速接入

本章将使用SmokeDetector样例指导您快速体验产品模型的创建、样例代码的配置与启动、以及SDK基本功能的使用。为了方便体验，提供了一个烟感的产品模型，烟感

会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。 以烟感为例，体验消息上报、属性上报、命令响应等功能。

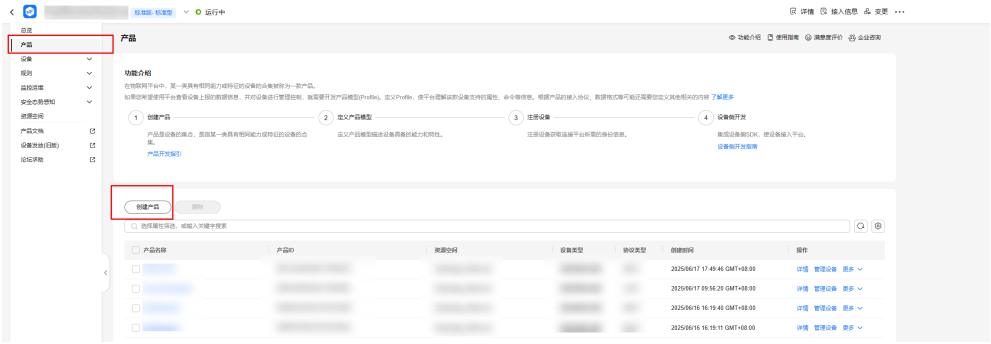
创建产品

为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感为例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

图 3-60 产品-创建产品



步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

图 3-61 创建产品-MQTT

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-62 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

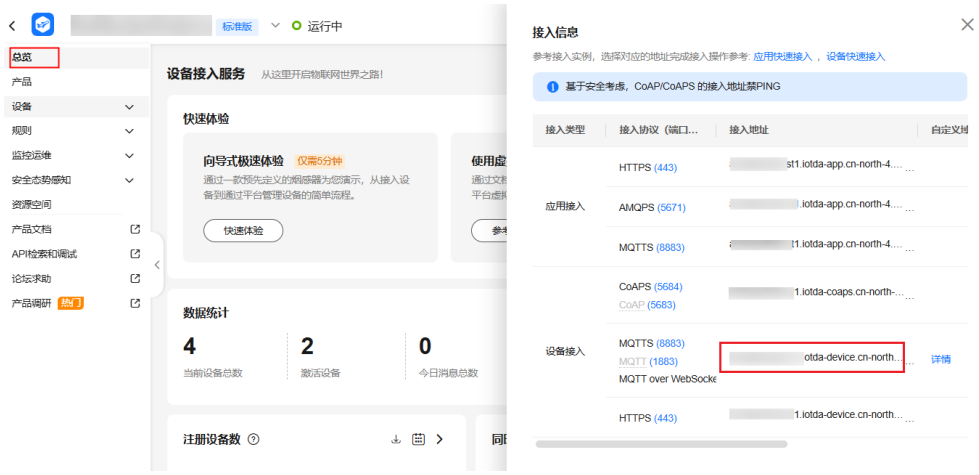
设备注册成功后保存设备标识码、设备ID、密钥。

----结束

设备接入

步骤1 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-63 接入信息-设备侧 MQTT 接入地址



步骤2 在iot_device_demo/device/smoke_detector.py中将获取的接入地址、设备ID以及设备密钥填入对应位置。

```
server_uri = "access address"  # 需要改为用户保存的接入地址
port = 8883
device_id = "your device id"
secret = "your device secret"
# iot平台的CA证书，用于服务端校验
iot_ca_cert_path = "./resources/root.pem"

connect_auth_info = ConnectAuthInfo()
connect_auth_info.server_uri = server_uri
connect_auth_info.port = port
connect_auth_info.id = device_id
connect_auth_info.secret = secret
connect_auth_info.iot_cert_path = iot_ca_cert_path
client_conf = ClientConf(connect_auth_info)

device = IotDevice(client_conf)
```

说明

平台的证书可以从源码中**/iot_device_demo/resources/root.pem**下载。

步骤3 编译&运行。执行样例，输出如下，如果包含connect result is : true则表示连接成功。

图 3-64 连接成功

```
I0819 10:19:13.294995   62732 connect_demo.go:25] connect result is : true
I0819 10:19:13.294995   62732 mqtt_device_client.go:352] connect from server.
```

步骤4 查看设备运行情况。在控制台的设备详情页面可以查看设备已在线以及设备上报的物模型数据。

图 3-65 设备列表-设备在线

图 3-66 查看上报数据-smokeDetector

----结束

须知

SDK默认提供断线重连功能，当调用设备初始化后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。若是客户不需要该机制，请见：**断线重连**。

消息上报

消息上报是指设备向平台上报消息。

1. iot_device_demo/message/message_delivery_sample.py是一个上报消息例子，可以调用report_device_message方法将消息通过默认topic上报给平台，也可以调用publish_raw_message方法将消息通过自定义topic上报给平台。

```
    """ create device code here """

    default_publish_listener = DefaultPublishActionListener()
    device_message = DeviceMessage()
    device_message.content = "Hello Huawei"

    device_message2 = DeviceMessage()
    device_message2.content = "Custom topic Message"

    device_message3 = DeviceMessage()
    device_message3.content = "Custom Policy topic Message"
    # 定时上报消息
    while True:
        # 通过平台默认topic上报消息
        device.get_client().report_device_message(device_message,
                                                  default_publish_listener)
        time.sleep(1)

        payload = json.dumps(device_message2.to_dict())
        # 通过平台自定义topic上报消息
        device.get_client().publish_raw_message(RawMessage(custom_topic, payload, 1),
default_publish_listener)
        time.sleep(1)

        payload = json.dumps(device_message3.to_dict())
        # 通过平台自定义策略中的topic上报消息
        device.get_client().publish_raw_message(RawMessage(custom_policy_topic, payload, 1),
default_publish_listener)
        time.sleep(5)
```

2. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

图 3-67 设备列表-设备在线

3. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

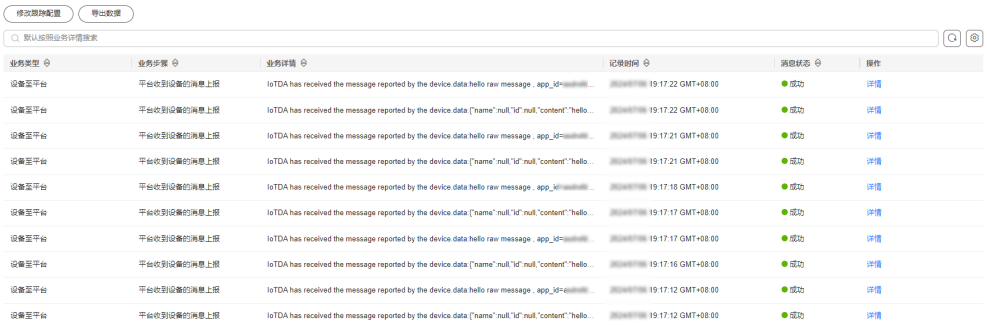
图 3-68 消息跟踪-启动消息跟踪

4. 查看消息跟踪，确认平台是否收到设备消息。

须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

图 3-69 消息跟踪-查看 device_sdk_java 消息跟踪



属性上报

属性上报指的是设备将当前属性值上报给平台。

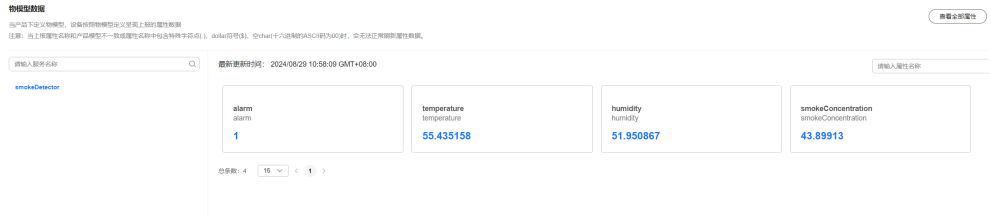
1. ./iot_device_demo/device/properties_sample.py是一个属性上报的例子，可以通过report_properties方法将属性上报给平台。

```
def run():
    < create device code here ... >

    if device.connect() != 0:
        logger.error('init failed')
        return
```

```
    service_property = ServiceProperty()
    service_property.service_id = 'smokeDetector'
    service_property.properties = {'alarm': 10, 'smokeConcentration': 36, 'temperature': 64, 'humidity':
32}
    services = [service_property.to_dict()]

    while True:
        device.get_client().report_properties(services, DefaultPublishActionListener())
        time.sleep(5)
```

2. 例子中alarm、smokeConcentration、temperature、humidity这四个属性将周期性地上报给平台。在左侧导航栏中选择“设备 > 所有设备”，设备列表中选择对应设备单击“详情”，可以查看设备上报的物模型数据。

图 3-70 物模型-属性上报



须知

若是没有查看到属性上报值，确认是否已上传产品模型：**上传产品模型**。

命令下发

设置命令监听器用来接收平台下发的命令，在回调接口里，将对命令进行处理，并上报响应。

1. ./samples/command/platform_command.go是一个处理平台命令下发的例子。代码中将CommandSampleListener的实例设置为命令监听器，CommandSampleListener类继承CommandListener类，实现了on_command方法。

```
device.get_client().set_command_listener(CommandSampleListener(device))
```

2. device收到命令时将自动调用监听器中的on_command方法。

```
class CommandSampleListener(CommandListener):
    def __init__(self, iot_device: IotDevice):
        """ 传入一个IotDevice实例 """
        self.device = iot_device

    def on_command(self, request_id, service_id, command_name, paras):
        logger.info('on_command requestId: ' + request_id)
        # 处理命令
        logger.info('begin to handle command')

        """ code here """
        logger.info(str(paras))

        # 命令响应
        command_rsp = CommandRsp()
        command_rsp.result_code = 0
        command_rsp.response_name = command_name
        command_rsp.paras = {"content": "Hello Huawei"}
        self.device.get_client().response_command(request_id, command_rsp)
```

```
def run():

    < create device code here ... >

    # 设置监听器
    device.get_client().set_command_listener(CommandSampleListener(device))

    if device.connect() != 0:
        logger.error('init failed')
        return

    while True:
        time.sleep(5)
```

3. 执行代码后，在平台给设备下发命令，查看代码日志如下。

图 3-71 Python 语言命令下发

3.8.3 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与IoTDA服务（后续出现的“服务”、“平台“均指IoTDA服务）双向通讯。 SDK当前支持的功能有：

表 3-21 Python 功能支持

功能	描述说明
设备连接鉴权	作为客户端使用MQTT协议接入到平台。分为证书鉴权与密钥鉴权两种认证方式。
断线重连	当设备由于网络不稳定导致连接失败或中断时，会自动进行重新连接。
消息上报	用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理。
属性上报	用于设备按产品模型中定义的格式将属性数据上报给平台。
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
设备影子	用于存储设备的在线状态、设备最近一次上报的设备属性值、应用服务器期望下发的配置。
软固件（OTA）升级	用于与平台配合下载OTA升级包。
时间同步	设备向平台发起时间同步请求。
网关与子设备	网关设备：通过平台支持的协议，直接连接到平台的设备。子设备：针对未实现TCP/IP协议栈的设备，由于无法直接同平台通信，它需要通过网关进行数据转发。当前仅支持通过mqtt协议直连到平台的设备作为网关设备。

功能	描述说明
文件上传/下载	支持设备将运行日志，配置信息等文件上传至平台，便于用户进行日志分析、故障定位、设备数据备份等。
规则引擎	通过条件触发，基于预设的规则，引发多设备的协同反应，实现设备联动、智能控制。目前平台支持两种联动规则：云端规则和端侧规则。
设备发放	分为证书认证、密钥认证。主要用于分发到不同局点、实例的场景，动态完成不同批次设备初始化配置。

3.9 设备侧 IoT Device Arkts(OpenHarmony) SDK 使用指南

3.9.1 使用说明

前言

设备侧 IoT Device Arkts(OpenHarmony) SDK提供设备接入华为云IoT物联网平台的ArkTS版本的SDK，提供设备和平台之间通讯能力，并且针对各种场景提供了丰富的demo代码。

准备工作

已安装 DevEco Studio 5.0.0及以上版本。

- 点此查看 DevEco Studio 安装指导
- 点此下载 DevEco Studio

使用说明

- 下载安装：在DevEco Studio中执行以下命令引入并安装SDK。
  ```
  ohpm install @huaweicloud/iot-device-sdk
  ```
- 权限配置：使用SDK需要网络连接的权限，需要在module.json5的requestPermissions中增加ohos.permission.INTERNET的权限，如下所示：
  ```
  {
    "module": {
      "requestPermissions": [
        {
          "name": "ohos.permission.INTERNET"
        }
      ]
    }
  }
  ```

版本更新说明

表 3-22 openHarmony Arkts 语言版本更新说明

版本号	变更类型	功能描述说明
0.0.1	新增功能	提供对接华为云物联网平台能力，方便用户实现接入、设备管理、命令下发等业务场景

3.9.2 快速接入

本章将使用SmokeDetector Demo 指导您快速体验产品模型的创建、Demo的配置与启动、以及SDK基本功能的使用。为了方便体验，提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。 以烟感例，体验消息上报、属性上报、命令响应等功能。

创建产品

为了方便体验，我们提供了一个烟感的产品模型，烟感会上报烟雾值、温度、湿度、烟雾报警、还支持响铃报警命令。以烟感例，体验消息上报、属性上报等功能。

步骤1 访问**设备接入服务**，单击“管理控制台”进入设备接入控制台，选择您的实例，单击实例卡片进入。查看MQTTS设备接入域名，保存该地址。

步骤2 单击左侧导航栏“产品”，单击页面左侧的“创建产品”。

步骤3 根据页面提示填写参数，然后单击“确定”完成产品的创建。

基本信息	
所属资源空间	平台自动将新创建的产品归属在默认资源空间下。如需归属在其他资源空间下，下拉选择所属的资源空间。如无对应的资源空间，请先创建**资源空间**。
产品名称	自定义。支持字母、数字、下划线（_）、连字符（-）的字符组合。
协议类型	选择“MQTT”。
数据格式	选择“JSON”。
设备类型选择	选择”自定义类型”
设备类型	填写“smokeDetector”
高级配置	
产品ID	不填写
产品描述	请根据实际情况填写。

----结束

上传产品模型

步骤1 单击下载烟感产品模型**smokeDetector**，获取产品模型文件。

步骤2 找到创建的产品，单击产品进入产品详情页。

步骤3 选择“基本信息”页签，单击“上传模型文件”，上传**1**获取的产品模型文件。

图 3-72 产品-上传产品模型

----结束

注册设备

步骤1 选择左侧导航栏“设备 > 所有设备”，单击“注册设备”。

步骤2 根据页面提示信息填写参数，然后单击“确定”。

参数名称	说明
所属资源空间	确保和创建的产品归属在同一个资源空间。
所属产品	选择创建的产品。
设备标识码	即nodeID，设备唯一物理标识。可自定义，由英文字母和数字组成。
设备名称	即device_name，可自定义。
设备认证类型	选择“密钥”。
密钥	设备密钥，可自定义。若不填写密钥，物联网平台会自动生成密钥。

设备注册成功后保存设备标识码、设备ID、密钥。

----结束

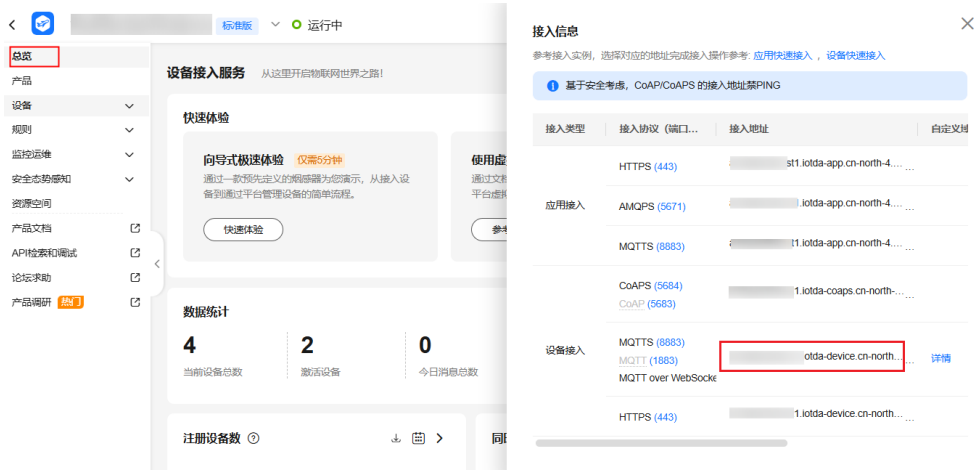
设备初始化

说明

设备初始化Demo，可参考**entry/src/main/ets/pages/Index.ets**。

1. 获取接入地址。可在控制台的“总览 > 接入信息”中查看。

图 3-73 接入信息-设备侧 MQTT 接入地址



2. 创建设备需要输入获取的设备ID、密码以及设备MQTT接入地址，注意格式为 ***ssl://域名信息:端口号*** 或 ***ssl://IP地址:端口号***，填入初始化方法内。

```
private device: IoTDevice | null = null;

// 用户请替换为自己的接入地址, 设备ID，设备密钥及证书路径（证书文件放在resource/resfile下，连接华为云时请使用对应的证书，可以在资源获取中下载证书文件）。
this.device = new IoTDevice("ssl://域名信息:8883","deviceId","mySecret","filePath");
```

3. 调用init建立连接。您可以使用异步或同步方式初始化。

```
// 使用异步方式初始化
this.device.init().then((data: boolean) => {
  // 连接成功处理
}).catch((err: string) => {
  // 连接失败处理
})

// 或使用同步方式初始化
// await this.device.init();
```

4. 查看设备日志打印，设备连接成功。

```
IoTDA_SDK# connect result is {"code":0,"message":"Connect Success"}
```

5. 创建设备并连接成功后，可以开始进行设备通信。调用IoT Device 的client接口获取设备客户端，客户端提供了消息、属性、命令等通讯接口。

须知

SDK默认提供断线重连功能，当调用device.init（ ）后，当由于网络不稳定、网络不可达、平台主动断开连接等，导致连接失败，会在后台自动申请重连。若是客户不需要该机制，请见：**断线重连**文档。

消息上报

说明

消息上报Demo，可参考entry/src/main/ets/pages/MessageSample.ets。

消息上报是指设备向平台上报消息。

1. 在初始化并连接平台成功后，调用客户端的reportDeviceMessage接口上报设备消息，publishRawMessage上报自定义Topic的消息。

```
// 上报系统topic消息
const reportMessage: DeviceMessage = { content: this.message }
this.device.client.reportDeviceMessage(reportMessage)
  .then((data: IoTMqttResponse) => {
    LogUtil.info(TAG, `report deviceMessage success ${JSON.stringify(data)}. DeviceMessage is $
{JSON.stringify(reportMessage)}`);
  })
  .catch((error: IoTMqttResponse | string) => {
    LogUtil.error(TAG, `report deviceMessage failed ${JSON.stringify(error)}`);
  })

// 上报自定义topic消息（$oc开头，注意需要先在平台配置自定义topic）
const topic = `$oc/devices/${this.device?.deviceId}/user/test`;
const rawMessage: RawMessage = {
  topic: topic,
  qos: 0,
  payload: this.message
}
this.device?.client.publishRawMessage((rawMessage)).then((res: IoTMqttResponse) => {
  LogUtil.info(TAG, `publish rawMessage(${rawMessage.topic}) success, message is $
{JSON.stringify(rawMessage)}}`);
}).catch((error: IoTMqttResponse | string) => {
  LogUtil.error(TAG, `publish rawMessage(${rawMessage.topic}) failed, error is $
{JSON.stringify(error)}}`);
});

// 上报自定义topic消息（非$oc开头，可用设备topic策略控制权限）
const topic = "hello/world";
const rawMessage: RawMessage = {
  topic: topic,
  qos: 0,
  payload: this.message
}
this.device?.client.publishRawMessage((rawMessage)).then((res: IoTMqttResponse) => {
  LogUtil.info(TAG, `publish rawMessage(${rawMessage.topic}) success, message is $
{JSON.stringify(rawMessage)}}`);
}).catch((error: IoTMqttResponse | string) => {
  LogUtil.error(TAG, `publish rawMessage(${rawMessage.topic}) failed, error is $
{JSON.stringify(error)}}`);
});
```

2. 上报成功，查看日志发送消息成功。

图 3-74 上报系统 Topic 消息日志

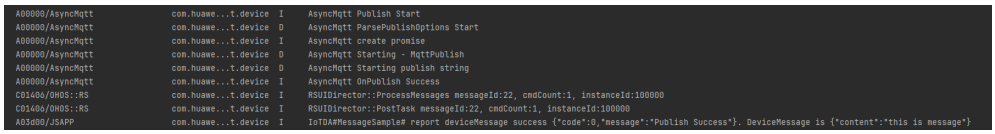


图 3-75 上报自定义 Topic（$oc 开头) 消息日志

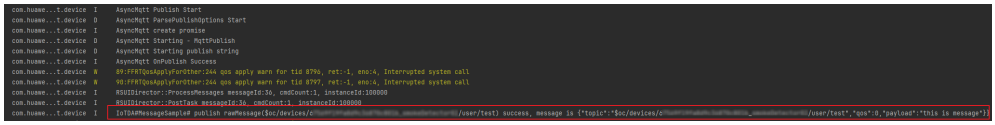
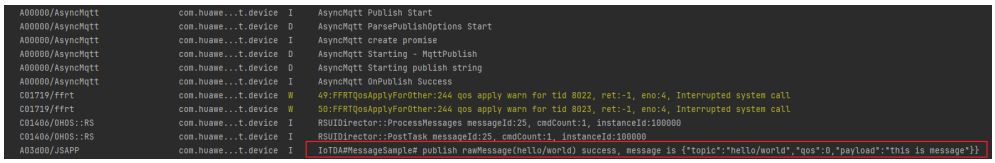


图 3-76 上报自定义 Topic（非$oc 开头）消息日志



3. 在设备接入控制台，选择“设备 > 所有设备”-查看设备是否在线。

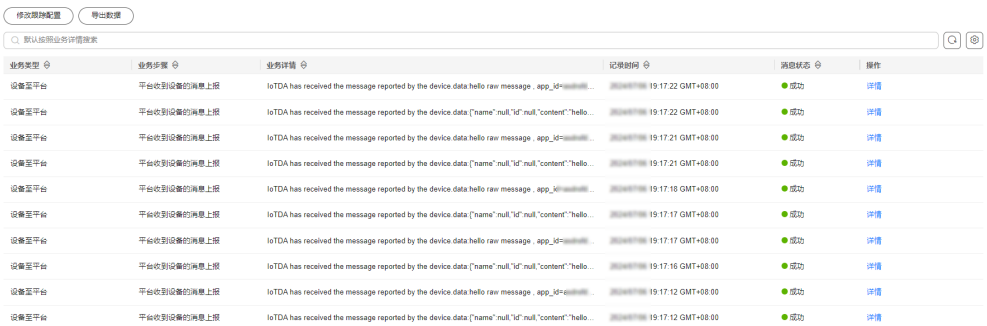
图 3-77 设备列表-设备在线

4. 选择对应设备，单击“详情”，进入设备详情页面启动设备消息跟踪。

图 3-78 消息跟踪-启动消息跟踪

5. 消息跟踪显示平台成功接收到设备的消息。

图 3-79 消息跟踪-查看 device_sdk_java 消息跟踪



须知

消息跟踪会有一定的延时，如果没有看到数据，请等待后刷新。

属性上报

说明

属性上报Demo，可参考entry/src/main/ets/pages/PropertySample.ets文件。

1. 在初始化并连接平台成功后，调用客户端的reportProperties接口上报设备属性。

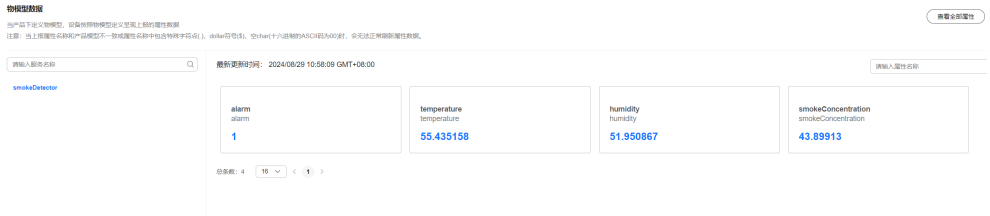
```
const properties: ServiceProperty[] =
[
  {
    "service_id": "smokeDetector",
    "properties": {
      "alarm": 1,
      "temperature": Math.random() * 100,
      "humidity": Math.random() * 100,
      "smokeConcentration": Math.random() * 100,
    }
  }
]
;
this.device.client.reportProperties(properties)
  .then((data: IoTMqttResponse) => {
    LogUtil.info(TAG, `report properties success ${JSON.stringify(data)}, properties is $
{JSON.stringify(properties)}`);  })
  .catch((error: IoTMqttResponse | string) => {
    LogUtil.error(TAG, `report properties failed ${JSON.stringify(error)}`);
  })
```

2. 上报成功，查看日志发送属性成功。

图 3-80 上报属性日志

3. 在设备接入控制台，选择“设备 > 所有设备”，选择对应设备，单击“详情”，进入设备详情页面可以看到最新上报的属性值。

图 3-81 物模型-属性上报



属性读写

说明

属性读写Demo，可参考entry/src/main/ets/pages/PropertySample.ets。

1. 在初始化连接成功后，调用客户端的propertyListener方法来设置属性回调接口。
   - 写属性处理：实现了alarm属性的写操作，其他属性不支持写操作。
   - 读属性处理：将本地属性值按照接口格式进行拼装。

```
let propertyListener: PropertyListener = {
  onPropertiesSet: (requestId: string, services: ServiceProperty[]): void => {
```

```
    this.logArr.unshift(`${new Date()}: onPropertiesSet requestId is ${requestId}, services is ${JSON.stringify(services)}`)
    // 遍历services
    services.forEach(serviceProperty => {
      LogUtil.info("onPropertiesSet, serviceId is ", serviceProperty.service_id);
      // 遍历属性
      Object.keys(serviceProperty.properties).forEach(name => {
        LogUtil.log(TAG, `property name is ${name}`);
        LogUtil.log(TAG, `set property value is ${serviceProperty.properties[name]}`);
      })
    })

    // 修改本地的属性
    this.device?.client.respondPropsSet(requestId, IotResult.SUCCESS);
  },
  onPropertiesGet: (requestId: string, serviceId?: string): void => {
    this.logArr.unshift(`${new Date()}: onPropertiesGet requestId is ${requestId}, serviceId is ${serviceId} and respondPropsGet`)
    LogUtil.info(TAG, `onPropertiesGet, the serviceId is ${serviceId}`);
    const serviceProperties: ServiceProperty[] = [
      {
        "service_id": "smokeDetector",
        "properties": {
          "alarm": 1,
          "temperature": Math.random() * 100,
          "humidity": Math.random() * 100,
          "smokeConcentration": Math.random() * 100,
        }
      }
    ];
    this.device?.client.respondPropsGet(requestId, serviceProperties);
  }
}
// 设置属性监听器
this.device.client.propertyListener = propertyListener;
```

说明

- 属性读写接口需要调用respondPropsGet和respondPropsSet接口来上报操作结果。
- 如果设备不支持平台主动到设备读，onPropertiesGet接口可以空实现。

2. 执行上述代码，设置属性监听器，在平台上设备影子页面查看当前alarm属性值为1，修改alarm属性为0后，查看设备侧日志，看到设备收到属性设置alarm属性为0。

图 3-82 设备影子-查看 alarm 属性

图 3-83 设备影子-属性配置 alarm

图 3-84 查看属性设置 alarm 为 0

命令下发

说明

命令下发Demo，可参考entry/src/main/ets/pages/CommandSample.ets。

设置命令监听器用来接收平台下发的命令，在回调接口里，需要对命令进行处理，并上报响应。

1. 在CommandSample例子中实现了命令的处理，收到命令后仅进行打印，然后调用respondCommand上报响应。

```
let commandListener: CommandListener = {
  onCommand: (requestId: string, serviceId: string, commandName: string, paras: object): void => {
    const command = `requestId is ${requestId}, serviceId is ${serviceId}, commandName is ${commandName}, paras is ${JSON.stringify(paras)}`;
    LogUtil.info(TAG, `received command is ${command}`);
    // 用户可以在该处进行命令处理

    const commandRsp: CommandRsp = {
      result_code: 0
    }
    this.device?.client.respondCommand(requestId, commandRsp).then((data: IoTMqttResponse) => {
      LogUtil.info(TAG, `respond command success ${JSON.stringify(data)}, commandRsp is ${commandRsp}}`);
    }).catch((err: IoTMqttResponse | string) => {
      LogUtil.error(TAG, `respond command failed ${JSON.stringify(err)}`);
    })
  }
}
this.device.client.commandListener = commandListener;
```

2. 执行上述代码设置命令监听后，在平台执行命令下发，其中serviceId为“smokeDetector”、命令名为“ringAlarm”、参数携带duration为整数20。
3. 查看日志，设备收到命令并成功上报响应。

面向物模型编程

说明

面向物模型编程Demo，可参考entry/src/main/ets/pages/ProfileSample.ets。

前面介绍了直接调用设备客户端的接口和平台进行通讯的方法，这种方式比较灵活，但用户需要妥善处理每一个接口，实现比较复杂。

SDK提供了一种更简单的方式，即面向物模型编程。面向物模型编程指基于SDK提供的物模型抽象能力，设备代码按照物模型定义设备服务，然后可以直接访问设备服务（即调用设备服务的属性读写接口），SDK就能自动和平台通讯，完成属性的同步和命令的调用。

相比直接调用客户端接口和平台进行通讯，面向物模型编程更简单，它简化了设备侧代码的复杂度，让设备代码只需要关注业务，而不用关注和平台的通讯过程。这种方式适合多数场景。

ProfileSample例子演示了如何面向物模型编程：

1. 首先定义一个烟感服务类，继承自AbstractService。（如果有多个服务，则需要定义多个服务类）：
   ```
   class SmokeDetector extends AbstractService  {

   }
   ```
2. 定义服务属性，私有变量以下划线开头，使用@Reflect.metadata("Property", { name: "***string***", writeable: ***boolean*** })注解表示一个属性，其中中name和产品模型中属性名保持一致。writeable用来标识属性是否可写
   ```
   @Reflect.metadata("Property", { name: "alarm", writeable: true })
   private _smokeAlarm: number = 1;

   @Reflect.metadata("Property", { name: "smokeConcentration", writeable: false })
   private _concentration: number = 0;

   @Reflect.metadata("Property", { name: "humidity", writeable: false })
   private _humidity: number = 0;

   @Reflect.metadata("Property", { name: "temperature", writeable: false })
   private _temperature: number = 10;
   ```
3. 定义服务的命令。设备收到平台下发的命令时，SDK会自动调用这里定义的命令。注解中name对应物模型的command_name，method对应接收命令的处理方法，命令的入参和返回值类型固定不能修改。

   这里定义的是一个响铃报警命令，命令名为ringAlarm，下发参数为”duration”，表示响铃报警的持续时间。
   ```
   @Reflect.metadata("DeviceCommand", {
     name: "ringAlarm",
     method: (paras: object): CommandRsp => {
       let duration: number = paras['duration'];
       LogUtil.log(TAG, `duration is ${duration}`);
           return IotResult.SUCCESS;
       }
   })
   private _alarm: Function = () => {};
   ```
4. 定义getter和setter接口。
   - 当设备收到平台下发的查询属性以及设备上报属性时，会自动调用getter方法。getter方法需要读取设备的属性值，可以实时到传感器读取或者读取本地的缓存
   - 当设备收到平台下发的设置属性时，会自动调用setter方法。setter方法需要更新设备本地的值。如果属性不支持写操作，setter保留空实现。

- setter和getter接口使用DevEco Studio右键的Generate的Getter and Setter自动生成，然后修改方法实现。

```
public set smokeAlarm(value: number) {
  this._smokeAlarm = value;
  if (value == 0) {
    LogUtil.info(TAG, "alarm is cleared by app");
  }
}

public get smokeAlarm(): number {
  return this._smokeAlarm;
}

public set concentration(value: number) {
  // 只读字段不需要实现set接口
}

public get concentration(): number {
  return Math.floor(Math.random() * 100);
}

public set humidity(value: number) {
  // 只读字段不需要实现set接口
}

public get humidity(): number {
  return Math.floor(Math.random() * 100);
}

public set temperature(value: number) {
  // 只读字段不需要实现set接口
}

public get temperature(): number {
  return Math.floor(Math.random() * 100);
}
```

5. 实现构造函数，完成属性和命令的初始化。

```
constructor() {
  super();
  const fields = Object.getOwnPropertyNames(this);
  this.init(fields);
}
```

6. 创建设备，注册烟感服务。

```
//创建设备服务
const smokeDetector = new SmokeDetector();
this.device.addService("smokeDetector", smokeDetector);
```

7. 开启周期上报。

```
//启动自动周期上报
this.device.getService("smokeDetector")?.enableAutoReport(10000);
```

8. 执行上述代码，查看日志上报属性。

```
com.huawe...t.device  I     AsyncMqtt Publish Start
com.huawe...t.device  D     AsyncMqtt Starting publish string
com.huawe...t.device  D     AsyncMqtt ParsePublishOptions Start
com.huawe...t.device  I     AsyncMqtt create promise
com.huawe...t.device  I     AsyncMqtt OnPublish Success
com.huawe...t.device  D     AsyncMqtt Starting - MqttPublish
com.huawe...t.device  D     AsyncMqtt Starting publish string
com.huawe...t.device  D     IoTDA_SDK# report properties success {"code":0,"message":"Publish Success"}
```

9. 在平台侧查看设备影子中属性alarm为1，修改alarm为0后，查看设备日志收到属性设置

图 3-85 设备影子-查看 alarm 属性

图 3-86 查看设备日志属性设置成功

10. 在平台下发ringAlarm命令，查看设备日志看到ringAlarm命令被调用，并且成功上报响应。

3.9.3 功能支持

SDK面向运算、存储能力较强的嵌入式终端设备，开发者通过调用SDK接口，便可实现设备与IoTDA服务（后续简出现的“服务”、“平台“均指IoTDA服务）双向通讯。SDK当前支持的功能有：

表 3-23 openHarmony Arkts 功能支持

功能	描述说明
设备初始化	作为客户端使用MQTT协议接入到华为云平台。当前仅支持密钥认证。
自定义选项	自定义选项提供配置自定义断线重连、连接状态监听器功能。
断线重连	设备连接失败或网络不稳定等原因导致被动断开连接时，会进行一个重连操作。
消息上报、下发	消息上报用于设备将自定义数据上报给平台，平台将设备上报的消息转发给应用服务器或华为云其他云服务上进行存储和处理，消息下发用于平台向设备下发消息，设备对收到的消息进行处理。
属性上报、设置、查询	属性上报用于设备按产品模型中定义的格式将属性数据上报给平台。属性设置用于平台设置设备属性。设备收到属性设置请求后，需要将执行结果返回给平台。属性查询用于平台查询设备的属性，设备收到属性查询请求后，需要将设备的属性数据返回给平台
命令下发	用于平台向设备下发设备控制命令。平台下发命令后，需要设备及时将命令的执行结果返回给平台。
获取设备影子	用于设备向平台获取设备影子数据。

功能	描述说明
面向物模型编程	面向物模型编程指的是，基于SDK提供的物模型抽象能力，设备代码只需要按照物模型定义设备服务，SDK就能自动的和平台通讯，完成属性的同步和命令的调用。 相比直接调用客户端接口和平台进行通讯，面向物模型编程简化了设备侧代码的复杂度，让设备代码只需要关注业务，而不用关注和平台的通讯过程。