设备接入服务

SDK 参考

文档版本 10

发布日期 2024-07-30

华为技术有限公司

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。

资源包名	描述	下载路径
应用侧开发 Java SDK	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**Java SDK使用指南**。	**Java SDK**
应用侧开发 .NET SDK	.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**.NET SDK使用指南**。	**.NET SDK**
应用侧开发 Python SDK	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**Python SDK使用指南**。	**Python SDK**
应用侧开发 Go SDK	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**Go SDK使用指南**。	**Go SDK**
应用侧开发 Node.js SDK	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**Node.js SDK使用指南**。	**Node.js SDK**
应用侧开发 PHP SDK	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**PHP SDK使用指南**。	**PHP SDK**
设备侧IoT Device SDK（Java）	设备可以通过集成IoT Device SDK(Java)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）**。	**IoT Device SDK(Java)**

资源包名	描述	下载路径
设备侧IoT Device SDK（C）	设备可以通过集成IoT Device SDK(C)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(C)使用指南**。	**IoT Device SDK(C)**
设备侧IoT Device SDK(C#)	设备可以通过集成IoT Device SDK(C#)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(C#)使用指南**。	**IoT Device SDK(C#)**
设备侧IoT Device SDK(Android)	设备可以通过集成IoT Device SDK(Android)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(Android)使用指南**。	**IoT Device SDK(Android)**
设备侧IoT Device SDK(Go社区版)	设备可以通过集成IoT Device SDK(Go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**IoT Device SDK(Go社区版)使用指南**。	**IoT Device SDK(Go社区版)**
设备侧IoT Device SDK Tiny（C）	设备可以通过集成IoT Device SDK Tiny (C)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 Tiny SDK(C)使用指南**。	**IoT Device SDK Tiny (C)**

2 应用侧 SDK

2.1 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)
             .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）**。

推荐您使用API在线调试工具**API Explorer**，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

2.2 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 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。

```
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
    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）**。

推荐您使用API在线调试工具**API Explorer**，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

2.3 .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.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(Credentials.DefaultDerivedPredicate);

        // 创建IoTDAClient实例并初始化
        var client = IoTDAAsyncClient.NewBuilder()
              .WithCredential(auth)
              // 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
              // .WithHttpConfig(new HttpConfig().WithIgnoreBodyForGetRequest(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());
        }
        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）**。

推荐您使用API在线调试工具**API Explorer**，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

2.4 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).
            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）**。

推荐您使用API在线调试工具**API Explorer**，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

2.5 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Node.js SDK的安装和配置，及使用Node.js SDK调用**应用侧API**的示例。

说明

当前SDK AK/SK的方式只支持基础版，不支持标准版和企业版，标准版和企业版建议使用token方式接入。

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>";
// 创建认证
const credentials = new core.BasicCredentials()
                    .withAk(ak)
                    .withSk(sk)
                    .withProjectId(project_id)
// 创建IoTDAClient实例并初始化
const client = iotda.IoTDAClient.newBuilder()
```

```
                        .withCredential(credentials)
                        .withEndpoint(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 ，根据你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 Node.js 软件开发工具包（Node.js SDK）**。

推荐您使用API在线调试工具**API Explorer** ，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

2.6 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PHP SDK的安装和配置，及使用PHP SDK调用**应用侧API**的示例。

说明

当前SDK AK/SK的方式只支持基础版，不支持标准版和企业版，标准版和企业版建议使用token方式接入。

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** 及以上版本。

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\Exceptions\ConnectionException;
use HuaweiCloud\SDK\Core\Exceptions\RequestTimeoutException;
use HuaweiCloud\SDK\Core\Exceptions\ServiceResponseException;
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>";
// 创建认证
$credentials = new BasicCredentials($ak,$sk,$projectId);
// 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($endpoint)
  ->withCredentials($credentials)
  ->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 ，根据你想操作的项目所属区域选择对应的项目 ID

----结束

更多

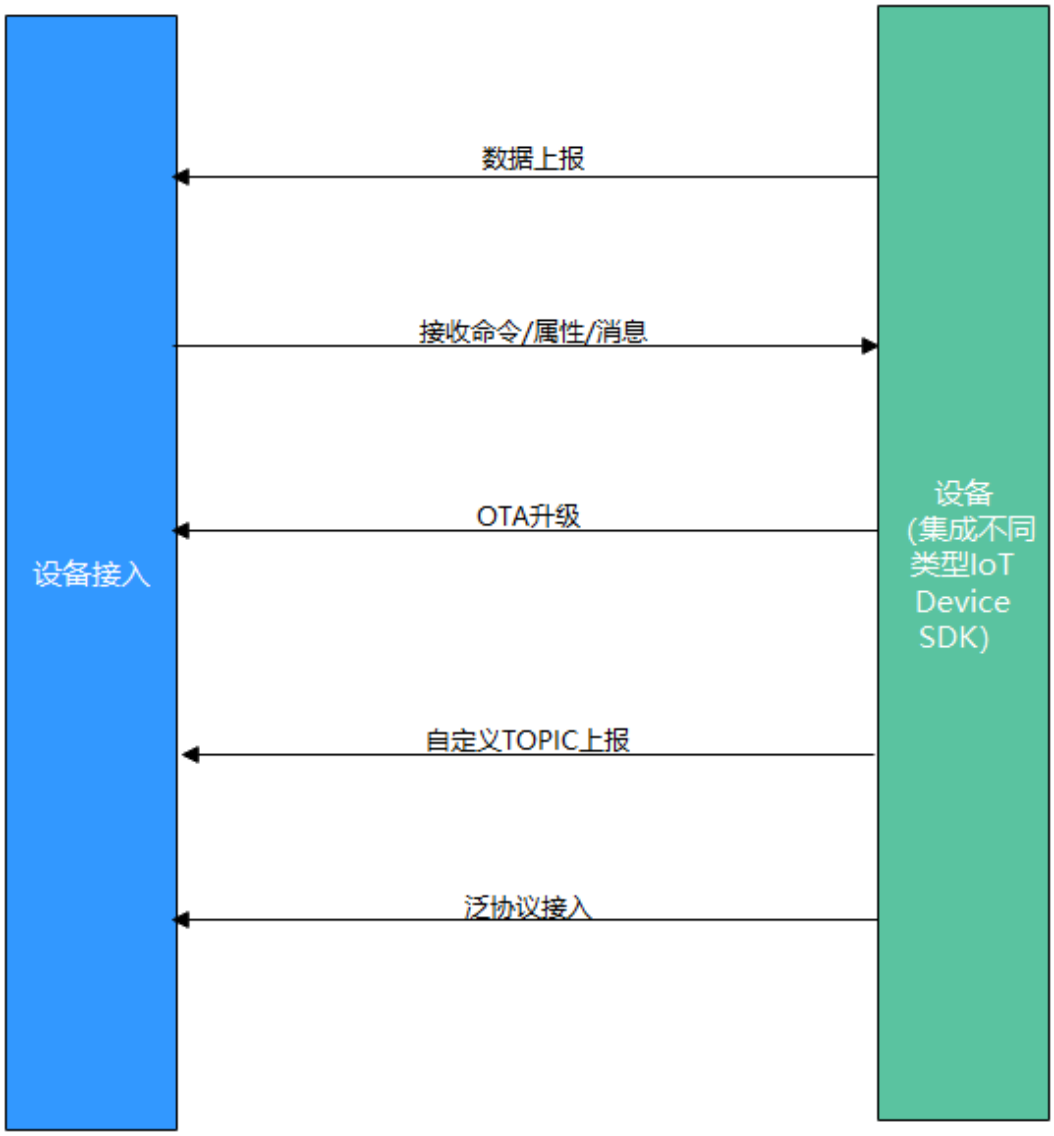
项目源码及更多详细的使用指导请参考**华为云开发者 PHP 软件开发工具包（PHP SDK）**。

推荐您使用API在线调试工具**API Explorer** ，API Explorer 支持快速调试和检索，调试API的同时，可以根据您的参数实时生成各种开发语言的SDK示例代码，方便您直接根据示例代码使用SDK。

3 设备侧 SDK

3.1 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IoT Device SDK。支持TCP/IP协议栈的设备集成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	无特殊要求	C

详细SDK使用指南，请参考：

- **IoT Device SDK使用指南（C）**
- **IoT Device SDK使用指南（Java）**
- **IoT Device SDK使用指南（C#）**
- **IoT Device SDK使用指南（Android）**

- **IoT Device SDK使用指南（Go社区版）**
- **IoT Device SDK Tiny使用指南**
- **IoT Device SDK 使用指南（OpenHarmony）**
- **IoT Device SDK使用指南（Python）**

SDK主要功能矩阵，请参考：

表 3-1 SDK 主要功能矩阵

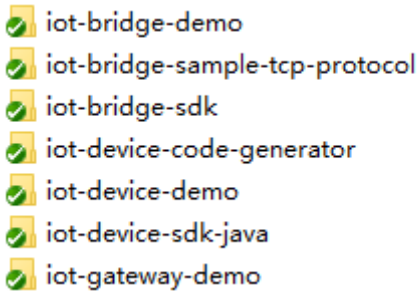
主要功能	C	Java	C#	Android	GO	python	C Tiny
属性上报	√	√	√	√	√	√	√
消息上报、下发	√	√	√	√	√	√	√
事件上报、下发	√	√	√	√	√	√	√
命令下发、响应	√	√	√	√	√	√	√
设备影子	√	√	√	√	√	√	√
OTA升级	√	√	√	√	√	√	√
bootstrap	√	√	√	√	√	√	√
时间同步	√	√	√	√	√	√	√
网关与子设备管理	√	√	√	√	√	√	√
端侧规则引擎	√	×	×	×	×	×	√
远程SSH	√	×	×	×	×	×	×
异常检测	√	×	×	×	×	×	×
端云安全通信（软总线）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

主要功能	C	Java	C#	Android	GO	python	C Tiny
M2M功能	√	×	×	×	×	×	×
泛协议接入	√	√	√	√	×	√	×

3.2 IoT Device SDK 使用指南（Java）

准备工作

- 开发环境要求：已经安装JDK（版本1.8以上）和maven
- 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协议链接网桥的样例

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

- 编译安装：进入到SDK根目录，执行mvn install

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 找到步骤3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”，上传步骤1获取的产品模型文件。

图 3-1 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	确保和步骤3创建的产品归属在同一个资源空间。
所属产品	选择步骤3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. 创建设备时，需要写入在注册设备时获取的设备ID、密码，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);
```

注意

所有涉及设备ID和密码的文件均需要修改成对应的信息。

2. 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
```

3. 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。

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这个例子中我们周期性上报消息：

```
while (true) {
    device.getClient().reportDeviceMessage(new DeviceMessage("hello"), new ActionListener() {
        @Override
```



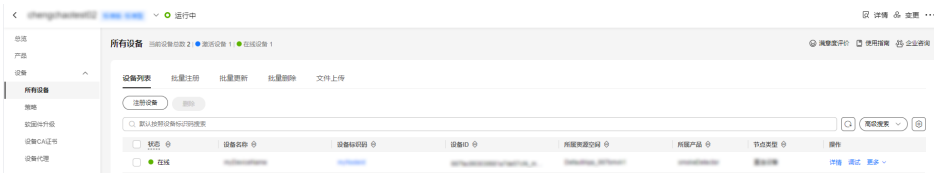
```
            public void onSuccess(Object context) {
                log.info("reportDeviceMessage ok");
            }

            @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，查看日志打印看到连接成功和发送消息的打印：

```
2024-04-16 16:43:09  INFO AbstractService:103 - create device, the deviceId is
5e06bfee334dd4f33759f5b3_demo
2024-04-16 16:43:09  INFO MqttConnection:233 - try to connect to ssl://域名信息:8883
2024-04-16 16:43:10  INFO MqttConnection:257 - connect success, the uri is ssl://域名信息:8883
2024-04-16 16:43:11  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect
success","timestamp":"1713256990817"},"service_id":"$log","event_type":"log_report","event_time":"20
240416T084310Z","event_id":null}]}
2024-04-16 16:43:11  INFO MqttConnection:140 - Mqtt client connected. address is ssl://域名信息:8883
2024-04-16 16:43:11  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"device_sdk_version":"JAVA_v1.2.0","fw_version":null,"sw_version":null},"service_id":"$sdk_info","event
_type":"sdk_info_report","event_time":"20240416T084311Z","event_id":null}]}
2024-04-16 16:43:11  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect complete, the url is ssl://域名信
息:8883","timestamp":"1713256991263"},"service_id":"$log","event_type":"log_report","event_time":"2
0240416T084311Z","event_id":null}]}
2024-04-16 16:43:11  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/messages/up, msg =
{"name":null,"id":null,"content":"hello","object_device_id":null}
2024-04-16 16:43:11  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/user/wpy, msg =  hello raw message
2024-04-16 16:43:11  INFO MessageSample:98 - reportDeviceMessage ok
2024-04-16 16:43:11  INFO MessageSample:113 - publishRawMessage 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 3-2 设备列表-设备在线

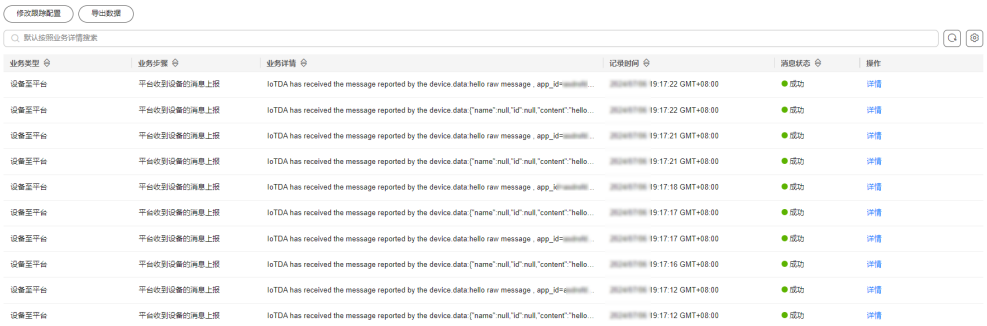


5. 选择对应设备，单击“详情”，在设备详情页面启动设备消息跟踪。

图 3-3 消息跟踪-启动消息跟踪

6. 平台收到了设备的消息。

图 3-4 消息跟踪-查看 device_sdk_java 消息跟踪



注：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

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);
    }
}
```

修改PropertySample的main函数后直接运行PropertySample类，查看日志看到发送成功的打印

```
2024-04-17 15:38:37  INFO AbstractService:103 - create device, the deviceId is
5e06bfee334dd4f33759f5b3_demo
2024-04-17 15:38:37  INFO MqttConnection:233 - try to connect to ssl://域名信息:8883
2024-04-17 15:38:38  INFO MqttConnection:257 - connect success, the uri is ssl://域名信息:8883
2024-04-17 15:38:38  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"661e35467bdccc0126d1a595_feng-sdk-test3","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect
success","timestamp":"1713339518043"},"service_id":"$log","event_type":"log_report","event_time":"2024041
7T073838Z","event_id":null}]}
2024-04-17 15:38:38  INFO MqttConnection:140 - Mqtt client connected. address is ssl://域名信息:8883
2024-04-17 15:38:38  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"661e35467bdccc0126d1a595_feng-sdk-test3","services":[{"paras":
{"device_sdk_version":"JAVA_v1.2.0","fw_version":null,"sw_version":null},"service_id":"$sdk_info","event_type"
:"sdk_info_report","event_time":"20240417T073838Z","event_id":null}]}
2024-04-17 15:38:38  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect complete, the url is ssl://域名信
息:8883","timestamp":"1713339518464"},"service_id":"$log","event_type":"log_report","event_time":"202404
17T073838Z","event_id":null}]}
2024-04-17 15:38:38  INFO MqttConnection:299 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/properties/report, msg =  {"services":[{"properties":
{"alarm":1,"temperature":55.435158,"humidity":51.950867,"smokeConcentration":43.89913},"service_id":"sm
okeDetector","event_time":null}]}
2024-04-17 15:38:38  INFO PropertySample:144 - pubMessage success
```

在平台设备详情页面可以看到最新上报的属性值：

图 3-5 单设备注册-gasdevice

属性读写

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，我们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));
        }
    });
```

注:

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

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 3-6 设备影子-查看 alarm 属性

我们把alarm属性修改为0：

图 3-7 设备影子-属性配置 alarm

查看设备侧日志，看到设备收到属性设置，alarm被修改为0：

```
INFO MqttConnection:66 - messageArrived topic =  $oc/devices/[redacted]_demo/sys/properties/set/reque
INFO PropertySample:53 - OnPropertiesSet, serviceId =  smokeDetector
INFO PropertySample:57 - property name = alarm
INFO PropertySample:58 - set property value = 0
```

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。

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));
            }

        });
```

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。

查看日志，看到设备收到命令并上报了响应：

```
INFO MqttConnection:66 - messageArrived topic =  $oc/devices/test_testDevice/sys/commands/request_id=4, msg = {"paras":{"duration":20},"service_id":"smo
INFO CommandSample:62 - onCommand, serviceId = smokeDetector
INFO CommandSample:63 - onCommand , name = ringAlarm
INFO CommandSample:64 - onCommand, paras =  {duration=20}
INFO MqttConnection:213 - publish message topic =  $oc/devices/test_testDevice/sys/commands/response/request_id=4, msg = {"paras":null,"result_code":0,"
```

面向物模型编程

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。

SmokeDetector例子演示了如何面向物模型编程：

1. 按照物模型定义服务类和服务的属性（如果有多个服务，则需要定义多个服务类）：

   ```
   public static class SmokeDetectorService extends AbstractService {
   ```



```
    //按照设备模型定义属性，注意属性的name和类型需要和模型一致，writeable表示属性知否可写，name指定属性名
    @Property(name = "alarm", writeable = true)
    int smokeAlarm = 1;

    @Property(name = "smokeConcentration", writeable = false)
    float concentration = 0.0f;

    @Property(writeable = false)
    int humidity;

    @Property(writeable = false)
    float temperature;
```

用@Property注解来表示是一个属性，可以用name指定属性名，如果不指定则使用字段名。

属性可以加上writeable来控制权限，如果属性只读，则加上writeable = false，如果不加，默认认为可读写。

2. 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。

   接口入参和返回值的类型是固定的不能修改，否则会出现运行时错误。

   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。

```
//定义命令，注意接口入参和返回值类型是固定的不能修改，否则会出现运行时错误
    @DeviceCommand(name = "ringAlarm")
    public CommandRsp alarm(Map<String, Object> paras) {
        int duration = (int) paras.get("duration");
        log.info("ringAlarm  duration = " + duration);
        return new CommandRsp(0);
    }
```

3. 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接口的命名应该符合java bean规范，sdk会自动调用这些接口
    public int getHumidity() {

        //模拟从传感器读取数据
        humidity = new Random().nextInt(100);
        return humidity;
    }

    public void setHumidity(int humidity) {
        //humidity是只读的，不需要实现
    }

    public float getTemperature() {

        //模拟从传感器读取数据
        temperature = new Random().nextInt(100);
        return temperature;
    }

    public void setTemperature(float temperature) {
        //只读字段不需要实现set接口
    }

    public float getConcentration() {

        //模拟从传感器读取数据
```



```
        concentration = new Random().nextFloat()*100.0f;
        return concentration;
    }

    public void setConcentration(float concentration) {
        //只读字段不需要实现set接口
    }

    public int getSmokeAlarm() {
        return smokeAlarm;
    }

    public void setSmokeAlarm(int smokeAlarm) {

        this.smokeAlarm = smokeAlarm;
        if (smokeAlarm == 0){
            log.info("alarm is cleared by app");
        }
    }
```

4. 在main函数中创建服务实例并添加到设备。

```
//创建设备
IoTDevice device = new IoTDevice(serverUri, deviceId, secret);

//创建设备服务
SmokeDetectorService smokeDetectorService = new SmokeDetectorService();
device.addService("smokeDetector", smokeDetectorService);

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

5. 开启周期上报：

```
//启动自动周期上报
smokeDetectorService.enableAutoReport(10000);
```

备注：如果不想周期上报，也可以调用firePropertiesChanged接口手工触发上报。

直接运行SmokeDetector类，查看日志在上报属性：

```
INFO MqttConnection:140 - try to connect to ssl://[redacted].myhuaweicloud.com:8883
INFO MqttConnection:147 - connect success ssl://[redacted].myhuaweicloud.com:8883
INFO MqttConnection:87 - Mqtt client connected. address :ssl://[redacted].myhuaweicloud.com:8883
connect ok
INFO MqttConnection:213 - publish message topic =  $oc/devices/5e06bfee334dd4f33759f5b3_demo/sys/properties/report, msg = {"service
```

在平台侧查看设备影子：

图 3-8 设备影子-查看 alarm 属性

在平台上修改属性alarm，查看设备日志收到属性设置：

```
INFO MqttConnection:66 - messageArrived topic =  $oc/devices/test_testDevice/sys/properties/set/request_id=2, msg = {"services":[{"pro
INFO AbstractService:187 - write property ok:alarm
```

在平台下发ringAlarm命令：

查看设备日志看到ringAlarm命令被调用，并且上报了响应：


```
INFO MqttConnection:66 - messageArrived topic =  $oc/devices/test_testDevice/sys/commands/request_id=1, msg = {"paras":{"duration":20},
INFO DeviceServiceSample$SmokeDetectorService:53 - ringAlarm  duration = 20
INFO MqttConnection:213 - publish message topic =  $oc/devices/test_testDevice/sys/commands/response/request_id=1, msg = {"paras":null,
```

使用代码生成器

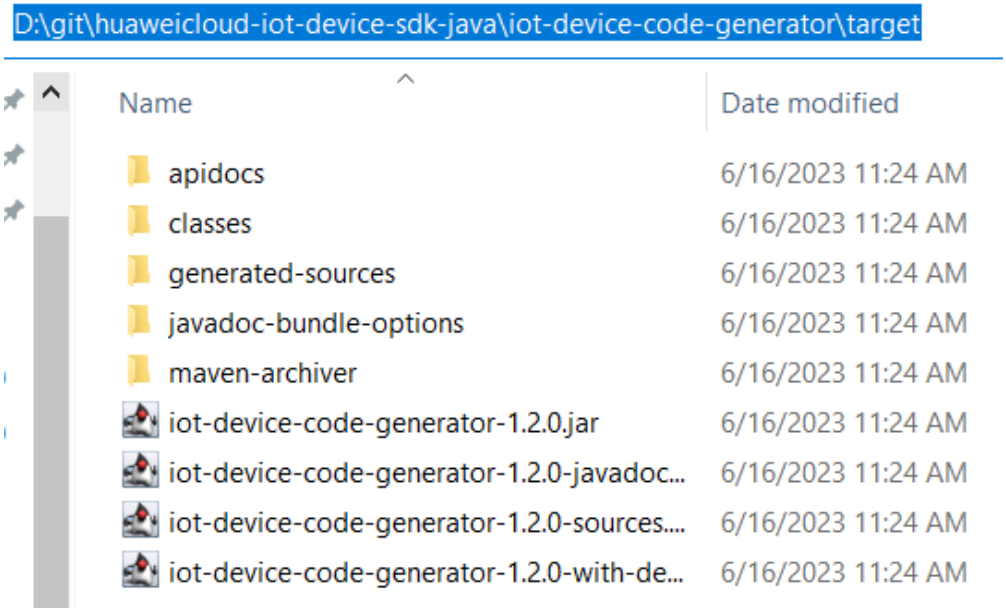
sdk提供了设备代码生成器，用户只需要提供产品模型文件，就能自动生成设备代码框架。代码生成器可以解析设备模型文件，然后对模型里定义的每个服务，生成对应的service类，然后生成一个设备主类，在main函数中创建设备并注册设备服务实例。

使用代码生成器生成设备代码的步骤：

1. 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”。

```
[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] ------------------------------------------------------------------------
```

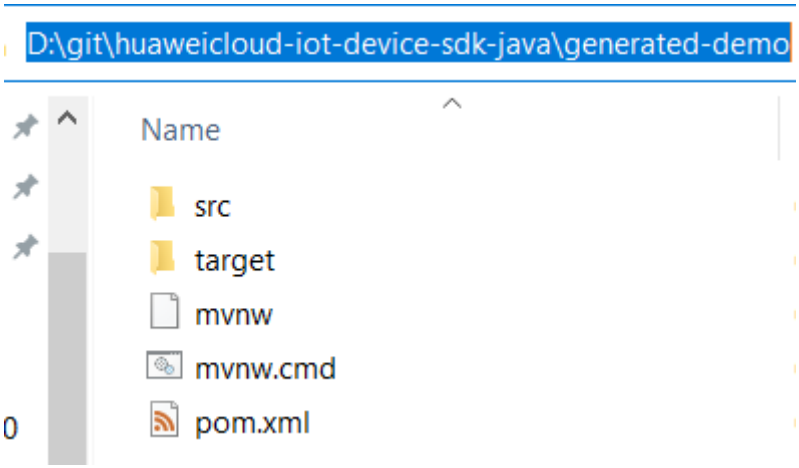
2. 执行完成会在iot-device-code-generator的target下生成可执行jar包。



3. 将产品模型文件保存到本地，比如我的模型文件“smokeDetector.zip”放到D盘。
4. 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”。

```
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
```

5. 在huaweicloud-iot-device-sdk-java目录下会生成generated-demo包。



至此，设备代码已经生成。

编译运行生成的代码：

1. 访问“huaweicloud-iot-device-sdk-java\generated-demo”，执行“mvn install”，在target下生成jar包。

```
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-javadoc.jar
-a----         6/16/2023  11:31 AM           6728 iot-device-demo-ganerated-1.2.0-sources.jar
-a----         6/16/2023  11:31 AM       11530020 iot-device-demo-ganerated-1.2.0-with-deps.jar
-a----         6/16/2023  11:31 AM           8031 iot-device-demo-ganerated-1.2.0.jar
```

2. 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
2024-04-17 15:50:55  INFO MqttConnection:268 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect
success","timestamp":"1713340255148"},"service_id":"$log","event_type":"log_report","event_time":"20
240417T075055Z","event_id":null}]}
2024-04-17 15:50:55  INFO MqttConnection:111 - Mqtt client connected. address is ssl://域名信息:8883
2024-04-17 15:50:55  INFO MqttConnection:268 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"device_sdk_version":"JAVA_v1.2.0","fw_version":null,"sw_version":null},"service_id":"$sdk_info","event
_type":"sdk_info_report","event_time":"20240417T075055Z","event_id":null}]}
2024-04-17 15:50:55  INFO MqttConnection:268 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/events/up, msg =
{"object_device_id":"5e06bfee334dd4f33759f5b3_demo","services":[{"paras":
{"type":"DEVICE_STATUS","content":"connect complete, the url is ssl://域名信
息:8883","timestamp":"1713340255496"},"service_id":"$log","event_type":"log_report","event_time":"2
0240417T075055Z","event_id":null}]}
2024-04-17 15:51:03  INFO smokeDetectorService:78 - report property alarm value =  50
2024-04-17 15:51:03  INFO smokeDetectorService:104 - report property temperature value =
0.3648571367849047
2024-04-17 15:51:03  INFO smokeDetectorService:91 - report property smokeConcentration value =
0.679772877336927
2024-04-17 15:51:03  INFO smokeDetectorService:117 - report property humidity value =  15
2024-04-17 15:51:03  INFO MqttConnection:268 - publish message topic is $oc/devices/
5e06bfee334dd4f33759f5b3_demo/sys/properties/report, msg =  {"services":[{"properties":
{"alarm":50,"temperature":0.3648571367849047,"smokeConcentration":0.679772877336927,"humidity
":15},"service_id":"smokeDetector","event_time":"20240417T075103Z"}]}
```

修改扩展生成的代码：

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。

1. 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);
}
```

2. 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。
3. 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。

如何开发网关

网关是一个特殊的设备，除具备一般设备功能之外，还具有子设备管理、子设备消息转发的功能。SDK提供了AbstractGateway抽象类来简化网关的实现。该类提供了子设备管理功能，需要从平台获取子设备信息并保存（需要子类提供子设备持久化接口）、子设备下行消息转发功能（需要子类实现转发处理接口）、以及上报子设备列表、上报子设备属性、上报子设备状态、上报子设备消息等接口。

- **使用AbstractGateway类**

  继承该类，在构造函数里提供子设备信息持久化接口，实现其下行消息转发的抽象接口：

  ```
  public abstract void onSubdevCommand(String requestId, Command command);

  public abstract void onSubdevPropertiesSet(String requestId, PropsSet propsSet);

  public abstract void onSubdevPropertiesGet(String requestId, PropsGet propsGet);

  public abstract void onSubdevMessage(DeviceMessage message);
  ```

- **iot-gateway-demo代码介绍**

  工程iot-gateway-demo基于**AbstractGateway**实现了一个简单的网关，它提供tcp设备接入能力。关键类：

  SimpleGateway：继承自AbstractGateway，实现子设备管理和下行消息转发

  StringTcpServer：基于netty实现一个TCP server，本例中子设备采用TCP协议，并且首条消息为鉴权消息

  SubDevicesFilePersistence：子设备信息持久化，采用json文件来保存子设备信息，并在内存中做了缓存

  Session：设备会话类，保存了设备id和TCP的channel的对应关系

- **SimpleGateway类**

  添加或删除子设备处理

  添加子设备：AbstractGateway的onAddSubDevices接口已经完成了子设备信息的保存。我们不需要再增加额外处理，因此SimpleGateway不需要重写onAddSubDevices接口

  删除子设备：我们不仅需要修改持久化信息，还需要断开当前子设备的连接。所以我们重写了onDeleteSubDevices接口，增加了拆链处理，然后调用父类的onDeleteSubDevices。

  ```
  @Override
    public int onDeleteSubDevices(SubDevicesInfo subDevicesInfo) {

        for (DeviceInfo subdevice : subDevicesInfo.getDevices()) {
            Session session = nodeIdToSesseionMap.get(subdevice.getNodeId());
            if (session != null) {
                if (session.getChannel() != null) {
                    session.getChannel().close();
                    channelIdToSessionMap.remove(session.getChannel().id().asLongText());
                    nodeIdToSesseionMap.remove(session.getNodeId());
                }
            }
        }
        return super.onDeleteSubDevices(subDevicesInfo);

    }
  ```

- **下行消息处理**

  网关收到平台下行消息时，需要转发给子设备。平台下行消息分为三种：设备消息、属性读写、命令。

  - **设备消息：**这里我们需要根据deviceId获取nodeId，从而获取session，从session里获取channel，就可以往channel发送消息。在转发消息时，可以根据需要进行一定的转换处理。

    ```
    @Override
      public void onSubdevMessage(DeviceMessage message) {

          //平台接口带的都是deviceId，deviceId是由nodeId和productId拼装生成的，即
    ```



```
        //deviceId = productId_nodeId
        String nodeId = IotUtil.getNodeIdFromDeviceId(message.getDeviceId());
        if (nodeId == null) {
            return;
        }

        //通过nodeId获取session，进一步获取channel
        Session session = nodeIdToSesseionMap.get(nodeId);
        if (session == null) {
            log.error("subdev is not connected " + nodeId);
            return;
        }
        if (session.getChannel() == null){
            log.error("channel is null " + nodeId);
            return;
        }

        //直接把消息转发给子设备
        session.getChannel().writeAndFlush(message.getContent());
        log.info("writeAndFlush " + message);
    }
```

- **属性读写：**

  属性读写包括属性设置和属性查询。

  属性设置：

```
@Override
    public void onSubdevPropertiesSet(String requestId, PropsSet propsSet) {

        if (propsSet.getDeviceId() == null) {
            return;
        }

        String nodeId = IotUtil.getNodeIdFromDeviceId(propsSet.getDeviceId());
        if (nodeId == null) {
            return;
        }

        Session session = nodeIdToSesseionMap.get(nodeId);
        if (session == null) {
            return;
        }

        //这里我们直接把对象转成string发给子设备，实际场景中可能需要进行一定的编解码转换
        session.getChannel().writeAndFlush(JsonUtil.convertObject2String(propsSet));

        //为了简化处理，我们在这里直接回响应。更合理做法是在子设备处理完后再回响应
        getClient().respondPropsSet(requestId, IotResult.SUCCESS);

        log.info("writeAndFlush " + propsSet);

    }
```

  属性查询：

```
@Override
    public void onSubdevPropertiesGet(String requestId, PropsGet propsGet) {

        //不建议平台直接读子设备的属性，这里直接返回失败
        log.error("not supporte onSubdevPropertiesGet");
        deviceClient.respondPropsSet(requestId, IotResult.FAIL);
    }
```

- **命令：**处理流程和消息类似，实际场景中可能需要不同的编解码转换。

```
@Override
    public void onSubdevCommand(String requestId, Command command) {

        if (command.getDeviceId() == null) {
            return;
        }
```

```
        String nodeId = IotUtil.getNodeIdFromDeviceId(command.getDeviceId());
        if (nodeId == null) {
            return;
        }

        Session session = nodeIdToSesseionMap.get(nodeId);
        if (session == null) {
            return;
        }

        //这里我们直接把command对象转成string发给子设备，实际场景中可能需要进行一定的编解码转换
        session.getChannel().writeAndFlush(JsonUtil.convertObject2String(command));

        //为了简化处理，我们在这里直接回命令响应。更合理做法是在子设备处理完后再回响应
        getClient().respondCommand(requestId, new CommandRsp(0));
        log.info("writeAndFlush " + command);
    }
```

- **上行消息处理**

  上行处理在StringTcpServer的channelRead0接口里。如果会话不存在，需要先创建会话：

  如果子设备信息不存在，这里会创建会话失败，直接拒绝连接

```
@Override
    protected void channelRead0(ChannelHandlerContext ctx, String s) throws Exception {
        Channel incoming = ctx.channel();
        log.info("channelRead0" + incoming.remoteAddress() + " msg :" + s);

        //如果是首条消息,创建session
//如果是首条消息,创建session
        Session session = simpleGateway.getSessionByChannel(incoming.id().asLongText());
        if (session == null) {
            String nodeId = s;
            session = simpleGateway.createSession(nodeId, incoming);

            //创建会话失败，拒绝连接
            if (session == null) {
                log.info("close channel");
                ctx.close();
            }
        }
```

  如果会话存在，则进行消息转发：

```
else {
            //如果需要上报属性则调用reportSubDeviceProperties
            DeviceMessage deviceMessage = new DeviceMessage(s);
            deviceMessage.setDeviceId(session.getDeviceId());
            simpleGateway.reportSubDeviceMessage(deviceMessage, null);

        }
```

  到这里，网关的关键代码介绍完了，其他的部分看源代码。整个demo是开源的，用户可以根据需要进行扩展。比如修改持久化方式、转发中增加消息格式的转换、实现其他子设备接入协议。

- **iot-gateway-demo的使用**

  a. 创建子设备的产品，步骤可参考**创建产品**。

  b. 在创建的产品中定义模型，添加服务，服务ID为parameter。并且新增alarm和temperature两个属性，如下图所示

图 3-9 模型定义-子设备产品

c. 修改StringTcpServer的main函数，替换构造参数，然后运行该类。

```
simpleGateway = new SimpleGateway(new SubDevicesFilePersistence(),
                "ssl://iot-acc.cn-north-4.myhuaweicloud.com:8883",
                "5e06bfee334dd4f33759f5b3_demo", "mysecret");
```

d. 在平台上看到该网关在线后，添加子设备。

图 3-10 设备-添加子设备

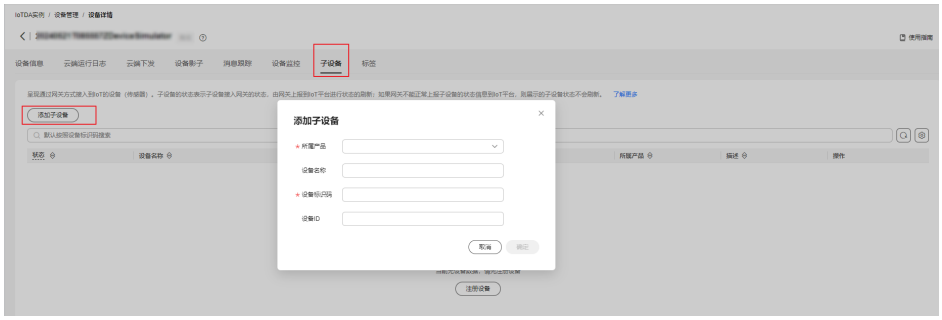


表 3-2 子设备参数

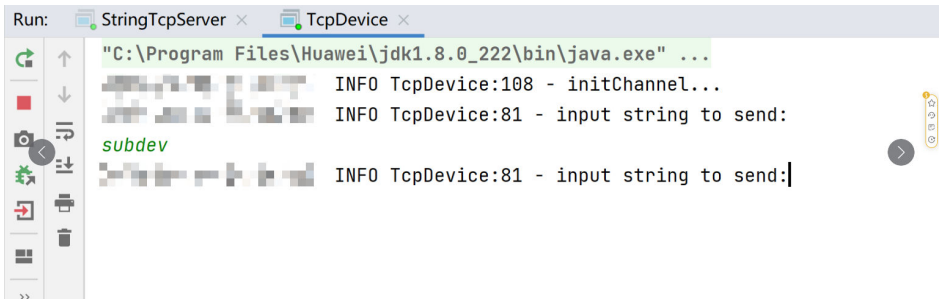
参数名称	参数描述
所属产品	子设备所属的产品，选择**步骤1**创建的产品。
设备名称	即device_name，可自定义，如subdev_name
设备标识码	即node_id，填写subdev。
设备ID	即devicee_id，可不填写，自动生成。

此时网关上日志打印：

2024-04-16 21:00:01 INFO SubDevicesFilePersistence:112 - add subdev, the nodeId is subdev

e. 运行TcpDevice类，建立连接后，输入步骤3中注册的子设备的nodeId，如subdev。

图 3-11 子设备连接

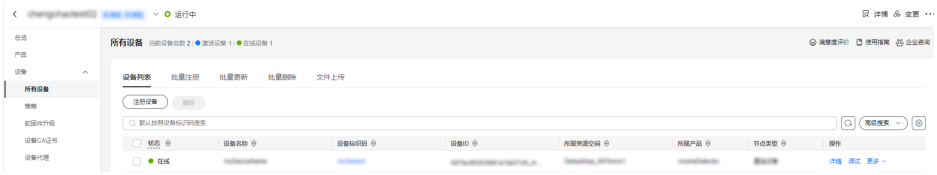


此时网关设备日志打印：

```
2024-04-16 21:00:54  INFO StringTcpServer:196 - initChannel: /127.0.0.1:21889
2024-04-16 21:01:00  INFO StringTcpServer:137 - channelRead0 is /127.0.0.1:21889, the msg is subdev
2024-04-16 21:01:00  INFO SimpleGateway:100 - create new session ok, the session is Session{nodeId='subdev', channel=[id: 0xf9b89f78, L:/127.0.0.1:8080 - R:/127.0.0.1:21889], deviceId='subdev_deviceId'}
```

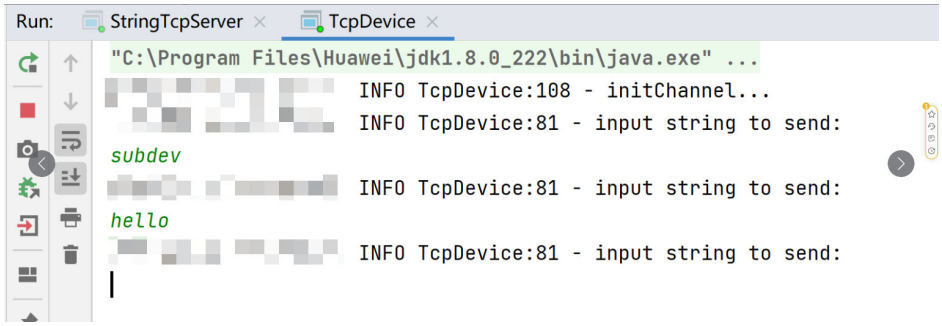
f. 在平台上看到子设备上线。

图 3-12 设备列表-设备在线



g. 子设备上报消息

图 3-13 子设备上报消息



查看日志看到上报成功

```
2024-04-16 21:02:36  INFO StringTcpServer:137 - channelRead0 is /127.0.0.1:21889, the msg is hello
2024-04-16 21:02:36  INFO MqttConnection:299 - publish message topic is $oc/devices/5e06bfee334dd4f33759f5b3_demo/sys/messages/up, msg = {"name":null,"id":null,"content":"hello","object_device_id":"subdev_deviceId"]
2024-04-16 21:02:36  INFO MqttConnection:299 - publish message topic is $oc/devices/5e06bfee334dd4f33759f5b3_demo/sys/gateway/sub_devices/properties/report, msg = {"devices":[{"services":[{"properties":{"temprature":2,"alarm":1},"service_id":"parameter","event_time":null}],"device_id":"subdev_deviceId"}]]
```

h. 查看消息跟踪

在平台上找到网关，选择 设备详情-消息跟踪，打开消息跟踪。继续让子设备发送数据，等待片刻后看到消息跟踪：

图 3-14 消息跟踪-直连设备

3.3 IoT Device SDK 使用指南（C）

IoT Device SDK（C）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**IoT Device SDK（C）使用指南**。

3.4 IoT Device SDK 使用指南（C#）

IoT Device SDK（C#）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**IoT Device SDK（C#）使用指南**。

3.5 IoT Device SDK 使用指南（Android）

IoT Device SDK（Android）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**IoT Device SDK（Android）使用指南**。

3.6 IoT Device SDK 使用指南（Go 社区版）

Go语言版的SDK提供了跟平台基础的通信能力，由开源社区提供，如果使用有问题请在**github**上提issue。

3.7 IoT Device SDK Tiny 使用指南（C）

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。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。

3.8 IoT Device SDK 使用指南（OpenHarmony）

IoT Device SDK（OpenHarmony）提供设备接入华为云IoT物联网平台的OpenHarmony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**IoT Device SDK（OpenHarmony）使用指南**。

3.9 IoT Device SDK 使用指南（Python）

IoT Device SDK（Python）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。相关集成指导请参考**IoT Device SDK（Python）使用指南**。