

IoT 平台增强版

# 开发指南

文档版本 09

发布日期 2019-01-25



版权所有 © 华为技术有限公司 2019。保留一切权利。

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

## 商标声明



HUAWEI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默示的声明或保证。

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

<b>1 新手入门.....</b>	<b>1</b>
1.1 平台简介.....	1
1.2 相关概念.....	1
1.3 前期准备.....	2
1.4 快速入门.....	4
1.4.1 开始前必读.....	4
1.4.2 案例 1：智能门锁.....	5
1.4.3 案例 2：补光灯.....	17
<b>2 开发准备.....</b>	<b>25</b>
2.1 能力要求.....	25
2.2 厂商配置.....	25
2.3 项目创建.....	26
<b>3 设备开发.....</b>	<b>29</b>
3.1 开发说明.....	29
3.2 LWM2M.....	29
3.2.1 产品创建.....	29
3.2.2 Profile 定义.....	35
3.2.3 插件开发.....	41
3.2.3.1 插件开发指导.....	41
3.2.3.2 插件开发实例.....	66
3.2.3.2.1 数据上报和命令下发的插件开发.....	66
3.2.3.2.2 多条数据上报消息的插件开发.....	74
3.2.3.2.3 字符串及可变长字符串数据类型的插件开发.....	87
3.2.3.2.4 数组及可变长数组数据类型的插件开发.....	103
3.2.3.2.5 含命令执行结果的编解码插件开发.....	120
3.2.4 设备集成.....	135
3.2.5 设备调测.....	136
3.2.6 认证测试.....	142
3.2.6.1 认证测试指导.....	142
3.2.6.2 认证测试用例.....	144
3.2.6.2.1 终端开机入网测试.....	144
3.2.6.2.2 数据上报测试.....	145

3.2.6.2.3 数据上报属性测试.....	146
3.2.6.2.4 控制命令下发测试.....	147
3.2.6.2.5 命令下发响应测试.....	148
3.2.7 产品发布.....	149
3.2.8 上线商用平台.....	150
3.3 MQTT.....	153
3.3.1 产品创建.....	153
3.3.2 Profile 定义.....	159
3.3.3 设备集成.....	163
3.3.4 设备调测.....	164
3.3.5 认证测试.....	170
3.3.5.1 认证测试指导.....	170
3.3.5.2 认证测试用例.....	172
3.3.5.2.1 终端开机入网测试.....	172
3.3.5.2.2 数据上报测试.....	173
3.3.5.2.3 数据上报属性测试.....	174
3.3.5.2.4 控制命令下发测试.....	175
3.3.5.2.5 命令下发响应测试.....	176
3.3.6 产品发布.....	177
3.3.7 上线商用平台.....	178
<b>4 应用开发.....</b>	<b>182</b>
4.1 开发说明.....	182
4.2 应用接入.....	182
4.3 订阅.....	184
4.4 注册设备.....	186
4.5 设备接入.....	187
4.6 数据上报.....	188
4.7 命令下发.....	189
4.8 其他接口.....	190
<b>5 参考信息.....</b>	<b>191</b>
5.1 准备 Java 开发环境.....	191
5.1.1 安装 JDK1.8.....	191
5.1.2 配置 Java 环境变量（Windows 操作系统）.....	191
5.1.3 安装 Eclipse.....	195
5.1.4 新建工程.....	195
5.1.5 导入样例代码.....	197
5.2 单步调测.....	199
5.3 使用 Postman 测试平台北向接口.....	202
5.4 Profile 文件.....	206
5.4.1 概念.....	206
5.4.2 线下开发参考.....	206
5.4.2.1 设备 Profile 写作.....	207

5.4.2.2 设备 Profile 提供形式.....	209
5.4.2.3 设备 Profile 文件字段含义说明.....	210
5.4.3 附录.....	219
5.5 编解码插件.....	228
5.5.1 整体方案.....	228
5.5.2 消息处理流程.....	229
5.5.3 线下开发参考.....	230
5.5.3.1 开发环境准备.....	230
5.5.3.2 开发编解码插件.....	233
5.5.3.2.1 导入编解码插件 DEMO 工程.....	233
5.5.3.2.2 开发插件.....	236
5.5.3.2.3 编解码插件打包.....	236
5.5.3.2.4 编解码插件质检.....	239
5.5.3.2.5 编解码插件包离线签名.....	242
5.5.4 附录：插件开发说明.....	244
5.5.4.1 接口说明.....	244
5.5.4.1.1 decode 接口说明.....	244
5.5.4.1.2 encode 接口说明.....	246
5.5.4.1.3 getManufacturerId 接口说明.....	249
5.5.4.1.4 getModel 接口说明.....	249
5.5.4.1.5 接口实现注意事项.....	249
5.5.4.2 编解码插件的输入/输出格式.....	253
5.5.4.3 实现样例讲解.....	255
5.5.5 附录：JDK 支持的加密算法.....	260
5.6 CA 证书.....	261
<b>6 设备集成指导.....</b>	<b>270</b>
6.1 NB-IoT 模组及终端应用指导.....	270
6.1.1 前言.....	270
6.1.1.1 概述.....	270
6.1.1.2 使用范围.....	270
6.1.2 NB-IoT 终端硬件设计.....	270
6.1.2.1 NB-IoT 应用系统架构.....	270
6.1.2.2 NB-IoT 终端产品硬件系统介绍.....	270
6.1.2.3 NB 模组硬件介绍.....	271
6.1.2.3.1 模组单 Boudica 方案.....	271
6.1.2.3.2 硬件约束.....	272
6.1.2.4 模组使用硬件接口设计.....	273
6.1.2.4.1 模组供电.....	273
6.1.2.4.2 复位设计.....	273
6.1.2.4.3 串口设计.....	273
6.1.2.4.4 USIM 卡.....	274
6.1.2.4.5 看门狗.....	274

6.1.2.4.6 Jlink.....	274
6.1.2.5 终端产品硬件设计.....	274
6.1.2.5.1 终端产品硬件要求.....	274
6.1.2.5.2 Device 产品硬件单板 PCB 设计.....	274
6.1.2.5.3 终端产品射频接收灵敏度.....	275
6.1.2.5.4 终端产品现网覆盖要求.....	275
6.1.2.6 终端产品调测.....	275
6.1.3 天线设计.....	276
6.1.3.1 设计关注点.....	276
6.1.3.2 NB 终端天线参考规格.....	276
6.1.3.3 NB 终端天线样例.....	277
6.1.3.3.1 单频单极子天线.....	277
6.1.3.3.2 PIFA 天线.....	278
6.1.3.3.3 PCB 天线.....	279
6.1.3.3.4 外置胶棒天线.....	280
6.1.3.4 NB 终端天线应用示例.....	281
6.1.3.4.1 Case1 水表+单极子天线.....	281
6.1.3.4.2 Case2 路灯+外置胶棒天线.....	281
6.1.3.4.3 Case3 电表+PIFA 天线.....	281
6.1.3.4.4 Case4 气表+FPC 天线.....	282
6.1.3.5 天线设计部分原则及建议.....	282
6.1.3.5.1 PCB 天线设计原则.....	282
6.1.3.5.2 PCB 板上匹配电路预留.....	282
6.1.3.5.3 内置天线对于终端整体设计的通用要求.....	283
6.1.3.5.4 射频板上外置天线接口预留.....	283
6.1.3.6 注意事项.....	283
6.1.4 电池选型.....	283
6.1.4.1 电池方案设计步骤.....	284
6.1.4.2 NB 模组供电要求.....	284
6.1.4.3 Case1.....	285
6.1.4.4 Case2.....	286
6.1.5 软件特性设计.....	288
6.1.5.1 设计应用约束条件.....	289
6.1.5.2 软件架构.....	290
6.1.5.3 LWM2M Server 对接软件.....	291
6.1.5.3.1 注册流程.....	291
6.1.5.3.2 对象操作.....	291
6.1.5.3.3 软件升级流程.....	292
6.1.5.3.4 终端设备管理软件.....	293
6.1.6 FOTA 升级特性设计.....	294
6.1.6.1 FOTA 特性概述.....	294
6.1.6.1.1 OTA 简介.....	294

6.1.6.1.2 系统整体架构.....	294
6.1.6.2 FOTA 升级与 MCU 交互.....	294
6.1.6.2.1 FOTA 升级时序处理.....	294
6.1.6.2.2 FOTA 升级对 MCU 的约束.....	295
1. 下载阶段.....	295
2. 升级阶段.....	295
3. 恢复网络阶段.....	296
6.1.6.3 FOTA 特性设计使用约束.....	296
6.1.6.4 设备侧 MCU 软件 FOTA 适配建议.....	296
6.1.6.4.1 设备 MCU 和模组串口通讯方式.....	297
6.1.6.4.2 设备 MCU 和模组串口中断方式通讯说明.....	297
6.1.7 可靠性设计.....	298
6.1.7.1 控制面防挂死设计.....	298
6.1.7.2 数据面防挂死设计.....	299
6.1.8 典型案例.....	300
6.1.8.1 模组 PSM 态功耗超标问题排查.....	300
6.1.8.2 终端近端维护串口.....	300
6.1.8.3 终端覆盖性能排除方法.....	301
6.1.9 修订记录.....	301
6.2 LiteOS SDK 集成开发指导.....	301
6.2.1 LiteOS SDK 端云互通组件概述.....	301
6.2.1.1 背景介绍.....	301
6.2.1.2 系统方案.....	304
6.2.1.3 集成策略.....	305
6.2.1.3.1 可集成性.....	305
6.2.1.3.2 可移植性.....	306
6.2.1.3.3 集成约束.....	307
6.2.1.4 安全.....	307
6.2.1.5 升级.....	308
6.2.2 设备接入 OceanConnect 集成开发流程.....	308
6.2.2.1 云侧配置流程.....	309
6.2.2.1.1 环境准备.....	309
6.2.2.1.2 创建应用.....	309
6.2.2.1.3 开发 Profile 文件.....	311
6.2.2.1.4 开发编解码插件.....	313
6.2.2.1.5 注册设备.....	317
6.2.2.2 端侧对接流程.....	319
6.2.2.2.1 环境准备.....	319
6.2.2.2.2 LiteOS SDK 端云互通组件入口函数.....	322
6.2.2.2.3 LiteOS SDK 端云互通组件初始化.....	322
6.2.2.2.4 创建数据上报任务.....	324
6.2.2.2.5 LiteOS SDK 端云互通组件命令处理接口.....	325

6.2.2.2.6 LiteOS SDK 端云互通组件主函数体.....	326
6.2.2.2.7 数据结构介绍.....	327
6.2.2.3 小结.....	329
6.2.3 LiteOS 端云互通组件实战演练.....	329
6.2.3.1 开发环境准备.....	329
6.2.3.2 (参考) 端云互通组件以太网接入实例.....	330
6.2.3.2.1 接入 IoT 平台.....	330
6.2.3.2.2 数据上报.....	332
6.2.3.2.3 命令下发.....	333
6.2.3.3 (参考) 端云互通组件无线接入实例.....	337
6.2.3.3.1 无线接入介绍.....	337
6.2.3.3.2 AT 框架介绍.....	337
6.2.3.3.3 移植 WIFI 模块-ESP8266.....	338
6.2.3.3.4 移植 GSM 模块-SIM900A.....	341
6.2.3.3.5 注意事项.....	343
6.2.3.4 (参考) 设备模拟器接入平台.....	343
6.2.3.4.1 设备模拟器接入平台.....	343
6.2.3.4.2 设备模拟器数据上报.....	344
6.2.3.4.3 应用模拟器命令下发.....	346
6.2.4.4 LwM2M 协议介绍.....	347
6.2.4.1 LwM2M 协议是什么.....	347
6.2.4.2 LwM2M 协议特性.....	347
6.2.4.3 LwM2M 体系架构.....	347
6.2.4.4 LwM2M 对象定义.....	348
6.2.4.5 LwM2M 资源定义.....	349
6.2.4.6 LwM2M 接口定义.....	350
6.2.4.7 固件升级.....	354
6.3 Agent Lite SDK 集成开发指导.....	358
6.3.1 直连场景.....	358
6.3.2 非直连场景.....	358
6.3.3 C-Linux.....	358
6.3.3.1 开发者必读.....	359
6.3.3.2 前期准备.....	359
6.3.3.2.1 获取物联网平台对接信息.....	359
6.3.3.2.2 下载开发资源.....	359
6.3.3.2.3 证书校验和权限控制.....	359
6.3.3.3 开发环境.....	360
6.3.3.4 交叉编译环境检测.....	360
6.3.3.5 导入样例代码.....	362
6.3.3.6 初始化.....	364
6.3.3.7 绑定.....	364
6.3.3.8 登录.....	366

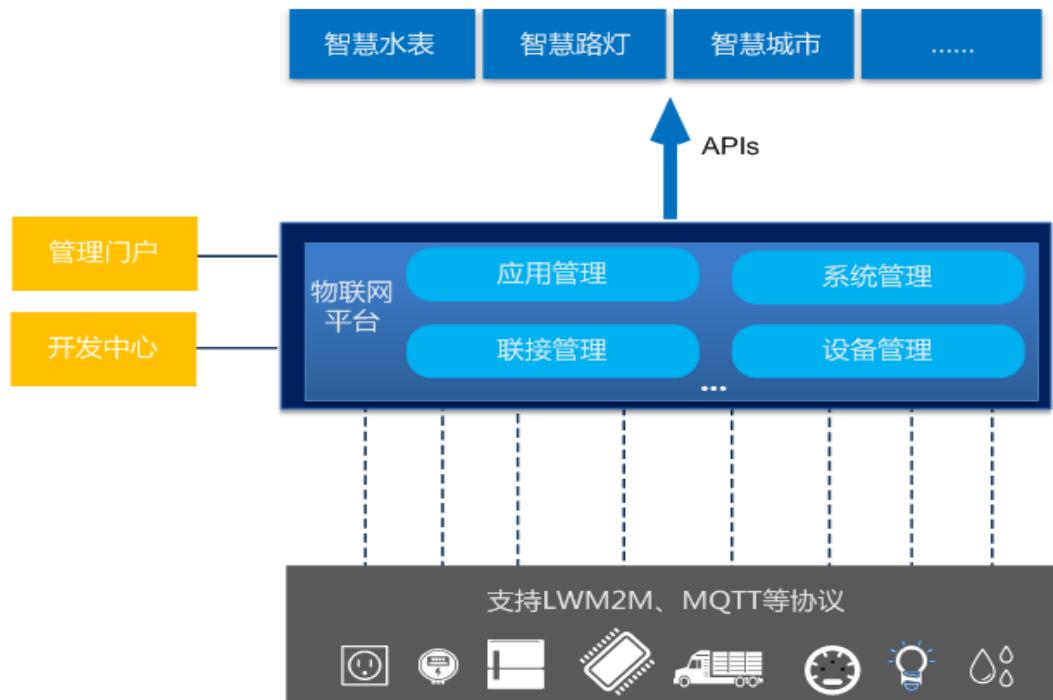
6.3.3.9 编译并运行程序.....	366
6.3.3.10 上传 Profile 并注册设备.....	367
6.3.3.11 上线.....	369
6.3.3.12 数据上报和数据发布.....	369
6.3.3.12.1 使用说明.....	370
6.3.3.12.2 设备服务数据上报.....	370
6.3.3.12.3 数据发布.....	371
6.3.3.13 命令接收和数据接收.....	371
6.3.3.13.1 使用说明.....	371
6.3.3.13.2 设备命令接收.....	371
6.3.3.13.3 数据接收.....	372
6.3.3.14 添加非直连设备.....	373
6.3.3.15 非直连设备状态更新.....	374
6.3.3.16 非直连设备数据上报.....	374
6.3.3.17 附录.....	375
6.3.3.17.1 流程图.....	375
1. 绑定和登录.....	375
2. 添加非直连设备和设备状态更新.....	376
3. 数据上报和命令接收.....	376
6.3.3.17.2 Topic 的订阅.....	376
6.3.4 Java.....	377
6.3.4.1 开发者必读.....	377
6.3.4.2 前期准备.....	377
6.3.4.2.1 获取物联网平台对接信息.....	377
6.3.4.2.2 下载开发资源.....	377
6.3.4.2.3 证书校验和权限控制.....	378
6.3.4.3 开发环境.....	378
6.3.4.4 导入样例代码.....	379
6.3.4.5 初始化.....	380
6.3.4.6 绑定.....	380
6.3.4.7 登陆.....	381
6.3.4.8 上传 Profile 并注册设备.....	382
6.3.4.9 上线.....	384
6.3.4.10 数据上报和数据发布.....	384
6.3.4.10.1 使用说明.....	384
6.3.4.10.2 设备服务数据上报.....	385
6.3.4.10.3 数据发布.....	385
6.3.4.11 命令接收和数据接收.....	386
6.3.4.11.1 使用说明.....	386
6.3.4.11.2 设备命令接收.....	386
6.3.4.11.3 数据接收.....	386
6.3.4.12 添加非直连设备.....	387

6.3.4.13 非直连设备状态更新.....	387
6.3.4.14 非直连设备数据上报.....	388
6.3.4.15 附录.....	389
6.3.4.15.1 流程图.....	389
1. 绑定和登录.....	389
2. 添加非直连设备和设备状态更新.....	390
3. 数据上报和命令接收.....	390
6.3.4.15.2 Topic 的订阅.....	391
<b>7 应用侧 SDK 使用指南.....</b>	<b>392</b>
<b>7.1 JAVA SDK 使用指南.....</b>	<b>392</b>
7.1.1 开发者必读.....	392
7.1.2 开发环境要求.....	392
7.1.3 下载相关开发资源.....	392
7.1.4 导入 Demo 工程.....	393
7.1.5 初始化及证书配置.....	395
7.1.6 业务接口调用方法.....	396
7.1.7 回调接口实现.....	397
7.1.8 回调证书制作、配置及上传.....	397
7.1.9 业务接口调用流程及注意事项.....	405
7.1.10 SDK 独立运行测试.....	406
<b>7.2 PHP SDK 使用指南.....</b>	<b>407</b>
7.2.1 开发者必读.....	407
7.2.2 开发环境要求.....	407
7.2.3 下载相关开发资源.....	407
7.2.4 导入 Demo 工程.....	408
7.2.5 初始化及证书配置.....	409
7.2.6 业务接口调用方法.....	409
7.2.7 回调接口实现.....	410
7.2.8 回调证书制作、配置及上传.....	411
7.2.9 业务接口调用流程及注意事项.....	418
<b>7.3 Python SDK 使用指南.....</b>	<b>420</b>
7.3.1 开发者必读.....	420
7.3.2 开发环境要求.....	420
7.3.3 下载相关开发资源.....	420
7.3.4 导入 Demo 工程.....	421
7.3.5 初始化及证书配置.....	422
7.3.6 业务接口调用方法.....	423
7.3.7 回调接口实现.....	424
7.3.8 回调证书制作、配置及上传.....	425
7.3.9 业务接口调用流程及注意事项.....	432

# 1 新手入门

## 1.1 平台简介

物联网平台包括应用管理、设备管理、系统管理等能力，实现统一安全的网络接入、各种终端的灵活适配、海量数据的采集分析，从而实现新价值的创造。物联网平台不仅可以简化各类终端厂家的开发，屏蔽各种复杂设备接口，实现终端设备的快速接入；同时面向各行业提供强大的开放能力，支撑各行业伙伴快速实现各种物联网业务应用，满足各行业客户的个性化业务需求。



## 1.2 相关概念

- **开发中心**

开发中心是基于物联网平台开放能力的一站式开发工具，帮助开发者快速构建基于物联网平台的解决方案。



- **项目**

项目指物联网平台的资源空间。开发者在基于开发中心进行物联网开发时，需要根据行业属性创建独立的项目，并在该项目空间内建设物联网产品和应用。

- **产品**

某一类具有相同能力或特征的设备的集合称为一款产品。除了设备实体，产品还包含该类设备在物联网能力建设中产生的产品信息、产品模型（Profile）、插件、测试报告等资源。

- **产品模型**

产品模型（也称Profile）用于描述设备具备的能力和特性。开发者通过定义Profile，在物联网平台构建一款设备的抽象模型，使平台理解该款设备支持的服务、属性、命令等信息。

- **SDK (Software Development Kit)**

软件开发工具包，是一些被软件工程师用于为特定的软件包、软件框架、硬件平台、操作系统等创建应用软件的开发工具集合。一般而言，SDK即开发Windows平台下的应用程序所使用的SDK。它可以简单的为某个程序设计语言提供应用程序接口的一些文件，但也可能包括能与某种嵌入式系统通讯的复杂的硬件。

我们会为开发者提供应用侧SDK和设备侧SDK，帮助开发者快速实现应用或设备与物联网平台的集成对接。

- **MQTT (Message Queue Telemetry Transport)**

MQTT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。

MQTT指MQTT+SSL/TLS，在MQTT中使用SSL/TLS协议进行加密传输。

- **CoAP (Constrained Application Protocol)**

受约束的应用协议（CoAP）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 (lightweight Machine to Machine)**

LWM2M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。

## 1.3 前期准备

### 获取物联网平台对接信息

请参考[业务使用全流程](#)完成入驻华为云和开通开发中心服务，在开通开发中心后，在管理控制台会出现开发中心的入口，且系统将会通过短信和邮件发送开发中心的账号和密码给您。登录开发中心后，可以在相应项目的“对接信息”界面，查看应用和设备的接入地址。



## 获取相关证书

在应用服务器对接物联网平台和设备对接物联网平台的部分场景中，需要在应用服务器侧和设备侧集成相应证书（[点击获取](#)），证书包的目录结构如下：

### 说明

此证书包只用于与华为云有云IoT平台的对接。

证书包名称	一级目录	二级目录	三级目录	说明
certificate	Northbound API	code	Java	该目录下的证书在应用服务器通过HTTPS协议调用物联网平台接口时使用。请根据应用服务器侧的编程语言选择相应目录下的证书文件，并置于应用服务器侧。
			PHP	
			Python	
	Agent Lite	postman	-	该目录下的证书在postman通过HTTPS协议调试物联网平台接口时使用。
		Android	-	该目录下的证书在终端设备或网关通过集成Agent Lite SDK接入物联网平台时使用。请根据终端设备或网关侧的编程语言选择相应目录下的证书文件，并置于终端设备或网关侧。
		C-Linux	-	
		Java	-	

## 求助方式

如果您在开发过程中遇到任何问题，有如下几种求助方式：

- 在华为云社区[物联网平台论坛](#)查找答案或发贴求助。

图 1-1 华为云社区物联网平台论坛



- 发邮件到[iotcloud@huawei.com](mailto:iotcloud@huawei.com)进行咨询。

## 1.4 快速入门

### 1.4.1 开始前必读

快速入门将基于实例，指导开发者快速构建简单的物联网方案，在实践过程中，帮助开发者理解物联网端到端开发过程和开发中心发挥的作用。

#### 前期准备

已经从物联网平台服务商获得开发中心的访问信息。

#### 开发指引

##### 阶段一：创建项目

在进行开发之前，开发者需要基于行业属性，创建一个独立的项目。在项目空间内，开发者可以开发相应的物联网产品和应用。

项目创建操作请参考[项目创建](#)。

##### 阶段二：开始开发

- 快速体验基于物联网平台的物联网开发，请参考[案例1：智能门锁、案例2：补光灯](#)。
- 深入学习基于物联网平台的物联网开发，请参考[设备开发](#)和[应用开发](#)。

##### 阶段三：提交审核

在通过开发中心的自助测试后，开发者可以在线提交申请，让物联网平台服务商审核产品是否满足发布标准。

## 1.4.2 案例 1：智能门锁

### 方案概述

本文档以开发一款门锁产品和应用为例，帮助开发者快速了解基于CoAP接入协议的端到端集成开发流程。

该款门锁通过NB-IoT网络接入物联网平台，且具备如下能力：

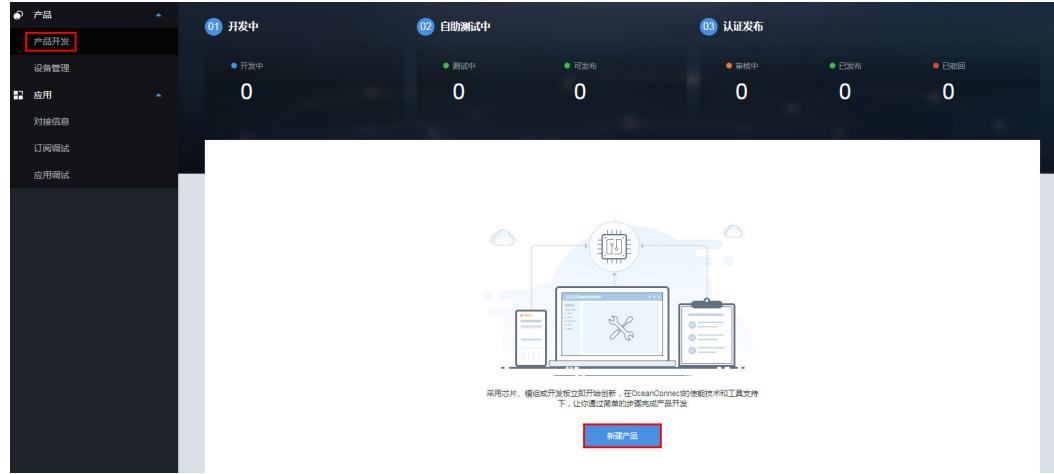
- 支持在开门时上报消息。
- 支持上报门锁工作情况，比如电量、信号强度等。
- 支持修改门锁密码的控制命令。
- 支持修改网络信息的控制命令。

该款门锁的产品能力模型如下表所示：

产品信息			
	设备类型	DoorLock	
	设备型号	DoorLock001	
	厂商名称	TestManuName	
	接入协议	CoAP	
	数据格式	二进制码流	
服务数据			
	服务1		<b>BusinessService</b>
		属性1	时间戳 (timeStamp) 数据类型: int
		属性2	用户ID (userId) 数据类型: int
		命令	修改密码 (CHANGE_PWD) 命令参数: newPassword (string) 命令响应: newPassRsp (0: 成功; 1: 失败)
	服务2		<b>NonBussinessService</b>
		属性	电量 (battery) 数据类型: int
		命令	修改网络 (CHANGE_NET) 命令参数: newNet (String) 命令响应: newNetRsp (0: 成功; 1: 失败)

## 产品开发

**步骤1** 在项目空间内，选择“产品 > 产品开发”，新建一款产品。



在“创建产品”界面，选择“自定义产品”，点击“+”按钮，使用自定义产品的方式创建产品。



在“完善产品信息”窗口，完成各个参数的配置后，点击“创建”。

配置项	取值
产品名称	DoorLock
型号	DoorLock001
厂商ID	系统自动生成
所属行业	智慧生活
设备类型	DoorLock
接入应用层协议类型	CoAP
数据格式	二进制码流

### 设置产品信息

×

\* 产品名称 : DoorLock

\* 型号 : DoorLock001

\* 厂商ID : e08ac2b7703849c784cb0ae23e0283c5

\* 所属行业 : 智慧生活

\* 设备类型 : DoorLock

\* 接入应用层协议类型 : CoAP

注意 : CoAP协议的设备需要完善数据解析，将设备上报的二进制数据转换为平台上的JSON数据格式

\* 数据格式 : 二进制码流

产品图片 : 20180912-155311(eSpace).png 



创建 取消

**步骤2** 在该款门锁产品的开发空间，选择“Profile定义”，根据该款门锁产品的能力模型定义Profile。

The screenshot shows the 'Profile Definition' step of the development process. It lists two services: 'BusinessService' and 'NonBusinessService'. Each service has an 'Attribute List' and a 'Command List'.

- BusinessService:**
  - Attribute List:** timeStamp (int, 0~128, .., .., checked, RE)
  - Attribute List:** userId (int, 0~128, .., .., checked, RE)
- Command List:** CHANGE\_PWD
  - 下发命令字段:** newPassword (string, 10, .., .., checked)
  - 响应命令字段:** newPassRsp (int, 0~1, .., .., checked)

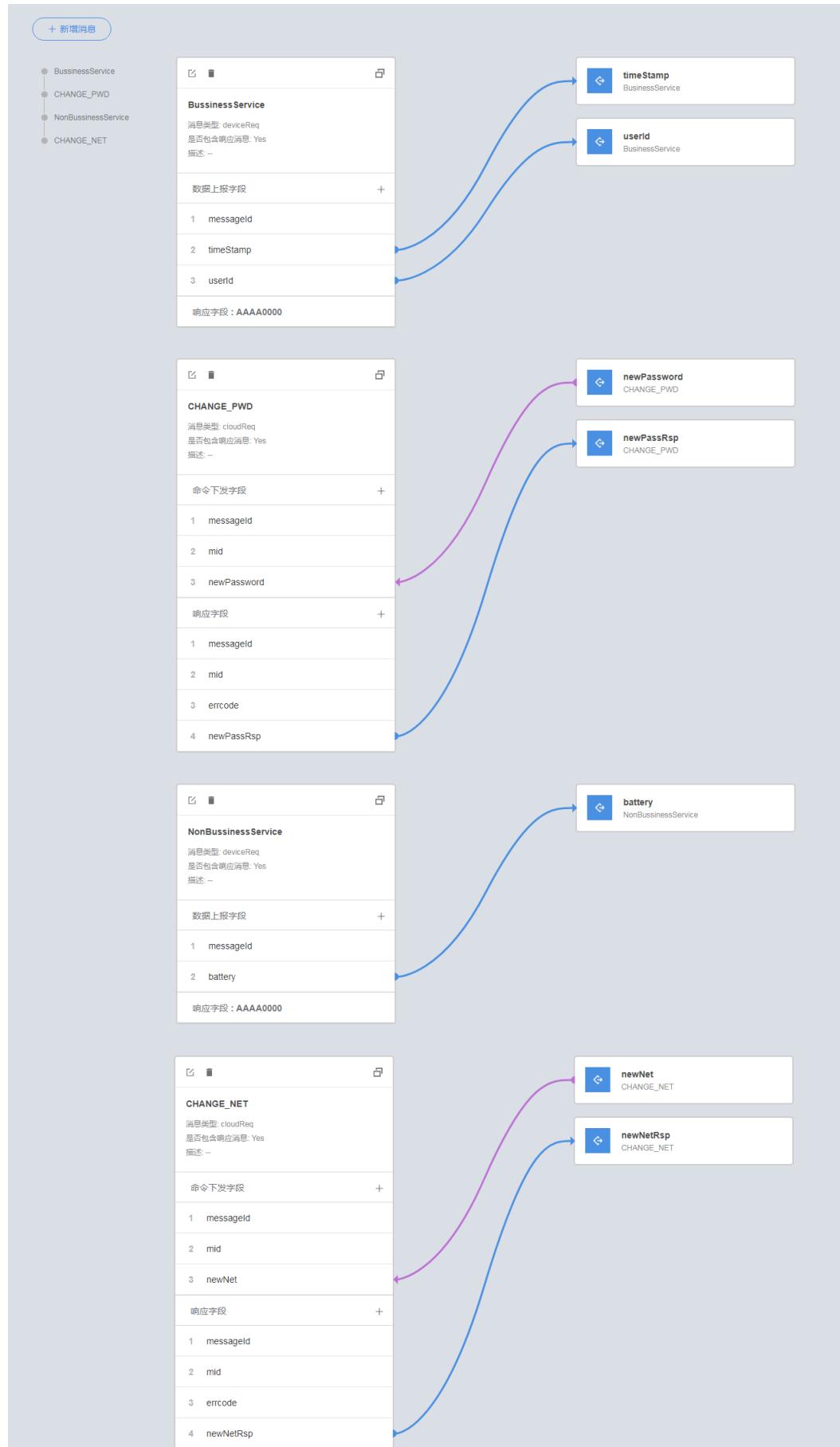
  

- NonBusinessService:**
  - Attribute List:** battery (int, 0~100, .., .., checked, RE)
- Command List:** CHANGE\_NET
  - 下发命令字段:** newNet (string, 30, .., .., checked)
  - 响应命令字段:** newNetRsp (int, 0~1, .., .., checked)

**步骤3** 在该款门锁产品的开发空间，选择“编解码插件开发”，根据定义的Profile进行插件开发。

- messageId，地址域字段，当存在相同类型的消息时，需要定义此字段，以便于编解码插件对消息进行区分。在本案例中，数据上报类型的消息有“BusinessService”和“NonBusinessService”两种，命令下发类型的消息有“CHANGE\_PWD”和“CHANGE\_NET”两种，因此均定义了地址域字段。
- mid，响应标识字段，当设备在接到命令后返回命令执行结果时，需要定义此字段，以便于编解码插件将命令下发消息和对应的命令响应消息进行关联。在本案例中，门锁在接到命令后会返回命令执行结果，因此在命令下发和命令响应中定义了此字段。





#### 步骤4 门锁设备集成NB-IoT模组，通过NB-IoT网络接入物联网平台。

模组厂商有移远、利尔达等，开发者需要向模组厂商购买模组，并获取相关的集成开发指导。另外，开发者还需要向运营商购买SIM卡，使门锁设备可以接入NB-IoT网络。

门锁设备集成模组后，通过一系列AT命令控制NB-IoT模组，具体AT命令以模组厂商提供的信息为准，后续步骤中的AT命令仅供参考。

##### 说明

在设备还未就绪的情况下，如果开发者需要进行业务调试，则可以通过串口工具执行AT命令，对NB-IoT模组进行控制，模拟设备接入的场景。

#### 步骤5 在该款门锁产品的开发空间，选择“在线调测”。



新增测试设备时，选择“有真实的物理设备”，“设备标识”填写IMEI号码，“验证码加密”根据设备的实际情况进行配置：选择不加密则表示设备使用CoAP/UDP协议接入物联网平台；选择加密则表示设备使用CoAPS/DTLS协议接入物联网平台。

A screenshot of the 'Add Test Device' dialog box. At the top left is the title '新增测试设备'. On the right is a close button 'X'. Below the title, the text '您现在' is followed by two radio buttons: '有真实的物理设备' (selected) and '没有真实的物理设备'. The next section is labeled '\*设备名称' (Device Name) with a text input field containing 'testdevice001'. The next section is labeled '\*设备标识' (Device Identifier) with a text input field containing '1234567890'. Below these fields are two radio buttons: '不加密' (selected) and '加密'. At the bottom are two buttons: a blue '创建' (Create) button and a white '取消' (Cancel) button.

设备创建成功后，将返回“设备ID”和“PSK码”。如果设备使用DTLS协议接入物联网平台，请妥善保存PSK码。

## 设备创建成功

X

请根据设备指导说明书为设备接通电源，配置好网络，开启设备，观察设备是否成功接入到平台，如果状态是在线（online）表示设备已经成功的接入到平台，接着就可以接收设备的数据。以下是您的设备信息，请牢记！

设备ID

bd331fb2-9443-4776-baac-70f381783be0

PSK码（使用DTLS协议时需要使用到该psk码，请您牢记！）

105a06ace1c388e1ef1c78eb79edf5f5

确定

**步骤6** 门锁设备接入物联网平台（[步骤4](#)的系列AT命令），新增加的测试设备由“离线”状态变为“在线”状态。

The screenshot shows a progress bar at the top with four steps: 01 Profile定义, 02 SDK集成, 03 编解码插件开发, and 04 在线调测 (Online Debugging), with the fourth step highlighted in blue. Below the progress bar is a table titled '设备列表' (Device List) with the following columns: 状态 (Status), 设备名称 (Device Name), 设备ID (Device ID), 所属产品 (Product), 型号 (Model), 设备类型 (Device Type), and 操作 (Operations). A single row is shown for 'testdevice001', which is marked as '在线' (Online) with a blue circle. The device ID is bd331fb2-9443-4776-baac-70f381783be0, it belongs to 'DoorLock' product, model 'DoorLock001', and type 'DoorLock'. There is a red box around the '在线' status indicator.

**步骤7** 门锁设备上报数据（AT+NMGS=x,xxxx：第1个参数为字节数，第2个参数为上报的十六进制业务码流），场景示例：

门锁被打开，上报开门信息。上报的十六进制码流为：000102；对应的AT命令为：AT+NMGS=3,000102。

字段含义	messageId	timeStamp	userId
码流字段	00	01	02

### 说明

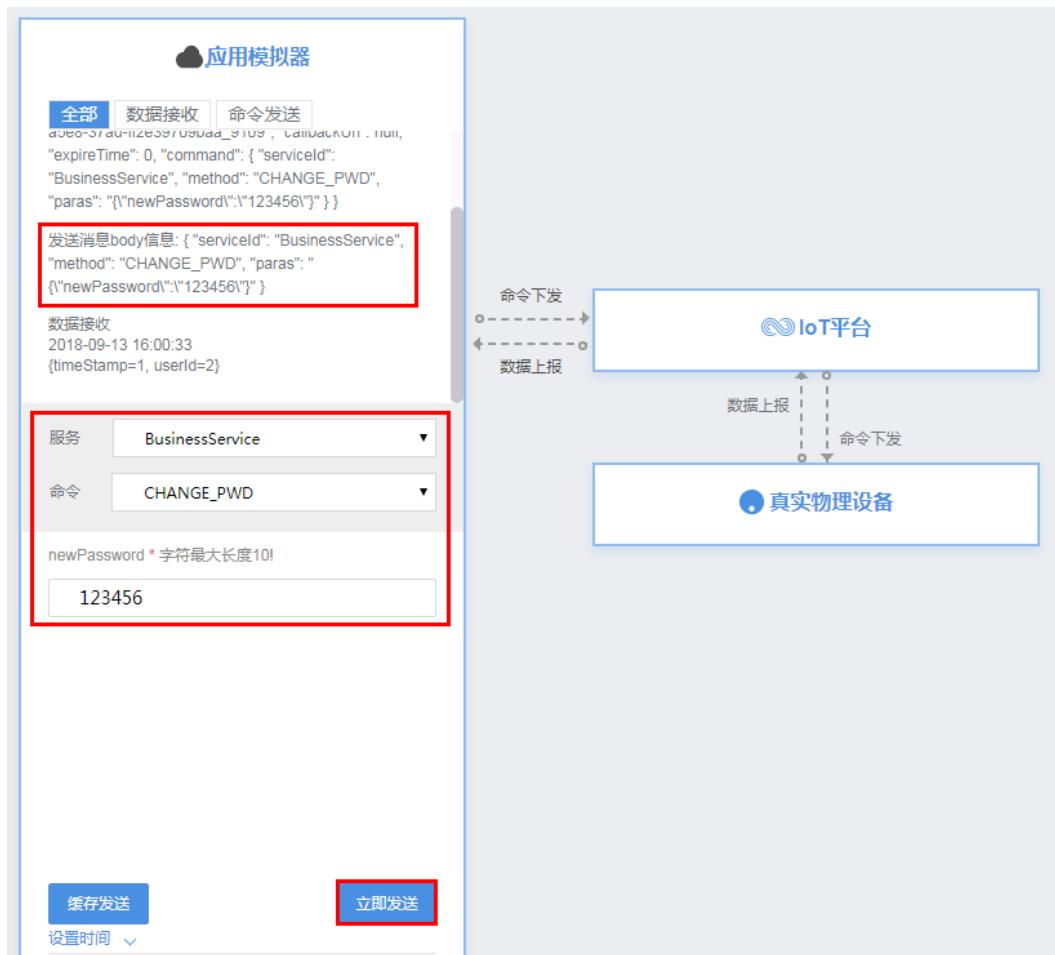
本案例进行编解码插件开发时，BussinessService消息中的messageId字段默认值为00。

进入新增测试设备的调测界面，查看“应用模拟器”接收的经过编解码插件解析后的JSON格式数据：{timeStamp=1, userId=2}。



### 步骤8 应用侧下发命令，场景示例：用户需要将密码修改为123456。

进入新增测试设备的调测界面，使用“应用模拟器”向门锁设备下发修改密码的命令：{ "serviceId": "BusinessService", "method": "CHANGE\_PWD", "paras": "{\"newPassword\":\"123456\""} }。



进入新增测试设备的详情界面，选择“历史命令”，查看命令下发状态：“已送达”。

The screenshot shows the IoT Platform's device management interface. At the top, there is a navigation bar with four steps: 01 Profile定义, 02 SDK集成, 03 编解码插件开发, and 04 在线调测 (Online Debugging), followed by a blue button labeled '发起自助测试' (Initiate Self-test). Below the navigation bar is a table titled '设备列表' (Device List) with one entry: 'testdevice001'. The table columns include: 状态 (Status), 设备名称 (Device Name), 设备ID (Device ID), 所属产品 (Product), 型号 (Model), 设备类型 (Device Type), and 操作 (Operations). The '操作' column for this device has a red box around it. Below the table, there is a tab bar with five items: 设备详情 (Device Details), 历史数据 (History Data), 设备日志 (Device Log), 历史命令 (History Command), and a fourth item which is also highlighted with a red box. Under the '历史命令' tab, there is a table with columns: 状态 (Status), 命令ID (Command ID), 命令创建时间 (Command Creation Time), 命令内容 (Command Content), 刷新 (Refresh), and 命令响应 (Command Response). One row in this table is highlighted with a red box, showing the status as '已送达' (Delivered).

门锁设备可以查询收到的命令消息：010001313233343536；对应的AT命令为：AT +NMGR。

字段含义	messageId	mid	newPassword
码流字段	01	0001	313233343536

### 说明

- 本案例进行编解码插件开发时，CHANGE\_PWD命令的messageId字段默认值为01。
- mid字段长度固定为“16位无符号整型”，取值由系统自动生成。
- newPassword字段的取值“313233343536”为字符串“123456”对应的十六进制码流。

门锁设备上报命令执行结果：0200010000；对应的AT命令为：AT +NMGS=5,0200010000。

字段含义	messageId	mid	errorcode	newPassRsp
码流字段	02	0001	00	00

### 说明

本案例进行编解码插件开发时，newPassRsp响应的messageId字段默认值为00。

进入新增测试设备的详情界面，选择“历史命令”，查看命令下发状态：“执行成功”。

The screenshot shows the IoT Platform's device management interface. It is similar to the previous one, but the status for the device 'testdevice001' in the '操作' column is now '执行成功' (Execution successful), indicated by a red box.

**步骤9** 在该款门锁产品的开发空间，点击“发起自助测试”，然后根据向导完成预检查和各项测试用例。



----结束

## 应用开发

**步骤1** 在项目空间内，选择“应用 > 对接信息”，在设备接入方式区域查看应用接入信息。



**步骤2** 根据[北向API参考](#)在应用服务器完成接口调用开发，[北向JAVA API Demo](#)中提供了常用接口调用的代码样例，供开发者参考。

**步骤3** 选择“应用 > 应用调试”，点击“使用虚拟设备”。



在系统弹出的“新建虚拟设备”窗口，选择该款门锁产品，完成虚拟设备的创建。



#### 步骤4 使用虚拟设备进行数据上报，场景示例：

门锁被打开，上报开门信息。上报的十六进制码流为：000102。

字段含义	messageId	timeStamp	userId
码流字段	00	01	02

#### 说明

本案例进行编解码插件开发时，BussinessService消息中的messageId字段默认值为00。

在“应用调试”界面，选择新创建的虚拟设备，使用“设备模拟器”上报十六进制码流：000102。

在回调地址查看物联网平台推送的数据：应用服务器需要调用[订阅平台业务数据](#)接口，物联网平台才会根据订阅的通知类型，向回调地址推送数据。如果回调地址为HTTPS地址，则需要在物联网平台[上传CA证书](#)。



**步骤5** 在应用服务器进行命令下发，场景示例：用户需要将密码修改为123456。

应用服务器通过调用[创建设备命令](#)接口，下发修改密码的命令：{ "serviceId": "BusinessService", "method": "CHANGE\_PWD", "paras": "{\"newPassword\": \"123456\"}" }。

在“应用调试”界面的“设备模拟器”区域，查看虚拟设备接收到的数据：010001313233343536。

字段含义	messageId	mid	newPassword
码流字段	01	0001	313233343536

#### 说明

- 本案例进行编解码插件开发时，CHANGE\_PWD命令的messageId字段默认值为01。
- mid字段长度固定为“16位无符号整型”，取值由系统自动生成。
- newPassword字段的取值“313233343536”为字符串“123456”对应的十六进制码流。



在“应用调试”界面，使用“设备模拟器”上报命令执行结果：0200010000。在命令响应回调地址查看物联网平台推送的数据。

字段含义	messageId	mid	errorcode	newPassRsp
码流字段	02	0001	00	00

### 说明

本案例进行编解码插件开发时，newPassRsp响应的messageId字段默认值为00。



----结束

## 1.4.3 案例 2：补光灯

### 方案概述

本文档以开发一款补光灯产品为例，帮助开发者快速了解基于Agent Lite SDK的设备侧集成开发流程。

该款补光灯通过网关接入物联网平台，网关上集成Agent Lite SDK。网关和补光灯具备如下能力：

**网关：**支持上报位置、磁盘空间、内存容量等信息。

**补光灯：**

- 支持上报灯光状态信息。
- 支持开/关控制命令。

补光灯产品（含网关）能力模型如下表所示：

### 说明

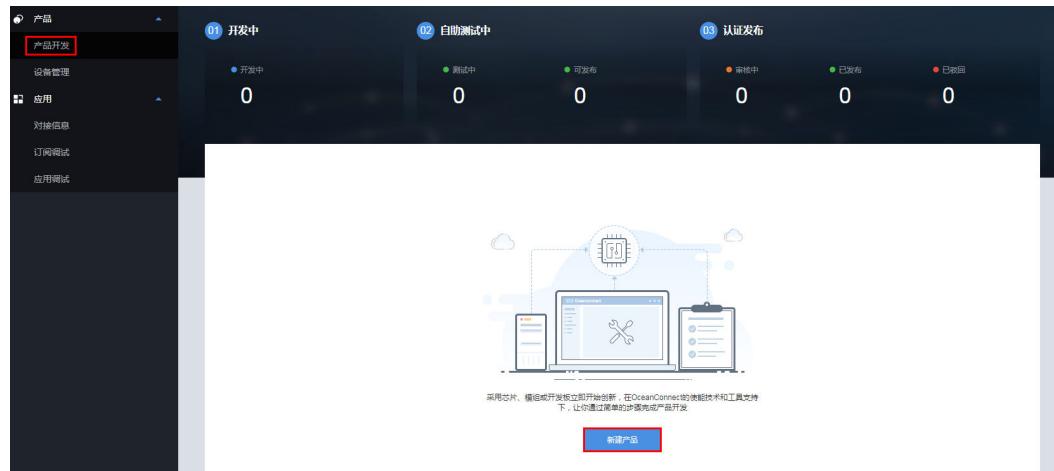
本文档中提及的“补光灯产品”均包含网关。网关集成Agent Lite SDK，与物联网平台直连，补光灯作为网关的子设备，需要通过网关与物联网平台进行数据交互。

产品信息			
------	--	--	--

	设备类型	Bulb	
	设备型号	BuLB001	
	厂商名称	TestManuName	
	接入协议	MQTT	
	数据格式	JSON	
服务数据			
	服务1		<b>Location</b>
		属性1	时间戳 (locationInfo) 数据类型: string
	服务2		<b>GatewayInfo</b>
		属性1	磁盘使用率 (storage) 数据类型: int
		属性2	内存使用率 (memory) 数据类型: int
	服务3		<b>ServiceBulb</b>
		属性	灯光状态 (status) 数据类型: int
		命令	开/关 (ON_OFF) 命令参数: toggleBulb (0: 关掉; 1: 打开) 命令响应: toggleBulbRsp (0: 成功; 1: 失败)

## 产品开发

**步骤1** 在项目空间内，选择“产品 > 产品开发”，新建一款产品。



在“选择产品模板”界面，点击“+”按钮，使用自定义产品模板的方式创建产品。



在“完善产品信息”窗口，完成各个参数的配置后，点击“创建”。

配置项	取值
产品名称	Bulb
型号	Bulb001
厂商ID	系统自动生成
所属行业	智慧生活
设备类型	Bulb
接入应用层协议类型	MQTT
数据格式	JSON

### 设置产品信息

×

* 产品名称 :	Bulb	
* 型号 :	Bulb001	
* 厂商ID :	e08ac2b7703849c784cb0ae23e0283c5	
* 所属行业 :	智慧生活	
* 设备类型 :	Bulb	
* 接入应用层协议类型 :	MQTT	
* 数据格式 :	JSON	
产品图片 :	Bulb.png	↑

×



---

创建 取消

**步骤2** 在该款补光灯产品的开发空间，选择“Profile定义”，根据该款补光灯产品的能力模型定义Profile。

The screenshot shows the development space interface. At the top, there are three steps: 01 Profile定义, 02 SDK集成, and 03 在线调测, with 02 highlighted. Below the steps is a navigation bar with tabs: +新建服务, 导出Profile, 和导入Profile. The main area displays three service configurations:

- Location**: Last modified: 2018/09/15 12:12:25. It has one attribute: locationInfo (string type, length 1, unit --, required checked, access mode RE). A command list is present but empty.
- GatewayInfo**: Last modified: 2018/09/15 12:12:25. It has two attributes: storage (int type, range 0~100, step 1, unit --, required checked, access mode RE) and memory (int type, range 0~100, step 1, unit --, required checked, access mode RE). A command list is present but empty.
- ServiceBulb**: Last modified: 2018/09/15 12:12:25. It has one attribute: status (int type, range 0~3, step 1, unit --, required checked, access mode RE). It also includes a command list with an ON\_OFF entry and two response fields: toggleBulb (int type, range 0~1, step 1, unit --, required checked) and toggleBulbRsp (int type, range 0~1, step 1, unit --, required checked).

**步骤3** 在该款补光灯产品的开发空间，选择“SDK集成”，获取适合网关系统的Agent Lite SDK版本。本文档以Linux系统为例进行说明。

The screenshot shows the Agent Lite SDK download page. It lists three options:

- Agent Lite SDK for Linux**: System (内核): Linux(Linux version 3.10.14). Includes download and usage documentation links.
- Agent Lite SDK for Windows**: System (内核): Windows(Windows [version 6.1.7601] x86\_64). Includes download and usage documentation links.
- Agent Lite SDK for Android**: System (内核): Android(Android version 4.0以上). Includes download and usage documentation links.

开发者可以参考[Agent Lite SDK集成开发指南](#)进行Agent Lite SDK集成。

#### 说明

如果开发者在“SDK集成”界面未找到适合设备系统平台的Agent Lite SDK版本，请提供交叉编译工具链给物联网平台服务商，服务商的工程师将为开发者编译适用的Agent Lite SDK版本。

**步骤4** 在该款补光灯产品的开发空间，选择“在线调测”。



新增测试设备时，选择“有真实的物理设备”，“设备标识”填写mac地址，“验证码加密”根据设备的实际情况进行配置：选择不加密则表示设备使用MQTT/UDP协议接入物联网平台；选择加密则表示设备使用MQTT/DTLS协议接入物联网平台。

#### 说明

新增的测试设备为网关设备，补光灯作为网关的子设备，在网关接入物联网平台后，通过调用Agent Lite SDK的接口进行添加。

### 新增测试设备

X

您现在

有真实的物理设备  没有真实的物理设备

\*设备名称

testdevice001

\*设备标识

00:0C:29:00:00:01

不加密  加密

创建

取消

设备创建成功后，将返回“设备ID”和“PSK码”。如果设备使用DTLS协议接入物联网平台，请妥善保存PSK码。

## 设备创建成功

X

请根据设备指导说明书为设备接通电源，配置好网络，开启设备，观察设备是否成功接入到平台，如果状态是在线（online）表示设备已经成功的接入到平台，接着就可以接收设备的数据。以下是您的设备信息，请牢记！

### 设备ID

bd331fb2-9443-4776-baac-70f381783be0

### PSK码（使用DTLS协议时需要使用到该psk码，请您牢记！）

105a06ace1c388e1ef1c78eb79edf5f5

确定

**步骤5** 网关设备接入物联网平台，新增加的测试设备由“离线”状态变为“在线”状态。

The screenshot shows the IoT Platform's device management interface. It has three tabs at the top: 'Profile定义' (Step 1), 'SDK集成' (Step 2), and '在线调测' (Step 3, highlighted in blue). Below the tabs is a table titled '设备列表' (Device List) with columns: 状态 (Status), 设备名称 (Device Name), 设备ID (Device ID), 所属产品 (Product), 型号 (Model), 设备类型 (Device Type), and 操作 (Operations). A red box highlights the '在线' (Online) status of the first row, which corresponds to 'testdevice001'. The device ID is 1f13ce0d-4229-45a0-9f69-cfa49769ed21, it belongs to the 'Bulb' product, model Bulb001, and is of type Bulb. There is also a '+新增测试设备' (Add Test Device) button.

网关依次调用Agent Lite SDK的IOTA\_Bind()、IOTA\_ConfigSetXXX()、LoginService.login()接口，接入物联网平台。

网关依次调用Agent Lite SDK的IOTA\_HubDeviceAdd()、IOTA\_DeviceStatusUpdate()接口，添加补光灯设备。

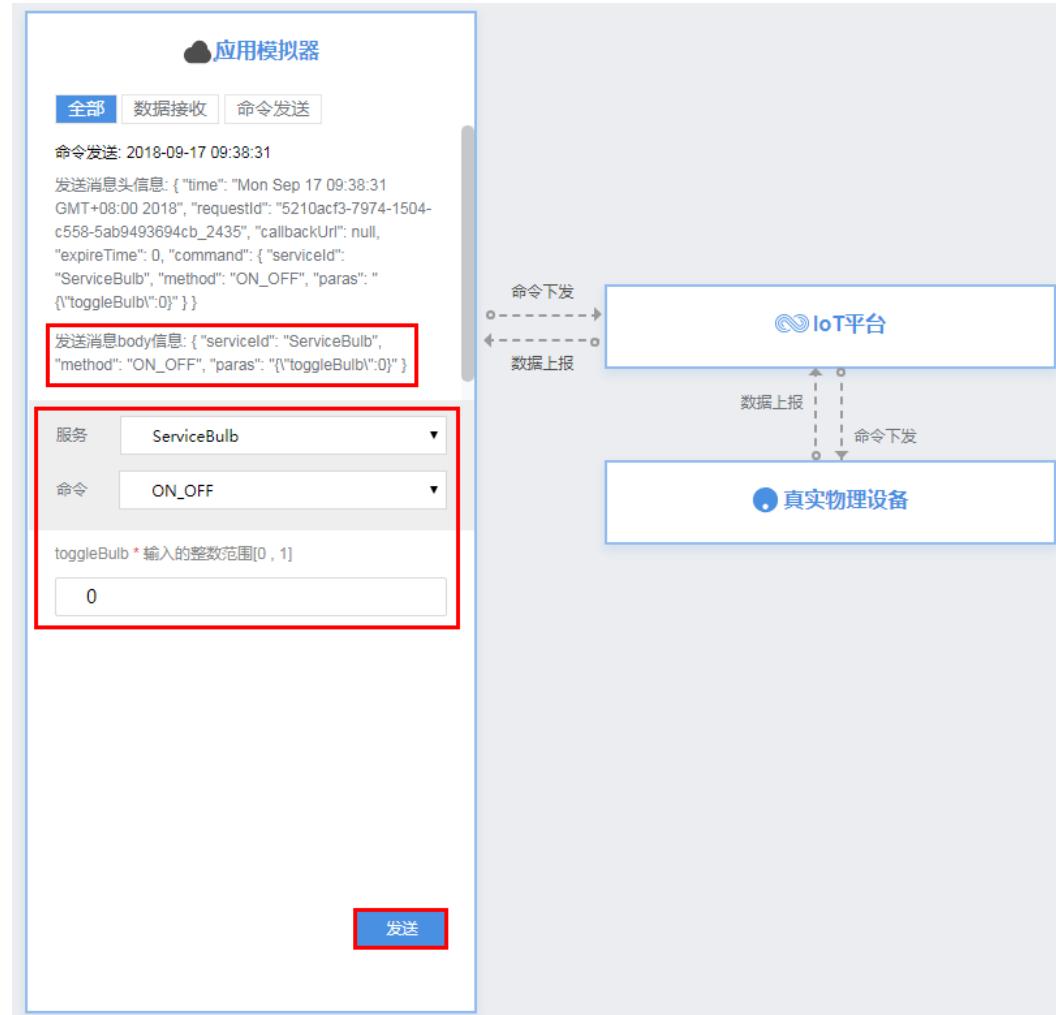
**步骤6** 补光灯设备上报状态数据：补光灯设备的状态数据需要通过网关调用IOTA\_ServiceDataReport()实现上报。

进入新增测试设备的调测界面，查看“应用模拟器”接收的JSON格式数据：{"status": "2"}。



**步骤7 应用侧下发命令：关掉补光灯。**

进入新增测试设备的调测界面，使用“应用模拟器”向补光灯设备下发关灯的命令：  
{ "serviceId": "ServiceBulb", "method": "ON\_OFF", "paras": "{\"toggleBulb\":0}" }。



网关通过调用Agent Lite SDK的IOTA\_TOPIC\_SERVICE\_COMMAND\_RECEIVE/{deviceId}接口，处理物联网平台下发的控制命令。

**步骤8 在该款补光灯产品的开发空间，点击“发起自助测试”，然后根据向导完成预检查和各项测试用例。**

# 2 开发准备

## 2.1 能力要求

物联网开发对开发者的基础能力有一定的要求：

### 应用侧能力要求

- 具备应用系统开发经验，熟悉后台开发的相关操作，比如环境搭建、HTTPS证书制作等。
- 熟悉发布-订阅开发模式，具备根据RESTful API参考调用API接口的能力。

### 设备侧能力要求

- 精通C语言，能独立进行单片机（51、AVR、ARM等）的开发。
- 熟悉嵌入式软件开发，有Keil、IAR、Eclipse等环境开发经验。
- 有嵌入式操作系统（UCOS、FreeRTOS、RT-Thread等）应用经验。

## 2.2 厂商配置

### 概述

厂商信息用于描述公司商标、公司名称、公司网站、公司规模等基本信息。开发者在首次访问开发中心时，需要首先将厂商信息补充完整。

### 配置厂商信息

**步骤1** 在开发中心首页，选择“厂商”，进入厂商信息编辑界面。



**步骤2** 根据公司实际情况，将厂商信息补充完整后，点击“保存”。

主页 > 厂商信息

\* 公司Logo :

图片大小不超过5M, 支持JPG/JPEG/PNG/GIF格式, 建议图片尺寸168\*70 ×

**LOGO**

\* 公司名称(中文) :

\* 公司名称(英文) :

公司网站 :

\* 公司年限 :  年

\* 雇员规模 :

\* 联络人姓名 :

\* 联络人手机号 :

客服电话 :

客服邮箱 :

通讯地址 :

----结束

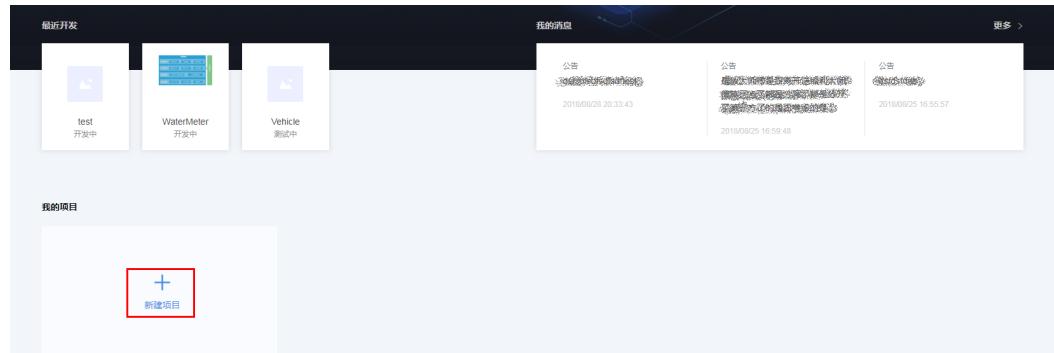
## 2.3 项目创建

### 概述

在进行开发之前，开发者需要基于行业属性，创建一个独立的项目。在项目空间内，开发者可以开发相应的物联网产品和应用。

### 创建项目

**步骤1** 在开发中心首页，点击“新建项目”。



**步骤2** 填写“项目名称”、“所属行业”、“描述”等项目信息后，点击“创建”。

**说明**

“项目名称”需要保持唯一，不可以和其他项目冲突，否则会创建失败。

**新建项目**

X

项目名称 \*

Quick\_Start

所属行业 \*

公用事业 (MC-IoT)

描述

**创建**

项目创建成功后，系统返回“应用ID”和“应用秘钥”。在应用对接物联网平台时需要这两个参数，请妥善保存，如果遗忘，可以在该项目的“应用 > 对接信息 > 应用安全”中进行重置。



### 项目创建成功

我们已为您分配应用ID及密钥，这是您记住密钥的唯一机会，请妥善保管。

如若遗忘密钥，可通过对接信息 > “重置密钥”进行重置。

应用ID

ffhVSWZAoAtvTb4cXrBWLu4NNsa

应用密钥

howol0rTR7y3oDHetjJhvlnTfZYa

确定

**步骤3** 选择新创建的项目，可以进入项目空间。

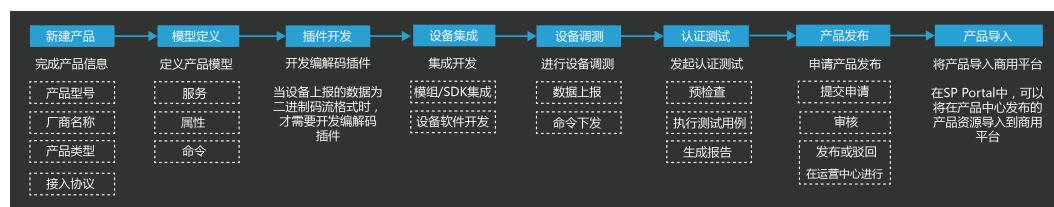
The screenshot shows the IoT Platform Enhanced Edition's main dashboard. In the top left, there's a 'Recent Development' section with three projects: 'test' (status: '开发中'), 'WaterMeter' (status: '开发中'), and 'Vehicle' (status: '测试中'). To the right of this is a 'My Projects' section where a project named 'Quick\_Start' has been highlighted with a red box. This project has a status of '待上线' and was created on '2018/09/02 16:19:56'. Below these sections is a 'Messages' area with several notifications. A large blue button labeled '确定' (Confirm) is centered at the bottom of the screen.

----结束

# 3 设备开发

## 3.1 开发说明

物联网平台支持NB-IoT、2/3/4G、有线网络等多样化的设备接入方式，不同类型的设备要接入到物联网平台并构建互联互通方案，需要经过Profile定义、设备开发、插件开发、测试认证等一系列过程。



## 3.2 LWM2M

### 3.2.1 产品创建

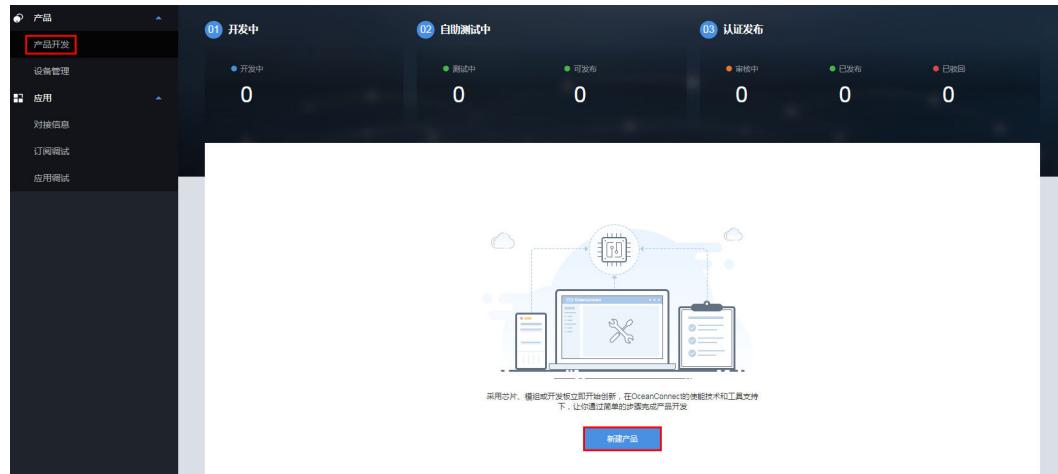
#### 概述

某一类具有相同能力或特征的设备的集合称为一款产品。除了设备实体，产品还包含该类设备在物联网能力建设中产生的产品信息、产品模型（Profile）、插件、测试报告等资源。

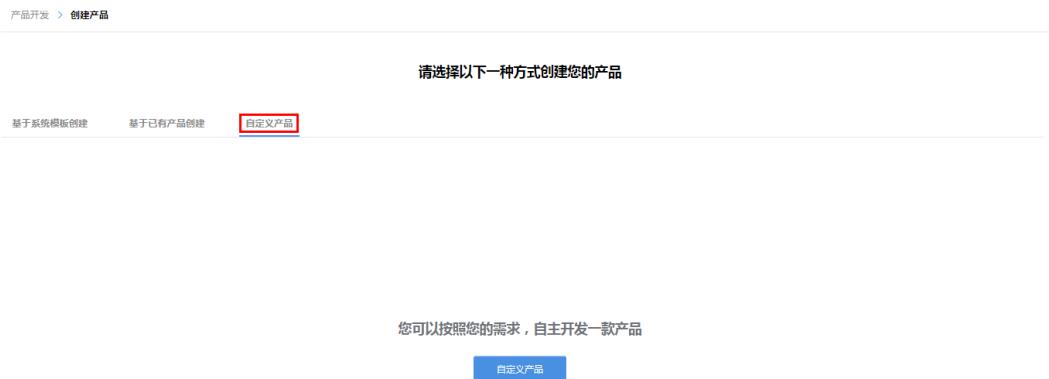
#### 自定义新建产品

自定义新建产品指不使用系统预置的产品模板，全新定义一款产品。

**步骤1** 在项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“自定义产品”界面，点击“自定义产品”。



**步骤3** 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“厂商名称”等信息后，点击“创建”。

**说明**

- “产品名称”、“型号”需要在项目内保持唯一，否则会创建失败。
- “所属行业”、“设备类型”根据实际情况进行填写。
- “接入应用层协议类型”配置为“LWM2M”，“数据格式”配置为“二进制码流”。

### 设置产品信息

\* 产品名称 : WaterMeter01

\* 型号 : NBiotDDevice01

\* 厂商ID : e08ac2b7703849c784cb0ae23e0283c5

\* 厂商名称 : TestManuName

\* 所属行业 : 智慧城市

\* 设备类型 : WaterMeter

\* 接入应用层协议类型 : LWM2M

注意 : CoAP协议的设备需要完善数据解析，将设备上报的二进制数据转换为平台上的JSON数据格式

\* 数据格式 : 二进制码流

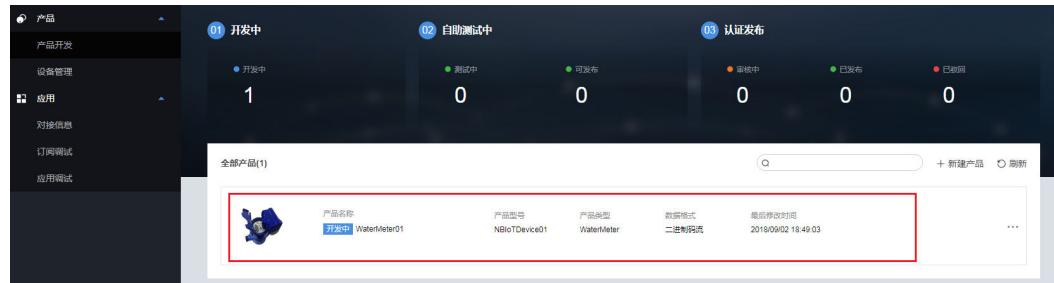
产品图片 : 请选择图片

图片大小为200\*200 ×



创建 取消

**步骤4** 在“产品开发”界面将会呈现已经创建的产品，选择具体产品，可以进入该产品的开发空间。

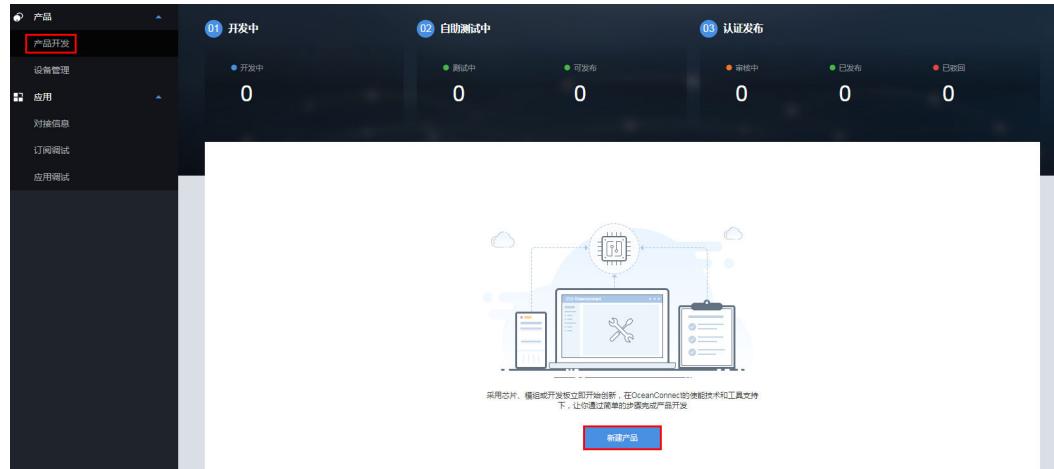


----结束

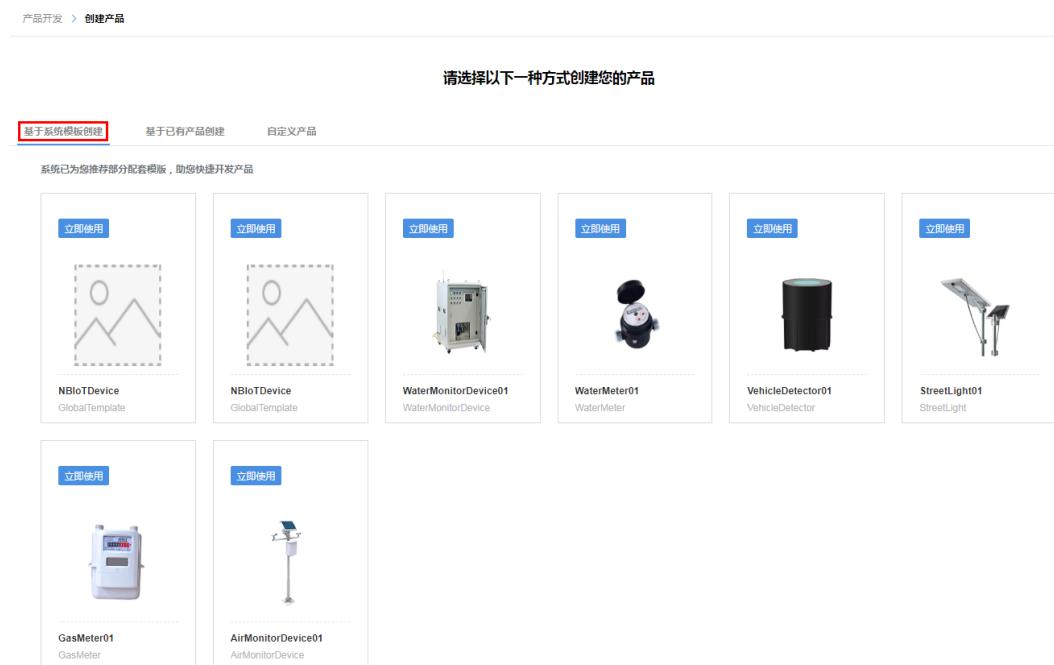
## 快速新建产品

快速新建产品指使用系统预置的产品模板（或已经创建的产品模板），快速定义一款产品。

**步骤1** 在项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“基于系统模板创建”界面，选择需要的系统模板，点击“立即使用”。



**步骤3** 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“厂商名称”等信息后，点击“创建”。

 **说明**

- “产品名称”、“型号”需要在项目内保持唯一，否则会创建失败。
- “所属行业”、“设备类型”根据实际情况进行填写。
- “接入应用层协议类型”配置为“LWM2M”，“数据格式”配置为“二进制码流”。

### 设置产品信息

\* 产品名称 : WaterMeter01

\* 型号 : NBiotDDevice01

\* 厂商ID : e08ac2b7703849c784cb0ae23e0283c5

\* 厂商名称 : TestManuName

\* 所属行业 : 智慧城市

\* 设备类型 : WaterMeter

\* 接入应用层协议类型 : LWM2M

注意 : CoAP协议的设备需要完善数据解析，将设备上报的二进制数据转换为平台上的JSON数据格式

\* 数据格式 : 二进制码流

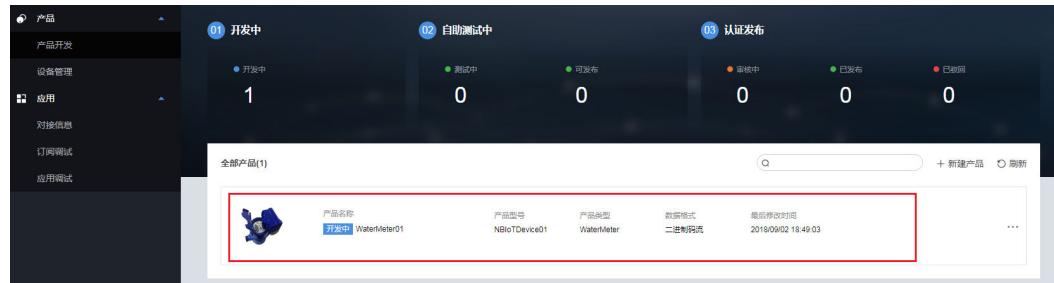
产品图片 : 请选择图片

图片大小为200\*200 ×



创建 取消

**步骤4** 在“产品开发”界面将会呈现已经创建的产品，选择具体产品，可以进入该产品的开发空间。

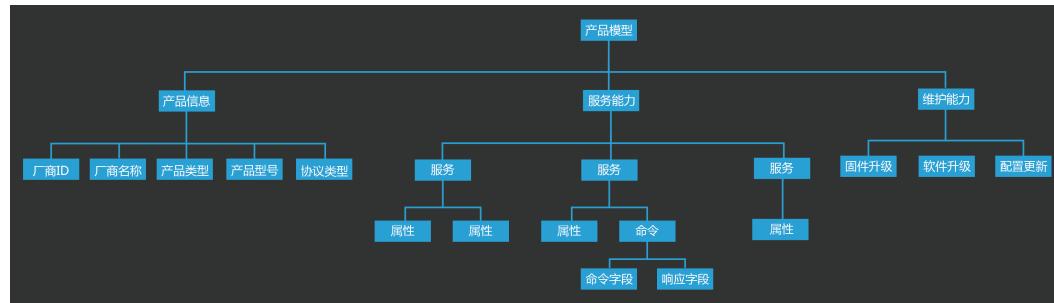


----结束

### 3.2.2 Profile 定义

#### 概述

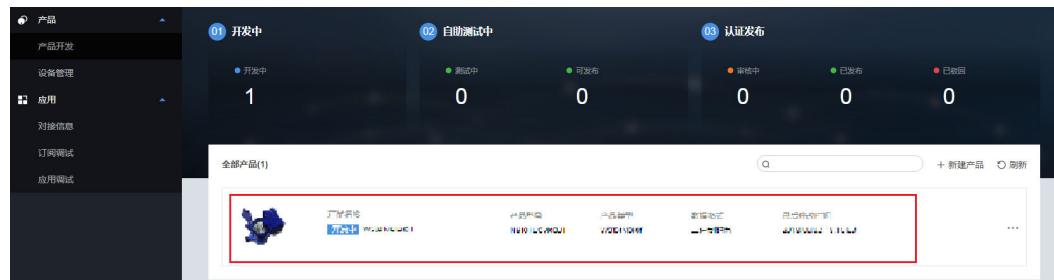
产品模型（也称Profile）用于描述设备具备的能力和特性。开发者通过定义Profile，在物联网平台构建一款设备的抽象模型，使平台理解该款设备支持的服务、属性、命令等信息。



#### 定义 Profile

在**产品创建**时：如果选择使用系统模板，则系统将会自动使用相应的Profile模板，开发者可以直接使用或在此基础上进行修改；如果选择自定义产品模板，则需要完整定义Profile，操作如下：

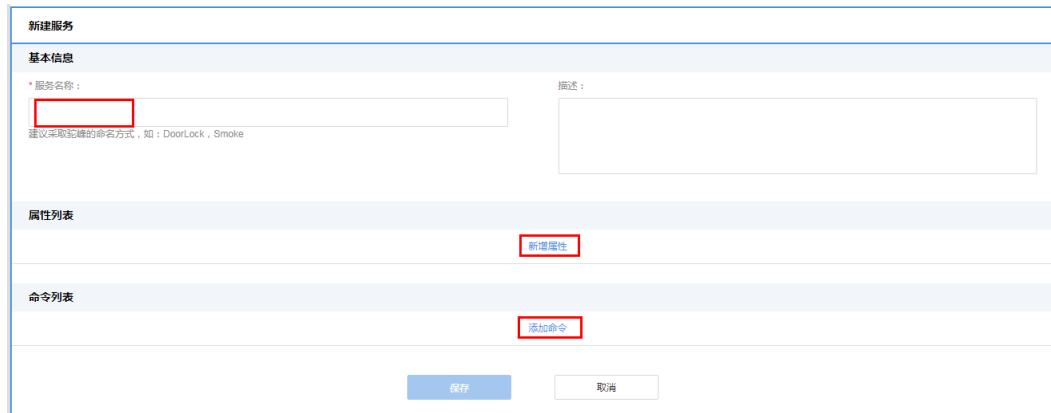
**步骤1** 在“产品开发”界面选择产品，选择具体产品，进入该产品的开发空间。



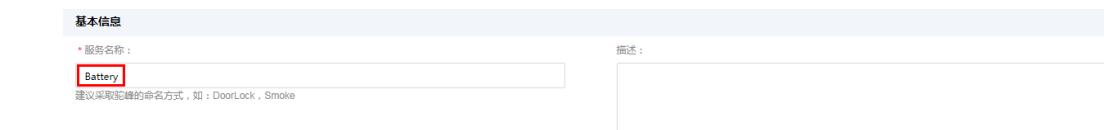
**步骤2** 在产品开发空间，点击“Profile定义”，然后点击“新建服务”。



**步骤3** 在“新建服务”区域，对服务名称、属性和命令进行定义。每个服务下，可以包含属性和命令，也可以只包含其中之一，请根据此类设备的实际情况进行配置。



1. 填写“服务名称”，“服务名称”采用首字母大写的命名方式，比如：WaterMeter、Battery。



2. 点击“新增属性”，在弹出窗口中配置属性的各项参数，点击“确定”。“属性名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如 batteryLevel、internalTemperature；其余参数，请根据此类设备的实际情况进行配置。

“数据类型”的配置可参考如下原则：

- **int:** 当上报的数据为整数或布尔值时，可以配置为此类型。
- **decimal:** 当上报的数据为小数时，可以配置为此类型。
- **string:** 当上报的数据为字符串、枚举值或布尔值时，可以配置为此类型。如果为枚举值或布尔值，值之间需要用英文逗号（“,”）分隔。
- **DateTime:** 当上报的数据为日期时，可以配置为此类型。
- **jsonObject:** 当上报的数据为json结构体时，可以配置为此类型。

### 新增属性

名称 \*

数据类型 \*

最小 \*

最大 \*

步长

单位

访问模式 \*

R 可以读取属性的值

W 你可以写 (更改) 属性的值

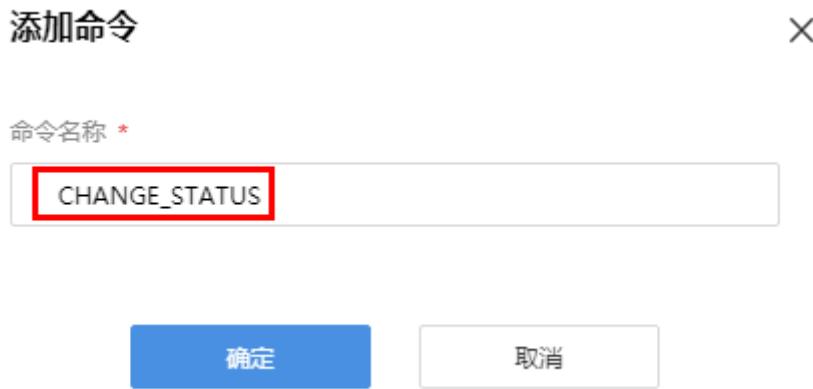
E 当属性值更改时上报事件

是否必选

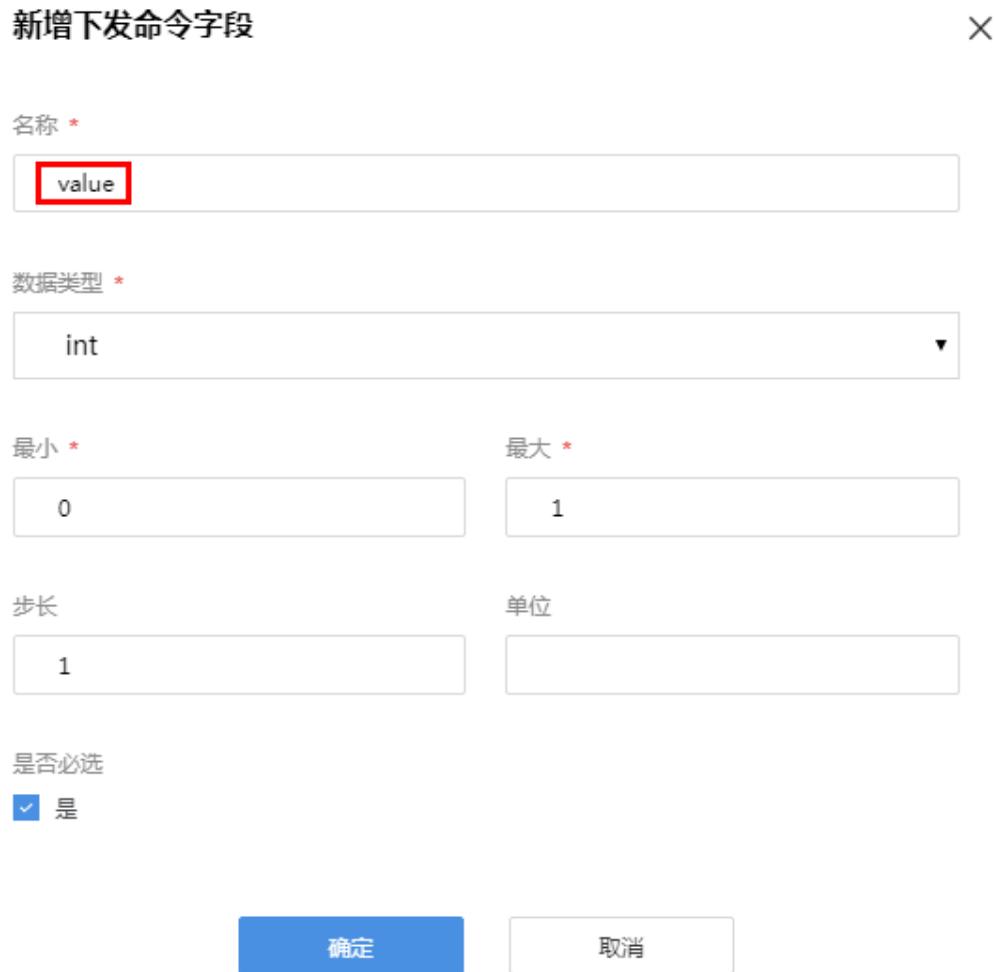
是

确定 取消

3. 点击“添加命令”，在弹出窗口中配置“命令名称”，点击“确定”。“命令名称”采用所有字母大写，单词间用下划线连接的命名方式，比如DISCOVERY, CHANGE\_STATUS。



4. 点击“添加下发字段”，在弹出窗口中配置下发命令字段的各项参数，点击“确定”。“命令字段名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如value；其余参数，请根据此类设备的实际情况进行配置。



5. 点击“添加响应字段”，在弹出窗口中配置响应命令字段的各项参数，点击“确定”。“响应字段名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如result；其余参数，请根据此类设备的实际情况进行配置。  
响应字段非必配参数，当需要设备返回命令执行结果时，才需要定义此字段。

## 新增响应命令字段

X

名称 \*

result

数据类型 \*

int

最小 \*

0

最大 \*

1

步长

1

单位

是否必选

 是

确定

取消

----结束

## 导出 Profile

当需要将在线开发的Profile文件移植到其他项目或平台时，则可以将Profile导出。Profile可以在“Profile定义”中导出，也可以在“产品详情”中导出。

在“Profile定义”中导出

**步骤1** 在“Profile定义”界面，点击“导出Profile”。

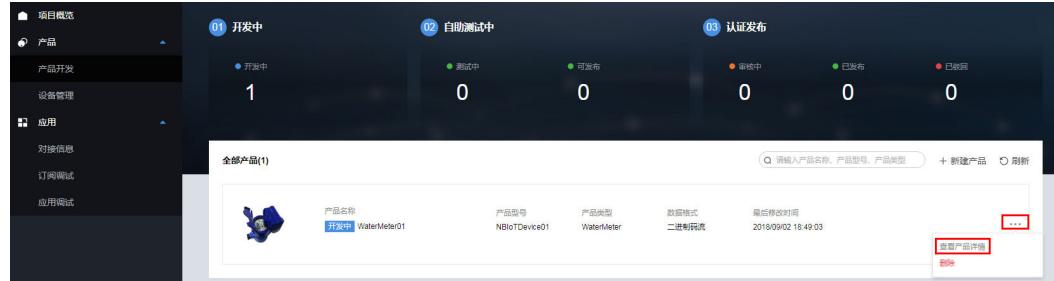
The screenshot shows the 'Profile Definition' tab selected in a navigation bar with four tabs: 01 Profile Definition, 02 SDK集成, 03 编解码插件开发, and 04 在线调测. Below the tabs is a service list table with one entry: 'Battery' (描述: Battery, 最后修改时间: 2018/11/16 15:36:52). At the top right of the table, there are three buttons: '+新建服务' (New Service), '立即导出Profile' (highlighted with a red box), and '导入Profile'.

**步骤2** 选择Profile的存放路径，将Profile下载到本地。

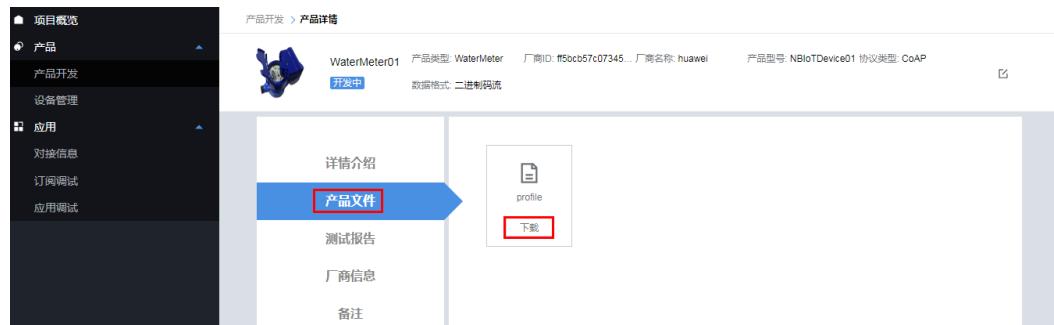
----结束

在“产品详情”中导出

**步骤1** 在“产品开发”界面选择需要查看详情的产品，点击该产品右侧“...”按钮，然后点击“查看产品详情”，进入“产品详情”界面。



**步骤2** 在“产品详情 > 产品文件”界面，点击Profile区域的“下载”。



----结束

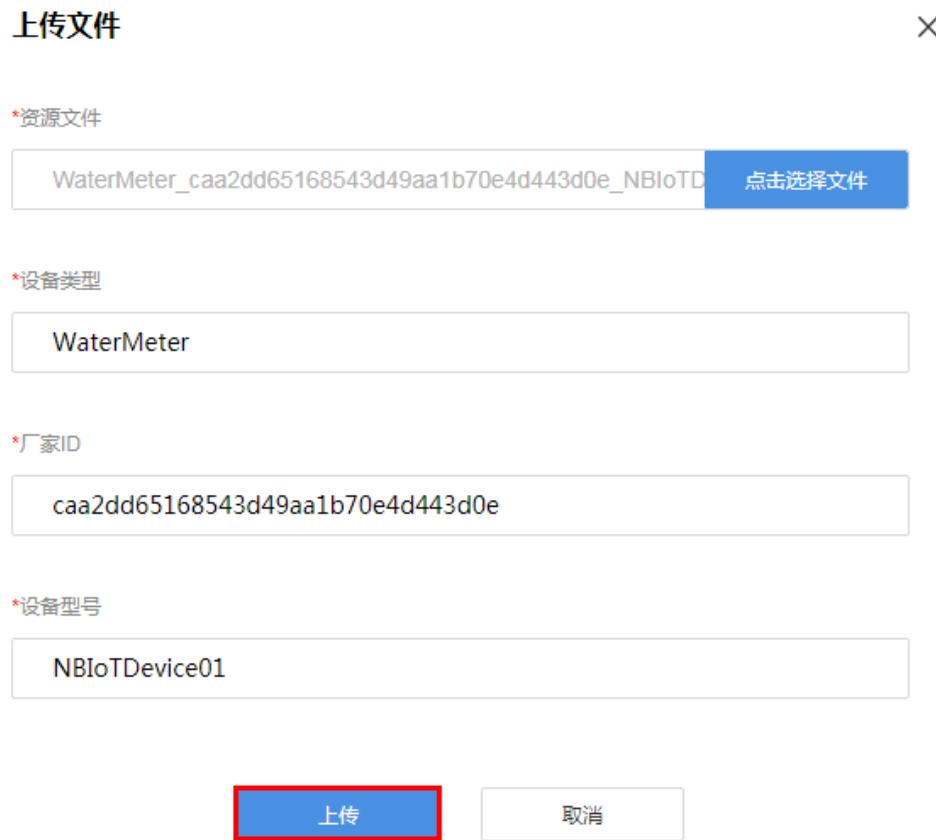
## 导入 Profile

当开发者已经有完备的Profile时（线下开发或从其他项目/平台导出），可以将Profile直接导入到物联网平台。

**步骤1** 在“Profile定义”界面，点击“导入Profile”。



**步骤2** 在弹窗中选择Profile文件后，点击“上传”。在选择“资源文件”后，“设备类型”、“厂家ID”、“设备型号”将会根据Profile文件的命名自动填充，且需要和创建产品时填写的信息保持一致。



----结束

### 3.2.3 插件开发

#### 3.2.3.1 插件开发指导

##### 概述

设备上报数据时，如果“数据格式”为“二进制码流”，则该产品下需要进行编解码插件开发；如果“数据格式”为“JSON”，则该产品下不需要进行编解码插件开发。

以NB-IoT场景为例，NB-IoT设备和物联网平台之间采用CoAP协议通讯，CoAP消息的payload为应用层数据，应用层数据的格式由设备自行定义。由于NB-IoT设备一般对省电要求较高，所以应用层数据一般不采用流行的JSON格式，而是采用二进制格式。但是，物联网平台与应用侧使用JSON格式进行通信。因此，开发者需要开发编码插件，供物联网平台调用，以完成二进制格式和JSON格式的转换。

##### 开发编解码插件

在[产品创建](#)时：如果选择使用系统模板，部分模板会包含编解码插件，开发者可以直接使用或在此基础上进行修改；如果选择自定义产品模板，则需要完整开发编解码插件，操作如下：

**步骤1** 在产品开发空间，点击“编解码插件开发”。

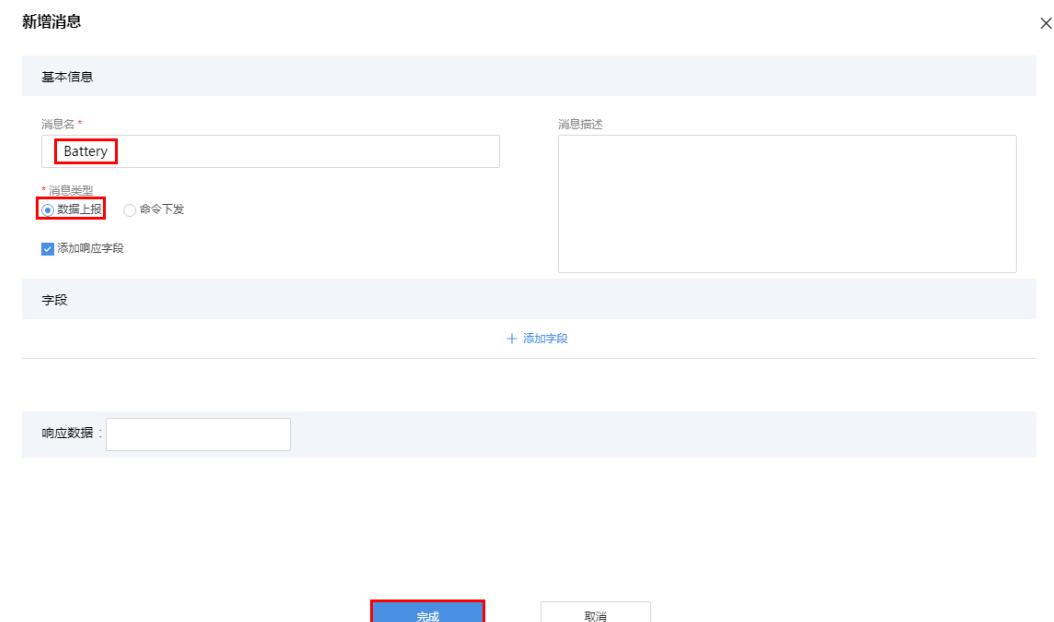


**步骤2** 在“在线编解码插件编辑器”区域，点击“新增消息”。



**步骤3** 系统将弹出“新建消息”窗口，填写“消息名”，“消息类型”选择“数据上报”，点击“完成”。

- 设备在上报数据后，如果需要物联网平台返回ACK响应消息，则需要勾选“添加响应字段”。ACK响应消息携带的数据可以在“响应数据”中配置，默认携带“AAAA0000”。
- “消息名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。



**步骤4** 点击数据上报字段后的“+”。



**步骤5** 系统将弹出“增加字段”窗口，勾选“标记为地址域”，其余参数将自动填充，点击“完成”。

当有相同类型的消息时（例如：两种数据上报的消息），需要添加地址域字段，且该字段在字段列表的位置必须一致。命令响应消息可看作一种数据上报消息，因此如果存在命令响应消息，则需要在数据上报消息中添加地址域。



**步骤6** 点击数据上报字段后的“+”。



**步骤7** 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

编辑字段 X

标记为地址域 ?

\* 名字

描述

数据类型  
 ▾

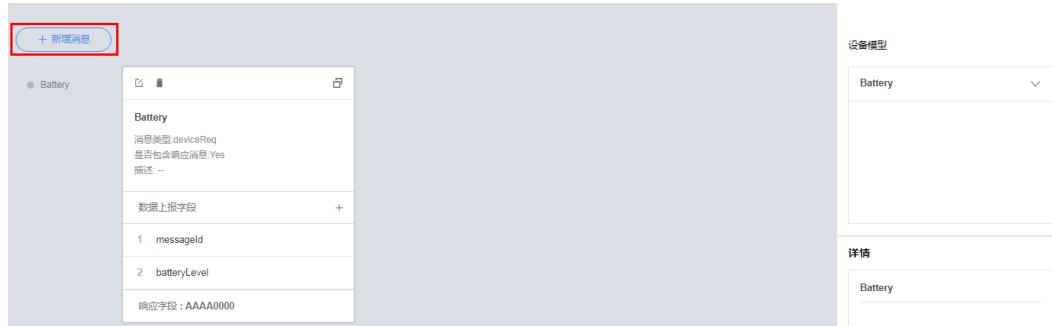
\* 长度 ?

默认值 ?

偏移值 ?

完成 取消

**步骤8** 在“编解码插件编辑器”区域，点击“新增消息”。



**步骤9** 系统将弹出“新建消息”窗口，填写“消息名”，“消息类型”选择“命令下发”，点击“完成”。

- 设备在接到命令后，如果需要返回命令执行结果，则需要勾选“添加响应字段”。勾选后：
  - 需要在数据上报消息和命令响应消息中均定义地址域字段，并且该字段在两种消息的字段列表中的位置必须相同，使编解码插件可以对数据上报消息和命令响应消息进行区分。
  - 需要在命令下发消息和命令响应消息中定义响应标识字段，并且该字段在两种消息的字段列表中的位置必须相同，使编解码插件可以将命令下发消息和对应的命令响应消息进行关联。
- “消息名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。

**步骤10** 点击命令下发字段后的“+”。



**步骤11** 系统将弹出“增加字段”窗口，勾选“标记为地址域”，其余参数将自动填充，点击“完成”。

当有相同类型的消息时（例如：两种命令下发的消息），需要添加地址域字段，且该字段在字段列表的位置必须一致。数据上报响应消息可看作一种命令下发消息，因此如果存在数据上报响应消息，则需要在命令下发消息中添加地址域。

添加字段 X

标记为地址域 ?

标记为响应标识字段 ?

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

描述

数据类型

▼

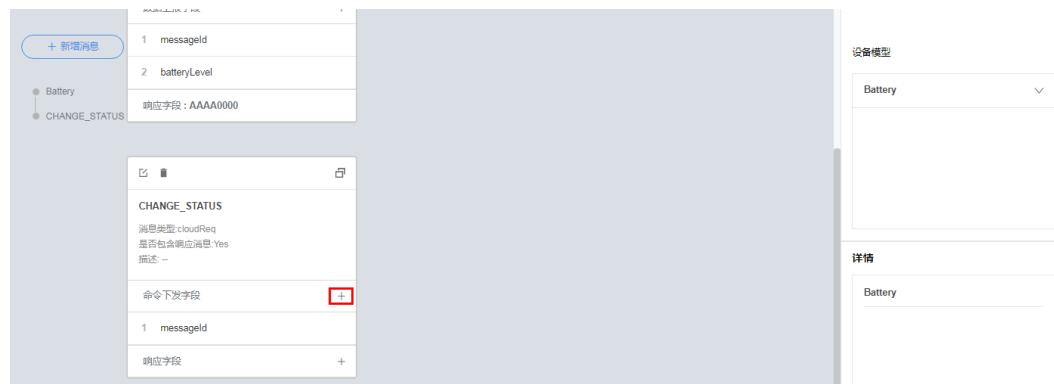
\* 长度 ?

\* 默认值 ?

偏移值 ?

完成 取消

**步骤12** 点击命令下发字段后的“+”。



**步骤13** 系统将弹出“增加字段”窗口，勾选“标记为响应标识字段”，其余参数将自动填充，点击“完成”。

添加字段

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

\* 名字 当标记为响应标识字段时，名字固定为mid；否则，名字不能设置为mid。

mid

描述

数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

偏移值 [?](#)

1-3

完成 取消

**步骤14** 点击命令下发后的“+”。



**步骤15** 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

添加字段 X

标记为地址域 ?

标记为响应标识字段 ?

\* 名字

描述

数据类型

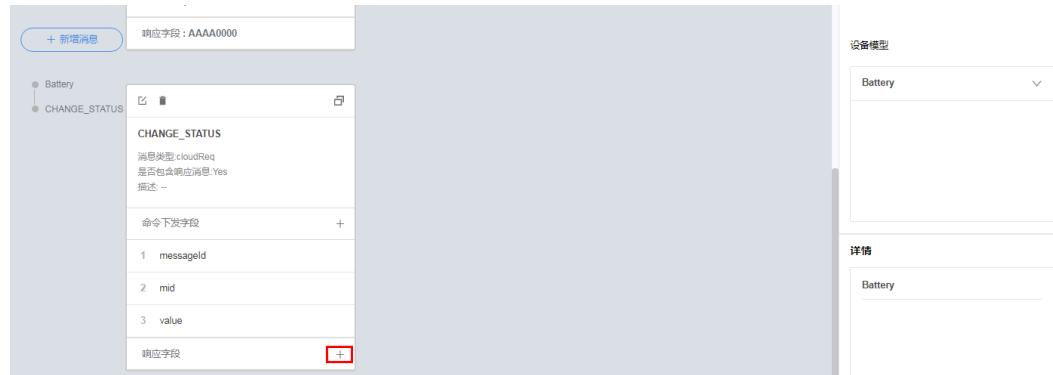
\* 长度 ?

默认值 ?

偏移值 ?

完成 取消

**步骤16** 点击响应字段后的“+”。



**步骤17** 系统将弹出“增加字段”窗口，勾选“标记为地址域”，其余参数将自动填充，点击“完成”。

添加字段 X

标记为地址域 (?)

标记为响应标识字段 (?)

标记为命令执行状态字段 (?)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

描述

数据类型

int8u(8位无符号整型) ▼

\* 长度 (?)

1

\* 默认值 (?)

0x2

偏移值 (?)

0-1

完成 取消

**步骤18** 点击响应字段后的“+”。



**步骤19** 系统将弹出“增加字段”窗口，勾选“标记为响应标识字段”，其余参数将自动填充，点击“完成”。

## 添加字段

X

标记为地址域 ⑦

标记为响应标识字段 ⑦

标记为命令执行状态字段 ⑦

\* 名字 当标记为响应标识字段时，名字固定为mid；否则，名字不能设置为mid。

mid

### 描述

### 数据类型

int16u(16位无符号整型)

▼

\* 长度 ⑦

2

默认值 ⑦

偏移值 ⑦

1-3

完成

取消

**步骤20** 点击响应字段后的“+”。



**步骤21** 系统将弹出“增加字段”窗口，勾选“标记为命令执行状态字段”，完成各项参数配置后，点击“完成”。

- “名字”将自动填充。
- “数据类型”根据设备命令响应的实际情况进行配置，需要和Profile相应字段的定义相匹配

添加字段 X

标记为地址域 (?)

标记为响应标识字段 (?)

标记为命令执行状态字段 (?)

\* 名字 当标记为命令执行状态字段时，名字固定为errcode；否则，名字不能设置为errcode。

描述

数据类型

▼

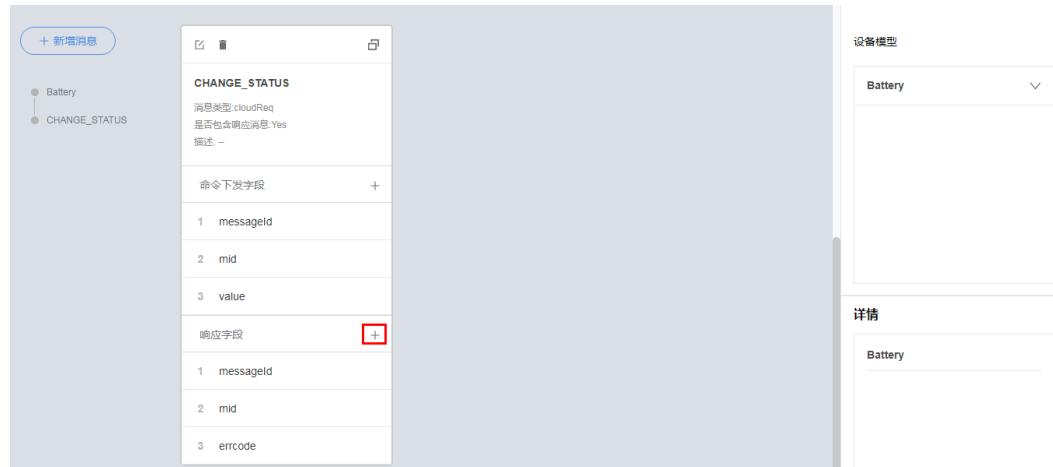
\* 长度 (?)

默认值 (?)

偏移值 (?)

完成 取消

**步骤22** 点击响应字段后的“+”。



**步骤23** 系统将弹出“增加字段”窗口，完成各项参数配置后，点击“完成”。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

### 添加字段

X

- 标记为地址域 ⑦
- 标记为响应标识字段 ⑦
- 标记为命令执行状态字段 ⑦

\* 名字

result

描述

数据类型

int8u(8位无符号整型)

▼

\* 长度 ⑦

1

默认值 ⑦

偏移值 ⑦

4-5

完成

取消

**步骤24** 拖动右侧“设备模型”区域的属性字段、命令字段和响应字段，与数据上报消息、命令下发消息和命令响应消息的相应字段建立映射关系。



**步骤25** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 下载编解码插件

编解码插件可以在“编解码插件开发”中下载，也可以在“产品详情”中下载。

在“编解码插件开发”中导出

**步骤1** 在“编解码插件开发”界面，选择“更多 > 下载”。



**步骤2** 系统弹出“请选择插件签名方式”窗口，请选择“生成新的公私钥签名”或“使用已有公私钥签名”：

- 如果选择生成新的公私钥进行签名，则输入“私钥密码”后，点击“下载”。系统将生成已签名的插件包和公私钥文件。

**说明**

密码长度至少6个字符，必须为字母和数字的组合。

**请选择插件签名方式**

X

- 生成新的公私钥签名  使用已有私钥签名

\* 私钥密码

1234567890

下载签名插件包需要设置签名密码。

**下载**

**取消**

- 如果选择已有私钥进行签名，则上传私钥文件，填写私钥密码后，点击“下载”。系统将生成已签名的插件包和公私钥文件。

**说明**

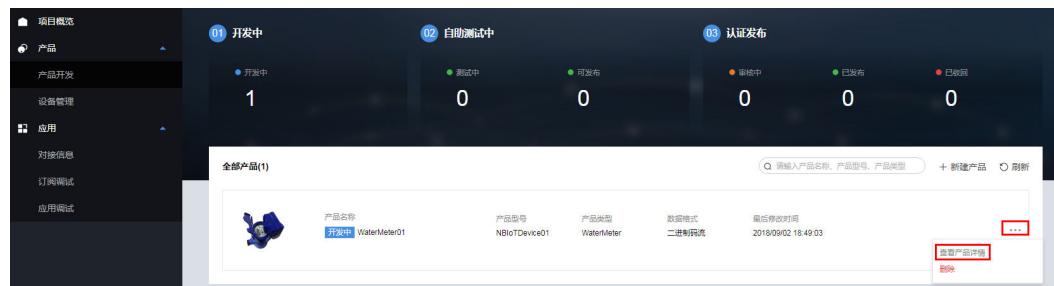
下载的压缩包里的私钥文件即为上传的私钥。



----结束

在“产品详情”中下载

**步骤1** 在“产品开发”界面选择需要查看详情的产品，点击该产品右侧“...”按钮，然后点击“查看产品详情”，进入“产品详情”界面。



**步骤2** 系统弹出“请选择插件签名方式”窗口，请选择“生成新的公私钥签名”或“使用已有公私钥签名”：

- 如果选择生成新的公私钥进行签名，则输入“私钥密码”后，点击“下载”。系统将生成已签名的插件包和公私钥文件。

#### 说明

密码长度至少6个字符，必须为字母和数字的组合。

### 请选择插件签名方式

X

- 生成新的公私钥签名  使用已有私钥签名

\* 私钥密码

下载签名插件包需要设置签名密码。

下载

取消

- 如果选择已有私钥进行签名，则上传私钥文件，填写私钥密码后，点击“下载”。系统将生成已签名的插件包和公私钥文件。



说明

下载的压缩包里的私钥文件即为上传的私钥。



----结束

## 上传编解码插件

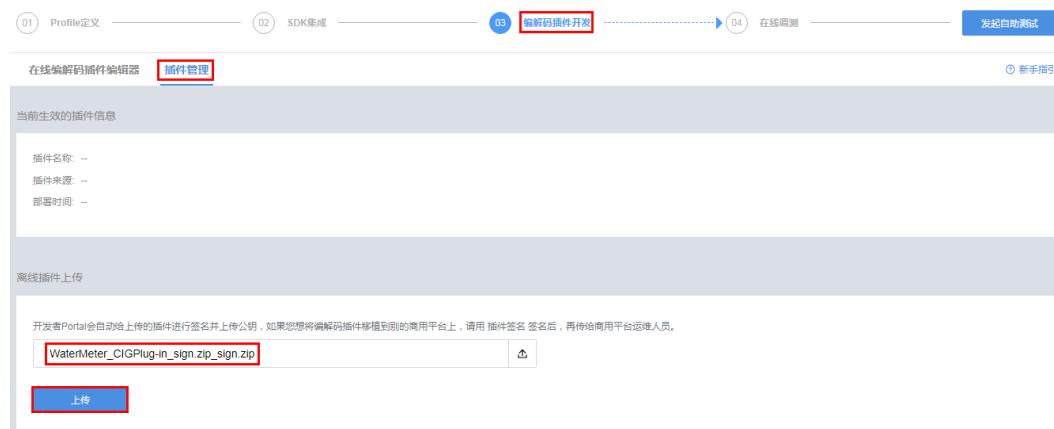
当本地已经准备好编解码插件（如线下开发）时，则可以通过开发中心将插件上传至物联网平台。

**步骤1** 在“编解码插件开发”界面，选择“插件管理”。

**步骤2** 选择需要上传的插件包，点击“上传”。

### 说明

插件包的设备类型、型号、厂商ID等信息需要与该产品保持一致，才可以成功上传。



当界面提示“离线插件上传成功”时，表示插件已经完成上传。



----结束

### 3.2.3.2 插件开发实例

#### 3.2.3.2.1 数据上报和命令下发的插件开发

##### 场景说明

有一款烟感设备，具有如下特征：

- 具有烟雾报警功能（火灾等级）和温度上报功能。
- 支持远程控制命令，可远程打开报警功能。比如火灾现场温度，远程打开烟雾报警，提醒住户疏散。

##### Profile 定义

在烟感产品的开发空间，完成Profile定义。

- level：火灾级别，用于表示火灾的严重程度。
- temperature：温度，用于表示火灾现场温度。
- SET\_ALARM：打开或关闭告警命令，value=0表示关闭，value=1表示打开。

The screenshot shows the 'Smoke' service configuration page. It includes sections for 'Attribute List' (with fields like level and temperature), 'Command List' (with a command named SET\_ALARM), and 'Response Command Field'. A note at the bottom states: '您还未创建任何响应命令。' (No response command has been created yet).

## 编解码插件开发

**步骤1** 在烟感产品的开发空间，选择“编解码插件开发”。



**步骤2** 配置数据上报消息。

The 'Add Message' dialog is shown with the following configuration:

- 基本信息**:
  - 消息名: smokeinfo (highlighted with a red box)
  - 消息类型: 数据上报 (highlighted with a red box)
  - 添加响应字段: (unchecked)
- 字段**:
  - + 添加字段

At the bottom right are '完成' (Finish) and '取消' (Cancel) buttons.

添加level字段，表示火灾级别。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。
- “长度”和“偏移值”根据“数据类型”的配置自动填充。

添加字段 X

标记为地址域 [?](#)

\* 名字

描述

数据类型

\* 长度 [?](#)

默认值 [?](#)

偏移值 [?](#)

完成 取消

添加temperature字段，表示温度。在Profile中，temperature属性最大值1000，因此在插件中定义temperature字段的“数据类型”为“int16u”，以满足temperature属性的取值范围。

添加字段 X

标记为地址域 [?](#)

\* 名字

temperature

描述

数据类型

int16u(16位无符号整型) ▾

\* 长度 [?](#)

2

默认值 [?](#)

偏移值 [?](#)

1-3

完成 取消

**步骤3** 配置命令下发消息。

新增消息

基本信息

消息名： 消息描述

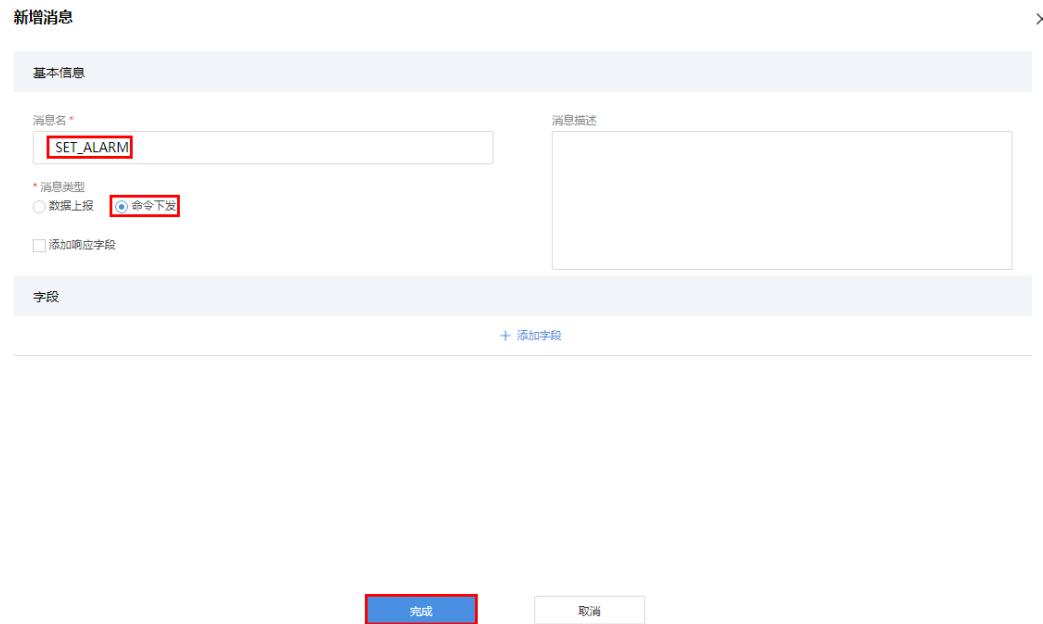
\* 消息类型  
 数据上报  命令下发

添加响应字段

字段

+ 添加字段

完成 取消



添加value字段，表示下发命令的参数值。

添加字段

标记为地址域 [?](#)

\* 名字

value

描述

数据类型

int8u(8位无符号整型) ▾

\* 长度 [?](#)

1

默认值 [?](#)

偏移值 [?](#)

0-1

完成 取消

**步骤4** 拖动右侧“设备模型”区域的属性字段和命令字段，数据上报消息和命令下发消息的相应字段建立映射关系。



**步骤5** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 调测编解码插件

**步骤1** 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。



勾选“没有真实的物理设备”，点击“创建”。

### 新增测试设备

您现在

有真实的物理设备

没有真实的物理设备

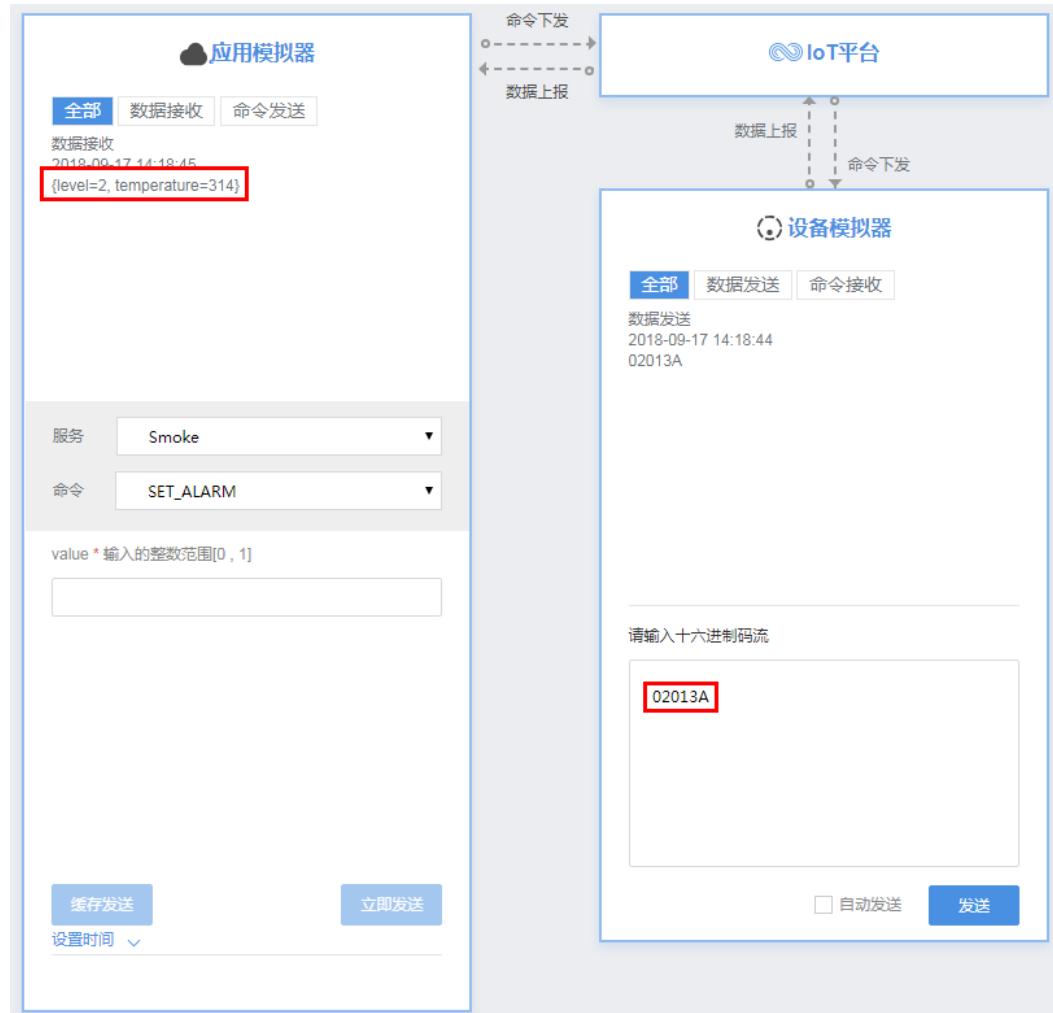
您正在注册一个虚拟的设备

**创建**

取消

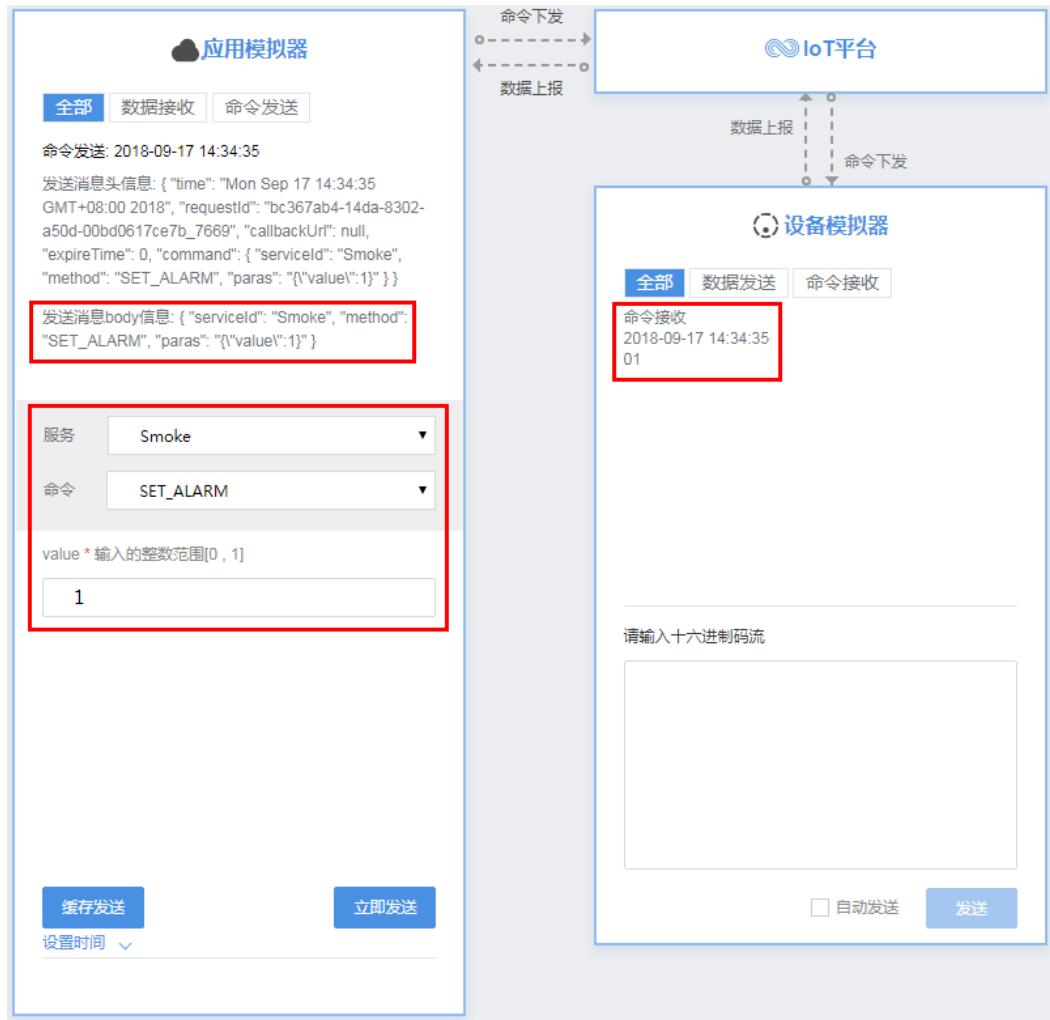
**步骤2** 使用设备模拟器进行数据上报。十六进制码流示例：02013A。02表示火灾级别，长度为1个字节；013A表示温度，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{level=2, temperature=314}。2为十六进制数02转换为十进制的数值；314为十六进制数013A转换为十进制的数值。



**步骤3** 使用应用模拟器进行命令下发：{ "serviceId": "Smoke", "method": "SET\_ALARM", "paras": "{\"value\":1}" }。

在“设备模拟器”区域查看命令接收的结果：01。01为十进制数1转换为十六进制的数值。



----结束

### 3.2.3.2.2 多条数据上报消息的插件开发

#### 场景说明

有一款烟感设备，具有如下特征：

- 具有烟雾报警功能（火灾等级）和温度上报功能。
- 支持远程控制命令，可远程打开报警功能。比如火灾现场温度，远程打开烟雾报警，提醒住户疏散。
- 支持同时上报烟雾报警（火灾等级）和温度，也支持单独上报温度。

#### Profile 定义

在烟感产品的开发空间，完成Profile定义。

- **level**: 火灾级别，用于表示火灾的严重程度。
- **temperature**: 温度，用于表示火灾现场温度。
- **SET\_ALARM**: 打开或关闭告警命令，value=0表示关闭，value=1表示打开。

The screenshot shows the 'Smoke' service configuration page. It includes sections for 'Attribute List' (with fields for level and temperature), 'Command List' (with a 'SET\_ALARM' command), 'Downstream Command Field' (with a 'value' field), and 'Response Command Field' (empty). There are also buttons for adding attributes, commands, and response fields.

## 编解码插件开发

**步骤1** 在烟感产品的开发空间，选择“编解码插件开发”。



**步骤2** 配置数据上报消息，上报火灾等级和温度。

The dialog has two tabs: 'Basic Information' and 'Fields'. In 'Basic Information', the message name is 'smokeinfo', the message type is 'Data Report' (selected), and there is a note about adding response fields. In 'Fields', there is a '+' button to add fields. At the bottom are 'Finish' and 'Cancel' buttons, with 'Finish' highlighted with a red box.

添加messageId字段，表示消息种类。

- 在本场景中，数据上报消息有两种，所以需要用messageId来标志消息种类。
- “数据类型”根据数据上报消息种类的数量进行配置。在本场景中，仅有两种数据上报消息，配置为“int8u”即可满足需求。
- “默认值”可以修改，但必须为十六进制格式，且数据上报消息的对应字段必须和默认值保持一致。在本场景中，用0x0标识上报火灾等级和温度的消息。

## 添加字段

X

 标记为地址域 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

## 描述

## 数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x0

偏移值 [?](#)

0-1

完成

取消

添加level字段，表示火灾级别。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

- “长度”和“偏移值”根据“数据类型”的配置自动填充。

**添加字段** X

标记为地址域 (?)

\* 名字

描述

数据类型

\* 长度 (?)

默认值 (?)

偏移值 (?)

完成 取消

添加temperature字段，表示温度。在Profile中，temperature属性最大值1000，因此在插件中定义temperature字段的“数据类型”为“int16u”，以满足temperature属性的取值范围。

添加字段

标记为地址域 [?](#)

\* 名字

temperature

描述

数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

偏移值 [?](#)

2-4

完成

取消

**步骤3** 配置数据上报消息，只上报温度。

新增消息

基本信息

消息名 \*  消息描述

\* 消息类型  数据上报  命令下发

添加响应字段

字段 [+ 添加字段](#)

添加 messageId 字段，表示消息种类。在本场景中，用 0x1 标识只上报温度的消息。

## 添加字段

X

标记为地址域 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

描述

数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x1

偏移值 [?](#)

0-1

完成

取消

添加temperature字段，表示温度。

添加字段

标记为地址域 [?](#)

\* 名字

temperature

描述

数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

偏移值 [?](#)

1-3

[完成](#) [取消](#)

**步骤4** 配置命令下发消息。

新增消息

基本信息

消息名： 消息描述

\* 消息类型  
 数据上报  命令下发

添加响应字段

字段

+ 添加字段

完成 取消

添加value字段，表示下发命令的参数值。

添加字段 ×

标记为地址域 ?

\* 名字 value

描述

数据类型 int8u(8位无符号整型) ▾

\* 长度 ? 1

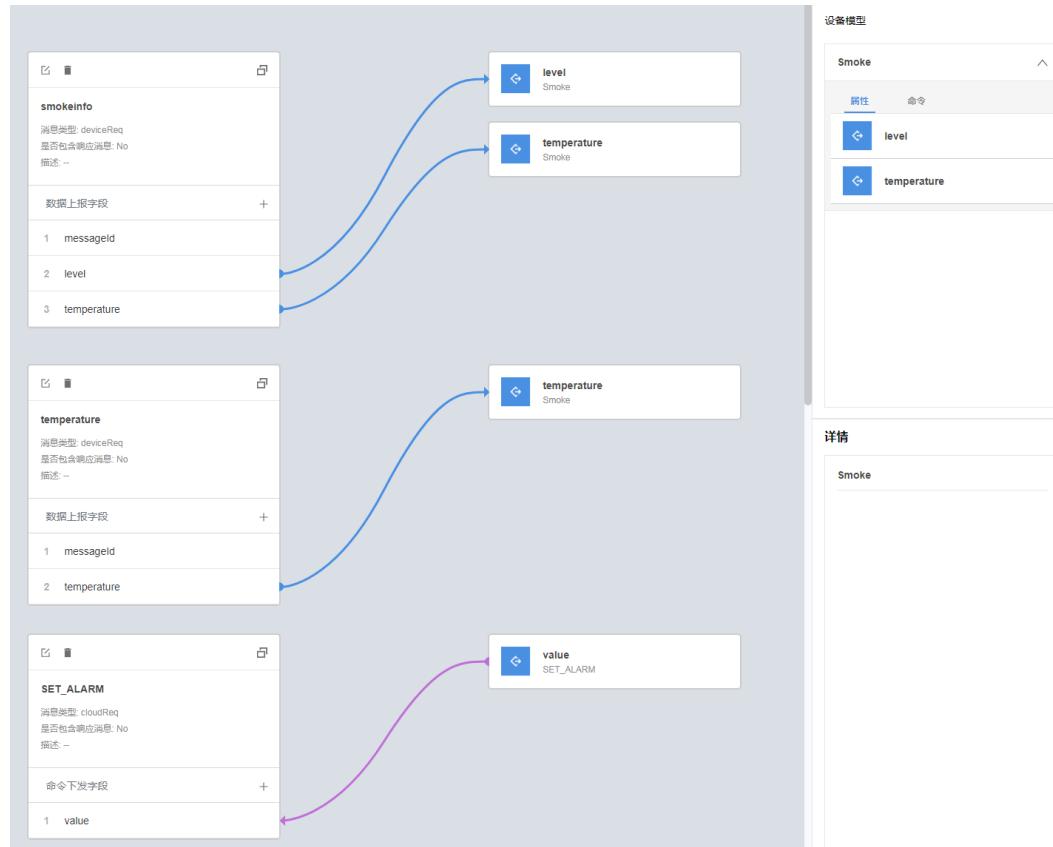
默认值 ?

偏移值 ? 0-1

完成 取消

**步骤5** 拖动右侧“设备模型”区域的属性字段和命令字段，与数据上报消息和命令下发消息的相应字段建立映射关系。

level字段和temperature字段分别与Profile中的对应属性建立映射关系， messageId用于帮插件识别消息种类，不需要建立映射关系。



**步骤6** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 调测编解码插件

**步骤1** 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。

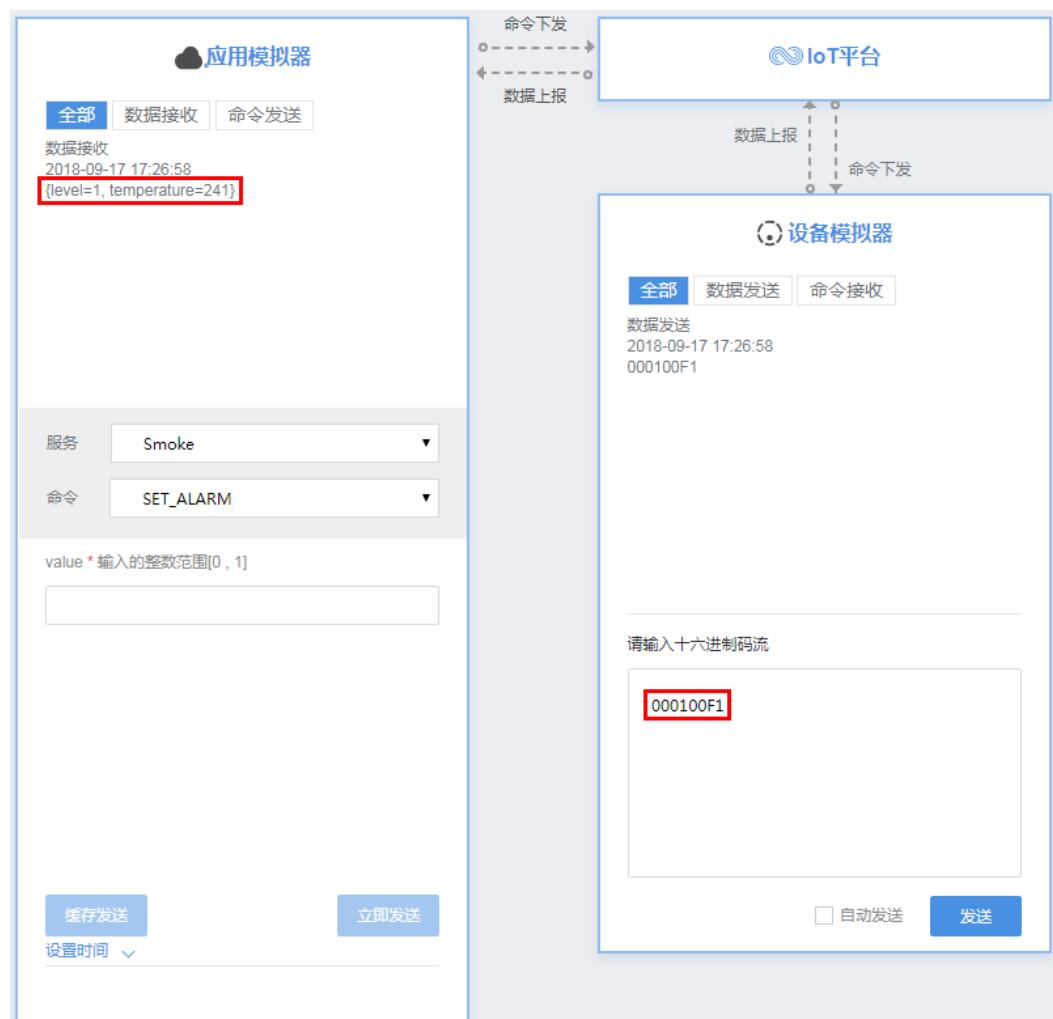


勾选“没有真实的物理设备”，点击“创建”。

**步骤2** 使用设备模拟器进行数据上报。

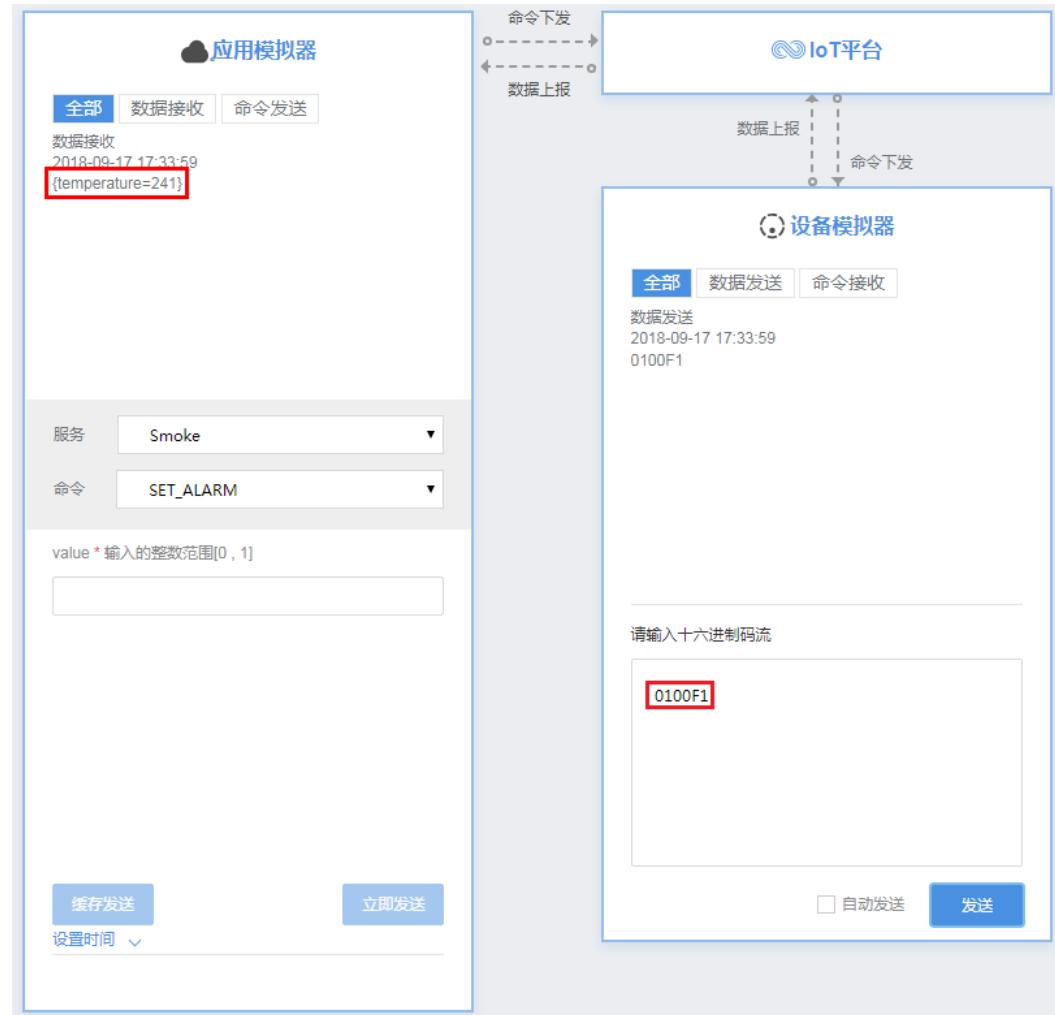
十六进制码流示例：000100F1。00表示messageId，此消息上报火灾等级和温度；01表示火灾级别，长度为1个字节；00F1表示温度，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{level=1, temperature=241}。1为十六进制数01转换为十进制的数值；241为十六进制数00F1转换为十进制的数值。



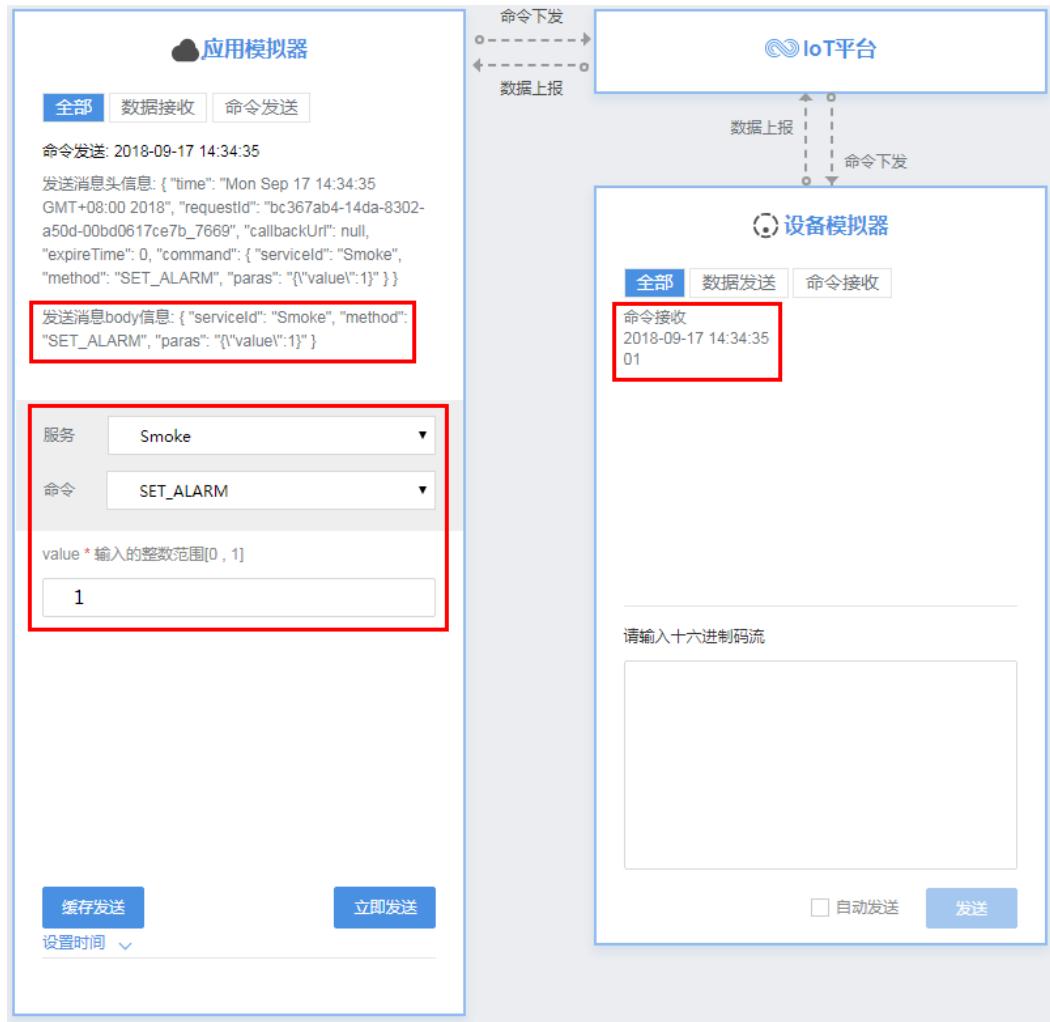
十六进制码流示例：0100F1。01表示messageId，此消息只上报火灾温度；00F1表示温度，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{temperature=241}。241为十六进制数00F1转换为十进制的数值。



**步骤3** 使用应用模拟器进行命令下发：{"serviceId": "Smoke", "method": "SET\_ALARM", "paras": "{\"value\":1}" }。

在“设备模拟器”区域查看命令接收的结果：01。01为十进制数1转换为十六进制的数值。



----结束

### 3.2.3.2.3 字符串及可变长字符串数据类型的插件开发

#### 场景说明

有一款烟感设备，具有如下特征：

- 具有烟雾报警功能（火灾等级）和温度上报功能，支持同时上报烟雾报警（火灾等级）和温度，也支持单独上报温度。
- 具备描述信息上报功能，描述信息支持字符串和可变长度字符串两种类型。



#### 说明

该场景旨在讲解字符串及可变长度字符串数据类型的编解码插件开发方式，数据上报和命令下发的插件开发方式类似，本章节以数据上报为例进行说明，因此省略命令下发的相关步骤。

#### Profile 定义

在烟感产品的开发空间完成Profile定义。

服务名称	描述	最后修改时间	操作
Smoke		2018/09/17 10:25:15	
<b>属性列表</b>			
level	数据类型 int 范围 0 ~ 3 步长 -- 单位 --	是否必选 <input checked="" type="checkbox"/>	访问模式 R  
temperature	数据类型 int 范围 0 ~ 1000 步长 -- 单位 --	是否必选 <input checked="" type="checkbox"/>	访问模式 R  
other_info	数据类型 string 长度 100 步长 -- 单位 --	是否必选 <input checked="" type="checkbox"/>	访问模式 R  

## 编解码插件开发

本章只讲解描述信息（other\_info）上报消息的插件开发步骤，烟雾报警（level）和温度（temperature）上报消息的插件开发步骤，请参见[多条数据上报消息的插件开发](#)。

**步骤1** 在烟感产品的开发空间，选择“编解码插件开发”。



**步骤2** 配置数据上报消息，上报火灾等级和温度，操作步骤详见[步骤2](#)。

**步骤3** 配置数据上报消息，只上报温度，操作步骤详见[步骤3](#)。

**步骤4** 配置数据上报消息，上报字符串类型的描述信息。

新增消息

基本信息

消息名 \*  消息描述

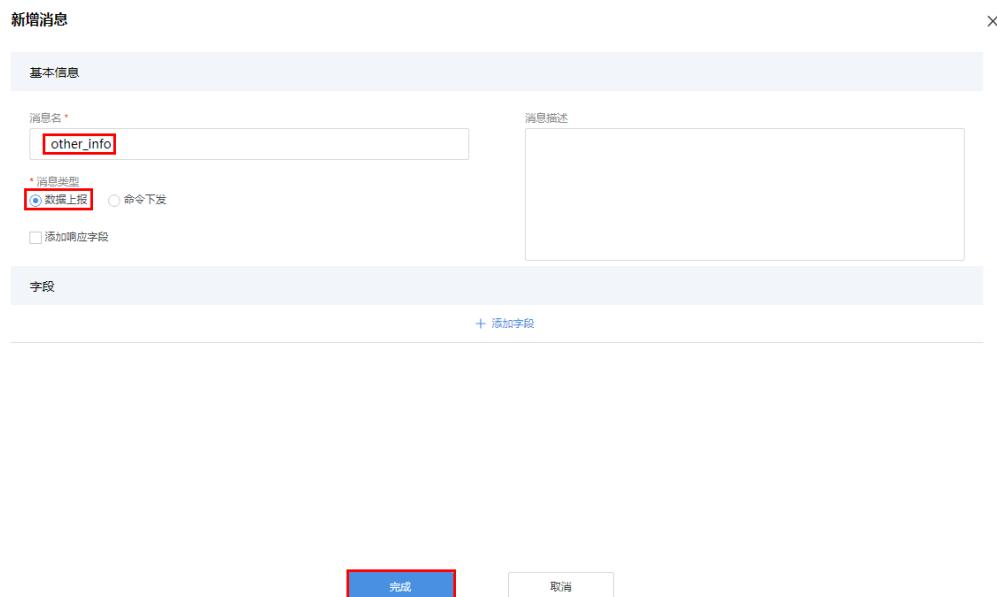
\* 消息类型  数据上报  命令下发

添加响应字段

字段

+ 添加字段

完成 完成 取消



添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x2用于标识上报描述信息（字符串类型）的消息。

## 添加字段

X

标记为地址域 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

描述

数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x2

偏移值 [?](#)

0-1

完成

取消

添加other\_info字段，表示字符串类型的描述信息。在本场景中，“长度”配置6个字节。

### 添加字段

标记为地址域 [?](#)

\* 名字

描述

数据类型

\* 长度 [?](#)

默认值 [?](#)

偏移值 [?](#)

完成 取消

**步骤5** 配置数据上报消息，上报可变长度字符串类型的描述信息。

新增消息

基本信息

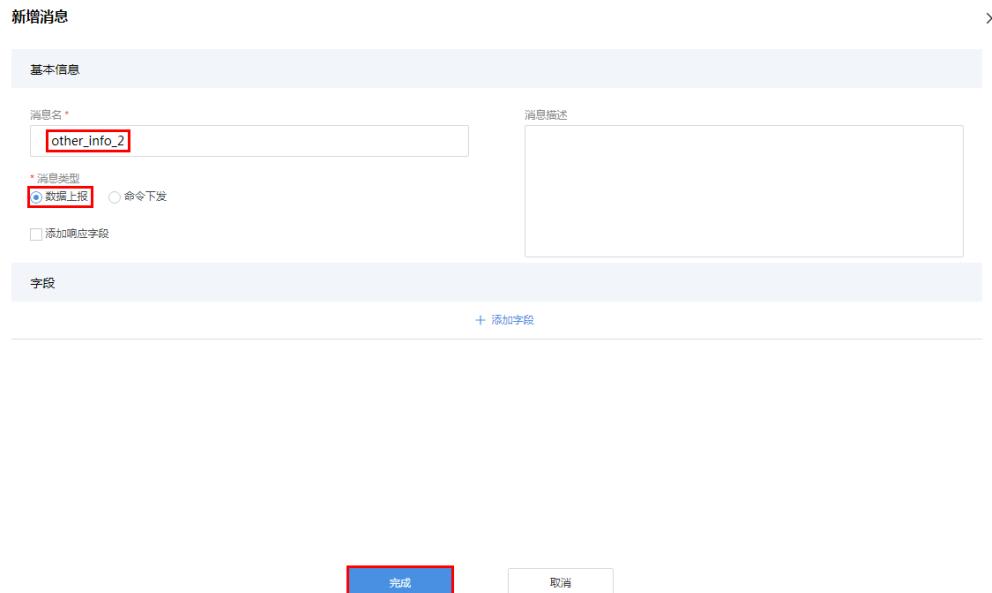
消息名 \*  消息描述

\* 消息类型  数据上报  命令下发

添加响应字段

字段 [+ 添加字段](#)

[完成](#) [取消](#)



添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x3用于标识上报描述信息（可变长度字符串类型）的消息。

### 添加字段

标记为地址域 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

描述

数据类型

int8u(8位无符号整型)

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x3

偏移值 [?](#)

0-1

[完成](#) [取消](#)

添加length字段，表示字符串长度。“数据类型”根据可变长度字符串的长度进行配置，长度在255以内，配置为“int8u”。

## 添加字段

X

标记为地址域 [?](#)

\* 名字

length

描述

数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

默认值 [?](#)

偏移值 [?](#)

1-2

完成

取消

添加other\_info字段，表示可变长度字符串类型的描述信息。“长度关联字段”选择“length”，“长度关联字段差值”和“数值长度”自动填充。

添加字段 X

标记为地址域 [?](#)

\* 名字

描述

数据类型

\* 长度关联字段 [?](#) \* 长度关联字段差值 [?](#)

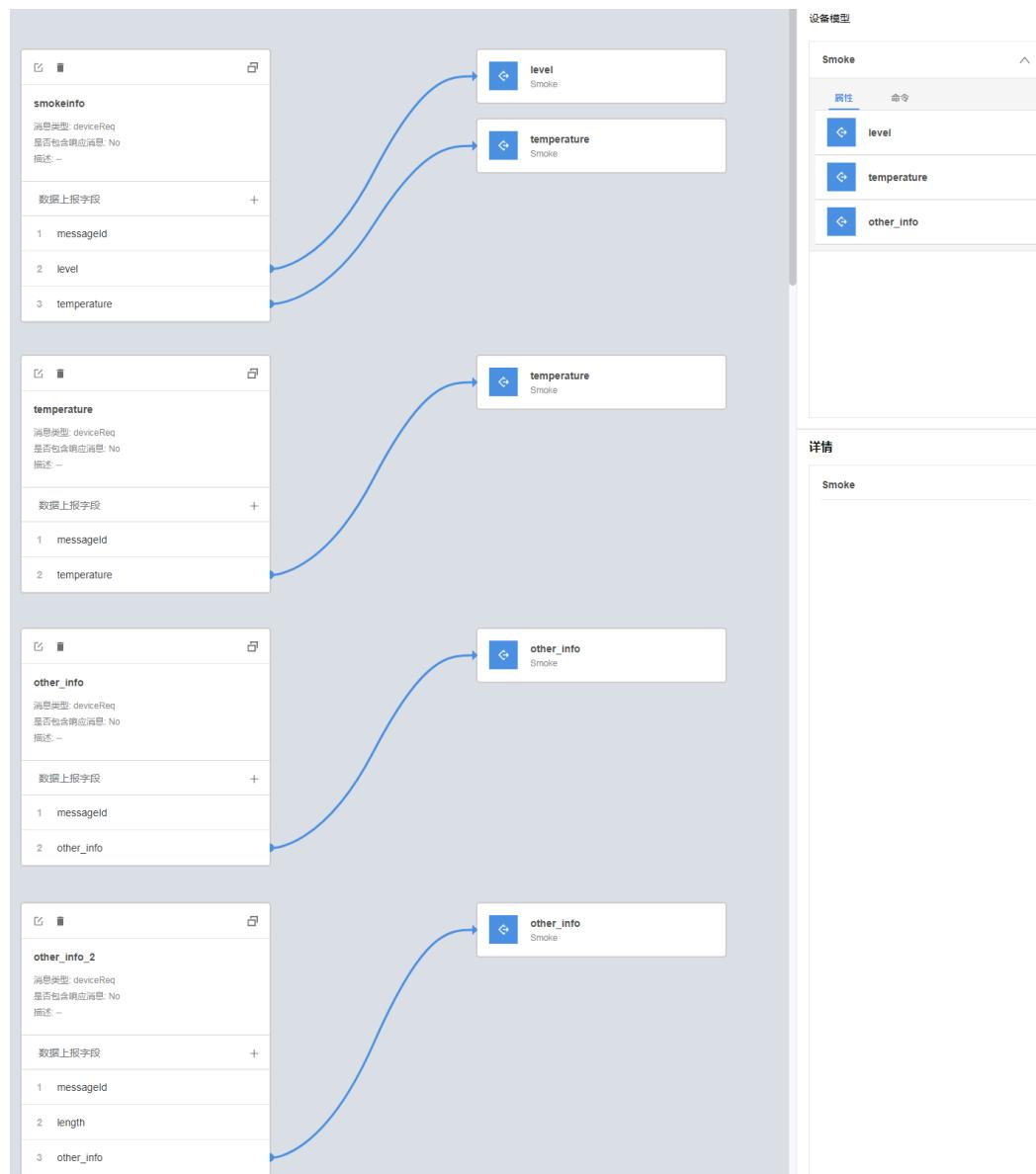
数值长度 [?](#) \* 默认值 [?](#)

掩码 [?](#)

偏移值 [?](#)

完成 取消

**步骤6** 拖动右侧“设备模型”区域的属性字段，与数据上报消息的相应字段建立映射关系。



**步骤7** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 调测编解码插件

**步骤1** 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。



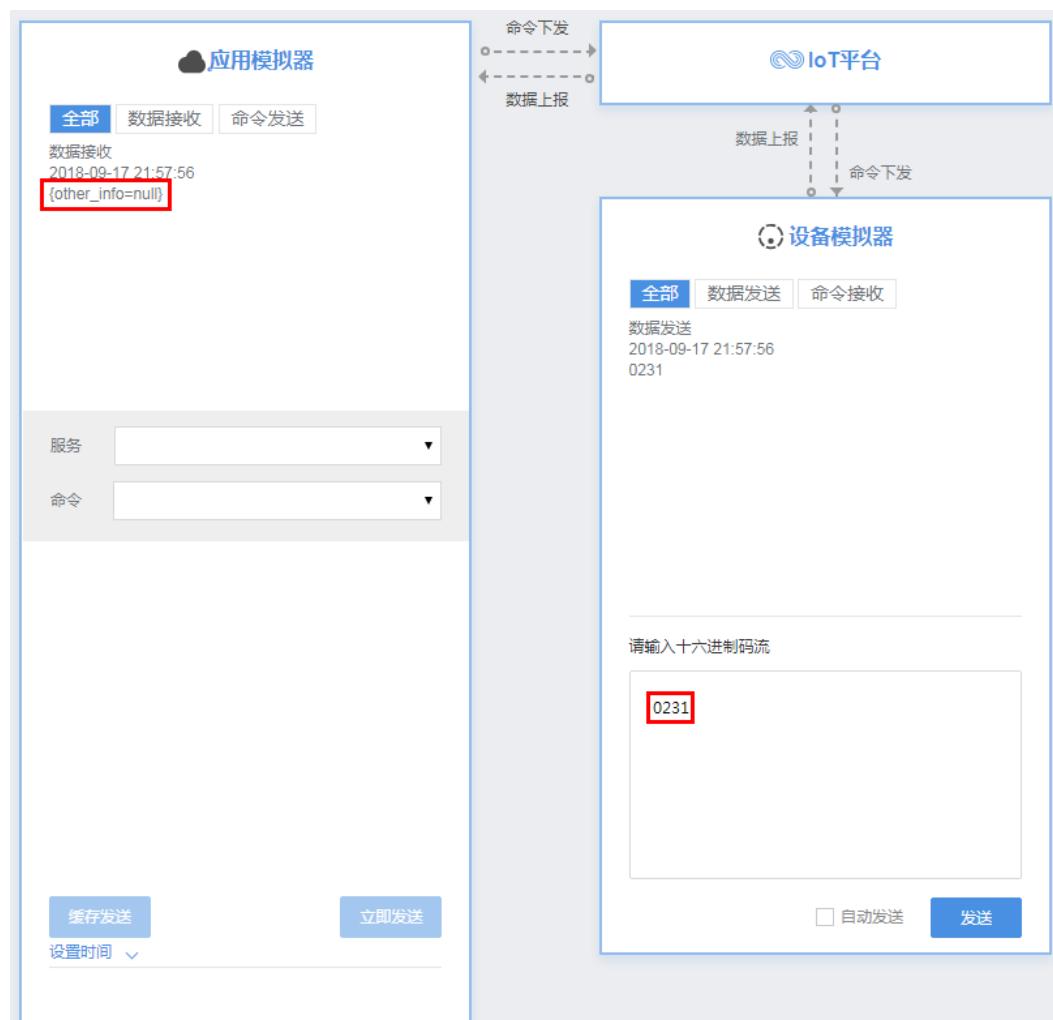
勾选“没有真实的物理设备”，点击“创建”。



## 步骤2 使用设备模拟器上报字符串类型的描述信息。

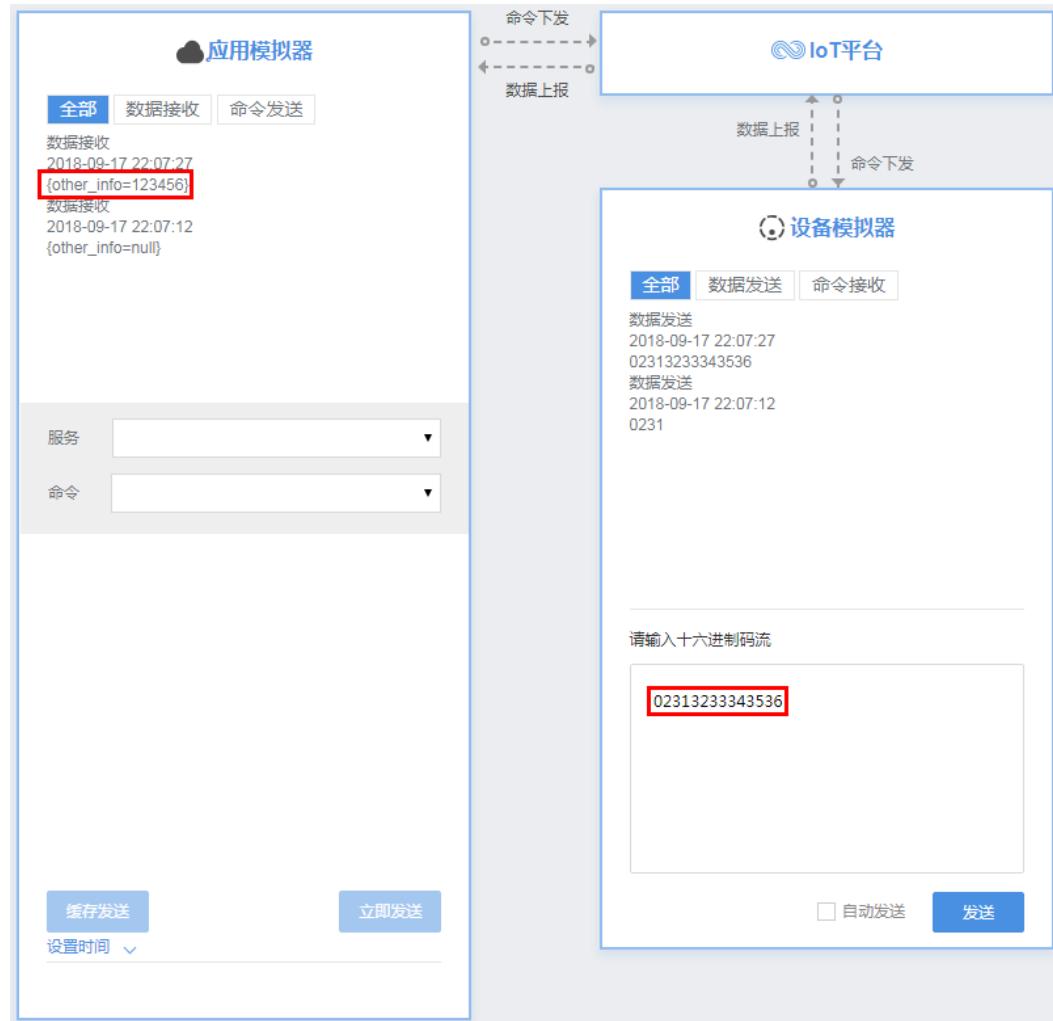
十六进制码流示例：0231。02表示messageId，此消息上报字符串类型的描述信息；31表示描述信息，长度为1个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=null}。描述信息不足6个字节，编解码插件无法解析。



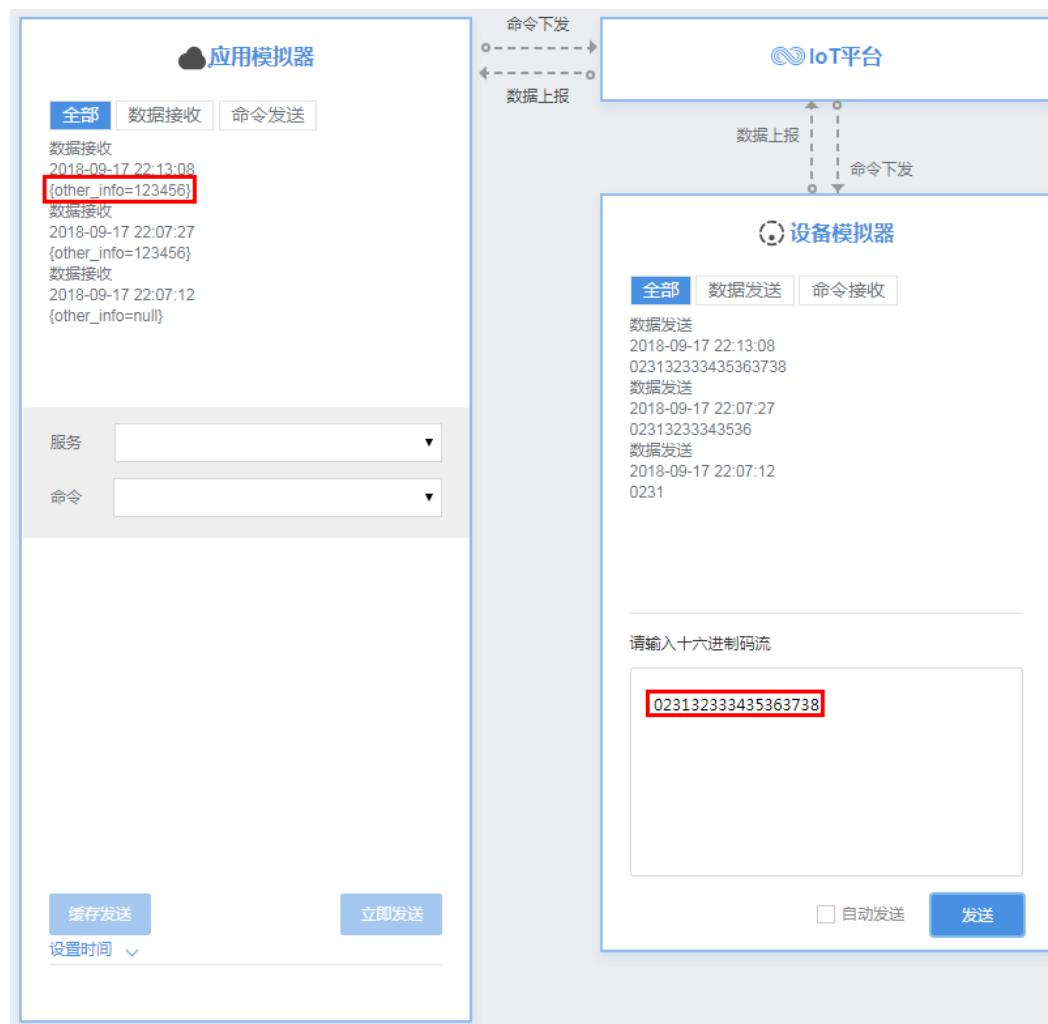
十六进制码流示例：02313233343536。02表示messageId，此消息上报字符串类型的描述信息；313233343536表示描述信息，长度为6个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=123456}。描述信息长度为6个字节，编解码插件解析成功。



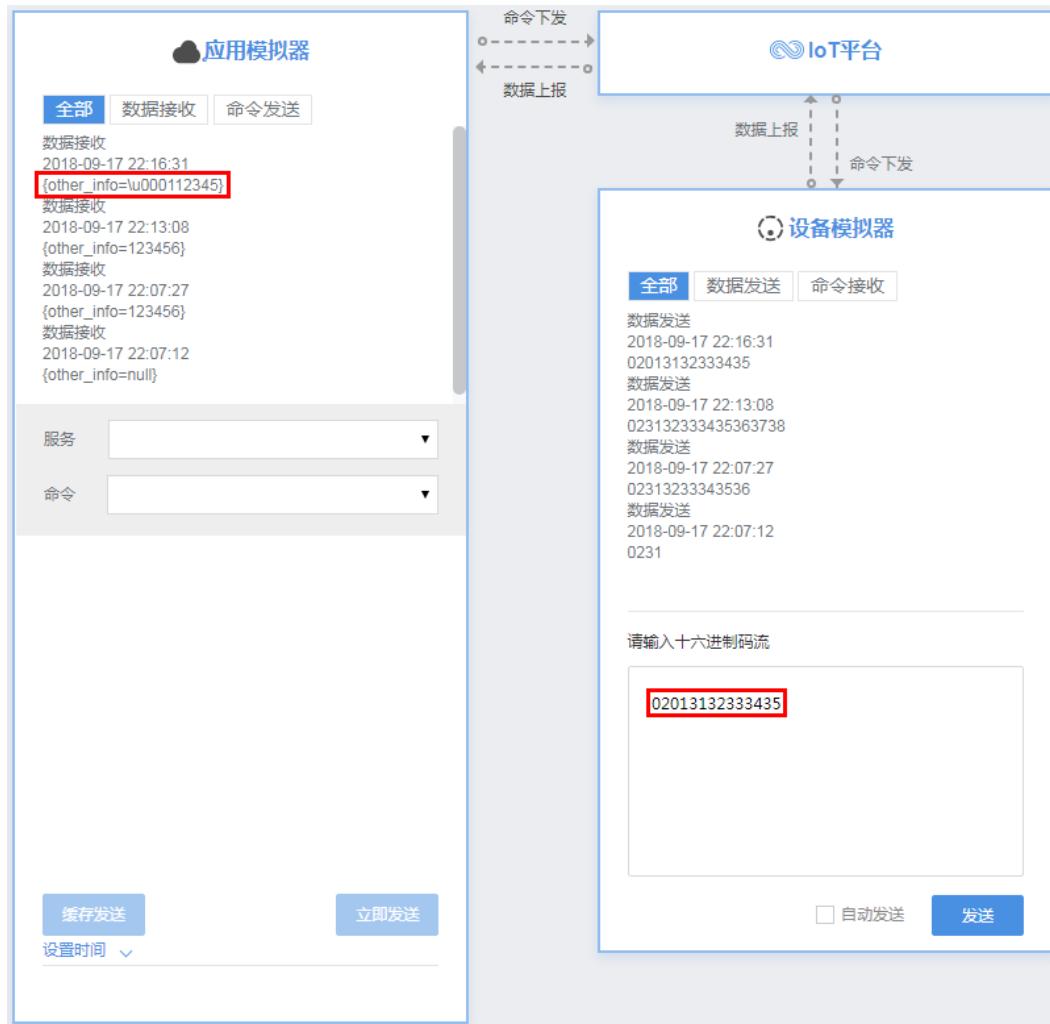
十六进制码流示例：023132333435363738。02表示messageId，此消息上报字符串类型的描述信息；3132333435363738表示描述信息，长度为8个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=123456}。描述信息长度超过6个字节，编解码插件截取前6个字节进行解析。



十六进制码流示例：02013132333435。02表示messageId，此消息上报字符串类型的描述信息；013132333435表示描述信息，长度为6个字节。

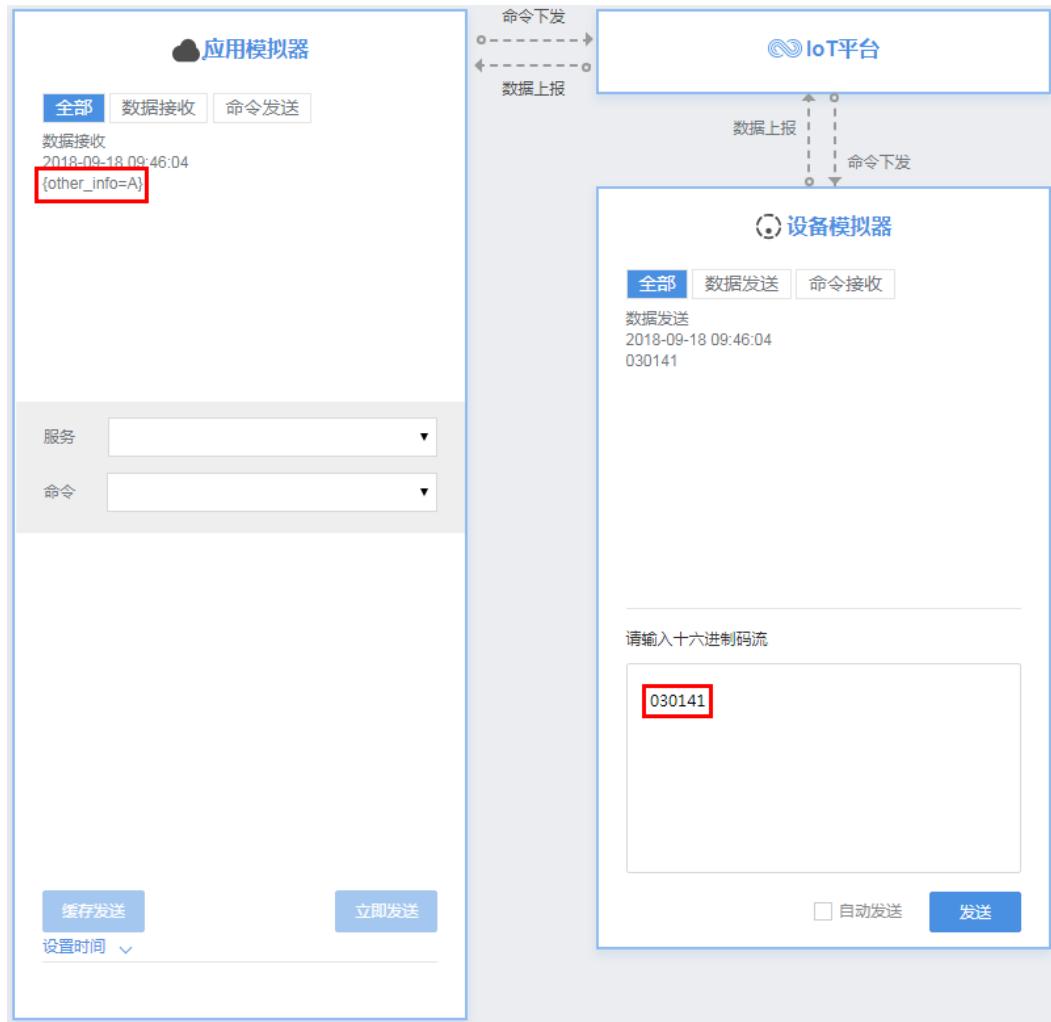
在“应用模拟器”区域查看数据上报的结果：{other\_info=\u000112345}。01在ASCII码表里表示“标题开始”，无法用具体字符表示，因此编解码插件解析为\u0001。



### 步骤3 使用设备模拟器上报可变长度字符串类型的描述信息。

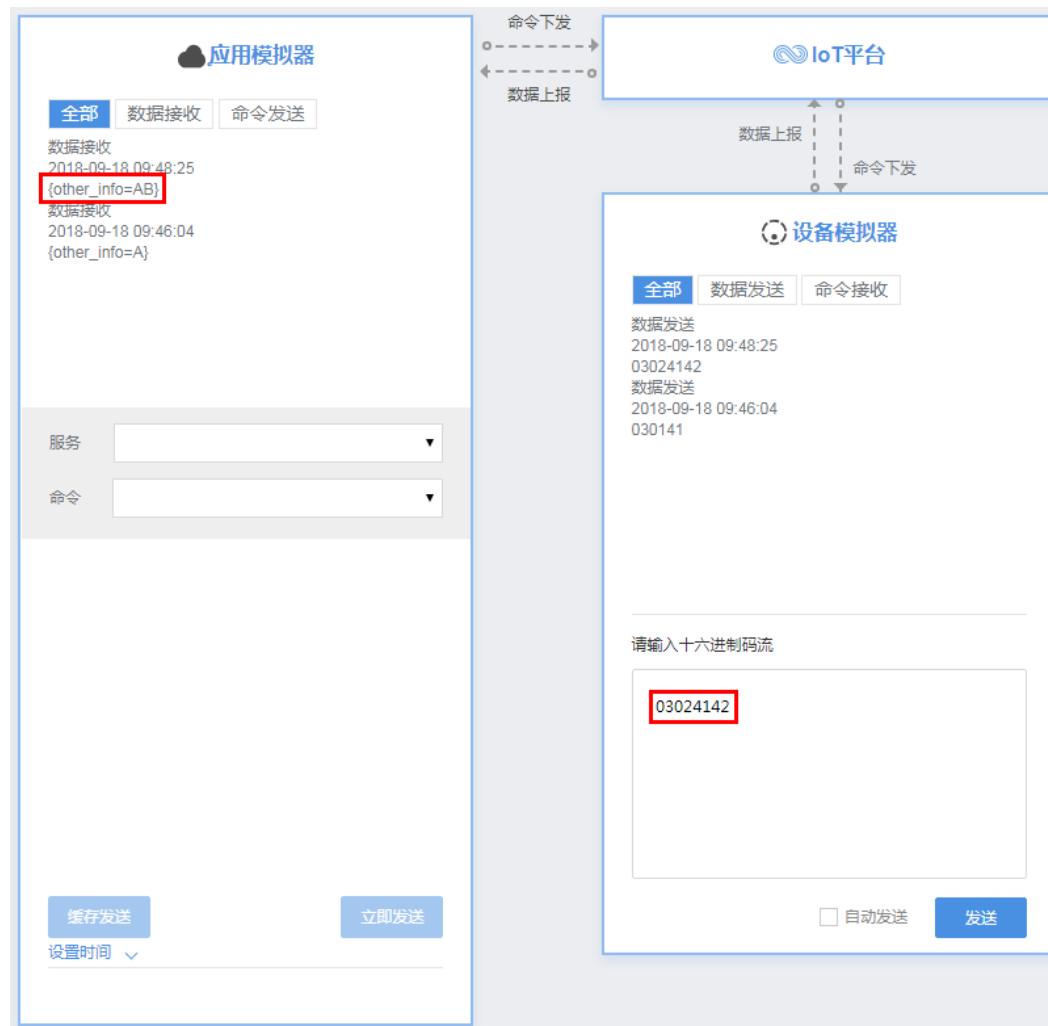
十六进制码流示例：030141。03表示messageId，此消息上报可变长度字符串类型的描述信息；01表示描述信息长度（1个字节），长度为1个字节；41表示描述信息，长度为1个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=A}。41是A的十六进制ASCII码。



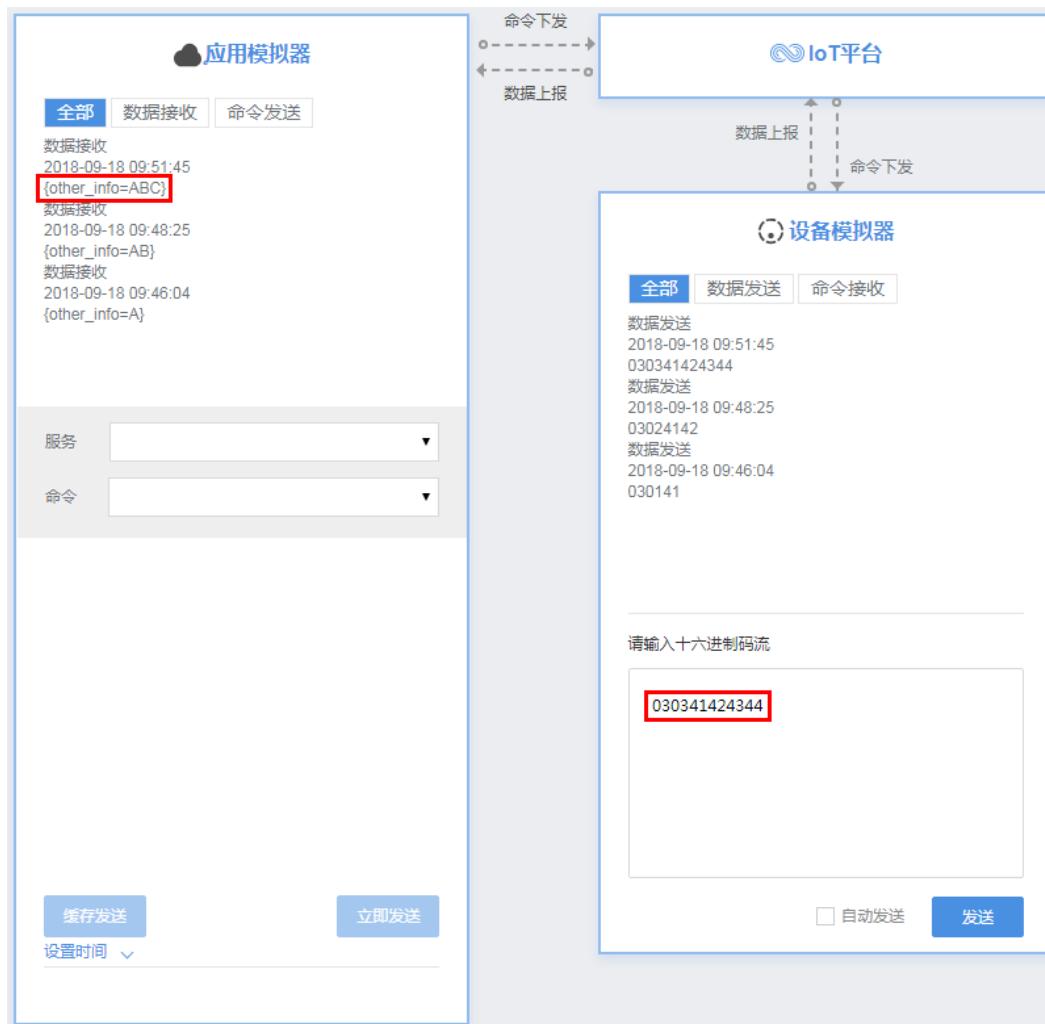
十六进制码流示例：03024142。03表示messageId，此消息上报可变长度字符串类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；4142表示描述信息，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=AB}。4142是AB的十六进制ASCII码。



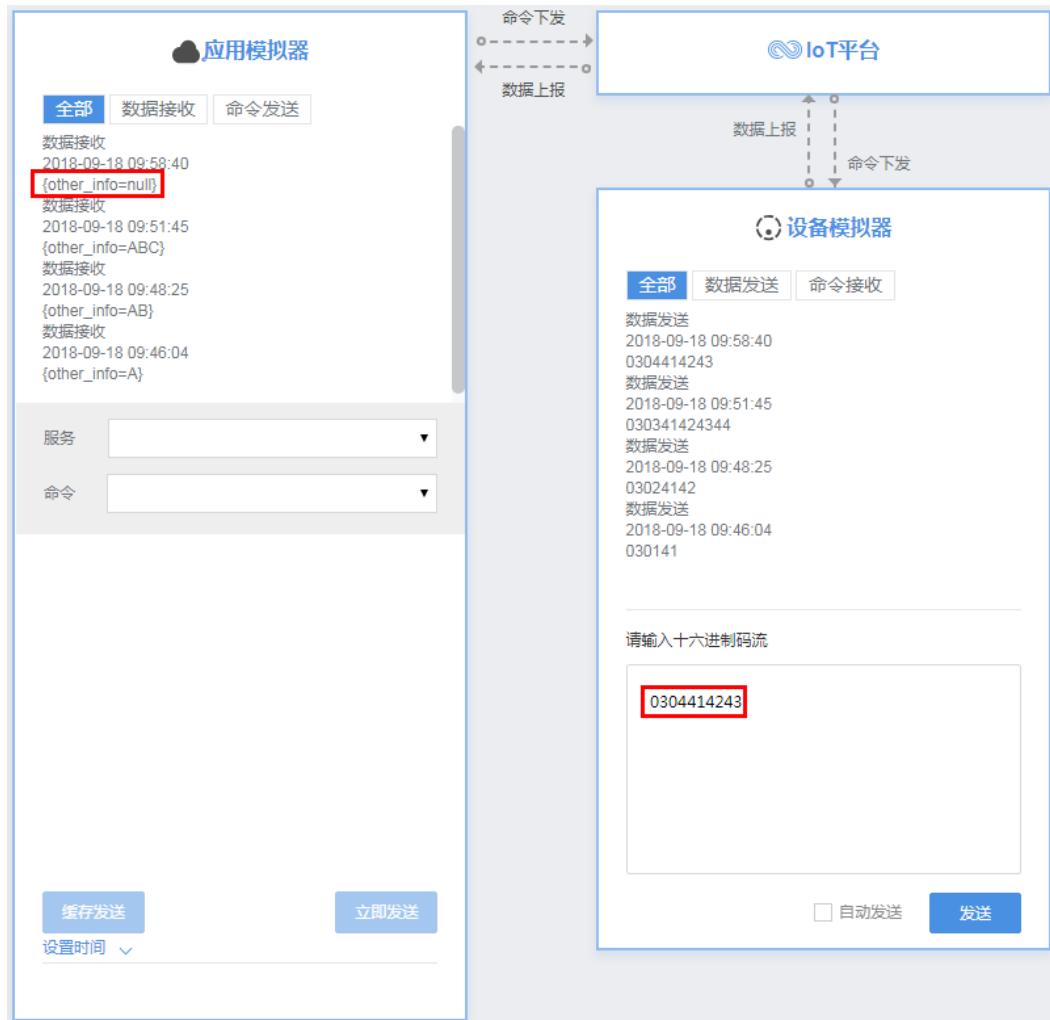
十六进制码流示例：030341424344。03表示messageId，此消息上报可变长度字符串类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；41424344表示描述信息，长度为4个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=ABC}。描述信息长度超过3个字节，编解码插件截取前3个字节进行解析，414243是ABC的十六进制ASCII码。



十六进制码流示例：0304414243。03表示messageId，此消息上报可变长度字符串类型的描述信息；04表示字符串长度（4个字节），长度为1个字节；414243表示描述信息，长度为4个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=null}。描述信息长度不足4个字节，编解码插件解析失败。



----结束

## 总结

- 当数据类型为字符串或可变长度字符串时，插件是按照ASCII码进行编解码的：上报数据时，将16进制码流解码为对应字符串，比如：21解析为“！”、31解析为“1”、41解析为“A”；下发命令时，将字符串编码对应的16进制码流，比如：“！”编码为21，“1”编码为31，“A”编码为41。
- 当某字段的数据类型为可变长度字符串时，该字段需要关联长度字段，长度字段的数据类型必须为int。
- 针对可变长度字符串，命令下发和数据上报的编解码插件开发方式相同。
- 在线开发的编解码插件使用ASCII码16进制的标准表对字符串和可变长度字符串进行编解码。解码时（数据上报），如果解析结果无法使用具体字符表示，如：标题开始、正文开始、正文结束等，则使用\u+2字节码流值表示（例如：01解析为\u0001，02解析为\u0002）；如果解析结果可以使用具体字符表示，则使用具体字符。

### 3.2.3.2.4 数组及可变长数组数据类型的插件开发

#### 场景说明

有一款烟感设备，具有如下特征：

- 具有烟雾报警功能（火灾等级）和温度上报功能，支持同时上报烟雾报警（火灾等级）和温度，也支持单独上报温度。
- 具备描述信息上报功能，描述信息支持数组和可变长度数组两种类型。

#### 说明

该场景旨在讲解数组及可变长度数组数据类型的编解码插件开发方式，数据上报和命令下发的插件开发方式类似，本章节以数据上报为例进行说明，因此省略命令下发的相关步骤。

## Profile 定义

在烟感产品的开发空间完成Profile定义。

服务名称	描述	最后修改时间	操作
Smoke		2018/09/17 10:25:15	
<strong>属性列表</strong>			
level	数据类型 int 范围 0 ~ 3 步长 -- 单位 --	是否必选 <input checked="" type="checkbox"/> 访问模式 R	
temperature	数据类型 int 范围 0 ~ 1000 步长 -- 单位 --	是否必选 <input checked="" type="checkbox"/> 访问模式 R	
other_info	数据类型 string 长度 100 单位 --	是否必选 <input checked="" type="checkbox"/> 访问模式 R	

## 编解码插件开发

本章只讲解描述信息（other\_info）上报消息的插件开发步骤，烟雾报警（level）和温度（temperature）上报消息的插件开发，请参见[多条数据上报消息的插件开发](#)。

**步骤1** 在烟感产品的开发空间，选择“编解码插件开发”。



**步骤2** 配置数据上报消息，上报火灾等级和温度，操作步骤详见**步骤2**。

**步骤3** 配置数据上报消息，只上报温度，操作步骤详见**步骤3**。

**步骤4** 配置数据上报消息，上报数组类型的描述信息。

The screenshot shows the '新增消息' (Add Message) dialog. The '基本信息' (Basic Information) tab is selected. In the '消息名\*' (Message Name) field, 'other\_info' is entered. The '消息类型' (Message Type) section has '数据上报' (Data Report) selected. Below it, there is a checkbox for '添加响应字段' (Add Response Field) which is unchecked. The '字段' (Fields) tab is visible at the bottom, showing a '+' button to add fields. At the bottom right are '完成' (Finish) and '取消' (Cancel) buttons, with '完成' highlighted by a red box.

添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x2用于标识上报描述信息（数组类型）的消息。

添加字段 X

标记为地址域 (?)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

描述

数据类型

int8u(8位无符号整型) ▼

\* 长度 (?)

1

\* 默认值 (?)

0x2

偏移值 (?)

0-1

完成 取消

添加other\_info字段，表示数组类型的描述信息。在本场景中，“长度”配置为5个字节。

添加字段

标记为地址域 [?](#)

\* 名字

other\_info

描述

数据类型

array(数组类型)

\* 长度 [?](#)

5

默认值 [?](#)

偏移值 [?](#)

1-6

[完成](#) [取消](#)

**步骤5** 配置数据上报消息，上报可变长度数组类型的描述信息。

新增消息

基本信息

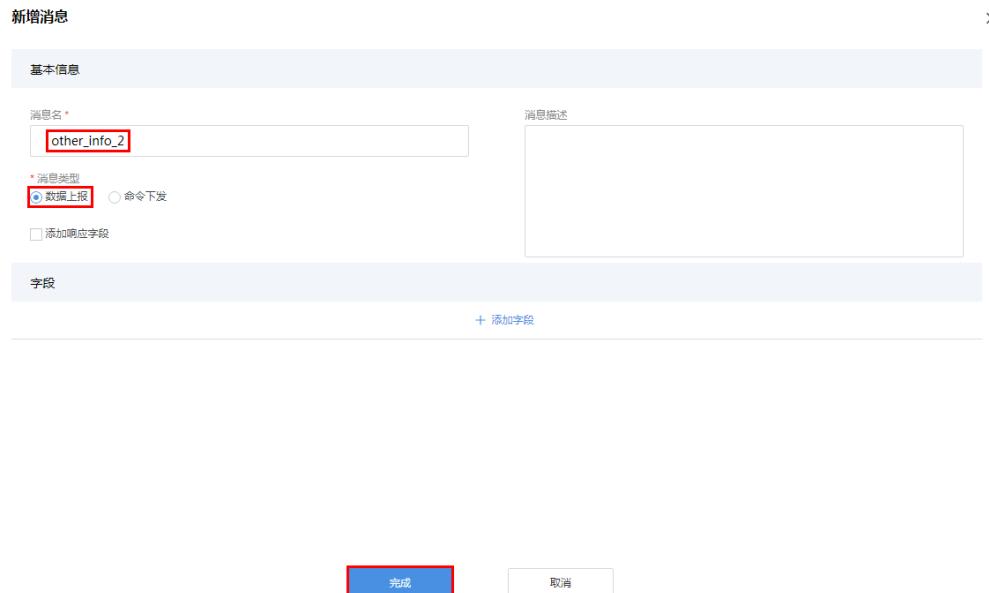
消息名 \*  消息描述

\* 消息类型  数据上报  命令下发

添加响应字段

字段 [+ 添加字段](#)

[完成](#) [取消](#)



添加messageId字段，表示消息种类。在本场景中，0x0用于标识上报火灾等级和温度的消息，0x1用于标识只上报温度的消息，0x3用于标识上报描述信息（可变长度数组类型）的消息。

## 添加字段

 标记为地址域 

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

## 描述

## 数据类型

int8u(8位无符号整型)



\* 长度

1

\* 默认值

0x3

偏移值

0-1

完成

取消

添加length字段，表示数组长度。“数据类型”根据可变长度数组的长度进行配置，长度在255以内，配置为“int8u”。

### 添加字段

标记为地址域 [?](#)

\* 名字

描述

数据类型

\* 长度 [?](#)

默认值 [?](#)

偏移值 [?](#)

完成 取消

添加other\_info字段，表示可变长度数组类型的描述信息。“长度关联字段”选择“length”，“长度关联字段差值”和“数值长度”自动填充。

添加字段

标记为地址域 [?](#)

\* 名字

other\_info

描述

数据类型

variant(可变长度数组类型)

\* 长度关联字段 [?](#) \* 长度关联字段差值 [?](#)

length 0

数值长度 [?](#) \* 默认值 [?](#)

1

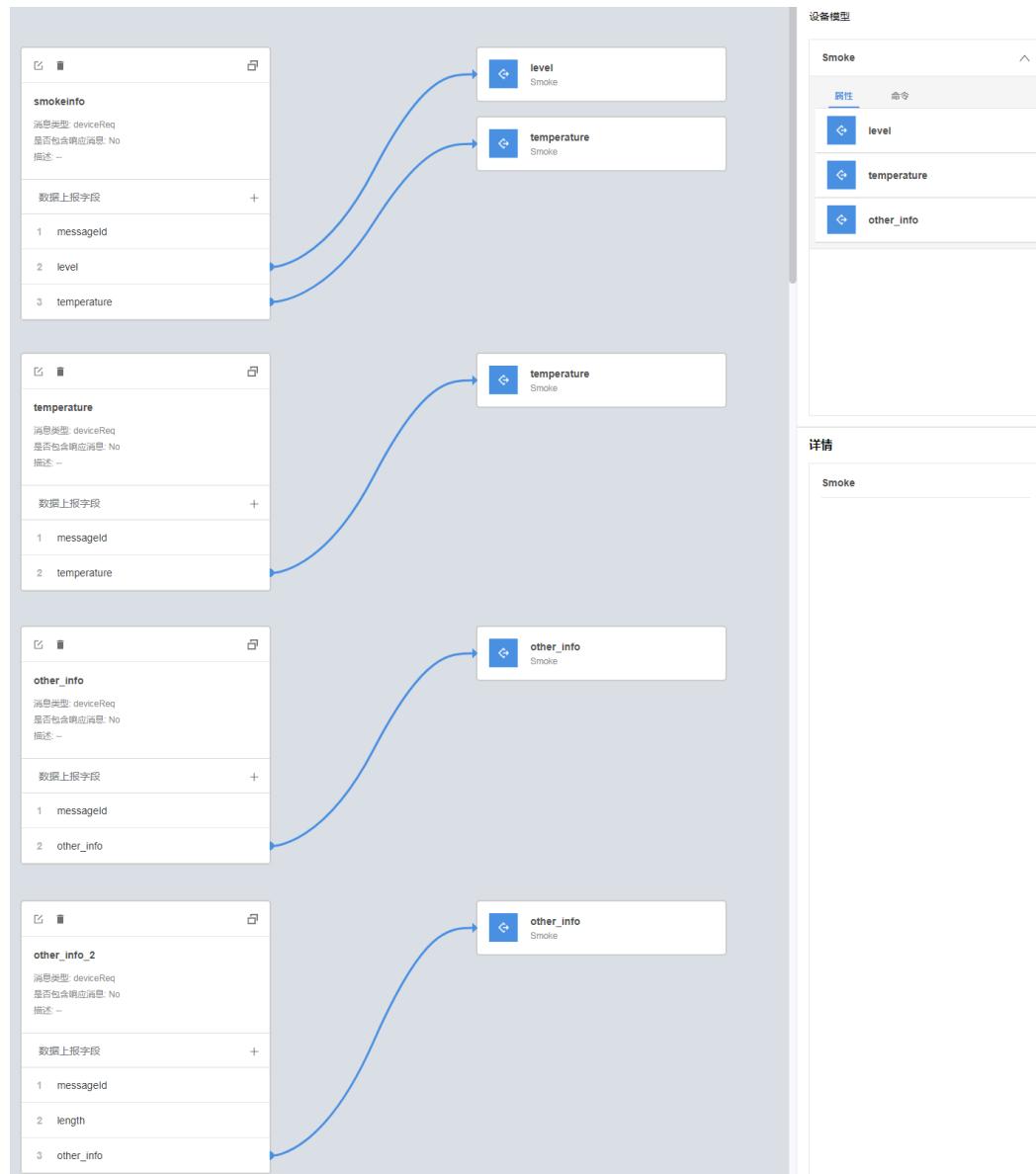
掩码 [?](#)

偏移值 [?](#)

2-3

[完成](#) [取消](#)

**步骤6** 拖动右侧“设备模型”区域的属性字段，与数据上报消息的相应字段建立映射关系。



**步骤7** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 调测编解码插件

**步骤1** 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。



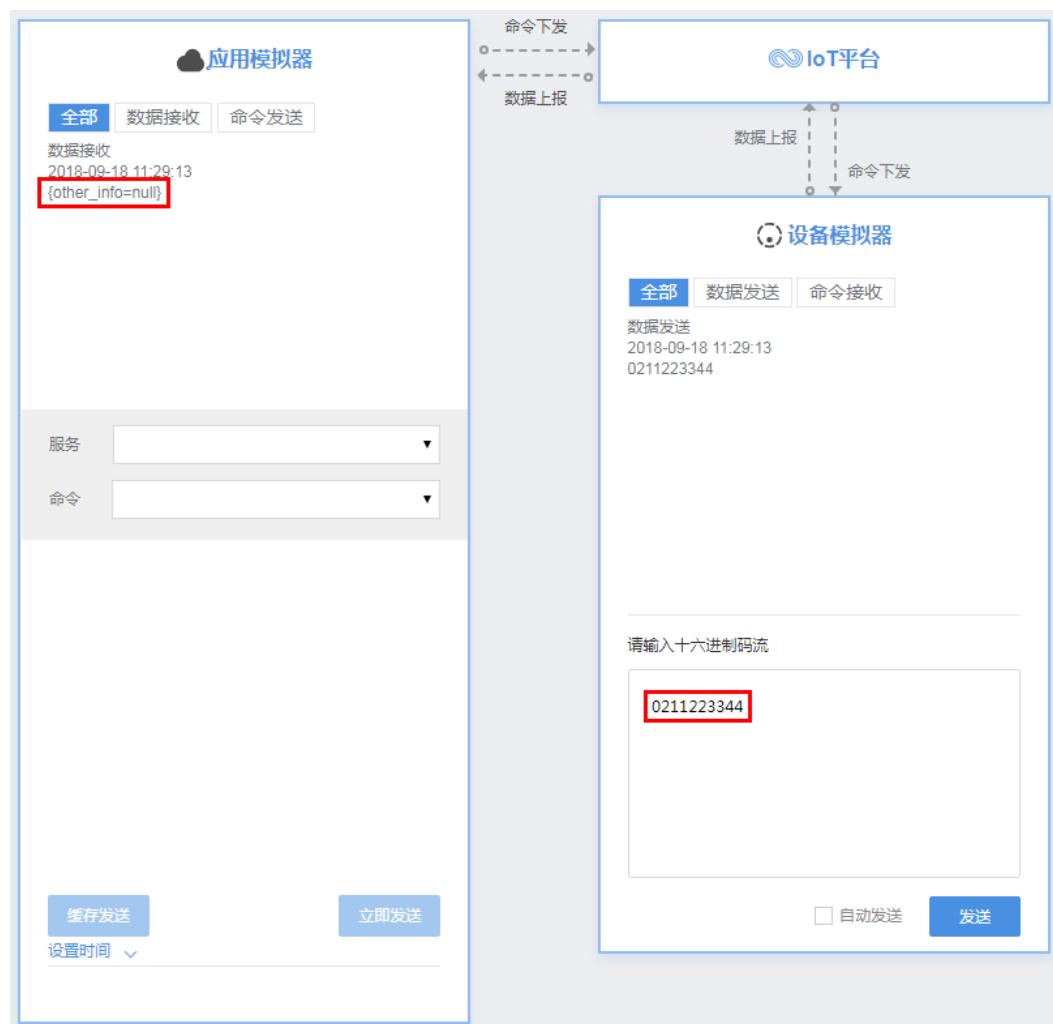
勾选“没有真实的物理设备”，点击“创建”。



## 步骤2 使用设备模拟器上报数组类型的描述信息。

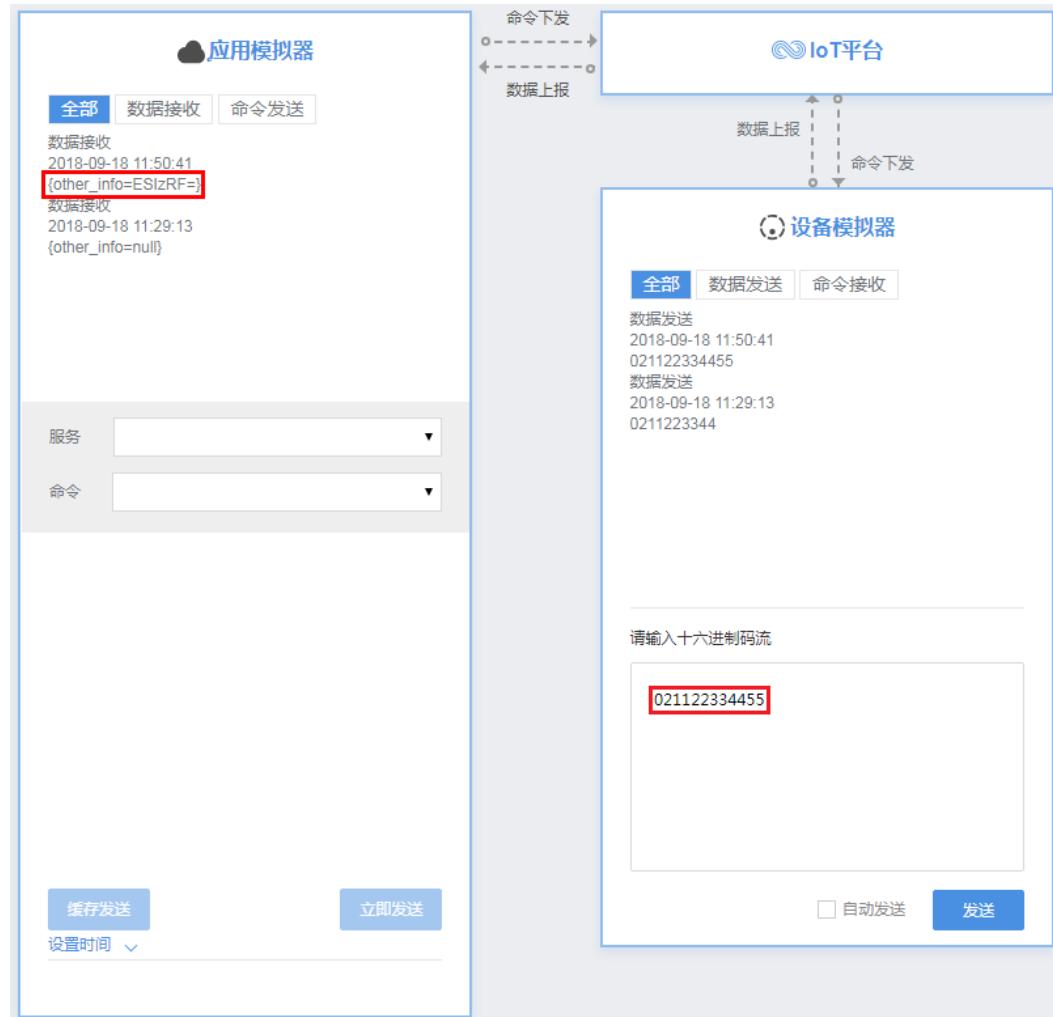
十六进制码流示例：0211223344。02表示messageId，此消息上报数组类型的描述信息；11223344表示描述信息，长度为4个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=null}。描述信息不足5个字节，编解码插件无法解析。



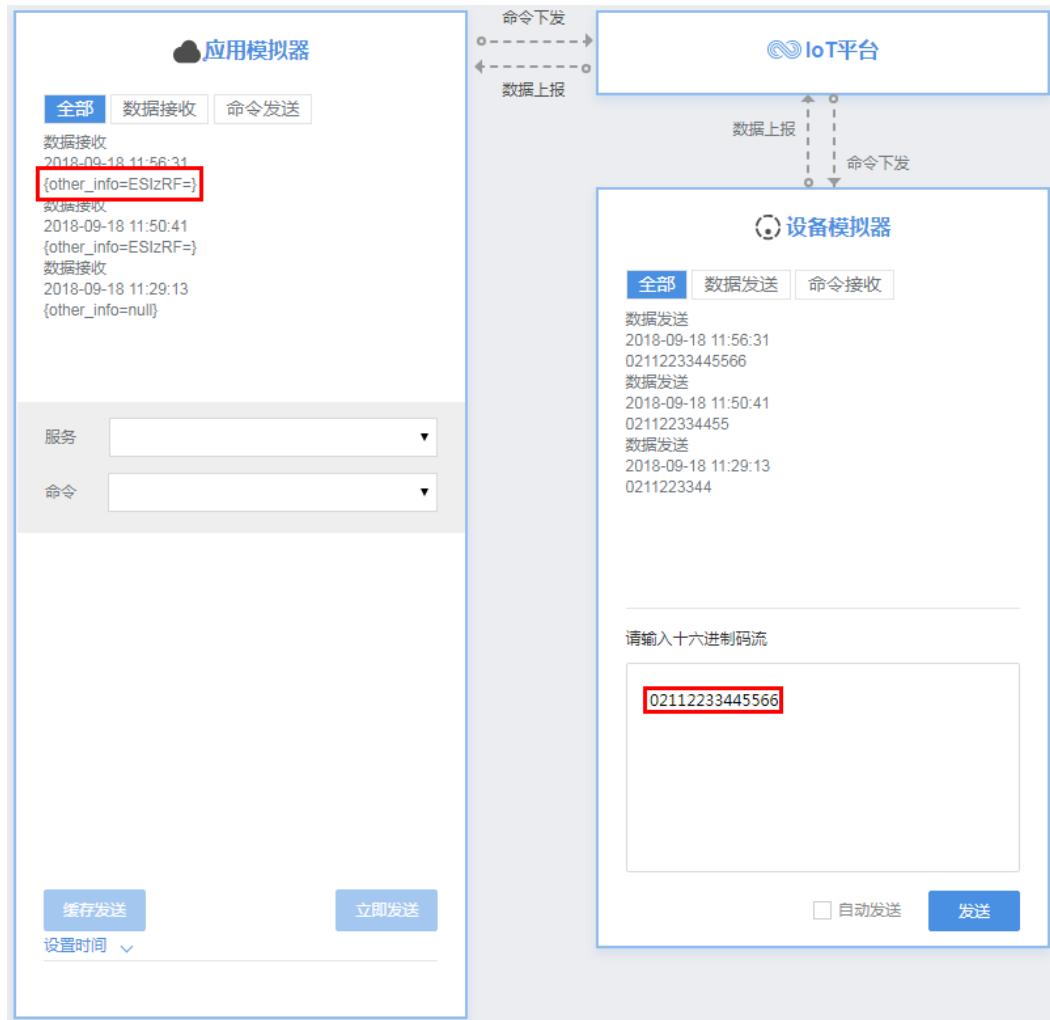
十六进制码流示例：021122334455。02表示messageId，此消息上报数组类型的描述信息；1122334455表示描述信息，长度为5个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=ESIzRF=}。描述信息长度为5个字节，编解码插件解析成功。



十六进制码流示例：02112233445566。02表示messageId，此消息上报数组类型的描述信息；112233445566表示描述信息，长度为6个字节。

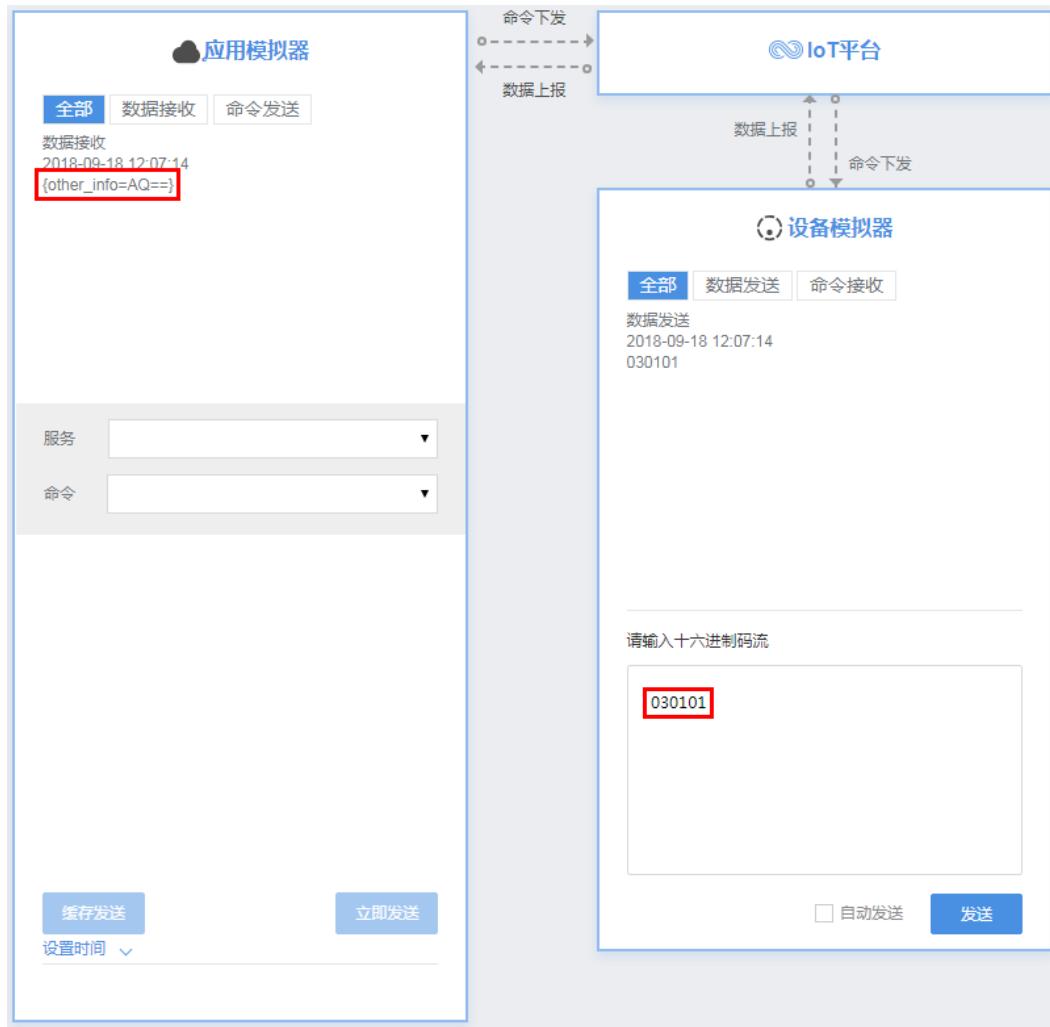
在“应用模拟器”区域查看数据上报的结果：{other\_info=ESIzRF=}。描述信息长度超过5个字节，编解码插件截取前5个字节进行解析。



### 步骤3 使用设备模拟器上报可变长度数组类型的描述信息。

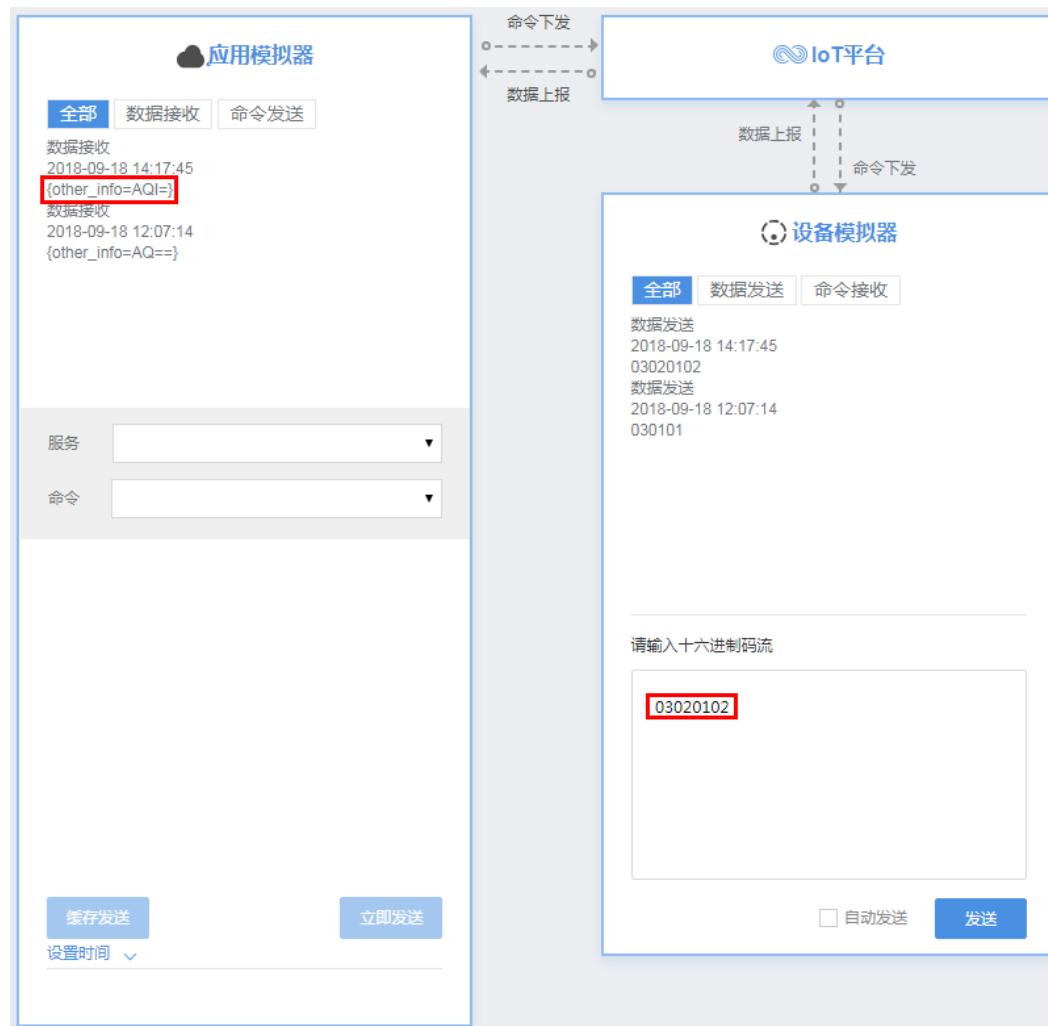
十六进制码流示例：030101。03表示messageId，此消息上报可变长度数组类型的描述信息；01表示描述信息长度（1个字节），长度为1个字节；01表示描述信息，长度为1个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=AQ==}。AQ==是01经过base64编码后的值。



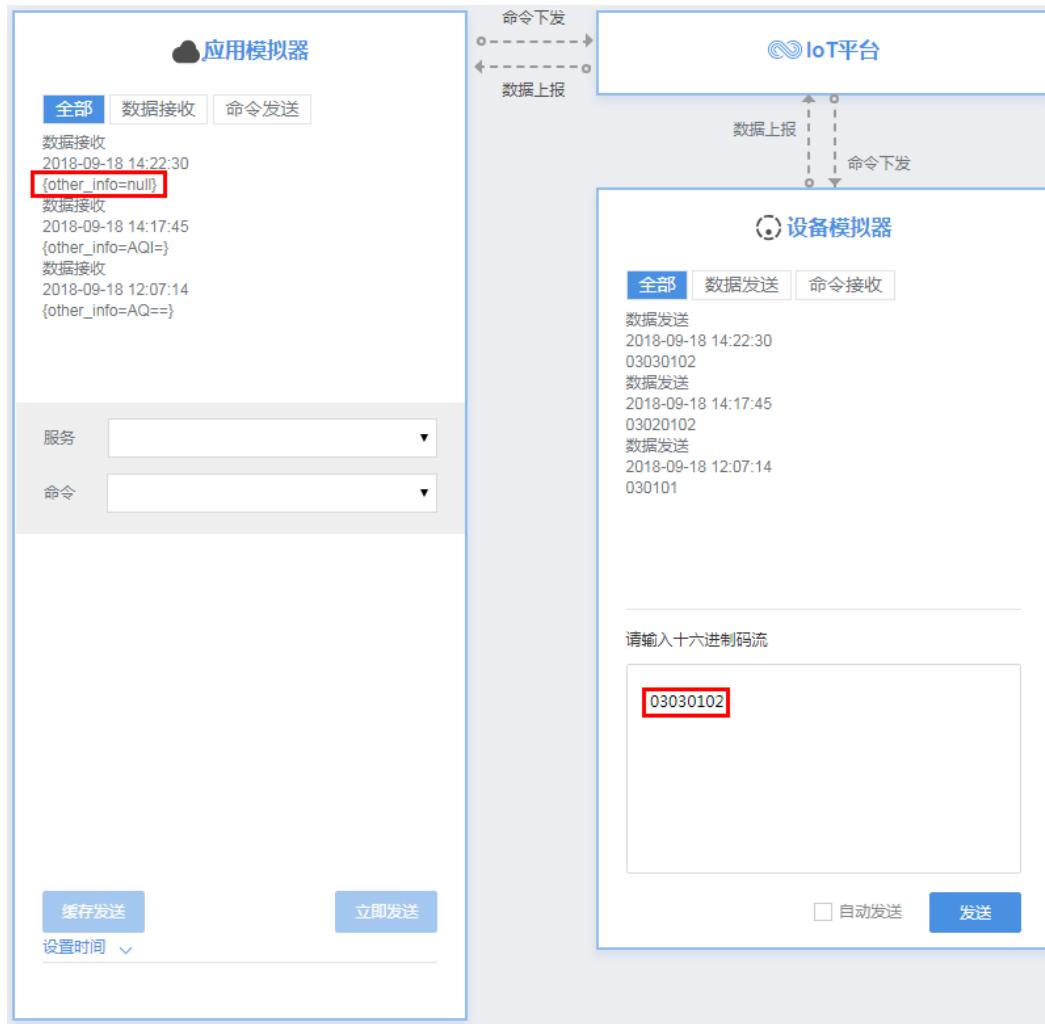
十六进制码流示例：03020102。03表示messageId，此消息上报可变长度数组类型的描述信息；02表示描述信息长度（2个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=AQI=}。AQI=是01经过base64编码后的值。



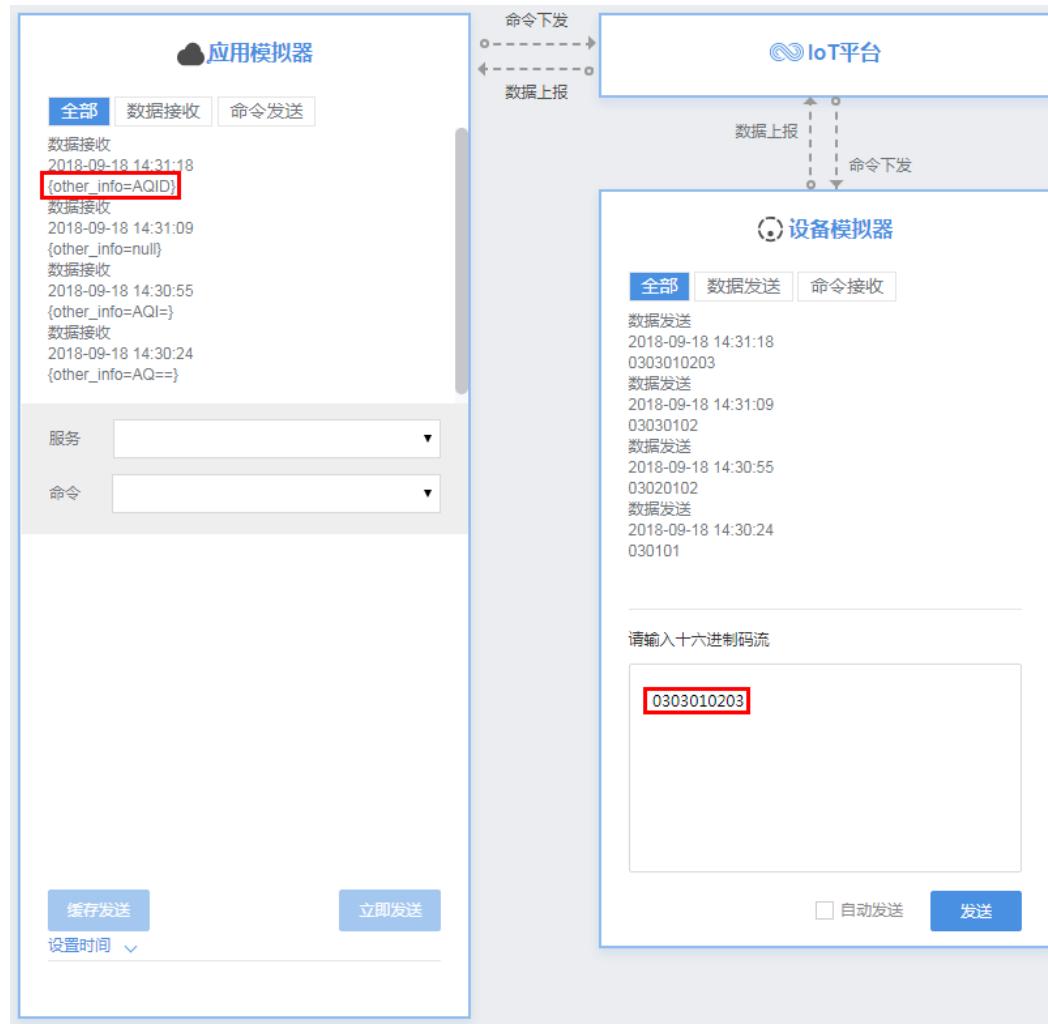
十六进制码流示例：03030102。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；0102表示描述信息，长度为2个字节。

在“应用模拟器”区域查看数据上报的结果：{other\_info=null}。描述信息长度不足3个字节，编解码插件解析失败。



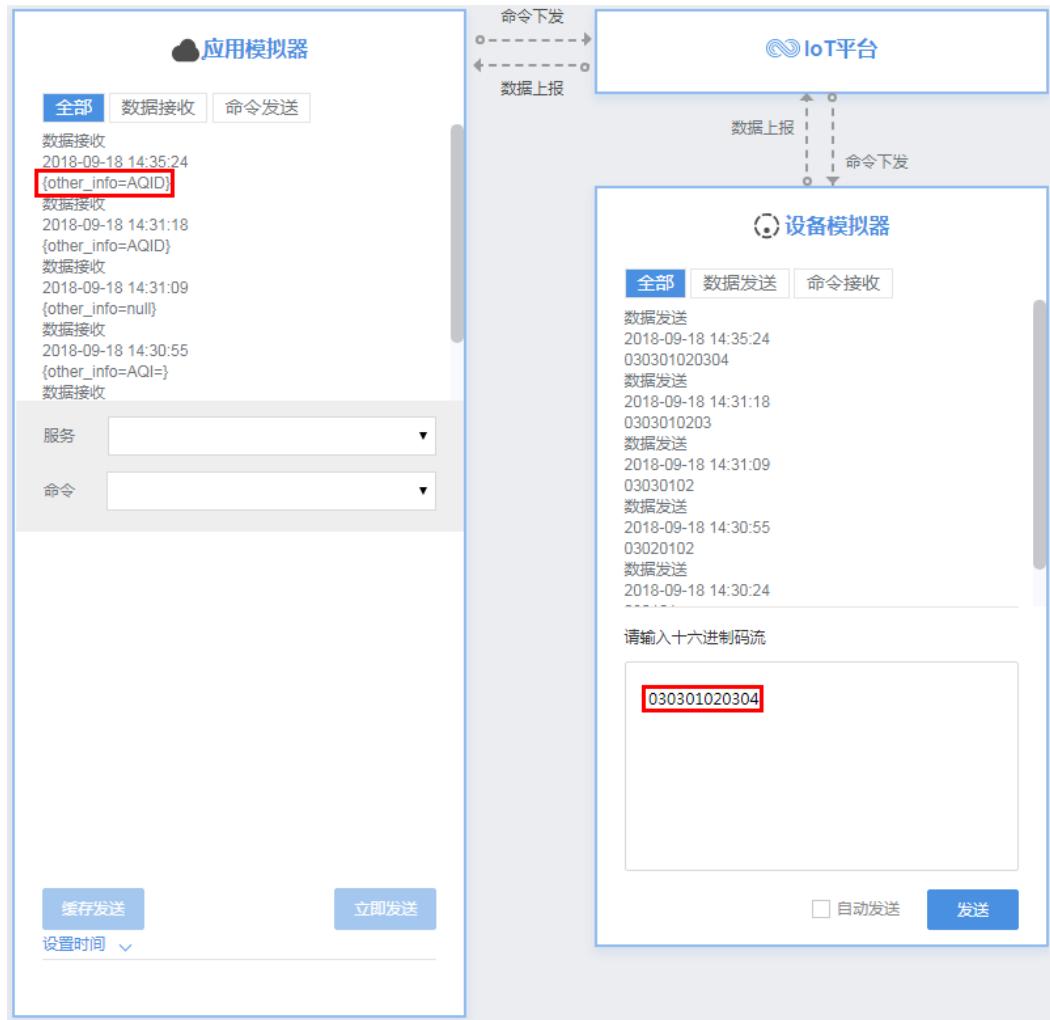
十六进制码流示例：0303010203。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；010203表示描述信息，长度为3个字节。

在“应用模拟器”区域查看数据上报的结果：`{other_info=AQID}`。AQID是010203经过base64编码后的值。



十六进制码流示例：030301020304。03表示messageId，此消息上报可变长度数组类型的描述信息；03表示描述信息长度（3个字节），长度为1个字节；01020304表示描述信息，长度为4个字节。

在“应用模拟器”区域查看数据上报的结果：`{other_info=AQID}`。描述信息长度超过3个字节，编解码插件截取前3个字节进行解析，AQID是010203经过base64编码后的值。



----结束

## base64 编码方式说明

base64编码方式会把3个8位字节（ $3 \times 8 = 24$ ）转化为4个6位字节（ $4 \times 6 = 24$ ），并在每个6位字节前补两个0，构成4个8位字节的形式。如果要进行编码的码流不足3个字节，则用0填充，使用0填充的字节经编码输出的字符为“=”。

base64可以将16进制码流当做字符或者数值进行编码，两种方式获得的编码结果不同。以16进制码流01为例进行说明：

- 把01当作字符，不足3个字符，补1个0，得到010。通过查询ASCII码表，将字符转换为8位二进制数，即：0转换为00110000、1转换为00110001，因此010可以转换为001100000011000100110000（ $3 \times 8 = 24$ ）。再转换为4个6位字节：001100、000011、000100、110000，并在每个6位字节前补两个0，得到：00001100、00000011、00000100、00110000。这4个8位字节对应的10进制数分别为12、3、4、48，通过查询base64编码表，获得M(12)、D(3)、E(4)，由于3个字符中，最后一个字符通过补0获得，因此第4个8位字节使用“=”表示。最终，把01当做字符，通过base64编码得到MDE=。
- 把01当作数值（即1），不足3个字符，补两个0，得到100。将数值转换为8位2进制数，即：0转换为00000000、1转换为00000001，因此100可以转换为00000001000000000000000000（ $3 \times 8 = 24$ ）。在转换为4个6位字节：000000、

010000、000000、000000，并在每个6位字节前补两个0，得到：00000000、00010000、00000000、00000000。这4个8位字节对应的10进制数分别为：0、16、0、0，通过查询base64编码表，获得A(0)、Q(16)，由于3个数值中，最后两个数值通过补0获得，因此第3、4个8位字节使用“=”表示。最终，把01当作数值，通过base64编码得到AQ==。

## 总结

- 当数据类型为数组或可变长度数组时，插件是按照base64进行编解码的：上报数据时，将16进制码流进行base64编码，比如：01编码为“AQ==”；命令下发时，将字符进行base64解码，比如：“AQ==”解码为01。
- 当某字段的数据类型为可变长度数组时，该字段需要关联长度字段，长度字段的数据类型必须为int。
- 针对可变长度数组，命令下发和数据上报的编解码插件开发方式相同。
- 在线开发的编解码插件使用base64进行编码时，是将16进制码流当做**数值**进行编码。

### 3.2.3.2.5 含命令执行结果的编解码插件开发

#### 场景说明

有一款烟感设备，具有如下特征：

- 具有烟雾报警功能（火灾等级）和温度上报功能。
- 支持远程控制命令，可远程打开报警功能。比如火灾现场温度，远程打开烟雾报警，提醒住户疏散。
- 支持上报命令执行结果。

#### Profile 定义

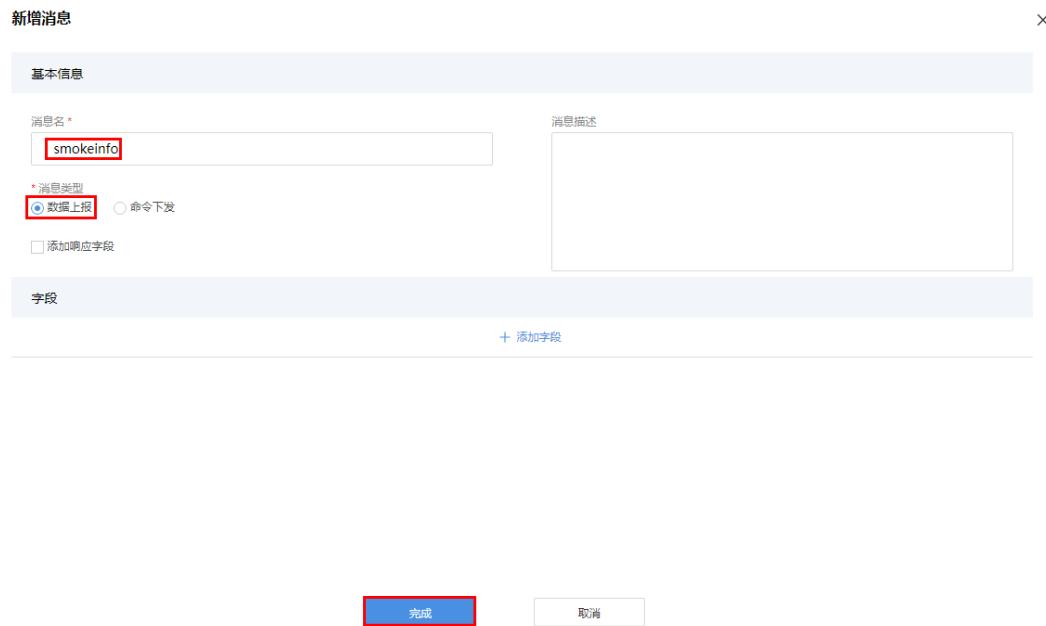
在烟感产品的开发空间完成Profile定义。

The screenshot shows the IoT Platform's Profile configuration interface for a 'Smoke' service. It includes sections for service details, attribute definitions, command definitions, and response field definitions.

- 服务名称:** Smoke
- 属性列表:**
  - level:** 数据类型 int, 范围 0 ~ 3, 步长 --, 单位 --, 是否必选 , 访问模式 R
  - temperature:** 数据类型 int, 范围 0 ~ 1000, 步长 --, 单位 --, 是否必选 , 访问模式 R
- 命令列表:**
  - SET\_ALARM:**
- 下发命令字段:**
  - value:** 数据类型 int, 范围 0 ~ 1, 步长 --, 单位 --, 是否必选
- 响应命令字段:**
  - result:** 数据类型 int, 范围 0 ~ 3, 步长 --, 单位 --, 是否必选

#### 编解码插件开发

**步骤1** 在烟感产品的开发空间，选择“编解码插件开发”。

**步骤2 配置数据上报消息，上报火灾等级和温度。**

添加messageId字段，表示消息种类。

- 在本场景中，数据上报消息有两种，所以需要用messageId来标志消息种类。
- “数据类型”根据数据上报消息种类的数量进行配置。在本场景中，仅有两种数据上报消息，配置为“int8u”即可满足需求。
- “默认值”可以修改，但必须为十六进制格式，且数据上报消息的对应字段必须和默认值保持一致。在本场景中，用0x0标识上报火灾等级和温度的消息。

## 添加字段

X

 标记为地址域 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

## 描述

## 数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x0

偏移值 [?](#)

0-1

完成

取消

添加level字段，表示火灾级别。

- “字段名”只能输入包含字母、数字、\_和\$，且不能以数字开头的字符。
- “数据类型”根据设备上报数据的实际情况进行配置，需要和Profile相应字段的定义相匹配。

- “长度”和“偏移值”根据“数据类型”的配置自动填充。

**添加字段** X

标记为地址域 (?)

\* 名字

描述

数据类型

\* 长度 (?)

默认值 (?)

偏移值 (?)

完成 取消

添加temperature字段，表示温度。在Profile中，temperature属性最大值1000，因此在插件中定义temperature字段的“数据类型”为“int16u”，以满足temperature属性的取值范围。

添加字段

标记为地址域 [?](#)

\* 名字

temperature

描述

数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

偏移值 [?](#)

2-4

[完成](#) [取消](#)

**步骤3** 配置命令下发消息。

新增消息

基本信息

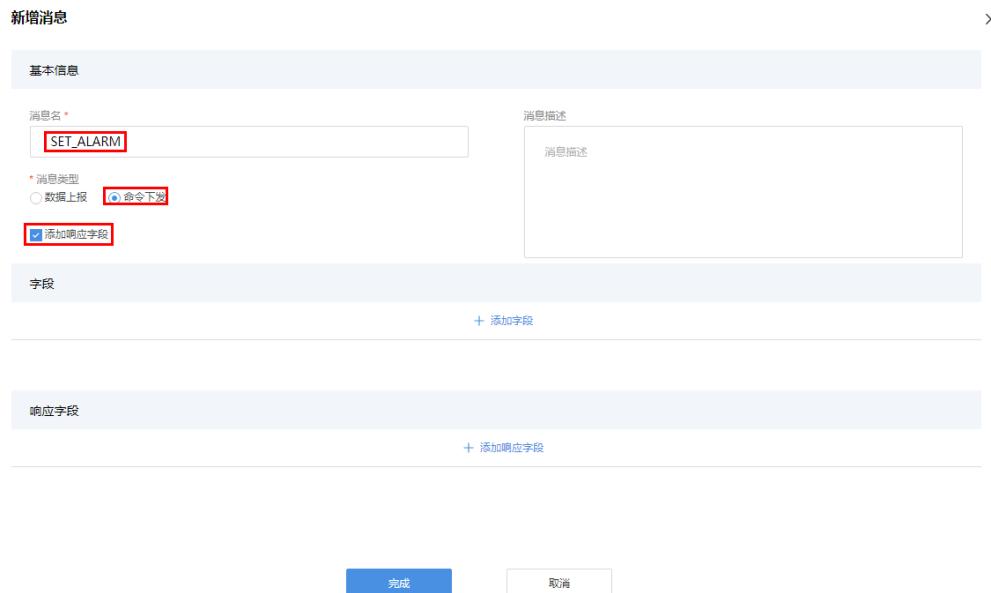
消息名： 消息描述

\*消息类型  命令下发  数据上报  添加响应字段

字段 [+ 添加字段](#)

响应字段 [+ 添加响应字段](#)

[完成](#) [取消](#)



添加messageId字段，表示消息种类。如果只有一种命令下发消息，则可以不配置此字段。

## 添加字段

X

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

\* 名字 当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。

messageId

### 描述

描述

### 数据类型

int8u(8位无符号整型)

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x1

偏移值 [?](#)

0-1

完成

取消

添加mid字段，用于将下发的命令和命令执行结果进行关联。

## 添加字段

X

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

\* 名字 当标记为响应标识字段时，名字固定为mid；否则，名字不能设置为mid。

mid

### 描述

描述

### 数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

默认值

偏移值 [?](#)

1-3

完成

取消

添加value字段，表示下发命令的参数值。

### 添加字段

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

\* 名字

描述

数据类型

int8u(8位无符号整型)

\* 长度 [?](#)

默认值 [?](#)

偏移值 [?](#)

[完成](#)[取消](#)

#### 步骤4 配置命令下发响应消息。

添加messageId，表示消息种类。命令执行结果为上行消息，需要通过messageId和数据上报消息进行区分。

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

标记为命令执行状态字段 [?](#)

\* 名字 **当标记为地址域时，名字固定为messageId；否则，名字不能设置为messageId。**

messageId

描述

描述

数据类型

int8u(8位无符号整型)

\* 长度 [?](#)

1

\* 默认值 [?](#)

0x2

偏移值 [?](#)

0-1

完成

取消

添加mid字段，用于将下发的命令和命令执行结果进行关联。

## 添加字段

X

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

标记为命令执行状态字段 [?](#)

\* 名字 当标记为响应标识字段时，名字固定为mid；否则，名字不能设置为mid。

mid

### 描述

描述

### 数据类型

int16u(16位无符号整型)

\* 长度 [?](#)

2

默认值 [?](#)

默认值

偏移值 [?](#)

1-3

完成

取消

添加errcode字段，用于表示命令执行状态：00表示成功，01表示失败，如果未携带该字段，则默认命令执行成功。

## 添加字段

X

- 标记为地址域 [?](#)
- 标记为响应标识字段 [?](#)
- 标记为命令执行状态字段 [?](#)

\* 名字 当标记为命令执行状态字段时，名字固定为errcode；否则，名字不能设置为errcode。

errcode

### 描述

### 数据类型

int8u(8位无符号整型)

▼

\* 长度 [?](#)

1

默认值 [?](#)

偏移值 [?](#)

3-4

完成

取消

添加result字段，用于表示命令执行结果。

### 添加字段

标记为地址域 [?](#)

标记为响应标识字段 [?](#)

标记为命令执行状态字段 [?](#)

\* 名字

描述

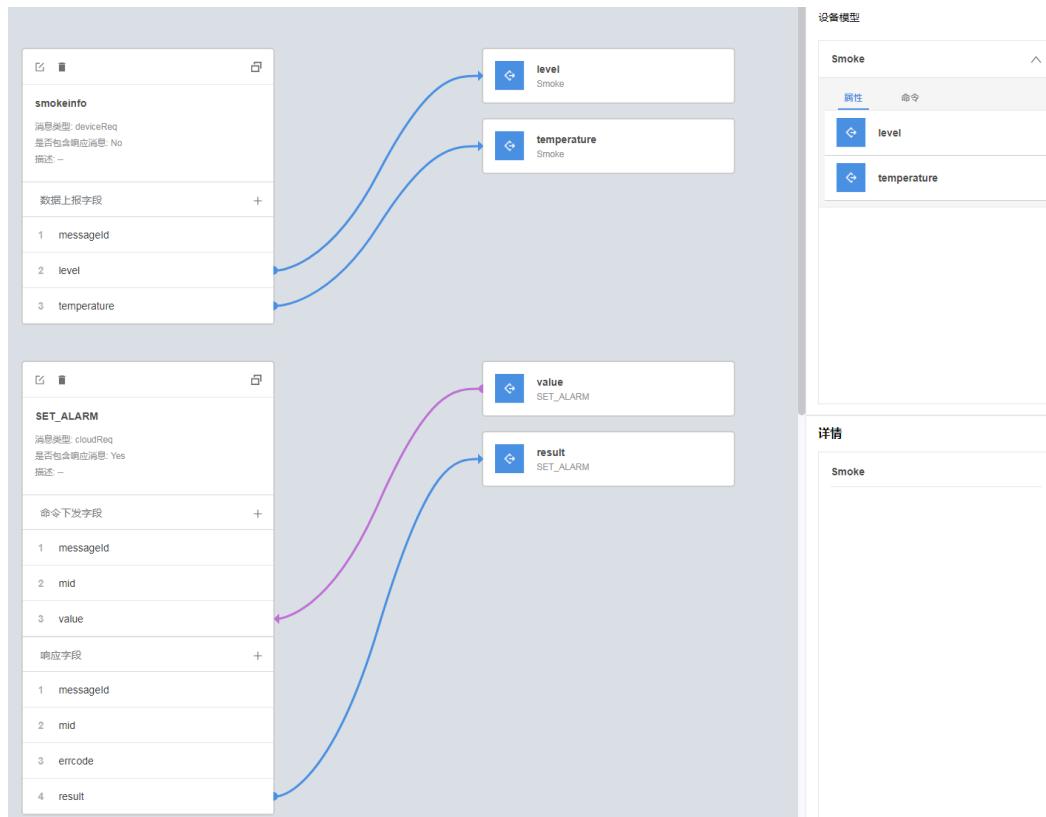
数据类型

\* 长度 [?](#)

默认值 [?](#)

偏移值 [?](#)

**步骤5** 拖动右侧“设备模型”区域的属性字段和命令字段，数据上报消息和命令下发消息的相应字段建立映射关系。



**步骤6** 点击“保存”，并在插件保存成功后点击“部署”，将编解码插件部署到物联网平台。



----结束

## 调测编解码插件

**步骤1** 在烟感产品的开发空间，选择“在线调测”，使用虚拟设备调试编解码插件。



勾选“没有真实的物理设备”，点击“创建”。

**新增测试设备**

您现在  
 有真实的物理设备  没有真实的物理设备

您正在注册一个虚拟的设备

**创建** **取消**

**步骤2** 使用应用模拟器进行命令下发: { "serviceId": "Smoke", "method": "SET\_ALARM", "paras": "{\"value\":0}" }。

在“设备模拟器”区域查看命令接收的结果: 01000100。01为messageId字段, 0001为mid字段, 00为value字段。

The diagram illustrates the communication flow between three components:

- Application Simulator**: Shows a message being sent from the simulator to the IoT Platform.
- IoT Platform**: Represented by a central box with bidirectional dashed arrows labeled "命令下发" (Command Downstream) and "数据上报" (Data Report).
- Device Simulator**: Shows a message being received by the device simulator from the IoT Platform.

**Application Simulator Details:**

- 命令发送: 2018-09-18 15:56:59
- 发送消息头信息: { "time": "Tue Sep 18 15:56:59 GMT+08:00 2018", "requestId": "05be64da-daaaf-d414-645f-f24554ba4cb7\_9927", "callbackUrl": null, "expireTime": 0, "command": { "serviceId": "Smoke", "method": "SET\_ALARM", "paras": "{\"value\":0}" }}
- 发送消息body信息: { "serviceId": "Smoke", "method": "SET\_ALARM", "paras": "{\"value\":0}" }

**Device Simulator Details:**

- 命令接收: 2018-09-18 15:56:59
- 01000100

**步骤3** 使用设备模拟器进行数据上报。

十六进制码流示例：0200010000。02表示messageId，此消息上报命令执行结果；0001表示mid，长度为2个字节；00表示命令执行状态，长度为1个字节；00表示命令执行结果，长度为1个字节。

在“设备详情 > 历史命令”查看命令执行状态：执行成功。



----结束

## 总结

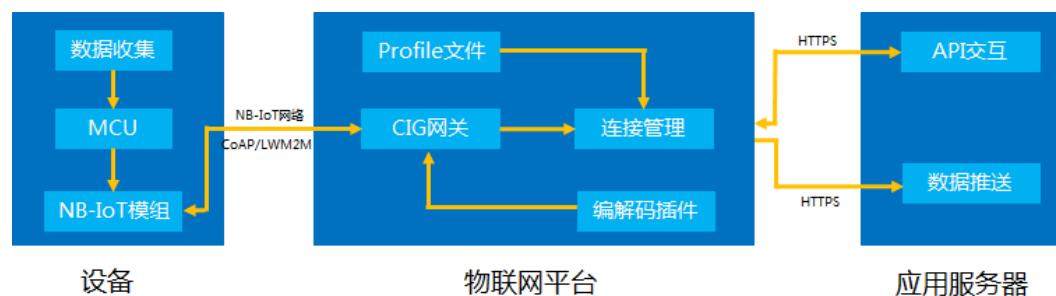
- 如果插件需要对命令执行结果进行解析，则必须在命令和命令响应中定义mid字段。
- 命令下发的mid是2个字节，对于每个设备来说，mid从1递增到65535，对应码流为0001到FFFF。
- 设备执行完命令，命令执行结果上报中的mid要与收到命令中的mid保持一致，这样平台才能刷新对应命令的状态。

## 3.2.4 设备集成

在CoAP或LWM2M协议接入场景下，设备可以通过集成NB-IoT模组或者LiteOS SDK实现与物联网平台的对接。

### 集成 NB-IoT 模组

集成NB-IoT模组的设备，可以通过NB-IoT网络接入物联网平台。



特点	<ul style="list-style-type: none"><li>覆盖广，相比LTE提升20dB以上的增益</li><li>低功耗，聚焦小数据量、小速率应用</li><li>海量连接，单扇区支持5万个连接</li><li>低成本，低速率、低功耗、低带宽等特点使NB-IoT芯片或模组具备低成本优势</li></ul>
应用场景	对数据时效性要求低，数据包较小，设备位置变化较小，需要电池供电，例如：智能抄表，智能路灯等。

网络需求	<ul style="list-style-type: none"><li>● NB-IoT网络：由运营商构建。</li><li>● NB-IoT SIM卡：向NB-IoT网络运营商购买。</li><li>● NB-IoT模组：向模组厂商购买。</li></ul>
通信协议	CoAP/LWM2M
相关资源	<ul style="list-style-type: none"><li>● 如果您需要基于NB-IoT模组进行智能终端开发，请参考<a href="#">NB-IoT模组及终端应用指导</a>，该文档描述了开发NB-IoT智能终端时需要注意的一些设计要点。 <b>说明</b> 适用于使用海思hi2110/hi2115芯片模组的终端产品。</li><li>● 如果您需要基于第三方厂商模组更进行智能终端开发，请从模组厂商获取更多信息和支持。</li></ul>

## 集成 LiteOS SDK

LiteOS SDK是用于设备侧集成的轻量化SDK，它的具体特征如下：

特点	<ul style="list-style-type: none"><li>● 屏蔽了协议和安全细节，用户可以专注自身的应用，无需关注协议和安全的具体实现。</li><li>● 提供适配层，用户只需适配少量接口，便可以将LiteOS SDK进行移植。</li><li>● 支持对终端设备上报的数据进行缓存，且具备重传和确认机制，保障数据上报的可靠性。</li><li>● 支持固件升级，并实现了断点续传、固件包完整性保护。</li><li>● 支持安全和非安全两种连接方式。</li></ul>
运行环境	RAM > 32KB FLASH > 128KB
网络需求	NB-IoT、2/3/4G、有线网络等。
通信协议	CoAP、LWM2M
相关资源	如果您选择在智能终端中集成LiteOS SDK，请参考 <a href="#">LiteOS SDK集成开发指导</a> 。

### 3.2.5 设备调测

#### 概述

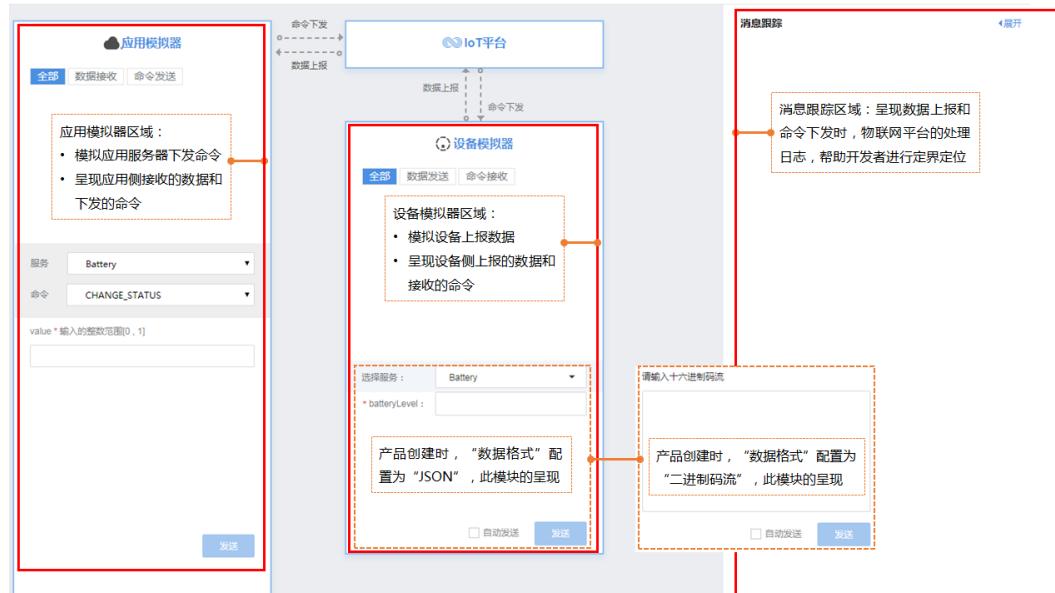
在线调测具备设备模拟和应用模拟功能，可以模拟数据上报和命令下发等场景，对设备、Profile、插件等进行调测。

进行在线调测时，可以使用真实设备调测，也可以使用虚拟设备调测：

- 当设备侧开发已经完成时，但应用侧开发还未完成时，开发者可以创建真实设备，使用应用模拟器对设备、Profile、插件等进行调测。真实设备调测界面结构如下：



- 当设备侧开发和应用侧开发均未完成时，开发者可以创建虚拟设备，使用应用模拟器和设备模拟器对Profile、插件等进行调测。虚拟设备调测界面结构如下：



## 使用真实设备调测

**步骤1** 在产品开发空间，点击“在线调测”。



**步骤2** 在“设备列表”区域，点击“新增测试设备”。



**步骤3** 系统将弹出“新增测试设备”窗口，勾选“有真实的物理设备”，完成各项参数配置后，点击“创建”。

- “设备名称”只允许大小写字母、数字和下划线，需要在产品下保持唯一。
- “设备标识”需要在产品下保持唯一，如设备的IMEI、mac等。
- “验证码加密”根据设备的实际情况进行配置。

**新增测试设备**

您现在  有真实的物理设备  没有真实的物理设备

\*设备名称

\*设备标识

不加密  加密

**创建** **取消**

设备创建成功后，将返回“设备ID”和“PSK码”。如果设备使用DTLS协议接入物联网平台，请妥善保存PSK码。

## 设备创建成功

X

请根据设备指导说明书为设备接通电源，配置好网络，开启设备，观察设备是否成功接入到平台，如果状态是在线（online）表示设备已经成功的接入到平台，接着就可以接收设备的数据。以下是您的设备信息，请牢记！

设备ID

c2c3ffb2-f8f5-48f1-b44e-5943ec64d575

PSK码（使用DTLS协议时需要使用到该psk码，请您牢记！）  
**ceb95d32c2da62b4a2cf9530096b178d**

确定

**步骤4** 在设备列表中，选择新创建的真实设备，进入调试界面。

设备列表							+新增测试设备
状态	设备名称	设备ID	所属产品	产品型号	产品类型	操作	
在线	testdevice001	c2c3ffb2-f8f5-48f1-b44e-5943ec64d575	WaterMeter01	WaterMeter01	WaterMeter		

**步骤5** 将真实设备接入到物联网平台，并进行数据上报，在“应用模拟器”区域查看数据上报的结果，在“消息跟踪”区域查看物联网平台处理日志。

The screenshot shows the IoT Platform interface with two main panels: "Application Simulator" on the left and "Message Trace" on the right.

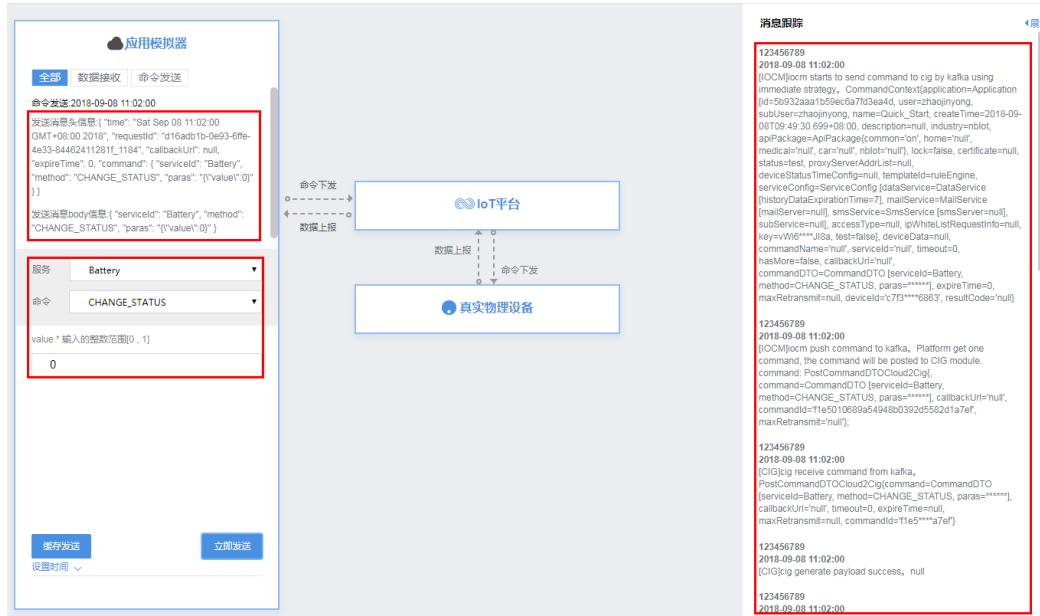
**Application Simulator Panel:**

- Header: 应用模拟器
- Buttons: 全部, 数据接收, 命令发送
- Log: 显示了2018-09-08 10:58:03 [batteryLevel=1] 的数据接收记录。
- Form:
  - 服务: Battery
  - 命令: CHANGE\_STATUS
  - value: 输入框 (显示为batteryLevel=1)
- Buttons: 立即发送, 延迟发送, 设置时间

**Message Trace Panel:**

- Header: 消息跟踪
- Log: 显示了多条处理日志，包括消息接收、命令下发、数据上报等记录，如：[CIG]ig received message from device, uid = 12\*\*\*99, [CIG]ig get plug-in success, [key = e08ac2b7703849c784cb0ae23e0283c5&&WaterMeter01], [CIG]ig ready to response to ue, Ready to encode the response to device, msgType : cloudRsp, input {"id": "123456789", "request": "\*\*\*\*\*", "errcode": 0, "hasMore": "0"}, [CIG]ig send data to kafka success, [DeviceData] reportType=DATA, service= Battery, data= [batteryLevel=1], time=20180908T025803Z, hasMore=false ] 等。

**步骤6** 在“应用模拟器”区域进行命令下发，在“消息跟踪”区域查看物联网平台处理日志，在真实设备上查看接收到的命令。



----结束

## 使用虚拟设备调测

**步骤1** 在产品开发空间，点击“在线调测”。



**步骤2** 在“设备列表”区域，点击“新增测试设备”。



**步骤3** 系统将弹出“新增测试设备”窗口，勾选“没有真实的物理设备”，点击“创建”。

### 新增测试设备

您现在

有真实的物理设备

没有真实的物理设备

您正在注册一个虚拟的设备

**创建**

取消

**步骤4** 在设备列表中，选择新创建的虚拟设备，进入调试界面。虚拟设备名称组成为：“产品名称” + “Simulator”，每款产品下只能创建一个虚拟设备。

状态	设备名称	设备ID	所属产品	产品型号	产品类型	操作
在线	WaterMeter01NB Simulator	d1ec9268-240c-4168-abe4-7bbaf60e583b	WaterMeter01	WaterMeter01	WaterMeter	<span style="color: red;">禁用</span>
离线	testdevice001	c2c3fb2-f8f5-48f1-b44e-5943ec64d575	WaterMeter01	WaterMeter01	WaterMeter	<span style="color: red;">禁用</span>

**步骤5** 在“设备模拟器”区域，输入十六进制码流或者JSON数据（以十六进制码流为例），点击“发送”，在“应用模拟器”区域查看数据上报的结果，在“消息跟踪”区域查看物联网平台处理日志。

**应用模拟器**

全部 数据接收 命令发送

数据接收  
2018-09-08 11:30:21  
(batteryLevel=1)

服务: Battery  
命令: CHANGE\_STATUS  
value \* 输入的整数范围[0, 1]

0001

自动发送 发送

**IoT平台**

命令下发  
数据上报  
命令下发

**设备模拟器**

全部 数据发送 命令接收

数据发送  
2018-09-08 11:30:21  
0001  
命令接收  
2018-09-08 11:30:21  
AAAA0000

请输入十六进制码流  
0001

自动发送 发送

**消息跟踪**

```

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg received message from device, ueld = zhaod***2910

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg plug-in success, { key
=e08ac2b77033849c784cb0ae23e0283c5&&WaterMeter01 }

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg sends data success, DecodeReportDTO [identifier
=null, msgType=DEVICEREQ, deviceInfo=*****,
hasMore=false, data=null, dataList=DecodeDataDTO[]]

[DecodeDataDTO [serviceId =Battery,
serviceData=*****], eventTime=null, ]]

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg encode ack message success! encodeResult = [-86,
-86, 0, 0]

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg is ready to response to ue, Ready to encode the
response to device, msgType : cloudRsp, input: [{"identifier":null,
"request":*****,"errcode":0,"hasMore":0}]

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg send data to kafka success, [DeviceData, reportType=
DATA, service=Battery, data=[batteryLevel=1], time=
20180908T033021Z, hasMore=false]

zhaojinyong1536377412910
2018-09-08 11:30:21
[CG]cg send ack message success, Response have been
posted to device, response: [-86, -86, 0, 0]

zhaojinyong1536377412910
2018-09-08 11:30:21
[OCM]ocm receives data from cig by kafka,
UpdateDeviceDataDTO[GW2Cloud [header=Header
[requestId=null, method=POST, timestamp=null, hasMore =
false], body=Body [services=DeviceServiceA [serviceId=Battery,
eventTime=20180908T033021Z]]], templated = ruleEngine
]

```

**步骤6** 在“应用模拟器”区域进行命令下发，在“设备模拟器”区域查看接收到的命令（以十六进制码流为例），在“消息跟踪”区域查看物联网平台处理日志。

**应用模拟器**

全部 数据接收 命令发送

命令发送  
2018-09-08 11:32:43  
{"time": "Sat Sep 08 11:32:43  
GMT+08 20 2010", "request": "tcb5f18b-0d58-cb63-  
e7-e19e5baed632\_4376", "callbackUrl": null,  
"expireTime": 0, "command": {"serviceId": "Battery",  
"method": "CHANGE\_STATUS", "paras": [{"value": "-1"}]}  
}

服务: Battery  
命令: CHANGE\_STATUS  
value \* 输入的整数范围[0, 1]  
1

自动发送 发送

**IoT平台**

命令下发  
数据上报  
命令下发

**设备模拟器**

全部 数据发送 命令接收

数据发送  
2018-09-08 11:30:21  
0001  
命令接收  
2018-09-08 11:32:43  
01000201  
命令接收  
2018-09-08 11:32:43  
01000401  
命令接收  
2018-09-08 11:30:21  
AAAA0000

请输入十六进制码流  
请输入十六进制码流

自动发送 发送

**消息跟踪**

```

zhaojinyong1536377412910
2018-09-08 11:33:39
[OCM]ocm starts to send command to cig by kafka using
immediate strategy., CommandContext[application=Application
[id=5b932aa1b59e6a7703ea4d, user=zhaojinyong,
subUser=zhaojinyong, name=Quick_Start, createTime=2018-09-
08 11:33:39, identifier=*****], service=DeviceServiceA,
industry=mbot,
appPackage=AppPerIotAppCommon, home=huit,
medical=null, car=null, mbot=null, lock=false, certificate=null,
status=test, proxyServer=Addit, lstrn,
deviceStatusTimeConfig=null, template=lruleEngine,
serviceConfig=ServiceConfig [dataService=DataService
(historyDataExpirationTime=7], mailService=MailService
[mailServer=null], smsService=SmsService [smsServer=null],
subService=null, accessType=OPEN, gwWhitelstRequestCount=null,
replicaCount=1, job=Job [jobName=deviceDataJob, data=null,
commandName=null, serviceId=null], timeout=0,
hasMore=false, callbackUrl=null], commandDTO=CommandDTO [serviceId=Battery,
method=CHANGE_STATUS, paras="*****"], callbackUrl=null,
commandId=9696859009b34cf49a6f354ae3f2dd5a",
maxRetransmit=null, deviceld=ed1*****65a0, resultCode=null]

zhaojinyong1536377412910
2018-09-08 11:33:39
[CG]cg receive command to kafka, Platform get one
command, the command will be posted to CG module.
command: PostCommandDTO[cloud2Cig,[command=CommandDTO
[serviceId=Battery, method=CHANGE_STATUS, paras="*****],
callbackUrl=null], timeout=0, expireTime=null,
maxRetransmit=null, commandId=9696859009b34cf49a6f354ae3f2dd5a]

zhaojinyong1536377412910
2018-09-08 11:33:39
[CG]cg handle the command results, The platform handles
the command results, the
commandId=9696859009b34cf49a6f354ae3f2dd5a, the mid=2,
the deviceld=ed160f5-1cf4-e3b-b993-b2cd7565a0, the

```

----结束

## 3.2.6 认证测试

### 3.2.6.1 认证测试指导

#### 概述

自助测试具备预检查和自助测试两个功能：

- 预检查：自动完成厂商信息、产品信息、Profile、插件的完整性和合法性检查。预检查结果有三种状态：
  - **预检查通过：**所有信息均通过预检查，可以进入下一阶段。
  - **预检查通过，部分信息缺失，但不影响自助测试：**部分信息不完整，但不影响自助测试，可以进入下一阶段。
  - **重要信息缺失，请完善：**存在重要信息缺失，预检查不通过，不能进入下一阶段。
- 自助测试：提供端到端测试用例，帮助开发者自助完成产品功能测试。测试完成后，开发中心将生成测试报告，用于进行产品发布认证。

#### 预检查和自助测试

**步骤1** 在产品开发空间，点击“发起自助测试”。

**步骤2** 系统进入“预检查”界面，点击“开始检查”。



**步骤3** 系统返回预检查结果：如果预检查通过，则继续进行下一阶段测试；如果预检查未通过，则根据提示信息进行修改，并重新进行预检查。





**步骤4** 预检查通过后，点击“自助测试”。开发者需要使用真实设备进行自助测试。



**步骤5** 根据测试用例说明，依次完成**终端开机入网测试>数据上报测试>数据上报属性测试>控制命令下发测试>命令下发响应测试**。所有测试用例全部执行成功，产品开发才全部完成。

#### 说明

当产品的“数据格式”为“二进制码流”时，才需要进行命令下发响应测试；否则，开发者不需要执行此测试用例。



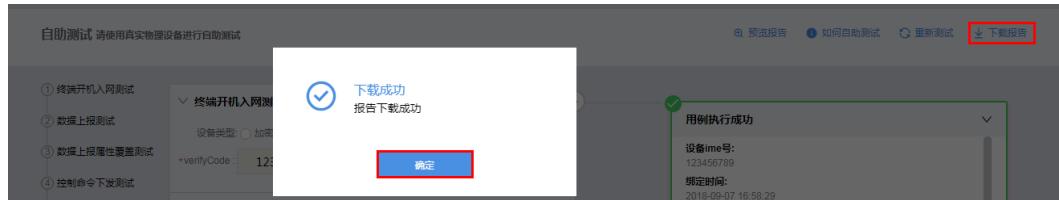
----结束

## 预览和下载测试报告

**步骤1** 在通过自助测试后，点击“预览报告”可以对测试报告进行预览。



**步骤2** 点击“下载报告”，将测试报告下载到本地。



----结束

### 3.2.6.2 认证测试用例

#### 3.2.6.2.1 终端开机入网测试

**步骤1** 选择“设备类型”，并填写“VerifyCode”。点击右方启动按钮开始测试。

如果设备类型选择“加密”，还需要填写“PSK”。



**步骤2** 使用真实设备进行绑定上线操作，点击右方停止按钮查看测试用例执行结果。



----结束

### 3.2.6.2.2 数据上报测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 使用真实设备上报Profile中定义的任意一个属性，点击右方停止按钮查看测试用例执行结果。



----结束

### 3.2.6.2.3 数据上报属性测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 使用真实设备上报Profile中定义的所有属性，点击右方停止按钮查看测试用例执行结果。





----结束

#### 3.2.6.2.4 控制命令下发测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 物联网平台会根据Profile的定义，向设备下发一条命令，命令取值随机。在真实设备侧，查看接收到的命令取值，并在测试窗口填写后，点击右方停止按钮查看测试用例执行结果。

当产品的“数据格式”为“二进制码流”时，则在测试窗口需要填写二进制或十六进制内容；当产品的“数据格式”为“JSON”时，则在测试窗口需要填写JSON格式的内容。此处以二进制或十六进制内容为例。



----结束

### 3.2.6.2.5 命令下发响应测试

当产品的“数据格式”为“二进制码流”时，才需要进行命令下发响应测试；否则，开发者不需要执行此测试用例。如果该款产品的设备不返回命令执行结果，则开发者也不需要执行此测试用例。

**步骤1** 将真实设备设置为收到命令后返回命令执行结果，点击右方启动按钮开始测试。

如果真实设备无法自动返回命令执行结果，则直接启动测试，并在**步骤2**中手动使真实设备上报命令执行结果。



**步骤2** 如果真实设备支持自动返回命令执行结果，则在真实设备收到命令后，点击右方停止按钮查看测试用例执行结果；如果真实设备不支持自动返回命令执行结果，则根据真实设备收到的命令，手动使真实设备上报命令执行结果后，点击右方停止按钮查看测试用例执行结果。



----结束

## 3.2.7 产品发布

### 概述

如果开发中心已经对接运营中心，则开发者可以向运营中心申请发布通过认证测试的产品：可以申请在产品中心公开展示，也可以申请仅自己可见。

### 申请发布产品

**步骤1** 产品在通过自助测试后，点击“申请发布”。



**步骤2** 选择产品发布模式：“产品中心展示”或“仅自己可见”，点击确定。



----结束

## 3.2.8 上线商用平台

### 概述

商用平台对接产品中心后，可以直接导入在产品中心发布的产品资源（含Profile、编解码插件等）。

### 导入产品

**步骤1** 购买IoT平台增强版服务后，在IoT平台控制台进入管理门户。购买方式请参考[如何订购IoT平台增强版服务](#)。

IoT平台增强版

再次购买 进入开发者平台

总览

IoT平台增强版

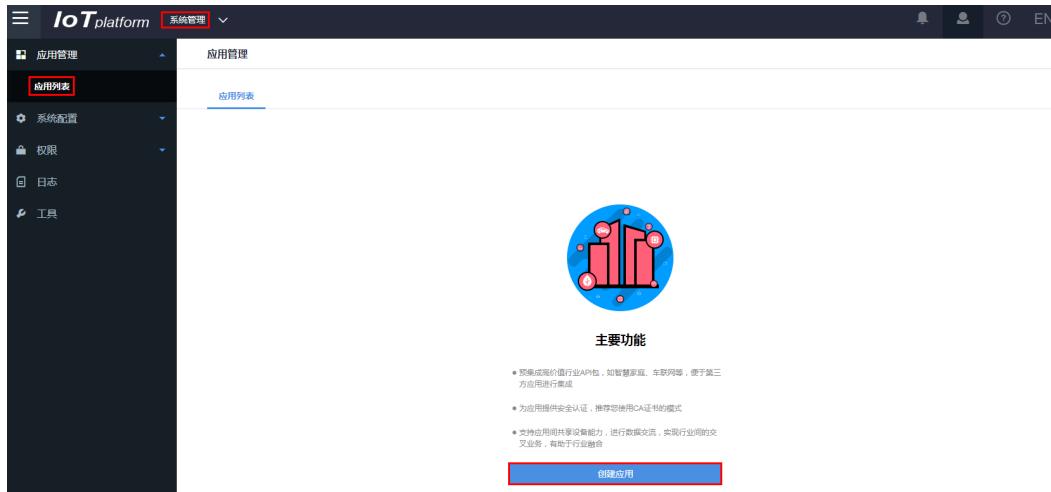
•您已购买1个IoT平台增强版设备管理基本包，对应的1个IoT平台账号如下表所示。  
•您可进入管理门户管理您的设备和应用，详情请查看[用户指南](#)。如需访问进入开发者平台，详情请参考[开发者指南](#)。

IoT平台账号	创建时间	到期时间	基本包状态	按需资源状态	计费模式	操作
[REDACTED]	2018-06-29 15:46:18	2019-06-29 23:59:59	运行中	运行中	包月/包年	<a href="#">进入管理门户</a> 更多

**步骤2** 在管理门户选择“系统管理 > 应用管理 > 应用列表”，点击“创建应用”。



如果相应的应用已经存在，则可以跳过此步骤。



在“创建应用”界面，完成各项参数的配置后，点击“确定”。

The screenshot shows the 'Create Application' configuration page. It includes three main sections:

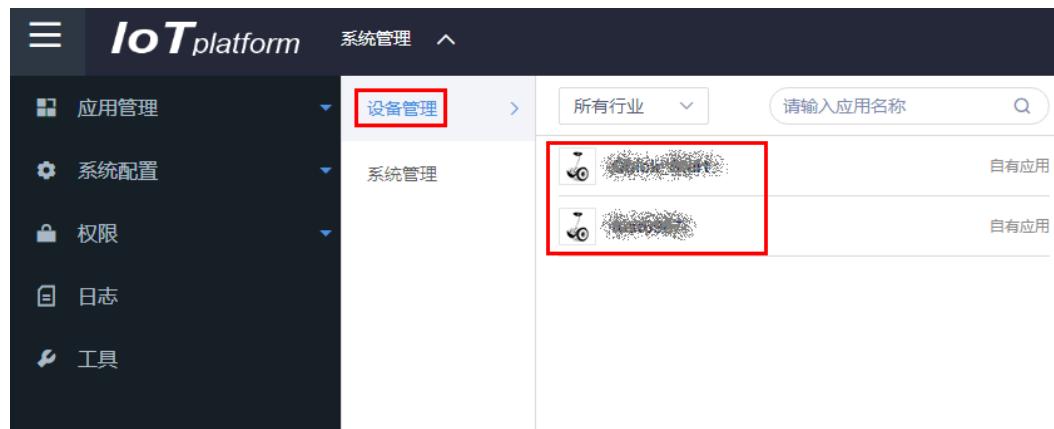
- 基本信息**: Fields for '应用名称' (myFirstApp) and '所属行业' (智慧家庭行业). A note below says: '关闭授权，如果出现故障时，可能没有足够的信息，将会降低问题定位效率，延长了问题解决时间，建议您授权给平台权限，如果有设备故障时，平台运维会更好的协作您一起解决问题。' (Close authorization, if there is a failure, there may not be enough information, which will reduce fault location efficiency and extend problem resolution time. It is recommended to grant platform permissions, so that when there is a device failure, platform operations and maintenance can better collaborate with you to solve the problem.)
- 消息推送**: '选择协议' (Protocol Selection) section with '推送协议' (Push Protocol) set to 'HTTPS'. A note says: '此方式平台需要对接入的应用服务器采用CA证书进行认证，认证通过后才允许接入，以保证数据的安全' (This way, the platform needs to authenticate the application server using a CA certificate. Once authenticated, access is allowed to ensure data security).
- 平台能力**: A note: '您可以选择不同的平台服务套餐，降低应用开发难度。平台能力有默认的基础配置，如需更丰富的功能，请在“应用信息 > 平台能力”进行配置' (You can choose different platform service packages to reduce application development difficulty. Platform capabilities have default basic configurations. If you need more features, please configure them in "Application Information > Platform Capabilities"). It includes checkboxes for '设备数据处理' (Device data processing) and '平台默认支持传输设备数据和存储设备数据，默认存储时间为7天，如果您仅需要平台传输数据的能力，可关闭存储历史数据的开关' (Platform default supports transmitting device data and storing device data, default storage time is 7 days. If you only need the platform's data transmission capability, you can turn off the switch for historical data storage).

At the bottom right are buttons: '我已阅读并同意《个人数据使用条款》' (I have read and agree to the 'Personal Data Usage Terms'), '确定' (Confirm), and '取消' (Cancel).

系统将弹出应用创建成功窗口，包含应用ID、应用秘钥等信息。密钥信息在应用详细页内不可见，请妥善保管。



**步骤3** 选择“设备管理 > 具体应用”，进入该应用的“设备管理”界面。



**步骤4** 在“产品模型”界面，点击“新建产品模型”，并选择“从产品中心导入”。



**步骤5** 在“产品中心”界面，选择需要导入的产品。



**步骤6** 在“产品详情”界面，点击“导入该产品”。

产品名称		型号	产品ID
水表	Water Meter	水表	5c1e6035538b34231e1aa68b
厂商名称		厂商ID	协议类型
华为		cc79868d27fb4f8b85095b2375916...	CoAP
Bundle名称		标签	所属行业
WaterMeter-cc79868d27fb4f8b850...		--	智慧生活

----结束

## 3.3 MQTT

### 3.3.1 产品创建

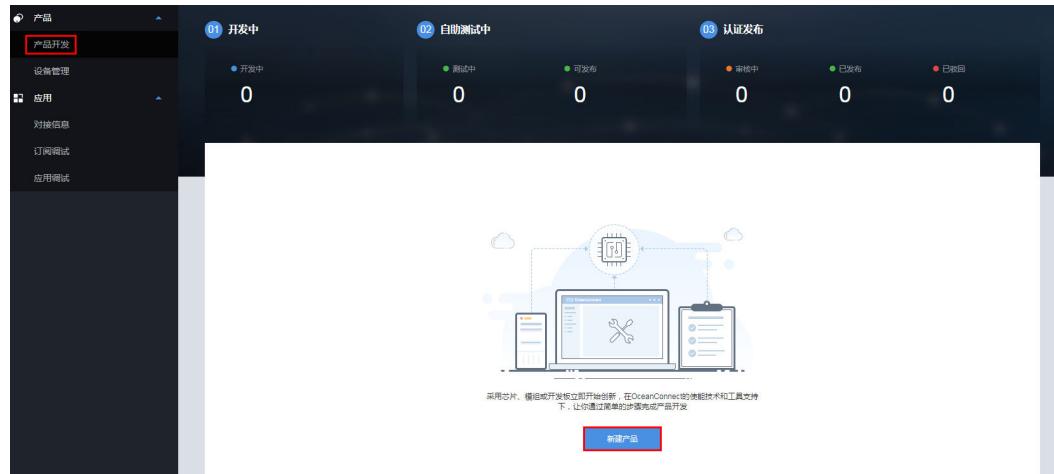
#### 概述

某一类具有相同能力或特征的设备的集合称为一款产品。除了设备实体，产品还包含该类设备在物联网能力建设中产生的产品信息、产品模型（Profile）、插件、测试报告等资源。

#### 自定义新建产品

自定义新建产品指不使用系统预置的产品模板，全新定义一款产品。

**步骤1** 在项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“自定义产品”界面，点击“自定义产品”。



**步骤3** 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“厂商名称”等信息后，点击“创建”。

#### 说明

- “产品名称”、“型号”需要在项目内保持唯一，否则会创建失败。
- “所属行业”、“设备类型”根据实际情况进行填写。
- “接入应用层协议类型”配置为MQTT，“数据格式”请根据真实设备上报的数据格式进行配置。

### 设置产品信息

×

* 产品名称 :	Bulb
* 型号 :	Bulb001
* 厂商ID :	e08ac2b7703849c784cb0ae23e0283c5
* 厂商名称 :	TestManuName
* 所属行业 :	智慧生活
* 设备类型 :	Bulb
* 接入应用层协议类型 :	MQTT
* 数据格式 :	JSON
产品图片 :	<p>请选择图片</p> <p>图片大小为200*200</p>  <p>X</p>
<p>创建 取消</p>	

**步骤4** 在“产品开发”界面将会呈现已经创建的产品，选择具体产品，可以进入该产品的开发空间。

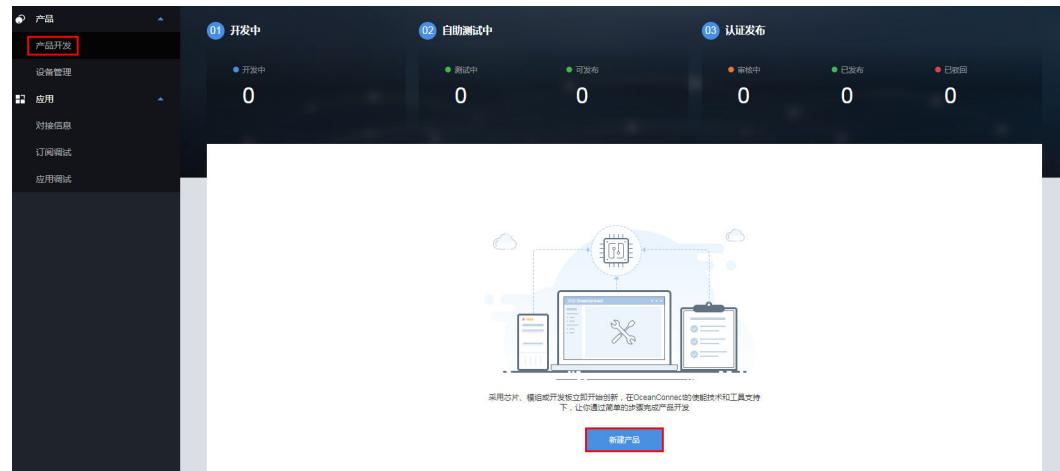


----结束

## 快速新建产品

快速新建产品指使用系统预置的产品模板（或已经创建的产品模板），快速定义一款产品。

**步骤1** 在项目空间内，选择“产品 > 产品开发”，点击“新建产品”。



**步骤2** 在“基于系统模板创建”界面，选择需要的系统模板，点击“立即使用”。

This screenshot shows the 'Based on System Template Creation' page. At the top, it says '请选择以下一种方式创建您的产品' (Please choose one way to create your product). Below that, there are three tabs: '基于系统模板创建' (selected), '基于已有产品创建', and '自定义产品'. A note below the tabs says '系统已为您推荐部分配套模版，助您快捷开发产品' (The system has recommended some accompanying templates to help you develop products quickly). The page displays a grid of product templates, each with an '立即使用' (Use Now) button. The templates include:

- NBIoTDevice GlobalTemplate
- NBIoTDevice GlobalTemplate
- WaterMonitorDevice01 WaterMonitorDevice
- WaterMeter01 WaterMeter
- VehicleDetector01 VehicleDetector
- StreetLight01 StreetLight
- GasMeter01 GasMeter
- AirMonitorDevice01 AirMonitorDevice

**步骤3** 系统将弹出“设置产品信息”窗口，填写“产品名称”、“产品型号”、“厂商名称”等信息后，点击“创建”。

 **说明**

- “产品名称”、“型号”需要在项目内保持唯一，否则会创建失败。
- “所属行业”、“设备类型”根据实际情况进行填写。
- “接入应用层协议类型”配置为MQTT，“数据格式”请根据真实设备上报的数据格式进行配置。

### 设置产品信息

×

* 产品名称 :	Bulb
* 型号 :	Bulb001
* 厂商ID :	e08ac2b7703849c784cb0ae23e0283c5
* 厂商名称 :	TestManuName
* 所属行业 :	智慧生活
* 设备类型 :	Bulb
* 接入应用层协议类型 :	MQTT
* 数据格式 :	JSON
产品图片 :	<p>请选择图片</p> <p>图片大小为200*200</p>  <p>X</p>
<p>创建 取消</p>	

**步骤4** 在“产品开发”界面将会呈现已经创建的产品，选择具体产品，可以进入该产品的开发空间。

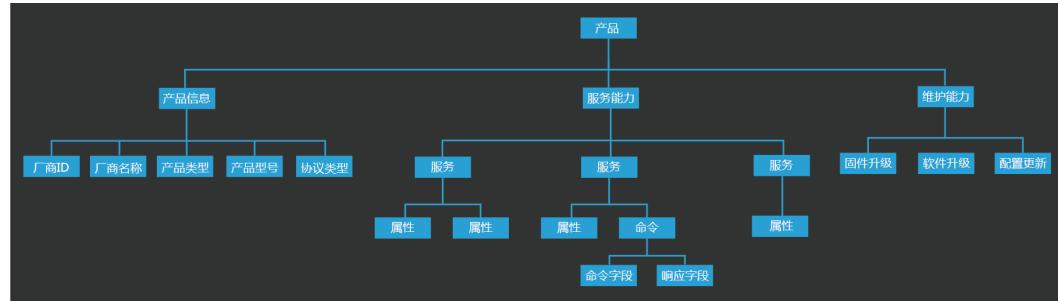


----结束

### 3.3.2 Profile 定义

#### 概述

产品模型（也称Profile）用于描述设备具备的能力和特性。开发者通过定义Profile，在物联网平台构建一款设备的抽象模型，使平台理解该款设备支持的服务、属性、命令等信息。



#### 定义 Profile

在**产品创建**时：如果选择使用系统模板，则系统将会自动使用相应的Profile模板，开发者可以直接使用或在此基础上进行修改；如果选择自定义产品模板，则需要完整定义Profile，操作如下：

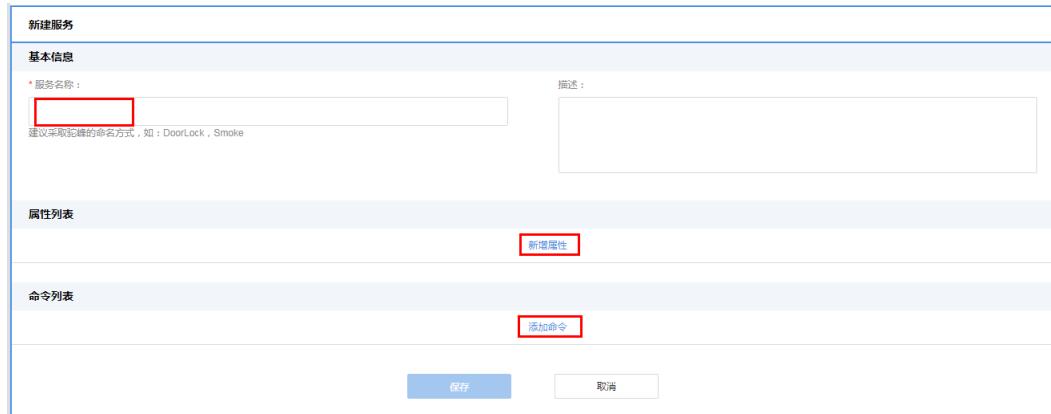
**步骤1** 在“产品开发”界面选择产品，选择具体产品，进入该产品的开发空间。



**步骤2** 在产品开发空间，点击“Profile定义”，然后点击“新建服务”。



**步骤3** 在“新建服务”区域，对服务名称、属性和命令进行定义。每个服务下，可以包含属性和命令，也可以只包含其中之一，请根据此类设备的实际情况进行配置。



1. 填写“服务名称”，“服务名称”采用首字母大写的命名方式，比如：WaterMeter、Battery。



2. 点击“新增属性”，在弹出窗口中配置属性的各项参数，点击“确定”。“属性名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如 batteryLevel、internalTemperature；其余参数，请根据此类设备的实际情况进行配置。

“数据类型”的配置可参考如下原则：

- **int:** 当上报的数据为整数或布尔值时，可以配置为此类型。
- **decimal:** 当上报的数据为小数时，可以配置为此类型。
- **string:** 当上报的数据为字符串、枚举值或布尔值时，可以配置为此类型。如果为枚举值或布尔值，值之间需要用英文逗号（“,”）分隔。
- **DateTime:** 当上报的数据为日期时，可以配置为此类型。
- **jsonObject:** 当上报的数据为json结构体时，可以配置为此类型。

### 新增属性

名称 \*

数据类型 \*

最小 \*

最大 \*

步长

单位

访问模式 \*

R 可以读取属性的值

W 你可以写 (更改) 属性的值

E 当属性值更改时上报事件

是否必选

是

确定 取消

3. 点击“添加命令”，在弹出窗口中配置“命令名称”，点击“确定”。“命令名称”采用所有字母大写，单词间用下划线连接的命名方式，比如DISCOVERY, CHANGE\_STATUS。

## 添加命令

X

命令名称 \*

CHANGE\_STATUS

确定

取消

4. 点击“添加下发字段”，在弹出窗口中配置下发命令字段的各项参数，点击“确定”。“命令字段名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如value；其余参数，请根据此类设备的实际情况进行配置。

## 新增下发命令字段

X

名称 \*

value

数据类型 \*

int

最小 \*

0

最大 \*

1

步长

1

单位

是否必选

 是

确定

取消

5. 点击“添加响应字段”，在弹出窗口中配置响应命令字段的各项参数，点击“确定”。“响应字段名称”采用第一个单词首字母小写，其余单词的首字母大写的命名方式，比如result；其余参数，请根据此类设备的实际情况进行配置。  
响应字段非必配参数，当需要设备返回命令执行结果时，才需要定义此字段。

## 新增响应命令字段

X

名称 \*

result

数据类型 \*

int

最小 \*

0

最大 \*

1

步长

1

单位

是否必选

 是

确定

取消

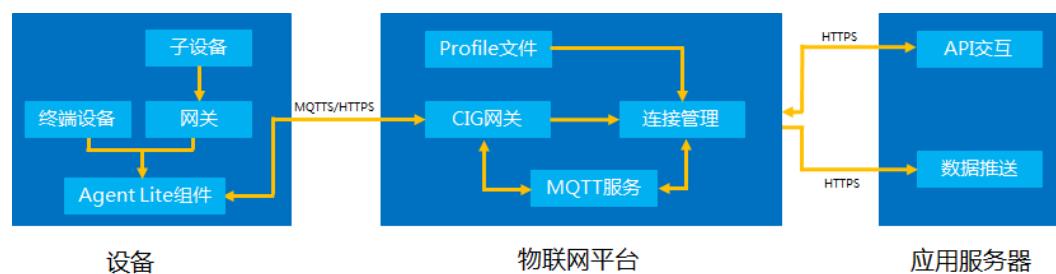
----结束

### 3.3.3 设备集成

在MQTT协议接入场景下，设备可以通过集成Agent Lite SDK或调用MQTT原生接口实现与物联网平台的对接。

#### 集成 Agent Lite SDK

Agent Lite SDK是用于设备侧集成的轻量化SDK，设备集成Agent Lite SDK后，可以通过调用SDK的接口实现与物联网平台的对接。



特点	<ul style="list-style-type: none"><li>● 网络接入方式不敏感，Wi-Fi、2/3/4G、有线网络等多种方式。</li><li>● 轻量化SDK，设备集成后，调用API即可完成与物联网平台的对接。</li><li>● 支持频繁、大数据量通讯，使用Json格式数据。</li></ul>
运行环境	RAM > 4M FLASH > 600KB 目前支持的平台： ARM Linux (Embedded Linux) MIPS Linux (Embedded Linux) x86 Linux x86_64 Linux x86 Windows x86_64 Windows Android (Java)
网络需求	2/3/4G、有线网络等。
通信协议	HTTPS、MQTT/MQTTTS
相关资源	基于Agent Lite SDK的设备接入方案分为直连和非直连两种场景： <ul style="list-style-type: none"><li>● 直连场景：终端设备集成Agent Lite SDK，直接接入到物联网平台。当终端设备具备IP能力，且满足Agent Lite SDK的运行环境时，适用于该场景。 直连场景下的Agent Lite SDK集成开发请参考<a href="#">直连场景</a>。</li><li>● 非直连场景：网关产品集成Agent Lite SDK，终端设备作为子设备连接到网关产品，并通过网关产品接入到物联网平台。当终端设备不具备IP能力，只支持近场通信（如Z-Wave、Zigbee）时，适用于该场景。 非直连场景下Agent Lite SDK集成开发请参考<a href="#">非直连场景</a>。</li></ul>

## 调用 MQTT 原生接口

即将上线，敬请期待。

### 3.3.4 设备调测

#### 概述

在线调测具备设备模拟和应用模拟功能，可以模拟数据上报和命令下发等场景，对设备、Profile、插件等进行调测。

进行在线调测时，可以使用真实设备调测，也可以使用虚拟设备调测：

- 当设备侧开发已经完成时，但应用侧开发还未完成时，开发者可以创建真实设备，使用应用模拟器对设备、Profile、插件等进行调测。真实设备调测界面结构如下：



- 当设备侧开发和应用侧开发均未完成时，开发者可以创建虚拟设备，使用应用模拟器和设备模拟器对Profile、插件等进行调测。虚拟设备调测界面结构如下：



## 使用真实设备调测

**步骤1** 在产品开发空间，点击“在线调测”。

This screenshot shows the product development space interface for online debugging:

- Header:** 产品开发 > Bulb. Includes links for "查看产品详情" (View product details) and "如何开发产品" (How to develop products).
- Product Information:** 型号: Bulb001, 所属行业: 智慧城市, 设备类型: Bulb, 数据格式: JSON, 厂商ID: 7c6908878a714ec5a54a31938839f14d, 接入协议: MQTT.
- Navigation:** 01 Profile定义, 02 SDK集成, 03 在线调测 (highlighted in red), 启发自助测试.
- Device List:** A table showing the device list with columns: 状态 (Status), 设备名称 (Device Name), 设备ID (Device ID), 所属产品 (Product), 型号 (Model), 设备类型 (Device Type), 操作 (Operations). It displays the message: "暂无任何设备" (No devices found).

**步骤2** 在“设备列表”区域，点击“新增测试设备”。



**步骤3** 系统将弹出“新增测试设备”窗口，勾选“有真实的物理设备”，完成各项参数配置后，点击“创建”。

- “设备名称”只允许大小写字母、数字和下划线，需要在产品下保持唯一。
- “设备标识”需要在产品下保持唯一，如设备的IMEI、mac等。
- “验证码加密”根据设备的实际情况进行配置。

**新增测试设备** ×

您现在

有真实的物理设备  没有真实的物理设备

\*设备名称  
testdevice001

\*设备标识

不加密  加密

创建 取消

设备创建成功后，将返回“设备ID”和“PSK码”。如果设备使用DTLS协议接入物联网平台，请妥善保存PSK码。

## 设备创建成功

X

请根据设备指导说明书为设备接通电源，配置好网络，开启设备，观察设备是否成功接入到平台，如果状态是在线（online）表示设备已经成功的接入到平台，接着就可以接收设备的数据。以下是您的设备信息，请牢记！

设备ID

c2c3ffb2-f8f5-48f1-b44e-5943ec64d575

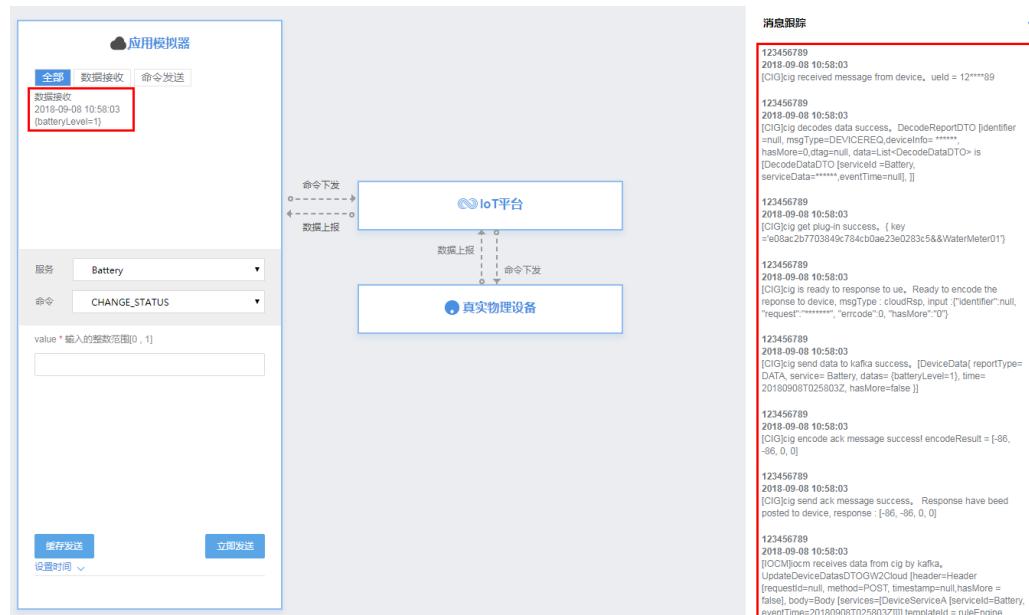
PSK码（使用DTLS协议时需要使用到该psk码，请您牢记！）  
**ceb95d32c2da62b4a2cf9530096b178d**

确定

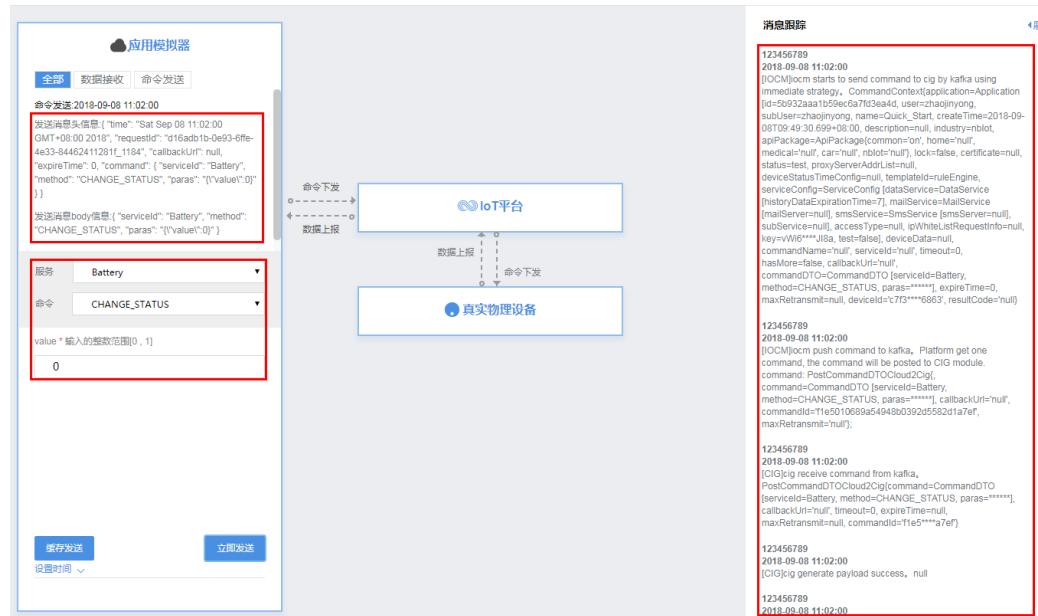
**步骤4** 在设备列表中，选择新创建的真实设备，进入调试界面。

设备列表						+新增测试设备
状态	设备名称	设备ID	所属产品	型号	设备类型	操作
●离线	testdevice001	09e11818-976c-4254-88d2-29a9e28ac4f8	Bulb	Bulb001	Bulb	<a href="#">调试</a> <a href="#">删除</a>

**步骤5** 将真实设备接入到物联网平台，并进行数据上报，在“应用模拟器”区域查看数据上报的结果，在“消息跟踪”区域查看物联网平台处理日志。



**步骤6** 在“应用模拟器”区域进行命令下发，在“消息跟踪”区域查看物联网平台处理日志，在真实设备上查看接收到的命令。



----结束

## 使用虚拟设备调测

**步骤1** 在产品开发空间，点击“在线调测”。

This screenshot shows the product development space for a 'Bulb' device. At the top, it displays basic device information: 型号: Bulb001, 所属行业: 智慧城市, 设备类型: Bulb, 数据格式: JSON, 接入协议: MQTT. Below this, there are three tabs: 01 Profile定义, 02 SDK集成, and 03 在线调试 (highlighted with a red box). Under the 03 tab, there's a section titled '设备列表' (Device List) with a table showing no devices. A red box highlights the '+新增测试设备' (Add Test Device) button at the top right of this section.

**步骤2** 在“设备列表”区域，点击“新增测试设备”。

This screenshot shows the 'Device List' page. It has a table for '设备列表' (Device List) with columns: 状态, 设备名称, 设备ID, 所属产品, 型号, 设备类型, 操作. A red box highlights the '+新增测试设备' (Add Test Device) button at the top right of the table area.

**步骤3** 系统将弹出“新增测试设备”窗口，勾选“没有真实的物理设备”，点击“创建”。



**步骤4** 在设备列表中，选择新创建的虚拟设备，进入调试界面。虚拟设备名称组成为：“产品名称” + “Simulator”，每款产品下只能创建一个虚拟设备。

设备列表							+新增测试设备
状态	设备名称	设备ID	所属产品	型号	设备类型	操作	
在线	Bulb001NoNB Simulator	38330a42-f480-4ef2-891a-43ef7f934e1a4	Bulb	Bulb001	Bulb	调试	删除
离线	testdevice001	08e11818-976c-4254-88d2-29a9e28ac4fb	Bulb	Bulb001	Bulb	调试	删除

**步骤5** 在“设备模拟器”区域，输入JSON数据，点击“发送”，在“应用模拟器”区域查看数据上报的结果，在“消息跟踪”区域查看物联网平台处理日志。

The screenshot displays the IoT Platform interface with three main components:

- 应用模拟器 (Application Simulator):** Shows a JSON message being sent: `{"batteryLevel": "2"}`. The message is shown in the "数据接收" (Data Received) section.
- 设备模拟器 (Device Simulator):** Shows the message being received. The "服务" (Service) is set to "Battery" and the "命令" (Command) is "CHANGE\_STATUS". The value is set to "2".
- 消息跟踪 (Message Trace):** Shows the log entries for the message exchange:
  - 2019-01-03 20:46:11 [CIG]Datareport cig received message from device. null
  - 2019-01-03 20:46:11 [CIG]Datareport cig received message from device. null
  - 2019-01-03 20:46:11 [CIG]Devicebind cig receive bind msg from ue. ueId = zhao\*\*\*\*1983
  - 2019-01-03 20:46:11 [CIG]Devicebind cig send bindmsg to iocm. ueId = zhao\*\*\*\*1983, port = 5684
  - 2019-01-03 20:46:11 [iocm]Devicebind iocm receive bindmsg from cig. BindDeviceByNodeIdTOGWCloud [nodeId=zhao\*\*\*\*1983, isSecure=true, requestId=null]
  - 2019-01-03 20:46:12 [cmnd]Sendcmd iocm send cmd to cig triggered. triggered by kafka topic "IOTCM.DEVICE.V1.registerRoute"
  - 2019-01-03 20:46:12 [cmnd]Sendcmd iocm no cache cmd in queue. processed in kafka handler "IOTCM.DEVICE.V1.registerRoute"

**步骤6** 在“应用模拟器”区域进行命令下发，在“设备模拟器”区域查看接收到的命令，在“消息跟踪”区域查看物联网平台处理日志。



----结束

### 3.3.5 认证测试

#### 3.3.5.1 认证测试指导

##### 概述

自助测试具备预检查和自助测试两个功能：

- 预检查：自动完成厂商信息、产品信息、Profile、插件的完整性和合法性检查。预检查结果有三种状态：
  - **预检查通过：**所有信息均通过预检查，可以进入下一阶段。
  - **预检查通过，部分信息缺失，但不影响自助测试：**部分信息不完整，但不影响自助测试，可以进入下一阶段。
  - **重要信息缺失，请完善：**存在重要信息缺失，预检查不通过，不能进入下一阶段。
- 自助测试：提供端到端测试用例，帮助开发者自助完成产品功能测试。测试完成后，开发中心将生成测试报告，用于进行产品发布认证。

##### 预检查和自助测试

**步骤1** 在产品开发空间，点击“发起自助测试”。

**步骤2** 系统进入“预检查”界面，点击“开始检查”。



**步骤3** 系统返回预检查结果：如果预检查通过，则继续进行下一阶段测试；如果预检查未通过，则根据提示信息进行修改，并重新进行预检查。

The screenshot shows the 'Pre-check' interface. At the top, there is a red-bordered button labeled 'Pre-check passed'. Below it, a status bar displays '检查项总数: 17' (Total items checked: 17), '通过项: 17' (Passed: 17), '警告项: 0' (Warning items: 0), and '严重项: 0' (Severe items: 0). The main content area is divided into two sections: 'Factory information check' and 'Product information check'. Under 'Factory information check', all items are marked as 'Passed'. Under 'Product information check', three items are listed: 'Product introduction check' (Failed), 'Functionality check' (Pending), and 'Usage specification check' (Failed). Each failed item has a red-bordered note indicating it affects the review result and a 'Modify' button.

**步骤4** 预检查通过后，点击“自助测试”。开发者需要使用真实设备进行自助测试。

The screenshot shows the 'Self-test' interface. It features a horizontal navigation bar with three steps: 1. Pre-check (red border), 2. Self-test (blue border), and 3. Application submission. Below the navigation, a central panel contains the text '请使用真实物理设备进行自助测试' (Please use a real physical device for self-test) and several small blue links for 'Preview report', 'How to self-test', 'Restart test', and 'Download report'.

**步骤5** 根据测试用例说明，依次完成**终端开机入网测试>数据上报测试>数据上报属性测试>控制命令下发测试>命令下发响应测试**。所有测试用例全部执行成功，产品开发才全部完成。

#### 说明

当产品的“数据格式”为“二进制码流”时，才需要进行命令下发响应测试；否则，开发者不需要执行此测试用例。

The screenshot shows the 'Test Case Execution' interface. It displays a series of test cases with their execution results. The first test case, '终端开机入网测试', is shown with its configuration (设备类型: 非加密, verifyCode: 123456789) and a green checkmark indicating 'Execution successful'. The second test case, '数据上报测试', also shows a green checkmark and a success message: '用例执行成功' (Test case executed successfully) with the message '(batteryLevel=1, SignalPower=0, upData=0, TxPower=0)' and '上报时间: 2018-09-07 16:58:33'.

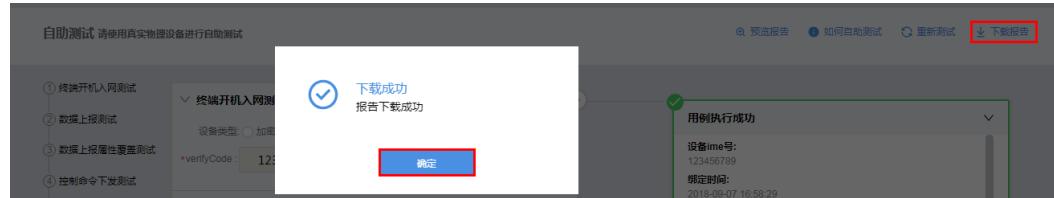
----结束

## 预览和下载测试报告

**步骤1** 在通过自助测试后，点击“预览报告”可以对测试报告进行预览。



**步骤2** 点击“下载报告”，将测试报告下载到本地。



----结束

### 3.3.5.2 认证测试用例

#### 3.3.5.2.1 终端开机入网测试

**步骤1** 选择“设备类型”，并填写“VerifyCode”。点击右方启动按钮开始测试。

如果设备类型选择“加密”，还需要填写“PSK”。



**步骤2** 使用真实设备进行绑定上线操作，点击右方停止按钮查看测试用例执行结果。



----结束

### 3.3.5.2.2 数据上报测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 使用真实设备上报Profile中定义的任意一个属性，点击右方停止按钮查看测试用例执行结果。



----结束

### 3.3.5.2.3 数据上报属性测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 使用真实设备上报Profile中定义的所有属性，点击右方停止按钮查看测试用例执行结果。





----结束

#### 3.3.5.2.4 控制命令下发测试

**步骤1** 点击右方启动按钮开始测试。



**步骤2** 物联网平台会根据Profile的定义，向设备下发一条命令，命令取值随机。在真实设备侧，查看接收到的命令取值，并在测试窗口填写后，点击右方停止按钮查看测试用例执行结果。

当产品的“数据格式”为“二进制码流”时，则在测试窗口需要填写二进制或十六进制内容；当产品的“数据格式”为“JSON”时，则在测试窗口需要填写JSON格式的内容。此处以二进制或十六进制内容为例。



----结束

### 3.3.5.2.5 命令下发响应测试

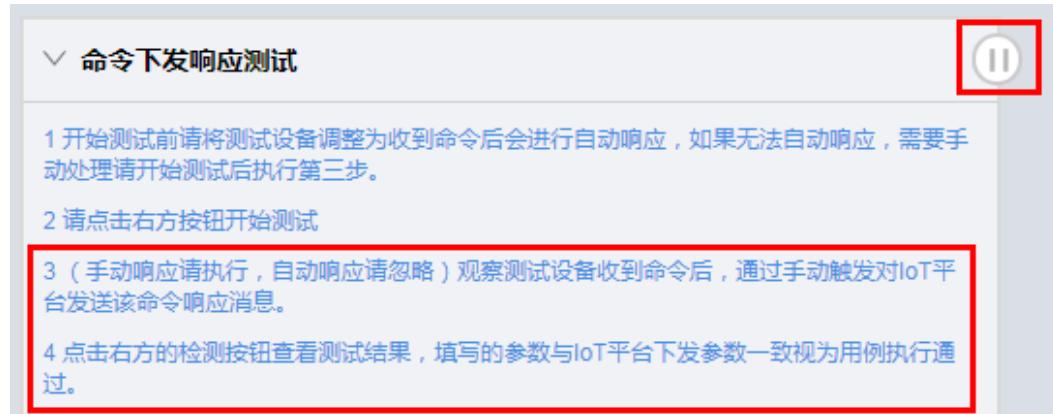
当产品的“数据格式”为“二进制码流”时，才需要进行命令下发响应测试；否则，开发者不需要执行此测试用例。如果该款产品的设备不返回命令执行结果，则开发者也不需要执行此测试用例。

**步骤1** 将真实设备设置为收到命令后返回命令执行结果，点击右方启动按钮开始测试。

如果真实设备无法自动返回命令执行结果，则直接启动测试，并在**步骤2**中手动使真实设备上报命令执行结果。



**步骤2** 如果真实设备支持自动返回命令执行结果，则在真实设备收到命令后，点击右方停止按钮查看测试用例执行结果；如果真实设备不支持自动返回命令执行结果，则根据真实设备收到的命令，手动使真实设备上报命令执行结果后，点击右方停止按钮查看测试用例执行结果。



----结束

### 3.3.6 产品发布

#### 概述

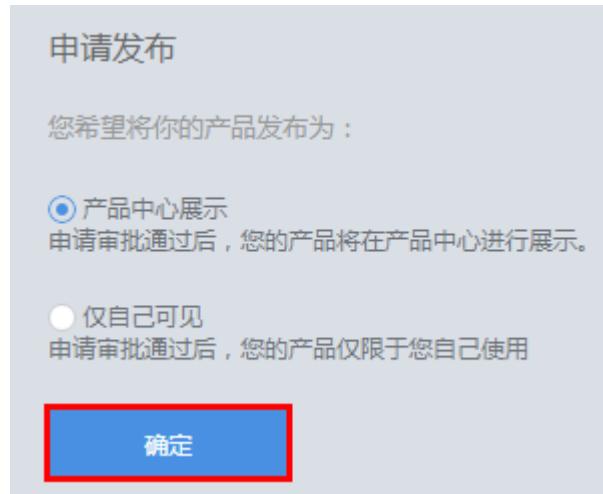
如果开发中心已经对接运营中心，则开发者可以向运营中心申请发布通过认证测试的产品：可以申请在产品中心公开展示，也可以申请仅自己可见。

#### 申请发布产品

**步骤1** 产品在通过自助测试后，点击“申请发布”。



**步骤2** 选择产品发布模式：“产品中心展示”或“仅自己可见”，点击确定。



----结束

### 3.3.7 上线商用平台

#### 概述

商用平台对接产品中心后，可以直接导入在产品中心发布的产品资源（含Profile、编解码插件等）。

#### 导入产品

**步骤1** 购买IoT平台增强版服务后，在IoT平台控制台进入管理门户。购买方式请参考[如何订购IoT平台增强版服务](#)。

IoT平台增强版

再次购买 进入开发者平台

总览

IoT平台增强版

•您已购买1个IoT平台增强版设备管理基本包，对应的1个IoT平台账号如下表所示。  
•您可进入管理门户管理您的设备和应用，详情请查看[用户指南](#)。如需访问进入开发者平台，详情请参考[开发者指南](#)。

IoT平台账号	创建时间	到期时间	基本包状态	按需资源状态	计费模式	操作
[REDACTED]	2018-06-29 15:46:18	2019-06-29 23:59:59	运行中	运行中	包月/包年	<a href="#">进入管理门户</a> 更多

**步骤2** 在管理门户选择“系统管理 > 应用管理 > 应用列表”，点击“创建应用”。



如果相应的应用已经存在，则可以跳过此步骤。



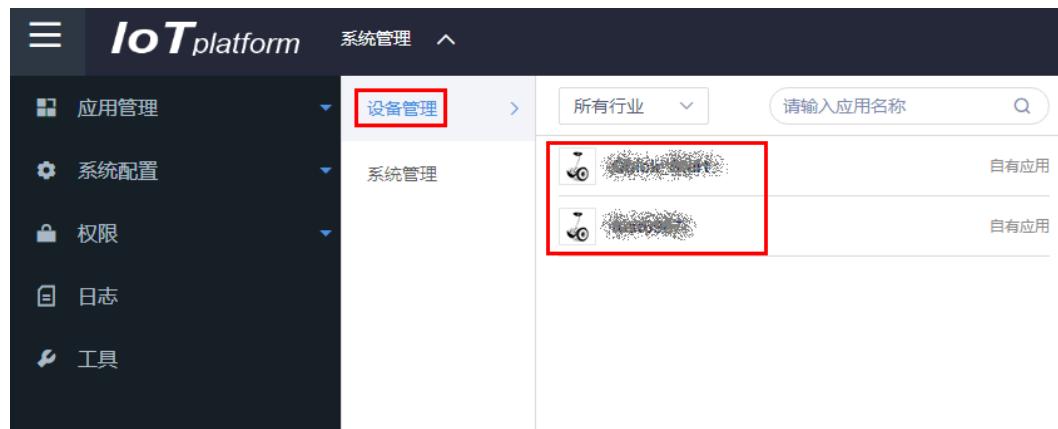
在“创建应用”界面，完成各项参数的配置后，点击“确定”。

The screenshot shows the 'Create Application' configuration page. It includes three main sections: '基本信息' (Basic Information) with fields for application name ('myFirstApp') and industry ('智慧家庭行业'), and a note about message tracking data collection; '消息推送' (Message Delivery) with a dropdown for '推送协议' ('HTTPS') and a note about CA certificates; and '平台能力' (Platform Capabilities) with a note about selecting service components. At the bottom are '确定' (Confirm) and '取消' (Cancel) buttons.

系统将弹出应用创建成功窗口，包含应用ID、应用秘钥等信息。密钥信息在应用详细页内不可见，请妥善保管。



**步骤3** 选择“设备管理 > 具体应用”，进入该应用的“设备管理”界面。



**步骤4** 在“产品模型”界面，点击“新建产品模型”，并选择“从产品中心导入”。



**步骤5** 在“产品中心”界面，选择需要导入的产品。



**步骤6** 在“产品详情”界面，点击“导入该产品”。



----结束

# 4 应用开发

## 4.1 开发说明

为了降低应用侧的开发难度、提升应用侧开发效率，物联网平台向应用侧开放了丰富的API。应用开发即应用侧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的过程。

在进行应用开发之前，请检查以下条件是否满足：

- 物联网平台已经创建对应的项目，详见[项目创建](#)。
- 物联网平台已经部署相应的。如果设备侧的“数据格式”为“JSON”，则不需要开发编解码插件。
- 应用侧通过HTTPS调用物联网平台的API前，需要集成相关证书，详见[获取相关证书](#)。
- 获取[调用API接口的代码样例（Java）](#)，并参考[准备Java开发环境](#)完成开发环境的配置和样例代码的导入。

### 说明

本文档基于调用API接口的代码样例进行指导。同时，物联网平台还提供了供应用侧集成的SDK（含Java、PHP、Python等语言），以便于应用侧进行开发，详见[应用侧SDK使用指南](#)。

## 4.2 应用接入

### 概述

应用服务器需要调用物联网平台的“鉴权”接口，完成应用服务器和物联网平台的对接，接口信息详见API参考文档。

本文档基于调用API接口的代码样例（Java）进行指导，帮助开发者理解“鉴权”接口的调用。

### 操作指导

**步骤1** 参考[准备Java开发环境](#)章节，准备好Java开发环境。

本手册以Java开发环境为例进行说明，如果您使用其它类型的开发环境，请根据自己的需要完成部署。

## 步骤2 在eclipse中，选择“src > com.huawei.utils > Constant.java”，修改BASE\_URL、APPID、SECRET。

```

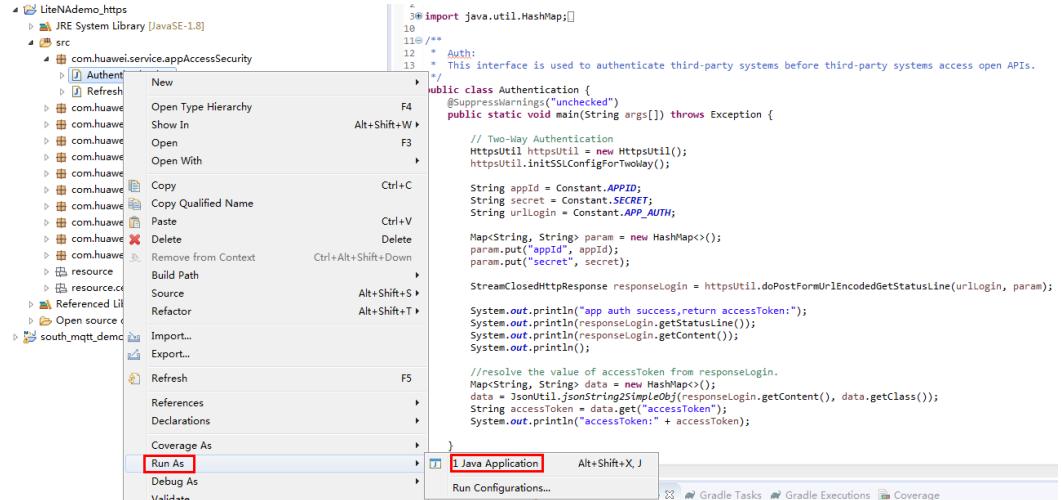
1 package com.huawei.utils;
2
3 public class Constant {
4
5     /*please replace the IP and Port of the IoT platform environment address, when you use the demo.
6     public static final String BASE_URL = "https://192.168.1.100:8080";
7
8     /*please replace the appId and secret, when you use the demo.
9     public static final String APPID = "56gPvge142Uf480gCQ/g1A0e";
10    public static final String SECRET = "HMPAYq8Slawkrzuh8O189Vzh";
11
12    /*
13     *IP and port of callback url.
14     *please replace the IP and Port of your Application deployment environment address, when you use the demo.
15     */
16
17    public static final String DEVICE_ADDED_CALLBACK_URL = "http://192.168.1.100:8080/addDevice";
18    public static final String DEVICE_INFO_CHANGED_CALLBACK_URL = "http://192.168.1.100:8080/updateDeviceInfo";
19    public static final String DEVICE_DELETED_CALLBACK_URL = "http://192.168.1.100:8080/deleteDeviceData";
20    public static final String MESSAGE_CONFIRM_CALLBACK_URL = "http://192.168.1.100:8080/commandConfirmData";
21    public static final String SERVICE_INFO_CHANGED_CALLBACK_URL = "http://192.168.1.100:8080/updateServiceInfo";
22    public static final String COMMAND_RSP_CALLBACK_URL = "http://192.168.1.100:8080/commandRspData";
23    public static final String RULE_EVENT_CALLBACK_URL = "http://192.168.1.100:8080/deviceRuleEvent";
24    public static final String DEVICE_DATA_CHANGED_CALLBACK_URL = "http://192.168.1.100:8080/updateDeviceData";
25    public static final String DEVICE_SHADOW_MODIFIED_CALLBACK_URL = "http://192.168.1.100:8080/modifyDeviceDesired";
26
27    /*
28     * complete callback url.
29     * please replace url, when you use the demo.
30     */
31
32    /*
33     * This interface is used to authenticate third-party systems before third-party systems access open APIs.
34     */
35
36    public static void main(String args[]) throws Exception {
37        // Two-Way Authentication
38        HttpsUtil httpsutil = new HttpsUtil();
39        httpsutil.initSSLConfigForTwoWay();
40
41        String appId = Constant.APPID;
42        String secret = Constant.SECRET;
43        String urlLogin = Constant.APP_AUTH;
44
45        Map<String, String> param = new HashMap<>();
46        param.put("appId", appId);
47        param.put("secret", secret);
48
49        StreamClosedHttpResponse responseLogin = httpsutil.doPostFormUrlEncodedGetStatusLine(urlLogin, param);
50
51        System.out.println("app auth success,return accessToken:");
52        System.out.println(responseLogin.getStatusLine());
53        System.out.println(responseLogin.getContent());
54        System.out.println();
55
56        //resolve the value of accessToken from responseLogin.
57        Map<String, String> data = new HashMap<>();
58        data = JsonUtil.jsonString2SimpleObj(responseLogin.getContent(), data.getClass());
59        String accessToken = data.get("accessToken");
60        System.out.println("accessToken:" + accessToken);
61
62    }
63
64}

```

配置说明如下：

- **BASE\_URL:** 填写应用对接地址/端口号。
- **APPID:** 填写创建应用或项目后获取的应用ID。
- **SECRET:** 填写创建应用或项目后获取的应用密钥。

## 步骤3 在eclipse中，选择“src > com.huawei.service.appAccessSecurity”，右键单击“Authentication.java”，选择“Run As > Java Application”。



## 步骤4 在控制台查看响应消息的打印日志，如果获得accessToken，说明鉴权成功。

accessToken请妥善保存，以便于在调用其它接口时使用。

```

[terminated] Authentication [Java Application] C:\Program Files\Java\jdk1.8.0_45\bin\javaw.exe (2018年3月14日下午4:37:54)
app auth success,return accessToken:
HTTP/1.1 200 OK{"accessToken":"8d44bc6cdb60e863dfdbb8ccfed4e51","tokenType":"bearer","refreshToken":"9e11397df47877fe2afe4cd5c463621","expiresIn":3600,
accessToken:8d44bc6cdb60e863dfdbb8ccfed4e51

```

 说明

- 如果没有得到正确的响应，请检查全局常量是否修改正确，并排除网络问题。或参考[单步调试](#)的内容进行问题定位。
- accessToken会在“expiresIn”所标志的时间内过期，“expiresIn”的单位为“秒”。
- accessToken过期后需要重新获取。可以使用“鉴权”接口重新获取，也可以使用上一次鉴权得到的refreshToken来获取新的accessToken。“刷新Token”接口信息详见API参考文档和“代码样例”中的“RefreshToken.java”。
- “北向JAVA API Demo”中提供了各接口调用的消息示例，参见“src > resource > demo\_TCP\_message.json”。

----结束

## 4.3 订阅

### 概述

应用服务器通过调用物联网平台的“订阅平台业务数据”接口，告知物联网平台消息推送的地址和通知类型，比如设备业务数据、设备告警等，接口信息详见API参考文档。

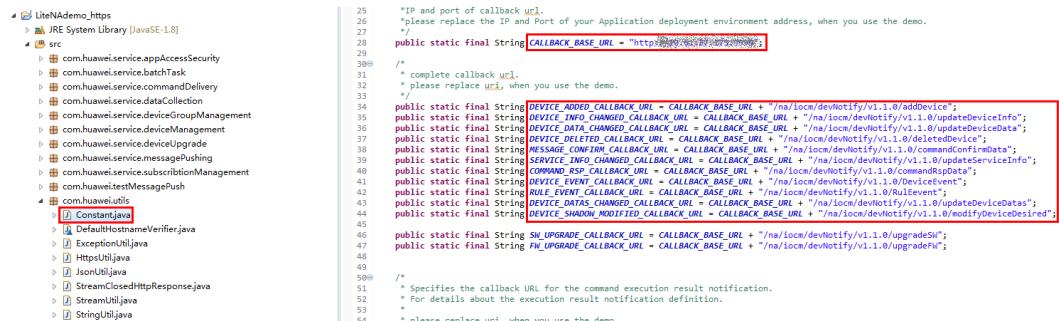
在订阅场景下，物联网平台是客户端，应用服务器是服务端，物联网平台调用应用服务器的接口，并向应用服务器推送消息。此时，如果订阅的回调地址为HTTPS地址，则需要在物联网上传CA证书。CA证书由应用服务器侧提供（证书获取方法可参考[导出CA证书](#)），并在开发中心的“应用 > 对接信息 > 应用安全 > 推送证书”上传，详见[上传CA证书](#)。

本文档基于调用API接口的代码样例（Java）进行指导，帮助开发者理解“订阅平台业务数据”接口的调用。

### 操作指导

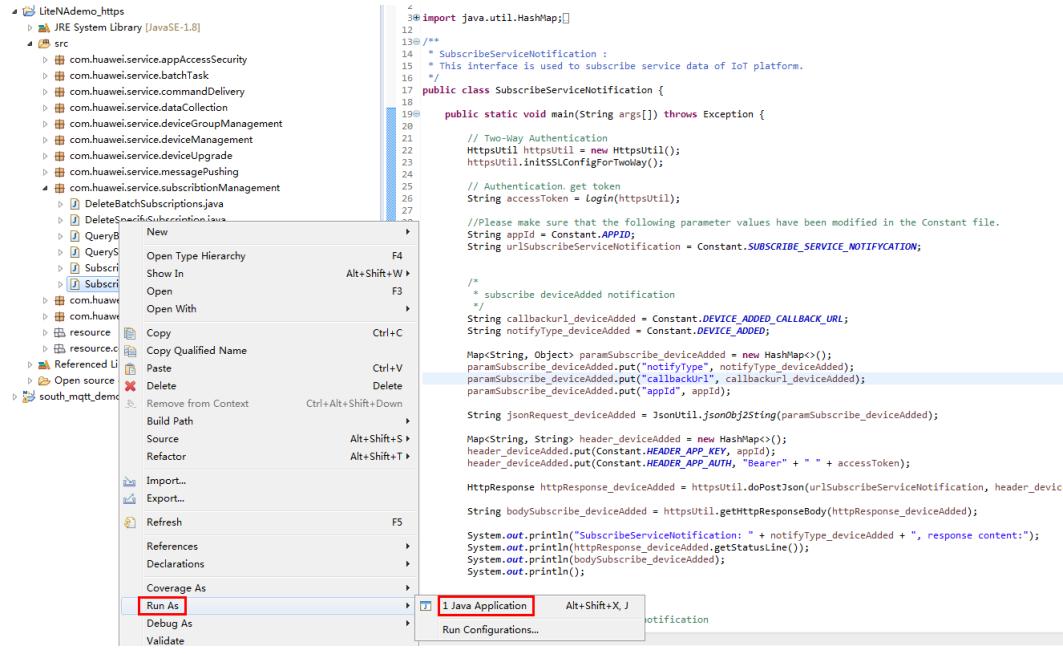
**步骤1** 在eclipse中，选择“src > com.huawei.utils > Constant.java”，修改“CALLBACK\_BASE\_URL”，填写回调的IP地址和端口号。

同一个应用下，所有订阅类型的回调地址的IP和端口必须一致。回调地址的合法性和连通性可以通过开发中心的“订阅调试”功能进行检测。



```
25 *IP and port of callback url.  
26 /*please replace the IP and Port of your Application deployment environment address, when you use the demo.  
27 */  
28 public static final String CALLBACK_BASE_URL = "http://192.168.1.100:9999";  
29 /*  
30 * complete callback url.  
31 * please replace url, when you use the demo.  
32 */  
33 /*  
34 public static final String DEVICE_ADDED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/addDevice";  
35 public static final String DEVICE_INFO_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceInfo";  
36 public static final String DEVICE_DATA_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceData";  
37 public static final String DEVICE_DELETED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/deleteDevice";  
38 public static final String MESSAGE_CONFIG_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/configMessage";  
39 public static final String SERVICE_INFO_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateServiceInfo";  
40 public static final String COMMAND_RSP_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/commandRpdata";  
41 public static final String DEVICE_EVENT_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/deviceEvent";  
42 public static final String DEVICE_UPGRADE_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/upgrade";  
43 public static final String DEVICE_DATA_CHANGED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/updateDeviceDatas";  
44 public static final String DEVICE_SHADOW_MODIFIED_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/modifDeviceDesired";  
45 /*  
46 public static final String SH_UPGRADE_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/upgradeSh";  
47 public static final String FW_UPGRADE_CALLBACK_URL = CALLBACK_BASE_URL + "/na/locm/devNotify/v1.1.0/upgradFw";  
48 */  
49 /*  
50 * Specifies the callback URL for the command execution result notification.  
51 * For details about the execution result notification definition.  
52 *  
53 * please replace url, when you use the demo.  
54 */
```

**步骤2** 在eclipse中，选择“src > com.huawei.service.subscriptionManagement”，右键单击“SubscribeServiceNotification.java”，选择“Run As > Java Application”。



**步骤3** 在控制台查看响应消息的打印日志，如果所有类型的订阅均获得“201 Created”响应，则说明订阅成功。

```

app auth success,return accessToken:
HTTP/1.1 200 OK("accessToken":"c5166051d46626b1f1a3590d17e4a3a","tokenType":"bearer","refreshToken":"357e88ea50a45d523b7e5ff9165cf58","expiresIn":3600,"

SubscribeNotification: deviceAdded, response content:
HTTP/1.1 201 Created

SubscribeNotification: deviceInfoChanged, response content:
HTTP/1.1 201 Created

SubscribeNotification: deviceDataChanged, response content:
HTTP/1.1 201 Created

SubscribeNotification: deviceDeleted, response content:
HTTP/1.1 201 Created

SubscribeNotification: messageConfirm, response content:
HTTP/1.1 201 Created

SubscribeNotification: serviceInfoChanged, response content:
HTTP/1.1 201 Created

SubscribeNotification: commandRsp, response content:
HTTP/1.1 201 Created

SubscribeNotification: deviceEvent, response content:
HTTP/1.1 201 Created

SubscribeNotification: ruleEvent, response content:
HTTP/1.1 201 Created

SubscribeNotification: deviceDatasChanged, response content:
HTTP/1.1 201 Created

```

### 说明

- 如需修改订阅的回调地址，在“Constants.java”类中修改“CALLBACK\_BASE\_URL”的值，再次运行“SubscribeServiceNotification.java”即可，新的回调地址会覆盖原来的回调地址。
- 订阅完成后，开发者可参考“src > com.huawei.testMessagePush > SimpleHttpServer.java”搭建一个应用服务器来接收平台推送的Post消息（仅供参考）。如果需要在本地测试平台回调功能和查看回调内容，可以使用“北向JAVA API Demo”提供的类“src > com.huawei.testMessagePush > TestSubscribeAllServiceNotification.java”，并参考[数据上报](#)中的操作。

----结束

## 4.4 注册设备

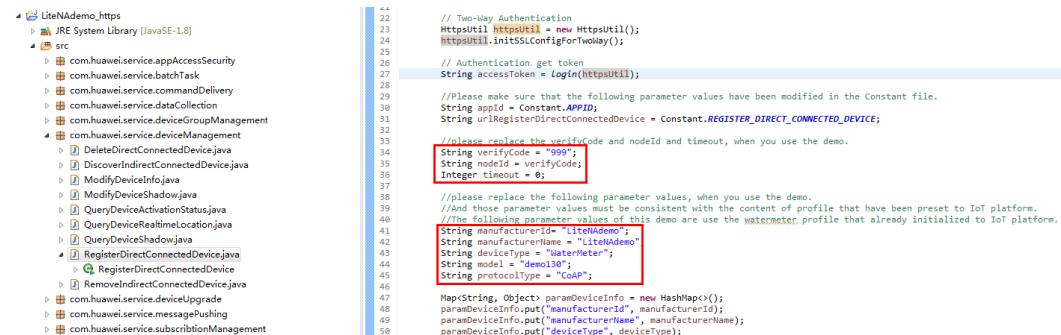
### 概述

应用服务器通过调用物联网平台的“注册直连设备”接口，在物联网平台添加设备，接口信息详见API参考文档。

本文档基于调用API接口的代码样例（Java）进行指导，帮助开发者理解“注册直连设备”接口的调用。

### 操作指导

- 步骤1** 在eclipse中，选择“src > com.huawei.service.deviceManagement > RegisterDirectConnectedDevice.java”，修改“verifyCode”、“nodeId”、“timeout”、“manufacturerId”、“manufacturerName”、“deviceType”、“model”、“protocolType”的取值。



```
22 // Two-way Authentication
23 HttpsUtil httpsUtil = new HttpsUtil();
24 httpsUtil.initSSLConfigForTwoWay();
25
26 // Authentication get token
27 String accessToken = Login(httpsUtil);
28
29 //Please make sure that the following parameter values have been modified in the Constant file.
30 String appid = Constant.APPID;
31 String urlRegisterDirectConnectedDevice = Constant.REGISTER_DIRECT_CONNECTED_DEVICE;
32
33 //Please replace the verifyCode and nodeId and timeout, when you use the demo.
34 String verifyCode = "999";
35 String nodeId = verifyCode;
36 Integer timeout = 0;
37
38 //Please replace the following parameter values, when you use the demo.
39 //And those parameter values must be consistent with the content of profile that have been preset to IoT platform.
40 //In order to use the demo use the watermeter profile that already initialized to IoT platform.
41 String manufacturerId= "LiteNADemo";
42 String manufacturerName = "LiteNADemo";
43 String deviceType = "WaterMeter";
44 String model = "demo130";
45 String protocolType = "CoAP";
46
47 Map<String, Object> paramDeviceInfo = new HashMap<>();
48 paramDeviceInfo.put("manufacturerId", manufacturerId);
49 paramDeviceInfo.put("manufacturerName", manufacturerName);
50 paramDeviceInfo.put("deviceType", deviceType);
```

配置说明如下：

- “verifyCode/nodeId”需要与真实设备的唯一标识符（IMEI或mac）一致。如果使用的是设备模拟器，则“verifyCode”可以是数字、字母和特殊符号的组合，开发者可自行定义，但不可以与其它设备的verifyCode重复。
- “timeout”单位是“秒”，“timeout”的取值作用如下：
  - timeout = 0，注册的设备不会过期。
  - timeout > 0，真实设备必须在设置的时间内上线，否则注册的设备会因为过期而被IoT平台删除。如果不携带timeout，则默认过期时间是180秒。
  - 在设备绑定成功后，“timeout”不再起作用，注册的设备不会过期。
- “manufacturerId”、“manufacturerName”、“deviceType”、“model”、“protocolType”需要与对应的Profile保持一致。

- 步骤2** 右键单击“RegisterDirectConnectedDevice.java”，选择“Run As > Java Application”。

- 步骤3** 在控制台查看响应消息的打印日志，如果获得“deviceId”，则说明注册成功。

可以在开发中心的“产品 > 设备管理”中，查看新注册的设备是否已经显示。此时，注册设备只有设备ID（deviceId）信息。

```
app auth success,return accessToken:  
HTTP/1.1 200 OK{"accessToken":"1f337bfa6cb85f83243b99fda22d21b","tokenType":"bearer","refreshToken":"be3cccdce5390ed14b81c85f58e2712b2","expiresIn":3600,  
RegisterDirectlyConnectedDevice, response content:  
HTTP/1.1 200 OK{"deviceId":"d0ac5cac-d3af-4e0c-8166-5a67e1c6f0a7","verifyCode":"9999","timeout":0,"psk":"5a7f074ffdb1c46ed104a8efcafd0a0e"}
```

----结束

## 4.5 设备接入

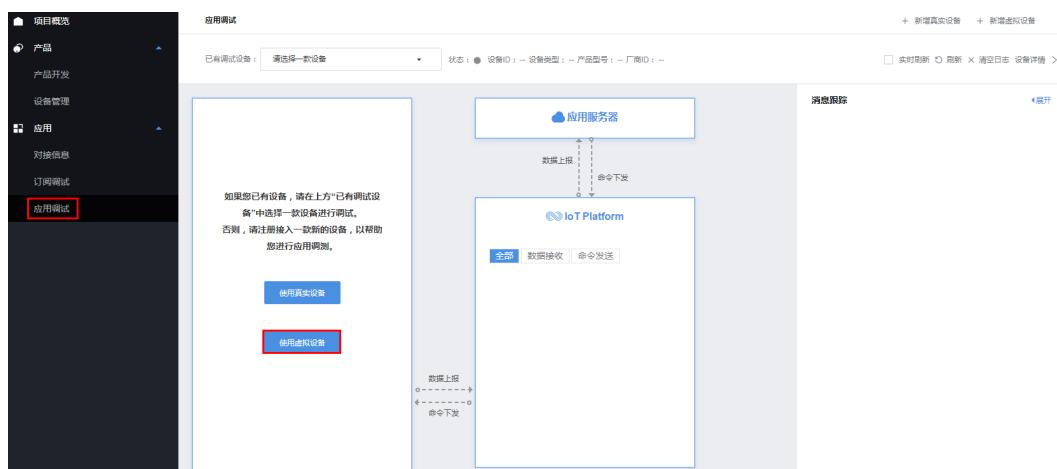
### 概述

设备接入物联网平台后，才可以经由物联网平台与应用服务器进行数据交互。

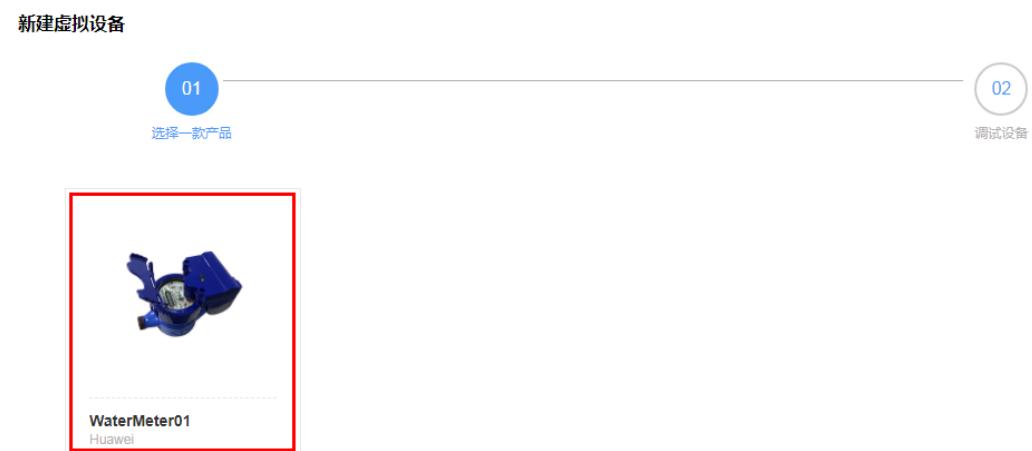
开发中心提供了应用调试功能，可以模拟设备接入物联网平台的场景。如果不使用开发中心的应用调试功能，请使用真实设备接入物联网平台。如下为使用开发中心模拟设备接入平台的操作指导：

### 操作指导

**步骤1** 选择“应用 > 应用调试”，点击“使用虚拟设备”。



**步骤2** 系统将弹出“新建虚拟设备”窗口，根据提示选择一款产品。



----结束

## 4.6 数据上报

### 概述

设备上报数据后，由物联网平台将设备上报的数据推送到订阅地址。开发中心提供了设备模拟器，可以模拟真实设备上报数据的场景。如果不使用开发中心的设备模拟器，请使用真实设备进行数据上报。

本文档基于设备模拟器和调用API接口的代码样例（Java）进行指导。为了便于开发者测试物联网平台是否已推送消息到订阅地址，调用API接口的代码样例（Java）中提供了一个简单的HTTP服务器，用于模拟接收推送数据的场景。

### 操作指导

- 步骤1** 在eclipse中，选择“src > com.huawei.testMessagePush > NotifyType.java”，修改“TEST\_CALLBACK\_BASE\_URL”，填写本地IP地址和端口号，端口号不能被本地其它程序占用。



The screenshot shows the Eclipse IDE interface with the project 'LiteNAdemo\_https' selected. The 'src' folder contains several Java files under the package 'com.huawei.testMessagePush'. The file 'NotifyType.java' is open in the editor. The code defines a class 'NotifyType' with a static final string 'TEST\_CALLBACK\_BASE\_URL' set to 'http://192.168.1.10:8080'. The code also includes a list of notify types and their descriptions.

```
2 import java.util.ArrayList;
3
4 public class NotifyType {
5
6     //please replace the IP and Port to your localhost IP and port, when you use the testMessagePush.
7     public static final String TEST_CALLBACK_BASE_URL = "http://192.168.1.10:8080";
8
9     public static List<String> notifyTypes = new ArrayList<>();
10
11    public static List<String> getNotifyTypes () {
12        notifyTypes.add(SERVICE_INFO_CHANGED);
13        notifyTypes.add(DEVICE_INFO_CHANGED);
14        notifyTypes.add(DEVICE_DATA_CHANGED);
15        notifyTypes.add(DEVICE_DELETED);
16        notifyTypes.add(DEVICE_ADDED);
17        notifyTypes.add(DEVICE_DELETED);
18        notifyTypes.add(MESSAGE_CONFIRM);
19        notifyTypes.add(COMMAND_RSP);
20        notifyTypes.add(DEVICE_EVENT);
21        notifyTypes.add(RULE_EVENT);
22        notifyTypes.add(DEVICE_DATAS_CHANGED);
23        notifyTypes.add(DEVICE_DESIRED MODIFY);
24    }
25
26    public static List<String> getManagementNotifyTypes () {
27        notifyTypes.add(SM_UPGRADE_STATE_CHANGED);
28        notifyTypes.add(SM_UPGRADE_RESULT);
29        notifyTypes.add(FW_UPGRADE_STATE_CHANGED);
30        notifyTypes.add(FW_UPGRADE_RESULT);
31    }
32}
```

- 步骤2** 右键单击“src > com.huawei.testMessagePush > TestSubscribeAllServiceNotification.java”，选择“Run As > Java Application”。

- 步骤3** 在相应的项目空间内，选择“应用 > 应用调试”，使用在**设备接入**时创建的虚拟设备，进行数据上报。

#### 说明

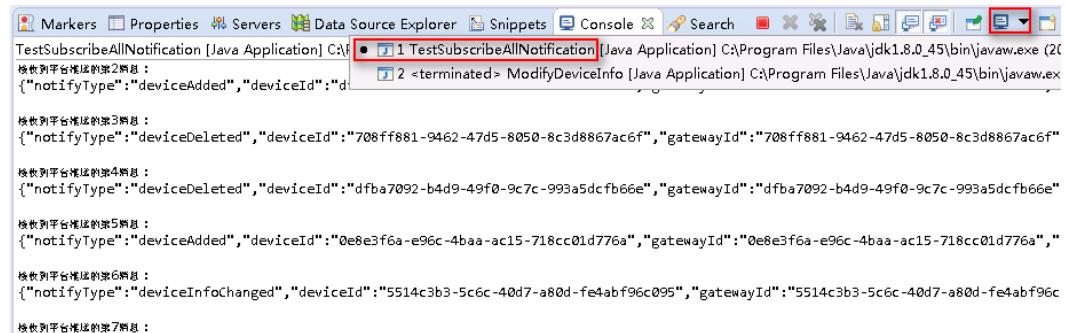
如果不使用开发中心的应用调试功能，请使用真实设备进行数据上报。

在“设备模拟器”区域，输入十六进制码流或者JSON数据（以十六进制码流为例），点击“发送”，在“IoT Platform”区域和应用服务器查看数据上报的结果，在“消息跟踪”区域查看物联网平台处理日志。



**步骤4** 在eclipse中“TestSubscribeAllNotification.java”的控制台查看物联网平台推送给应用服务器的消息。

此步骤也可以对“订阅”结果进行调测，比如订阅了“deviceAdded（添加新设备）”，则可以执行[注册设备](#)的操作后，在“TestSubscribeAllNotification.java”的控制台查看物联网平台推送的消息。



----结束

## 4.7 命令下发

### 概述

应用服务器需要调用物联网平台的“创建设备命令”接口或“设备服务调用”接口，对设备下发控制指令，接口信息详见API参考文档。

- 当设备应用层接入协议为LWM2M时，使用“创建设备命令”进行命令下发。
- 当设备应用层接入协议为MQTT协议时，使用“设备服务调用”进行命令下发。

本文档基于“创建设备命令”接口和调用API接口的代码样例（Java）进行指导，帮助开发者理解命令下发的场景。

### 操作指导

**步骤1** 在eclipse中，选择“src > com.huawei.service.commandDelivery > CreateDeviceCommand.java”，修改“deviceId”、“serviceId”、“method”、“paras”的取值。

```

public static void main(String[] args) throws Exception {
    // Two-way Authentication
    24   HttpsUtil httpsUtil = new HttpsUtil();
    25   httpsUtil.initSSLConfigForTwoway();
    26
    27   // Authentication get token
    28   String accessToken = Login(httpsUtil);
    29
    30   //Please make sure that the following parameter values have been modified in the Constant file.
    31   String urlCreateDeviceCommand = Constant.CREATE_DEVICE_CMD;
    32   String appid = Constant.APPID;
    33
    34   //please replace the deviceId, when you use the demo.
    35   String deviceId = "22ed0124-b429-4daa-97cb-c04b6707cc19";
    36   String callbackUrl = Constant.REPORT_CMD_EXEC_RESULT_CALLBACK_URL;
    37   Integer maxRetransmit = 3;
    38
    39   //please replace the following parameter values, when you use the demo.
    40   //And those parameter values must be consistent with the content of profile that have been preset to IoT platform.
    41   //The following parameter values of this demo are use the watermeter profile that already initialized to IoT platform.
    42   String serviceId = "Brightness";
    43   String method = "SET_DEVICE_LEVEL";
    44   ObjectNode paras = JsonUtil.convertObject2ObjectNode("{\"value\":\"12\"}");
    45
    46   Map<String, Object> paramCommand = new HashMap<>();
    47   paramCommand.put("serviceId", serviceId);
    48   paramCommand.put("method", method);
    49   paramCommand.put("paras", paras);
    50
    51
}

```

配置说明如下：

- “deviceId” 在注册设备时获得。
- “serviceId”、“method”、“paras” 和Profile的定义一致。

**步骤2** 右键点击“CreateDeviceCommand.java”，选择“Run As > Java Application”。

**步骤3** 在控制台查看命令下发的打印日志，如果获得“201 Created”响应，则说明命令已经下发到物联网平台。

```

app auth success,return accessToken:
HTTP/1.1 200 OK{"accessToken":"49b9b60d439d9f018a23c2c25d9e71","tokenType":"bearer","refreshToken":"51762486bb4df2142f904696a8ffb743","expiresIn":3600,"scope
PostAsynCommand, response content:
HTTP/1.1 201 Created{"commandId":"7e7ed2fd17c2449582c71adb0efa18b5","appId":"pQJNChou8mBo7anA7Fkw07luOaa","deviceId":"85ef387c-49cd-455d-9f70-a1bf1a990054",

```

如果开发者使用开发中心的应用调试功能模拟设备接入和数据上报，可以在相应的项目空间内，选择“应用 > 应用调试”，选择在[设备接入](#)时创建的虚拟设备，查看接收到的命令。

使用应用服务器进行命令下发后，在“设备模拟器”区域查看接收到的命令（以十六进制码流为例），在“消息跟踪”区域查看物联网平台处理日志。



----结束

## 4.8 其他接口

请参考API参考文档完成其他接口业务的开发。

# 5 参考信息

## 5.1 准备 Java 开发环境

本章节以Java语言为例，提供了安装JDK、配置环境变量、安装Eclipse的方法。

### 5.1.1 安装 JDK1.8

下载JDK1.8版本（比如：jdk-8u161-windows-x64.exe），双击进行安装。

JDK1.8官网下载地址：<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

### 5.1.2 配置 Java 环境变量（Windows 操作系统）

**步骤1** 右键单击“计算机”，选择“属性”。

图 5-1 计算机属性



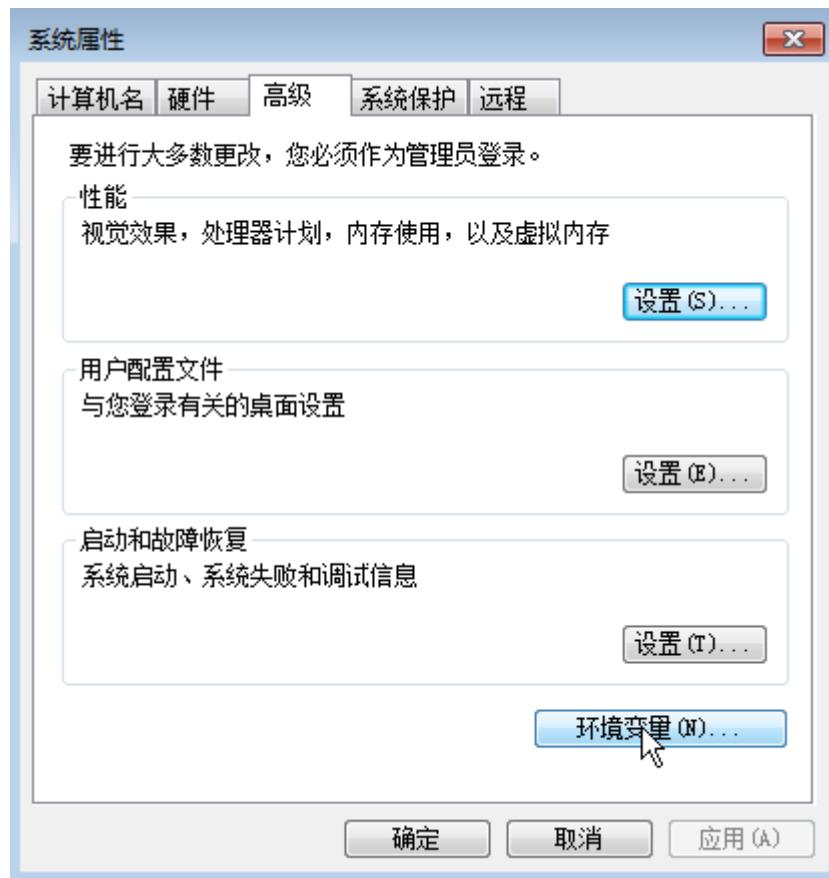
步骤2 点击“高级系统设置”。

图 5-2 系统



步骤3 点击“环境变量”。

图 5-3 系统属性



**步骤4** 配置系统变量。需配置3个变量：JAVA\_HOME、Path、CLASSPATH（不区分大小写）。若变量名已经存在，则点击“编辑”；若变量名不存在，则点击“新建”。一般Path变量存在，JAVA\_HOME变量和CLASSPATH变量需要新增。

图 5-4 环境变量



JAVA\_HOME指明JDK安装路径，配置示例：C:\ProgramFiles\Java\jdk1.8.0\_45。此路径下包括lib, bin等文件夹。

图 5-5 新建 JAVA\_HOME



Path变量使系统可以在任何路径下识别Java命令。如果Path变量已经存在，则需在变量值最后添加路径，配置示例：;C:\Program Files\Java\jdk1.8.0\_45\bin;C:\Program Files\Java\jdk1.8.0\_45\jre\bin。

两个路径之间需要使用“;”分割，分号是英文半角。

图 5-6 编辑 Path 变量

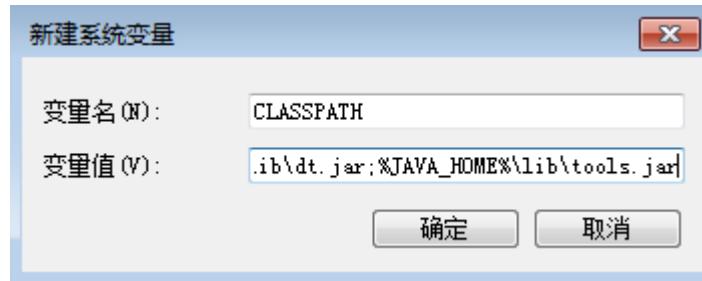


CLASSPATH为Java加载类（class或lib）路径，只有配置CLASSPATH，Java命令才能识别。配置示例：.;%JAVA\_HOME%\lib\dt.jar;%JAVA\_HOME%\lib\tools.jar。

**说明**

路径以“.”开始，表示当前路径。

图 5-7 编辑 CLASSPATH 变量



**步骤5** 重启系统，使环境变量生效。

**步骤6** 选择“开始 > 运行”，输入c “md” ，执行命令：java -version、java、javac。如果命令可以执行，则说明环境变量配置成功。

图 5-8 验证环境变量

```
C:\Users\z00293999>java -version
java version "1.8.0_45"
Java(TM) SE Runtime Environment (build 1.8.0_45-b15)
Java HotSpot(TM) Client VM (build 25.45-b02, mixed mode)
```

----结束

### 5.1.3 安装 Eclipse

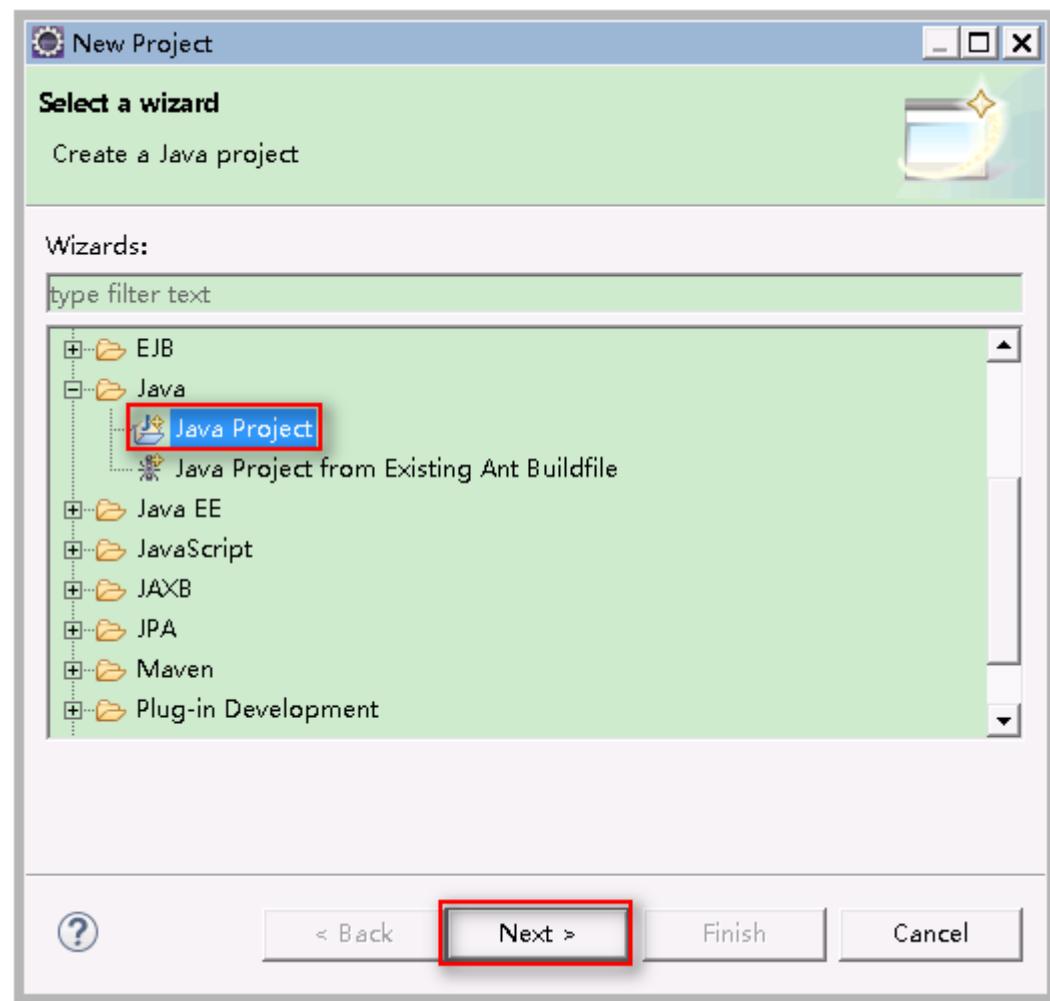
下载Eclipse安装包，直接解压缩到本地即可使用。

官网下载地址：<http://www.eclipse.org/downloads>。

### 5.1.4 新建工程

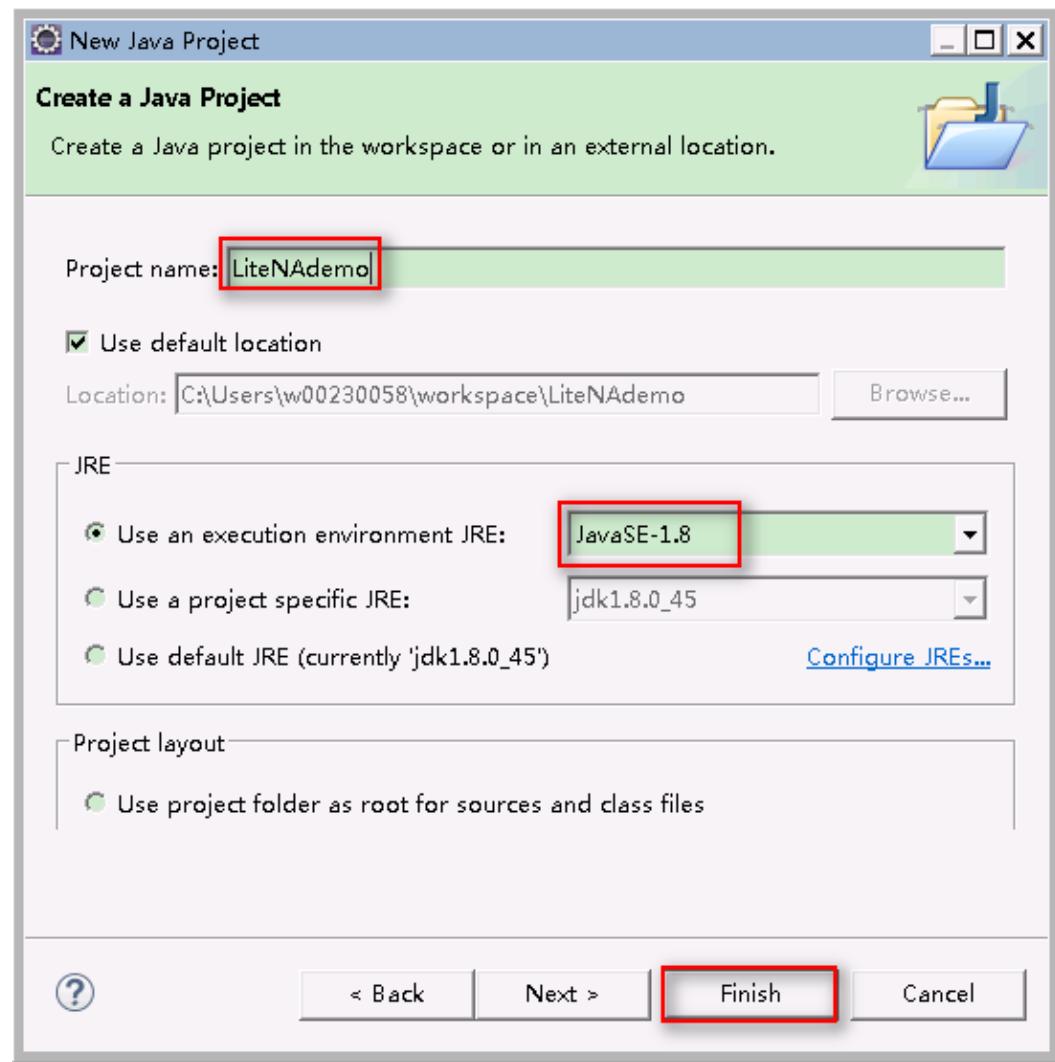
**步骤1** 运行Eclipse，选择“File > New > Project”，在弹出的对话框中选择“Java Project”，点击“Next”。

图 5-9 创建 Java 工程



**步骤2** 填写“Project name”，JRE版本选择“JavaSE-1.8”，点击“Finish”。

图 5-10 配置工程名称



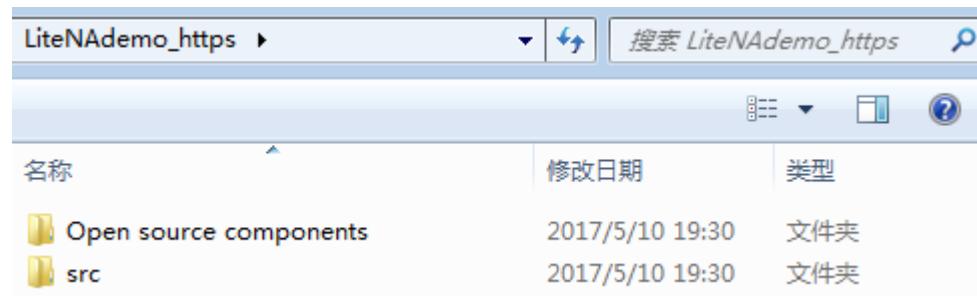
----结束

### 5.1.5 导入样例代码

**步骤1** 解压JAVA语言的接口调用样例（[点击获取](#)）。

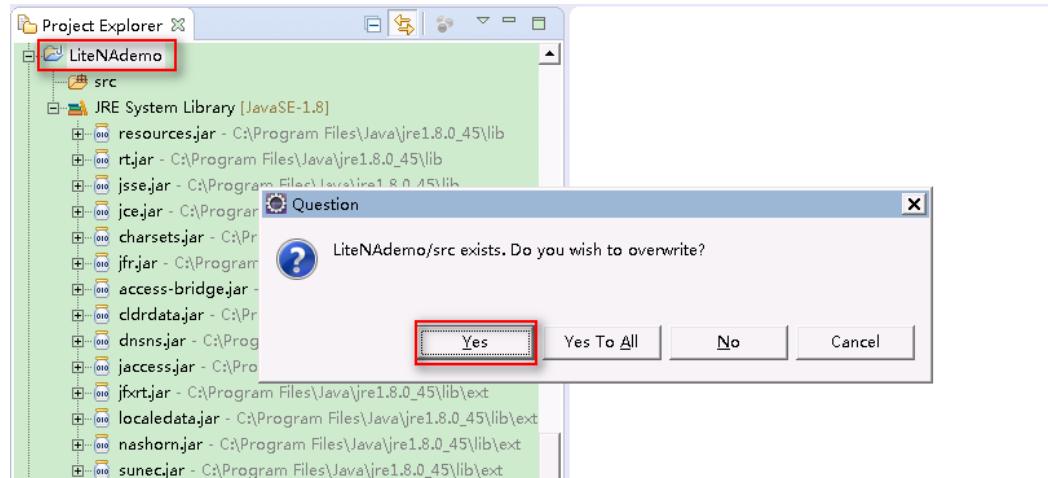
**步骤2** 完成解压后，拷贝（Ctrl+C）“Open source components”和“src”文件夹。

图 5-11 拷贝文件



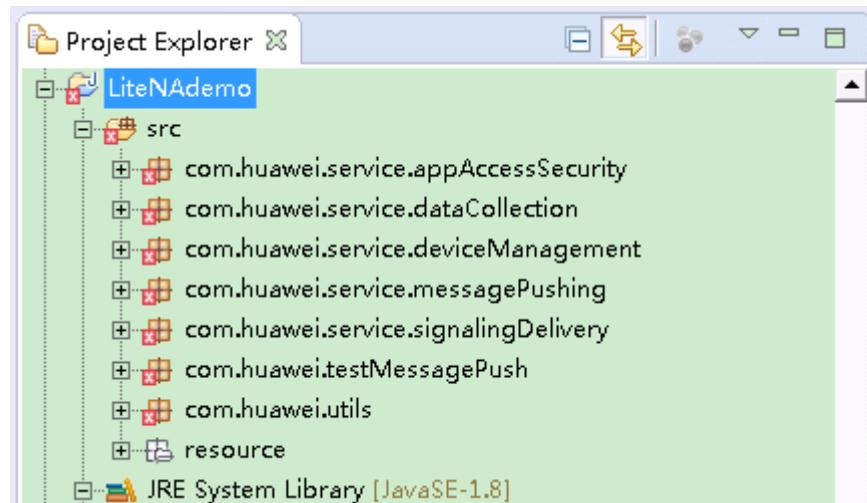
**步骤3** 打开在Eclipse创建的工程，点击选中工程名称，将拷贝的文件粘贴（Ctrl+V）到该工程目录下。

图 5-12 粘贴文件



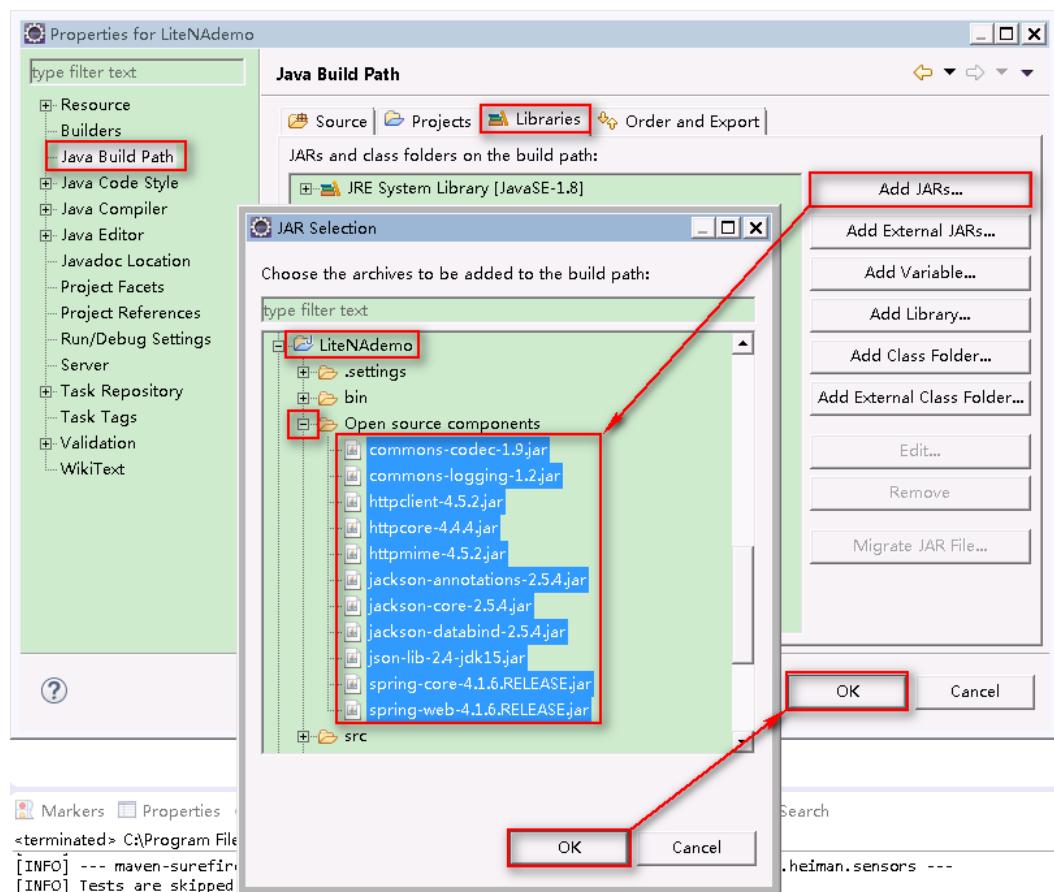
粘贴完成后，“src”目录下的文件存在错误。

图 5-13 “src” 目录存在错误



**步骤4** 右键单击工程名称，选择“Properties > Java Build Path > Libraries > Add JARs”，在弹出框中选中“Open source components”目录下的所有jar文件，点击“OK”。

图 5-14 导入 jar 文件



导入jar文件之后，“src”目录下文件的错误消失。

图 5-15 “src” 目录错误消失



----结束

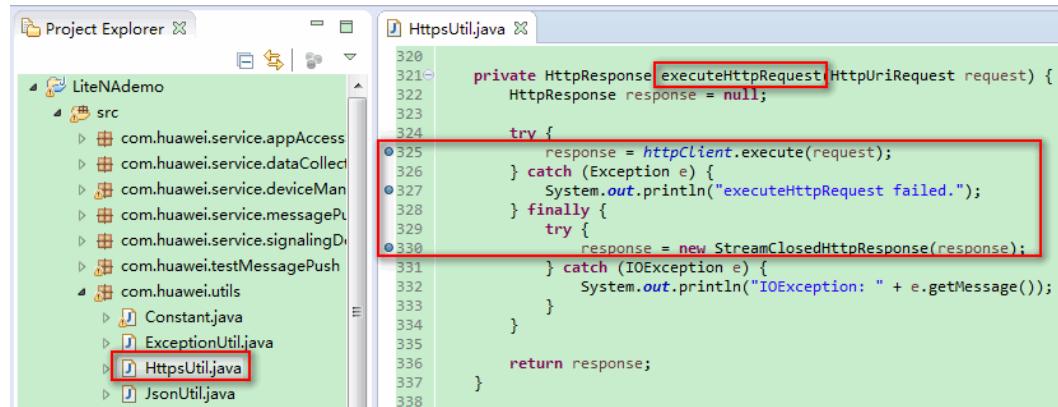
## 5.2 单步调测

为了更直观地查看应用程序发送的消息及IoT平台的响应消息，以下方法使用了Eclipse的断点调试方法。如果您使用Postman测试接口，请参考[使用Postman测试平台北向接口](#)。

**步骤1** 在最终发出http/https消息的代码处设置断点。

例如：在样例代码“HttpsUtil.java”中的“executeHttpRequest”方法设置3个断点（请根据您代码的实际情况打断点）。

**图 5-16 设置断点**

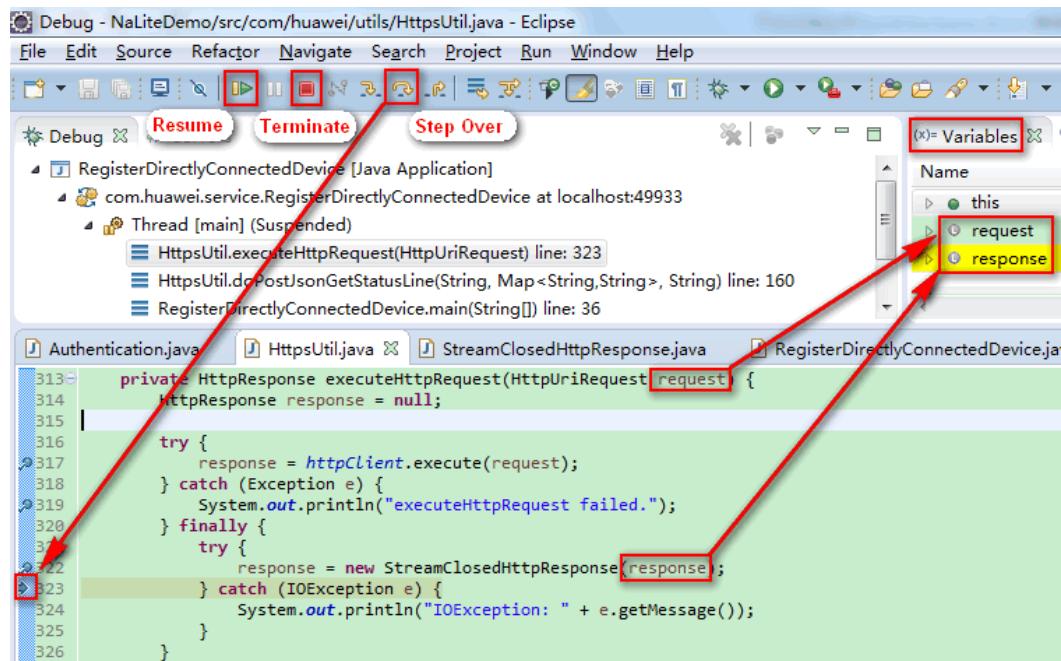


**步骤2** 右键单击需要调测的类，例如：“Authentication.java”（根据您建立的工程类型进行选择），选择“Debug As > Java Application”。

**步骤3** 当程序在断点位置停止运行后，点击“Step Over”进行单步调试。

此时可以在“Variables”窗口查看相应变量的内容，包括发送的消息及物联网平台的响应消息。

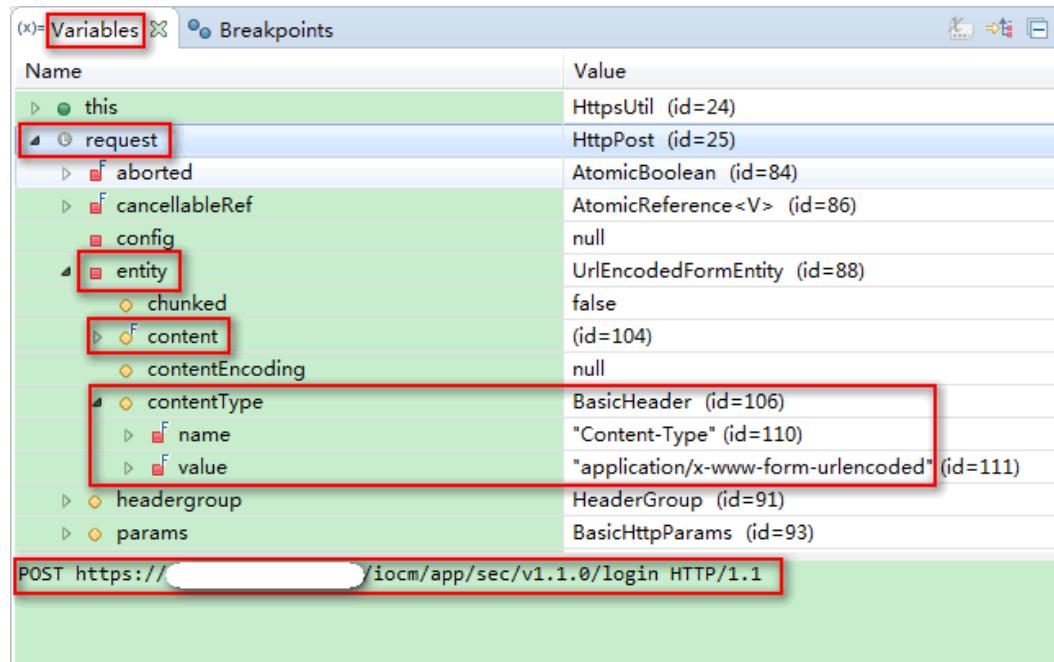
**图 5-17 单步调试**



**步骤4** 在“Variables”窗口中展开“request”变量，查看请求消息的内容。

选中“request”变量时，可以在下方内容展示区看到应用程序发送请求的URL；在“entity”中可以看到发送的消息内容。

图 5-18 展开“request”变量



应用ID（appId）和应用密钥（secret）在“content”字段内，使用十进制的ASCII码表示，需要对照ASCII码表将其转化为字母和符号。

图 5-19 查看“content”字段

Name	Value
this	HttpsUtil (id=24)
request	HttpPost (id=25)
aborted	AtomicBoolean (id=301)
cancellableRef	AtomicReference<V> (id=303)
config	null
entity	UrlEncodedFormEntity (id=305)
chunked	false
content	(id=311)
[0]	97 a
[1]	112 p
[2]	112 p
[3]	73 i
[4]	100 d
[5]	61 =
[6]	111
[7]	121
[8]	98

步骤5 在“Variables”窗口中展开“response”变量，查看响应消息的内容。

图 5-20 展开“response”变量

Name	Value
↳ this	HttpsUtil (id=24)
↳ request	HttpPost (id=25)
↳ response	StreamClosedHttpResponse (id=294)
↳ content	{"accessToken":"ff5478725ebaffbeadda5"} ↳ hash ↳ value (id=297)
↳ original	\$Proxy0 (id=45) ↳ h ↳ original
↳ code	200 ↳ entity ↳ headergroup ↳ locale ↳ params ↳ reasonCatalog ↳ reasonPhrase ↳ statusline
↳ BasicManagedEntity (id=56) ↳ HeaderGroup (id=58) ↳ Locale (id=63) ↳ ClientParamsStack (id=66) ↳ EnglishReasonPhraseCatalog (id=68) ↳ "OK" (id=70) ↳ BasicStatusLine (id=78)	

#### 说明

在代码样例中，“Authentication.java”之外的类均会先调用鉴权接口。因此，在对“Authentication.java”之外的类进行单步调试时，需要程序第二次运行到设置断点的位置时，再查看变量内容。

----结束

## 5.3 使用 Postman 测试平台北向接口

在使用本方法前，您需要：

- 获取IoT平台开放给应用的IP和端口（HTTPS协议）。
- 安装并运行Postman。

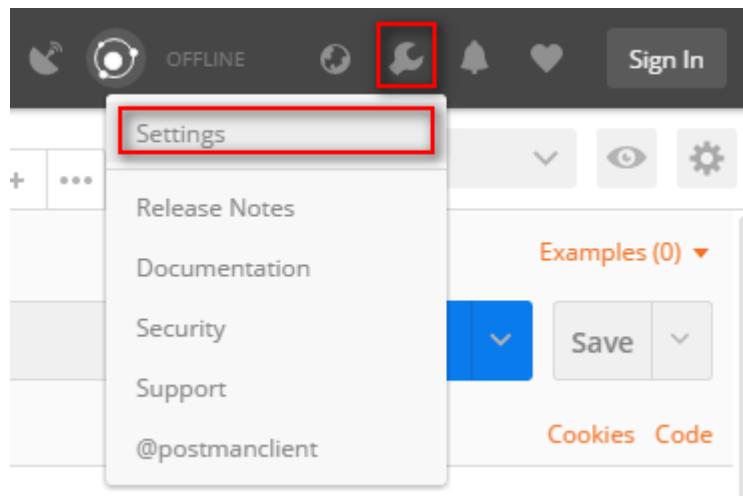
#### 说明

Postman下载链接：<https://www.getpostman.com>

### 配置 Postman

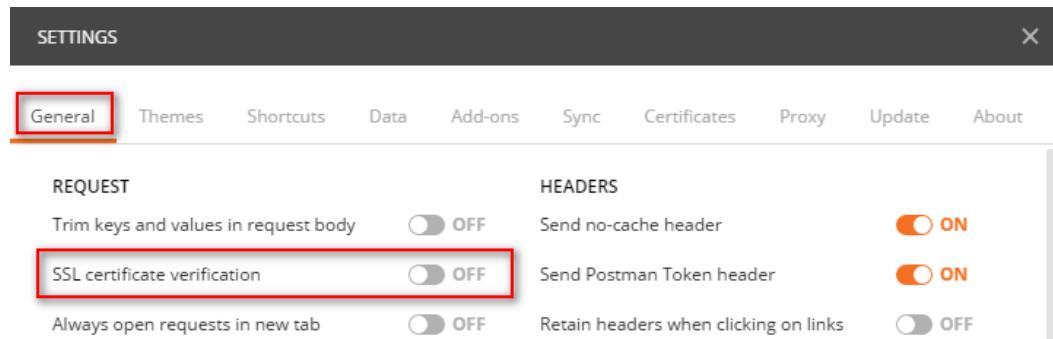
**步骤1** 打开Postman的“Settings”菜单。

图 5-21 进入“Settings”菜单



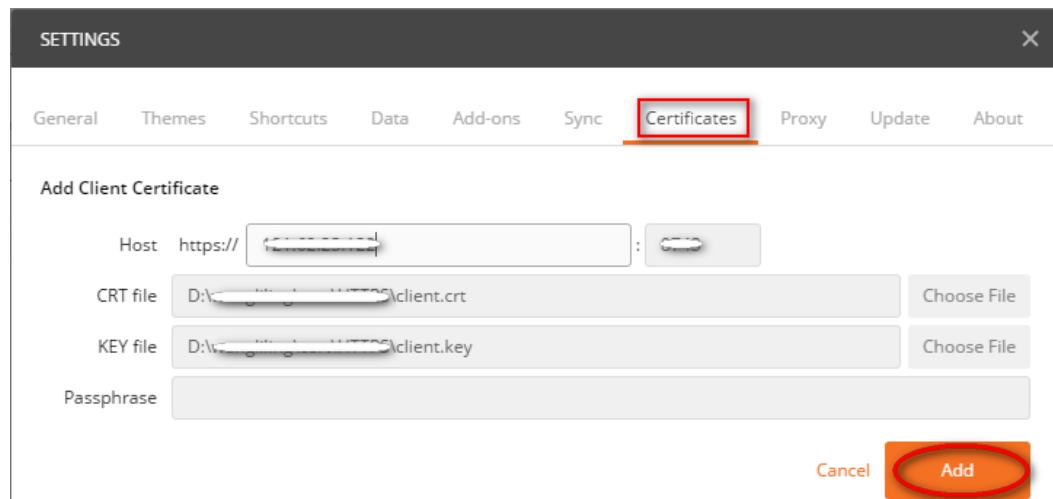
**步骤2** 关闭证书校验，使Postman不再校验服务端的证书。

图 5-22 关闭证书校验



**步骤3** 配置客户端证书，“Host”栏的地址和端口填写物联网平台开放给应用的IP与端口（HTTPS协议）。

图 5-23 配置客户端证书



----结束

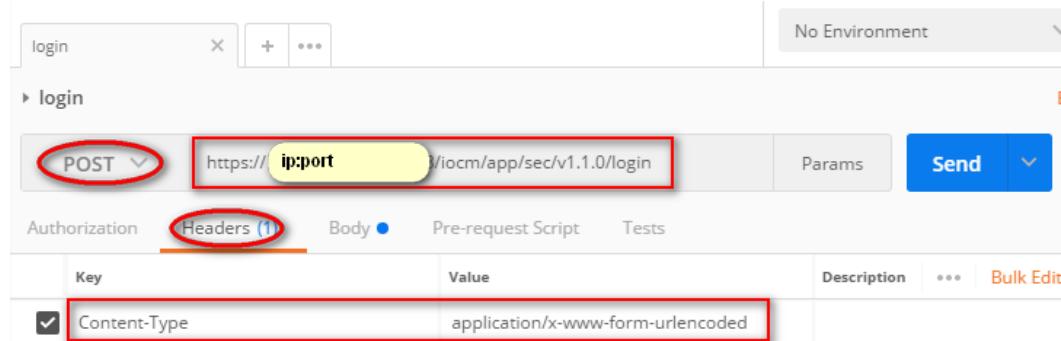
## 说明

client.crt与client.key为客户端证书和私钥文件，获取方法请参考[获取相关证书](#)。

## 调测“鉴权”接口

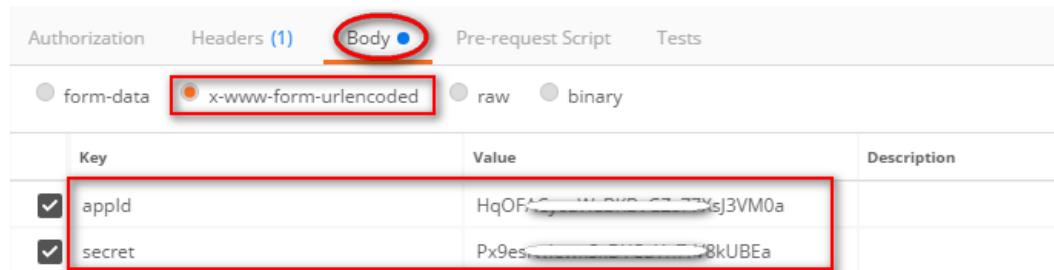
**步骤1** 配置“鉴权”接口的HTTP方法、URL和Headers。

图 5-24 配置 HTTP 方法、URL 和 Headers（鉴权）



**步骤2** 配置“鉴权”接口的Body。

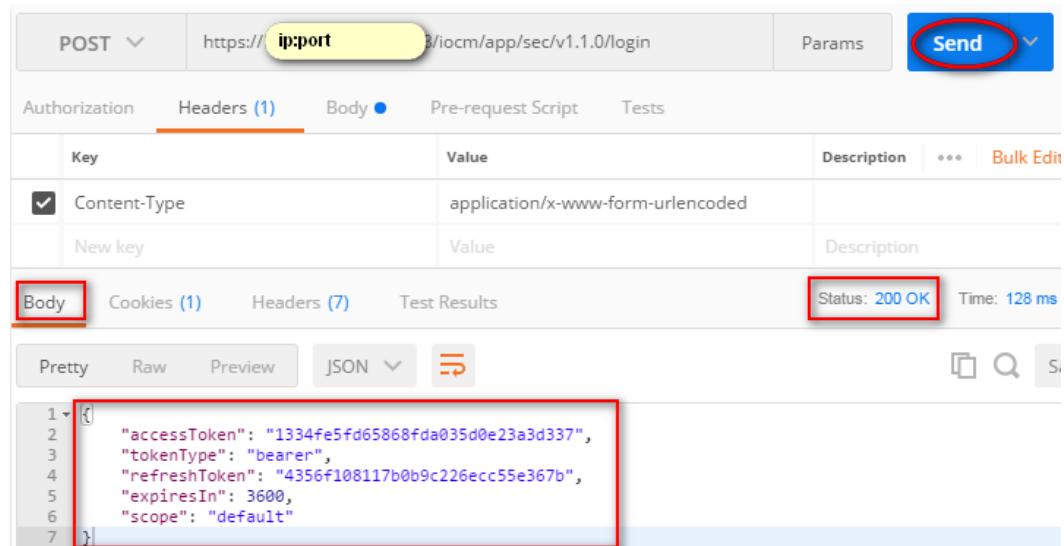
图 5-25 配置 Body（鉴权）



**步骤3** 点击“Send”，在下方查看返回码和响应消息内容。

请将返回的accessToken妥善保存，以便于在调用其它接口时使用。

图 5-26 查看响应信息（鉴权）

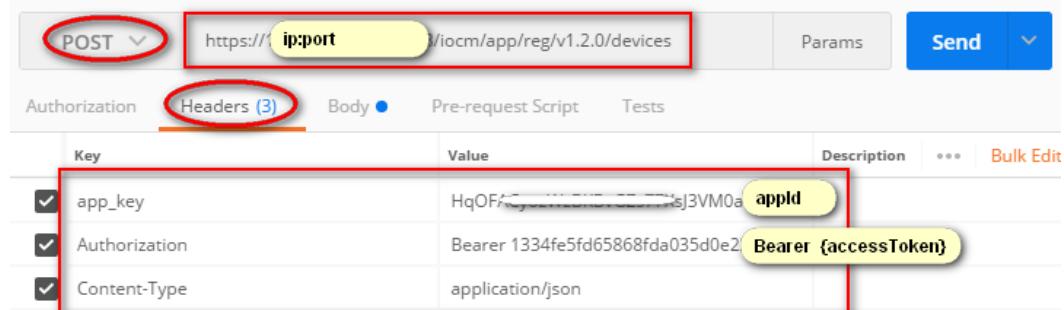


----结束

## 调测“注册直连设备”接口

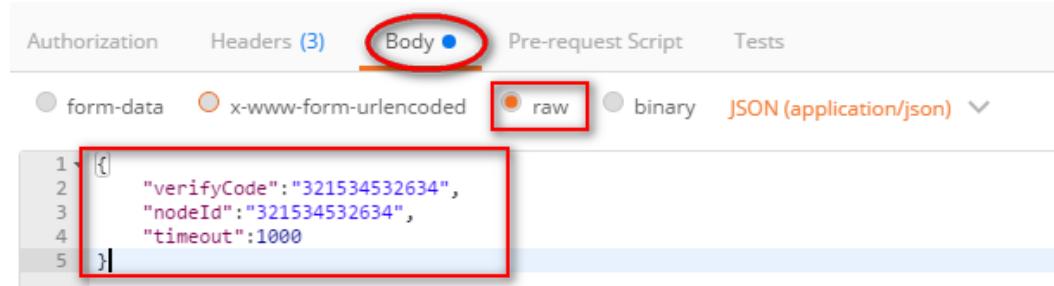
**步骤1** 配置“注册直连设备”接口的HTTP方法、URL和Headers。

图 5-27 配置 HTTP 方法、URL 和 Headers（注册直连设备）



**步骤2** 配置“注册直连设备”接口的Body。

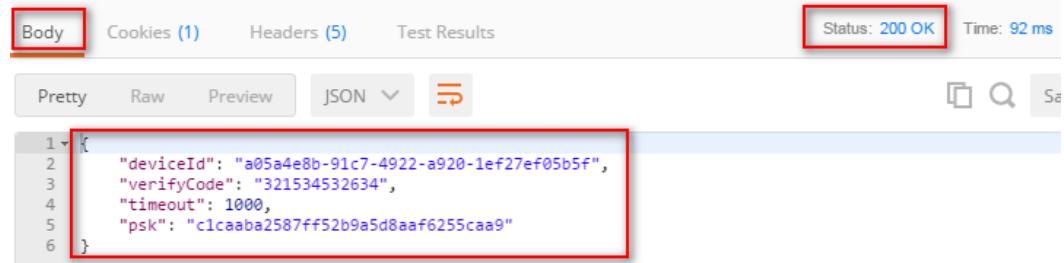
图 5-28 配置 Body（注册直连设备）



**步骤3** 点击“Send”，在下方查看返回码和响应消息内容。

请将返回的设备ID（`deviceId`）妥善保存，以便于在调用其它接口时使用。

图 5-29 查看响应信息（注册直连设备）



```
1 < [  
2   "deviceId": "a05a4e8b-91c7-4922-a920-1ef27ef05b5f",  
3   "verifyCode": "321534532634",  
4   "timeout": 1000,  
5   "psk": "c1caaba2587ff52b9a5d8aaf6255caa9"  
6 ]
```

----结束

## 5.4 Profile 文件

### 5.4.1 概念

设备的Profile文件是用来描述设备类型和设备服务能力的文件。它定义了设备具备的服务能力，每个服务具备的属性、命令以及命令的参数。Profile文件会被上传到物联网平台。

#### ● 设备能力 (Device Capability)

描述一款水表设备的能力特征，包括设备类型、厂商、型号、协议类型以及提供的服务类型。

例如：水表的制造厂商为“HZYB”，制造商ID为“TestUtf8ManuId”，型号为“NB-IoTDevice”，协议类型为“CoAP”。

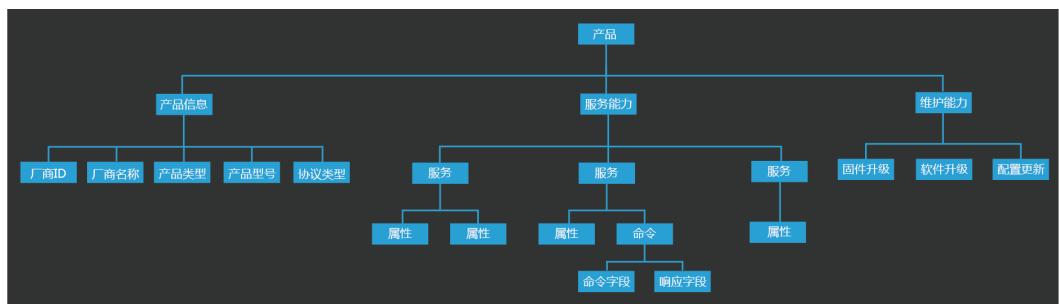
此款水表的服务包括：基础（WaterMeterBasic），告警（WaterMeterAlarm），电池（Battery），传输规则（DeliverySchedule），连接（Connectivity）。其中，电池为可选服务（Optional），其余为必选服务（Mandatory）。

#### ● 服务(Service)

描述设备具备的服务能力，每个服务具备的属性、命令以及命令的参数。

例如：水表基础（WaterMeterBasic），告警（WaterMeterAlarm），电池（Battery），传输规则（DeliverySchedule），连接（Connectivity）五个服务（service），每个服务包含相应的属性或命令。

图 5-30 Profile 文件结构



### 5.4.2 线下开发参考

### 5.4.2.1 设备 Profile 写作

设备的Profile文件为json格式的文件，需要包含如下信息：

设备型号识别属性：设备类型、厂商、型号、协议类型。

服务列表：具体的功能服务说明列表。

## 命名规范

在Profile文件的开发过程中，需要遵循如下命名规范：

- 设备类型（deviceType）、服务类型（serviceType）、服务标识（serviceId）采用单词首字母大写的命名法。例如：WaterMeter、Battery。
- 属性使用第一个单词首字母小写，其余单词的首字母大写的命名法。例如：batteryLevel、internalTemperature。
- 命令使用所有字母大写，单词间用下划线连接的格式。例如：DISCOVERY，CHANGE\_COLOR。
- 设备能力描述json文件固定命名devicetype-capability.json。
- 服务能力描述json文件固定命名servicetype-capability.json。
- 厂商ID/厂商名称和设备型号唯一标识一款设备，故这些信息的组合在不同的Profile文件中不能重复，且仅支持英文。
- 要注重名称的通用性，简洁性；对于服务能力描述，还要考虑其功能性。例如：对于多传感器设备，可以命名为MultiSensor；对于具有显示电量的服务，可以命名为Battery。

#### 说明

在一些Profile样例中，您可能会遇到命名为devicetype-display.json或servicetype-display.json的文件，这些文件是用于智慧家庭领域的一些场景中的，如果物联网平台服务商与您在方案交流中未涉及，则在您的Profile中可以不包含这些文件。

如果您需要制作智慧家庭领域的Profile文件，可以向物联网平台支撑人员咨询。

## 设备 Profile

将一款新设备接入到物联网平台，首先需要编写这款设备的Profile。物联网平台提供了一些Profile文件模板，如果新增接入设备的类型和功能服务已经在物联网平台提供的设备Profile文件模板中包含，则可以直接选择使用；如果在物联网平台提供的设备Profile文件模板中未包含，则需要自己定义。

例如：接入一款水表，可以直接选择物联网平台上对应的Profile文件模板，修改设备型号标识属性和设备服务列表。

#### 说明

物联网平台提供的Profile文件模板会不断更新，如下表格列举设备类型和服务类型示例，仅供参考。

**设备型号识别属性：**

属性	Profile中key	属性值
设备类型	deviceType	WaterMeter
制造商ID	manufacturerId	TestUtf8ManuId

属性	Profile中key	属性值
制造商名称	manufacturerName	HZYB
设备型号	model	NBIoTDevic
协议类型	protocolType	CoAP

### 设备的服务列表

服务描述	服务标识 (serviceId)	服务类型 (serviceType)	选项 (option)
水表的基本功能	WaterMeterBasic	Water	Mandatory
告警服务	WaterMeterAlarm	Battery	Mandatory
电池服务	Battery	Battery	Optional
数据的上报规则	DeliverySchedule	DeliverySchedule	Mandatory
水表的连通性	Connectivity	Connectivity	Mandatory

完整样例参见[附录一 WaterMeter Profile样例](#)。您可以根据需要，对服务的定义进行实例化修改，如：可以调整属性的取值范围、或枚举值等。

#### 说明

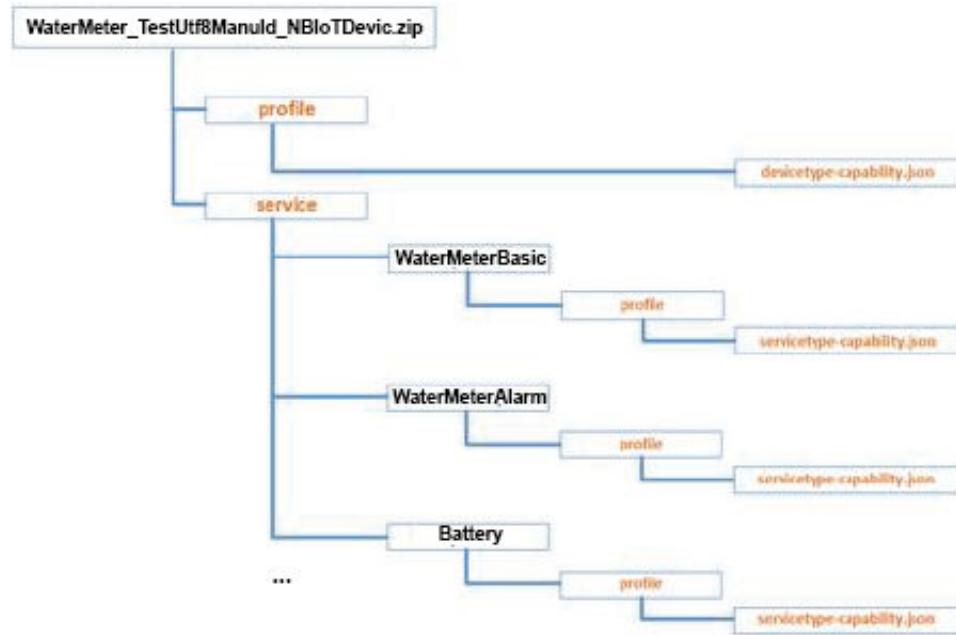
开发者可以通过咨询物联网平台支撑人员，以判断物联网平台是否支持自己的设备类型。如果开发者的设备类型或服务类型已经支持，则开发者可以向物联网平台支撑人员获取设备及服务的Profile文件参考。

设备型号建议由产品类型ID和产品ID组合构成，例如一家厂商水表的ProductTypeId为0x0168，ProductId为0x0188，则设备型号对应为“0168-0188”。

## Profile 打包

Profile写作完成后，需要按如下层级结构打包：

图 5-31 Profile 层级结构



Profile打包需要遵循如下几点要求：

- Profile文件的目录层级结构必须如图5-31所示，不能增删。例如：第二层级只能有“profile”和“service”两个文件夹，每个服务下面必须包含“profile”文件夹等。
- 图5-31中橙色的命名不能改动。
- Profile文件以zip形式压缩。
- Profile文件的命名必须按照deviceType\_manufacturerId\_model的格式命名，其中的deviceType、manufacturerId、model必须与devicetype-capability.json中对应字段的定义一致。例如：本实例中devicetype-capability.json的主要字段如下：

```
{  
    "devices": [  
        {  
            "manufacturerId": "TestUtf8Manuld",  
            "manufacturerName": "HZYB",  
            "model": "NBiotTDevice",  
            "protocolType": "CoAP",  
            "deviceType": "WaterMeter",  
            "serviceTypeCapabilities": "***"  
        }  
    ]  
}
```

- 图5-31中的WaterMeterBasic、WaterMeterAlarm、Battery等都是devicetype-capability.json中定义的服务。
- Profile文件中的文档格式都是json，在编写完成后可以在互联网上查找一些格式校验网站，检查json的合法性。

### 5.4.2.2 设备 Profile 提供形式

设备Profile写作完成后，需要发给IoT平台管理员审核，审核通过后，IoT平台管理员会将Profile导入到IoT实验室。

### 5.4.2.3 设备 Profile 文件字段含义说明

#### 设备能力

devicetype-capability.json记录了该设备的基础信息：

```
{  
    "devices": [  
        {  
            "manufacturerId": "TestUtf8ManuId",  
            "manufacturerName": "HZYB",  
            "model": "NBLoTDevice",  
            "protocolType": "CoAP",  
            "deviceType": "WaterMeter",  
            "omCapability": {  
                "upgradeCapability": {  
                    "supportUpgrade": true,  
                    "upgradeProtocolType": "PCP"  
                },  
                "fwUpgradeCapability": {  
                    "supportUpgrade": true,  
                    "upgradeProtocolType": "LWM2M"  
                },  
                "configCapability": {  
                    "supportConfig": true,  
                    "configMethod": "file",  
                    "defaultConfigFile": {  
                        "waterMeterInfo": {  
                            "waterMeterPirTime": "300"  
                        }  
                    }  
                }  
            },  
            "serviceTypeCapabilities": [  
                {  
                    "serviceId": "WaterMeterBasic",  
                    "serviceType": "WaterMeterBasic",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "WaterMeterAlarm",  
                    "serviceType": "WaterMeterAlarm",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "Battery",  
                    "serviceType": "Battery",  
                    "option": "Optional"  
                },  
                {  
                    "serviceId": "DeliverySchedule",  
                    "serviceType": "DeliverySchedule",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "Connectivity",  
                    "serviceType": "Connectivity",  
                    "option": "Mandatory"  
                }  
            ]  
        }  
    ]  
}
```

各字段的解释：

字段	子字段		可选/必选	描述
devices			必选	包含了一个设备的完整能力信息（根节点不能修改）。
	manufacturerId		必选	指示设备的制造商ID。
	manufacturerName		必选	指示设备的制造商名称（只允许英文）。
	model		必选	指示设备的型号，考虑到一款设备下的多种型号，建议包含字母或数字以保证可扩展性。
	protocolType		必选	指示设备接入物联网平台的协议类型。如NB-IoT的设备取值为CoAP。
	deviceType		必选	指示设备的类型。
	omCapability		可选	定义设备的软件升级、固件升级和配置更新的能力，字段含义详情见下文中的：omCapability结构描述。 如果设备不涉及软件/固件升级，本字段可以删除。
	serviceType Capabilities		必选	包含了设备具备的服务能力描述。
		serviceId	必选	服务的Id，如果设备中同类型的服务类型只有一个则serviceId与serviceType相同，如果有多个则增加编号，如三键开关 Switch01、Switch02、Switch03。
		serviceType	必选	服务类型，与servicetype-capability.json 中serviceType字段保持一致。
		option	必选	标识服务字段的类型，取值范围：Master（主服务），Mandatory（必选服务），Optional（可选服务）。 目前本字段为非功能性字段，仅起到描述作用。

## omCapability结构描述

字段	子字段	可选/必选	描述
upgradeCapability		可选	设备软件升级能力。

字段	子字段	可选/ 必选	描述
	supportUpgrade	可选	true: 设备支持软件升级。 false: 设备不支持软件升级。
	upgradeProtocolType	可选	升级使用的协议类型，此处不同于设备的 protocolType，例如CoAP设备软件升级协议使用PCP。
fwUpgradeCapability		可选	设备固件升级能力。
	supportUpgrade	可选	true: 设备支持固件升级。 false: 设备不支持固件升级。
	upgradeProtocolType	可选	升级使用的协议类型，此处不同于设备的 protocolType，当前物联网平台仅支持LWM2M 固件升级。
configCapability		可选	设备配置更新能力。
	supportConfig	可选	true: 设备支持配置更新。 false: 设备不支持配置更新。
	configMethod	可选	file: 使用文件的方式下发配置更新。
	defaultConfigFile	可选	设备默认配置信息（Json格式），具体配置信息由设备商自定义。物联网平台只储存该信息供下发时使用，不解析处理配置字段的具体含义。

## 服务能力

servicetype-capability.json记录了该设备的服务信息：

```
{  
    "services": [  
        {  
            "serviceType": "WaterMeterBasic",  
            "description": "WaterMeterBasic",  
            "commands": [  
                {  
                    "commandName": "SET_PRESSURE_READ_PERIOD",  
                    "paras": [  
                        {  
                            "paraName": "value",  
                            "dataType": "int",  
                            "required": true,  
                            "min": 1,  
                            "max": 24,  
                            "step": 1,  
                            "maxLength": 10,  
                            "unit": "hour",  
                            "enumList": null  
                        }  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

```
        ],
        "responses": [
            {
                "responseName": "SET_PRESSURE_READ_PERIOD_RSP",
                "paras": [
                    {
                        "paraName": "result",
                        "dataType": "int",
                        "required": true,
                        "min": -1000000,
                        "max": 1000000,
                        "step": 1,
                        "maxLength": 10,
                        "unit": null,
                        "enumList": null
                    }
                ]
            }
        ],
        "properties": [
            {
                "propertyName": "registerFlow",
                "dataType": "int",
                "required": true,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "R",
                "unit": null,
                "enumList": null
            },
            {
                "propertyName": "currentReading",
                "dataType": "string",
                "required": false,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "M",
                "unit": "L",
                "enumList": null
            },
            {
                "propertyName": "timeOfReading",
                "dataType": "string",
                "required": false,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "M",
                "unit": null,
                "enumList": null
            },
            {
                "propertyName": "internalTemperature",
                "dataType": "int",
                "required": false,
                "min": 0,
                "max": 0,
                "step": 1,
                "maxLength": 0,
                "method": "M",
                "unit": "0.01° C",
                "enumList": null
            }
        ]
    }
}
```

```
        },
        {
            "propertyName": "dailyFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "M",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "dailyReverseFlow",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "M",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "peakFlowRate",
            "dataType": "int",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "M",
            "unit": "L/H",
            "enumList": null
        },
        {
            "propertyName": "peakFlowRateTime",
            "dataType": "string",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "M",
            "unit": null,
            "enumList": null
        },
        {
            "propertyName": "intervalFlow",
            "dataType": "array",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
            "method": "M",
            "unit": "L",
            "enumList": null
        },
        {
            "propertyName": "pressure",
            "dataType": "array",
            "required": false,
            "min": 0,
            "max": 0,
            "step": 1,
            "maxLength": 0,
```

```
        "method": "0",
        "unit": "kPa",
        "enumList": null
    },
    {
        "propertyName": "temperature",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "0.01° C",
        "enumList": null
    },
    {
        "propertyName": "vibration",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "0.01g",
        "enumList": null
    }
]
}
]
```

各字段的解释：

字段	子字段			必选/ 可选	描述
services				必选	包含了一个服务的完整信息（根节点不可修改）。
serviceType				必选	指示服务的类型，与devicetype-capability.json中serviceType字段保持一致。
description				必选	指示服务的描述信息。 非功能性字段，仅起到描述作用，可置为null。
commands				必选	指示设备可以执行的命令，如果本服务无命令则置null。
	commandName			必选	指示命令的名字，命令名与参数共同构成一个完整的命令。
	paras			必选	命令包含的参数。
		paraName		必选	命令中参数的名字。

字段	子字段			必选/ 可选	描述
		dataType		必选	<p>指示命令参数的数据类型。</p> <p>取值范围: string、int、string list、decimal、DateTime、jsonObject</p> <p>上报数据时，复杂类型数据格式如下：</p> <ul style="list-style-type: none"><li>● string list: ["str1","str2","str3"]</li><li>● DateTime: yyyyMMdd'T'HHmmss'Z' 如:20151212T121212Z</li><li>● jsonObject: 自定义json结构体，物联网平台不解析，仅进行透传</li></ul>
		required		必选	<p>指示本命令是否必选，取值为true或false，默认取值false（非必选）。</p> <p>目前本字段是非功能性字段，仅起到描述作用。</p>
		min		必选	<p>指示最小值。</p> <p>仅当dataType为int、decimal时生效。</p>
		max		必选	<p>指示最大值。</p> <p>仅当dataType为int、decimal时生效。</p>
		step		必选	<p>指示步长。</p> <p>暂不使用，填0即可。</p>
		maxLength		必选	<p>指示字符串长度。</p> <p>仅当dataType为string、string list、DateTime时生效。</p>
		unit		必选	<p>指示单位，英文。</p> <p>取值根据参数确定，如：</p> <p>温度单位：“C”或“K”</p> <p>百分比单位：“%”</p> <p>压强单位：“Pa”或“kPa”</p>
		enumList		必选	<p>指示枚举值。</p> <p>如开关状态status可有如下取值：</p> <p>"enumList" : ["OPEN","CLOSE"]</p> <p>目前本字段是非功能性字段，仅起到描述作用，建议准确定义。</p>
		responses		必选	命令执行的响应。
		responseName		必选	命名可以在该responses对应命令的commandName后面添加“_rsp”。

字段	子字段			必选/ 可选	描述
		paras		必选	命令响应的参数。
			paraName	必选	命令中参数的名字。
			dataType	必选	<p>指示数据类型。 取值范围: string、 int、 string list、 decimal、 DateTime、 jsonObject 上报数据时，复杂类型数据格式如下：</p> <ul style="list-style-type: none"><li>● string list: ["str1","str2","str3"]</li><li>● DateTime: yyyyMMdd' T' HHmmss' Z' 如:20151212T121212Z</li><li>● jsonObject: 自定义json结构体，物联网平台不解析，仅进行透传</li></ul>
			required	必选	<p>指示本命令响应是否必选，取值为true或false， 默认取值false（非必选）。</p> <p>目前本字段是非功能性字段，仅起到描述作用。</p>
			min	必选	<p>指示最小值。</p> <p>仅当dataType为int、 decimal时生效，逻辑大于等于。</p>
			max	必选	<p>指示最大值。</p> <p>仅当dataType为int、 decimal时生效，逻辑小于等于。</p>
			step	必选	<p>指示步长。</p> <p>暂不使用，填0即可。</p>
			maxLength	必选	<p>指示字符串长度。</p> <p>仅当dataType为string、 string list、 DateTime时生效。</p>
			unit	必选	<p>指示单位，英文。</p> <p>取值根据参数确定，如： 温度单位：“C”或“K” 百分比单位：“%” 压强单位：“Pa”或“kPa”</p>

字段	子字段			必选/ 可选	描述
			en u m Li st	必选	指示枚举值。 如开关状态status可有如下取值： "enumList" : ["OPEN","CLOSE"] 目前本字段是非功能性字段，仅起到 描述作用，建议准确定义。
	pro pert ies			必选	上报数据描述，每一个子节点为一条 属性。
		property Name		必选	指示属性名称。
		dataTyp e		必选	指示数据类型。 取值范围：string、int、string list、 decimal、DateTime、jsonObject 上报数据时，复杂类型数据格式如 下： <ul style="list-style-type: none"><li>● string list: ["str1","str2","str3"]</li><li>● DateTime: yyyyMMdd' T' HHmmss' Z' 如:20151212T121212Z</li><li>● jsonObject: 自定义json结构体，物 联网平台不解析，仅进行透传</li></ul>
		required		必选	指示本条属性是否必选，取值为true 或false，默认取值false（非必选）。 目前本字段是非功能性字段，仅起到 描述作用。
		min		必选	指示最小值。 仅当dataType为int、decimal时生效， 逻辑大于等于。
		max		必选	指示最大值。 仅当dataType为int、decimal时生效， 逻辑小于等于。
		step		必选	指示步长。 暂不使用，填0即可。
		method		必选	指示访问模式。 R:可读； W:可写； E可订阅 取值范围： R、 RW、 RE、 RWE 、 null

字段	子字段	必选/可选	描述
	unit	必选	指示单位，英文。 取值根据参数确定，如： 温度单位：“C”或“K” 百分比单位：“%” 压强单位：“Pa”或“kPa”
	maxLength	必选	指示字符串长度。 仅当dataType为string、string list、 DateTime时生效。
	enumList	必选	指示枚举值。 如电池状态（batteryStatus）可有如下 取值： "enumList" : [0, 1, 2, 3, 4, 5, 6] 目前本字段是非功能性字段，仅起到 描述作用，建议准确定义。

## 5.4.3 附录

### 附录一 WaterMeter Profile 样例

样例由六个文件构成，文件名和文件内容如下。

#### 1. devicetype-capability.json

```
{  
    "devices": [  
        {  
            "manufacturerId": "TestUtf8ManuId",  
            "manufacturerName": "HZYB",  
            "model": "NB IoT Device",  
            "protocolType": "CoAP",  
            "deviceType": "WaterMeter",  
            "serviceTypeCapabilities": [  
                {  
                    "serviceId": "WaterMeterBasic",  
                    "serviceType": "WaterMeterBasic",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "WaterMeterAlarm",  
                    "serviceType": "WaterMeterAlarm",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "Battery",  
                    "serviceType": "Battery",  
                    "option": "Optional"  
                },  
                {  
                    "serviceId": "DeliverySchedule",  
                    "serviceType": "DeliverySchedule",  
                    "option": "Mandatory"  
                },  
                {  
                    "serviceId": "DataLog",  
                    "serviceType": "DataLog",  
                    "option": "Optional"  
                }  
            ]  
        }  
    ]  
}
```

```
{  
    "services": [  
        {  
            "serviceId": "Connectivity",  
            "serviceType": "Connectivity",  
            "option": "Mandatory"  
        }  
    ]  
}
```

2. **servicetype-capability.json (Battery)**

```
{  
    "services": [  
        {  
            "serviceType": "Battery",  
            "description": "Battery",  
            "commands": null,  
            "properties": [  
                {  
                    "propertyName": "batteryLevel",  
                    "dataType": "int",  
                    "required": true,  
                    "min": 0,  
                    "max": 100,  
                    "step": 1,  
                    "maxLength": 0,  
                    "method": "RE",  
                    "unit": "%",  
                    "enumList": null  
                },  
                {  
                    "propertyName": "batteryThreshold",  
                    "dataType": "int",  
                    "required": false,  
                    "min": 0,  
                    "max": 100,  
                    "step": 1,  
                    "maxLength": 0,  
                    "method": "RE",  
                    "unit": "%",  
                    "enumList": null  
                },  
                {  
                    "propertyName": "batteryStatus",  
                    "dataType": "int",  
                    "required": false,  
                    "min": 0,  
                    "max": 0,  
                    "step": 1,  
                    "maxLength": 0,  
                    "method": "RE",  
                    "unit": null,  
                    "enumList": [  
                        0,  
                        1,  
                        2,  
                        3,  
                        4,  
                        5,  
                        6  
                    ]  
                }  
            ]  
        }  
    ]  
}
```

3. **servicetype-capability.json (ConnectivityMonitoring)**

```
{  
    "services": [  
        {  
            "serviceId": "Connectivity",  
            "serviceType": "Connectivity",  
            "option": "Mandatory"  
        }  
    ]  
}
```

```
{  
    "serviceType": "Connectivity",  
    "description": "Connectivity",  
    "commands": null,  
    "properties": [  
        {  
            "propertyName": "signalStrength",  
            "dataType": "int",  
            "required": true,  
            "min": -110,  
            "max": -48,  
            "step": 1,  
            "maxLength": 0,  
            "method": "RE",  
            "unit": "dbm",  
            "enumList": null  
        },  
        {  
            "propertyName": "linkQuality",  
            "dataType": "int",  
            "required": false,  
            "min": -110,  
            "max": -48,  
            "step": 1,  
            "maxLength": 0,  
            "method": "RE",  
            "unit": "dbm",  
            "enumList": null  
        },  
        {  
            "propertyName": "cellId",  
            "dataType": "int",  
            "required": false,  
            "min": 0,  
            "max": 268435455,  
            "step": 1,  
            "maxLength": 0,  
            "method": "RE",  
            "unit": null,  
            "enumList": null  
        }  
    ]  
}
```

#### 4. servicetype-capability.json (DeliverySchedule)

```
{  
    "services": [  
        {  
            "serviceType": "DeliverySchedule",  
            "description": "DeliverySchedule",  
            "commands": null,  
            "properties": [  
                {  
                    "propertyName": "startTime",  
                    "dataType": "int",  
                    "required": true,  
                    "min": 0,  
                    "max": 0,  
                    "step": 1,  
                    "maxLength": 0,  
                    "method": "RW",  
                    "unit": "sec",  
                    "enumList": null  
                },  
                {  
                    "propertyName": "UTCOffset",  
                    "dataType": "string",  
                    "required": true,  
                    "maxLength": 0,  
                    "method": "RW",  
                    "unit": "sec",  
                    "enumList": null  
                }  
            ]  
        }  
    ]  
}
```

```
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RW",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "frequency",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RW",
        "unit": "sec",
        "enumList": null
    },
    {
        "propertyName": "randomisedDeliveryWindow",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RW",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "retries",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RW",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "retryPeriod",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RW",
        "unit": null,
        "enumList": null
    }
]
}
]
```

## 5. servicetype-capability.json (WaterMeterAlarm)

```
{
    "services": [
        {
            "serviceType": "WaterMeterAlarm",
            "description": "WaterMeterAlarm",
            "commands": null,
            "properties": [
                {

```

```
        "propertyName": "lowFlowAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "highFlowAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "tamperAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "lowBatteryAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "batteryRunOutAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "highInternalTemperature",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    }
]
```

```
        "enumList": null
    },
    {
        "propertyName": "reverseFlowAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "highPressureAlarm",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "lowPressureAlarm",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "highTemperatureAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "lowTemperatureAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "innerErrorAlarm",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
```

```
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "storageFault",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "waterTempratureSensorFault",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "innerTempratureSensorFault",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "pressureSensorFault",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "vibrationSensorFault",
        "dataType": "int",
        "required": true,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "strayCurrent",
        "dataType": "int",
        "required": true,
```

```
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "RE",
        "unit": null,
        "enumList": null
    }
]
}
]
```

## 6. servicetype-capability.json (WaterMeterBasic)

```
{
    "services": [
        {
            "serviceType": "WaterMeterBasic",
            "description": "WaterMeterBasic",
            "commands": null,
            "properties": [
                {
                    "propertyName": "registerFlow",
                    "dataType": "int",
                    "required": true,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "R",
                    "unit": null,
                    "enumList": null
                },
                {
                    "propertyName": "currentReading",
                    "dataType": "string",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "M",
                    "unit": "L",
                    "enumList": null
                },
                {
                    "propertyName": "timeOfReading",
                    "dataType": "string",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "M",
                    "unit": null,
                    "enumList": null
                },
                {
                    "propertyName": "internalTemperature",
                    "dataType": "int",
                    "required": false,
                    "min": 0,
                    "max": 0,
                    "step": 1,
                    "maxLength": 0,
                    "method": "M",
                    "unit": "0.01° C",
                    "enumList": null
                },
                {

```

```
        "propertyName": "dailyFlow",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "L",
        "enumList": null
    },
    {
        "propertyName": "dailyReverseFlow",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "L",
        "enumList": null
    },
    {
        "propertyName": "peakFlowRate",
        "dataType": "int",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "L/H",
        "enumList": null
    },
    {
        "propertyName": "peakFlowRateTime",
        "dataType": "string",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": null,
        "enumList": null
    },
    {
        "propertyName": "intervalFlow",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "L",
        "enumList": null
    },
    {
        "propertyName": "pressure",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "O",
        "unit": "kPa",
        "enumList": null
    }
]
```

```
        "enumList": null
    },
    {
        "propertyName": "temperature",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "0.01° C",
        "enumList": null
    },
    {
        "propertyName": "vibration",
        "dataType": "array",
        "required": false,
        "min": 0,
        "max": 0,
        "step": 1,
        "maxLength": 0,
        "method": "M",
        "unit": "0.01g",
        "enumList": null
    }
]
}
]
```

## 5.5 编解码插件

### 5.5.1 整体方案

设备上报数据时，如果“数据格式”为“二进制码流”，则该产品下需要进行编解码插件开发；如果“数据格式”为“JSON”，则该产品下不需要进行编解码插件开发。

以NB-IoT场景为例，NB-IoT设备和物联网平台之间采用CoAP协议通讯，CoAP消息的payload为应用层数据，应用层数据的格式由设备自行定义。由于NB-IoT设备一般对省电要求较高，所以应用层数据一般不采用流行的JSON格式，而是采用二进制格式。但是，物联网平台与应用侧使用json格式进行通信。因此，开发者需要开发编码插件，供物联网平台调用，以完成二进制格式和JSON格式的转换。

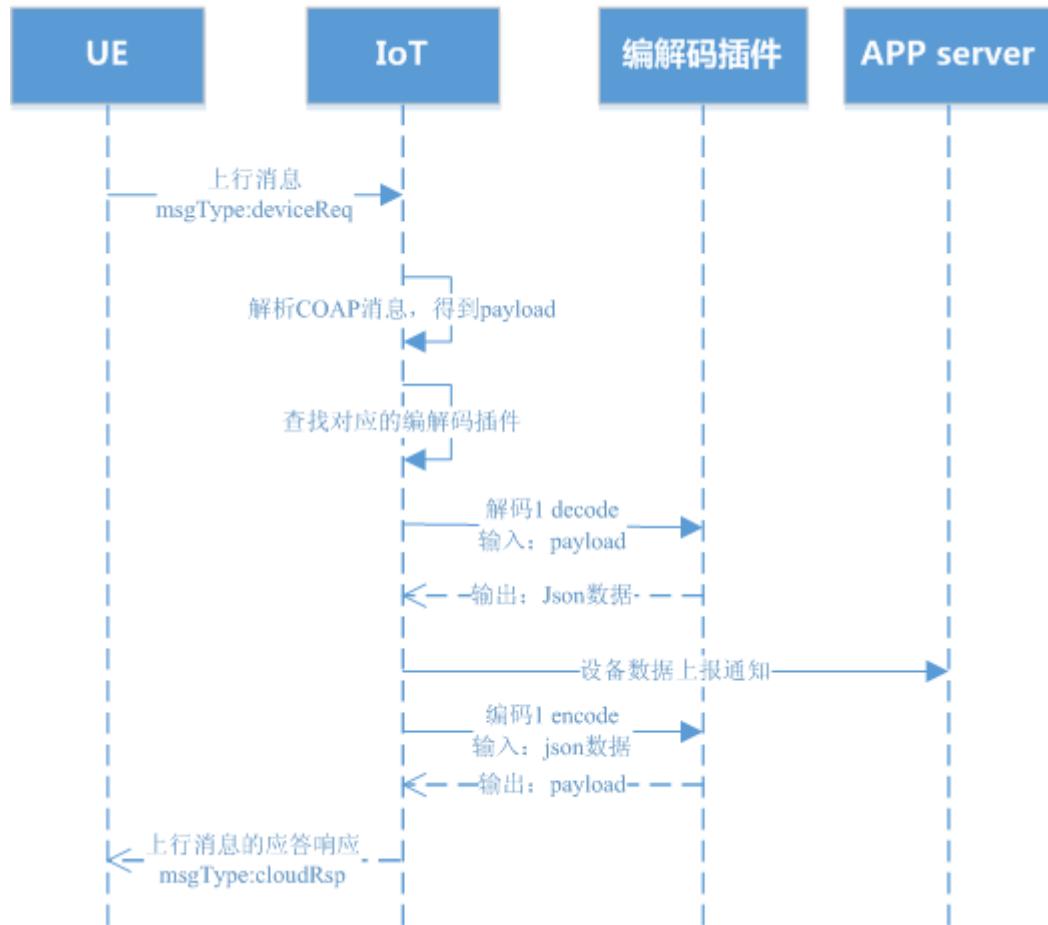
图 5-32 整体方案



## 5.5.2 消息处理流程

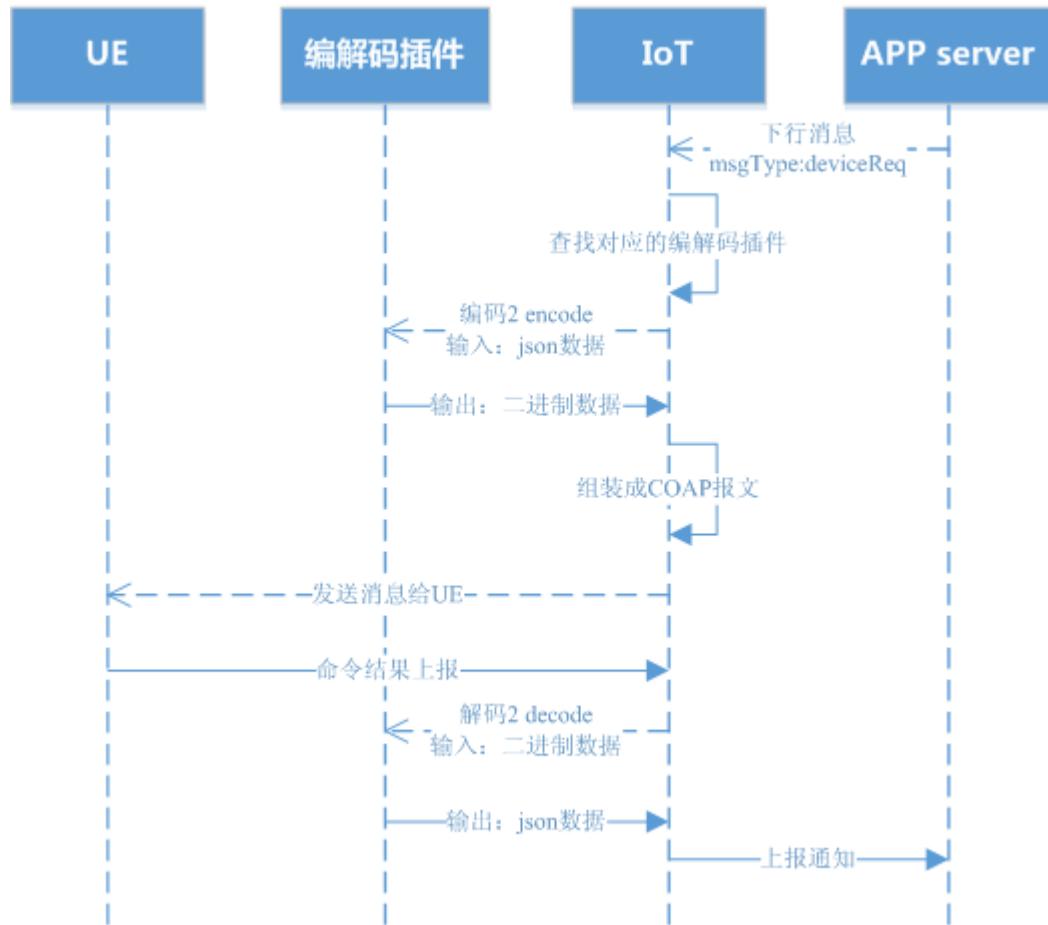
### 数据上报

图 5-33 数据上报处理流程



## 命令下发处理

图 5-34 命令下发处理流程



## 5.5.3 线下开发参考

### 5.5.3.1 开发环境准备

#### 下载 eclipse

下载Eclipse安装包，直接解压缩到本地即可使用。

官网下载地址：<http://www.eclipse.org/downloads>。

#### 下载 maven 插件

下载Maven插件包（zip格式），直接解压缩到本地。

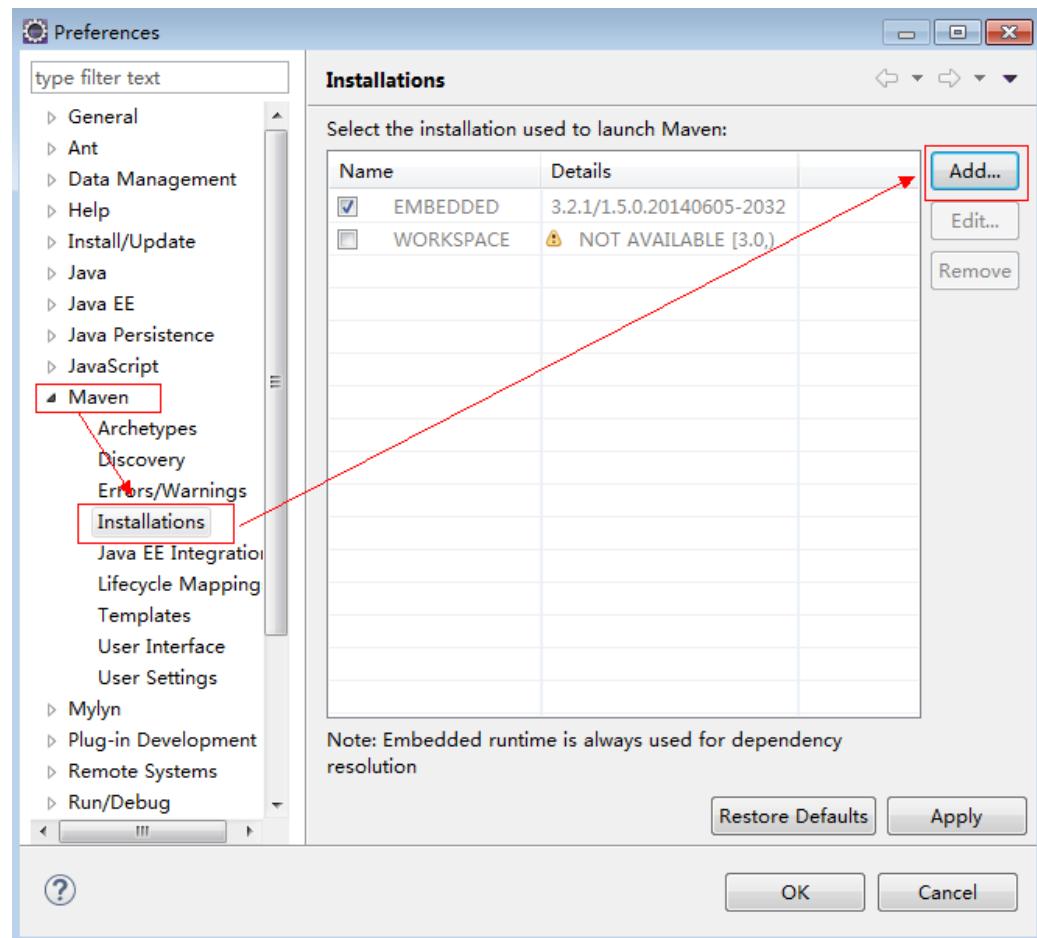
官网下载地址：<http://maven.apache.org/download.cgi>

#### 配置 Maven 插件

Maven的配置涉及Windows环境变量的配置与在Eclipse中的配置，环境变量的配置请参考网上资源，本节仅介绍Maven在Eclipse中的配置。

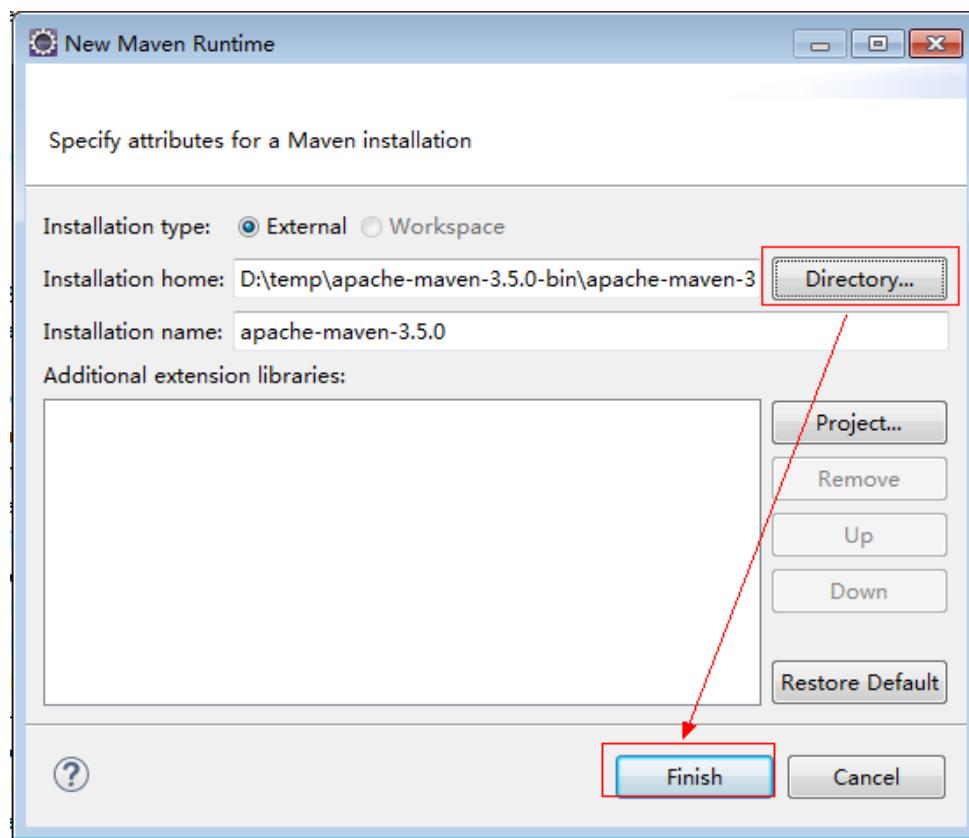
**步骤1** 选择Eclipse菜单“Windows”->“Preferences”，打开Preferences窗口，选择“Maven”->“Installations”->“Add”。

图 5-35 配置 Maven 插件 1



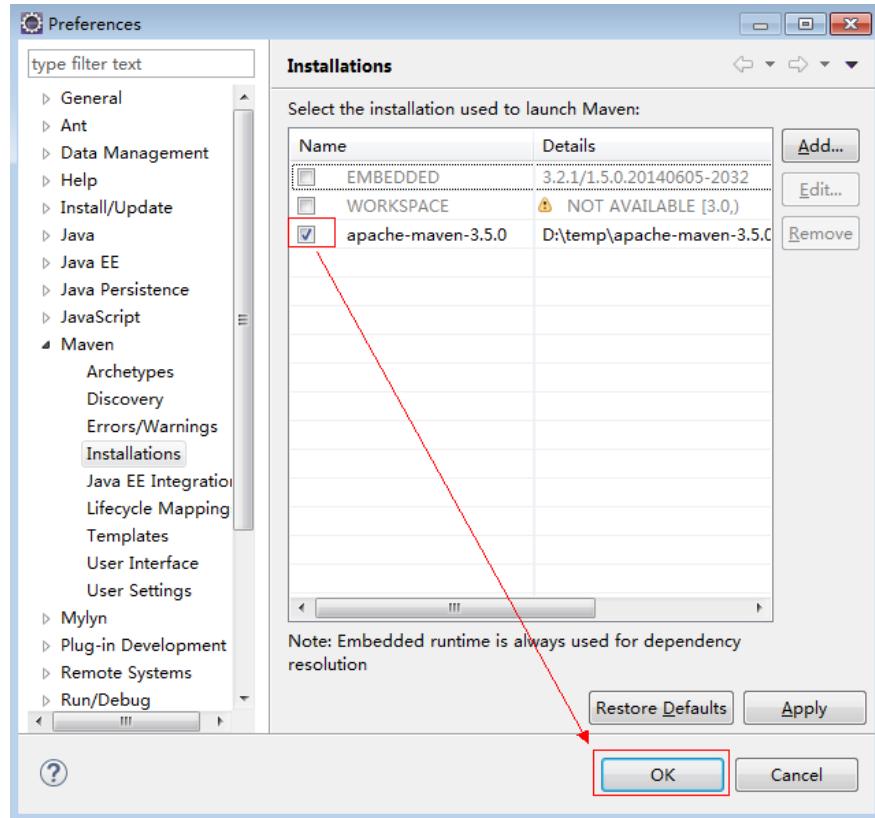
**步骤2** 选择maven插件包路径，点击“Finish”，导入Maven插件。

图 5-36 配置 Maven 插件 2



步骤3 选择导入的maven插件，点击“OK”。

图 5-37 配置 Maven 插件 3



#### 说明

JDK的安装和Java环境变量的配置请参见[安装JDK1.8和配置Java环境变量（Windows操作系统）](#)。

----结束

### 5.5.3.2 开发编解码插件

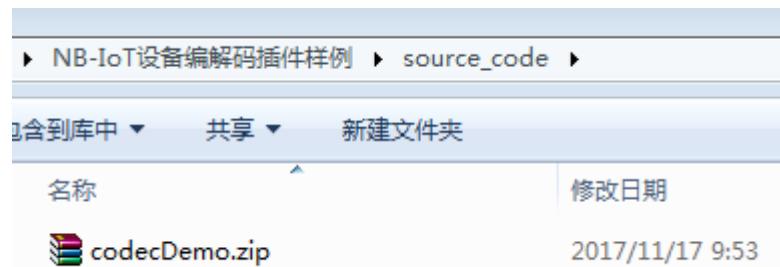
编解码插件实现二进制消息转json格式的功能，Profile文件定义了该json格式的具体内容。因此，编解码插件开发前需要先编写设备的Profile文件，Profile开发指导详见[Profile文件](#)。

为了提高集成效率，我们提供了编解码插件的DEMO工程（[点击获取](#)），建议开发者基于DEMO工程进行二次开发。

#### 5.5.3.2.1 导入编解码插件 DEMO 工程

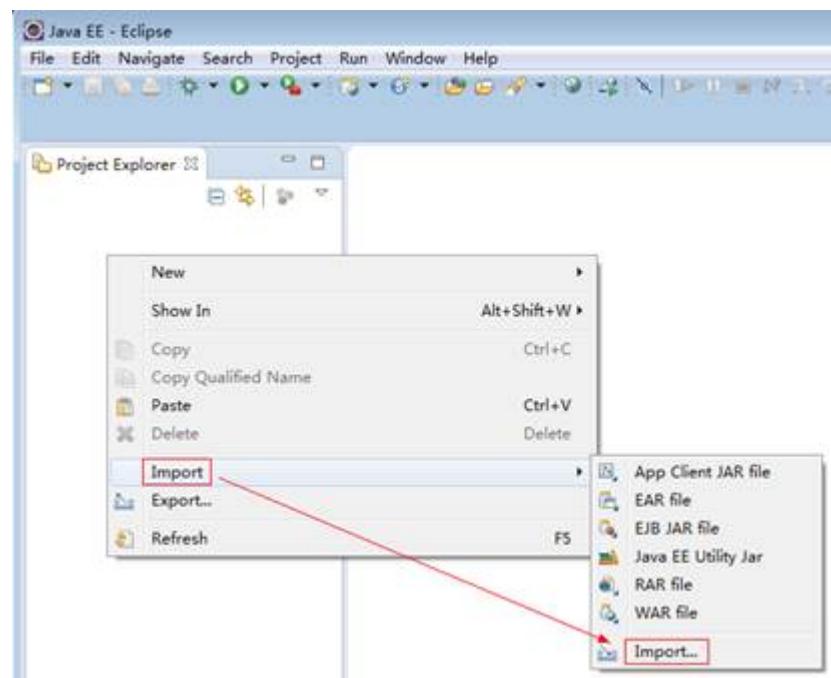
**步骤1** 下载编解码插件DEMO工程，在“source\_code”文件夹中获取“codecDemo.zip”，将其解压到本地。

图 5-38 编解码插件 DEMO 的位置



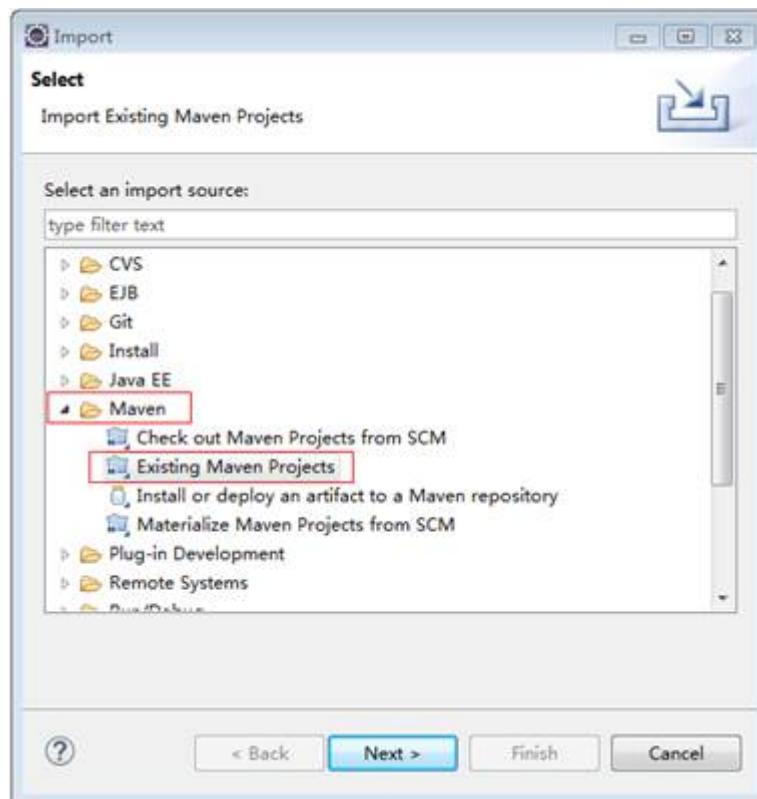
步骤2 打开Eclipse，右击Eclipse左侧“Project Explorer”空白处，选择“Import > Import...”。

图 5-39 导入 DEMO 工程 1



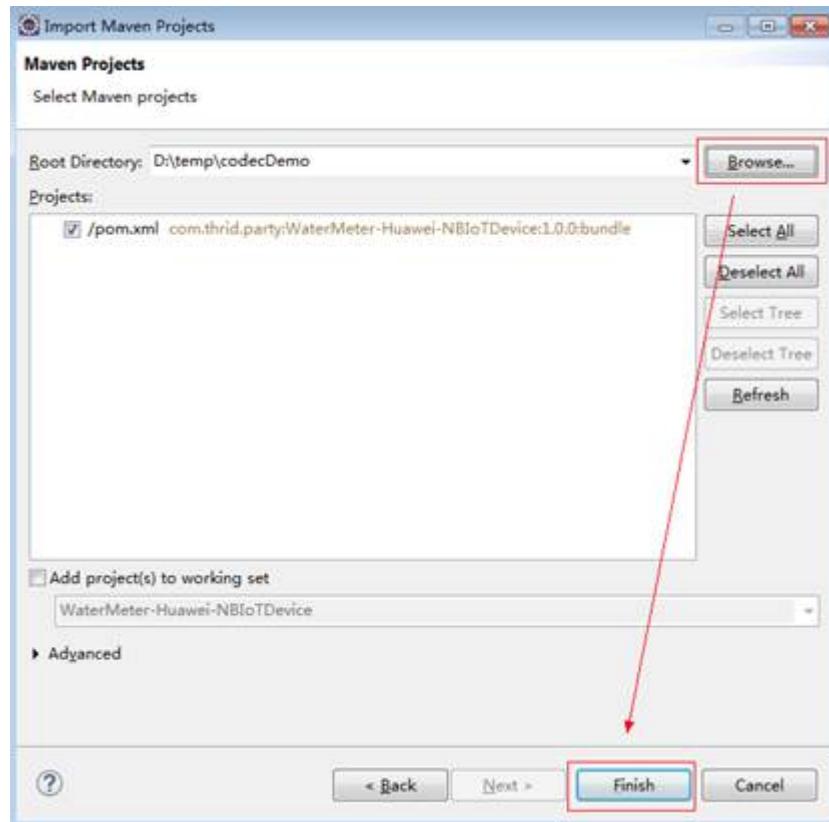
步骤3 展开“Maven”，选择“Existing Maven Projects”，点击“Next”。

图 5-40 导入 DEMO 工程 2



**步骤4** 点击“Browse”，选择**步骤1**解压获得的“codecDemo”文件夹，勾选“/pom.xml”，点击“Finish”。

图 5-41 导入 DEMO 工程 3



----结束

### 5.5.3.2.2 开发插件

编解码插件DEMO的Maven工程架构不需要修改，依据[附录：插件开发说明](#)，对DEMO进行修改。

### 5.5.3.2.3 编解码插件打包

本节介绍编解码完成后如何打包和制作插件包。

#### 编解码 Maven 打包

完成插件编程后使用Maven进行打包，Windows中步骤如下：

- 步骤1 打开DOS窗口，进入“pom.xml”所在的目录。
- 步骤2 输入maven打包命令：mvn package。
- 步骤3 DOS窗口中显示“BUILD SUCCESS”后，打开与“pom.xml”目录同级的target文件夹，获取打包好的jar包。

jar包命名规范为：设备类型-厂商ID-设备型号-版本.jar，例如：WaterMeter-Huawei-NBIoTDevice-version.jar。

图 5-42 Jar 包结构图

名称	大小
..(上层目录)	7.66 KB
com	1 KB
META-INF	1 KB
OSGI-INF	1 KB
json-lib-2.4-jdk15.jar	155.39 KB

- com目录存放的是class文件。
- META-INF下存放的是OSGI框架下的jar的描述文件（根据pom.xml配置生成的）。
- OSGI-INF下存放的是服务配置文件，把编解码注册为服务，供平台调用（只能有一个xml文件）。
- 其他jar是编解码引用到的jar包。

----结束

## 制作插件包

**步骤1** 新建文件夹命名为“package”，包含一个“preload/”子文件夹。

**步骤2** 将打包好的jar包放到“preload/”文件夹。

图 5-43 插件包结构图

名称	修改日期	类型
preload	2017/2/13 15:31	文件夹
package-info.json	2017/2/13 15:31	JSON 文件
WaterMeter-Huawei-NBIoTDevice-1.0.0.jar		

**步骤3** 在“package”文件夹中，新建“package-info.json”文件。该文件的字段说明和模板如下：

### 说明

“package-info.json”需要以UTF-8无BOM格式编码。仅支持英文字符。

表 5-1 “package-info.json” 字段说明

字段名	字段描述	是否必填
specVersion	描述文件版本号，填写固定值：“1.0”。	是
fileName	软件包文件名，填写固定值：“codec-demo”	是
version	软件包版本号。描述package.zip的版本，请与下面的bundleVersion取值保持一致。	是

字段名	字段描述	是否必填
deviceType	设备类型，与Profile文件中的定义保持一致。	是
manufacturerName	制造商名称，与Profile文件中的定义保持一致，否则无法上传到平台。	是
model	产品型号，与Profile文件中的定义保持一致。	是
platform	平台类型，本插件包运行的IoT平台的操作系统，填写固定值："linux"。	是
packageType	软件包类型，该字段用来描述本插件最终部署的平台模块，填写固定值："CIGPlugin"。	是
date	出包时间，格式为："yyyy-MM-dd HH-mm-ss"，如"2017-05-06 20:48:59"。	否
description	对软件包的自定义描述。	否
ignoreList	忽略列表，默认为空值。	是
bundles	一组bundle的描述信息。 <b>说明</b> bundle就是压缩包中的jar包，只需要写一个bundle。	是

表 5-2 bundles 的字段说明

字段名	字段描述	是否必填
bundleName	插件名称，和上文中pom.xml的Bundle-SymbolicName保持一致。	是
bundleVersion	插件版本，与上面的version取值保持一致。	是
priority	插件优先级，可赋值默认值：5。	是
fileName	插件jar的文件名称。	是
bundleDesc	插件描述，用来介绍bundle功能。	是
versionDesc	插件版本描述，用来介绍版本更迭时的功能特性。	是

**package-info.json**文件模板：

```
{
  "specVersion": "1.0",
  "fileName": "codec-demo",
  "version": "1.0.0",
  "deviceType": "WaterMeter",
  "manufacturerName": "Huawei",
  "model": "NB IoT Device",
  "description": "codec",
  "platform": "linux",
  "packageType": "CIGPlugin",
  "date": "2017-02-06 12:16:59",
}
```

```
"ignoreList": [],
"bundles": [
{
    "bundleName": "WaterMeter-Huawei-NBIoTDevice",
    "bundleVersion": "1.0.0",
    "priority": 5,
    "fileName": "WaterMeter-Huawei-NBIoTDevice-1.0.0.jar",
    "bundleDesc": "",
    "versionDesc": ""
}
]
```

**步骤4** 选中“package”文件夹中的全部文件，打包成zip格式（“package.zip”）。

 **说明**

“package.zip”中不能包含“package”这层目录。

----结束

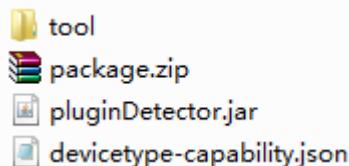
#### 5.5.3.2.4 编解码插件质检

编解码插件的质检用于检验编解码是否可以正常使用。

**步骤1** 从物联网平台服务商获取检测工具。

**步骤2** 将检测工具“pluginDetector.jar”、Profile文件的“devicetype-capability.json”和需要检测的编解码插件包“package.zip”和tool文件夹放在同一个目录下。

**图 5-44 文件准备目录**



**步骤3** 获取设备数据上报的码流，并在检测工具的“data report”页签，将码流以十六进制格式输入，例如：AA72000032088D0320623399。

**步骤4** 点击检测工具的“start detect”，查看解码后的json数据。

日志文本框会打印解码数据，如果提示“report data is success”，表示解码成功；如果提示“ERROR”，表示解码出现错误。

图 5-45 上报数据解码成功

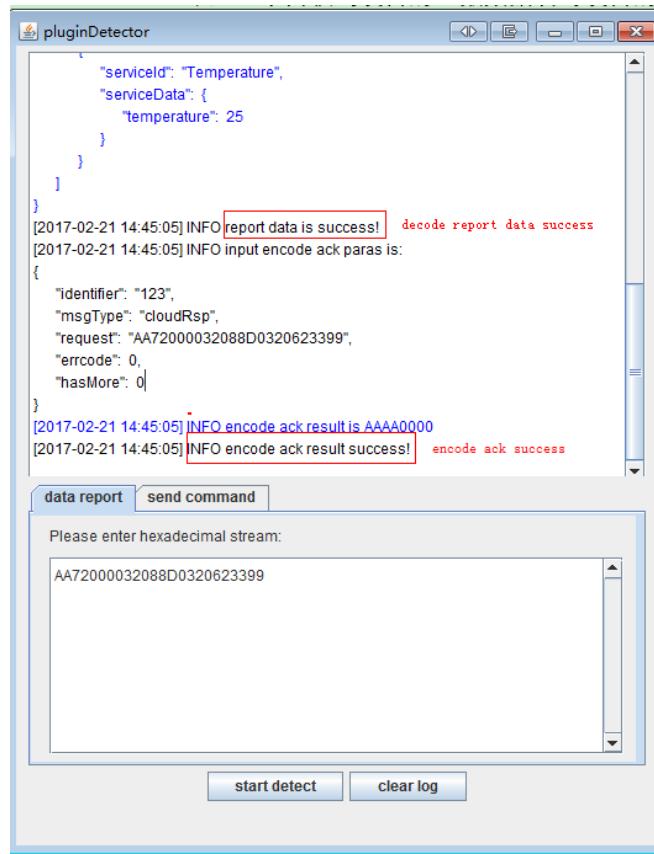
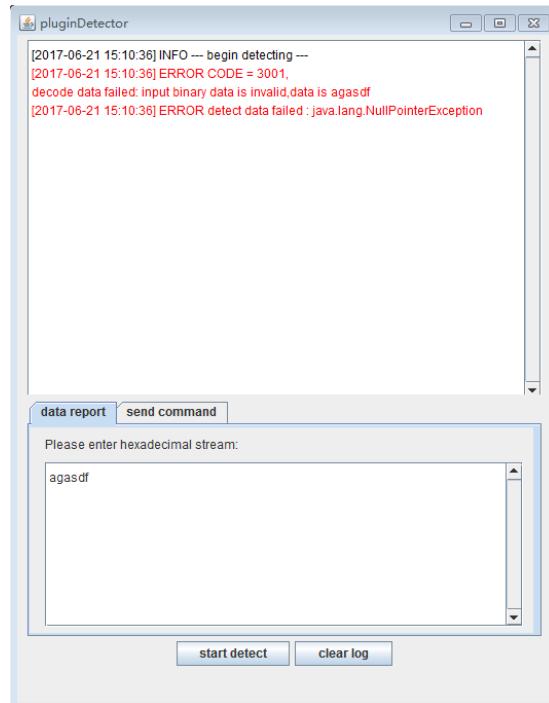


图 5-46 上报数据解码失败



**步骤5** 当解码成功后，检测工具会继续调用编解码插件包的encode方法，对应答消息进行编码。

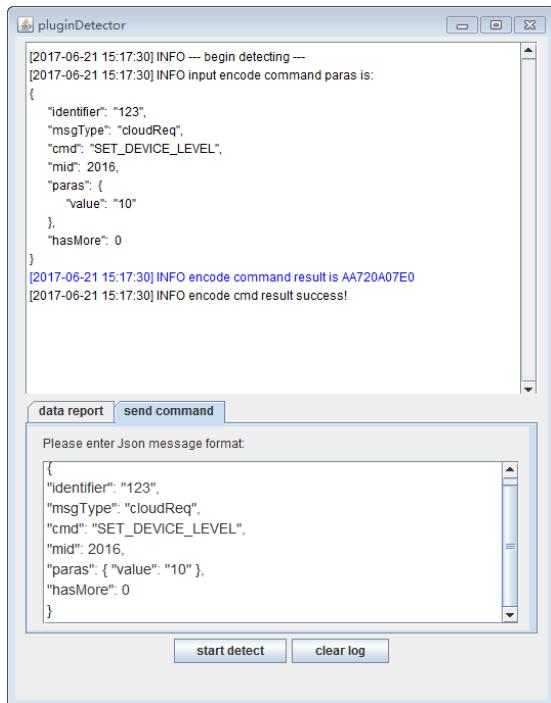
当提示“encode ack result success”时，表示对设备的应答消息编码成功。

**步骤6** 获取应用服务器下发的命令（应用服务器通过调用物联网平台的“创建设备命令”接口进行命令下发），并在检测工具的“data report”页签输入。

**步骤7** 点击检测工具的“start detect”，检测工具会调用encode接口对控制命令进行编码。

如果提示“encode cmd result success”，表示对命令编码成功；如果提示“ERROR”，表示对命令编码出现错误。

**图 5-47 编码控制命令下发成功**



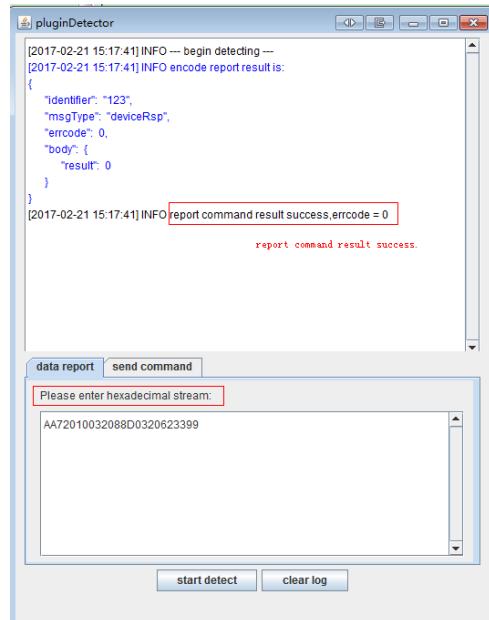
#### 命令示例：

```
{  
    "identifier": "123",  
    "msgType": "cloudReq",  
    "serviceId": "NBWaterMeterCommon",  
    "cmd": "SET_DEVICE_LEVEL",  
    "mid": 2016,  
    "paras": {  
        "value": "10"  
    },  
    "hasMore": 0  
}
```

**步骤8** 获取设备命令执行结果上报的码流，并在检测工具的“data report”页签，将码流以十六进制格式输入，例如：AA7201000107E0。

**步骤9** 点击检测工具的“start detect”，查看解码后的Json数据。

日志文本框会打印解码数据，如果提示“report command result success”，表示解码成功；如果提示“ERROR”，表示解码出现错误。

**图 5-48 命令执行结果解码成功**

----结束

#### 5.5.3.2.5 编解码插件包离线签名

当编解码插件开发完后，在安装到平台之前，需要先对插件包进行签名。此时需要下载离线签名工具，并进行签名操作。

**步骤1** 登录物联网平台的SP Portal。

**步骤2** 选择“系统管理 > 工具”，点击“离线签名工具”，将工具下载到本地。

**图 5-49 下载离线签名工具**

**步骤3** 解压“signtool.zip”，双击“signtool.exe”，运行离线签名工具。

图 5-50 运行离线签名工具



**步骤4** 在“生成数字签名公私钥对”区域，选择“签名算法”，设置“私钥加密口令”，点击“生成公私密钥”，在弹出的窗口中选择需要保存的目录，点击“确定”。

请根据需要选择“签名算法”，当前提供两种签名算法：

- ECDSA\_256K1+SHA256
- RSA2048+SHA256

设置“私钥加密口令”时，口令复杂度需要满足如下条件：

- 口令长度至少为6个字符
- 口令必须包含如下至少两种字符的组合：
  - A-Z
  - a-z
  - 0-9
  - :~`@#\$%^&\*()\_-\_=+|?/<>[]{};,;! ”

在保存目录下将生成公私密钥两个文件：

- 公钥文件：public.pem
- 私钥文件：private.pem

**步骤5** 在“软件包数字签名”区域导入私钥文件并输入口令后，点击“确定”。口令为在步骤4中设置的“私钥加密口令”。

**步骤6** 选择需要数据签名的软件包，点击“进行数字签名”。

数字签名成功后，将会在原软件包所在目录下生成名称为“xxx\_signed.xxx”的带签名软件包。

**说明**

离线签名工具只能对.zip格式的压缩包进行数字签名。

**步骤7** 在“软件包签名验证”区域导入公钥文件后，点击“确定”。**步骤8** 选择需要签名验证的软件包（**步骤6**生成的带签名软件包），点击“进行软件包验签”。

- 验证成功则弹出“验证签名成功！”提示框。
- 验证失败则弹出“验签异常！”提示框。

**说明**

在进行软件包验签时，带签名软件包的存放路径不能包含中文字符。

----结束

## 5.5.4 附录：插件开发说明

### 5.5.4.1 接口说明

为方便问题定位，开发者需要在编解码插件日志中打印必要的定位信息（包括但不限于设备上报信息、命令下发信息等），并对敏感信息做脱敏处理。

#### 5.5.4.1.1 decode 接口说明

decode接口的入参binaryData为设备发过来的CoAP报文的payload部分。

设备的上行报文可以分为两种情况：设备上报数据、设备对平台命令的应答（对应下图中的消息①和⑤；消息④是模组回复的协议ACK，无需插件处理）。两种情况下解码输出的字段不同。

图 5-51 上行报文

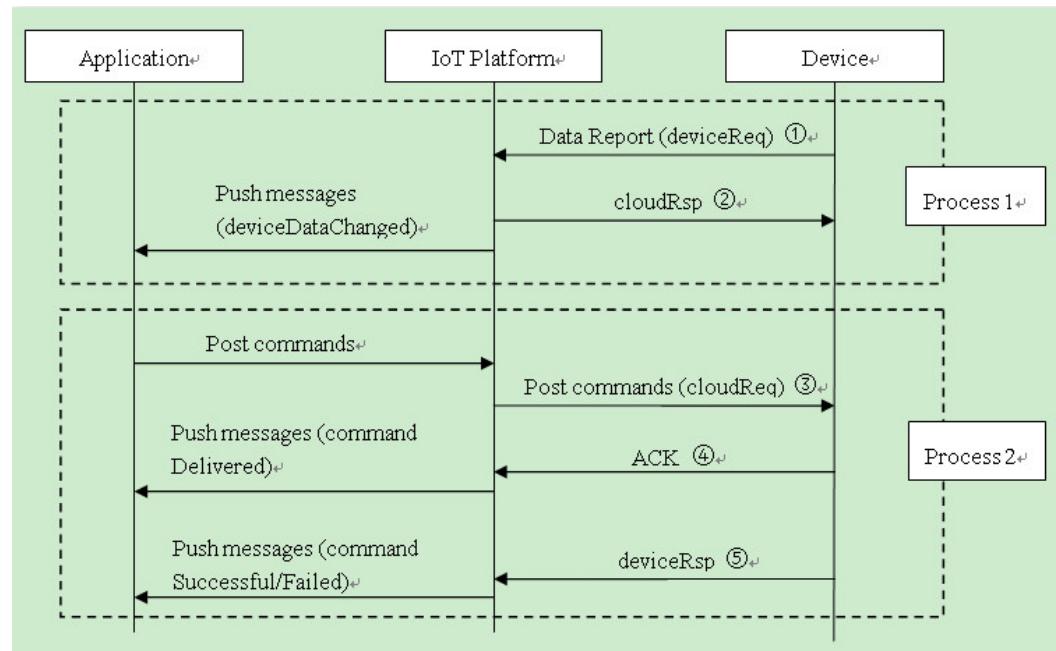


表 5-3 设备上报数据

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识， IoT平台通过 decode接口解析码流时获取该参数，通过 encode接口编码时将该参数放入码流。	否
msgType	String	固定值"deviceReq"， 表示设备上报数据。	是
hasMore	Int	表示设备是否还有后续数据上报， 0表示没有， 1表示有。  后续数据是指，设备上报的某条数据可能分成多次上报，在本次上报数据后， IoT平台以hasMore字段判定后续是否还有消息。 hasMore字段仅在PSM模式下生效，当上报数据的hasMore字段为1时， IoT平台暂时不下发缓存命令，直到收到hasMore字段为0的上报数据，才下发缓存命令。如上报数据不携带hasMore字段，则IoT平台按照hasMore字段为0处理。	否
data	ArrayNode	设备上报数据的内容（详见 <a href="#">表2</a> ）。	是

表 5-4 ArrayNode 定义

字段名	类型	参数描述	是否必填
serviceId	String	服务的id。	是
serviceData	ObjectNode	一个服务的数据，具体字段在profile里定义。	是
eventTime	String	设备采集数据时间（格式：yyyyMMddTHH:mm:ssZ）。 如：20161219T114920Z。	否

示例：

```
{
  "identifier": "123",
  "msgType": "deviceReq",
  "hasMore": 0,
  "data": [
    {"serviceId": "NBWaterMeterCommon",
      "serviceData": {
        "meterId": "xxxx",
        "dailyActivityTime": 120,
        "flow": "565656",
        "cellId": "5656",
        "signalStrength": "99",
        "batteryVoltage": "3.5"
      }
    },
    {"eventTime": "20160503T121540Z"} ,
    {"serviceId": "waterMeter",
      "serviceData": {"internalTemperature": 256},
      "eventTime": "20160503T121540Z"}
  ]
}
```

```
        ]  
    }  
}
```

表 5-5 设备对平台命令的应答

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识，IoT平台通过decode接口解析码流时获取该参数，通过encode接口编码时将该参数放入码流。	否
msgType	String	固定值"deviceRsp"，表示设备的应答消息。	是
mid	Int	2字节无符号的命令id。在设备需要返回命令执行结果（deviceRsp）时，用于将命令执行结果（deviceRsp）与对应的命令进行关联。  IoT平台在通过encode接口下发命令时，把IoT平台分配的mid放入码流，和命令一起下发给设备；设备在上报命令执行结果（deviceRsp）时，再将此mid返回IoT平台。否则IoT平台无法将下发命令和命令执行结果（deviceRsp）进行关联，也就无法根据命令执行结果（deviceRsp）更新命令下发的状态（成功或失败）。	是
errcode	Int	请求处理的结果码，IoT平台根据该参数判断命令下发的状态。  0表示成功，1表示失败。	是
body	ObjectNode	命令的应答，具体字段由profile定义。 <b>说明</b> body体不是数组。	否

示例：

```
{  
    "identifier": "123",  
    "msgType": "deviceRsp",  
    "mid": 2016,  
    "errcode": 0,  
    "body": {  
        "result": 0  
    }  
}
```

#### 5.5.4.1.2 encode 接口说明

encode接口的入参json格式数据，是平台下发的命令或应答。

平台的下行报文可以分为两种情况：平台命令下发、平台对设备上报数据的应答（对应下图中的消息②和③）。两种情况下编码输出的字段不同。

图 5-52 下行报文

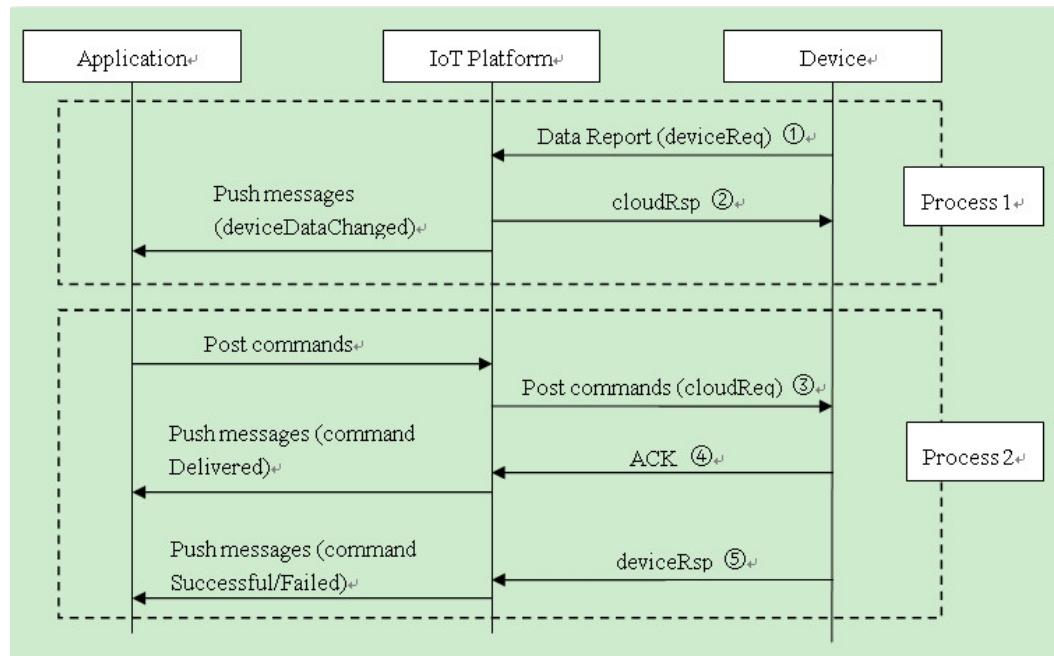


表 5-6 平台下发命令 encode 接口的入参结构定义

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识，IoT平台通过decode接口解析码流时获取该参数，通过encode接口编码时将该参数放入码流。	否
msgType	String	固定值"cloudReq"，表示平台下发的请求。	是
serviceId	String	服务的id。	是
cmd	String	服务的命令名，参见profile的服务命令定义。	是
paras	ObjectNode	命令的参数，具体字段由profile定义。	是
hasMore	Int	表示平台是否还有后续命令下发，0表示没有，1表示有。 后续命令是指，平台还有待下发的消息，以hasMore字段告知设备不要休眠。hasMore字段仅在PSM模式下生效，且需要“下行消息指示”开启。	是

字段名	类型	参数描述	是否必填
mid	Int	<p>2字节无符号的命令id，由IoT平台内部分配（范围1-65535）。</p> <p>IoT平台在通过encode接口下发命令时，把IoT平台分配的mid放入码流，和命令一起下发给设备；设备在上报命令执行结果（deviceRsp）时，再将此mid返回IoT平台。否则IoT平台无法将下发命令和命令执行结果（deviceRsp）进行关联，也就无法根据命令执行结果（deviceRsp）更新命令下发的状态（成功或失败）。</p>	是

示例：

```
{
  "identifier": "123",
  "msgType": "cloudReq",
  "serviceId": "NBWaterMeterCommon",
  "mid": 2016,
  "cmd": "SET_TEMPERATURE_READ_PERIOD",
  "paras": {
    "value": 4
  },
  "hasMore": 0
}
```

表 5-7 平台收到设备的上报数据后对设备的应答 encode 接口的入参结构定义

字段名	类型	参数描述	是否必填
identifier	String	设备在应用协议里的标识，IoT平台通过decode接口解析码流时获取该参数，通过encode接口编码时将该参数放入码流。	否
msgType	String	固定值"cloudRsp"，表示平台收到设备的数据后对设备的应答。	是
request	byte[]	设备上报的数据。	是
errcode	int	请求处理的结果码，IoT平台根据该参数判断命令下发的状态。 0表示成功，1表示失败。	是
hasMore	int	<p>表示平台是否还有后续消息下发，0表示没有，1表示有。</p> <p>表示平台是否还有后续命令下发，0表示没有，1表示有。</p> <p>后续命令是指，平台还有待下发的消息，以hasMore字段告知设备不要休眠。hasMore字段仅在PSM模式下生效，且需要“下行消息指示”开启。</p>	是

 **说明**

在cloudRsp场景下编解码插件检测工具显示返回null时，表示插件未定义上报数据的应答，设备侧不需要IoT平台给予响应。

示例：

```
{  
    "identifier": "123",  
    "msgType": "cloudRsp",  
    "request": [  
        1,  
        2  
    ],  
    "errcode": 0,  
    "hasMore": 0  
}
```

#### 5.5.4.1.3 getManufacturerId 接口说明

返回厂商ID字符串。IoT平台通过调用该接口获取厂商ID，以实现编解码插件和Profile文件的关联。只有厂商ID和设备型号都一致时，才关联成功。

示例：

```
@Override  
public String getManufacturerId() {  
    return "TestUtf8ManuId";  
}
```

#### 5.5.4.1.4 getModel 接口说明

返回设备型号字符串。IoT平台通过调用该接口获取设备型号，以实现编解码插件和Profile文件的关联。只有设备型号和厂商ID都一致时，才关联成功。

示例：

```
@Override  
public String getModel() {  
    return "TestUtf8Model";  
}
```

#### 5.5.4.1.5 接口实现注意事项

##### 接口需要支持线程安全

decode和encode函数需要支持线程安全，不得添加成员变量或静态变量来缓存过程数据。

错误示例：多线程并发时A线程将status设置为Failed，B线程可能会同时设置为Success，从而导致status不正确，引起程序运行异常。

```
public class ProtocolAdapter {  
private String status;  
  
    @Override  
    public ObjectNode decode(final byte[] binaryData) throws Exception {  
        if (binaryData == null) {  
            status = "Failed";  
            return null;  
        }  
        ObjectNode node;  
        ...;  
        status = "Success";  
    }  
}
```

```
    return node;
}

@Override
public byte[] encode(final ObjectNode input) throws Exception {
    if ("Failed".equals(status)) {
        status = null;
        return null;
    }
    byte[] output;
    ...
    status = null;
    return output;
}
}
```

正确示例：直接使用入参编解码，编解码库不做业务处理。

```
public class ProtocolAdapter {
    @Override
    public ObjectNode decode(final byte[] binaryData) throws Exception {
        ObjectNode node;
        ...
        return node;
    }

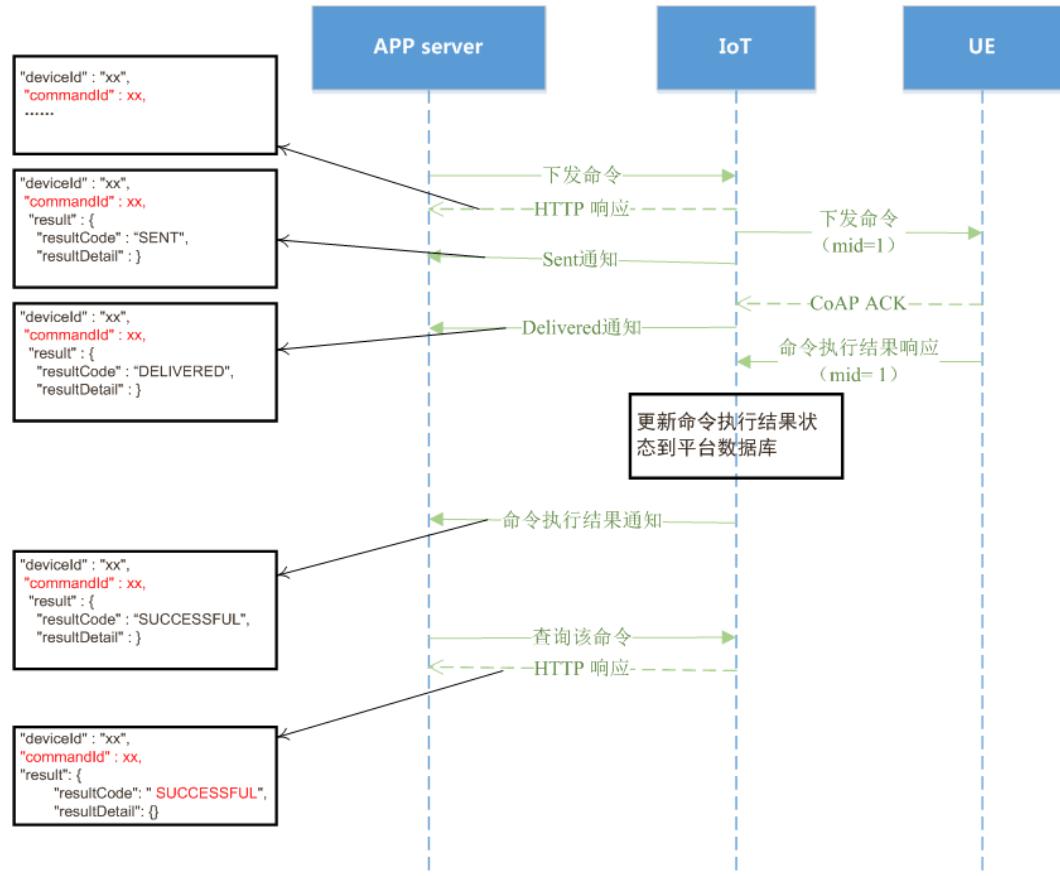
    @Override
    public byte[] encode(final ObjectNode input) throws Exception {
        byte[] output;
        ...
        return output;
    }
}
```

## mid 字段的解释

IoT平台是依次进行命令下发的，但IoT平台收到命令执行结果响应的次数未必和命令下发的次序相同，mid就是用来将命令执行结果响应和下发的命令进行关联的。在IoT平台，是否实现mid，消息流程也有所不同：

### 实现mid

图 5-53 实现 mid 的消息流程

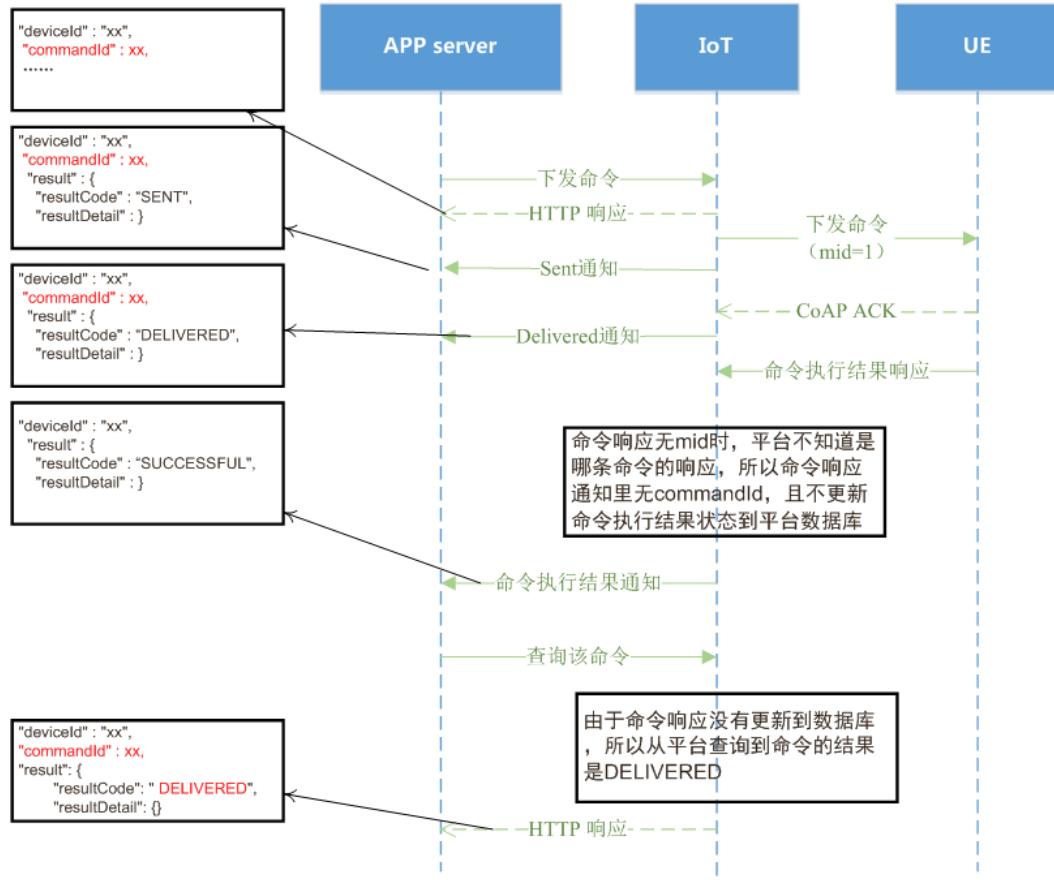


若实现了mid，并且命令执行结果已上报成功，则：

1. 命令执行结果响应中的状态（SUCCESSFUL/FAILED）会刷新到平台数据库中该命令的记录；
2. 平台推送给应用服务器的命令执行结果通知中携带commandId；
3. 应用服务器查询会得到该命令的状态为SUCCESSFUL/FAILED。

**不实现mid**

图 5-54 不实现 mid 的消息流程



若不实现mid，并且命令执行结果已上报成功，则：

1. 命令执行结果响应中的状态（SUCCESSFUL/FAILED）不会刷新到平台数据库中该命令的记录；
2. 平台推送给应用服务器的命令执行结果通知中不携带commandId；
3. 应用服务器查询会得到该命令的最终状态为DELIVERED。

#### 说明

- 上述两个消息流程旨在解释mid字段的作用，部分消息流程在图中简化处理。
- 针对只关注命令是否送达设备，不关注设备对命令执行情况的场景，设备和编解码插件不需要实现对mid的处理。
- 如果厂商评估后不实现mid，则应用服务器不能在IoT平台获取命令的执行结果，需要应用服务器自行实现解决方案。比如应用服务器在收到命令执行结果响应（不带commandId）后，可以根据如下方法来进行响应匹配：
  - 根据命令下发的顺序。使用此方法，平台在对同一设备同时下发多条命令时，一旦发生丢包，将会导致命令执行结果和已下发的命令匹配错误。因此，建议应用服务器每次对同一设备仅下发一条命令，在收到命令执行结果响应后，再下发下一条命令。
  - 编解码插件可以在命令响应消息的resultDetail里加上命令的相关信息来帮助识别命令，比如命令码。应用服务器根据resultDetail里的信息来识别命令执行结果响应和已下发命令的对应关系。

## 禁止使用 DirectMemory

DirectMemory是直接调用操作系统接口申请内存，不受JVM的控制，使用不当很容易造成操作系统内存不足，因此编解码插件代码中禁止使用DirectMemory。

错误示例：使用UNSAFE.allocateMemory申请直接内存

```
if ((maybeDirectBufferConstructor instanceof Constructor))
{
    address = UNSAFE.allocateMemory(1L);
    Constructor<?> directBufferConstructor;
    ...
}
else
{
    ...
}
```

### 5.5.4.2 编解码插件的输入/输出格式

表 5-8 某款水表支持的服务定义

服务类型	属性名称	属性说明	属性类型（数据类型）
Battery	-	-	-
-	batteryLevel	电量(0--100)%	int
Meter	-	-	-
-	signalStrength	信号强度	int
-	currentReading	当前读数	int
-	dailyActivityTime	日激活通讯时长	string

那么数据上报时decode接口的输出：

```
{
    "identifier": "12345678",
    "msgType": "deviceReq",
    "data": [
        {
            "serviceId": "Meter",
            "serviceData": {
                "currentReading": "46.3",
                "signalStrength": 16,
                "dailyActivityTime": 5706
            },
            "eventTime": "20160503T121540Z"
        },
        {
            "serviceId": "Battery",
            "serviceData": {
                "batteryLevel": 10
            },
            "eventTime": "20160503T121540Z"
        }
    ]
}
```

收到数据上报后，平台对设备的应答响应，调用encode接口编码，输入为

```
{  
    "identifier": "123",  
    "msgType": "cloudRsp",  
    "request": [  
        1,  
        2  
    ],  
    "errcode": 0,  
    "hasMore": 0  
}
```

#### 说明

request的取值[1,2]是模拟数据，以实际情况为准。

表 5-9 命令定义

基本功能名称	分类	名称	命令参数	数据类型	枚举值
WaterMeter	水表	-	-	-	-
-	CMD	SET_TEMPERATURE_READ_PERIOD	-	-	-
-	-	-	value	int	-
-	RSP	SET_TEMPERATURE_READ_PERIOD	-	-	-
-	-	-	result	int	0表示成功，1表示输入非法，2表示执行失败

那么命令下发调用encode接口时，输入为

```
{  
    "identifier": "12345678",  
    "msgType": "cloudReq",  
    "serviceId": "WaterMeter",  
    "cmd": "SET_TEMPERATURE_READ_PERIOD",  
    "paras": {  
        "value": 4  
    },  
    "hasMore": 0  
}
```

收到设备的命令应答后，调用decode接口解码，解码的输出

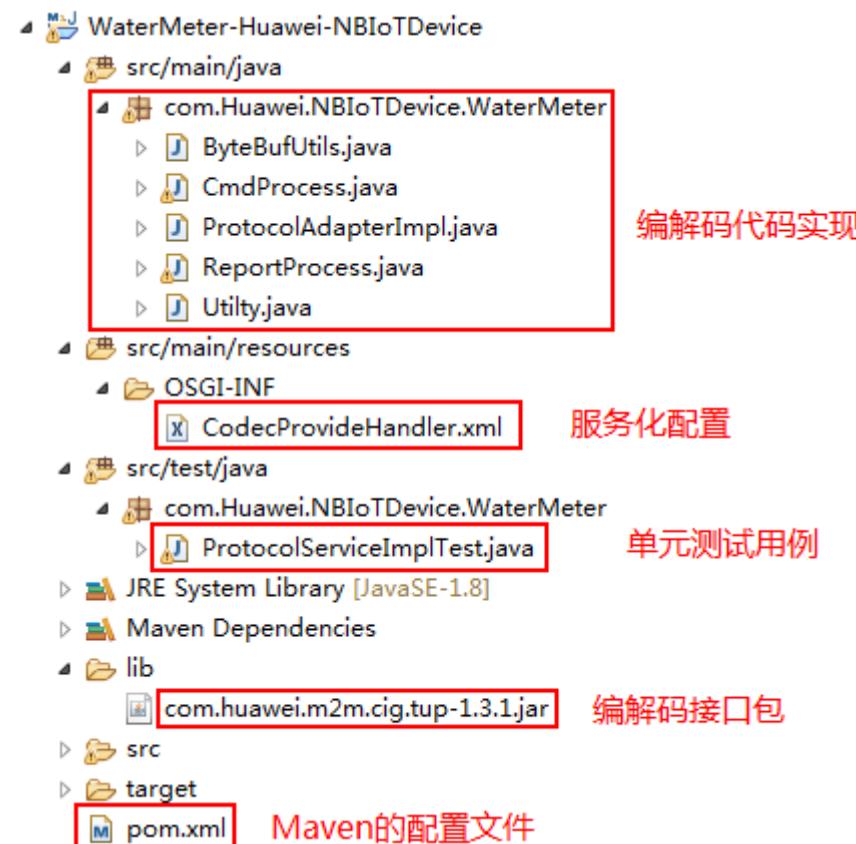
```
{  
    "identifier": "123",  
    "msgType": "deviceRsp",  
    "errcode": 0,
```

```
    "body": {  
        "result": 0  
    }  
}
```

### 5.5.4.3 实现样例讲解

在编解码插件的DEMO工程（[点击获取](#)）中，提供了编解码插件样例，样例工程结构如下图所示。

图 5-55 样例工程结构图



本工程是一个Maven工程，开发者可在此样例工程的基础上修改如下部分，适配成自己需要的编解码插件：

#### 说明

加密算法请使用JDK自带的加密算法，JDK支持的加密算法，详见附录：[JDK支持的加密算法](#)。

#### ● Maven的配置文件

在pom.xml文件中，根据命令规范，修改编解码插件的名字。

```
<groupId>com.thrid.party</groupId>  
<!-- 请修改为你的编解码插件的名字，命名规范：设备类型-厂商ID-设备型号，例如：WaterMeter-  
Huawei-NBIoTDevice -->  
<artifactId>WaterMeter-Huawei-NBIoTDevice</artifactId>  
<version>1.0.0</version>  
<!-- 请检查这里的值为bundle，不能为jar -->  
<packaging>bundle</packaging>
```

#### ● 编解码代码实现

- 在ProtocolAdapterImpl.java中，修改厂商ID（MANU\_FACTURERID）和设备型号（MODEL）的取值。IoT平台通过厂商ID和设备型号将编解码插件和Profile文件进行关联。

```
private static final Logger logger = LoggerFactory.getLogger(ProtocolAdapterImpl.class);
// 厂商名称
private static final String MANU_FACTURERID = "Huawei";
// 设备型号
private static final String MODEL = "NBiotDevice";
```

- 修改CmdProcess.java中的代码，实现插件对下发命令和上报数据响应的编码能力。

```
package com.Huawei.NBIoTDevice.WaterMeter;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

public class CmdProcess {

    //private String identifier = "123";
    private String msgType = "deviceReq";
    private String serviceId = "Brightness";
    private String cmd = "SET_DEVICE_LEVEL";
    private int hasMore = 0;
    private int errcode = 0;
    private int mid = 0;
    private JsonNode paras;

    public CmdProcess() {
    }

    public CmdProcess(ObjectNode input) {

        try {
            // this.identifier = input.get("identifier").asText();
            this.msgType = input.get("msgType").asText();
            /*
            平台收到设备上报消息，编码ACK
            {
                "identifier": "0",
                "msgType": "cloudRsp",
                "request": ***,//设备上报的码流
                "errcode": 0,
                "hasMore": 0
            }
            */
            if (msgType.equals("cloudRsp")) {
                //在此组装ACK的值
                this.errcode = input.get("errcode").asInt();
                this.hasMore = input.get("hasMore").asInt();
            } else {
            /*
            平台下发命令到设备，输入
            {
                "identifier": 0,
                "msgType": "cloudReq",
                "serviceId": "WaterMeter",
                "cmd": "SET_DEVICE_LEVEL",
                "paras": {"value": "20"},
                "hasMore": 0
            }
            */
            //此处需要考虑兼容性，如果没有传mid，则不对其进行编码
            if (input.get("mid") != null) {
                this.mid = input.get("mid").intValue();
            }
            this.cmd = input.get("cmd").asText();
            this.paras = input.get("paras");
        }
    }
}
```

```
        this.hasMore = input.get("hasMore").asInt();
    }

} catch (Exception e) {
    e.printStackTrace();
}

}

public byte[] toByte() {
try {
    if (this.msgType.equals("cloudReq")) {
        /*
        应用服务器下发的控制命令，本例只有一条控制命令：SET_DEVICE_LEVEL
        如果有其他控制命令，增加判断即可。
        */
        if (this.cmd.equals("SET_DEVICE_LEVEL")) {
            int brightlevel = paras.get("value").asInt();
            byte[] byteRead = new byte[5];
            ByteBufUtils buf = new ByteBufUtils(byteRead);
            buf.writeByte((byte) 0xAA);
            buf.writeByte((byte) 0x72);
            buf.writeByte((byte) brightlevel);

            //此处需要考虑兼容性，如果没有传mid，则不对其进行编码
            if (Utility.getInstance().isValidofMid(mid)) {
                byte[] byteMid = new byte[2];
                byteMid = Utility.getInstance().int2Bytes(mid, 2);
                buf.writeByte(byteMid[0]);
                buf.writeByte(byteMid[1]);
            }
        }
        return byteRead;
    }
}

/*
平台收到设备的上报数据，根据需要编码ACK，对设备进行响应，如果此处返回null，
表示不需要对设备响应。
*/
else if (this.msgType.equals("cloudRsp")) {
    byte[] ack = new byte[4];
    ByteBufUtils buf = new ByteBufUtils(ack);
    buf.writeByte((byte) 0xAA);
    buf.writeByte((byte) 0xAA);
    buf.writeByte((byte) this.errcode);
    buf.writeByte((byte) this.hasMore);
    return ack;
}
return null;
} catch (Exception e) {
    // TODO: handle exception
    e.printStackTrace();
    return null;
}
}

}
```

- 修改ReportProcess.java中的代码，实现插件对设备上报数据和命令执行结果的解码能力。

```
package com.Huawei.NBIoTDevice.WaterMeter;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.fasterxml.jackson.databind.node.ObjectNode;

public class ReportProcess {
    //private String identifier;
```

```
private String msgType = "deviceReq";
private int hasMore = 0;
private int errcode = 0;
private byte bDeviceReq = 0x00;
private byte bDeviceRsp = 0x01;

//serviceId=Brightness字段
private int brightness = 0;

//serviceId=Electricity字段
private double voltage = 0.0;
private int current = 0;
private double frequency = 0.0;
private double powerfactor = 0.0;

//serviceId=Temperature字段
private int temperature = 0;

private byte noMid = 0x00;
private byte hasMid = 0x01;
private boolean isContainMid = false;
private int mid = 0;

/**
 * @param binaryData 设备发送给平台coap报文的payload部分
 *          本例入参: AA 72 00 00 32 08 8D 03 20 62 33 99
 *          byte[0]--byte[1]: AA 72 命令头
 *          byte[2]: 00 mstType 00表示设备上报数据deviceReq
 *          byte[3]: 00 hasMore 0表示没有后续数据, 1表示有后续数据, 不带
按照0处理
 *          byte[4]--byte[11]:服务数据, 根据需要解析//如果是
deviceRsp, byte[4]表示是否携带mid, byte[5]--byte[6]表示短命令Id
 * @return
 */
public ReportProcess(byte[] binaryData) {
    // identifier参数可以根据入参的码流获得, 本例指定默认值123
    // identifier = "123";

    /*
    如果是设备上报数据, 返回格式为
    {
        "identifier": "123",
        "msgType": "deviceReq",
        "hasMore": 0,
        "data": [
            {"serviceId": "Brightness",
             "serviceData": {"brightness": 50},
             {
                 "serviceId": "Electricity",
                 "serviceData": {"voltage": 218.9, "current": 800, "frequency": 50.1, "powerfactor": 0.98},
                 {
                     "serviceId": "Temperature",
                     "serviceData": {"temperature": 25},
                 }
             }
         ]
    }
    */
    if (binaryData[2] == bDeviceReq) {
        msgType = "deviceReq";
        hasMore = binaryData[3];

        //serviceId=Brightness 数据解析
        brightness = binaryData[4];

        //serviceId=Electricity 数据解析
        voltage = (double) (((binaryData[5] << 8) + (binaryData[6] & 0xFF)) * 0.1f);
        current = (binaryData[7] << 8) + binaryData[8];
        powerfactor = (double) (binaryData[9] * 0.01);
        frequency = (double) binaryData[10] * 0.1f + 45;
    }
}
```

```
//serviceId=Temperature 数据解析
temperature = (int) binaryData[11] & 0xFF - 128;
}
/*
如果是设备对平台命令的应答，返回格式为：
{
    "identifier": "123",
    "msgType": "deviceRsp",
    "errcode": 0,
    "body" : {****} 特别注意该body体为一层json结构。
}
*/
else if (binaryData[2] == bDeviceRsp) {
    msgType = "deviceRsp";
    errcode = binaryData[3];
    //此处需要考虑兼容性，如果没有传mid，则不对其进行解码
    if (binaryData[4] == hasMid) {
        mid = Utility.getInstance().bytes2Int(binaryData, 5, 2);
        if (Utility.getInstance().isValidofMid(mid)) {
            isContainMid = true;
        }
    }
} else {
    return;
}

}

public ObjectNode toJsonNode() {
try {
    //组装body体
    ObjectMapper mapper = new ObjectMapper();
    ObjectNode root = mapper.createObjectNode();

    // root.put("identifier", this.identifier);
    root.put("msgType", this.msgType);

    //根据msgType字段组装消息体
    if (this.msgType.equals("deviceReq")) {
        root.put("hasMore", this.hasMore);
        ArrayNode arrynode = mapper.createArrayNode();

        //serviceId=Brightness 数据组装
        ObjectNode brightNode = mapper.createObjectNode();
        brightNode.put("serviceId", "Brightness");
        ObjectNode brightData = mapper.createObjectNode();
        brightData.put("brightness", this.brightness);
        brightNode.put("serviceData", brightData);
        arrynode.add(brightNode);
        //serviceId=Electricity 数据组装
        ObjectNode electricityNode = mapper.createObjectNode();
        electricityNode.put("serviceId", "Electricity");
        ObjectNode electricityData = mapper.createObjectNode();
        electricityData.put("voltage", this.voltage);
        electricityData.put("current", this.current);
        electricityData.put("frequency", this.frequency);
        electricityData.put("powerfactor", this.powerfactor);
        electricityNode.put("serviceData", electricityData);
        arrynode.add(electricityNode);
        //serviceId=Temperature 数据组装
        ObjectNode temperatureNode = mapper.createObjectNode();
        temperatureNode.put("serviceId", "Temperature");
        ObjectNode temperatureData = mapper.createObjectNode();
        temperatureData.put("temperature", this.temperature);
        temperatureNode.put("serviceData", temperatureData);
        arrynode.add(temperatureNode);
    }
}
}
```

```
//serviceId=Connectivity 数据组装
ObjectNode ConnectivityNode = mapper.createObjectNode();
ConnectivityNode.put("serviceId", "Connectivity");
ObjectNode ConnectivityData = mapper.createObjectNode();
ConnectivityData.put("signalStrength", 5);
ConnectivityData.put("linkQuality", 10);
ConnectivityData.put("cellId", 9);
ConnectivityNode.put("serviceData", ConnectivityData);
arrynode.add(ConnectivityNode);

//serviceId=battery 数据组装
ObjectNode batteryNode = mapper.createObjectNode();
batteryNode.put("serviceId", "battery");
ObjectNode batteryData = mapper.createObjectNode();
batteryData.put("batteryVoltage", 25);
batteryData.put("batteryLevel", 12);
batteryNode.put("serviceData", batteryData);
arrynode.add(batteryNode);

root.put("data", arrynode);

} else {
    root.put("errcode", this.errcode);
    //此处需要考虑兼容性，如果没有传mid，则不对其进行解码
    if (isContainMid) {
        root.put("mid", this.mid); //mid
    }
    //组装body体，只能为ObjectNode对象
    ObjectNode body = mapper.createObjectNode();
    body.put("result", 0);
    root.put("body", body);
}
return root;
} catch (Exception e) {
    e.printStackTrace();
    return null;
}
}
}
```

## 5.5.5 附录：JDK 支持的加密算法

### 摘要算法

算法名称	算法	摘要长度	备注
MD	MD2	128	-
	MD5	128	-
SHA	SHA-1	160	-
	SHA-256	256	-
Hmac	SHA-384	384	-
	SHA-512	512	-
Hmac	HmacMD5	128	-
	HmacSHA1	160	-
	HmacSHA256	256	-

算法名称	算法	摘要长度	备注
	HmacSHA384	384	-
	HmacSHA512	512	-

## 对称加密算法

算法名称	密钥长度	默认	工作模式	填充方式	备注
DES	56	56	ECB、CBC、PCBC、CTR、CTS、CFB、CFB8到128、OFB、OFB8到128	NoPadding、PKCS5Padding、ISO10126Padding	-
3DES	112、168	16 8	ECB、CBC、PCBC、CTR、CTS、CFB、CFB8到128、OFB、OFB8到128	NoPadding、PKCS5Padding、ISO10126Padding	-
AES	128、192、256	12 8	ECB、CBC、PCBC、CTR、CTS、CFB、CFB8到128、OFB、OFB8到128	NoPadding、PKCS5Padding、ISO10126Padding	256位密钥需要获得无政策限制权限文件

## 非对称加密算法

算法名称	密钥长度	默认	工作模式	填充方式	备注
DH	512~1024(64倍数)	1024	无	无	-

以及Base64。

## 5.6 CA 证书

### 导出 CA 证书

应用服务器侧的CA证书，可以通过如下方式导出。

**步骤1** 使用浏览器打开回调地址，以IE为例。

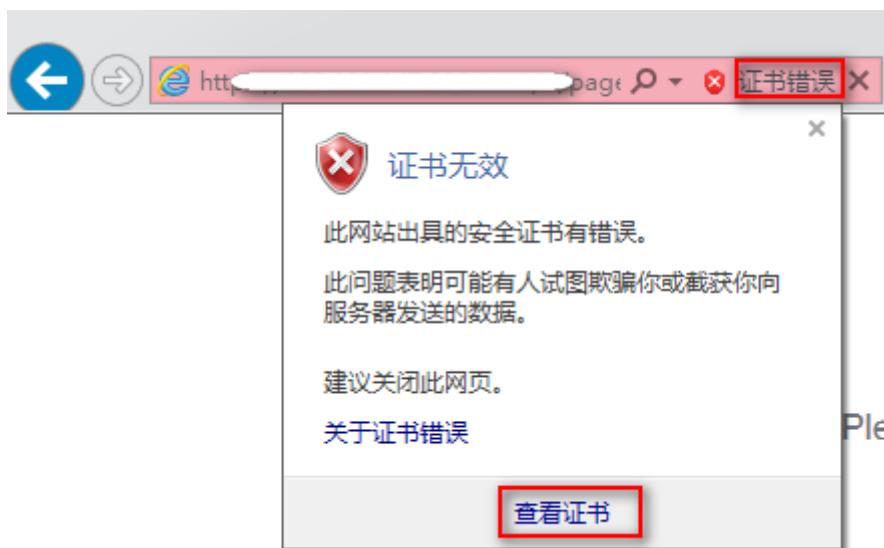
**步骤2** 查看证书。自签名证书和非自签名证书的查看方式不同：

- 如果回调地址使用自签名证书，则会出现“此网站的安全证书存在问题”提示，选择继“续浏览此网站 > 证书错误 > 查看证书”。

图 5-56 自签证书回调地址提示



图 5-57 查看自签证书



- 如果回调地址使用非自签证书，则选择“安全报告 > 查看证书”。

图 5-58 查看非自签证书



**步骤3** 在“证书路径”中查看证书级别，当前查看的为证书的最后一级。

图 5-59 证书路径

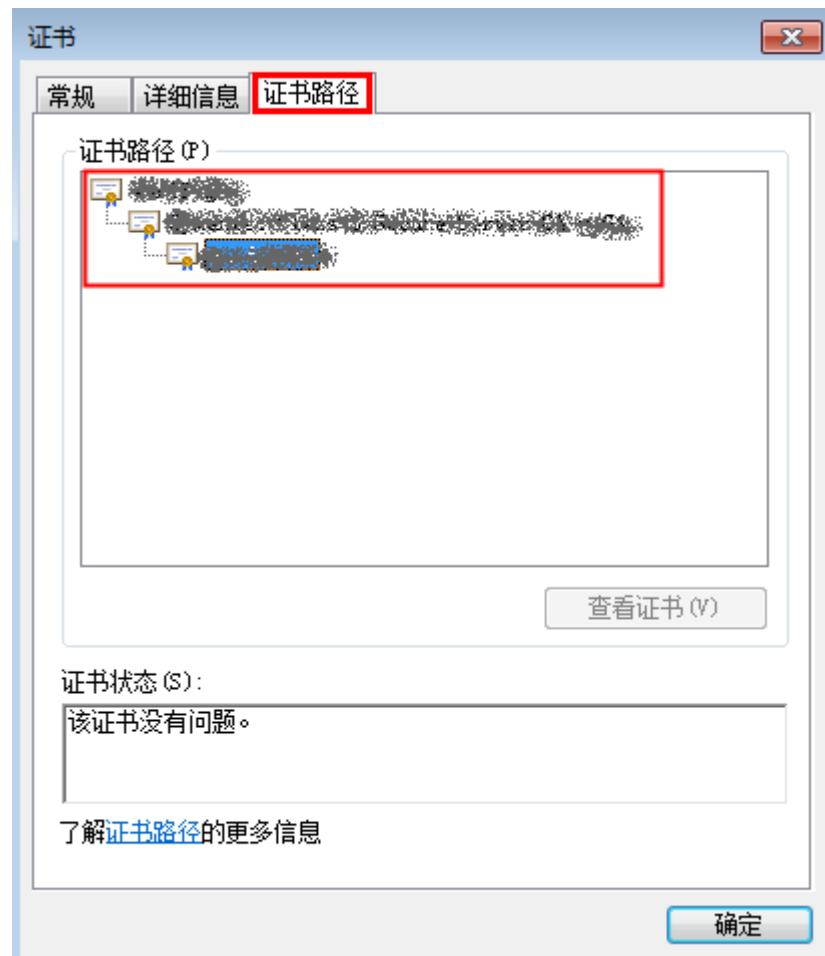
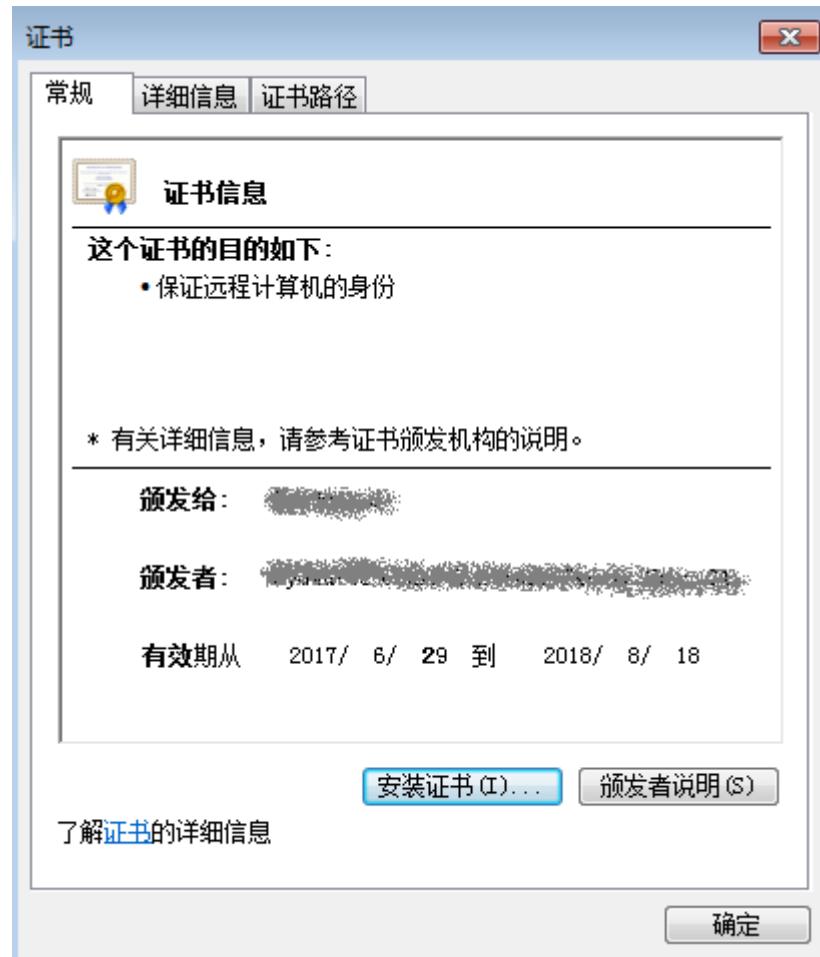


图 5-60 证书常规信息



**步骤4** 在“详细信息”页签，选择“复制到文件 > Base64”，按照证书导出向导将当前级别证书导出。

图 5-61 证书详细信息

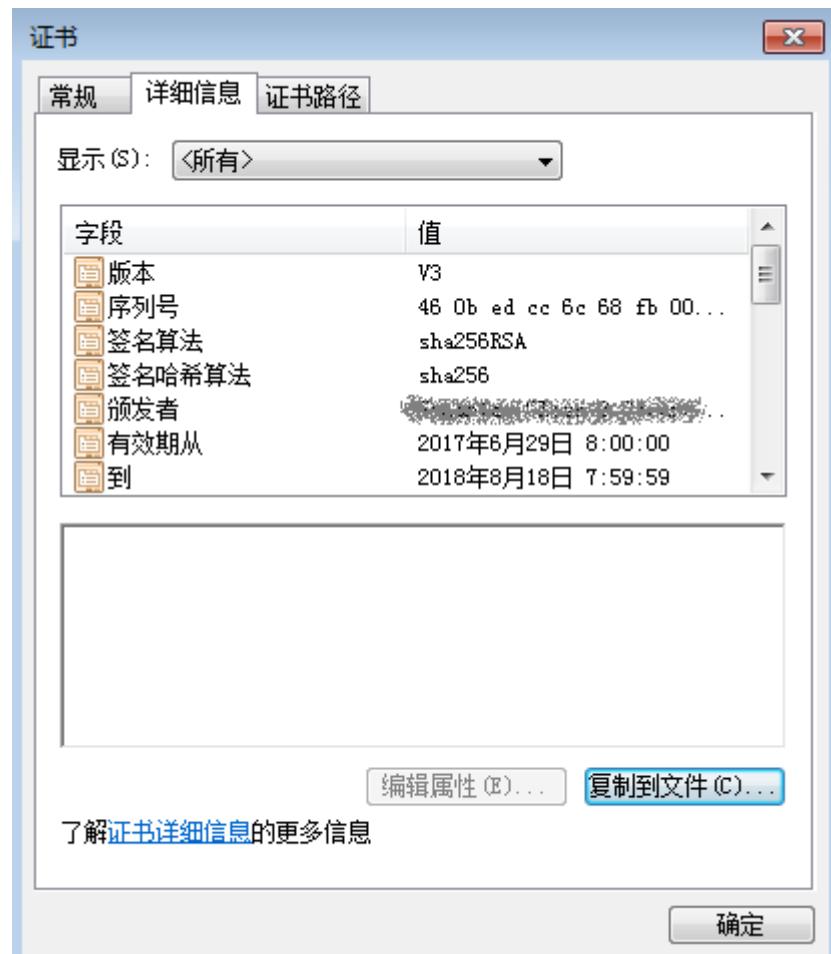
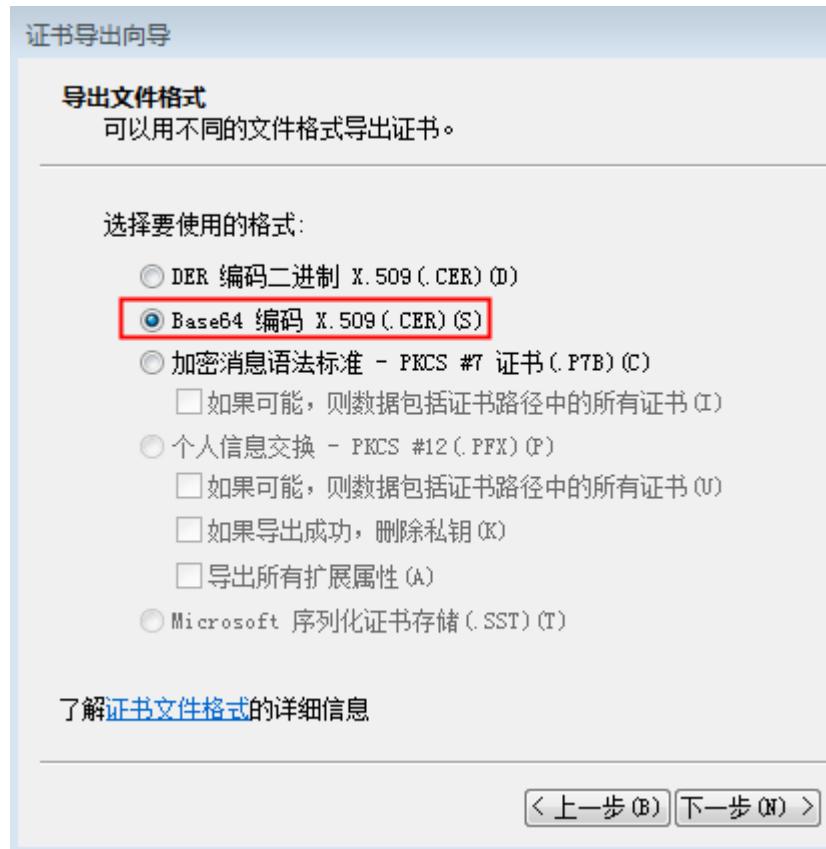


图 5-62 选择证书导出格式



**步骤5** 双击上一级证书，在弹出框中选择“详细信息 > 复制到文件 > Base64编码”，按证书导出向导将上一级证书导出。

**步骤6** 重复步骤5的操作，直到完成所有级别证书的导出。

**步骤7** 使用文本编辑器，将所有导出的证书以首尾相连的方式，合并到一个文件。

#### 说明

合并的文件之间不能存在换行符。

图 5-63 合并证书

```
pLkJkLQYWBSHuxOizGdwCjw1mAT5G9+443fNDsgN3BAAAFC8uXxRAAABAMARjBE
AiA+PkSvZOvIBmxkaBLXLceHH1NqW+Mcrz+xtXNmJ012E1QIgfVeG4xeuWrb7bpoU
smR+LStIG1TPCwimn3JRE3we/fYwDQYJKoZIhvcNAQELBQADggEBADjrCz8a7cax
h7vpyuUFZ/fiKBHE7VLqfppgf3XYNBoqh21qM6gTGzdiSeZj+vx+KOUn38f080Rg
N2aEkag3n03cufIXR8Yn8haXcusz5PONS1MQnN5rZBwpZ8obItiO8KGOh51gHQ+s
S1oX/j8nDDCQqrNkcG2A78nUT+VxGGENxnPmqajP/O2h/kg02qjcnPoj6Elmm/At
5dWWANX374yS7c0fgLZZ1mfZoIqooaRxssSJ15RzyRNu3Bzv5CZCJCGYFqC3RS28Q
vTCjde7TMsAQiWkZ97IK1UMXdbHManm7K85eWcG4Wg8isr9d2GPUZYgcUSc8KfWY
aP5MzoeU6ug=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIFODCCBCCgAwIBAgIQUT+5dDhwtxRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
yjELMAkGA1UEBhMCVVMxFzAVBqNVBAoTD1Z1cm1TaWduLCBJbmMuMR8wHQYDVQQL
ExZWZXJpU2lnbiBUcnVzdCBOZXr3b3JrMTowOAYDVQQLEzEoYykMjAwNiBWZXJp
U2lnbiwgSW5jLiAtIEZvciBhdXRob3JpemVkIHVzZSBvbmx5MUUwQwYDVQQDEzxW
ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcm1tYXJ5IEN1cnRpZmljYXRpb24gQXV0
```

**步骤8** 将合并后的证书文件后缀修改为pem。

**步骤9** 在开发中心的“对接信息 > 应用安全 > 推送证书”中，将证书上传至物联网平台。

**图 5-64 上传证书**



----结束

## 上传 CA 证书

当物联网平台向应用服务器推送HTTPS消息时，需要在物联网平台上传应用服务器侧的CA证书。CA证书可以在开发中心或SP Portal上传：

### 在开发中心上传Profile

**步骤1** 选择“应用 > 对接信息”，在“推送证书”区域，点击“证书管理”。



**步骤2** 系统弹出“CA证书”窗口，检查相应的CA证书是否已上传，如未上传，点击“添加”。



**步骤3** 系统弹出“上传证书”窗口，选择证书文件，并完成各项参数配置后，点击“上传”。



----结束

### 在SP Portal上传Profile

**步骤1** 在“系统管理 > 应用管理 > 应用列表”中选择对应的应用，在“应用定义”界面点击“证书管理”。

The screenshot shows the 'Application Management' interface with the 'Application List' tab selected. On the left, there's a sidebar with 'System Configuration', 'Permissions', 'Logs', and 'Tools'. The main area shows an application named 'FirstApp' with its basic information: Application ID, Create Time (2019-01-15 11:39), and Application Port (8743). In the 'Certificates' tab, there are sections for 'Basic' (including Application ID, Name, Create Time, and Application Port) and 'Advanced' (including Application Port, Device Port, and Push Port). The 'Advanced' section also includes a 'Certificates' sub-section with a 'Manage' button, which is highlighted with a red box.

**步骤2** 系统弹出“CA证书”窗口，检查相应的CA证书是否已上传，如未上传，点击“添加”。



**步骤3** 系统弹出上传证书窗口，选择证书文件，并完成各项参数配置后，点击“确定”。



----结束

# 6 设备集成指导

## 6.1 NB-IoT 模组及终端应用指导

### 6.1.1 前言

#### 6.1.1.1 概述

本文档主要描述垂直行业使用NB模组，开发NB智能终端中需要注意的一些设计要点。

#### 6.1.1.2 使用范围

本文档仅适用于使用海思hi2110/hi2115芯片模组的终端产品，用于指导终端厂家使用海思芯片NB模组进行终端产品开发。

### 6.1.2 NB-IoT 终端硬件设计

#### 6.1.2.1 NB-IoT 应用系统架构

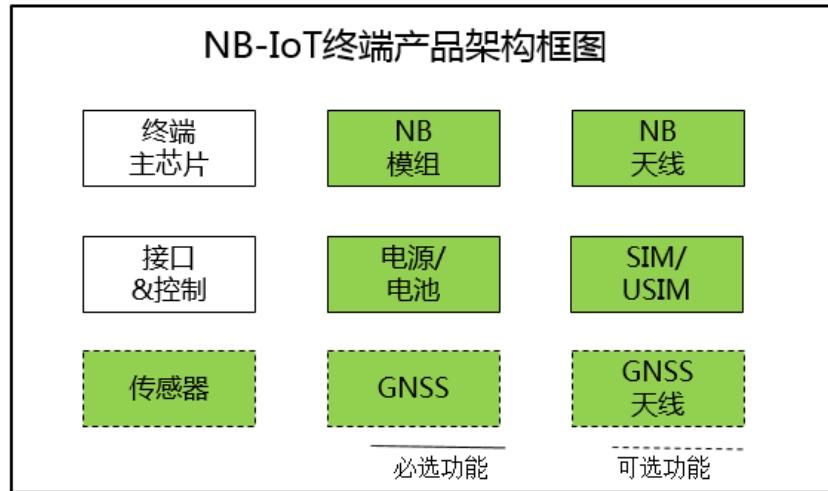
基于NB-IoT技术的终端产品应用系统，主要包括NB智能终端、NB-IoT基站、核心网、IoT连接管理平台、行业应用服务器5部分，相互关系如下图所示。

图 6-1 垂直行业 NB-IoT 应用系统架构图



#### 6.1.2.2 NB-IoT 终端产品硬件系统介绍

采用NB-IoT无线传输技术的终端产品系统架构如下图所示。



传统设备要扩展实现NB-IoT联网，需要硬件增加上图绿色功能模块，虚线功能可以根据具体应用选择配置，NB模组、SIM/USIM、NB天线、模组供电是必须增加的功能块。

下面分功能块简单介绍下设计开发中需要特别注意的关键点。

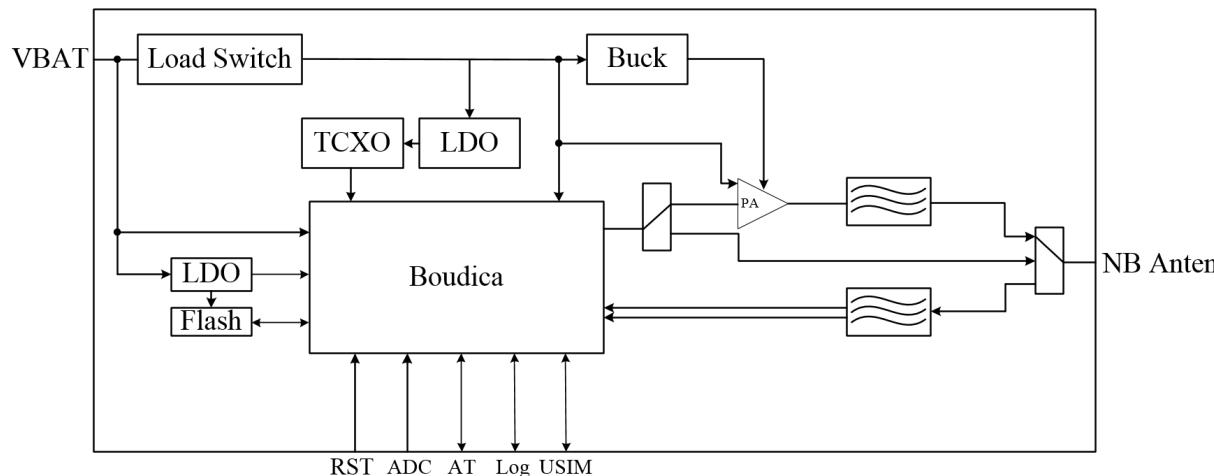
### 6.1.2.3 NB 模组硬件介绍

NB模组主要涉及供电、时钟、射频上下行链路。时钟需要满足频率偏移及相噪的相关要求；电源（包括Buck及LDO）需要满足纹波噪声、动态、环路稳定性等的相关要求；射频需要满足系统发射功率、EVM、接收灵敏度、隔离度、噪声系数等相关指标要求。

由于NB业务场景多，且各场景对硬件软件要求均不一致，对于复杂应用，需要扩展低功耗MCU；对于通用的应用场景，单Boudica即可实现。

#### 6.1.2.3.1 模组单 Boudica 方案

系统架构如下图所示。



由于模组需要在PSM的时候关闭射频及其余耗电组件，因此增加“Load Switch”开关。控制信号由Boudica实现控制；为了满足后续日志记录及版本升级，模组增加“SPI Flash”。

### 6.1.2.3.2 硬件约束

序号	特性	约束说明
1	时钟	主时钟必须选用38.4MHz的温补晶振TCXO，且综合频偏范围控制在10PPM以下，需要确保整个模组生命周期内时钟老化满足系统规格。
2	电源	<ol style="list-style-type: none"><li>1. 模组电源输入3.1~4.2V。</li><li>2. 如果输入电压低于3.1V，需要模组或者底板增加升压电路，保证电压满足芯片工作稳定、性能可靠要求。</li><li>3. Boudica芯片GPIO共有三个电源域，需要外供电源，给Boudica芯片的SIM卡接口及AT串口的电源必须常在线。</li></ol>
3	外置flash	<ol style="list-style-type: none"><li>1. 为支持Boudica V120 FOTA升级特性，需要增加芯片外置FLASH，最小选择2MB flash。</li><li>2. 外置FLASH可用于代码、数据存储，具体容量需要根据模组特性、实际应用要求进行选择。</li></ol>
4	Reset按钮	芯片支持串口升级，升级过程中依赖复位Boudica进入近端升级模式，因此模组外需要设计芯片的复位按钮。
5	MCU选择	MCU根据应用可选，建议具备低功耗、深睡眠模式、串口唤醒等功能。
6	Jlink SWD 接口	项目前期，建议模组外预留Jlink SWD接口，批量发货产品请根据项目应用情况Disable或保留。
7	USIM	<ol style="list-style-type: none"><li>1. 芯片方案支持1.8V或者3.0V，如果USIM要自适应支持1.8V/3.0V，需要在模组或者底板上增加额外设计支持。</li><li>2. 数据线增加20K上拉电阻。</li></ol>
8	串口波特率	<ol style="list-style-type: none"><li>1. 默认AT串口波特率9600bps，终端侧设备需要适配到9600bps。</li><li>2. 默认log串口波特率为921600bps，终端需要区分适配串口速率。</li><li>3. 串口近端版本升级hi2110会发起切波特率，外置MCU串口需要波特率自适应。</li><li>4. 当前芯片平台只支持2线串口。</li></ol>
9	看门狗	芯片无内置硬件看门狗，需要DEVICE系统设计考虑。 可以在模组或者底板上增加硬件看门狗，通过芯片RESET实现芯片异常复位。
10	上电	模组VBAT上电要早于终端底板接口电源。
11	唤醒	模组处于PSM模式下不支持GPIO唤醒，仅支持AT串口信号唤醒。

序号	特性	约束说明
12	POWER ON POWER OFF	Boudica芯片默认上电开机，如果需要控制模组开关机，需在设备底板或者模组内部增加专门控制设计。

### 6.1.2.4 模组使用硬件接口设计

#### 6.1.2.4.1 模组供电

##### 1. 对于外部适配器供电或直流源供电情况

通过增加Buck或者LDO降压到模组的典型值，如3.6V给模组供电，供电系统电流大于500mA可满足要求，电源瞬态1A时确保系统不掉电。电源设计需要考虑纹波<1%，噪声<3%，功能需要防反接、短路保护、过流保护、大动态切换等。

##### 2. 对于使用电池供电情况

通常情况下，NB模组的输入电源为3.1~4.2V。由于电池工作电压会随工作时间逐渐降低，电压低于3.1V，系统需要增加升压Boost器件来保证模组正常的工作。目前Boost器件可选择范围太广，在此不做型号推荐，选型时需要重点考虑效率、漏电、输入电压范围等指标。

#### 说明

当前海思NB-IoT芯片支持2.8V以下工作，但是整个模组受射频器件如PA等约束，工作电压要求3.1V以上。为提升电池供电场景的使用效率，模组硬件设计可以考虑给PA供电单独增加升压器件，以支持电池低电压工作。

#### 6.1.2.4.2 复位设计

由于Boudica近端升级过程中，需要复位操作切换内部运行状态，实现升级版本的加载，因此对于近端升级场景，客户底板需增加复位按键，且考虑到后续单板和结构件集成后的近端升级场景，复位按键建议设计为针孔方案，即免拆壳按复位按钮，同时也避免误触。设计按键时需要考虑按键的触感及回馈力度，同时增加电容防抖动，器件选型上需要考虑使用寿命。

#### 6.1.2.4.3 串口设计

NB当前有两路串口，AT命令口及日志Log检测口。

- 对于终端底板没有MCU的场景，AT命令口需要对接客户的设备，由于NB的串口是3.0V电平，需要客户做好电平转换，并在接口处增加TVS等防护。建议AT/Log串口都预留接插件（建议整机结构件预留），AT串口实现近端升级，Log口用于项目前期的问题定位。
- 对于终端底板有MCU的场景，AT串口需要MCU做串口透传，建议Log也接到MCU串口管脚，便于后续MCU记录日志等功能。同时模组出来的AT/Log串口都预留接插件（建议整机结构件预留），AT串口实现近端升级，Log口用于项目前期的问题定位。

#### 6.1.2.4.4 USIM 卡

当前模组USIM卡的电平是3.0V。硬件设计上USIM由于经常手动插拔，因此信号线上建议增加TVS管防护，同时数据线增加20K上拉电阻（USIM协议要求）。

#### 6.1.2.4.5 看门狗

为了增加Device设备系统的可靠性，建议在Device底板增加硬件看门狗电路，避免模组软件异常后系统挂死。

#### 6.1.2.4.6 Jlink

底板建议设计预留Jlink接口，可以通过测试点或者接插件的方式方便产品调测。

### 6.1.2.5 终端产品硬件设计

#### 6.1.2.5.1 终端产品硬件要求

NB-IoT终端产品应用Case多，使用场景复杂多样，上网应用产品数量巨大，各种应用具体要求也不相同。产品实际设计中要根据应用规格需求做分析设计，特别关注产品使用年限、寿命、可靠性等要求。在产品设计阶段必须保证寿命、性能指标、可靠性满足产品实际使用要求，开发中需要关注以下几点：

- 终端产品设计要考虑系统级可靠性，建议在设备底板增加硬件看门狗电路，避免模组软件异常后系统挂死。
- 终端硬件单板器件选型要满足产品使用年限。产品器件选型、硬件可靠性要按照最长使用年限设计，并通过可靠性测试认证。
- 终端设备单板、电池等器件选型，要满足设备使用环境温度要求。
- 终端产品开发阶段，需要进行高低温测试、中长期可靠性等各种测试验证，提前暴露、发现产品可靠性缺陷，并进行优化改进，通过产品可靠性测试认证，确保最终批量上网产品可靠性能够满足最长使用年限要求，避免海量产品在使用生命周期中间出现产品批次硬件故障。

#### 6.1.2.5.2 Device 产品硬件单板 PCB 设计

NB-IoT终端产品应用场景复杂，为保障最终的NB-IoT信号接收性能，NB-IoT终端产品硬件设计需要做好以下几点：

- PCB布局时，做好隔离分区，避免射频信号受干扰：
  - 数字信号和模拟信号分开，避免相互干扰。
  - 电源和射频信号放置在芯片两侧，减少电源对射频的干扰。
  - 如果PCB空间容许，时钟最好单独隔腔处理。
  - 如果有NB和GNSS天线，两个天线通道分开走线，避免相互干扰。
- 模组到天线口的射频走线采用微带线设计，表层走线尽量短，不要穿层，减少损耗：
  - 射频信号线处禁止放置电源、时钟等强辐射器件。
  - 射频走线控制线宽，实现50欧姆阻抗设计。
  - 天线接口处预留 $\pi$ 型匹配电路，用于天线和单板进行50欧姆阻抗匹配调整。
- 做好PCB叠层设计，实现50欧姆阻抗匹配：
  - 终端产品板推荐至少4层PCB设计，做好隔离和屏蔽，避免单板PCB设计不好影响NB模组射频接收性能。

- 建议第二层作为完整地，实现良好隔离。
  - 天线信号处大面积铺铜，打地孔包住射频线。
- 终端产品研发阶段做好开发调测，确保射频性能指标满足设计要求：
    - 终端硬件开发阶段首先要保证单板传导测试接收灵敏度达到设计指标要求，整机带天线要在暗室中测试OTA指标，满足NB-IOT测试规范要求。
    - 采用海思NB-IoT芯片的整机产品，通过Openlab测试后方可批量上网。
    - NB-IoT整机产品批量上网前应满足工信部NB-IOT终端进网检测要求，通过整机OTA、性能指标等测试认证。

电池供电产品，电池的选择需考虑电池自身损耗、受温度影响等诸多因素。建议提供终端业务模型给专业电池厂家，进行电池评估选型，以确保产品使用寿命功耗要求。

为提升电池利用效率，硬件可以考虑增加升压器件。

#### 6.1.2.5.3 终端产品射频接收灵敏度

终端产品最终的网上表现和产品的射频性能指标强相关，针对终端单板，研发阶段传导灵敏度建议单次指标优于-115dBm。终端整机产品最终接收性能除受单板射频灵敏度影响外，还受天线性能、天线和单板的匹配影响。终端整机产品要在微波暗室中测试TIS、TRP。当前NB产品，23dBm发射功率，TRP建议19以上，TIS建议<=-112dBm。

最终产品的射频指标，要根据产品使用行业要求和测试认证要求设计，满足各种测试认证规范后方可批量上网使用。

#### 6.1.2.5.4 终端产品现网覆盖要求

产品现网使用体验受网络质量影响，通常网络覆盖情况可以通过信号强度RSRP和信号质量SINR两个参数来评估，信号强度或者信号质量越好，则业务性能越好。3GPP将NB-IoT的覆盖划分3个等级，即覆盖等级0,1,2，且主要由信号强度RSRP划分。覆盖等级0覆盖最好。假设NB基站发射功率20W的情况下，推荐的覆盖等级门限如下：

覆盖等级	RSRP门限	SINR门限
0	$\geq -105\text{dBm}$	$\geq 7\text{dB}$
1	$-105\text{dBm} > \text{RSRP} \geq -115\text{dBm}$	$7\text{dB} > \text{SINR} \geq -3\text{dB}$
2	$-115\text{dBm} > \text{RSRP} \geq -125\text{dBm}$	$-3\text{dB} > \text{SINR}$



##### 说明

RSRP的判定门限可以通过网络进行配置，建议覆盖等级0或者1。

终端整机产品在现网调测使用中，可以通过AT命令查询如上RSRP、SINR参数，确认网络实时状态。

#### 6.1.2.6 终端产品调测

产品研发阶段要进行信号质量测试、可靠性测试，确保单板设计质量。批量生产加工前必须完成可靠性测试，试验总体可分为四大类，包括环境适应性试验、可靠性极限测试、长期可靠性测试、小批量可靠性测试。

## 环境适应性试验

包括高温存储、低温存储、运输试验、高温工作、低温工作、太阳辐射等，主要是保障产品设计满足业界的相关标准，是产品设计的底线。

## 可靠性极限测试

包括HALT、温度应力极限、电应力极限等测试，主要是保障产品设计有一定裕度，提升产品市场竞争力，同时也可以避免硬件离散性导致的质量问题。

## 长期可靠性测试

包括长期温循、长期湿热、长期高温、盐雾腐蚀、气体腐蚀等。

## 小批量可靠性测试

主要指小批量温循（高温、低温、温变结合的综合应力），小批量温循的主要目的是发现容差设计类问题。

由于NB物联网产品全球使用，不同国家地区其测试要求、认证标准也各有差异。终端Device产品在商用前，需要确保产品通过使用地的一些特殊测试、认证规范要求，如CE、GCF、FCC、PCTRB等。

## 6.1.3 天线设计

### 6.1.3.1 设计关注点

1. 明确客户重点及硬件布局约束条件，客户重点包括成本，尺寸布局，电气性能，以客户关注重点展开天线设计及选型方案。
2. 终端产品外壳不能使用全金属材料封闭，最低也要满足天线辐射处金属开窗要求。
3. 天线布局要满足尽量远离射频器件，电源，金属屏蔽盖器件的原则。
4. PCB类的单极子微带天线，天线走线下方的地必须清空，用金属实现的单机子天线，可进行局部折叠以充分利用三维空间节省总体体积。
5. PIFA也较常用，设计形式可依托PCB，金属弹片，弹片+支架等不同形式实现，具体选择方式取决于产品空间布局及关注重点。

NB终端产品应用场景及形态多样，天线实现方式也多样，本文所列几款天线设计参考不能完全包含所有场景，仅适合部分应用场景，针对具体应用场景的设计及选择应紧密结合产品ID及应用进行选择。

### 6.1.3.2 NB 终端天线参考规格

项目	单频	多频
频段	824-960MHz	824-960MHz/1710-2200MHz
驻波	<3	<3
峰值增益dBi	1.5+/-0.5	2.5+/-0.5
效率	30%-50%	30%-60%

项目	单频	多频
辐射方向	全向	全向
阻抗ohm	50	50
极化	线极化	线极化

### 6.1.3.3 NB 终端天线样例

NB-IoT物联网应用多样，天线设计、选择形式也呈现多样化特点，下面通过几个例子简单介绍不同种类天线特点。

#### 6.1.3.3.1 单频单极子天线

##### 原理

1/4波长单极子天线。

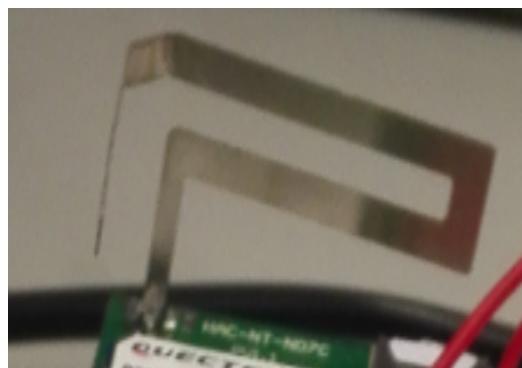
##### 样例尺寸

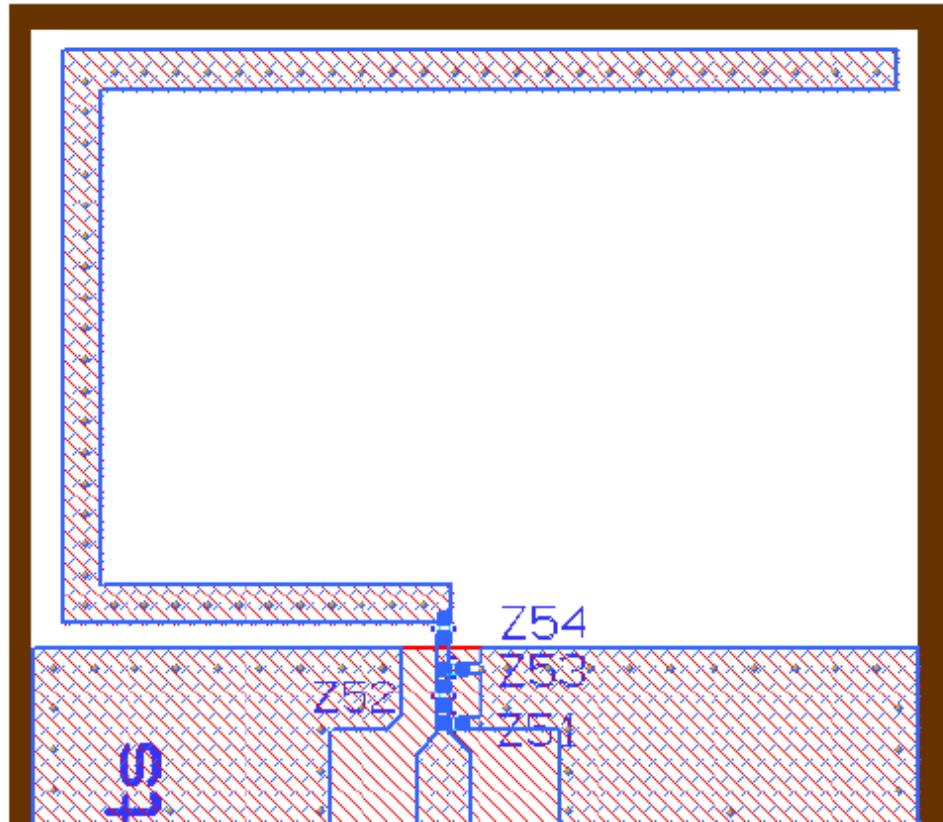
支持频段690-960MHz中频825MHz1/4波长约90mm,即L=90mm，可根据谐振点位置需要适当调整。

##### 样例布局

90mm直接走线布局相对较大，因此可根据产品布局特点进行调整。如1/4波长单极子天线载体是微带PCB,可使用折弯走线或蛇形线，载体为金属片或金属棒则可进行弯折并充分利用垂直方向空间以减小水平方向尺寸面积。

##### 样例视图





#### 说明

单极子天线是NB-IoT单频终端设备可选用的最简易的天线方案，设计形式最简单，成本也相对较低。

### 6.1.3.3.2 PIFA 天线

#### 原理

PIFA天线来源于倒F型天线，增大辐射电阻和提高辐射效率而采用顶部加载的技术，将顶部的辐射线用辐射平面替代，从而形成平面辐射单元，同时，将接地线和馈电线用具有一定宽度的金属片取代可以增大分布电容和减小分布电感，从而增大天线带宽。这样就形成了PIFA天线。

#### 样例尺寸

可支持双频的设计参考，开缝PIFA天线实现双频支持。

## 样例视图

图 6-2 双频 PIFA 天线

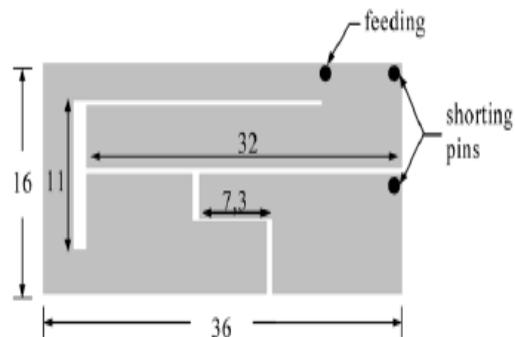
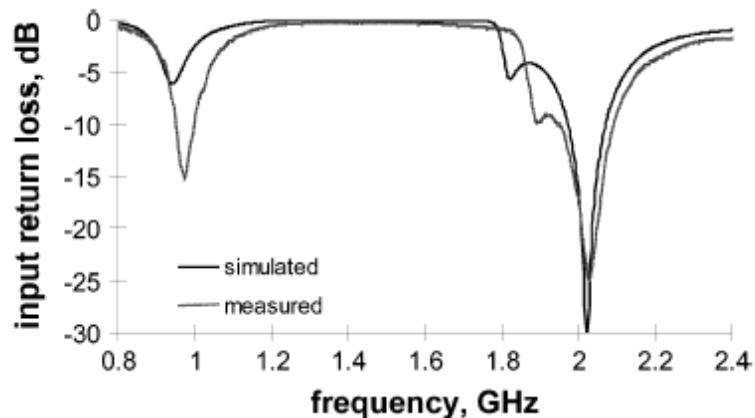


图 6-3 双频 PIFA 天线对应的 S 参数情况



### 6.1.3.3.3 PCB 天线

#### 原理

PCB天线是在PCB上通过走线实现辐射功能的天线，可区分为普通PCB和柔性PCB。

普通PCB即常规的以FR4为基材的普通印制电路板，硬度相对较高，也可使用其他基材，如AD300C或RF30之类的板材做天线基板。当前针对终端类产品天线，单面或双面覆铜的FR4板设计简单、成本低廉所以更受青睐，其典型DK值为4.4。

FPC即Flexible Printed Antenna，柔性电路板，是使用挠性基材制作的印制电路板。其重量比普通PCB轻，厚度薄，占体积小。

#### 样例尺寸

75mm\*25mm\*0.8mm(FPC天线)

## 样例视图

图 6-4 PCB 天线



图 6-5 FPC 天线



### 6.1.3.3.4 外置胶棒天线

#### 原理

外置胶棒天线顾名思义，即天线不在内部或依托终端的PCB等而存在，天线外置于终端。外置天线是一大类，实现方式多种。

#### 样例尺寸

168\*18\*13mm,Φ13mm (宽频带LTE天线)

#### 样例视图

如图所示，天线SMA转接处可折叠



### 6.1.3.4 NB 终端天线应用示例

#### 6.1.3.4.1 Case1 水表+单极子天线

水表视图	天线应用
	<p>如图所示，</p> <ol style="list-style-type: none"><li>1. 红色框选区域，在某水表结构件顶层上做局部长条凸起处理，水表外壳材料是不屏蔽天线辐射信号的塑料材料。</li><li>2. 此应用case，天线可推荐使用文中3.31节所示的金属棒/片实现的1/4波长单极子天线。</li><li>3. 在凸起处使用内置PCB载体的单极子天线，微带走线折叠可减小天线体积，结构件局部变形要求小。</li><li>4. 若垂直表盘面方向有足够空间可操作，建议结构件占用更小的水平方向面积但在垂直方向上局部凸起更高，此种case可配合使用弹簧天线。辐射面原理射频板，电性能更优。</li></ol>

#### 6.1.3.4.2 Case2 路灯+外置胶棒天线

路灯视图	天线应用
	<p>如图所示，</p> <ol style="list-style-type: none"><li>1. 在路灯罩顶部使用外置胶棒天线，也可在灯罩下方垂直放置。</li><li>2. 此种空间布局充足，对环境要求相对略低，不要求天线一定内置的情况下，推荐优选外置胶棒天线，可实现最优的辐射性能。</li><li>3. 设计需注意外部接口的预留。</li></ol>

#### 6.1.3.4.3 Case3 电表+PIFA 天线

检测器视图	天线应用

	<p>如图所示，</p> <ol style="list-style-type: none"><li>是电柜上的电表，圆柱形状，此种case可以使用金属片+塑胶材质支撑件来的PIFA天线，可充分利用三维立体布局以节省空间。</li><li>若射频板对面积限制要求不高但对产品高度方向尺寸有要求，则推荐使用微带PCB的PIFA天线形式。</li></ol>
---	---

#### 6.1.3.4.4 Case4 气表+FPC 天线

燃气表视图	天线应用
	<p>如图所示，</p> <ol style="list-style-type: none"><li>是某物联网应用燃气表，燃气表结构件已固定难以再变更，但体积相对略大，内部有空间可操作。</li><li>此类case，天线选择可推荐适应内置天线，PCB及FPC是其典型应用，FPC更具有灵活布局的特点。可将PCB或FPC天线布局燃气表内的黑框位置。</li></ol>

### 6.1.3.5 天线设计部分原则及建议

#### 6.1.3.5.1 PCB 天线设计原则

根据天线技术发展及工艺技术的成熟，PCB类别的天线在终端射频类天线中占重大比例，印制电路板工艺技术的成熟发展，精度的加工技术及高度的一致性，PCB类天线在终端射频天线中占比很大，本文提供几条PCB天线设计的基本原则如下：

- 确保传输线的线特性阻抗是50ohm。
- PCB走线尽可能短，以减小天线线路损耗。
- PCB尽可能走直线，避免直角走线，减少或避免通过过孔连接到不同层的情况出现。
- PCB走线周围要有良好的参考地，避免其它信号线靠近天线走线而没有地隔离。

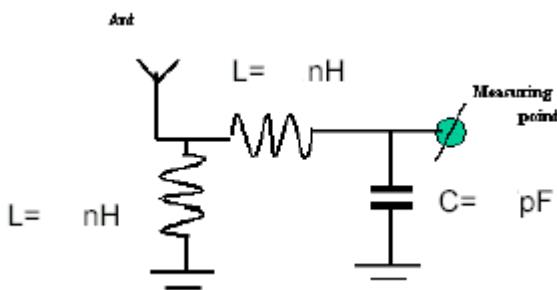
#### 6.1.3.5.2 PCB 板上匹配电路预留

射频终端天线需要与射频前端匹配，这种匹配主要是阻抗的匹配。

单频与双频天线匹配电路：一般采用T型或pi型网络进行匹配。所以需要在射频前段预留T型或pi型网络。

匹配电路示意如图：

图 6-6 pi 型网络图



### 注意

匹配网络应尽可能靠近天线端摆放，目的是使天线匹配到 $50\text{ohm}$ 后与 $50\text{ohm}$ 线直连不反射，即正确的做法是天线先匹配到 $50\text{ohm}$ ，在通过 $50\text{ohm}$ 走线连接到模组。

#### 6.1.3.5.3 内置天线对于终端整体设计的通用要求

1. **走线：**在关联RF的布线时要注意转弯处运用45度角走线或圆弧处理以减小寄生电容，做好铺地隔离和走线的特性阻抗仿真；RF地要合理设计，RF信号走线的参考地平面要找对，并保证RF信号走线时信号回流路径最短，且RF信号线与地之间的相应层没有其它走线影响它（主要是方便PCB布线的微带线阻抗的计算和仿真）。
2. **布局：**布板RF模块附近避免安置一些零散的非屏蔽元件，屏蔽盒尽量规整一体，同时少开散热孔，最忌讳长条形状孔槽。
3. **壳体：**外壳的表面喷涂材料不能含有金属成分，壳体靠近天线的周围不要设计任何金属装饰件或电镀件。若有需要，应采用非金属工艺实现。
4. **干扰：**为减少其他信号及器件对射频的干扰，射频走线应该远高速数字信号、开关电源、振荡器、滤波器或其他射频器件，射频走线下方严禁有电源走线穿越。
5. **材料：**外壳的表面喷涂材料不能含有金属成分，壳体靠近天线的周围不要设计任何金属装饰件或电镀件。若有需要，应采用非金属工艺实现。

#### 6.1.3.5.4 射频板上外置天线接口预留

如射频模组在原始配置是内置天线，若部分特殊情况有需求要能兼顾外置天线的使用时，可考虑在射频板上预留射频IPEX或SMA接口。

#### 6.1.3.6 注意事项

此规格书中关于天线选型的具体指标要求仅供参考。由于实际应用中场景众多、环境各异、PCB天线走线形式不同，建议结合具体应用场景选择适配整机(水/电/气表等)实际测试指标最优的天线设计方案。

### 6.1.4 电池选型

电池方案只考虑NB模组的需求。

### 6.1.4.1 电池方案设计步骤

NB模组电池选型，各个设计环节需要相关领域提供对应信息，同时利用功耗评估计算工具进行初步计算，并最终联系专业电池厂家确定电池选型，详细步骤如下表。

**表 6-1 NB-IoT 电池选型设计分析步骤**

步骤	简介	描述	提供方
1	应用场景	工作环境（温度范围、温度全年时间占比、湿度等）、电池种类、使用寿命。	终端客户
2	业务模型	上下行的数据大小、频度、模组工作模式（PSM/DRX/eDRX）。	终端客户
3	基站网络配置	网络覆盖情况、基站参数配置。	运营商
4	时延需求	根据NB设备业务模型计算每次业务状态、休眠状态的占用时间，主要涉及：RX/TX/STANDBY/SLEEP。	华为
5	电压电流	模组工作电压电流。	模组厂家
6	电池厂家评估	提供如上使用规格需求信息给相关电池厂家，由厂家提供最终电池选型方案。	电池厂家



#### 说明

不同电池厂家可能会根据自己产品的特点，额外提出一些输入要求。

### 6.1.4.2 NB 模组供电要求

**表 6-2 单 NB 模组供电关键需求**

项目	单位	条件	min	typical	max	备注
电压	V	@all temp	3.1	3.6	4.2	-
电流	mA	脉冲放电	300	-	-	-
		脉冲最大电流	-	1500	-	UL 200B、覆盖等级2。
		瞬态最大	-	1000	-	-
工作温度	°C	正常	-20	25	60	-
		扩展	-40	25	85	-
寿命	年	满足应用标准	6	10	-	以具体case要求为准。

 说明

温度、寿命和认证等以不同终端客户要求为准。

### 6.1.4.3 Case1

以某气表为例，说明电池方案选型过程和参考设计。

## 场景

工作环境：-20~60度

电池种类：一次电池

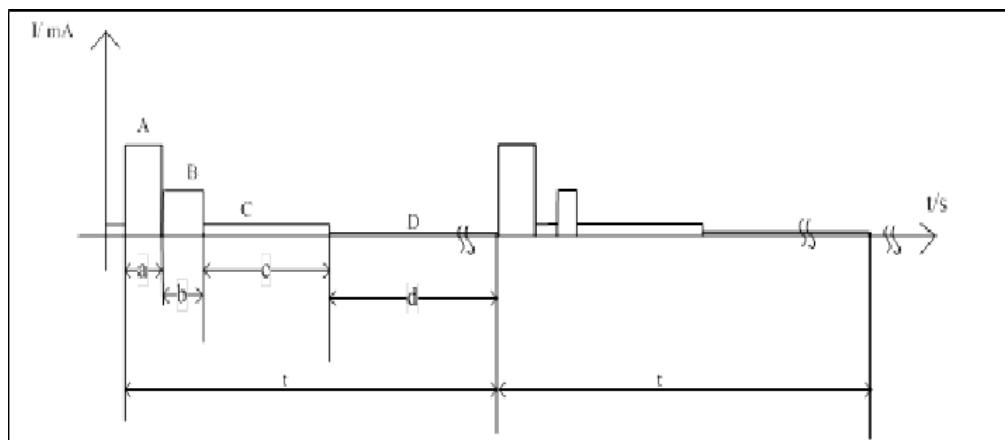
使用寿命：10年

NB模组工作模式：PSM

## 业务模型

UL: 200BYTE, 4次/天

## 各个状态时间



根据典型基站配置，JJFA提供各个状态时间：

时间	基站参数配置1 -1 (InBand)		
-	覆盖等级0	覆盖等级1	覆盖等级2
b/ms	763	1505	14241
a/ms	193	2029	39047
c/ms	24163	24145	18750
d/ms	21574881	21572321	21527962
t/Hour	6	6	6

实际产品可以在实验室根据业务模型进行实际功耗数据、时间测试，计算出单次业务工作功耗数据，从而评估整个生命周期电池消耗。

## NB 模组各个状态电流

模组电流依据各个NB模组厂家提供的数据，如下仅为示例参考。

业务模型	UL:200 BYTE
电流	NB模组示例
B_RX/Ma	75
A_TX/Ma	250
C_STANDBY/mA	18
D_SLEEP/Ma	0.008

## 电池厂家提供电池方案参考

工作模式	覆盖等级0	覆盖等级1	覆盖等级2
电池型号	2*ER18505+SPC1550	ER26500+SPC1550	4*ER34615+SPC1550
容量C (mAh) =CN*Kd*Kg	2*4000*0.8*0.95=6080	8500*0.8*0.95=6460	4*19000*0.8*0.95=57760
使用功耗消耗容量△C1 (mAh/年)	291.3	508.2	4648.4
年自放电消耗容量△C2 (mAh/年)	2*40+43.8=123.8	85+43.8=128.8	4*190+43.8=803.8
△C (mAh/年)	291.3+123.8=415.1	508.2+128.8=637	4648.4+803.8=5452.2

### 说明

电池组实际的放电容量会随着放电电流、温度、使用情况、截止电压以及客户用电器的功耗的不同而变化，电池选型建议寻找专业电池厂家进行评估，以确保容量使用寿命评估准确。

### 6.1.4.4 Case2

某NB-IoT物联网项目，电池初步选定容量4000mAh，希望满足2年使用寿命，下面分析评估该选型是否满足使用目标。

## 场景

工作环境：室内工作环境，具体规格待确认

电池种类：一次ER电池

使用寿命：2年

NB模组工作模式：PSM

## 业务模型

UL：每4小时一次，一次30字节

DL：应答，每次20字节

## 业务功耗

表 6-3 openlab 实测单次业务和休眠态平均功耗

编 号	路 损	1. 数据交互阶段					2. Sleep	
		单消息 激活时 间	单消 息 平均 电流	单消息 能耗	单消息 日均电流	日能耗 (1)	平均 电流	日能耗 (2)
		(s)	(mA)	uAH	uA	uAH	mA	uAH
1	12 4	58	11.24	135.47	5.6	812.8	5.280	240.0
3	12 9	57	11.64	142.47	5.9	854.8	4.300	240.0
5	13 4	58	11.96	136.83	5.7	821.0	4.300	240.0
6	13 9	79.5	11.32	136.83	5.7	821.0	4.300	240.0
7	14 4	43.3	21.18	286.66	11.9	1720.0	5.300	240.0
8	14 9	40.8	31.95	457.23	19.1	2743.4	6.300	240.0
9	15 4	38.7	38.52	562.83	23.5	3377	7.300	240.0
10	15 9	44.4	79.38	1526.67	63.6	9160	8.300	240.0
11	16 4	44.4	154.3 8	5430.00	226.3	32580	8.300	240.0

## 电量和寿命估算

建议有效放电效率按照60%计算：

电池容量	8500	mAH
电池效率	60%	-
自损系数	0.02	-
话务模型	UL	DL
发包频率	6	次/天
寿命要求	5	年

编号	路损	数据交互阶段	寿命
		单消息激活时间(S)	
1	124	58	9.20
3	129	57	8.95
5	134	58	9.15
6	139	79.5	9.15
7	144	43.3	5.76
8	149	40.8	4.05
9	154	38.7	3.42
10	159	44.4	1.42
11	164	44.4	0.42

## 现网信号覆盖情况摸底测试

在产品部署前，需要摸测现网信号覆盖情况，得到信号强度及SNR的分布。示例产品测试SNR>-3，现网属于覆盖等级1，即对应实验室测试MCL小于154dB。

## 评估结果

根据上面分析，建议电池更换为容量8500mAh，满足2年寿命比较保险。

### 6.1.5 软件特性设计

### 6.1.5.1 设计应用约束条件

序号	特性	设计约束
1	FOTA（必备）	硬件诉求 模组自身软件升级要求Boudica120外置flash大于2M。 如果对Device MCU有升级诉求，模组的外挂FLASH资源需满足模组+Device MCU版本升级空间要求。 详细的FOTA特性约束参见 <a href="#">FOTA特性设计使用约束</a> 。
2	NB模组软件 FOTA升级对 Device设备侧要 求	Device侧软件适配要求请参加 <a href="#">FOTA特性设计使用约束</a> 。
3	BIP Over TCP Support via BIP (Bearer Independent Protocol)	BIP特性对资源占用较大，如果客户自行开发，TCP需要部署在外置MCU。
4	1.Support via UART 2.3GPP TR 45.820 3.Support LGU+ AT_CMD List	兼容芯片SDK AT命令接口。

序号	特性	设计约束
5	DTLS	<p>Flash大于30K; RAM大于7K。</p> <p><b>设计约束:</b></p> <p>1) 华为 NB-IoT 模组支持 DTLS 标准方案和 DTLS 优化方案, 终端厂商需要根据对接的 IoT platform DTLS 支持情况进行 DTLS 功能预制 (模组初始化配置为不使用 DTLS 加密)。</p> <p>2) IoT platform 与终端 DTLS 预制匹配列表:</p> <ul style="list-style-type: none"><li>A、 IoT platform 不支持 DTLS, 终端不需要 DTLS 功能预制;</li><li>B、 IoT platform 支持 DTLS 标准方案, 终端预制为标准 DTLS;</li><li>C、 IoT platform 支持 DTLS 优化方案, 终端可以预制为标准 DTLS 或 DTLS 优化方案 (终端根据业务情况自行选择使用哪套方案)。</li></ul> <p>3) 终端预制信息包括: DTLS 方案类型、 PSK 信息, 握手时间。 AT+MSECSWT = &lt; type &gt;, &lt; pskid &gt;, &lt; StrPSK &gt;, &lt; hand shark expire time &gt;</p> <p>参数描述:</p> <ul style="list-style-type: none"><li>type: 0: 关闭 DTLS; 1: 标准 DTLS 并且有重协商; 2: DTLS 优化 RESUME;</li><li>PSKid: 最长 32 字节字符串;</li><li>StrPSK: 固定 16 字节 16 进制数, 预制 PSK 秘钥;</li><li>hand shark expire time: 超时重协商时间, 只对“标准 DTLS 并且有重协商”场景有效。单位: 分钟, 取值范围 0~525600。其中 0 表示每次发消息都要重协商。这个参数配置时必须小于网络层的 NAT 保活时间。</li></ul> <p>4) 特殊情况下, 考虑兼容性和可靠性, 终端可以先预制为支持 DTLS 优化方案, 待优化方案无法与 IoT platform 握手成功后, 回退为 DTLS 标准方案 (整个回退策略和实现机制需要终端根据业务要求慎重考虑)。</p>

### 6.1.5.2 软件架构

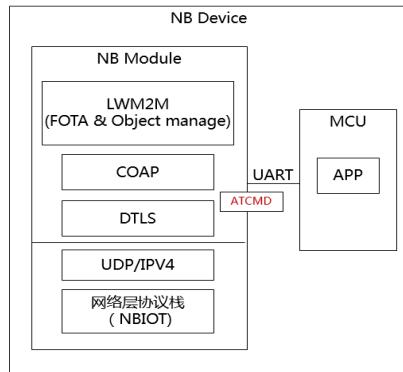
NB模组包含LWM2M、CoAP、UDP、DTLS、IP等协议，终端MCU可以通过AT命令与模组进行交互。

LWM2M可以与支持LWM2M协议的IoT平台进行对接。IoT平台可以通过LWM2M对NB模组进行软件升级和LWM2M一些对象管理，比如CELLID上报。

DTLS可以与IoT平台侧DTLS协议对接，保障NB模组到IoT平台之间的链路安全。

AT命令可通过LWM2M或UDP/IP层收发数据，也可对模组进行近端配置管理，如近端复位模组。

详细软件协议分布如下图所示：

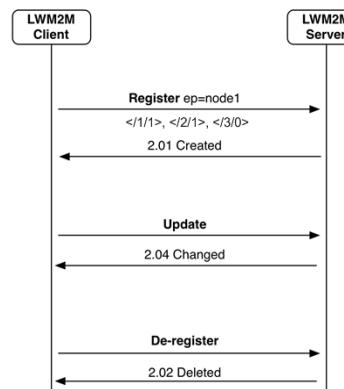


### 6.1.5.3 LWM2M Server 对接软件

LWM2M为标准流程，包含BOOTSTRAP、注册以及基于对象的操作方法（READ、WRITE、EXECUTE、OBSERVER等）。

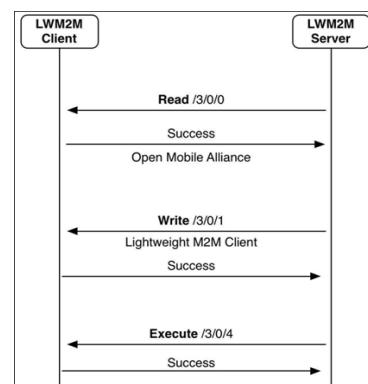
BOOTSTRAP一般支持工厂模式或client init strap流程，不同流程选择可以通过AT命令配置。

#### 6.1.5.3.1 注册流程

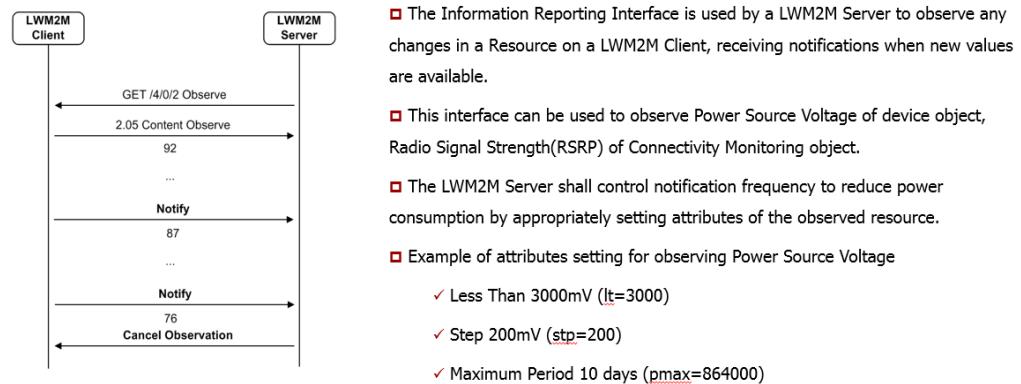


- ❑ After the LWM2M Device is turned on and the bootstrap procedure has been completed, the LWM2M Client MUST perform a “Register” operation to each LWM2M Server.
- ❑ The “Register” operation includes the following parameters
  - ✓ **Endpoint Client Name:** IMEI URN is recommended (`urn:imei:#####`)
  - ✓ **Lifetime:** Specify the lifetime of the registration in seconds.
  - ✓ **Binding Mode:** “UQ” (UDP binding with queue mode)
  - ✓ **Objects and Object Instances:** The list of Objects supported and Object Instances available on the LWM2M Client

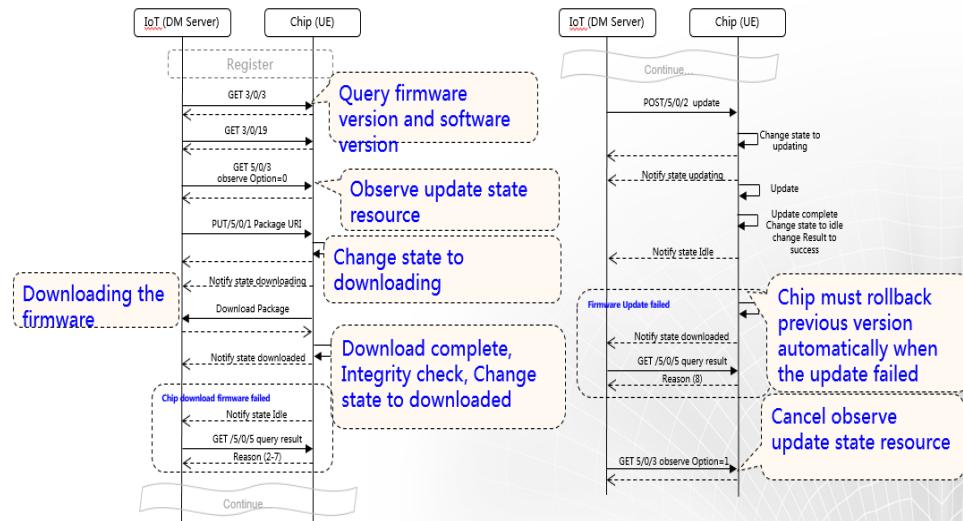
#### 6.1.5.3.2 对象操作



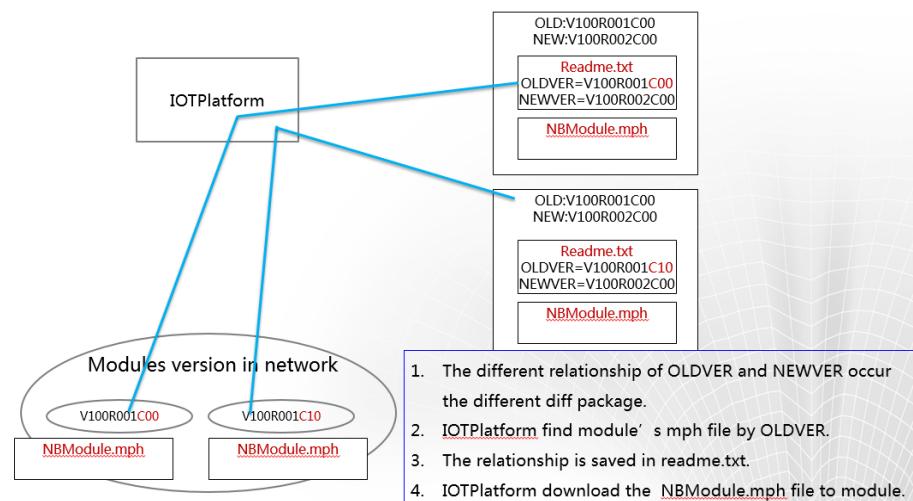
- ❑ The Device Management and Service Enable Interface is used by the LWM2M Server to access Object Instances and Resources available from the LWM2M Client.
- ❑ LWM2M resource is identified by /object id/object instance id/ resource id, for example /3/0/0 means resource ‘Manufacturer’ of object ‘device’ .
- ❑ This interface can be used to query Firmware Version of Device object(Read 3/0/3), Reboot the device(Execute 3/0/4).



### 6.1.5.3.3 软件升级流程



软件采用差分升级可以降低软件包大小，差分算法采用BSDIFF/BSPATCH。打包工程内置BSDIFF，生成两个版本的差异版本，成为差分包。差分包通过如上LWM2M标准流程下载到模组，模组内置BSPATCH算法与老版本进行合并，生成新版本完成升级。



两个不同的新老版本差异关系，会生成不同的差分软件包。

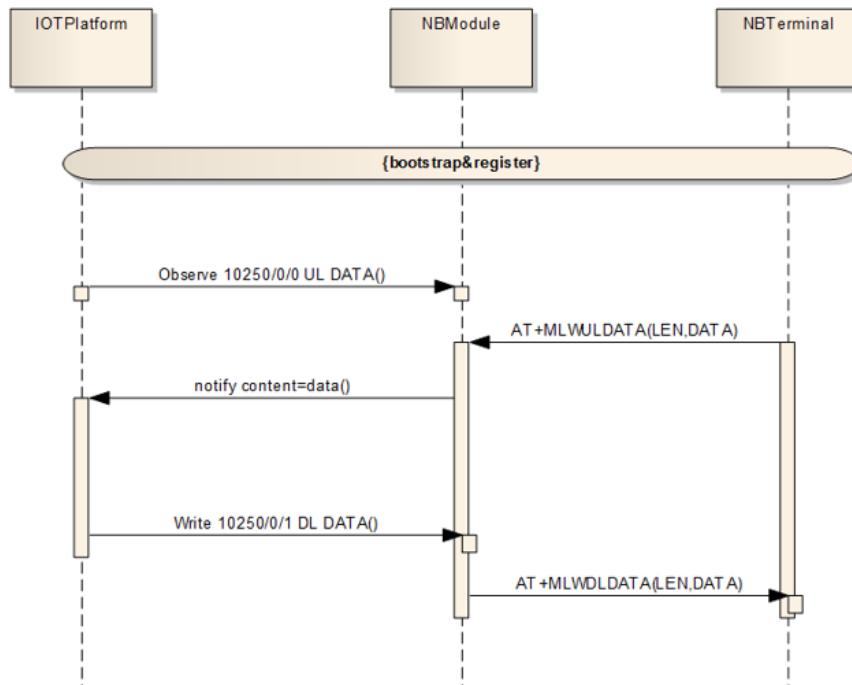
差分版本软件包可以采用如上格式，在差分包中新增README.TXT文件，文件内容说明差分包的新老版本号。IoT平台通过LWM2M流程查询模组真实软件版本，然后读取每个差分包的README.TXT文件，确定下载的差分软件。

芯片软件升级支持异常回退能力，模组设计需要为芯片外置FLASH用于存放老软件和差分软件包。

#### 6.1.5.3.4 终端设备管理软件

华为已经向LWM2M标准组织申请10250对象，用于终端MCU数据透传。

Object definition								
Name	Object ID	Instances	Mandatory	Object URN				
App Data Container	10250	Single	Optional	urn:oma:lwm2m:x:10250				
Resource definition								
ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
0	UL data	R	Single	Mandatory	Opaque			Uplink application data, e.g. gas meter reporting data
1	DL data	W	Single	Mandatory	Opaque			Downlink application data, e.g. application response message of uplink data



基于LWM2M 10250对象进行数据传输，参考流程如下：

- 前提是LWM2M Client已完成与IoT平台注册流程。IoT平台需要订阅10250对象的UL DATA(0)方法。
- 气表MCU通过AT+MLWULDATA发送气表协议内容给模组。
- BOUDICA收到AT+MLWULDATA后，调用LWM2M的NOTIFY方法将数据发送给IoT平台。
- BOUDICA收到IoT平台下行WRITE 10250对象的DL DATA(1)方法后，将PAYLOAD加入到AT+MLWDLDATA发送给终端MCU处理。
- 10250通道数据内容格式由终端厂家定义，NB模组不关注其数据内容，仅作透传。

## 6.1.6 FOTA 升级特性设计

### 6.1.6.1 FOTA 特性概述

#### 6.1.6.1.1 OTA 简介

OTA (Over the Air) 一种终端应用的“空中下载”技术，用于网络设备的空中升级协议，普遍用于物联网设备上。

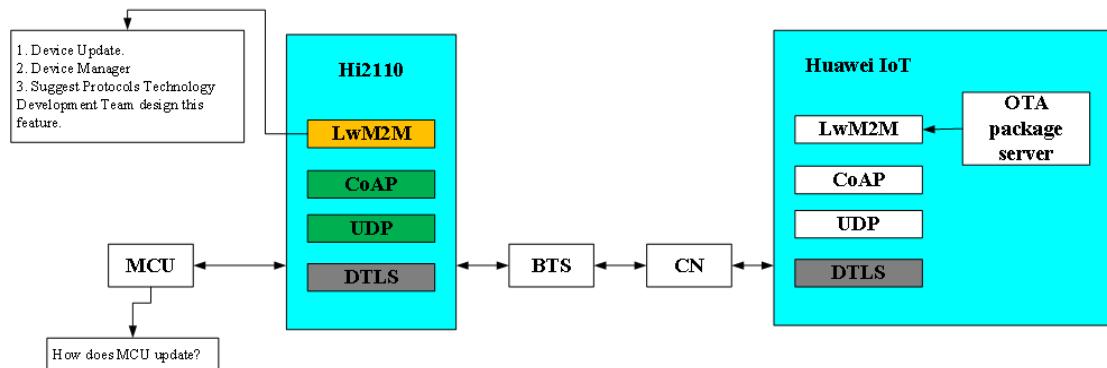
OTA主要分为3个阶段：下载，升级，上报升级结果。

下载、上报升级结果：基于LWM2M 管理整个升级流程，和IoT平台进行包下载。

升级：基于差分、patch（补丁）、全镜像三种方案，通过“flash”刷新系统架构。

#### 6.1.6.1.2 系统整体架构

图 6-7 OTA E2E 系统架构图



如上图，OTA协议依赖的是LWM2M，用于设备管理。目前Boudica移植该协议仅用于升级。

### 6.1.6.2 FOTA 升级与 MCU 交互

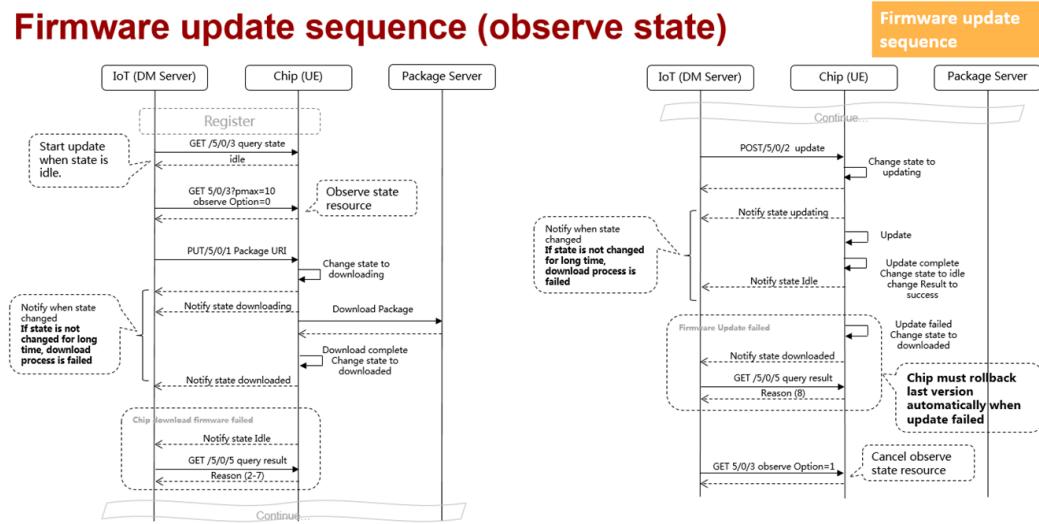
#### 6.1.6.2.1 FOTA 升级时序处理

FOTA时序图如下图所示，主要体现的是FOTA下载、升级，和升级完成之后恢复网络的时序图。



NB-IoT模组（简称“模组”）和IoT平台间的时序图仅在状态变更时通知IoT平台（省电）。

图 6-8 FOTA 时序图



### 6.1.6.2.2 FOTA 升级对 MCU 的约束

#### ?1. 下载阶段

- 当设备发起注册或者上报数据到IoT平台，IoT平台感知设备在线。如果有升级任务，IoT平台则会向设备下发查询版本号/4/0/8、小区ID、信号强度、升级状态的消息。
- 如果IoT平台判断可以发起升级，IoT平台向设备对升级资源/5/0/3发起订阅observe。成功之后下发升级包URI给设备，升级状态从IDLE转换成 DOWNLOADING。模组发送FIRMWARE DOWNLOADING给设备MCU，通知MCU下载升级包已开始。此时，MCU不应该断电，且也不能发送数传相关的AT命令。
- 设备获取到IoT平台下发的升级包URI后，向IoT平台请求升级包数据。如果下载过程中有异常导致下载失败，升级状态从DOWNLOADING转变到IDLE，模组向设备MCU发送FIRMWARE DOWNLOAD FAILED，此时若设备正常联网，MCU仍可以正常处理业务。IoT平台先向设备下发查询失败原因的消息/5/0/5，收到设备回复，再下发取消订阅observe cancel。模组停止FOTA任务，向设备MCU发送FIRMWARE UPDATE OVER，表示FOTA任务结束。
- 当升级包下载完成，校验成功。升级状态由DOWNLOADING转换到DOWNLOADED，模组向设备MCU发送FIRMWARE DOWNLOADED。当升级包下载完成，校验失败。升级状态由DOWNLOADING转换到IDLE，IoT平台先向设备下发查询失败原因的消息/5/0/5，收到设备回复，再下发取消订阅observe cancel。模组停止FOTA任务，向设备MCU发送FIRMWARE UPDATE OVER，表示FOTA任务结束，设备MCU可以正常处理业务。

#### ?2. 升级阶段

升级包校验完成，IoT平台向设备下发升级命令/5/0/2，升级状态由DOWNLOADED转换到UPDATING，模组向设备MCU发送FIRMWARE UPDATING，通知MCU升级开始。此时，MCU不可以断电，不可以发送AT命令，也无法紧急告警。

### 7.3. 恢复网络阶段

- 升级成功，升级状态由UPDATING转换到IDLE，模组向设备MCU发送FIRMWARE UPDATE SUCCESS，待模组成功入网后，MCU可以正常处理业务，IoT平台向设备下发取消订阅observe cancel。模组停止FOTA任务，向设备MCU发送FIRMWARE UPDATE OVER，表示FOTA任务结束。
- 升级失败，升级状态由UPDATING转换到DOWNLOADED，模组向设备MCU发送FIRMWARE UPDATE FAILED，待模组成功入网后，MCU可以正常处理业务。IoT平台先向设备下发查询失败原因的消息/5/0/5，收到设备回复，再下发取消订阅observe cancel。模组停止FOTA任务，向设备MCU发送FIRMWARE UPDATE OVER，表示FOTA任务结束，MCU可以正常处理业务。

#### 说明

FOTA升级重启后设置在KV当中的配置会丢失，因此FOTA任务结束后设备MCU需要重新进行设置。

#### 6.1.6.3 FOTA 特性设计使用约束

序号	特性	设计约束
1	FOTA(必备)	<ol style="list-style-type: none"><li>硬件诉求 模组自身软件升级要求Boudica外置flash大于2M。</li><li>B657SP1版本支持模组FOTA升级。</li><li>B657SP1以前版本如需升级，请单独联系海思了解操作流程。</li><li>B657SP1仅支持模组软件升级，不支持设备MCU版本升级。</li><li>模组FOTA升级需设备配合修改程序，具体要求参见下条“模组软件FOTA升级对设备侧要求”。</li></ol> <p><b>说明</b> FOTA特性详细的实现流程、要求可以参考前面章节内容。</p>
2	模组软件FOTA升级对设备侧要求	<ol style="list-style-type: none"><li>设备收到IoT平台升级消息，模组发送FIRMWARE DOWNLOADING给设备MCU，通知MCU模组开始自身软件升级。</li><li>设备MCU收到模组进行FOTA升级的消息后，需进入FOTA升级保护状态，应该保障MCU不断电，不发送数传相关的AT命令给模组。</li><li>模组FOTA升级完成后，发送FIRMWARE UPDATE OVER给设备MCU，结束FOTA升级状态保护，进入正常工作模式，MCU可以正常处理业务。如果需要控制设备下电，发送AT+CSCON?查询设备状态，如果返回+CSCON:1,0可以执行断电操作；如果返回其他结果，需继续等待。</li><li>设备MCU处于正常工作状态时，给模组通过AT命令发送完数据后，如果需要控制模组下电，需发送AT+CSCON?查询设备状态，如果返回+CSCON:1,0可以执行断电操作；如果返回其他结果，需继续等待。</li></ol>

#### 6.1.6.4 设备侧 MCU 软件 FOTA 适配建议

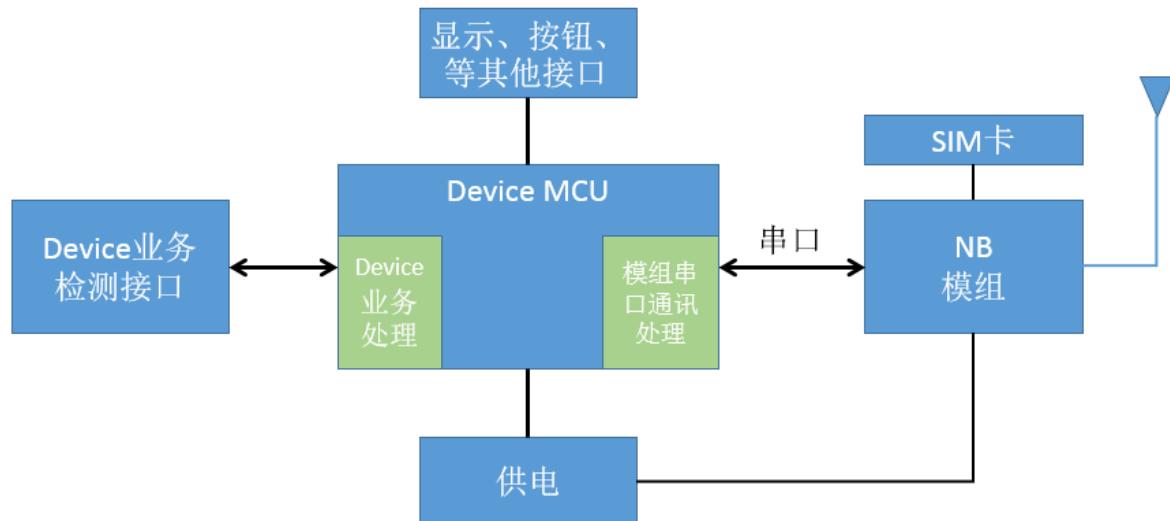
#### 6.1.6.4.1 设备 MCU 和模组串口通讯方式

模组通过串口和设备MCU进行通讯，当前主要有两种串口通讯方式：查询和中断。如果采用查询方式，会有如下影响：

1. 每次设备MCU发送完消息，需要监听串口新消息，导致设备MCU被串口监听任务占用，不能及时处理自身业务。
2. 如果设备发送完数据，每次进行长时间串口监听，会导致功耗数据增大。
3. 模组FOTA特性使用概率极低，如果每天为FOTA升级进行串口监听，会引起功耗增加，电池损耗加大。

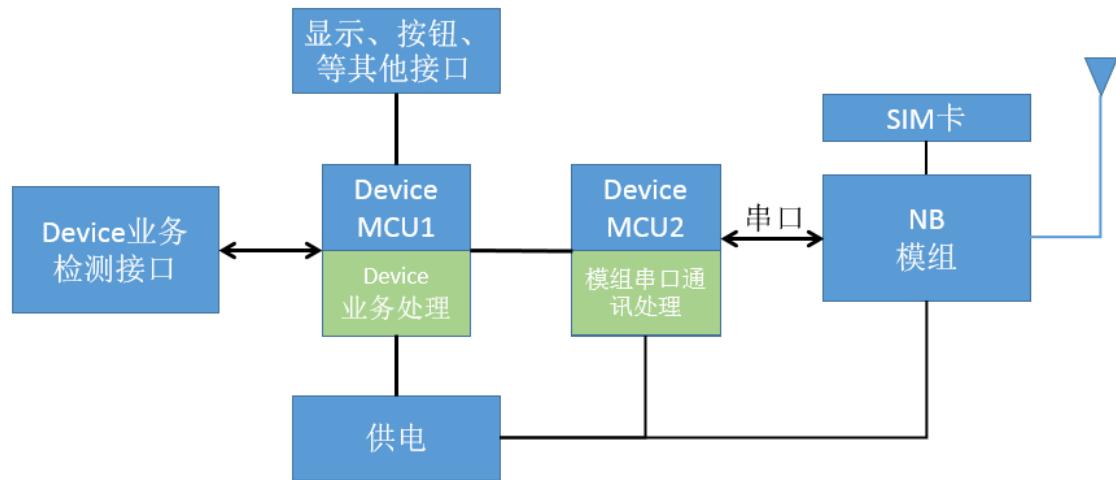
#### 6.1.6.4.2 设备 MCU 和模组串口中断方式通讯说明

设备侧有一个 MCU，同时负责设备自身业务以及设备 MCU 和模组的通讯



- 设备MCU和模组的串口通讯，建议采用中断方式，避免串口监听时间过长，影响设备自身业务。在设备MCU等待模组消息的同时，可以处理自身其他业务。
- 在中断方式下，设备上报数据后，可以响应模组下发的升级消息(中断服务程序只做响应和简单接收，不做大任务处理，避免长时间占用中断)。
- 当设备MCU中断接收到IoT平台下发的FOTA升级消息FIRMWARE DOWNLOADING，设备MCU需要按照**FOTA特性设计使用约束**：进入FOTA升级保护模式，不对模组断电、不发送数传相关的AT命令给模组。
- 如果IoT平台不进行FOTA升级，设备完成自身业务后，查询模组状态。如果模组状态是IDLE，就可以继续处理设备自身业务。

设备侧有两个 MCU，一个负责设备自身业务，一个负责设备 MCU 和模组的通讯



- 设备有2个MCU的应用场景，通常模组和MCU2的串口通讯不会影响MCU1自身业务的处理。MCU2既可采用串口查询方式，也可使用中断方式工作。
- 如果MCU2采用中断方式，具体实现参照上面单MCU中断处理方式。
- 如果MCU2采用查询方式，仅处理模组串口消息。设备MCU1发送业务数据后，MCU2需要等待至少2分钟监听串口消息，以确认是否有FOTA升级以及其他串口消息，等待期间自身业务由MCU1进行处理。
- 当设备MCU2接收到IoT平台下发的FOTA升级消息FIRMWARE DOWNLOADING，需要按照**FOTA特性设计使用约束**：进入FOTA升级保护模式，不对模组断电、不发送数传相关的AT命令给模组。
- 如果IoT平台不进行FOTA升级，设备完成自身业务后，查询模组状态。如果模组状态是IDLE，就可以继续处理设备自身业务。

## 6.1.7 可靠性设计

### 6.1.7.1 控制面防挂死设计

系统类的异常，包括死循环挂死、空指针拷贝导致的指令跑飞、局部变量内存拷贝导致堆栈溢出、动态内存不释放导致内存泄露无可用内存等。

### 支持看门狗

看门狗使用2级管理机制：软件看门狗和硬件看门狗。软件看门狗监控系统内的重要任务，硬件看门狗监控软件看门狗。

1. 新增一个比当前业务优先级更高的软狗任务。

```
//          TASKNAME,      ENTRYPPOINT,      PRIORITY,      STACKSIZE,      QUEUEFUNC
APP_TASK_DEF( "Softwatdog",      Softwatdog _main,      6U,      100,      NULL),
```
2. Softwatdog任务给其它子任务（app、rpc、lwm2m、log）分别设置一个任务状态标记，在主流程(for();)循环体入口)设置为0xff。
3. 在每个子任务的主流程(for();)循环体入口)给自己的任务状态标记清零。
4. Softwatdog任务每5秒检测一次子任务状态标记，如果有某个任务连续6次状态持续为0xff，记录任务名称、当前时间，重启系统。
5. 在Softwatdog任务中喂硬狗（watchdog\_kick），删除当前硬狗喂狗调用点（当前是在IDLE任务中喂狗）。

## 堆栈溢出

使用freeRTOS的堆栈溢出检测自愈功能；不支持freeRTOS的S核无法实现。

freeRTOS支持堆栈溢出的方法：

1. 将configCHECK\_FOR\_STACK\_OVERFLOW设置为1或2。
2. 在操作系统回调vApplicationStackOverflowHook中进行复位自愈。

### 6.1.7.2 数据面防挂死设计

终端数据上传路径有2个风险：

1. Boudica没有硬狗，如果芯片软件彻底挂死之后没有自愈机制。
2. MCU到boudica的路径上可能会出现软件原因导致数据发送失败问题。

需要增加检测和自愈手段，防止终端数据长时间无法上报。

- MCU和boudica之间的消息发送必须等应答。
- MCU每次给boudica透传终端数据（IP/UDP方式）时必须等待boudica的应答，最长等待30秒。只有收到应答或等待超时之后才能发送下一条消息。
- 增加检测机制。
  - 有MCU的场景

MCU每次给boudica透传终端数据（IP/UDP方式）后，如果无法收到boudica的应答，或者收到boudica的失败应答（仅包括系统内部失败场景，剔除外部环境导致的失败）：

以35秒周期执行+MUESTATS，如果连续2次无法收到应答或收到boudica的失败应答，认为boudica异常。

如果在重传周期之内收到其它AT，暂停重传，暂停计数。

如果在重传周期之内收到数据转发命令，停止重传，计数清零。

- 无MCU的场景

A核每次给C核透传终端数据（IP/UDP方式）之后，如果收到boudica的失败应答（仅包括系统内部失败场景，剔除外部环境导致的失败）：

以30秒周期执行+MUESTATS，如果连续5次都失败，认为boudica异常。

如果在重传周期之内收到其它AT，暂停重传，暂停计数。

如果在重传周期之内收到数据转发命令，停止重传，计数清零。

- 增加自愈机制

- 有MCU的场景

MCU检测到boudica异常之后，控制boudica RST管脚重启boudica，之后再重启MCU。

- 无MCU的场景

A核检测到boudica异常之后，重启boudica。

- 防误自愈处理

所有重启类的自愈措施必须增加防止误自愈处理。一旦误自愈，后果比自愈还要严重。同时，自愈和误自愈要有一个平衡，不能因为害怕出现误自愈问题而不做自愈。

## 6.1.8 典型案例

终端\模组研发过程中典型问题参考分析。

### 6.1.8.1 模组 PSM 态功耗超标问题排查

- 首先梳理模组上的耗电器件列表，对比海思模组的参考设计，总结出改动点，包括新增部分。
- 由于海思模组PSM功耗可以达到3uA左右，如果原理图设计一致，则通常情况下不会影响PSM态的功耗，PCB设计不一致会影响指标性能。
- 对比原理图上的所有改动，如果Boudica芯片的外围控制以及上电时序逻辑一致，则可以排除这部分；Boudica V120的管脚定义如下，设计时请保持一致，尤其是SIM卡供电使用Boudica的GPIO11管脚，如果担心GPIO的驱动能力，可以在LDO输出到SIM卡供电上，串联三级管，使用GPIO11做逻辑控制（此部分如果设计异常，PSM功耗约增加400uA左右）。

电压域	管脚	EVK信号定义	EVK信号
R1	GPIO0	PA_EN	SIM_VCC/控制
	GPIO1	PA_MODE0	SIM_I
	GPIO2	PA_MODE1	AT_R
	GPIO3	TX_RX	SIM_O
	GPIO4	PA_PWR_EN	SIM_D
	GPIO5	空余	Logview
R2	GPIO6	SPI_DO	Logview
	GPIO7	SPI_CLK	AT_T
	GPIO8	空余	PWR
	GPIO9	SPI_DI	
	GPIO10	SPI_CS0	
开关信号	SPI Flash信号	SIM卡信号	UART

- 此时相对海思模组参考设计，就剩新增部分，如扩展MCU、Flash等器件；对于新增的MCU及Flash等器件，选型时请务必确认PSM态各器件的漏电指标，建议总体不超过3uA；这部分的功耗可以先在厂家的Demo板上进行开发测试，Demo一般都有预留功耗测试端口，待稳定后再做集成。

通常情况下，如果Boudica的外围设计和海思参考一致，则增加功耗的部分，很有可能有两个来源，其一就是SIM卡的供电没有做逻辑控制，其二就是增加的器件没有真正进入PSM态，重点关注MCU及Flash器件。

### 6.1.8.2 终端近端维护串口

建议外壳预留模组的串口升级接口（TX/RX/GND）以及复位按钮，如果结构件有IPxx密封要求，建议增加胶条等密封条装置。

### 6.1.8.3 终端覆盖性能排除方法

终端带天线在屏蔽柜测试下行MCL不满足164dB的问题定位思路：

1. 确认模组RF直连指标是否满足规格：模组ANT口焊接同轴线，连接仪表测试模组接收灵敏度；或ANT口同轴线通过有线方式和基站直连，确认直连MCL指标。
2. 确认天线系统效率和OTA-TIS测试规格，理论上直连MCL-天线系统效率=带天线MCL，排查测试结果是否满足理论预算。
3. 想办法将天线和电路板拉远，排除下是否存在单板对天线的干扰；如果拉远指标有改善，需要详细查找干扰源，增加滤波后者屏蔽措施。
4. 优化天线系统效率。

### 6.1.9 修订记录

表 6-4 文档修订记录

日期	版本	修订内容
2017-7-20	1.0	发布初稿
2017-8-11	1.1	<ol style="list-style-type: none"><li>1. FOTA特性设计约束刷新。</li><li>2. 增加DTLS设计约束。</li><li>3. 增加FOTA升级特性章节（6 FOTA升级特性设计），明确FOTA升级特性Device 设计要求。</li></ol>
2017-9-20	2.0	<ol style="list-style-type: none"><li>1. 增加Device硬件设计、PCB设计要求。</li><li>2. 增加Device侧FOTA特性适配建议。</li></ol>
2018-3-30	2.1	<ol style="list-style-type: none"><li>1. 修改NB应用系统架构图、终端产品硬件系统。</li><li>2. 增加终端Device产品射频接收灵敏度指标建议。</li><li>3. 增加现网调测网络覆盖建议。</li></ol>

## 6.2 LiteOS SDK 集成开发指导

### 6.2.1 LiteOS SDK 端云互通组件概述

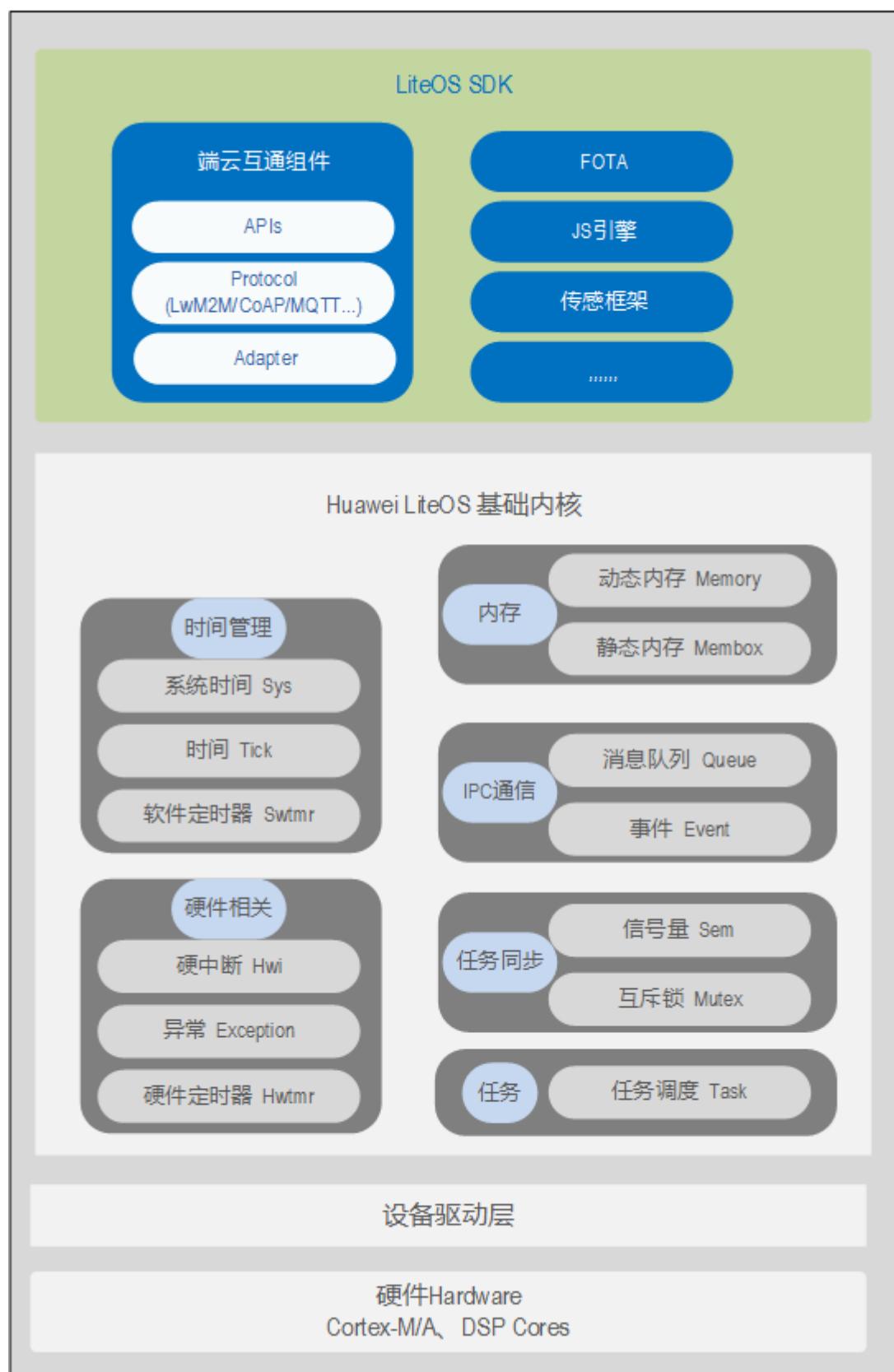
#### 6.2.1.1 背景介绍

LiteOS SDK是Huawei LiteOS软件开发工具包（Software Development Kit），包括端云互通组件、FOTA、JS引擎、传感框架等内容。

本文档介绍的LiteOS SDK包括了LiteOS SDK端云互通组件。端云互通组件是华为物联网解决方案中，资源受限终端对接到 IoT 云平台的重要组件。端云互通组件提供端云协同能力，集成了 LwM2M、CoAP、MQTT、mbedTLS、LwIP 等全套 IoT 互联互通协议栈，且在 LwM2M 的基础上，提供端云互通组件开放API，用户只需关注自身的应用，而不必关注 LwM2M 实现细节，直接使用 LiteOS SDK 端云互通组件封装的 API，通过四个步骤就能简单快速地实现与华为 OceanConnect IoT 平台安全可靠连接。使用

LiteOS SDK端云互通组件，用户可以大大减少开发周期，聚焦自己的业务开发，快速构建自己的产品。

图 6-9 Huawei LiteOS 架构图



### 6.2.1.2 系统方案

Huawei LiteOS SDK端云互通组件针对“单模组、单MCU”和“外置MCU+模组”两种应用场景，提供了不同的软件架构：

图 6-10 单模组/单 MCU 软件架构

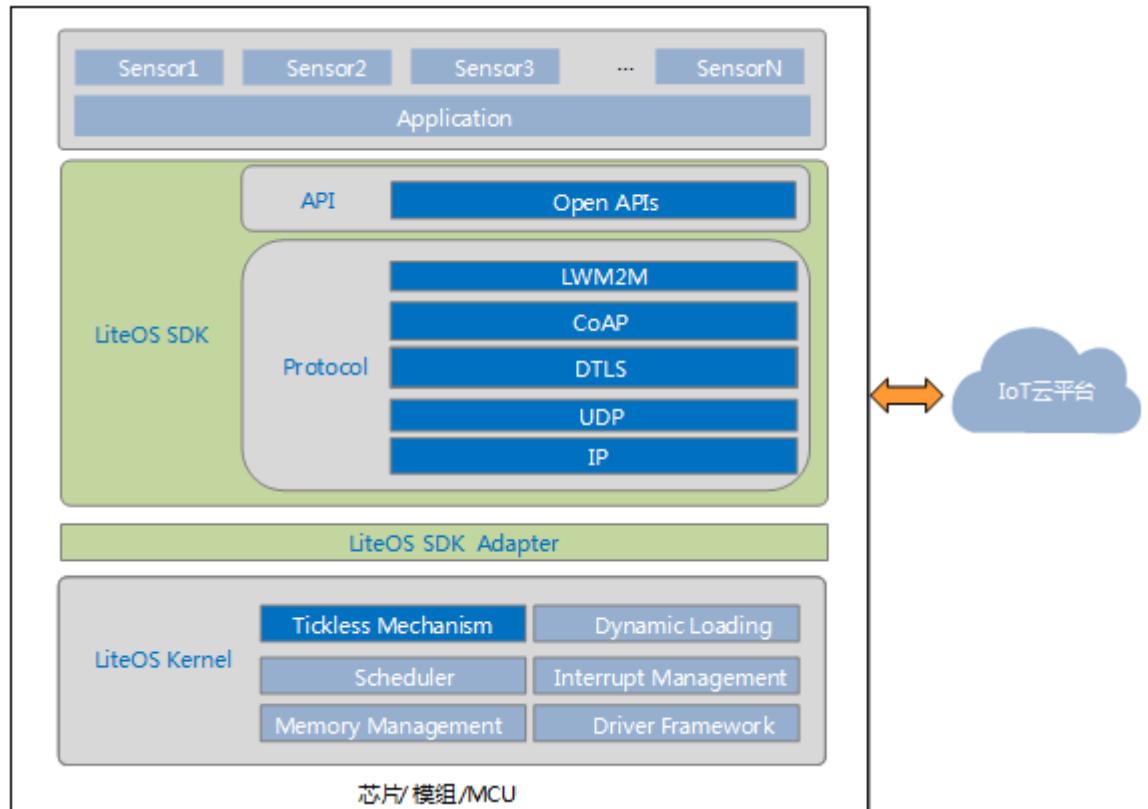
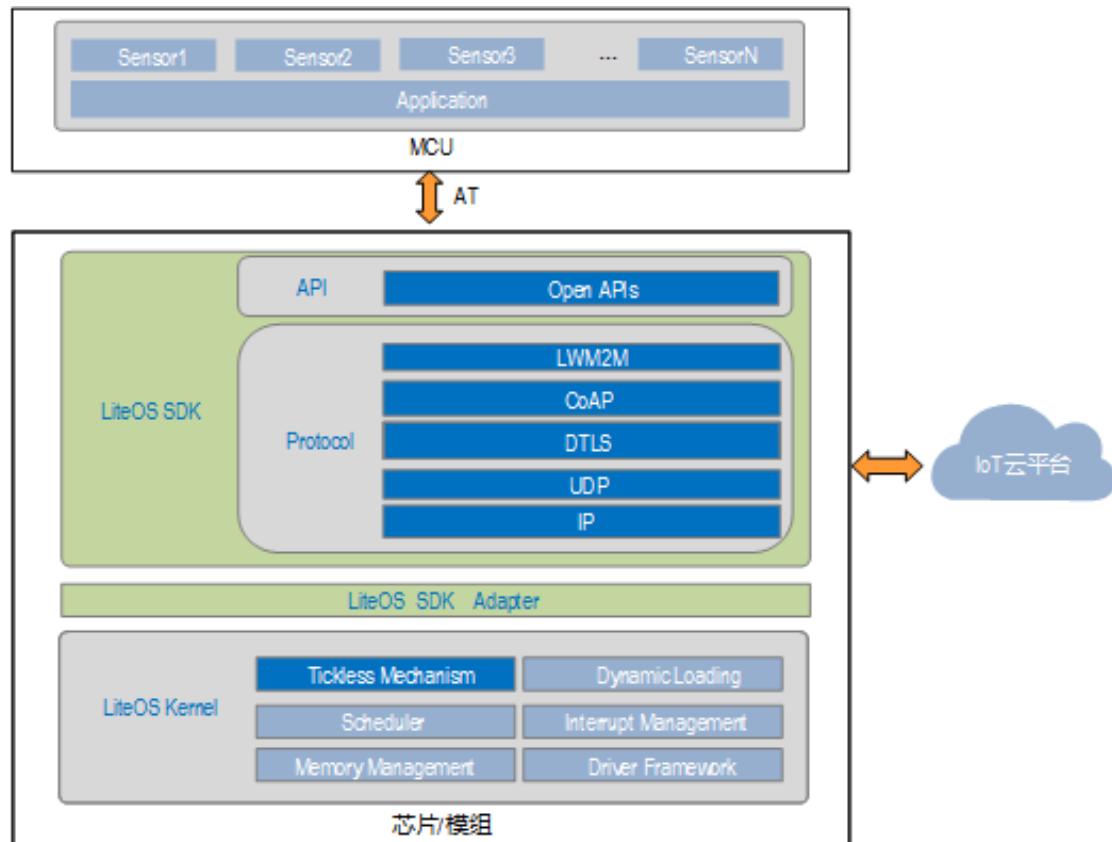


图 6-11 MCU+芯片/模组软件架构



LiteOS SDK 端云互通组件软件主要由三个层次构成：

- **开放API层：** LiteOS SDK端云互通组件的开放API为应用程序定义了通用接口，终端设备调用开放API能快速完成华为OceanConnect IoT平台的接入、业务数据上报、下发命令处理等。对于外置MCU+模组的场景，LiteOS SDK端云互通组件还提供了AT命令适配层，用于对AT命令做解析。
- **协议层：** LiteOS SDK端云互通组件集成了LwM2M/CoAP/DTLS/TLS/UDP等协议。
- **驱动及网络适配层：** LiteOS SDK端云互通组件为了方便终端设备进行集成和移植，提供了驱动及网络适配层，用户可以基于SDK提供的适配层接口列表，根据具体的硬件平台适配硬件随机数、内存管理、日志、数据存储以及网络Socket等接口。

**LiteOS基础内核：** 为用户终端设备提供RTOS特性。

### 6.2.1.3 集成策略

#### 6.2.1.3.1 可集成性

LiteOS SDK端云互通组件作为独立的组件，不依赖特定的芯片架构和网络硬件类型，可以轻松地集成到各种通信模组上，如NB-IoT模组、eMTC模组、WIFI模组、GSM模组、以太网硬件等。

### 6.2.1.3.2 可移植性

LiteOS SDK端云互通组件的Adapter层提供了常用的硬件及网络适配接口，终端或者模组厂家可以根据自己的硬件实现这些接口后，即可完成LiteOS SDK端云互通组件的移植。需要移植的接口列表及相关函数如下：

**表 6-5 LiteOS SDK 端云互通组件需要移植适配的接口列表**

接口分类	接口名	说明
网络Socket相关接口	atiny_net_connect	创建socket网络连接
	atiny_net_recv	接收函数
	atiny_net_send	发送函数
	atiny_net_recv_timeout	阻塞式接收函数
	atiny_net_close	关闭socket网络连接
硬件相关接口	atiny_gettime_ms	获取系统时间，单位ms
	atiny_usleep	延时函数，单位us
	atiny_random	硬件随机数函数
	atiny_malloc	动态内存申请
	atiny_free	动态内存释放
	atiny_snprintf	格式化字符串
	atiny_printf	日志输出
资源互斥相关接口	atiny_mutex_create	创建互斥锁
	atiny_mutex_destroy	销毁互斥锁
	atiny_mutex_lock	获取互斥锁
	atiny_mutex_unlock	释放互斥锁

#### 说明

LiteOS SDK端云互通组件支持OS方式移植，也支持无OS方式移植，推荐使用支持OS方式移植。

LiteOS SDK端云互通组件支持固件升级，需要适配atiny\_storage\_devcie\_s对象，供组件使用。

```
atiny_storage_devcie_s *atiny_get_hal_storage_device(void);  
  
struct atiny_storage_device_tag_s;  
struct atiny_storage_device_tag_s;  
typedef struct atiny_storage_device_tag_s atiny_storage_device_s;  
struct atiny_storage_device_tag_s  
{  
    //设备初始化  
    int (*init)( storage_device_s *this);  
    //准备开始写  
    int (*begin_software_download)( storage_device_s *this);  
    //写软件，从offset写，buffer为内容，长度为len
```

```
int (*write_software)( storage_device_s *this , uint32_t offset, const char *buffer, uint32_t len);

//下载结束
int (*end_software_download)( storage_device_s *this);
//激活软件
int (*active_software)( storage_device_s *this);
//得到激活的结果, 0成功, 1失败
int (*get_active_result)( storage_device_s *this);
//写update_info, 从offset写, buffer为内容, 长度为len
int (*write_update_info)( storage_device_s *this, long offset, const char *buffer, uint32_t len);
//读update_info, 从offset写, buffer为内容, 长度为len
int (*read_update_info)( storage_device_s *this, long offset, char *buffer, uint32_t len);
};
```

### 6.2.1.3.3 集成约束

LiteOS SDK端云互通组件集成需要满足一定的硬件规格：

- 要求模组/芯片有物理网络硬件支持，能支持UDP协议栈。
- 模组/芯片有足够的Flash和RAM资源供LiteOS SDK端云互通组件协议栈做集成。  
建议的硬件选型规格如下表所示：

**表 6-6 硬件选型规格建议**

RAM	Flash
>32K	>128K



#### 说明

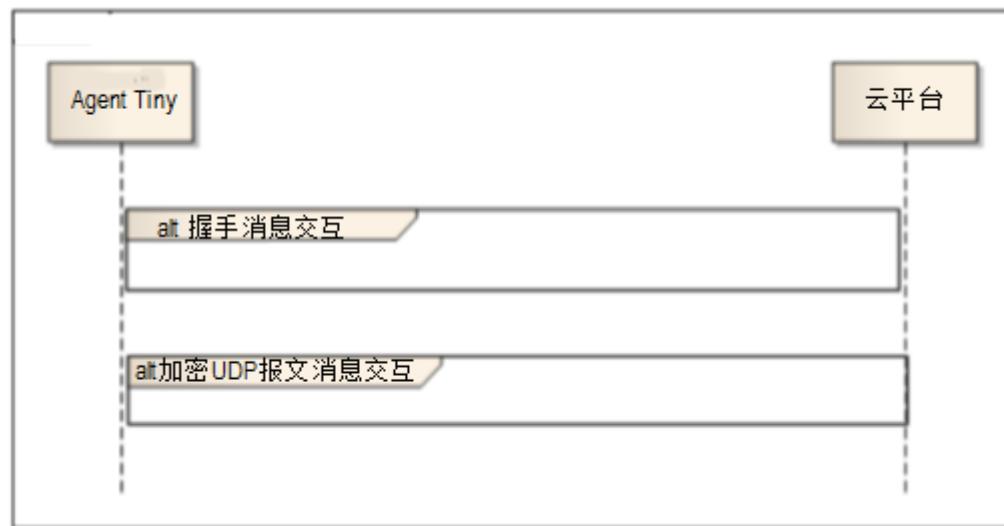
推荐的硬件选型规格考虑LiteOS SDK端云互通组件本身占用的资源（开放API+物联网协议栈+安全协议+SDK驱动及网络适配层），也考虑用户业务demo的最小实现占用的资源（芯片驱动程序、传感器驱动程序、基本业务流程等）。该规格仅为推荐规格，具体选型需要用户根据自身业务再做评估。

### 6.2.1.4 安全

LiteOS SDK端云互通组件支持DTLS(Datagram Transport Layer Security)，即数据包传输层安全性协议。目前支持PSK(Pre-Shared Keys)预共享密钥模式，后续会扩展支持其他模式。

LiteOS SDK端云互通组件首先和物联网开放平台完成握手流程，后续的应用数据将全部为加密数据，如图所示：

图 6-12 DTLS 协议交流互动



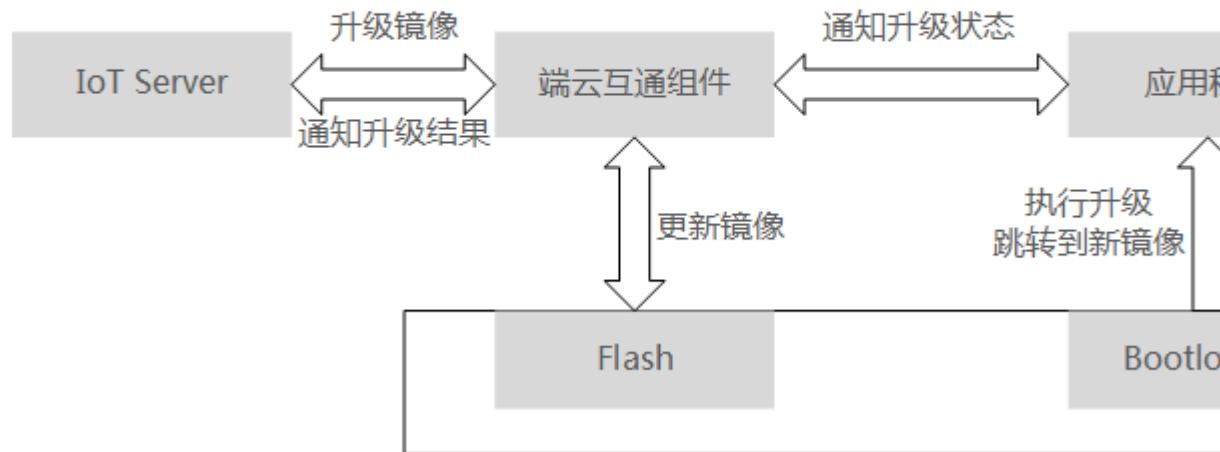
### 6.2.1.5 升级

LiteOS SDK端云互通组件支持物联网开放平台的远程固件升级，且具备断点续传、固件包完整性保护等特性。

固件升级功能和流程如图所示：

图 6-13 固件升级功能示意图

固件升级示意图



### 6.2.2 设备接入 OceanConnect 集成开发流程

本章将分别从IoT平台侧和端侧详细地阐述端云互通组件的开发流程，旨在帮助开发者在IoT设备上集成LiteOS SDK端云互通组件，进行IoT应用开发和调测。LiteOS SDK端云互通组件接入华为OceanConnect IoT云平台默认采用的是以太网方式（即以太网口驱

动+LwIP网络协议栈+LwM2M协议+LiteOS SDK端云互通组件对接云平台），同时也支持WIFI、GSM、NB-IoT等无线方式。

OceanConnect即华为IoT联接管理平台（IoT Connection Management Platform）是面向运营商和企业/行业领域的统一开放云平台，支持SIM和非SIM场景的各种联接和联接管理。通过开放的APIs，向上集成各种行业应用，向下接入各种传感器、终端和网关，帮助运营商和企业/行业客户实现多种行业终端的快速接入，多种行业应用的快速集成。华为IoT联接管理平台提供安全可控的全联接管理，使能行业革新，构建IoT生态（本章中提到的IoT平台指OceanConnect）。

## 6.2.2.1 云侧配置流程

### 6.2.2.1.1 环境准备

在开发之前，需要提前获取如下信息：

- 开发者Portal的访问地址/账号/密码，需要向OceanConnect IoT平台申请
- 设备对接地址/端口号

### 6.2.2.1.2 创建应用

通过创建应用，开发者可以根据自身应用的特征，选择不同的平台服务套件，降低应用开发难度。

- 步骤1** 登录OceanConnect IoT平台的开发者Portal。开发者Portal的访问地址、账号和密码需要向IoT平台服务商申请。
- 步骤2** 登录界面会跳出弹框，提示“当前账号没有应用！请先创建应用！”，点击“创建应用”。

图 6-14 创建应用



- 步骤3** 在新弹出窗口中，配置应用信息，点击“确定”。

配置示例如下图，点击“确定”后，IoT平台会返回应用ID和应用密钥，请妥善保存应用密钥，以便于应用服务器接入平台时使用。如果遗忘密钥，需要通过“对接信息”->“重置密钥”进行重置。

图 6-15 配置应用

**创建应用** X

\* 应用名称

\* 所属行业

\* 关联API包

\* 应用能力

确定 取消

 **说明**

如上配置仅为参考举例，具体配置请以现网需求为准。

图 6-16 应用创建成功



----结束

### 6.2.2.1.3 开发 Profile 文件

Profile文件用来描述设备类型和设备服务能力。它定义了设备具备的服务能力，每个服务具备的属性、命令以及命令参数。

**步骤1** 登录IoT平台的开发者Portal。开发者Portal的访问地址、账号和密码需要向IoT平台服务商申请。

**步骤2** 选择“Profile开发”->“Profile在线开发”->“自定义产品”，点击右上角“+创建全新产品”。

IoT平台提供了Profile模板库，开发者可以根据自己需要，选择合适的模板直接使用。如果在模板库中未找到需要的Profile，再自己定义，示例如下。

图 6-17 创建 Profile 文件

创建全新产品 X

设备类型 \* ▼

Other  
LedLight

设备型号 \* ▼

Device001

厂商ID \* ▼

Huawei

厂商名称 \* ▼

Huawei

协议类型 \* ▼

LWM2M

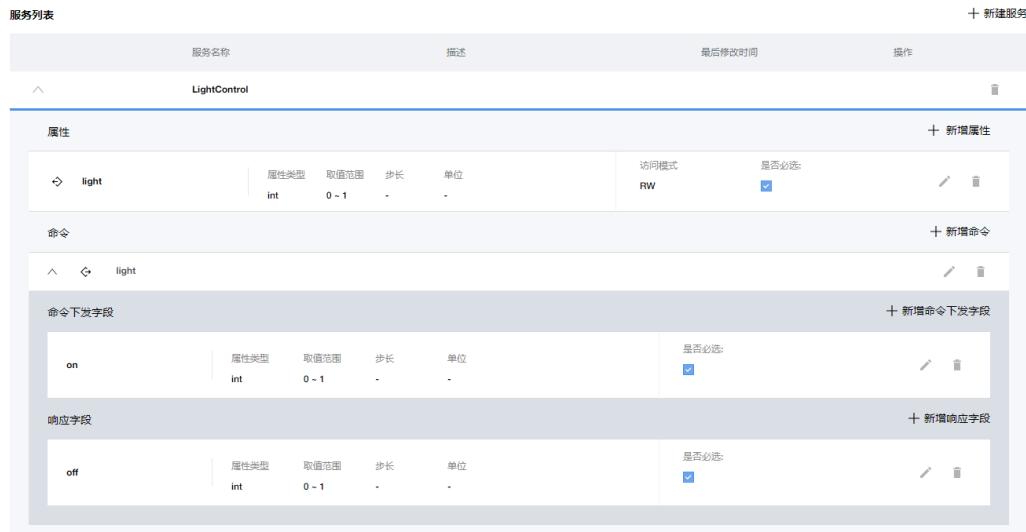
 说明

如上配置仅为参考举例，具体配置请以现网需求为准。

**步骤3** 选择新创建的Profile文件，点击“新建服务”，配置设备的服务能力。

可参考“Profile开发”->“Profile在线开发”中的“产品模板”进行配置。例如新建一个名为LightControl的服务，包含一种属性light（灯的亮灭状态）和一种命令（设置灯亮on或者灭off）

图 6-18 新建 LightControl 服务



**步骤4** (可选) 开发者Portal提供了Profile文件的导出功能。

选择“Profile开发”->“Profile在线开发”->新创建的Profile文件，点击右上角“导出该产品Profile”，可以对线上开发的Profile文件进行导出。

图 6-19 导出 Profile 文件



----结束

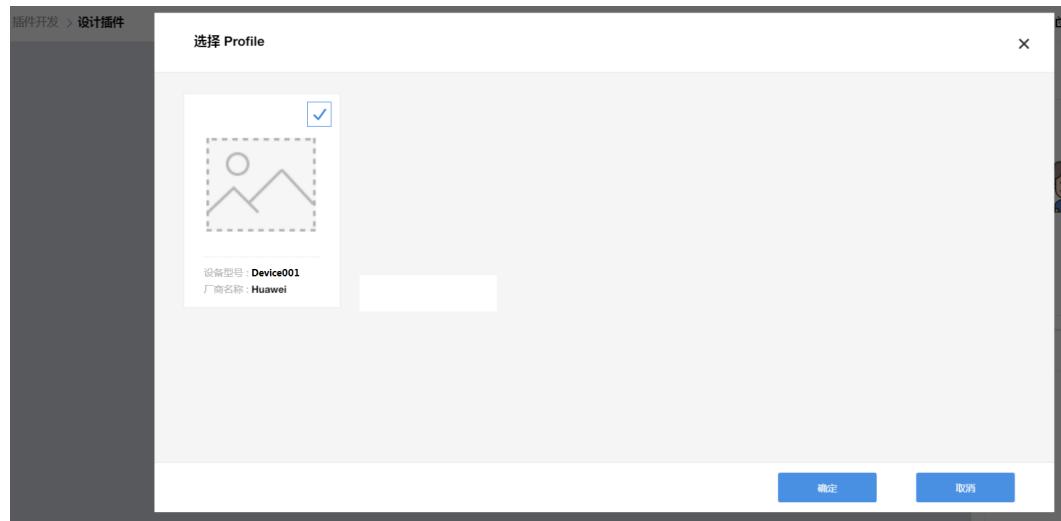
#### 6.2.2.1.4 开发编解码插件

IoT设备和IoT平台之间采用LwM2M协议通信，LwM2M消息的数据为应用层数据，应用层数据的格式由设备厂商自行定义。由于IoT设备对省电要求较高，所以应用层数据一般采用二进制格式。IoT平台在对应用层数据进行协议解析时，会转换成统一的json格式，以方便应用服务器使用。要实现二进制消息与json格式消息的转换，IoT平台需要使用编解码插件。

**步骤1** 选择“插件开发”->“插件开发”->“开始设计”，点击右上角“+新建插件”。在弹出框中，选择Profile文件。

IoT平台提供了插件模板库，开发者可以根据自己需要，选择合适的模板直接使用。如果在模板库中未找到需要的插件，再自己定义。

图 6-20 创建插件



**步骤2** 点击“新增消息”，配置二进制码流和Profile属性/命令/命令响应的映射关系。

可参考“插件开发”->“插件开发”->“开始设计”中的“新手指导”和“插件模板”进行配置。

图 6-21 开发插件（新建数据上报消息）

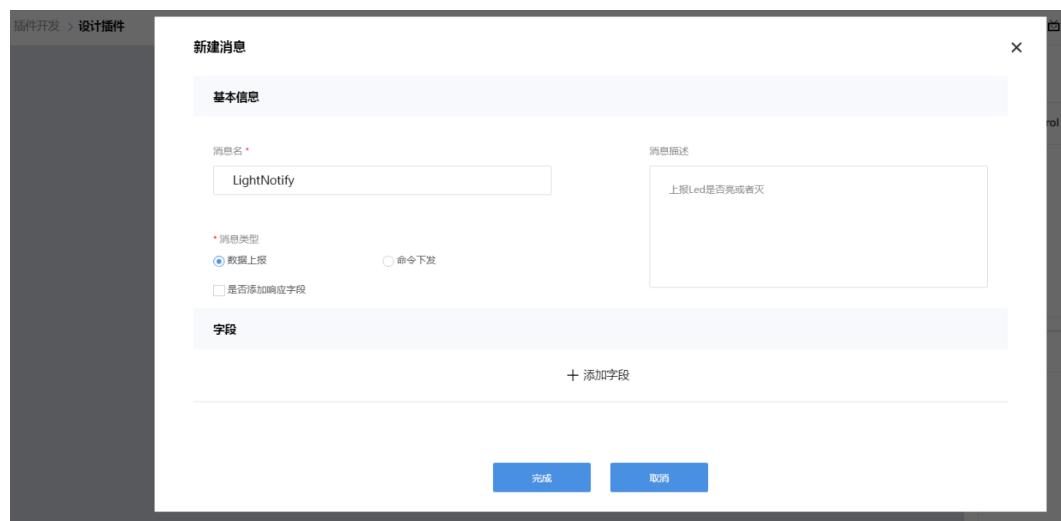


图 6-22 开发插件（添加字段）

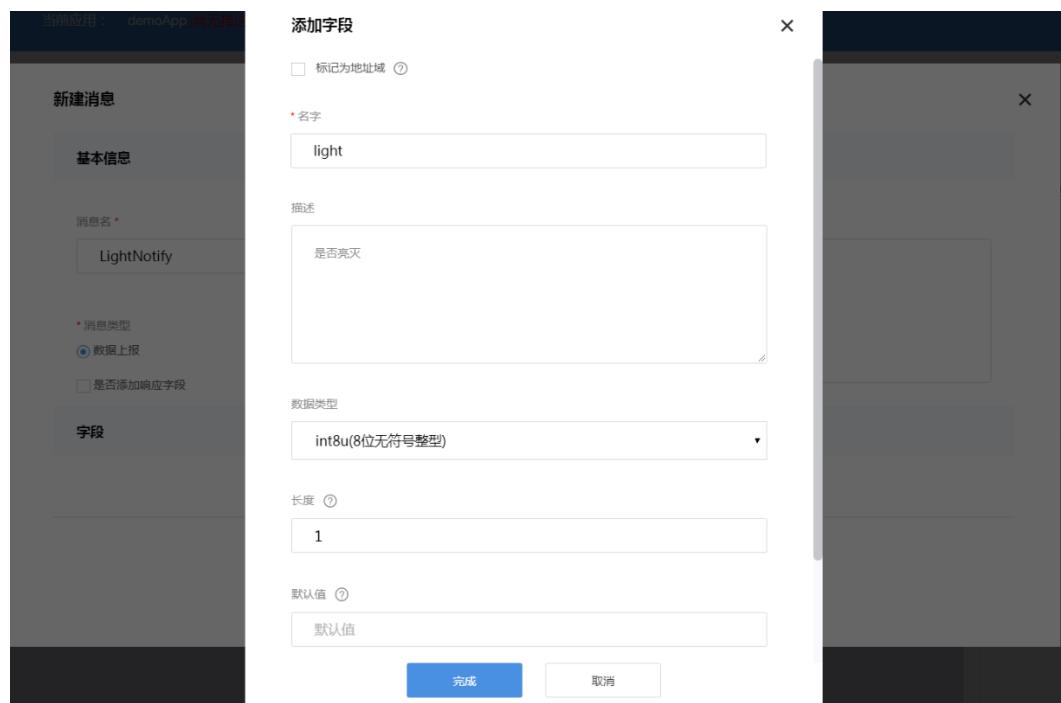


图 6-23 开发插件（新建命令下发消息）

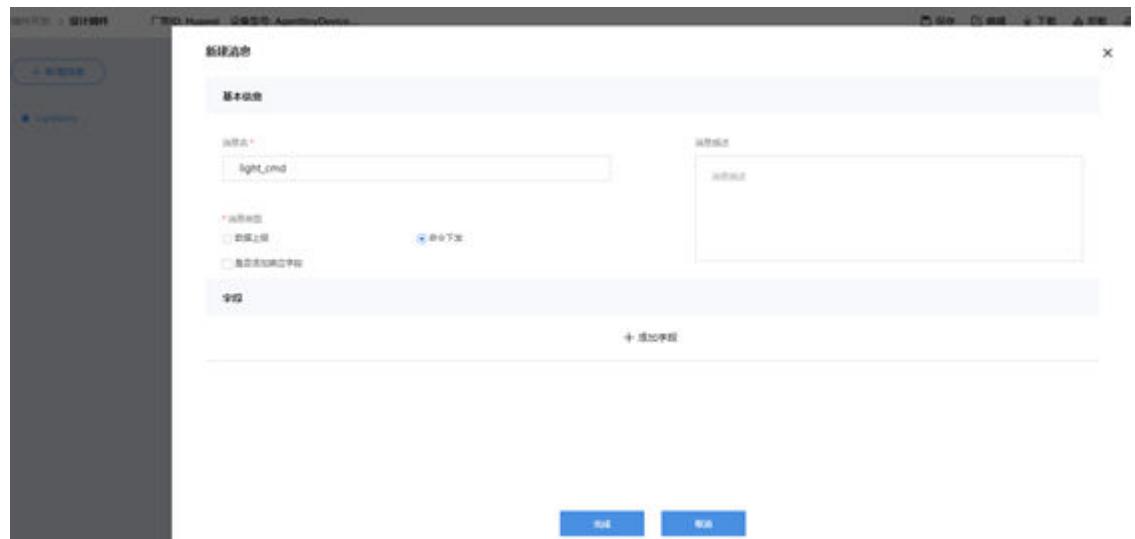
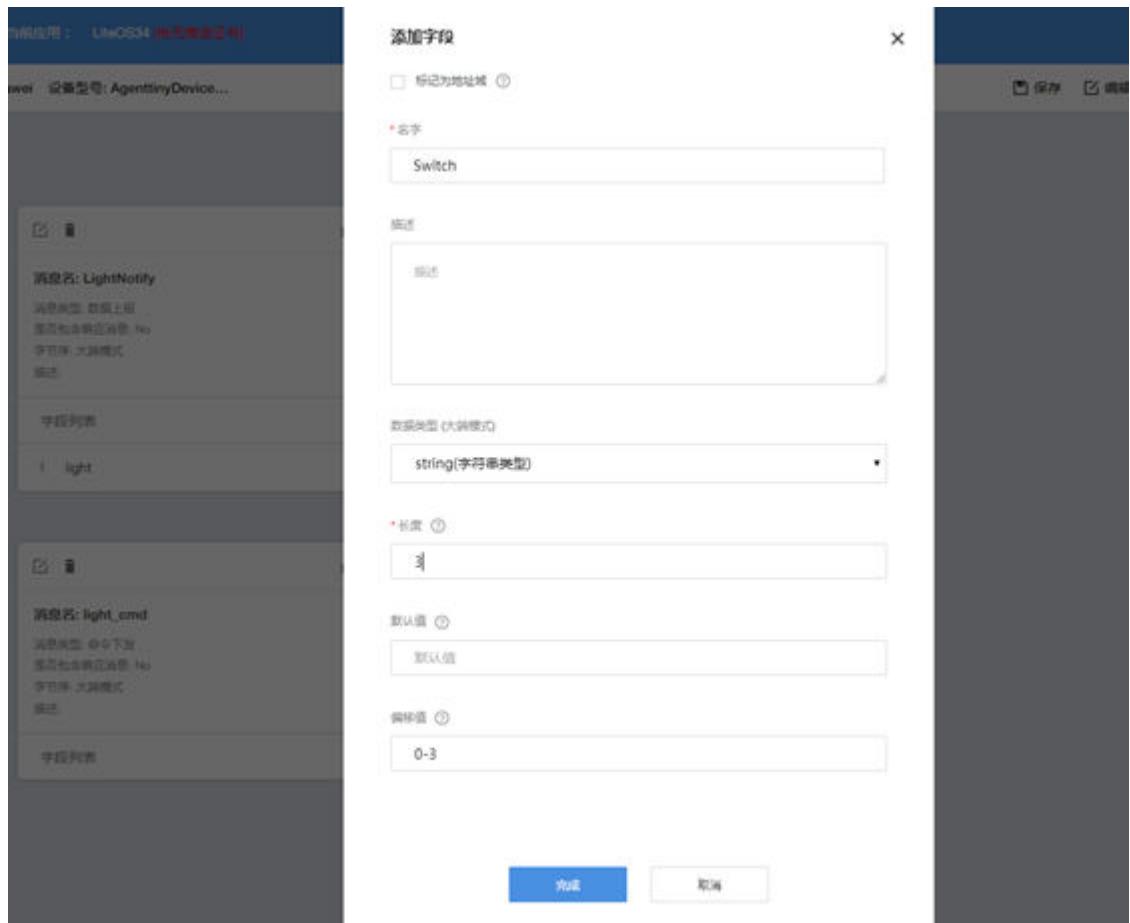


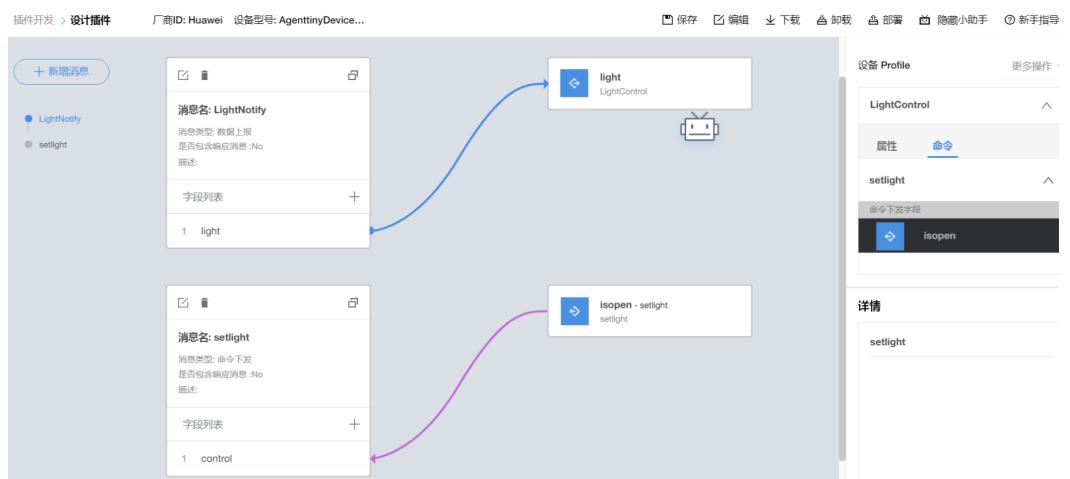
图 6-24 开发插件（添加字段）



编解码插件的开发，即定义：

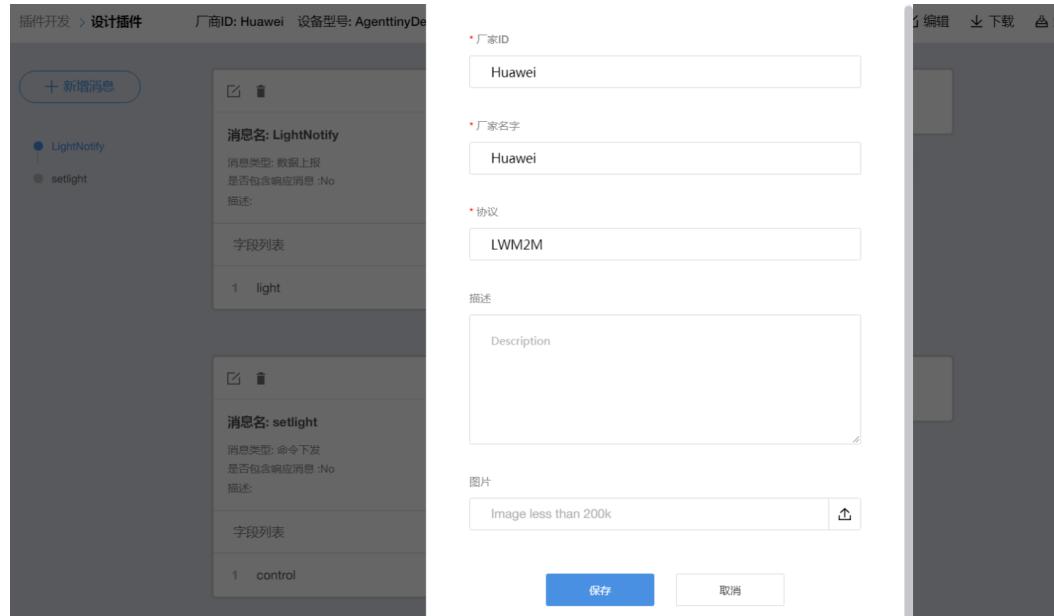
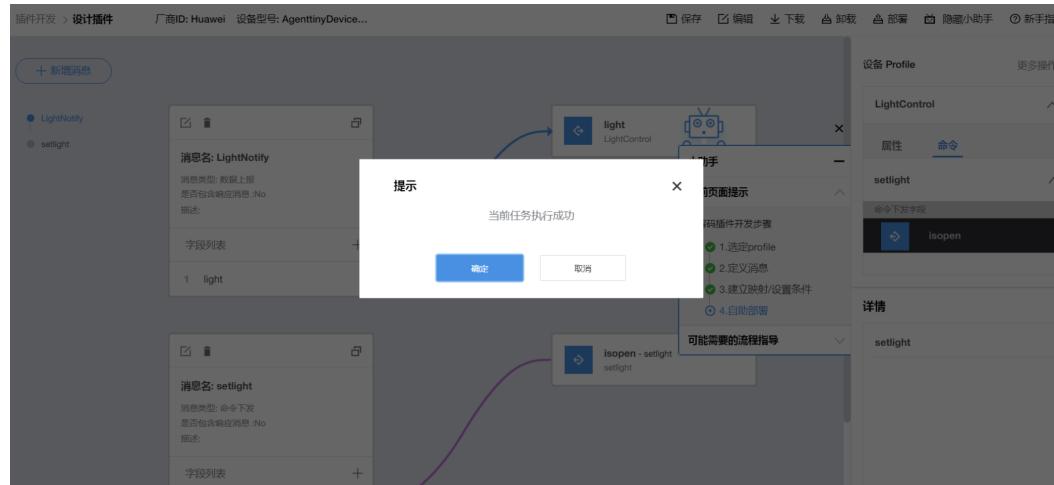
- Profile文件定义的属性/响应在设备上报的二进制码流中的位置，以便于平台对设备上报数据和命令响应进行解码。
- Profile文件定义的命令在平台下发的二进制码流中的位置，以便于平台对下发命令进行编码。

图 6-25 二进制码流和 Profile 文件的映射关系



**步骤3** 点击右上角“部署”。

点击部署后，需要先“保存”插件，之后才开始部署。部署需要等待时间小于60s。

**图 6-26 保存插件****图 6-27 部署插件****步骤4** (可选) 开发者Portal提供了编解码插件的下载功能。

选择“插件开发”->“插件开发”->新开发的编解码插件，点击右上角“下载”，可以对线上开发的编解码插件进行导出。

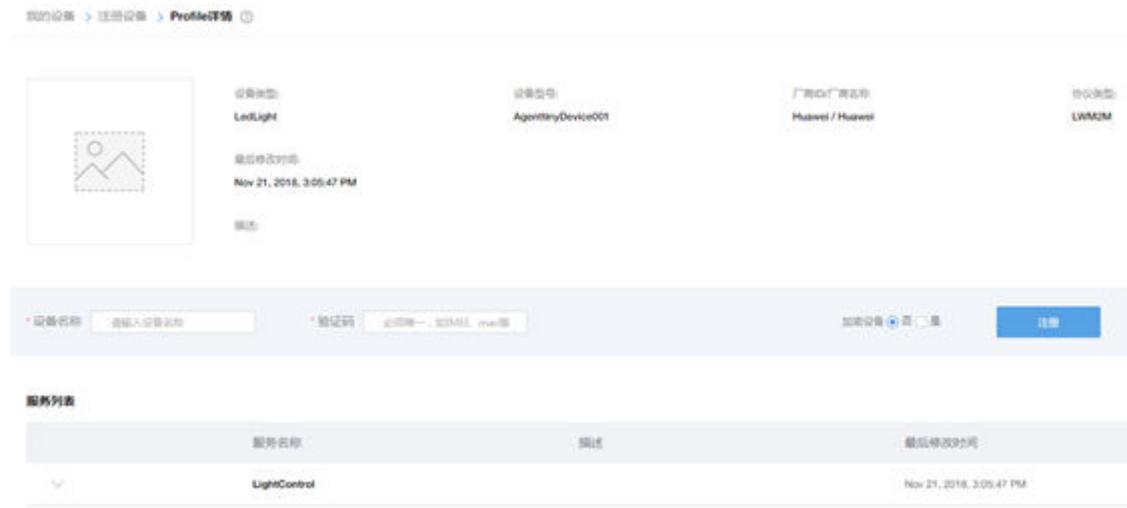
----结束

### 6.2.2.1.5 注册设备

应用服务器需要调用IoT平台的注册设备接口，在IoT平台添加设备。

**步骤1** 选择“我的设备”->“注册设备”->“需要注册设备的Profile”，输入设备名称和验证码（verifyCode），并根据业务需求选择是否加密设备。最后点击“注册”。

图 6-28 需要注册设备的 Profile



注册设备后，IoT平台会返回设备ID和PSK码，请妥善保存。新增注册的设备状态为“未绑定（not bound）”。

图 6-29 注册设备

### 注册设备成功

请根据设备指导说明书为设备接通电源，配置好网络，开启设备，观察设备是否成功接入到平台。如果状态是在线（online）表示设备已经成功的接入到平台，接着就可以接收设备的数据。

以下是您的设备信息，请牢记

#### 设备ID

84150fc1-13a0-4373-a9ba-0644f1d5f06c

PSK码 (使用DTLS协议时需要使用到该psk码，请您牢记!)

f95722886f70a228153e98322c45eb6d

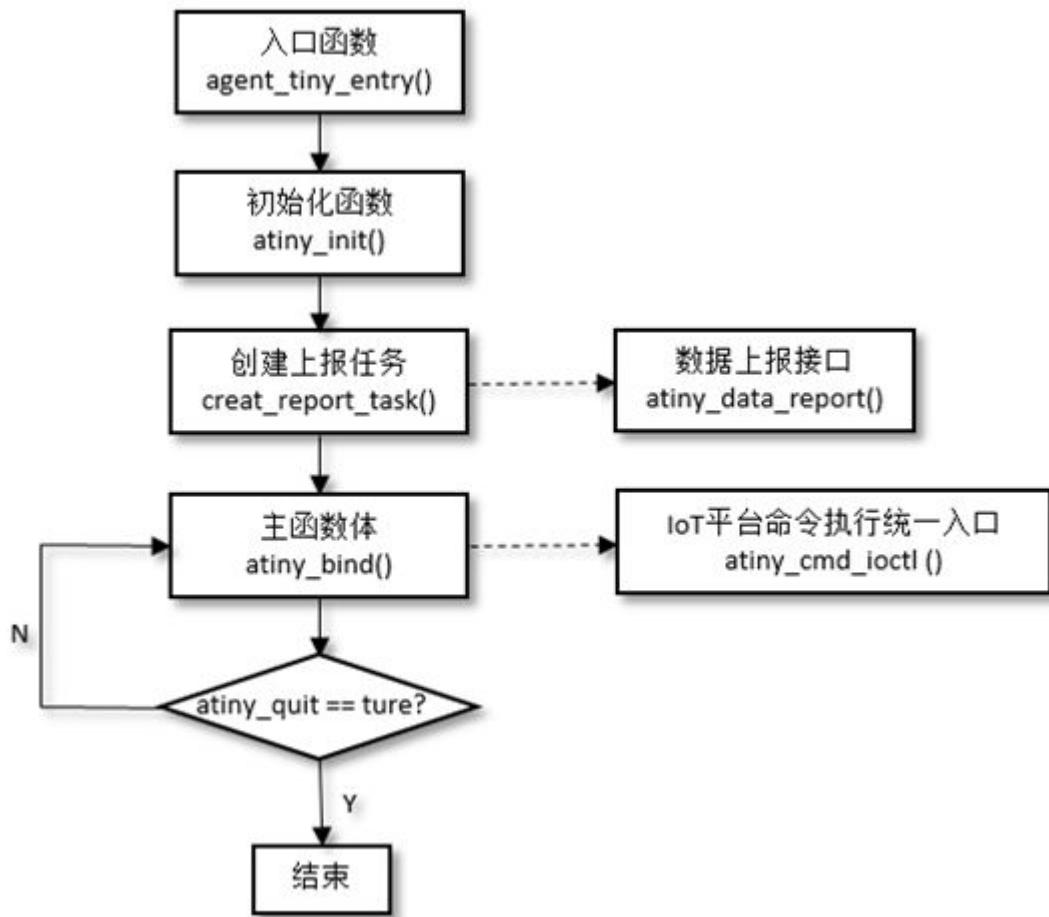


----结束

### 6.2.2.2 端侧对接流程

设备接入IoT平台后，IoT平台才可以对设备进行管理。设备接入平台时，需要保证IoT平台已经有对应应用，并且已经在该应用下注册了此设备。本节介绍端侧设备是如何通过端云互通组件与IoT平台实现对接的。首先给出端侧设备对接IoT平台的整体示意图。

图 6-30 端侧设备对接 IoT 平台的整体示意图



本小节将根据上图所示的流程，向开发者介绍终端设备是如何一步步地接入IoT平台，并进行数据上报与命令执行的。

#### 6.2.2.1 环境准备

在开发之前，需要提前获取如下信息：

- Huawei LiteOS及LiteOS SDK源代码。工程整体结构如下。

```
|--- arch //架构相关文件  
|   |--- arm  
|   |--- msp430  
|--- build
```

```
| └── Makefile
| └── components //LiteOS各类组件
|   | └── connectivity
|   | └── fs
|   | └── lib
|   | └── log
|   | └── net
|   | └── ota
|   | └── security
| └── demos //示例程序
|   | └── agenttiny_lwm2m //本章中列出的所有示例程序，均来自该目录下的
|   |   agent_tiny_demo.c文件
|   | └── agenttiny_mqtt
|   | └── dtls_server
|   | └── fs
|   | └── kernel
|   | └── nbiot_without_atiny
| └── doc //说明文档
|   | └── Huawei_LiteOS_Developer_Guide_en.md
|   | └── Huawei_LiteOS_Developer_Guide_zh.md
|   | └── Huawei_LiteOS_SDK_Developer_Guide.md
|   | └── LiteOS_Code_Info.md
|   | └── LiteOS_Commit_Message.md
|   | └── LiteOS_Contribute_Guide_GitGUI.md
|   | └── LiteOS_Supported_board_list.md
| └── meta
| └── include //工程需要的头文件
|   | └── at_device
|   | └── at_frame
|   | └── atiny_lwm2m
|   | └── atiny_mqtt
|   | └── fs
|   | └── log
```

```
|   └── nb_iot
|   └── osdepends
|   └── ota
|   └── sal
|   └── sota
├── kernel //系统内核
|   └── base
|   └── extended
|   └── include
|   └── los_init.c
|   └── Makefile
├── LICENSE //许可
├── osdepends //依赖项
|   └── liteos
└── README.md
├── targets //BSP工程
|   └── Cloud_STM32F429IGTx_FIRE
|   └── Mini_Project
|   └── NXP_LPC51U68
|       └── STM32F103VET6_NB_GCC
└── tests //测试用例
    └── cmockery
    └── test_agenttiny
    └── test_main.c
    └── test_sota
    └── test_suit
```

源代码托管在GitHub，地址为<https://github.com/LiteOS/LiteOS>。

- 集成开发工具：

- MDK 5.18版本或者以上版本，从MDK官方网站下载。
- MDK依赖的pack包



#### 说明

MDK工具需要license，请从MDK官方获取。

### 6.2.2.2.2 LiteOS SDK 端云互通组件入口函数

使用LiteOS SDK端云互通组件agent tiny对接IoT平台，首先需要一个入口函数agent\_tiny\_entry()。

接口名	描述
void agent_tiny_entry(void)	LiteOS SDK端云互通组件的入口函数。 该接口将进行agent tiny的初始化相关操作，创建上报任务，并调用agent tiny主函数体。 参数列表：空 返回值：空

开发者可以通过LiteOS内核提供的任务机制，创建一个主任务main\_task。在主任务中调用入口函数agent\_tiny\_entry()，开启agent tiny工作流程。

```
UINT32 creat_main_task()
{
    UINT32 uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;
    task_init_param.usTaskPrio = 0;
    task_init_param.pcName = "main_task";
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)main_task;
    task_init_param.uwStackSize = 0x1000;
    uwRet = LOS_TaskCreate(&g_TskHandle, &task_init_param);
    if(LOS_OK != uwRet)
    {
        return uwRet;
    }
    return uwRet;
}
```

### 6.2.2.2.3 LiteOS SDK 端云互通组件初始化

在入口函数中，需要调用atiny\_init()进行agent tiny的初始化相关操作。

接口名	描述
int atiny_init(atiny_param_t* atiny_params, void** phandle)	LiteOS SDK端云互通组件的初始化接口，由LiteOS SDK端云互通组件实现，设备调用。 参数列表：参数atiny_params为入参，包含初始化操作所需的各个变量，具体请参考服务器参数结构体atiny_param_t；参数phandle为出参，表示当前创建的agent tiny的句柄。 返回值：整形变量，标识初始化成功或失败的状态。

对于入参atiny\_params的设定，要根据具体的业务来进行。开发者可以参考下面的代码。

```
#ifdef CONFIG_FEATURE_FOTA
    hal_init_ota(); //若定义FOTA功能，则需进行FOTA相关初始化
#endif

#ifndef WITH_DTLS
    device_info->endpoint_name = g_endpoint_name_s; //加密设备验证码
#else
    device_info->endpoint_name = g_endpoint_name; //非加密设备验证码
#endif
#ifndef CONFIG_FEATURE_FOTA
    device_info->manufacturer = "Lwm2mFota"; //设备厂商
    device_info->dev_type = "Lwm2mFota"; //设备类型
#else
    device_info->manufacturer = "Agent_Tiny";
#endif
atiny_params = &g_atiny_params;
atiny_params->server_params.binding = "UQ"; //绑定方式
atiny_params->server_params.life_time = 20; //生命周期
atiny_params->server_params.storing_cnt = 0; //缓存数据报文个数

atiny_params->server_params.bootstrap_mode = BOOTSTRAP_FACTORY; //引导模式
atiny_params->server_params.hold_off_time = 10; //等待时延

//pay attention: index 0 for iot server, index 1 for bootstrap server.
iot_security_param = &(atiny_params->security_params[0]);
bs_security_param = &(atiny_params->security_params[1]);

iot_security_param->server_ip = DEFAULT_SERVER_IPV4; //服务器地址
bs_security_param->server_ip = DEFAULT_SERVER_IPV4;

#ifndef WITH_DTLS
    iot_security_param->server_port = "5684"; //加密设备端口号
    bs_security_param->server_port = "5684";

    iot_security_param->psk_Id = g_endpoint_name_iots; //加密设备验证码
    iot_security_param->psk = (char *)g_psk_iot_value; //PSK码
    iot_security_param->psk_len = sizeof(g_psk_iot_value); //PSK码长度

    bs_security_param->psk_Id = g_endpoint_name_bs;
    bs_security_param->psk = (char *)g_psk_bs_value;
    bs_security_param->psk_len = sizeof(g_psk_bs_value);
#else
    iot_security_param->server_port = "5683"; //非加密设备端口号
    bs_security_param->server_port = "5683";

    iot_security_param->psk_Id = NULL; //非加密设备，无需PSK相关参数设置
    iot_security_param->psk = NULL;
    iot_security_param->psk_len = 0;

    bs_security_param->psk_Id = NULL;
    bs_security_param->psk = NULL;
    bs_security_param->psk_len = 0;
#endif
```

设定好atiny\_params后，即可根据设定的参数对agent tiny进行初始化。

```
if(ATINY_OK != atiny_init(atiny_params, &g_phandle))
{
    return;
}
```

对于初始化接口atiny\_init()内部，主要进行入参合法性的检验，agent tiny所需资源的创建等工作，一般不需要开发者进行修改。

### 6.2.2.2.4 创建数据上报任务

在完成agent tiny的初始化后，需要通过调用creat\_report\_task()创建一个数据上报的任务app\_data\_report()。

```
UINT32 creat_report_task()
{
    UINT32 uwRet = LOS_OK;
    TSK_INIT_PARAM_S task_init_param;
    UINT32 TskHandle;
    task_init_param.usTaskPrio = 1;
    task_init_param.pcName = "app_data_report";
    task_init_param.pfnTaskEntry = (TSK_ENTRY_FUNC)app_data_report;
    task_init_param.uwStackSize = 0x400;
    uwRet = LOS_TaskCreate(&TskHandle, &task_init_param);
    if(LOS_OK != uwRet)
    {
        return uwRet;
    }
    return uwRet;
}
```

在app\_data\_report()中应该完成对数据上报数据结构data\_report\_t的赋值，包括数据缓冲区地址buf，收到平台ack响应后的回调函数callback，数据cookie，数据长度len，以及数据上报类型type（在这里固定为APP\_DATA）。

```
uint8_t buf[5] = {0, 1, 6, 5, 9};
data_report_t report_data;
int ret = 0;
int cnt = 0;
report_data.buf = buf;
report_data.callback = ack_callback;
report_data.cookie = 0;
report_data.len = sizeof(buf);
report_data.type = APP_DATA;
```

完成对report\_data的赋值后，即可通过接口atiny\_data\_report()上报数据。

接口名	描述
int atiny_data_report(void* phandle, data_report_t* report_data)	LiteOS SDK端云互通组件数据上报接口，由LiteOS SDK端云互通组件实现，设备调用，设备应用数据使用该接口上报。该接口为阻塞接口，不允许在中断中使用。 参数列表：参数phandle为调用初始化接口atiny_init()得到的agent tiny的句柄；参数report_data为数据上报数据结构。 返回值：整形变量，标识数据上报成功或失败的状态。

示例代码中的上报任务实现方法如下。

```
while(1)
{
    report_data.cookie = cnt;
    cnt++;
    ret = atiny_data_report(g_phandle, &report_data); //数据上报接口
    ATINY_LOG(LOG_DEBUG, "data report ret: %d\n", ret);
```

```
    (void)LOS_TaskDelay(250 * 8);  
}
```

### 6.2.2.5 LiteOS SDK 端云互通组件命令处理接口

IoT平台下发的各类命令，都通过接口atiny\_cmd\_ioctl()来具体执行。

接口名	描述
int atiny_cmd_ioctl(atiny_cmd_e cmd, char* arg, int len);	<p>LiteOS SDK端云互通组件申明和调用，由开发者实现。该接口是LwM2M标准对象向设备下发命令的统一入口。</p> <p>参数列表：参数cmd为具体命令字，比如下发业务数据，下发复位，升级命令等；参数arg为存放命令参数的缓存；参数len为缓存大小。</p> <p>返回值：空。</p>

atiny\_cmd\_ioctl()是LiteOS SDK端云互通组件定义的一个通用可扩展的接口，其命令字如atiny\_cmd\_e所定义，用户根据自身需求进行选择性实现，也可以根据自身需求进行扩展。常用的接口定义如下表所示，每一个接口都和atiny\_cmd\_e的枚举值一一对应：

回调接口函数	描述
int atiny_get_manufacturer(char* manufacturer,int len)	获取厂商名字，参数manufacturer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的厂商名字，长度不能超过参数len。
int atiny_get_dev_type(char * dev_type,int len)	获取设备类型，参数dev_type指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备类型，长度不能超过参数len。
int atiny_get_model_number((char * model_numer, int len)	获取设备模型号，参数model_numer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备模型号，长度不能超过参数len。
int atiny_get_serial_number(char* num,int len)	获取设备序列号，参数numer指向的内存由LiteOS SDK端云互通组件分配，户填充自身的设备序列号，长度不能超过参数len。
int atiny_get_dev_err(int* arg, int len)	获取设备状态，比如内存耗尽、电池不足、信号强度低等，参数arg由LiteOS SDK端云互通组件分配，用户填充，长度不能超过len。
int atiny_do_dev_reboot(void)	设备复位。
int atiny_do_factory_reset(void)	厂商复位。
int atiny_get_battery_level(int* voltage)	获取电池剩余电量。
int atiny_get_memory_free(int* size)	获取空闲内存大小。

int atiny_get_total_memory(int* size)	获取总共内存大小。
int atiny_get_signal_strength(int* singal_strength)	获取信号强度。
int atiny_get_cell_id(long* cell_id)	获取小区ID。
int atiny_get_link_quality(int* quality)	获取信道质量。
int atiny_write_app_write(void* user_data, int len)	业务数据下发。
int atiny_update_psk(char* psk_id, int len)	预置共享密钥更新。

其中，开发者需要根据自身的业务，在接口atiny\_write\_app\_write()中实现自己的命令响应。

```
int atiny_write_app_write(void* user_data, int len)
{
    (void)atiny_printf("write num19 object success\r\n");
    return ATINY_OK;
}
```

### 6.2.2.2.6 LiteOS SDK 端云互通组件主函数体

完成了数据上报任务的创建与命令处理接口的实现，agent tiny进入到对接IoT平台的核心步骤atiny\_bind()。

接口名	描述
int atiny_bind(atiny_device_info_t* device_info, void* phandle)	LiteOS SDK端云互通组件的主函数体，由LiteOS SDK端云互通组件实现，设备调用，调用成功后，不会返回。该接口是LiteOS SDK端云互通组件主循环体，实现了LwM2M协议处理，注册状态机，重传队列，订阅上报。 参数列表：参数device_info为终端设备参数结构体；参数phandle为调用初始化接口atiny_init()得到的agent tiny的句柄。 返回值：整形变量，标识LiteOS SDK端云互通组件主函数体执行的状态。只有执行失败或者调用了LiteOS SDK端云互通组件去初始化接口atiny_deinit()才会返回。

atiny\_bind()会根据LwM2M协议标准，进行LwM2M客户端创建与注册，并将数据上报任务app\_data\_report()中上报的数据递交给通信模块发送到IoT平台，同时接受IoT平台下发的命令消息，解析后由命令处理接口atiny\_cmd\_ioctl()统一进行处理。与atiny\_init()一样，atiny\_bind()内部一般不需要开发者进行修改。

说明：关于LwM2M协议相关内容，请开发者参考附录。

LiteOS SDK端云互通组件通过主函数体，不断地进行数据上报与命令处理。当调用LiteOS SDK端云互通组件去初始化接口atiny\_deinit()时，退出主函数体。

接口名	描述
void atiny_deinit(void* phandle);	LiteOS SDK端云互通组件的去初始化接口，由LiteOS SDK端云互通组件实现，设备调用。该接口为阻塞式接口，调用该接口时，会直到agent tiny主任务退出，资源释放完毕，该接口才会退出。 参数列表：参数phandle为调用atiny_init()获取到的LiteOS SDK端云互通组件句柄。 返回值：空

### 6.2.2.7 数据结构介绍

- 平台下发命令枚举类型

```
typedef enum
{
    ATINY_GET_MANUFACTURER, /*获取厂商名字*/
    ATINY_GET_MODEL_NUMBER, /*获取设备模型，由厂商定义和使用*/
    ATINY_GET_SERIAL_NUMBER, /*获取设备序列号*/
    ATINY_GET_FIRMWARE_VER, /*获取固件版本号*/
    ATINY_DO_DEV_REBOOT, /*下发设备复位命令*/
    ATINY_DO_FACTORY_RESET, /*厂商复位*/
    ATINY_GET_POWER_SOURCE, /*获取电源*/
    ATINY_GET_SOURCE_VOLTAGE, /*获取设备电压*/
    ATINY_GET_POWER_CURRENT, /*获取设备电流*/
    ATINY_GET_BATTERY_LEVEL, /*获取电池剩余电量*/
    ATINY_GET_MEMORY_FREE, /*获取空闲内存*/
    ATINY_GET_DEV_ERR, /*获取设备状态，比如内存耗尽、电池不足等*/
    ATINY_DO_RESET_DEV_ERR, /*获取设备复位状态*/
    ATINY_GET_CURRENT_TIME, /*获取当前时间*/
    ATINY_SET_CURRENT_TIME, /*设置当前时间*/
    ATINY_GET_UTC_OFFSET, /*获取UTC时差*/
    ATINY_SET_UTC_OFFSET, /*设置UTC时差*/
    ATINY_GET_TIMEZONE, /*获取时区*/
    ATINY_SET_TIMEZONE, /*设置时区*/
    ATINY_GET_BINDING_MODES, /*获取绑定模式*/
    ATINY_GET_FIRMWARE_STATE, /*获取固件升级状态*/
    ATINY_GET_NETWORK_BEARER, /*获取网络通信承载类型，比如GSM、WCDMA等*/
    ATINY_GET_SIGNAL_STRENGTH, /*获取网络信号强度*/
    ATINY_GET_CELL_ID, /*获取网络小区ID*/
    ATINY_GET_LINK_QUALITY, /*获取网络链路质量*/
    ATINY_GET_LINK_UTILIZATION, /*获取网络链路利用率*/
    ATINY_WRITE_APP_DATA, /*业务数据下发命令字*/
    ATINY_UPDATE_PSK, /*更新psk命令字*/
    ATINY_GET_LATITUDE, /*获取设备所处纬度*/
    ATINY_GET_LONGITUDE, /*获取设备所处经度*/
    ATINY_GET_ALTITUDE, /*获取设备所处高度*/
    ATINY_GET_SPEED, /*获取设备运行速度*/
    ATINY_GET_TIMESTAMP, /*获取时间戳*/
} atiny_cmd_e;
```

- 关键事件枚举类型

该枚举类型用于LiteOS SDK端云互通组件把自身状态通知用户

```
typedef enum
{
    ATINY_REG_OK, /*设备注册成功*/
    ATINY_REG_FAIL, /*设备注册失败*/
}
```

```
    ATINY_DATA_SUBSCRIBLE,      /*数据开始订阅，设备侧允许上报数据 */
    ATINY_DATA_UNSUBSCRIBLE,    /*数据取消订阅，设备侧停止上报数据*/
    ATINY_FOTA_STATE           /*固件升级状态*/
} atiny_event_e;
```

- LwM2M协议参数结构体

```
typedef struct
{
    char* binding;           /*目前支持U或者UQ*/
    int life_time;           /*LwM2M协议生命周期，默认50000*/
    unsigned int storing_cnt; /*LwM2M缓存区总字节个数*/
} atiny_server_param_t;
```

- 安全及服务器参数结构体

```
typedef struct
{
    bool is_bootstrap;        /*是否bootstrap服务器*/
    char* server_ip;          /*服务器ip，字符串表示，支持ipv4和ipv6*/
    char* server_port;        /*服务器口号*/
    char* psk_Id;             /*预置共享密钥ID*/
    char* psk;                /*预置共享密钥*/
    unsigned short psk_len;   /*预置共享密钥长度*/
} atiny_security_param_t;
```

- 上报数据的枚举类型

用户上报数据的数据类型，用户根据自身应用扩展

```
typedef enum
{
    FIRMWARE_UPDATE_STATE = 0,    /*设备固件升级状态*/
    APP_DATA                 /*用户数据*/
} atiny_report_type_e;
```

- 服务器参数结构体

```
typedef struct
{
    atiny_server_param_t server_params;
    atiny_security_param_t security_params[2]; /*支持一个IOT服务器，一个bootstrap服务器*/
} atiny_param_t;
```

- 终端设备参数结构体

```
typedef struct
{
    char* endpoint_name;        /*北向申请产生的设备标识码*/
    char* manufacturer;        /*北向申请产生的厂商名称*/
    char* dev_type;             /*北向申请产生的设备类型*/
} atiny_device_info_t;
```

- 数据上报数据结构

以下枚举值，表述了用户上报的数据，最终的反馈类型，比如数据发送成功，数据发送但未得到确认，具体定义如下：

```
typedef enum
{
    NOT_SENT = 0,              /*待上报的数据未发送*/
    SENT_WAIT_RESPONSE,        /*待上报的数据已发送，等待响应*/
    SENT_FAIL,                 /*待上报的数据发送失败*/
    SENT_TIME_OUT,             /*待上报的数据已发送，等待响应超时*/
    SENT_SUCCESS,              /*待上报的数据发送成功*/
    SENT_GET_RST,              /*待上报的数据已发送，但对端响应RST报文*/
    SEND_PENDING,              /*待上报的数据等待发送*/
} data_send_status_e;
```

//用户使用以下数据结构上报数据：

```
typedef struct _data_report_t
{
```

```
atiny_report_type_e type; /*数据上报类型，比如业务数据，电池剩余电量等 */
int cookie; /*数据cookie, 用以在ack回调中，区分不同的数据*/
int len; /*数据长度，不应大于MAX_REPORT_DATA_LEN*/
uint8_t* buf; /*数据缓冲区首地址*/
atiny_ack_callback callback; /*ack回调，其入参取值data_send_status_e类型 */
} data_report_t;
```

### 6.2.2.3 小结

本章从终端设备对接IoT平台的具体流程出发，分别从云侧和端侧详细地阐述了端云互通组件的开发流程。在云侧，本章介绍了创建应用，制作profile，部署编解码插件，注册设备的具体步骤；在端侧，本章从LiteOS SDK端云互通组件的入口函数开始介绍，开发者只需要根据自己的具体业务，实现数据上报任务与命令响应接口，通过LiteOS SDK端云互通组件提供的接口，可以很容易地对接到IoT平台：

```
if(ATINY_OK != atiny_init(atiny_params, &g_phandle)) //初始化
{
    return;
}
uwRet = creat_report_task(); //创建数据上报任务
if(LOS_OK != uwRet)
{
    return;
}
(void)atiny_bind(device_info, g_phandle); //主函数体
```

通过本章的内容，希望开发者能够掌握LiteOS SDK端云互通组件开发流程，进行IoT应用开发和调测。

## 6.2.3 LiteOS 端云互通组件实战演练

LiteOS SDK端云互通组件接入华为OceanConnect IoT云平台既可以通过以太网方式，同时也支持WIFI、GSM、NB-IoT等无线方式。对于这两种不同的方式，本章将通过具体的实例，指导开发者针对自身的开发环境，对LiteOS SDK端云互通组件进行配置，并最终对接IoT平台。对于其中涉及的AT框架相关概念，本章中也会进行简单的介绍。

### 6.2.3.1 开发环境准备

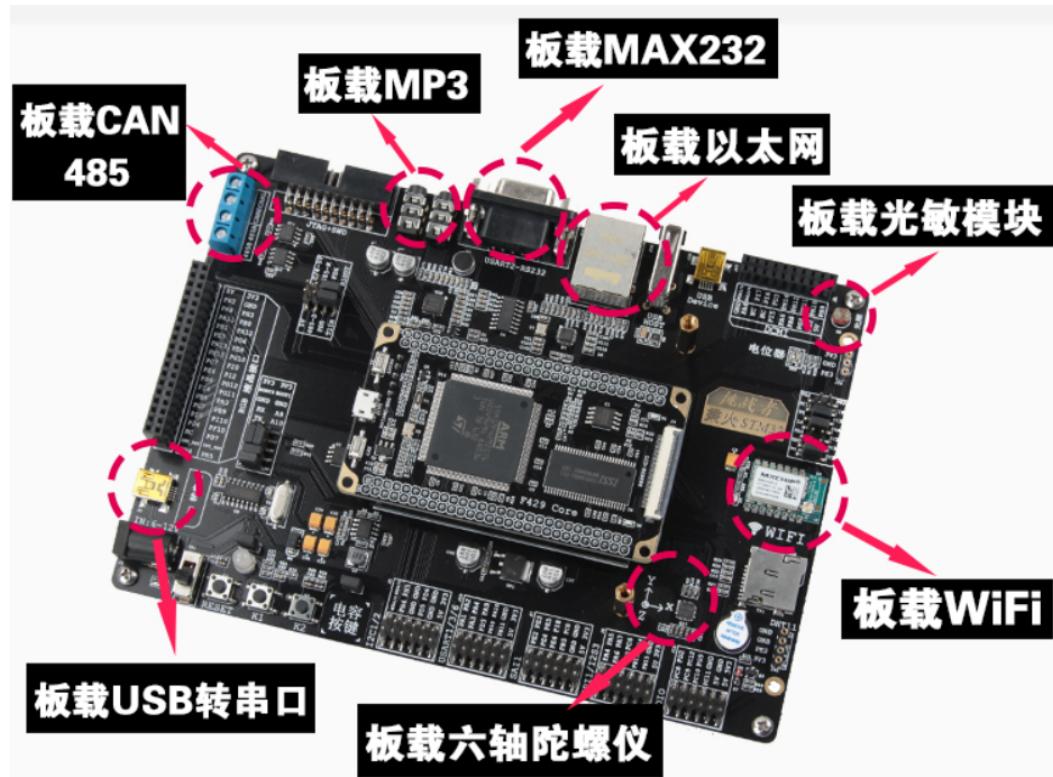
- LiteOS SDK端云互通组件代码：  
获取地址: <https://github.com/LiteOS/LiteOS.git>
- 硬件设备：野火STM32F429开发板，调试下载器（J-Link、ST-Link等）、网线、路由器。



#### 说明

本章以野火STM32F429IG开发板为例进行介绍，板子的详细资料可以从<http://www.firebbs.cn/forum.php>下载。

1. STM32F429IG\_FIRE开发板外设



### 6.2.3.2 (参考) 端云互通组件以太网接入实例

#### 6.2.3.2.1 接入 IoT 平台

**步骤1** 开发板的网口通过网线连接到路由器。

**步骤2** 设置本地IP。

在sys\_init.c中修改device接入的局域网的IP地址值。目前demo程序采用的是静态IP地址的方式，如果需要使用DHCP方式，请在main.c中顶部头文件包含之后定义USE\_DHCP宏即可。

```
void net_init(void)
{
    /* IP addresses initialization */
    IP_ADDRESS[0] = 192;
    IP_ADDRESS[1] = 168;
    IP_ADDRESS[2] = 0;
    IP_ADDRESS[3] = 115;
    NETMASK_ADDRESS[0] = 255;
    NETMASK_ADDRESS[1] = 255;
    NETMASK_ADDRESS[2] = 255;
    NETMASK_ADDRESS[3] = 0;
    GATEWAY_ADDRESS[0] = 192;
    GATEWAY_ADDRESS[1] = 168;
    GATEWAY_ADDRESS[2] = 0;
    GATEWAY_ADDRESS[3] = 1;
}
```

接口net\_init()的调用在agent tiny入口函数agent\_tiny\_entry()之前，作用是完成lwip协议相关的初始化。

sys\_init.c位于 LiteOS/targets/Cloud\_STM32F429IGTx\_FIRE/Src。

**步骤3** 网口的mac地址修改。

在eth.c中将MAC\_ADDR0~MAC\_ADDR5修改成真实的mac地址值保证不重复。

```
static int8_t eth_init(struct netif* netif)
{
    HAL_StatusTypeDef hal_eth_init_status;
    MACAddr[0] = 0x00;
    MACAddr[1] = 0x80;
    MACAddr[2] = 0xE1;
    MACAddr[3] = 0x00;
    MACAddr[4] = 0x00;
    MACAddr[5] = 0x00;
}
```

**注意**

接口eth\_init()将在步骤2中的net\_init()中被调用。eth.c位于LiteOS/targets/Cloud\_STM32F429IGTx\_FIRE/Src。

**步骤4** 设置云平台IP以及设备EP Name和PSK。

现在需要设定相关配置参数。这些参数将作为入参传入atiny\_init()以对LiteOS端云互通组件进行初始化。EP Name就是在云平台上注册设备时开发者设定的验证码，必须保证是唯一的；而PSK（预共享密钥）是用来加密传输的秘钥，agent\_tiny\_demo.c中示例如下：

```
#define DEFAULT_SERVER_IPV4 "139.159.140.34" //OC
char * g_endpoint_name = "44440003"; //与IoT平台上一致
#ifndef WITH_DTLS
char *g_endpoint_name_s = "11110006";
unsigned char g_psk_value[16] = {0xef, 0xe8, 0x18, 0x45, 0xa3, 0x53, 0xc1, 0x3c, 0x0c, 0x89, 0x92, 0xb3, 0x1d,
0x6b, 0xa, 0x96};
#endif
```

agent\_tiny\_demo.c位于LiteOS/demos/agenttiny\_lwm2m。

**步骤5** 编译并运行程序。**步骤6** 查看设备状态。

登录IoT平台开发者Portal，选择“我的设备”，在设备列表中查看对应设备的状态。如果状态为“绑定（bound）”，则表示设备已经成功接入IoT平台。

**图 6-31** 查看设备状态



----结束

### 6.2.3.2.2 数据上报

本文档在第4章中详细介绍了LiteOS SDK端云互通组件设备进行数据上报的完整流程。对于开发者来说，只需要获取传感器数据，并在接口app\_data\_report()中将其传递给数据上报结构体report\_data即可。具体调测过程如下：

**步骤1** 在设备侧执行app\_data\_report函数，使设备上报数据。

修改agent\_tiny\_demo.c中的函数app\_data\_report如下：

```
struct Led_Light
{
    ...
};

extern get_led_lightvalue (void); // 获取传感器数据
void app_data_report(void)
{
    struct Led_Light light;
    data_report_t report_data;
    int ret;
    int cnt = 0;
    report_data.buf = (uint8_t *)&light;
    report_data.callback = ack_callback;
    report_data.cookie = 0;
    report_data.len = sizeof(struct Led_Light);
    report_data.type = APP_DATA;
    while(1)
    {
        report_data.cookie = cnt;
        cnt++;
        ret = atiny_data_report(g_phandle, &report_data);
        printf("report ret:%d\n", ret);
        (void)LOS_TaskDelay(250*8);
    }
}
```

agent\_tiny\_demo.c位于 LiteOS/demos/agenttiny\_lwm2m。

**步骤2** 查看设备状态

登录IoT平台的开发者Portal，在“我的设备”界面的设备列表中，选择上报数据的设备，查看“历史数据”，验证设备数据上报的情况。

图 6-32 使用 LiteOS SDK 端云互通组件的 IoT 设备数据上报业务流程

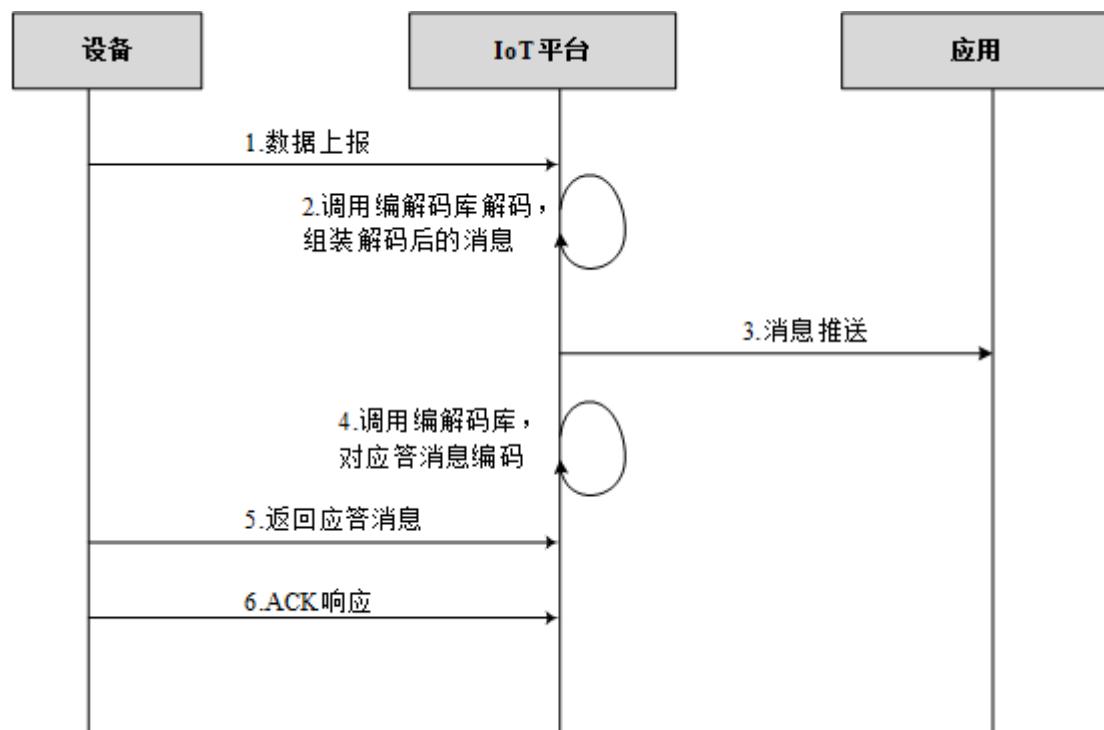


图 6-33 查看数据上报结果



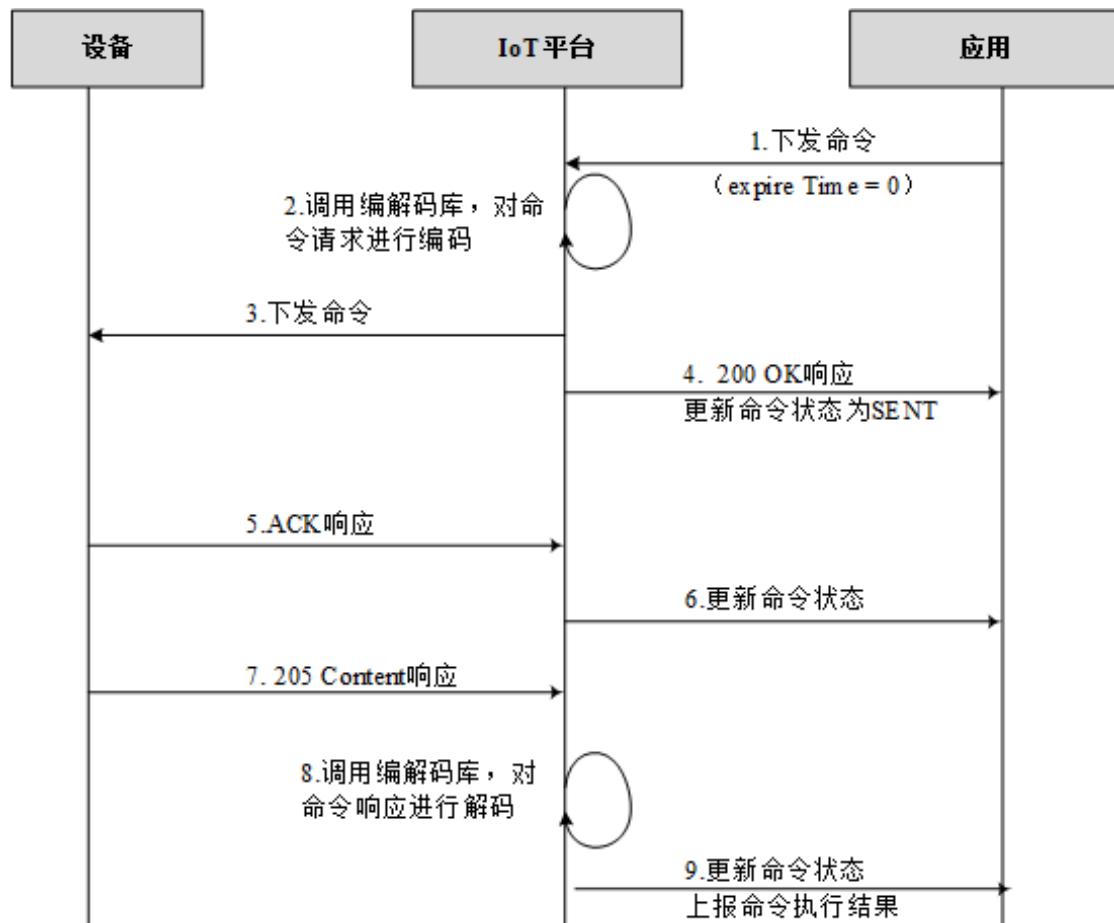
----结束

### 6.2.3.2.3 命令下发

命令下发一般分为两种形式：立即下发和缓存下发。

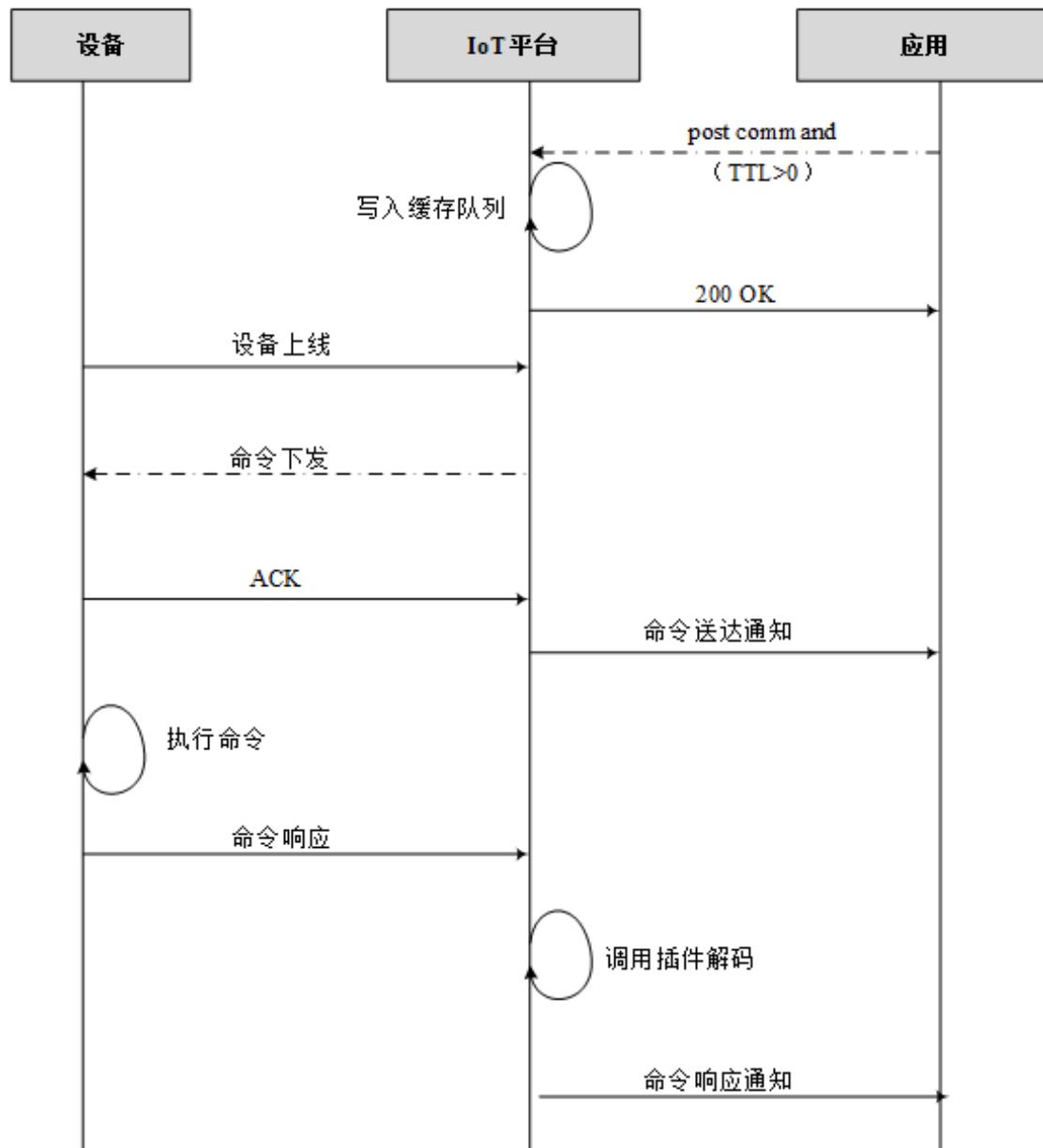
- **立即下发：** IoT平台立即发送收到的命令，如果设备不在线或者设备没收到指令则下发失败。立即下发适合对命令实时性有要求的场景，比如路灯开关灯，燃气表开关阀。使用立即下发时，应用服务器需要自己保证命令下发的时机。

图 6-34 命令立即下发流程



- **缓存下发：**平台收到命令后放入队列。在设备上线的时候，平台依次下发命令队列中的命令。缓存下发适合对命令实时性要求不高的场景，比如配置水表的参数。缓存下发平台根据设备的省电模式进行不同处理。

图 6-35 命令缓存下发流程



应用服务器向IoT平台下发命令时，携带参数expireTime（简称TTL，表示最大缓存时间）。如果不带expireTime，则默认expireTime为48小时。

expireTime=0：命令立即下发。

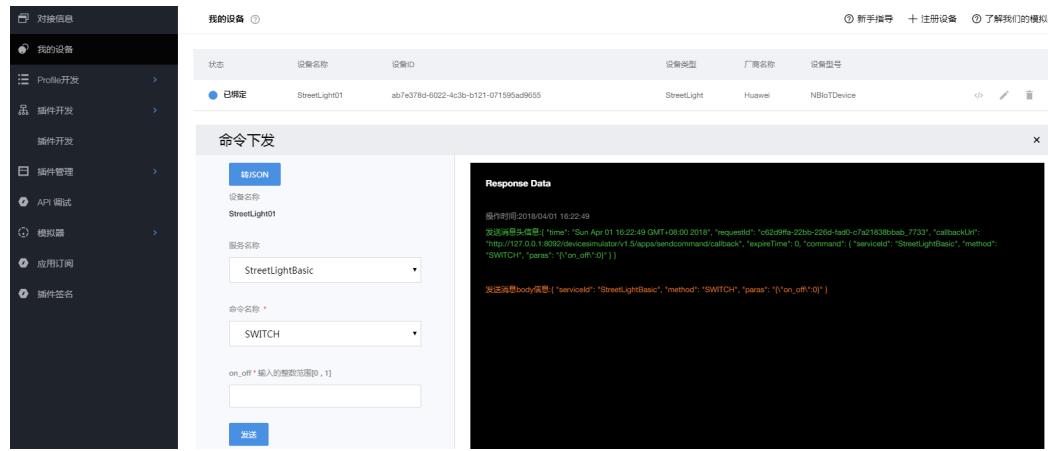
expireTime>0：命令缓存下发。

LiteOS SDK端云互通组件场景命令下发的调测过程，命令下发步骤如下：

**步骤1** 登录IoT平台的开发者Portal。开发者Portal的访问地址、账号和密码需要向IoT平台服务提供商申请。

**步骤2** 在“我的设备”界面的设备列表中，选择接收命令的设备，点击“命令下发(</>)”。在弹出界面中，配置下发给设备的命令参数。

图 6-36 命令下发



**步骤3** 在“我的设备”界面的设备列表中，选择接收命令的设备->“历史命令”，查看“状态”栏的显示。

图 6-37 命令下发状态



状态说明如下：

- **超期:** 表示命令在IoT平台缓存时间超期，未向设备下发。
- **成功:** 表示IoT平台已经将命令下发给设备，且收到设备上报的命令执行结果。
- **失败:** 表示编解码插件解析为空，或执行结果响应里面有“ERROR CODE”等。
- **超时:** 表示IoT平台等待ACK响应超时。
- **取消:** 表示应用侧已经取消命令下发。
- **等待:** 表示命令在IoT平台缓存，还未下发给设备。
- **已发送:** 表示IoT平台已经将命令下发给设备。
- **已送达:** 表示IoT平台已经将命令下发给设备，且收到设备返回的ACK消息。

**步骤4** LiteOS SDK端云互通组件从消息缓存中获取消息码流并解析，agent\_tiny\_cmd\_ioctl.c中atiny\_cmd\_ioctl()接口对应的回调函数（实际调用atiny\_write\_app\_write()处理下发命令）。

```
int atiny_write_app_write(void* user_data, int len)
{
    int i;
    uint8_t cmd_data[len];
    memcpy(cmd_data, user_data, len);
    for(i=0;i<len;i++)
    {
        //打印下发的命令数据，用户可以处理下发的命令，根据具体的命令控制硬件设备
        printf("##### %d", cmd_data[i]);
    }
}
```

```
(void)atiny_printf("write num19 object success\r\n");
    return ATINY_OK;
}
```

agent\_tiny\_cmd\_ioctl.c位于 LiteOS/demos/agenttiny\_lwm2m。

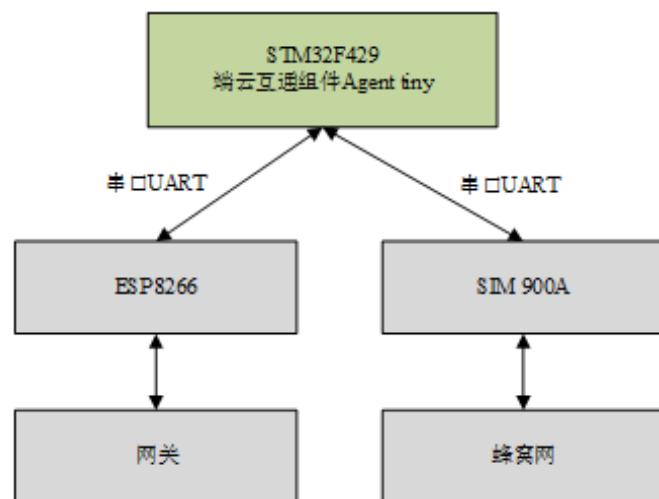
----结束

### 6.2.3.3 (参考) 端云互通组件无线接入实例

#### 6.2.3.3.1 无线接入介绍

无线的接入方式包括WIFI、GSM、NB-IoT、Zigbee、蓝牙等，本文主要讲解WIFI和GSM（GPRS）的接入方式。对物联网开发者来说，WIFI或者GSM一般都是一个单独的模块，运行在MCU上的LiteOS SDK端云互通组件需要使用WIFI或者GSM提供的网络服务时，需要通过串口AT指令就可以了，如下图所示，ESP8266是乐鑫的WIFI模组，SIM900A是SIMCom芯讯通推出的GSM/GPRS模组。

图 6-38 Huawei LiteOS SDK 端云互通组件无线接入方案示意图

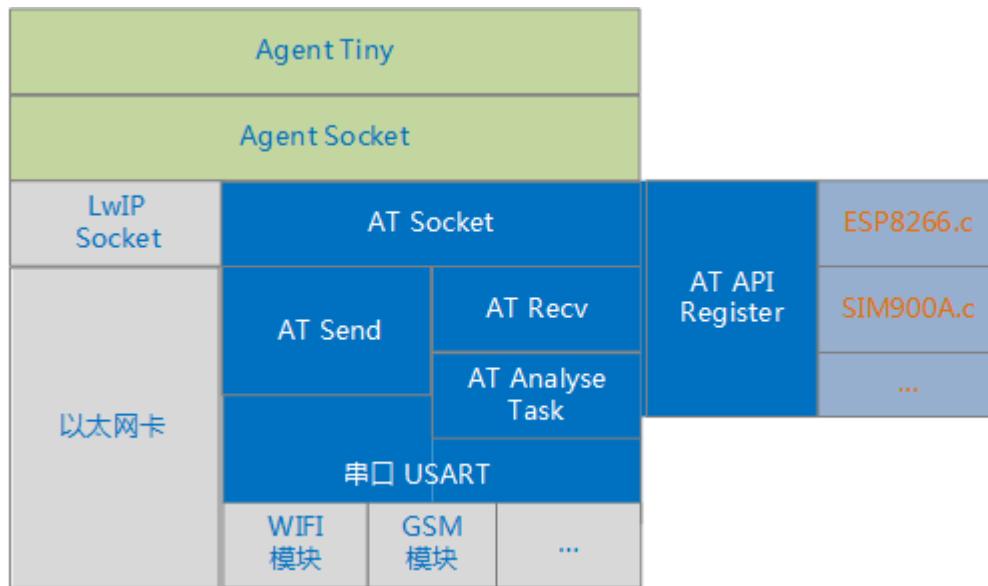


AT 即 Attention，AT指令集是从终端设备 (Terminal Equipment, TE)或者数据终端设备 (Data Terminal Equipment, DTE)向终端适配器(Terminal Adapter, TA)或数据电路终端设备 (Data Circuit Terminal Equipment, DCE)发送的。通过TA, TE发送AT指令来控制移动台(Mobile Station, MS)的功能，与GSM 网络业务进行交互。用户可以通过AT指令进行呼叫、短信、电话本、数据业务、传真等方面控制。

#### 6.2.3.3.2 AT 框架介绍

不论使用ESP8266还是SIM900A，都可以使用AT+UART方式接入，主要的差别在于具体的AT指令，但很多情况下都是类似的，LiteOS SDK端云互通组件提供了一种AT框架，也可以称之为AT模板，方便用户移植不同串口通信模块（需要支持TCP/IP协议栈），AT框架的方案如下图所示。

图 6-39 AT 框架方案结构图



结构图中AT Socket用于适配Atiny Socket接口，类似posix socket，AT Send用于调用at\_cmd发送AT命令，AT Recv用于AT Analyse Task，通过LiteOS消息队列Post消息到用户接收任务。AT Analyse Task的主要功能是解析来自串口的消息，包括用户数据和命令的响应，串口USART主要是在中断或者DMA模式下接收数据，AT API Register是提供设备模块注册的API函数。

结构图中深蓝色的部分是AT框架公共部分代码，开发者不需要修改；浅蓝色的部分是设备相关代码，开发者需要编写相应的设备代码，根据at\_api.h文件的定义，开发者只要实现以下函数接口即可：

```
typedef struct {
    int32_t (*init)(void); /* 初始化，初始化串口、IP网络等*/
    int8_t (*get_localmac)(int8_t *mac); /* 获取本地MAC*/
    int8_t (*get_localip)(int8_t *ip, int8_t *gw, int8_t *mask); /* 获取本地IP*/
    /* 建立TCP或者UDP连接*/
    int32_t (*connect)(const int8_t *host, const int8_t *port, int32_t proto);
    /* 发送，当命令发送后，如果超过一定的时间没收到应答，要返回错误*/
    int32_t (*send)(int32_t id, const uint8_t *buf, uint32_t len);
    int32_t (*recv_timeout)(int32_t id, int8_t *buf, uint32_t len, int32_t timeout);
    int32_t (*recv)(int32_t id, int8_t *buf, uint32_t len);

    int32_t (*close)(int32_t id); /* 关闭连接*/
    int32_t (*recv_cb)(int32_t id); /* 收到各种事件处理，暂不实现 */
    int32_t (*deinit)(void);
} at_adaptor_api;
```

at\_api.h位于 LiteOS/include/at\_frame。

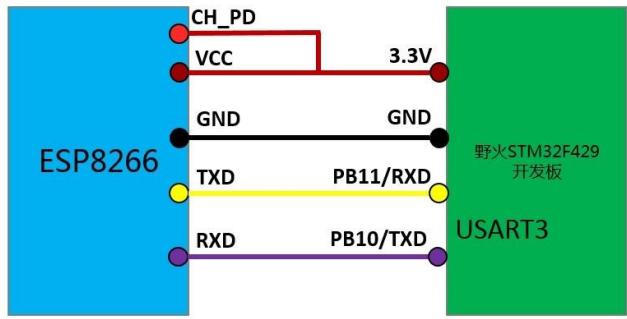
### 6.2.3.3 移植 WIFI 模块-ESP8266

上一小节中，本文对AT框架进行了简单的介绍。其中需要开发者实现at\_api\_interface.h中所定义的接口，之后通过AT API Register进行注册，供上层的Agent Socket调用。本节给出WIFI模块ESP8266的具体例子，帮助开发者进行移植。

**步骤1** STM32F429开发板上连接ESP8266 串口wifi模块，如下图所示：



ESP8266模块



ESP8266与STM32F429开发板连接关系

**步骤2** 首先在设备文件esp8266.c定义API结构体。

```
at_adaptor_api at_interface = {  
    .init = esp8266_init,  
    .get_localmac = esp8266_get_localmac, /*get local MAC*/  
    .get_localip = esp8266_get_localip, /*get local IP*/  
    /*build TCP or UDP connection*/  
    .connect = esp8266_connect,  
    .send = esp8266_send,  
    .recv_timeout = esp8266_recv_timeout,  
    .recv = esp8266_recv,  
    .close = esp8266_close, /*close connection*/  
    .recv_cb = esp8266_recv_cb, /* operation for events, not implements yet */  
    .deinit = esp8266_deinit,  
};
```

esp8266.c位于 LiteOS/components/net/at\_device/wifi\_esp8266

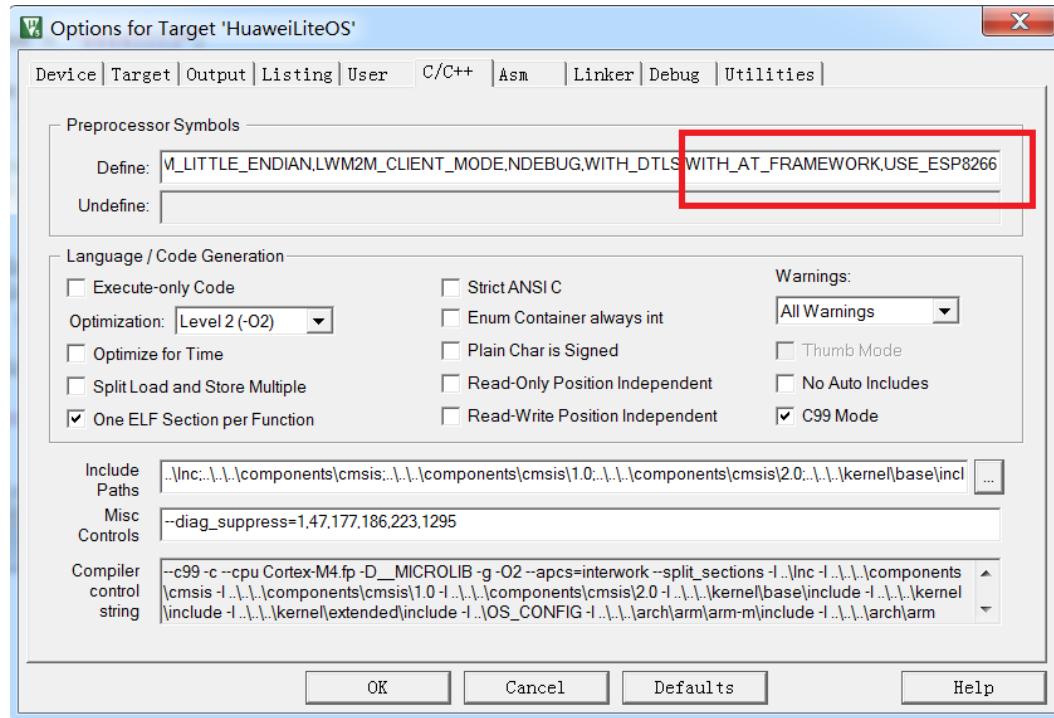
**步骤3** 在main.c文件中，代码如下：

```
#elif defined(WITH_AT_FRAMEWORK) && (defined(USE_ESP8266) || defined(USE_SIM900A))  
    extern at_adaptor_api at_interface;  
    at_api_register(&at_interface); //注册开发者定义的接口  
    agent_tiny_entry();  
#endif
```

main.c位于 LiteOS/targets/Cloud\_STM32F429IGTx\_FIRE/Src。

**步骤4** 确保打开了编译宏。

图 6-40 全局宏包含 WITH\_AT\_FRAMEWORK 和 USE\_ESP8266



### 步骤5 在esp8266.c实现具体设备API接口。

例如demo例程初始化如下：

```
int32_t esp8266_init()
{
    at.init();
    at.oob_register(AT_DATAF_PREFIX, strlen(AT_DATAF_PREFIX), esp8266_data_handler);
#ifdef USE_USARTRX_DMA
HAL_UART_Receive_DMA(&at_usart, &at.recv_buf[at_user_conf.user_buf_len*0], at_user_conf.user_buf_len);
#endif
    esp8266_reset();
    esp8266_echo_off();
    esp8266_choose_net_mode(STA);
    while(AT_FAILED == esp8266_joinap(WIFI_SSID, WIFI_PASSWD))

    {
        AT_LOG("connect_ap failed, repeat...");

    };
    esp8266_set_mux_mode(at.mux_mode);
    static int8_t ip[32];
    static int8_t gw[32];
    static int8_t mac[32];
    esp8266_get_localip(ip, gw, NULL);
    esp8266_get_localmac(mac);
    AT_LOG("get ip:%s, gw:%s mac:%s", ip, gw, mac);
    return AT_OK;
}
```

其它几个接口参考esp8266.c即可，而ESP8266模块AT命令定义的宏在esp8266.h，具体含义可以查看ESP8266官方手册，另外用户需要在esp8266.h中修改自己连接的wifi的ssid和密码。

```
#define AT_CMD_RST          "AT+RST"
#define AT_CMD_ECHO_OFF       "ATE0"
#define AT_CMD_CWMODE          "AT+CWMODE_CUR"
#define AT_CMD_JOINAP          "AT+CWJAP_CUR"
#define AT_CMD_MUX             "AT+CIPMUX"
#define AT_CMD_CONN            "AT+CIPSTART"
#define AT_CMD_SEND            "AT+CIPSEND"
#define AT_CMD_CLOSE           "AT+CIPCLOSE"
#define AT_CMD_CHECK_IP        "AT+CIPSTA_CUR?"
#define AT_CMD_CHECK_MAC       "AT+CIPSTAMAC_CUR?"
```

esp8266.h位于 LiteOS/components/net/at\_device/wifi\_esp8266。

----结束

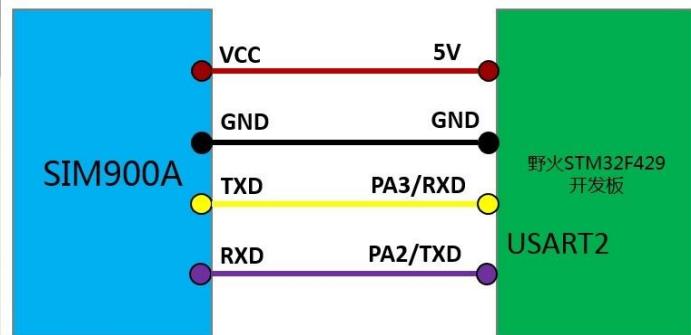
#### 6.2.3.3.4 移植 GSM 模块-SIM900A

与ESP8266非常类似，只不过具体AT命令有稍微差异。

**步骤1** STM32F429开发板上连接SIM900A串口GSM模块，如下图所示



SIM900A模块



SIM900A模块与STM32开发板连接关系图

**步骤2** 在设备文件sim900a.c定义API结构体。

```
at_adaptor_api at_interface = {
    .init = sim900a_ini,
    /*TCP or UDP connect*/
    .connect = sim900a_connect,
    /*send data, if no response, retrun error*/
    .send = sim900a_send,
    .recv_timeout = sim900a_recv_timeout,
    .recv = sim900a_recv,
    .close = sim900a_close,/*close connect*/
    .recv_cb = sim900a_recv_cb,/*receive event handle, no available by now */
    .deinit = sim900a_deinit,
};
```

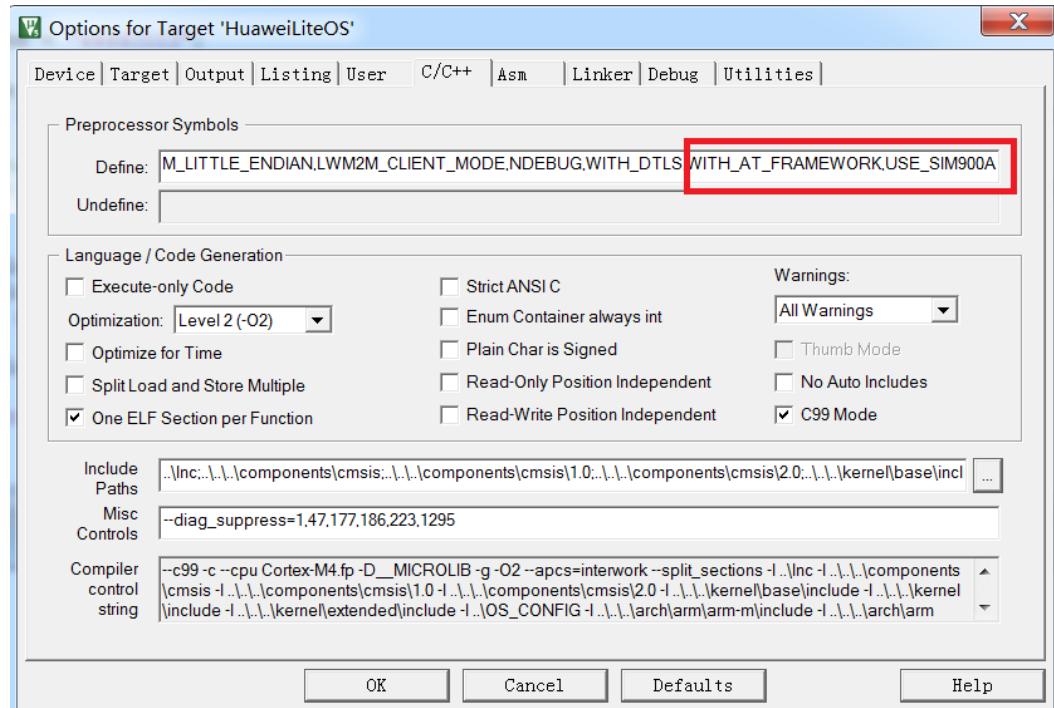
sim900a.c位于 LiteOS/components/net/at\_device/gprs\_sim900a。

**步骤3** 在main.c文件中，代码如下：

```
#elif defined(WITH_AT_FRAMEWORK) && (defined(USE_ESP8266) || defined(USE_SIM900A))
    extern at_adaptor_api at_interface;
    at_api_register(&at_interface);
    agent_tiny_entry();
#endif
```

**步骤4** 确保打开了编译宏

图 6-41 全局宏包含 WITH\_AT\_FRAMEWORK 和 USE\_SIM900A



### 步骤5 在sim900a.c实现具体设备API接口。

例如demo例程发送和接收函数如下：

```
int32_t sim900a_recv_timeout(int32_t id, int8_t * buf, uint32_t len, int32_t timeout)
{
    uint32_t qlen = sizeof(QUEUE_BUFF);
    QUEUE_BUFF qbuf = {0, NULL};
    printf("****at.linkid[%d].qid=%d***\n", at.linkid[id].qid);
    int ret = LOS_QueueReadCopy(at.linkid[id].qid, &qbuf, &qlen, timeout);
    AT_LOG("ret = %x, len = %d, id = %d", ret, qbuf.len, id);
    if (qbuf.len) {
        memcpy(buf, qbuf.addr, qbuf.len);
        atiny_free(qbuf.addr);
    }
    return qbuf.len;
}
int32_t sim900a_send(int32_t id, const uint8_t *buf, uint32_t len)
{
    int32_t ret = -1;
    char cmd[64] = {0};
    if (AT_MUXMODE_SINGLE == at.mux_mode)
    {
        sprintf(cmd, 64, "%s=%d", AT_CMD_SEND, len);
    }
    else
    {
        sprintf(cmd, 64, "%s=%d,%d", AT_CMD_SEND, id, len);
    }
    ret = at.write((int8_t *)cmd, "SEND OK", (int8_t*)buf, len);
    return ret;
}
```

而SIM900A模块AT命令定义的宏在sim900a.h定义如下，具体含义可以查看SIM900A官方手册。

```
#define AT_CMD_AT          "AT"
#define AT_CMD_CPIN         "AT+CPIN?"/check sim card
#define AT_CMD_COPS          "AT+COPS?"/check register network
#define AT_CMD_CLOSE         "AT+CIPCLOSE"
#define AT_CMD_SHUT          "AT+CIPSHUT"
#define AT_CMD_ECHO_OFF      "ATEO"
#define AT_CMD_ECHO_ON       "ATE1"
#define AT_CMD_MUX           "AT+CIPMUX"
#define AT_CMD_CLASS          "AT+CGCLASS"/set MS type
#define AT_CMD_PDP_CONT      "AT+CGDCONT"/configure pdp context

#define AT_CMD_PDP_ATT        "AT+CGATT"/pdp attach network
#define AT_CMD_PDP_ACT        "AT+CGACT"/active pdp context
#define AT_CMD_CSTT          "AT+CSTT"/start task
#define AT_CMD_CIICR         "AT+CIICR"/start gprs connect
#define AT_CMD_CIFSR          "AT+CIFSR"/get local ip
#define AT_CMD_CIPHEAD        "AT+CIPHEAD"
#define AT_CMD_CONN          "AT+CIPSTART"
#define AT_CMD_SEND          "AT+CIPSEND"
#define AT_CMD_CLOSE          "AT+CIPCLOSE"
```

sim900a.h位于 LiteOS/components/net/at\_device/gprs\_sim900a。

----结束

### 6.2.3.3.5 注意事项

由于LiteOS SDK端云互通组件的发送和接收在同一个任务中，接收消息的接口不能一直是阻塞的，而必须使用带有超时机制的接收接口，即我们总是实现int32\_t (\*recv\_timeout)(int32\_t id, int8\_t \*buf, uint32\_t len, int32\_t timeout)这个接口，且接收超时时间目前是10秒（#define BIND\_TIMEOUT(10)）。

如果用户设计的应用发送消息和接收消息在不同的任务中，那么可以使用阻塞接口int32\_t (\*recv)(int32\_t id, int8\_t \*buf, uint32\_t len)。

### 6.2.3.4（参考）设备模拟器接入平台

#### 6.2.3.4.1 设备模拟器接入平台

设备接入IoT平台后，IoT平台才可以对设备进行管理。

IoT平台提供了设备模拟器，可以模拟真实设备接入IoT平台的场景。本节基于设备模拟器进行操作。

**步骤1** 选择“模拟器”->“NB设备模拟器”->“绑定设备”，输入验证码，点击“确定”。

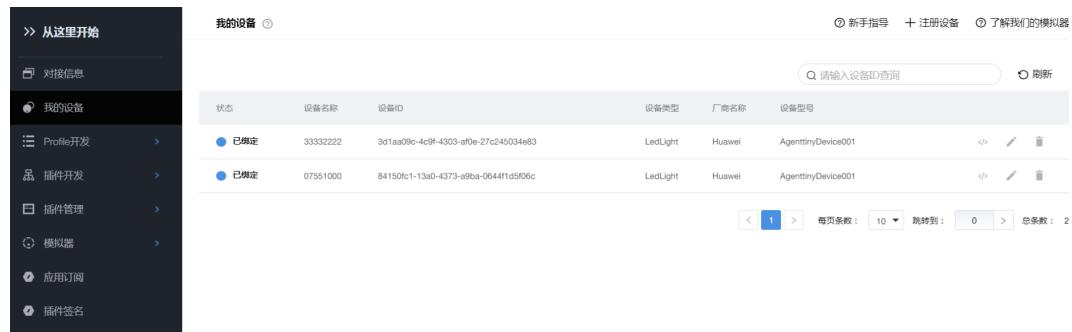
输入的验证码需要和注册设备时使用的验证码（verifyCode）一致。

图 6-42 设备模拟器



**步骤2** 选择“我的设备”，在设备列表中查看对应设备的状态。如果状态为“绑定（bound）”，则表示设备已经成功接入IoT平台。

图 6-43 查看设备状态



----结束

#### 6.2.3.4.2 设备模拟器数据上报

设备在收到平台下发命令或者资源订阅后，会上报命令响应或资源订阅消息，由IoT平台将设备上报的消息推送到应用服务器或订阅的地址。如果上报数据的南向设备是NB-IoT设备或者使用LiteOS SDK端云互通组件集成的设备，IoT平台在将消息推送到应用服务器或订阅的地址之前，会先调用编解码插件对消息进行解析。

IoT平台提供了设备模拟器，可以模拟真实设备上报数据的场景。本节基于NB设备模拟器（NB设备模拟器也可以模拟LiteOS SDK端云互通组件的数据上报功能）进行操作。

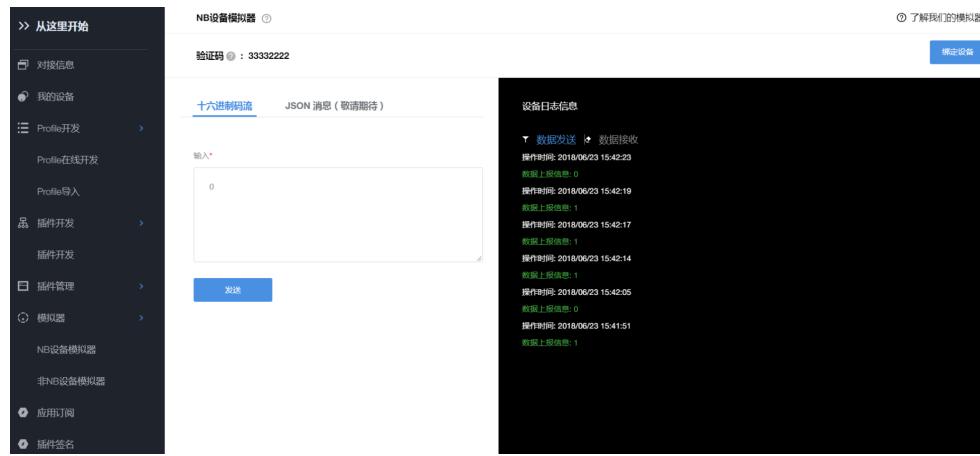
**步骤1** 登录IoT平台的开发者Portal。开发者Portal的访问地址、账号和密码需要向IoT平台服务提供商申请。

**步骤2** 选择“模拟器”->“NB设备模拟器”，输入需要上报的码流，点击“发送”。

在“设备日志信息”->“数据发送”中，可以查看数据上报信息。

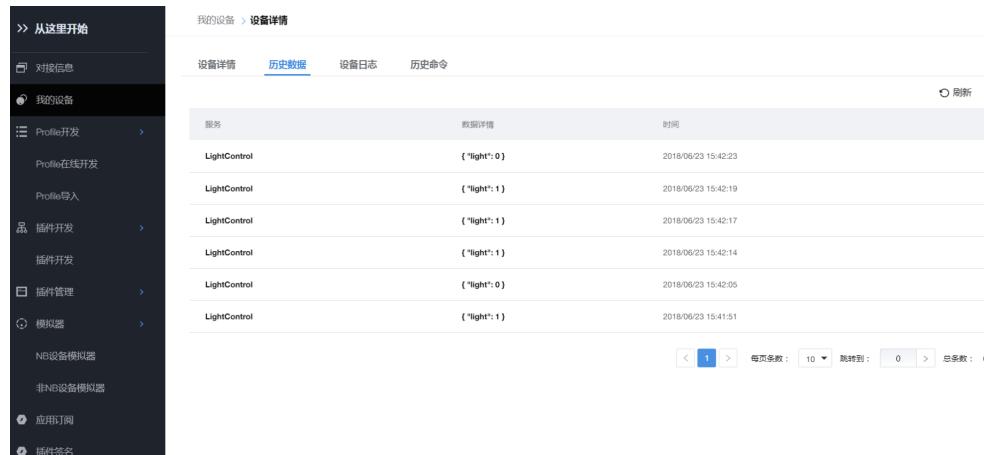
在“设备日志信息”->“数据接收”中，可以查看数据上报响应信息。

图 6-44 模拟数据上报



**步骤3** 在“我的设备”界面的设备列表中，选择上报数据的设备，查看“历史数据”，验证编解码插件是否可以对上报数据进行解析。

图 6-45 查看数据上报结果



以一款Led灯设备的编解码插件为例进行说明，该款设备包含一种服务LightControl（设置多种服务包含多种属性、多种命令类似）：

- LightControl服务：包含light一种属性（灯亮或者灭）和一种命令（设置灯亮或者灭）。

使用设备模拟器中上报“01”的十六进制码流后，在“历史数据中”获得的编解码插件解码结果将会为：

LightControl: { "light": 1 }

**步骤4** 在“我的设备”界面的设备列表中，选择上报数据的设备，查看“历史数据”，验证设备数据上报的情况。

“历史数据”中显示为经过编解码插件解析后的结果。

----结束

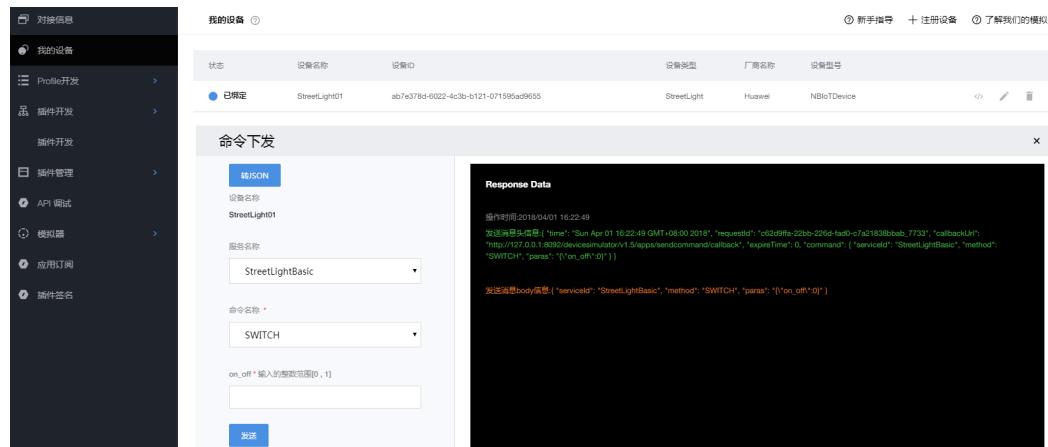
### 6.2.3.4.3 应用模拟器命令下发

应用服务器需要调用IoT平台的命令下发接口，对设备下发控制指令。如果接收命令的设备是NB-IoT设备（或者集成LiteOS SDK端云互通组件的南向设备），IoT平台收到应用服务器下发的命令后，会先调用编解码插件进行转换，再发送给设备。

IoT平台提供了应用模拟器，可以模拟应用服务器下发命令的场景。本节基于应用模拟器进行操作。

- 步骤1** 在“我的设备”界面的设备列表中，选择接收命令的设备，点击“命令下发(</>)”。  
在弹出界面中，配置下发给设备的命令参数。

**图 6-46 命令下发**



- 步骤2** 在“我的设备”界面的设备列表中，选择接收命令的设备->“历史命令”，查看“状态”栏的显示。

**图 6-47 命令下发状态**



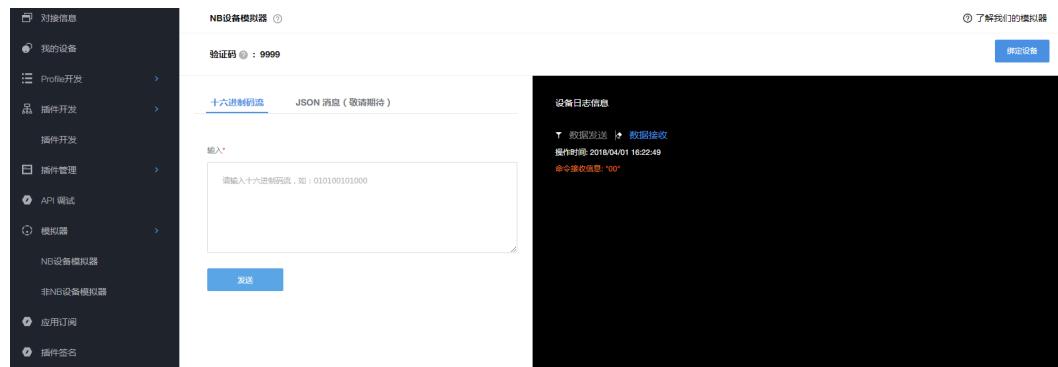
状态说明如下：

- 超期：**表示命令在IoT平台缓存时间超期，未向设备下发。
- 成功：**表示IoT平台已经将命令下发给设备，且收到设备上报的命令执行结果。
- 失败：**表示编解码插件解析为空，或执行结果响应里面有“ERROR CODE”等。
- 超时：**表示IoT平台等待ACK响应超时。
- 取消：**表示应用侧已经取消命令下发。
- 等待：**表示命令在IoT平台缓存，还未下发给设备。
- 已发送：**表示IoT平台已经将命令下发给设备。

- 已送达：表示IoT平台已经将命令下发给设备，且收到设备返回的ACK消息。

**步骤3** 选择“模拟器”->“NB设备模拟器”->“设备日志信息”->“数据接收”，查看设备模拟器收到的命令信息。

**图 6-48 命令接收信息**



----结束

## 6.2.4 附录 LwM2M 协议介绍

### 6.2.4.1 LwM2M 协议是什么

LwM2M (Lightweight M2M, 轻量级M2M)，由开发移动联盟 (OMA) 提出，是一种轻量级的、标准通用的物联网设备管理协议，可用于快速部署客户端/服务器模式的物联网业务。

LwM2M为物联网设备的管理和应用建立了一套标准，它提供了轻便小巧的安全通信接口及高效的数据模型，以实现M2M设备管理和服务支持。

### 6.2.4.2 LwM2M 协议特性

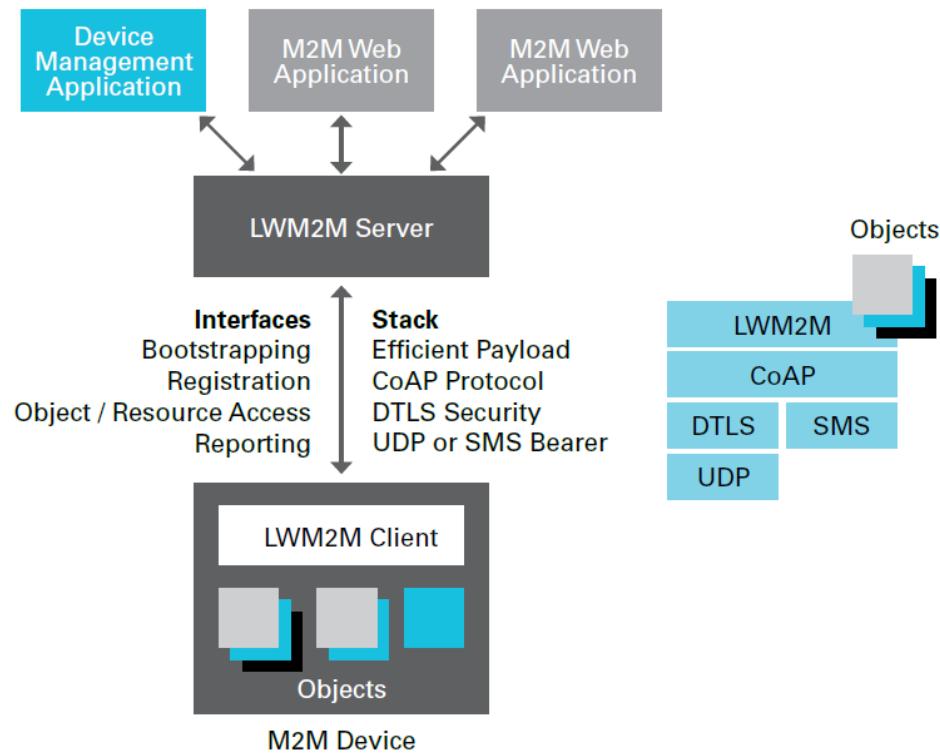
LwM2M协议主要特性包括：

- 基于资源模型的简单对象
- 资源操作：创建/检索/更新/删除/属性配置
- 资源的观察/通知
- 支持的数据格式：TLV/JSON/Plain Text/Opaque
- 传输层协议：UDP/SMS
- 安全协议：DTLS
- NAT/防火墙应对方案：Queue模式
- 支持多LwM2M Server
- 基本的M2M功能：LwM2M Server，访问控制，设备，网络连接监测，固件更新，位置和定位服务，统计

### 6.2.4.3 LwM2M 体系架构

LwM2M体系架构如图所示：

图 6-49 LwM2M 体系架构



#### 6.2.4.4 LwM2M 对象定义

##### 对象概念

对象是逻辑上用于特定目的的一组资源的集合。例如固件更新对象，它就包含了用于固件更新目的的所有资源，例如固件包、固件URL、执行更新、更新结果等。

使用对象的功能之前，必须对该对象进行实例化，对象可以有多个对象实例，对象实例的编号从0开始递增。

OMA 定义了一些标准对象，LwM2M 协议为这些对象及其资源已经定义了固定的ID。例如：固件更新对象的对象ID为5，该对象内部有8个资源，资源ID分别为0~7，其中“固件包名字”这个资源的ID为6。因此，URI 5/0/6表示：固件更新对象第0个实例的固件包名字这个资源。

##### 对象定义的格式

Name	Object ID	Instances	Mandatory	Object URN
Object Name	16-bit Unsigned Integer	Multiple/Single	Mandatory/Optional	urn:oma:LwM2M:{oma,ext,x}:{Object ID}

## OMA 定义的标准对象

OMA的LwM2M规范中定义了7个标准对象:

Object	object id	description
LwM2M Security	0	LwM2M (bootstrap) server的URI, payload的安全模式, 一些算法/密钥, server的短ID等信息。
LwM2M Server	1	server的短ID, 注册的生命周期, observe的最小/最大周期, 绑定模型等。
Access Control	2	每个Object的访问控制权限。
Device	3	设备的制造商, 型号, 序列号, 电量, 内存等信息。
Connectivity Monitoring	4	网络制式, 链路质量, IP地址等信息。
Firmware	5	固件包, 包的URI, 状态, 更新结果等。
Location	6	经纬度, 海拔, 时间戳等。
Connectivity Statistics	7	收集期间的收发数据量, 包大小等信息。

LiteOS SDK端云互通组件配合Huawei Ocean Connect物联网开发平台能力, 还支持的19号LwM2M APPDATA对象:

Object	object id	description
LwM2M APPDATA	19	此LwM2M对象提供LWM2M 服务器相关的应用业务数据, 例如水表数据。



说明

OMA组织定义的其它常用对象, 详见<http://www.openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html>。

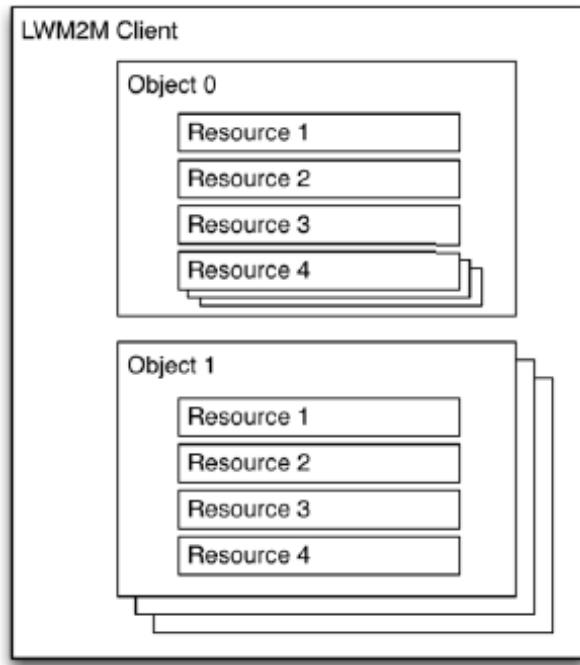
### 6.2.4.5 LwM2M 资源定义

#### 资源模型

LwM2M定义了一个资源模型, 所有信息都可以抽象为资源以提供访问。资源是对象的内在组成, 隶属于对象, LwM2M客户端可以拥有任意数量的资源。和对象一样, 资源也可以有多个实例。

LwM2M客户端、对象以及资源的关系如图所示:

图 6-50 LwM2M 客户端、对象、资源关系图



## 资源定义的格式

ID	Name	Operations	Instance	Mandator	Type	Range or Enumeration	Units	Description
0	Resource Name	R (Read), W (Write), E (Execute)	Multiple/Single	Mandatory/Optional	String, Integer, Float, Boolean, Opaque, Time, Objlnk none	If any	If any	Description

### 6.2.4.6 LwM2M 接口定义

#### 概述

LwM2M引擎主要有两个组件：LwM2M服务器和LwM2M客户端。LwM2M标准为两个组件之间的交互设计了4种主要的接口：

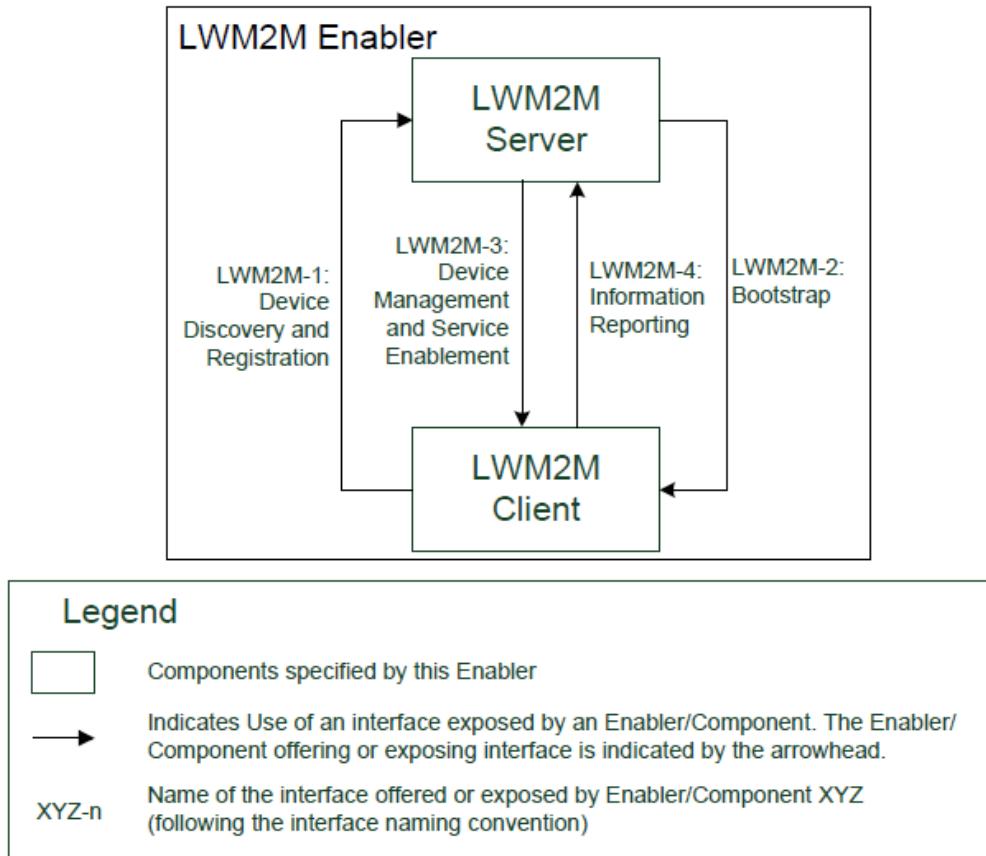
- 设备发现和注册
- 引导程序
- 设备管理和服务实现

- 信息上报

## 接口模型图

LwM2M 接口模型如图所示：

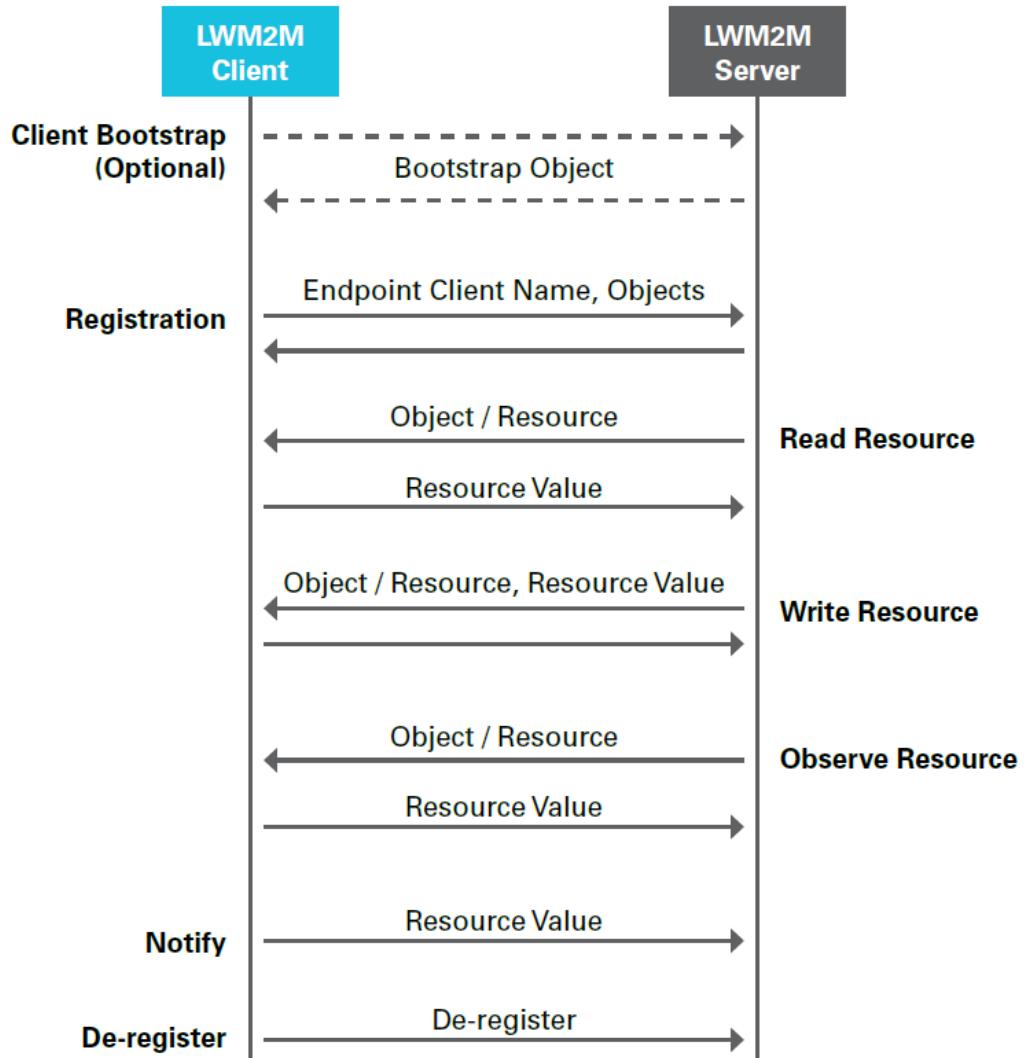
图 6-51 LwM2M 接口模型



## 消息流程示例

LwM2M 的消息交互流程如图所示：

图 6-52 LwM2M 消息流程



## 设备管理和服务实现接口

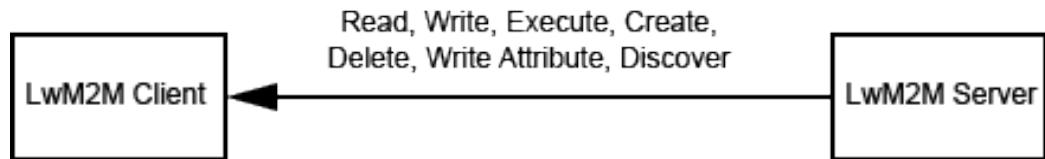
LwM2M的接口表示一类功能，设备管理和服务实现接口是LwM2M的四种接口之一。

接口的具体功能是由一系列的操作来实现的，LwM2M的4种接口被分为上行操作和下行操作。

- 上行操作：LwM2M Client -> LwM2M Server
- 下行操作：LwM2M Server -> LwM2M Client

LwM2M Server使用设备管理和服务实现接口来访问LwM2M Client的对象实例和资源。该接口包括7种操作：“Create”、“Read”、“Write”、“Delete”、“Execute”、“Write Attributes”和“Discover”。

图 6-53 设备管理和服务实现接口操作



接口	操作	方向
设备管理和服务实现	Create, Read, Write, Delete, Execute, Write Attributes, Discover	下行

设备管理和服务实现接口的交互过程如图所示：

图 6-54 设备管理&amp;服务使能接口示例

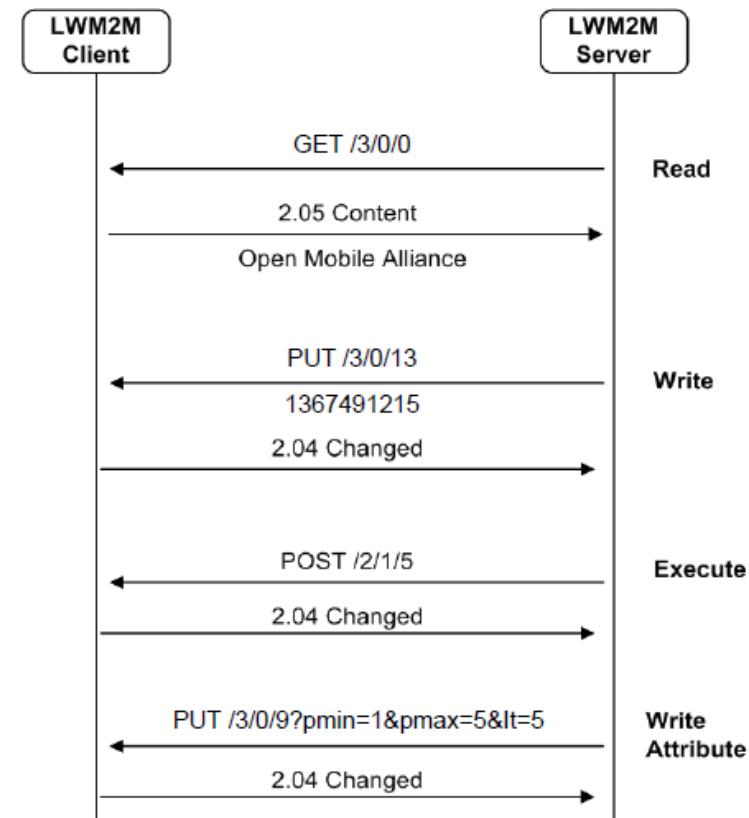
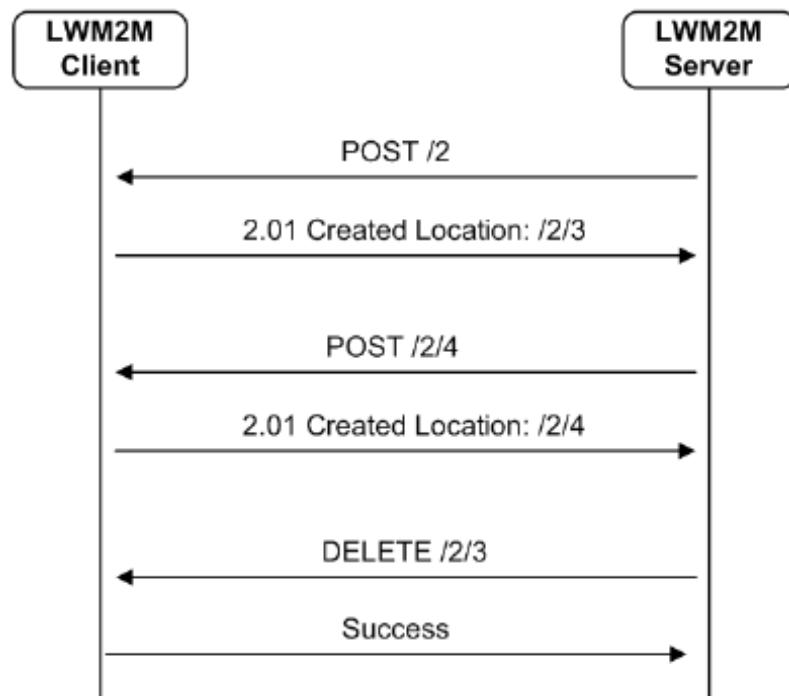


图 6-55 对象创建和删除示例



#### 6.2.4.7 固件升级

LwM2M的固件升级对象使得固件升级的管理成为可能。固件升级对象包括安装固件包、更新固件、以及更新固件之后执行的其它动作。成功进行了固件升级后，device必须要重启，以使新的固件生效。

在设备重启之后，如果“Packet”资源包含了一个合法的但还未被安装的固件包，“State”资源必须为<Downloaded>状态，否则须为<Idle>状态。

在设备重启之前，标识更新结果的相关数值必须保存。

#### 固件升级对象定义

Name	Object ID	Instances	Mandatory	Object URN
Firmware Update	5	Single	Optional	rn:oma:LwM2M:oma:5

#### 固件升级对象的资源定义

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
0	Package	W	Single	Mandatory	Opaque	-	固件包。

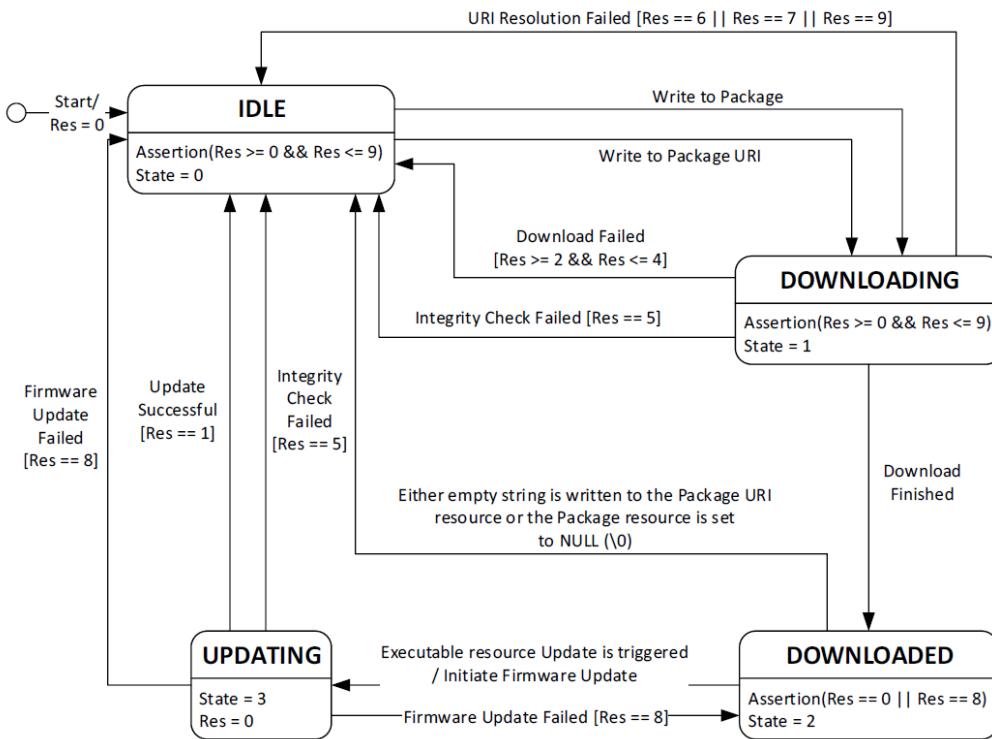
ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
1	Package URI	W	Single	Mandatory	String	0-255 bytes	固件包的下载URI。
2	Update	E	Single	Mandatory	none	no argument	更新固件。 只有当State资源是Downloaded状态时，该资源才是可执行的。
3	State	R	Single	Mandatory	Integer	0-3	固件升级的状态。这个值由客户端设置。0: Idle (下载之前或者成功更新之后) 1: Downloading (下载中) 2: Downloaded (下载已完成) 3: Updating (更新中)在Downloaded状态下，如果执行Resource Update操作，状态则切换为Updating。 如果更新成功，状态切换为Idle；如果更新失败，状态切换回Downloaded。
4	Update Supported Objects	RW	Single	Optional	Boolean	-	默认值是false。 如果该值置为true，则固件成功更新了之后，客户端必须通过更新消息或注册消息，向服务器通知Object的变动。 如果更新失败了，Object参数通过下一阶段的更新消息报告。

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Description
5	Update Result	R	Single	Mandatory	Integer	0-8	下载/更新固件的结果： 0: 初始值。一旦开始更新流程（下载/更新）开始，该资源的值必须被置为0。 1: 固件升级成功 2: 没有足够的空间存储新固件包 3: 下载过程中内存不足 4: 下载过程中连接丢失 5: 新下载的包完整性检查失败 6: 不支持的包类型 7: 不合法的URI 8: 固件升级失败该资源可以通过Observe操作上报。
6	PkgName	R	Single	Optional	String	0-255 bytes	固件包的名字。
7	PkgVersion	R	Single	Optional	String	0-255 bytes	固件包的版本。

## 固件升级状态机

固件升级状态之间的变换关系如图所示：

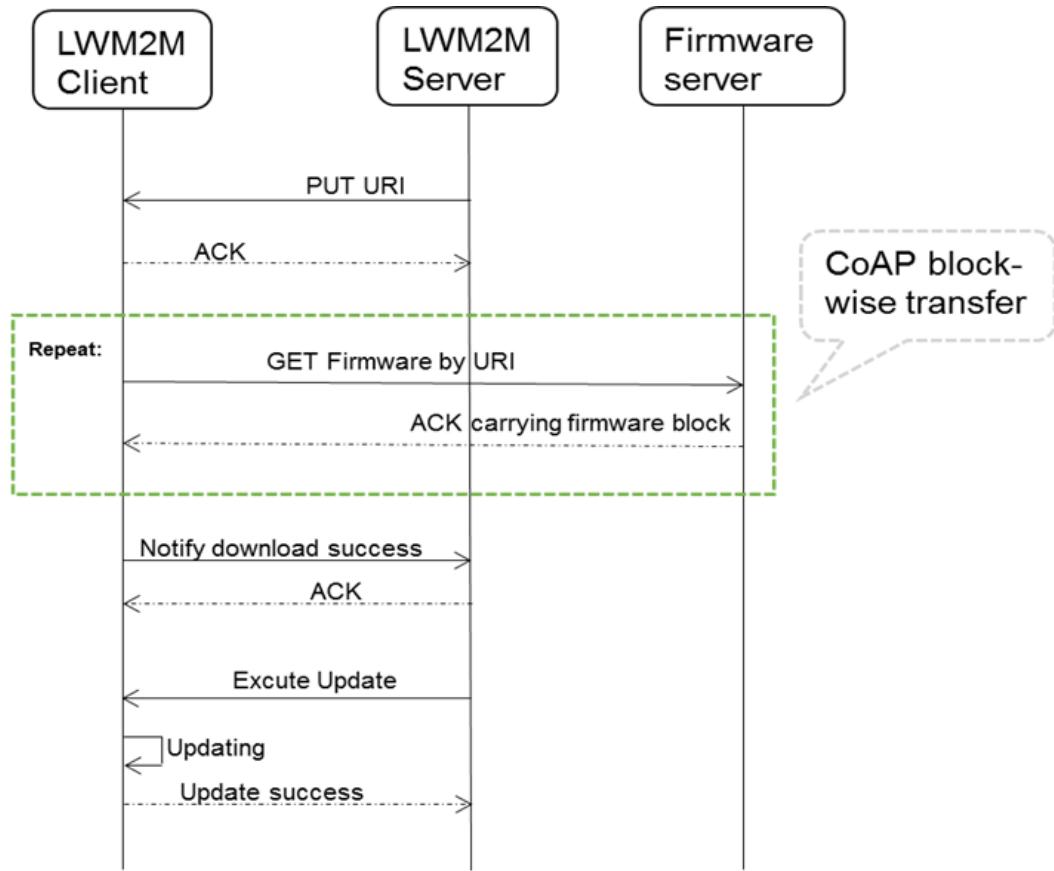
图 6-56 固件升级状态图



## 固件升级流程

固件升级流程如图所示：

图 6-57 固件升级流程



## 6.3 Agent Lite SDK 集成开发指导

### 6.3.1 直连场景

终端设备集成Agent Lite SDK，直接接入到物联网平台。当终端设备具备IP能力，且满足Agent Lite SDK的运行环境时，适用于该场景。

在直连场景下，Agent Lite SDK的集成对接流程为：初始化>绑定>登录>编译安装程序>上传Profile并注册设备>上线>数据上报和数据发布>命令接收和数据接收。

### 6.3.2 非直连场景

网关产品集成Agent Lite SDK，终端设备作为子设备连接到网关产品，并通过网关产品接入到物联网平台。当终端设备不具备IP能力，只支持近场通信（如Z-Wave、Zigbee）时，适用于该场景。

在非直连场景下，Agent Lite SDK的集成对接流程为：初始化>绑定>登录>编译安装程序>上传Profile并注册设备>上线>数据上报和数据发布>命令接收和数据接收>添加非直连设备>非直连设备状态更新>非直连设备数据上报。

### 6.3.3 C-Linux

### 6.3.3.1 开发者必读

按照本文档的指导，开发者可以体验直连设备通过集成Agent Lite快速接入平台，体验“数据上报”、“命令接收”、“添加非直连设备”等功能。

Agent Lite以SDK的形式嵌入第三方软件中。本文档以Agent Lite Linux Demo为例，指导开发者使用Agent Lite SDK中的接口，实现“直连设备登录”、“数据上报”和“命令下发”等功能。

- 开发者可以基于Agent Lite Linux Demo开发，也可参考Agent Lite Linux Demo，自行集成Agent Lite SDK(Linux)。
- Agent Lite Linux Demo使用的IDE工具为**Visual Studio**。开发者可以使用其他IDE工具。

### 6.3.3.2 前期准备

#### 6.3.3.2.1 获取物联网平台对接信息

请参考[业务使用全流程](#)完成入驻华为云和开通开发中心服务，在开通开发中心后，在管理控制台会出现开发中心的入口，且系统将会通过短信和邮件发送开发中心的账号和密码给您。登录开发中心后，可以在相应项目的“对接信息”界面，查看应用和设备的接入地址。



#### 6.3.3.2.2 下载开发资源

下载Agent Lite SDK集成开发的相关资源：

- [Agent Lite API参考\(C\)](#)
- [Agent Lite Demo\(C-Linux\)](#)
- [Agent Lite SDK\(Linux通用\)或Agent Lite SDK\(工具链\)](#)



请根据工具链名称选择合适的Agent Lite SDK。如果没有与自己设备匹配的工具链类型，则请将工具链提供给华为工程师，由华为工程师编译相应的SDK。

#### 6.3.3.2.3 证书校验和权限控制

对于Agent Lite的权限控制，数字证书会根据用户身份给予相应的访问资源的权限，TLS证书在校验时使用，当前默认采用证书校验模式，只有导入了数字证书并通过了证书校验才能进行登录的流程。如果采用无证书校验模式，数据传输通道可能会导致不安全的行为。数字证书是类似于钥匙一样的数字凭证，可以增强用户的使用安全性。密钥不允许明文保存在本地，密钥材料及密钥组件在本地保存时必须严格控制其权限（如：400或600），密钥在本地加密时应进行MAC校验或数字签名。密钥组件的文件名不能包含“加密算法”、“密钥长度”、“密钥组件序号”、“密钥组件总数”、

“密钥名称”、“组件名称”、“密钥组件构成”、“与其它密钥组件的关系”、“加密模式”、“域参数”等信息（如：Key\_AES128\_Component3）。对于本地存储（本地文件、数据库等）的密钥（通常以密文形式存在），应被完全删除（包括密钥的全部副本）并对相应的磁盘空间重复写入其他数据（推荐写“0”或“1”覆盖三次或以上）。对于产品预置的证书，要使用X.509v3格式的数字证书，要使用的签名算法是“SHA256RSA”或“SHA256ECDSA”。

### 6.3.3.3 开发环境

#### 开发环境要求

硬件规格要求

CPU	RAM	FLASH	电池
暂无	>4M	>600KB	带电设备

#### 工程目录结构及文件说明

工程目录结构及文件说明

目录结构	目录	说明
Agent Lite	include	存放Agent Lite头文件
	lib	存放Agent Lite编译后的库文件和第三方库文件
└── linux	demo	存放演示工程源文件和头文件
└── include	conf	存放TLS证书文件
└── demo	test	用于交叉编译验证的测试程序
└── lib		
└── platform1		
└── platform2		
└── test		
└── platform1		
└── platform2		
└── workdir		
└── conf		



#### 说明

如果开发者没有设备，可以直接在X86 Linux系统进行开发。

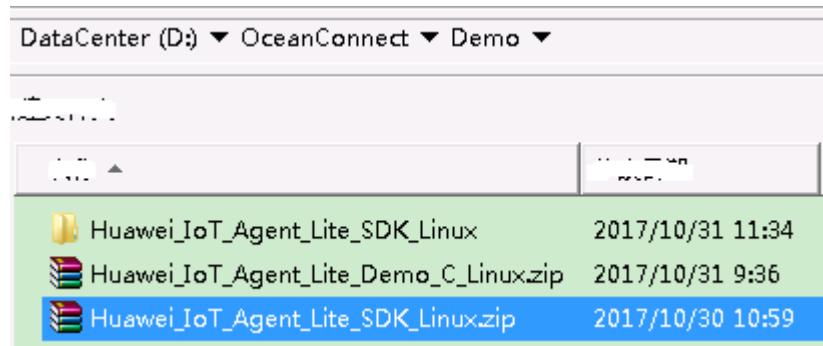
### 6.3.3.4 交叉编译环境检测

**步骤1** 准备网关或设备。开发者需要根据[开发环境](#)章节，确认待集成的网关或设备（传感器）的开发环境是否满足要求。

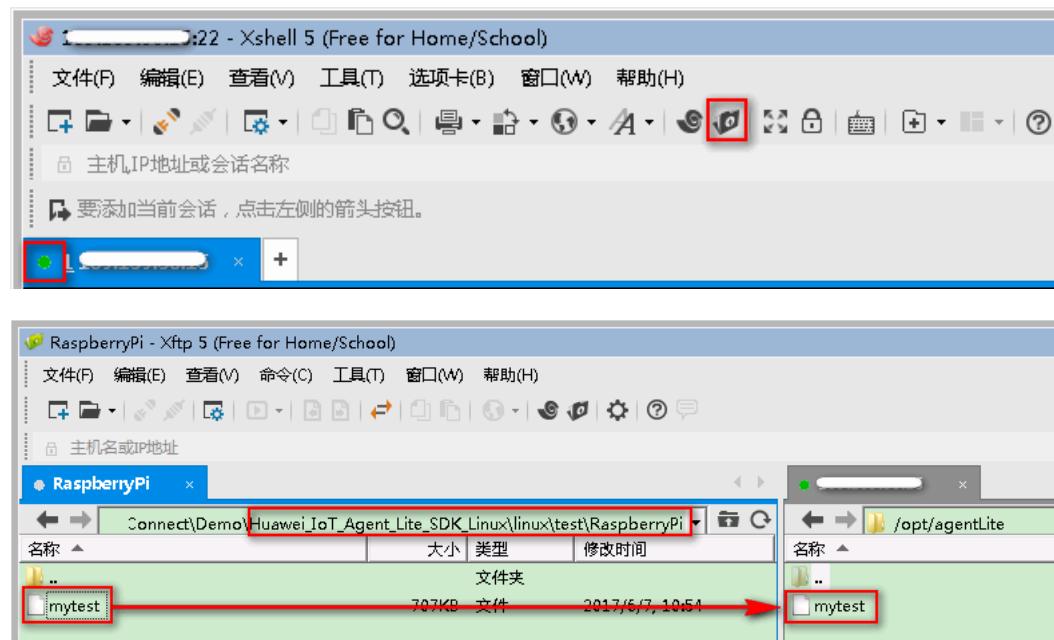
### 说明

本文档将以树莓派为例，说明如何集成网关。

#### 步骤2 将Agent Lite SDK (Linux)解压到本地。



#### 步骤3 用Xshell登录树莓派，再使用Xshell的Xftp功能把mytest传到树莓派上。



### 说明

- 如果没有安装Xftp，请先安装Xftp再上传文件到树莓派上。
- 因为使用的设备是树莓派，所以使用RaspberryPi目录下的mytest测试工具。开发者可以选择根据实际情况选择不同目录下的mytest测试工具进行测试，尽量选择与自己的系统信息相近的目录下的测试工具进行测试。

#### 步骤4 修改所有文件的权限，进入mytest所在目录，运行“mytest”。

```
cd /opt/agentLite  
chmod -R 777 *  
./mytest
```

如果最后出现“AUTO TEST END SUCC”字样，说明通过检测。

```
Case 1023: ULogin_TestOnRecvSALogoutCheck() 67 ..... SetOK
Case 1023: ..... OK
Case 1024: ULogin_TestOnRecvSALoginCheck() 99 ..... SetOK
Case 1024: ..... OK
Test OK <--- ULogin_TestLocal() .....
===== Check Mem Leak =====
===== AUTO TEST END SUCC =====
```

#### 说明

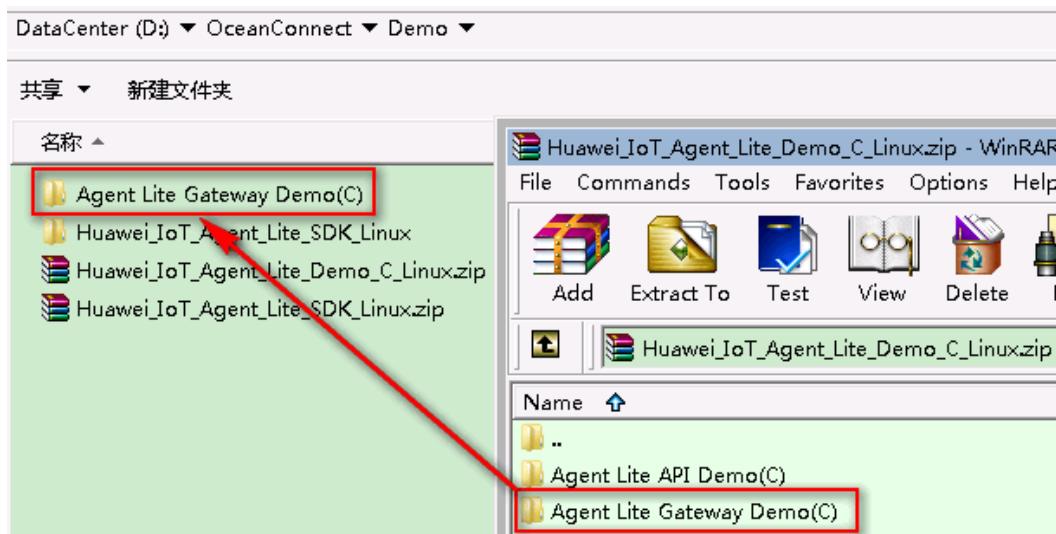
如果检测不通过，开发者需要提交交叉编译工具给华为工程师，并说明交叉编译工具运行的环境要求。华为工程师会使用提交的交叉编译工具编译出新Agent Lite SDK库，发送到开发者的邮箱。

----结束

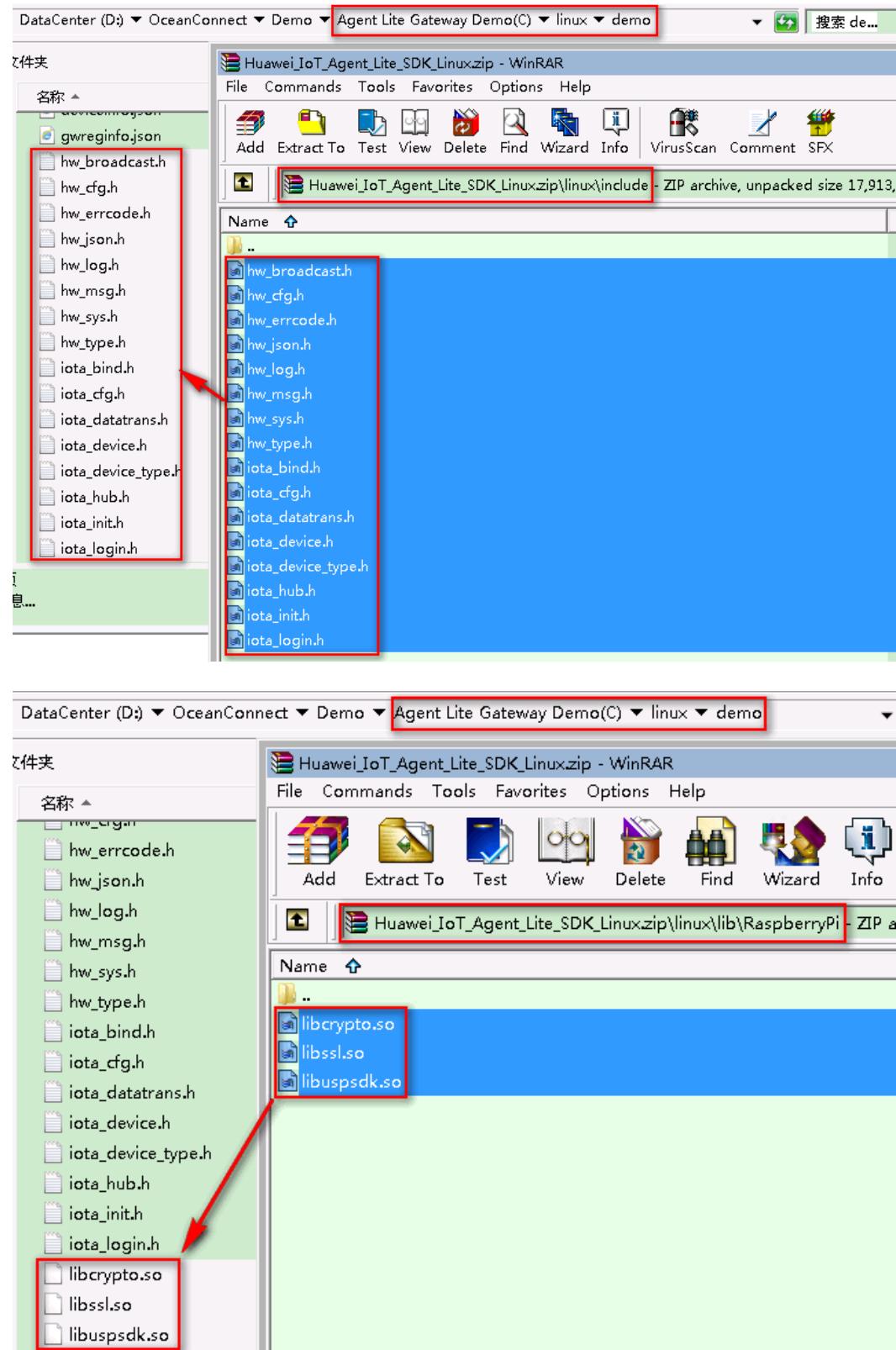
### 6.3.3.5 导入样例代码

在交叉编译环境检测通过后，再进行设备集成开发。下面将以Agent Lite Gateway Demo(C)中的样例代码为例，说明如何进行Agent Lite SDK的集成。

**步骤1** 将Agent Lite Demo (C-Linux)压缩包中的Agent Lite Gateway Demo(C)解压到本地。



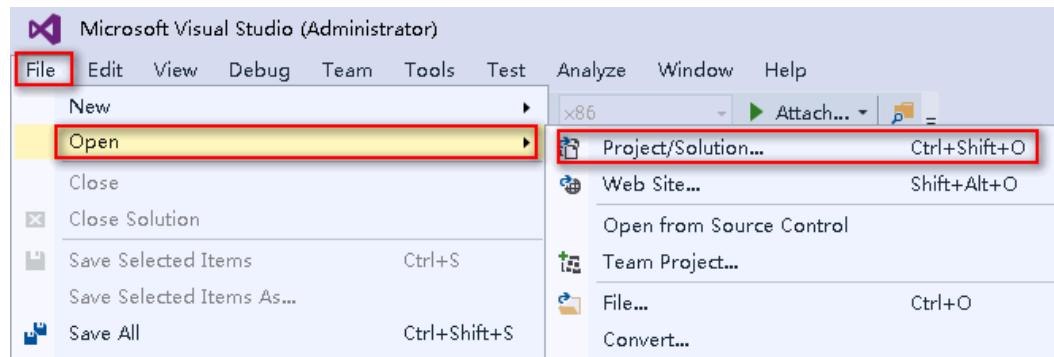
**步骤2** 将SDK库中的头文件和so库都放到“Agent Lite Gateway Demo(C)/linux/demo”目录下。



### 说明

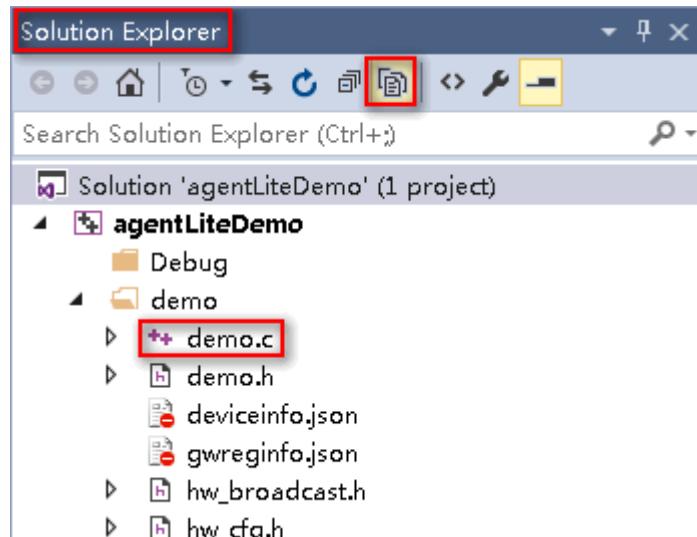
因为使用的设备是树莓派，所以使用RaspberryPi目录下的库。

**步骤3** 打开Visual Studio，选择“File > Open > Project/Solution”。



**步骤4** 进入“Agent Lite Gateway Demo(C)/linux”目录，选择“agentLiteDemo.sln”，点击“打开”按钮即可。

在Solution Explorer窗口中，双击打开“demo.c”文件。



----结束

### 6.3.3.6 初始化

在发起业务前，需要先初始化Agent Lite相关资源，调用API接口**IOTA\_Init()**，初始化Agent Lite资源。具体API接口的参数使用请参考Agent Lite API接口文档。

```
IOTA_Init(const CONFIG_PATH, const HW_NULL);
```

#### 说明

- 请参考demo.c中**main()**方法对**IOTA\_Init()**的调用。
- “**CONFIG\_PATH**”为工作路径，不能为空，该参数必须带结束符‘\0’。
- 第二个参数为打印日志路径，当它为空时，打印路径默认为工作路径。开发者也可以自己定义打印日志路径，该参数必须带结束符‘\0’。

### 6.3.3.7 绑定

设备或网关第一次接入物联网平台时需要进行绑定操作，从而将设备或网关与平台进行关联。开发者通过传入设备序列号以及设备信息，将设备或网关绑定到物联网平台。

绑定前先调用API接口**IOTA\_ConfigSetXXX()**设置绑定服务器（即物联网平台）的IP与端口，接着调用API接口**IOTA\_Bind()**进行绑定，主要入参为MAC地址（即“设备的标识”）和必要的设备信息（包括“nodeId”、“厂家Id”、“设备类型”、“设备型号”、“协议类型”，其中“nodeId”的值与MAC地址保持一致）。

调用**IOTA\_ConfigSetXXX()**设置绑定参数（平台的IP和端口）。

```
IOTA_ConfigSetStr(EN_IOTA_CFG_IOCM_ADDR, const pucPlatformAddr);  
IOTA_ConfigSetUint(EN_IOTA_CFG_IOCM_PORT, uiPort);
```

注册广播接收器对设备绑定结果进行相应处理。

```
HW_BroadCastReg(IOTA_TOPIC_BIND_RSP, Device_RegResultHandler);
```

调用**IOTA\_Bind()**绑定设备。

```
HW_UINT DEVICE_BindGateWay()  
{  
    .....  
    stDeviceInfo.pcMac = HW_JsonGetStr(json, IOTA_DEVICE_MAC);  
    stDeviceInfo.pcNodeId = stDeviceInfo.pcMac;  
    stDeviceInfo.pcManufacturerId = HW_JsonGetStr(json, IOTA_MANUFACTURE_ID);  
    stDeviceInfo.pcDeviceType = HW_JsonGetStr(json, IOTA_DEVICE_TYPE);  
    stDeviceInfo.pcModel = HW_JsonGetStr(json, IOTA_MODEL);  
    stDeviceInfo.pcProtocolType = HW_JsonGetStr(json, IOTA_PROTOCOL_TYPE);  
    .....  
    IOTA_ConfigSetStr(EN_IOTA_CFG_IOCM_ADDR, pucPlatformAddr);  
    IOTA_ConfigSetUint(EN_IOTA_CFG_IOCM_PORT, uiPort);  
    IOTA_Bind(stDeviceInfo.pcMac, &stDeviceInfo);  
}
```

#### 说明

- 设备或网关绑定成功，后续就不再需要再绑定了，除非设备或网关被删除，才需要重新绑定。
- “stDeviceInfo”存储设备信息，“stDeviceInfo.pcMac”为设备唯一标识符。
- 绑定时使用的设备固有信息（如设备型号等）是从“gwreginfo.json”文件中读取的，所以需要修改demo目录下的“gwreginfo.json”。修改“gwreginfo.json”中的如下信息：
  - “platformaddr”：物联网平台地址，可以从开发中心的“对接信息”界面获取。
  - “mac”：MAC地址，每个设备对应一个MAC地址，不可重复，所以建议使用IMEI或者MAC地址等天然的设备标识。测试时只要输入一个没有使用过的MAC地址即可）。
  - “manufacturerId”、“deviceType”、“model”和“protocolType”：与Profile文件中的定义保持一致。
- 设备绑定成功会收到广播，广播内容请参考[设备绑定](#)接口的返回结果说明和demo中[Device\\_RegResultHandler](#)函数的处理。[Device\\_RegResultHandler](#)把从广播中得到的网关设备信息保存在“gwbndinfo.json”文件中。

gwreginfo.json文件中设备固有示例：

```
1  {  
2      "mac": "1.....6",  
3      "platformAddr": "1.....",  
4      "platformPort": 8943,  
5      "manufacturerId": "Huawei",  
6      "deviceType": "Gateway",  
7      "model": "AgentLite01",  
8      "protocolType": "HuaweiM2M",  
9      "loglevel": 255  
10 }
```

### 6.3.3.8 登录

设备或网关绑定成功后或重启后，需要进行登录的流程，在设备或网关成功登录物联网平台后，才可以进行其它服务操作，比如接入其他传感器，数据上报等等。如果设备或网关登录成功，那么设备或网关在平台的状态显示为已在线。

登录前需要通过参数配置API接口**IOTA\_ConfigSetXXX()**传入所需的登录信息，再调用API接口**LoginService.login()**进行登录平台的流程，具体API的参数使用参考Agent Lite API接口文档中的[设备登录](#)接口说明。

**设置登录参数：**

```
IOTA_ConfigSetStr(EN_IOTA_Cfg_DeviceID, g_stGateWayInfo.pcDeviceID);
IOTA_ConfigSetStr(EN_IOTA_Cfg_IoCM_Addr, g_stGateWayInfo.pcIoCMAddr);
IOTA_ConfigSetStr(EN_IOTA_Cfg_AppID, g_stGateWayInfo.pcAppID);
IOTA_ConfigSetStr(EN_IOTA_Cfg_DeviceSecret, g_stGateWayInfo.pcSecret);
IOTA_ConfigSetStr(EN_IOTA_Cfg_MQTT_Addr, g_stGateWayInfo.pcIoCMAddr);
IOTA_ConfigSetUint(EN_IOTA_Cfg_MQTT_Port, g_stGateWayInfo.pcMqttPort);
IOTA_ConfigSetUint(EN_IOTA_Cfg_IoCM_Port, g_stGateWayInfo.pcIoCMPort);
```

**说明**

- “设备Id”（“EN\_IOTA\_Cfg\_DeviceID”），“appId”（“EN\_IOTA\_Cfg\_AppID”）和“密码”（“EN\_IOTA\_Cfg\_DeviceSecret”）从绑定成功的广播中得到的。
- “HTTP地址”（“EN\_IOTA\_Cfg\_IoCM\_Addr”）和“MQTT地址”（“EN\_IOTA\_Cfg\_MQTT\_Addr”）一般为同一个地址，可以从绑定成功的广播中得到。一般情况下，这个地址和Agent Lite设备或网关对接的平台地址一致。
- 绑定成功的广播参数获取可以参考**Device\_RegResultHandler**函数的处理。

注册广播接收器对设备登录结果进行相应处理。

```
HW_BroadCastReg(IOTA_TOPIC_CONNECTED_NTY, Device_ConnectedHandler);
```

调用**LoginService.login()**进行直连设备登录。

```
IOTA_Login();
```

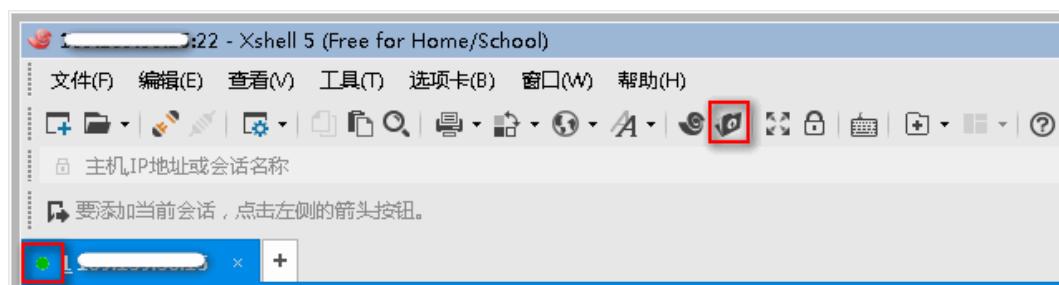
**说明**

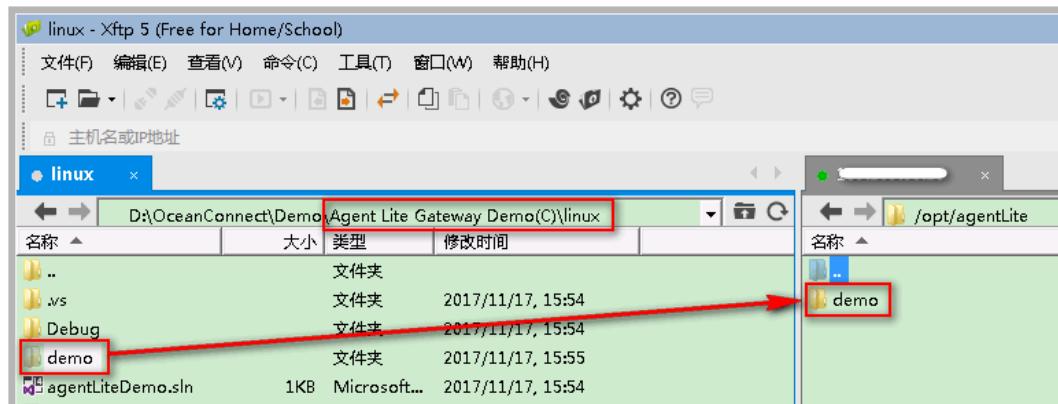
开发者可以参考附录中的[流程图](#)，以便对整体流程有更全面的理解。

### 6.3.3.9 编译并运行程序

绑定和登录功能完成后，可以先测试一下设备或网关是否能正常与平台对接，再进行后续功能开发。

**步骤1** 用Xshell登录树莓派，再使用Xshell的Xftp功能把demo目录传到树莓派上。



**步骤2** 进行交叉编译。

1. 进入“/opt/agentLite/demo/”路径。
2. 输入“chmod -R 777 \*”命令修改权限。
3. 输入“export LD\_LIBRARY\_PATH=.”设置程序共享库位置。
4. 输入“gcc demo.c -L /opt/agentLite/demo/ -lpthread -ldl -lrt -luspdk -lssl -lcrypto -o sdk.out”命令，如果交叉编译成功，会产生sdk.out库文件。

```
root@raspberrypi:/ $cd /opt/agentLite/demo/
root@raspberrypi:/opt/agentLite/demo $chmod -R 777 *
root@raspberrypi:/opt/agentLite/demo $export LD_LIBRARY_PATH=.
root@raspberrypi:/opt/agentLite/demo $gcc demo.c -L /opt/agentLite/demo/ -lpthread -ldl -lrt -luspdk -lssl -lcrypto -o sdk.out
root@raspberrypi:/opt/agentLite/demo $
```

**说明**

- “/opt/agentLite/demo/” 路径需要根据实际情况修改。
- “luspdk”、“lssl” 和 “lcrypto” 分别表示库文件 “libuspdk.so”、“libssl.so” 和 “libcrypto.so”，具体linux交叉编译命令gcc请开发者自行学习。

**步骤3** 运行Demo。

启动demo命令为“./ sdk.out”，输入该命令程序就能运行起来了。

**说明**

程序运行后，Agent Lite SDK就会主动发送绑定消息给平台。因为此时还没有使用北向接口在平台上注册此设备，所以会提示“The device has not registered yet”。

----结束

**6.3.3.10 上传 Profile 并注册设备**

下载**Profile模板**，并上传模板中的profile文件：“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

**步骤1** 登录SP Portal，进入“设备管理”，选择对应的应用，选择“产品模型”，点击右上角“+新增产品模型”下的“本地导入”，进入“手动导入产品”界面。

**图 6-58 导入产品模型**



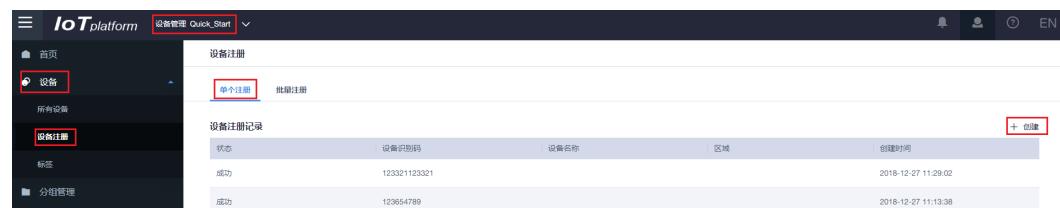
**步骤2** 在弹出窗口中输入“产品名称”，添加“资源文件”，点击“确定”，将Profile导入到物联网平台。

图 6-59 手动导入产品



**步骤3** 选择“设备管理 > 设备 > 设备注册 > 单个注册”，点击右上角“+创建”，进入“单个设备注册”页面。

图 6-60 注册单个设备



**步骤4** 在弹出窗口中填写设备必要的信息。

图 6-61 填写设备基本信息

设备注册 > 单个设备注册

基本信息

\*注册类型 动态密码模式  
该模式下，推荐在平台配置设备的“设备标识码”（标识码要求唯一，如IMEI、MAC地址等），设备在接入平台时携带该标识码完成接入验证。

\*设备识别码 aa123456789

\*设备定义 自定义 产品

\*设备类型 WaterMeter

\*厂商名称 HZYB

\*生产商ID TestUHFManuld

\*设备型号 TestUHFModelM2M

\*协议类型 LWM2M

可选信息

安全设备 请选择  
安全设备由5684端口接入,非安全设备由5683端口接入

设备名称 (数字,字母,中文,\_,<=256字符)

提交 取消

### 说明

- “设备名称”可以随意取。
- “注册类型”有“动态密码模式”、“初始密码模式”，默认选择“动态密码模式”。
- “设备标识”（NodeID）需要与AgentLiteDemo所运行的设备上设置的设备标识（即MAC地址）一致。
- “设备类型”、“厂商名称”、“厂商ID”、“协议类型”与profile中的定义保持一致即可。
- 其他信息根据实际情况填写，不影响业务功能。非必要信息不必填写。

----结束

### 6.3.3.11 上线

注册设备后，如果Agent Lite SDK还在发送bind消息，则在SP Portal “设备管理 > 设备 > 所有设备” 页面可以看到设备状态变成“在线”。



### 说明

如果demo日志中出现“lota\_BindDestroy”，说明Agent Lite SDK多次发送bind消息，并且没有收到正确响应，设备绑定已超时。这时需要再次运行“sdk.out”程序（先用“ctrl+c”退出，再用“./sdk.out”运行程序），Agent Lite SDK就会再发起bind消息。平台收到正确的bind消息后设备就会变成“在线”状态。

### 6.3.3.12 数据上报和数据发布

### 6.3.3.12.1 使用说明

设备或网关向物联网平台上报数据可以通过调用SDK的“设备服务数据上报”接口或“数据发布”接口：

- “设备服务数据上报”接口：topic固定为“/cloud/signaltrans/v2/categories/data”；pcDeviceId、pcRequestId和pcServiceId由SDK组装为消息的header；pcServiceProperties由SDK组装为消息的body。消息组装格式为JSON。
- “数据发布”接口：topic作为入参可以调整，可以为“/cloud/signaltrans/v2/categories/data”，也可以传入其他topic；“pbstrServiceData”参数作为消息体（包括header和body），SDK只进行透传，不进行格式调整和组装。

因此：当开发者希望设备或网关使用SDK的固定topic（“/cloud/signaltrans/v2/categories/data”）和固定格式（JSON）上报数据时，可以调用[设备服务数据上报](#)接口；当开发者希望设备或网关使用自定义的topic和格式上报数据时，可以调用[数据发布](#)接口。

### 6.3.3.12.2 设备服务数据上报

设备或网关登录成功后可以调用[IOTA\\_ServiceDataReport](#)接口上报数据。

```
HW_VOID Gateway_DataReport(HW_CHAR **pcJsonStr)
{
    HW_JSON json;
    HW_JSONOBJ hJsonObj;

    hJsonObj = HW_JsonObjCreate();
    json = HW_JsonGetJson(hJsonObj);
    HW_JsonAddUInt(json, (HW_CHAR*)"storage", (HW_INT)10240);
    HW_JsonAddUInt(json, (HW_CHAR*)"usedPercent", (HW_INT)20);
    *pcJsonStr = HW_JsonEncodeStr(hJsonObj);
    Device_ServiceDataReport(g_stGateWayInfo.pcDeviceID, "Storage", pcJsonStr);
    //another method
    //Device_ServiceDataReport(pcDeviceId, "Storage", "{\"storage\":2040,\"usedPercent\":20}");
}

HW_INT Device_ServiceDataReport(const HW_CHAR *pcSensorDeviceID, const HW_CHAR *pcServiceId, const
HW_CHAR *pcServiceProperties)
{
    HW_CHAR azsRequestId[BUFF_MAX_LEN];

    HW_GetRequestId(aszRequestId);

    if (HW_TRUE != g_uiLoginFlg)
    {
        //TODO e.g. resend service data
        HW_LOG_INF("Device_MApiMsgRcvHandler():GW discon, pcJsonStr=%s", pcServiceProperties);
        return HW_ERR;
    }

    IOTA_ServiceDataReport(HW_GeneralCookie(), azsRequestId, pcSensorDeviceID, pcServiceId,
pcServiceProperties);
    return HW_OK;
}
```

注册广播接收器对网关数据上报结果进行相应处理。

```
sprintf(acBuf, "%s/%s", IOTA_TOPIC_DATATRANS_REPORT_RSP, g_stGateWayInfo.pcDeviceID);
HW_BroadCastReg(acBuf, Device_ServiceDataReportResultHandler);
```

## 说明

- 广播过滤参数为“`IOTA_TOPIC_DATATRANS_REPORT_RSP/{deviceId}`”。
- 当设备主动上报数据时，“`pcRequestId`”可以为空。
- 当上报的数据为某个命令的响应时，“`pcRequestId`”必须与下发命令中的`requestId`保持一致。`“pcRequestId”`可以从广播中获取，请参考API文档中“设备命令接收”接口的“`EN_IOTA_DATATRANS_IE_REQUESTID`”参数。
- “`pcServiceID`”要与Profile中定义的某个“`serviceId`”保持一致，否则无法上报数据。
- “`pcServiceData`”实际上是一个JSON字符串，内容是键值对（可以有多组键值对）。每个键是Profile中定义的属性名（`propertyName`），值就是具体要上报的数据。

### 6.3.3.12.3 数据发布

设备或网关登录成功后可以调用**IOTA\_MqttDataPub**接口发布数据。

```
HW_INT Device_ServiceDataPub(const HW_UCHAR *pucTopic, HW_UINT uiQos, const HW_BYTES
*pbstrServiceData)
{
    IOTA_MqttDataPub(HW_GeneralCookie(), pucTopic, uiQos, pbstrServiceData);
    return HW_OK;
}
```

注册广播接收器对网关数据上报结果进行相应处理。

```
HW_BroadCastReg(IOTA_TOPIC_DATATRANS_PUB_RSP, Device_ServiceDataPubResultHandler);
```

## 说明

- 广播过滤参数为“`IOTA_TOPIC_DATATRANS_PUB_RSP`”。
- “`pucTopic`”为发布数据的topic。
- “`uiQos`”为mqtt协议中的一个参数。
- “`pbstrServiceData`”实际上是一个json字符串，内容是键值对（可以有多组键值对）。每个键是profile中定义的属性名（`propertyName`），值就是具体要上报的。

### 6.3.3.13 命令接收和数据接收

#### 6.3.3.13.1 使用说明

设备或网关从物联网平台接收数据可以通过调用SDK的“设备命令接收”接口或“数据接口”接口：

- “设备命令接收”接口：只接收topic为“`/gws/deviceid/signaltrans/v2/categories/`”的消息，并对消息中的header进行解析。
- “数据接收”接口：接收topic订阅过的消息，且不对接收到的消息进行解析，仅进行透传。

因此：当开发者希望设备或网关只接收topic为“`/gws/deviceid/signaltrans/v2/categories/`”的消息，且对消息中的header进行解析时，可以调用**设备命令接收**接口；当开发者希望设备或网关接收topic订阅过的消息，且仅对消息进行透传时，可以调用**数据接收**接口。

#### 6.3.3.13.2 设备命令接收

应用服务器可以调用物联网平台的应用侧API接口给设备或网关下发命令，所以设备或网关需要随时检测命令下发的广播，以便在接收到命令时进行相应业务处理。

注册**IOTA\_TOPIC\_SERVICE\_COMMAND\_RECEIVE/{deviceId}**广播接收器对命令下发进行相应处理。

```
sprintf(acBuf, "%s/%s", IOTA_TOPIC_SERVICE_COMMAND_RECEIVE, g_stGateWayInfo.pcDeviceID);  
HW_BroadCastReg(acBuf, Device_ServiceCommandReceiveHandler);
```

### 具体的处理函数Device\_ServiceCommandReceiveHandler。

```
HW_INT Device_ServiceCommandReceiveHandler(HW_UINT uiCookie, HW_MSG pstMsg)  
{  
    HW_CHAR *pcDevId;  
    HW_CHAR *pcReqId;  
    HW_CHAR *pcServiceId;  
    HW_CHAR *pcMethod;  
    HW_BYTES *pbstrContent;  
  
    pcDevId = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_DEVICEID);  
    pcReqId = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_REQUESTID);  
    pcServiceId = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_SERVICEID);  
    pcMethod = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_METHOD);  
    pbstrContent = HW_MsgGetBstr(pstMsg, EN_IOTA_DATATRANS_IE_CMDCONTENT);  
  
    HW_LOG_INF(" ----- CommandReceiveHandler ----- DeviceId=%s", pcDevId);  
  
    if ((HW_NULL == pcDevId) || (HW_NULL == pcReqId) || (HW_NULL == pcServiceId) || (HW_NULL ==  
pcMethod))  
    {  
        HW_LOG_ERR("RcvCmd is invalid, pcDevId=%s, pcReqId=%s, pcServiceId=%s, pcMethod=%s.",  
pcDevId, pcReqId, pcServiceId, pcMethod);  
        return HW_ERR;  
    }  
  
    if (0 == strncmp(METHOD_REMOVE_GATEWAY, pcMethod, strlen(METHOD_REMOVE_GATEWAY)))  
    {  
        IOTA_RmvGateWay();  
    }  
  
    return HW_OK;  
}
```

#### 说明

- 物联网平台下发命令的接口可以查看应用侧API接口文档中的[设备服务调用](#)接口。
- 网关接收到的命令可能是给网关本身的，也可能是给网关下面的某个非直连设备，网关要根据具体的命令和业务场景做出具体的响应。

在SP Portal “设备列表 > 设备详情 > 子设备”界面中，点击“删除”子设备的按钮，这样就能在demo界面上看到广播接收时的日志打印命令下发，子设备添加详见[添加非直连设备](#)。

图 6-62 命令下发-“删除非直连设备”



#### 说明

非直连设备的删除需要网关的确认，正常业务情况下，网关又需要跟具体的设备确认，所以收到删除非直连设备的命令也不会立刻将设备删除。

### 6.3.3.13.3 数据接收

应用服务器可以调用物联网平台的应用侧API接口给设备或网关下发数据，所以设备或网关需要随时检测数据下发的广播，以便在接收到数据时进行相应业务处理。

在调用SDK的“数据接收”接口前，需要先调用SDK的“Topic订阅”接口向物联网平台订阅相应topic的数据。完成订阅后，物联网平台才可以将该topic的消息发送给设备或网关。订阅topic方法详见[Topic的订阅](#)。

注册IOTA\_MSG\_RECV\_RSP广播接收器对命令下发进行相应处理。

```
HW_BroadCastReg(IOTA_MSG_RECV_RSP, Device_DevRecvMsgHandler);
```

具体的处理函数Device\_DevRecvMsgHandler:

```
Device_DevRecvMsgRspHandler(HW_UINT uiCookie, HW_MSG pstMsg)
{
    HW_UCHAR *topic=HW_NULL;
    HW_BYTEWS *body=HW_NULL;
    topic = HW_MsgGetStr(pstMsg, EN_IOTA_DATATRANS_IE_MSGTOPIC);
    body = HW_MsgGetBstr(pstMsg, EN_IOTA_DATATRANS_IE_MSGCONTENT);
    if(topic!= HW_NULL && body!= HW_NULL)
    {
        HW_LOG_INF(" ---RecvMsg success ---");
        HW_LOG_INF("RecvMsg is:topic=%s,MsgContent=%s.", topic, body->pcByte);
    }
    else
    {
        HW_LOG_ERR(" ---RecvMsg failed ---");
        return HW_ERR;
    }
    return HW_OK;
}
```

#### 说明

网关接收到的数据可能是给网关本身的，也可能是给网关下面的某个非直连设备，此接口只是作为下发数据透传的接口，具体数据的功能需要设备或网关自己解析，并做相应的处理。

### 6.3.3.14 添加非直连设备

在设备或网关登录成功后，可以调用**IOTA\_HubDeviceAdd**接口添加非直连设备。

```
HW_VOID AddSensors()
{
    ST_IOTA_DEVICE_INFO stDeviceInfo = {0};
    FILE *fp = NULL;
    HW_CHAR szDeviceInfoFileName[BUFF_MAX_LEN] = {0};
    HW_INT file_size;
    HW_CHAR *pcJsonStr;
    HW_JSONOBJ jsonObj;
    HW_JSON json;

    //get device info
    stDeviceInfo.pcNodeId = "SN Number_8532157";
    stDeviceInfo.pcManufacturerName = "Huawei";
    stDeviceInfo.pcManufacturerId = "Huawei";
    stDeviceInfo.pcDeviceType = "Motion";
    stDeviceInfo.pcModel = "test01";
    stDeviceInfo.pcProtocolType = "Z-Wave";

    IOTA_HubDeviceAdd(g_uiCookie, &stDeviceInfo);
    return;
}
```

注册广播接收器对添加设备结果进行相应处理。

```
HW_BroadCastReg(IOTA_TOPIC_HUB_ADDDEV_RSP, Device_AddResultHandler);
```

### 说明

- 添加的非直连设备的profile已经上传，详见[上传Profile并注册设备步骤](#)。
- 此处非直连设备的设备固有信息是测试数据。真实情况下，网关需要与具体的非直连设备交互，才能得到非直连设备的固有信息。
- Demo在设备添加成功一段时间后会再调用删除设备接口进行设备删除。如果再次运行sdk.out前，还没执行删除设备接口，再次添加相同设备会失败。可以修改“NodeId”的值或者调用设备删除接口把原来的设备删除掉再进行测试。
- 添加非直连设备成功后就能从广播中得到非直连设备的deviceId。

非直连设备添加成功后可以在网关的“子设备”列表中看到一条记录。

图 6-63 添加非直连设备



### 6.3.3.15 非直连设备状态更新

非直连设备添加上时，一般情况下是“离线”状态。所以在非直连设备添加成功后，或者在非直连设备上报数据前，要调用**IOTA\_DeviceStatusUpdate()**进行设备状态更新。

```
IOTA_DeviceStatusUpdate(g_uiCookie, g_cDeviceId, "ONLINE", "NONE");
```

### 说明

- AgentLiteDemo中只添加了一个非直连设备，所以**IOTA\_DeviceStatusUpdate**中的**g\_cDeviceId()**直接使用全局变量。
- 设备状态包括“ONLINE”和“OFFLINE”两种。

注册广播接收器对非直连设备状态更新结果进行相应处理。

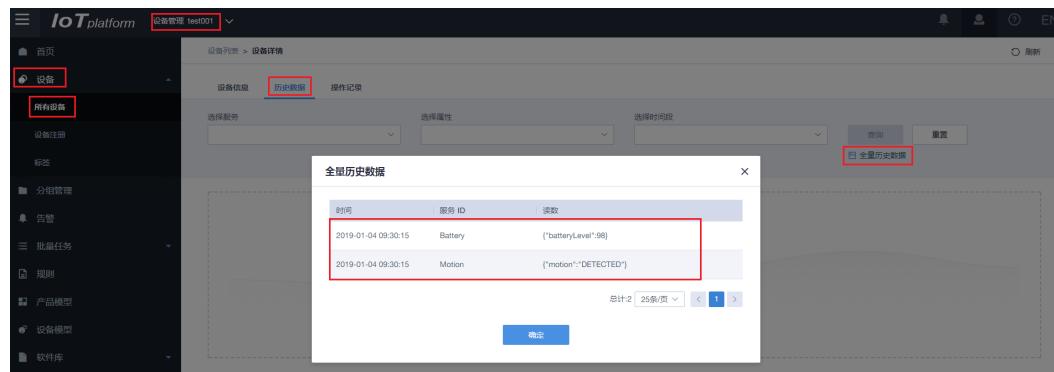
```
HW_BroadCastReg(IOTA_TOPIC_DEVUPDATE_RSP, Device_DevUpDateHandler);
```

### 6.3.3.16 非直连设备数据上报

请参考[数据上报和数据发布](#)章节，调用**IOTA\_ServiceDataReport**或**IOTA\_MqttDataPub**接口进行数据上报，各个参数使用非直连设备的相关数据即可，此处不再复述。

设备数据上报成功后，可以在非直连设备的“设备详情 > 全量历史数据”中查看上报的数据。

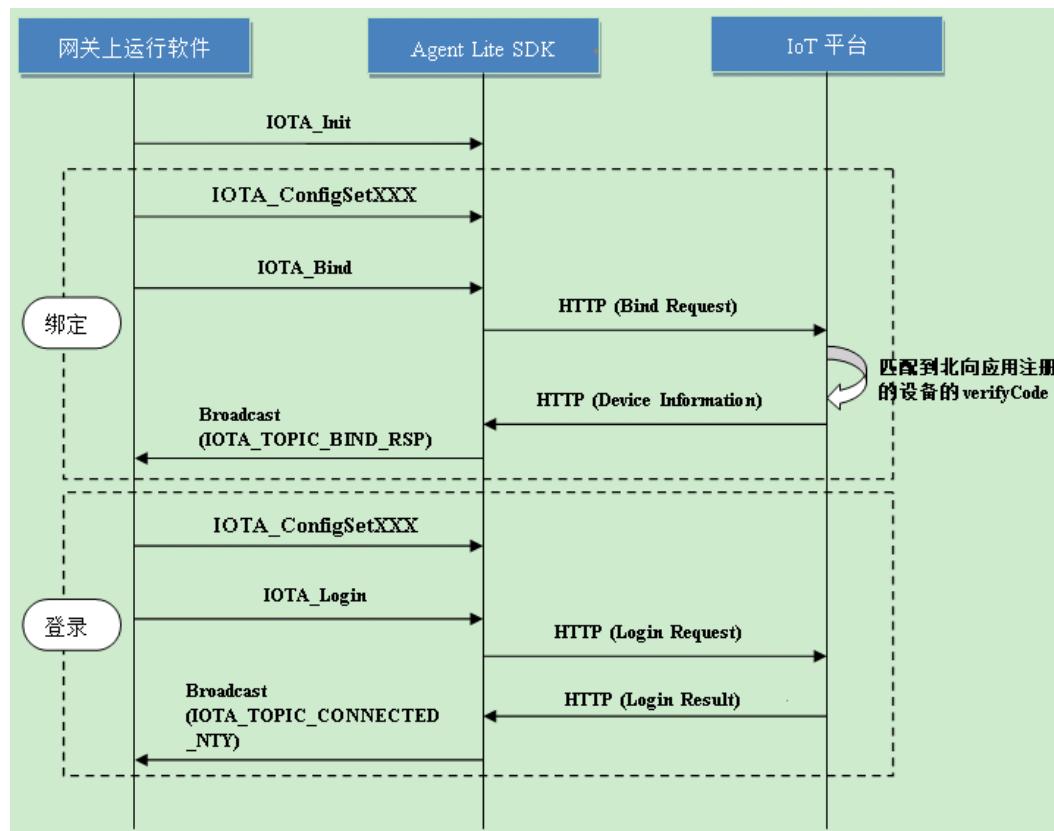
图 6-64 非直连设备查看上报的数据



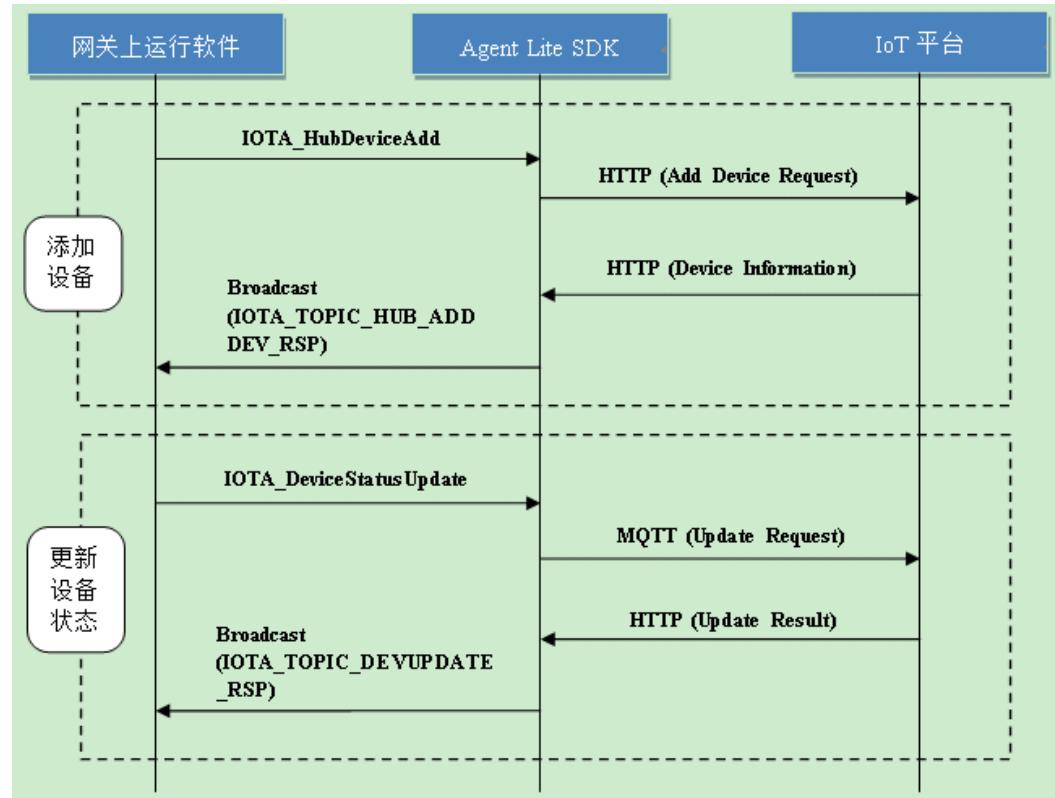
### 6.3.3.17 附录

#### 6.3.3.17.1 流程图

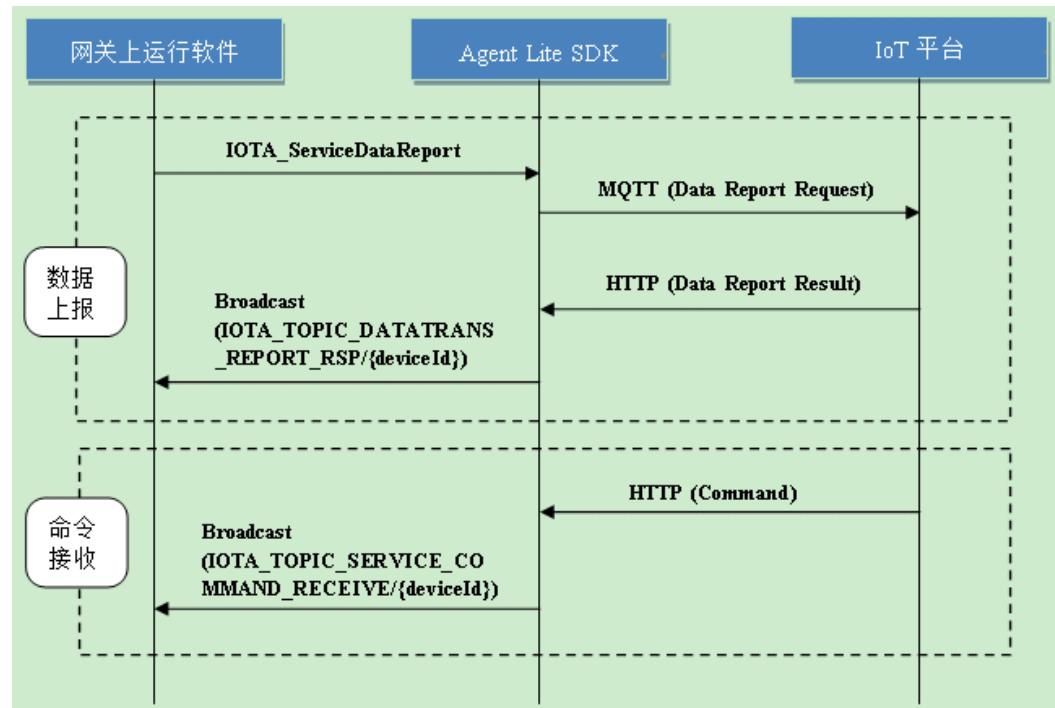
##### ?1. 绑定和登录



## ?.2. 添加非直连设备和设备状态更新



## ?.3. 数据上报和命令接收



### 6.3.3.17.2 Topic 的订阅

网关登录成功后可以调用 `IOTA_SubscribeTopic` 接口发布数据。

```
HW_INT Device_ServiceSubTopic(HW_UCHAR *pucTopic, HW_UINT uiQos)
{
    IOTA_SubscribeTopic(HW_GeneralCookie(), pucTopic, uiQos);
    return HW_OK;
}
```

注册广播接收器对网关数据上报结果进行相应处理。

```
HW_BroadCastReg(IOTA_TOPIC_SUB_RSP, Device_ServiceSubTopicResultHandler);
```

#### 说明

- 广播过滤参数为“IOTA\_TOPIC\_SUB\_RSP”。
- “pucTopic”为需要订阅的topic。
- “uiQos”为mqtt协议中的一个参数。

## 6.3.4 Java

### 6.3.4.1 开发者必读

按照本文档的指导，开发者可以体验直连设备通过集成Agent Lite快速接入平台，体验“数据上报”、“命令接收”、“添加非直连设备”等功能。

Agent Lite以SDK的形式嵌入第三方软件中。本文档以Agent Lite Java Demo为例，指导开发者使用Agent Lite SDK中的接口，实现“直连设备登录”、“数据上报”和“命令下发”等功能。

- 开发者可以基于Agent Lite Java Demo开发，也可参考Agent Lite Java Demo，自行集成Agent Lite SDK(Java)。
- Agent Lite java Demo使用的IDE工具为Eclipse,使用的SDK版本为jdk1.8.0\_45。

### 6.3.4.2 前期准备

#### 6.3.4.2.1 获取物联网平台对接信息

请参考[业务使用全流程](#)完成入驻华为云和开通开发中心服务，在开通开发中心后，在管理控制台会出现开发中心的入口，且系统将会通过短信和邮件发送开发中心的账号和密码给您。登录开发中心后，可以在相应项目的“对接信息”界面，查看应用和设备的接入地址。



#### 6.3.4.2.2 下载开发资源

下载Agent Lite SDK集成开发的相关资源：

- [AgentLite API参考\(Java\)](#)

### ● Agent Lite Demo(Java)



SDK在Demo的libs文件夹下。

#### 6.3.4.2.3 证书校验和权限控制

对于Agent Lite的权限控制，数字证书会根据用户身份给予相应的访问资源的权限，TLS证书在校验时使用，当前默认采用证书校验模式，只有导入了数字证书并通过了证书校验才能进行登录的流程。如果采用无证书校验模式，数据传输通道可能会导致不安全的行为。数字证书是类似于钥匙一样的数字凭证，可以增强用户的使用安全性。密钥不允许明文保存在本地，密钥材料及密钥组件在本地保存时必须严格控制其权限（如：400或600），密钥在本地加密时应进行MAC校验或数字签名。密钥组件的文件名不能包含“加密算法”、“密钥长度”、“密钥组件序号”、“密钥组件总数”、“密钥名称”、“组件名称”、“密钥组件构成”、“与其它密钥组件的关系”、“加密模式”、“域参数”等信息（如：Key\_AES128\_Component3）。对于本地存储（本地文件、数据库等）的密钥（通常以密文形式存在），应被完全删除（包括密钥的全部副本）并对相应的磁盘空间重复写入其他数据（推荐写“0”或“1”覆盖三次或以上）。对于产品预置的证书，要使用X.509v3格式的数字证书，要使用的签名算法是“SHA256RSA”或“SHA256ECDSA”。

#### 6.3.4.3 开发环境

##### 开发环境要求

硬件规格：

CPU	RAM	FLASH	电池
暂无	>4M	>600KB	带电设备

#### 工程目录结构及文件说明

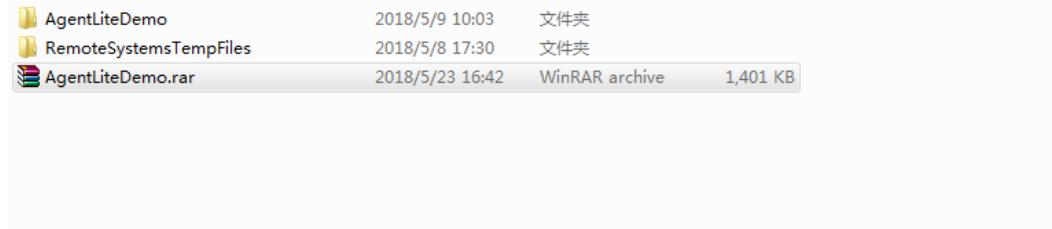
目录结构	目录	说明
AgentLiteDemo	src	存放Agentlite Demo代码。
     └── src         └── com         └── huawei         └── agentlitedemo	libs	存放Agentlite提供的jar包和第三方jar包，以及.dll动态库（工程必须的三个文件，“agentlite-0.0.1-SNAPSHOT”，“usp_agentlite.jar”，“.dll”文件）。
     └── libs         └── agentlite-0.0.1-SNAPSHOT             └── usp_agentlite.jar             └── .dll     └── workdir	workdir	存放工程日志文件，由agentlite初始化资源设置。
     └── workdir     └── conf	conf	存放TLS证书文件。

**说明**

如果开发者没有设备，可以直接在X86 Linux系统进行开发。

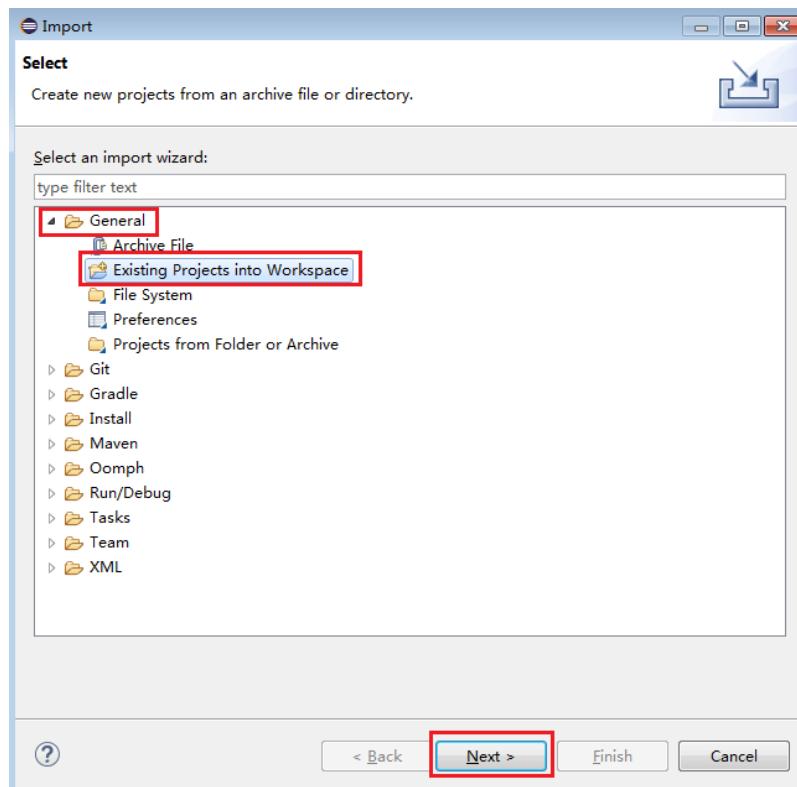
### 6.3.4.4 导入样例代码

**步骤1** 将AgentLiteDemo压到本地。

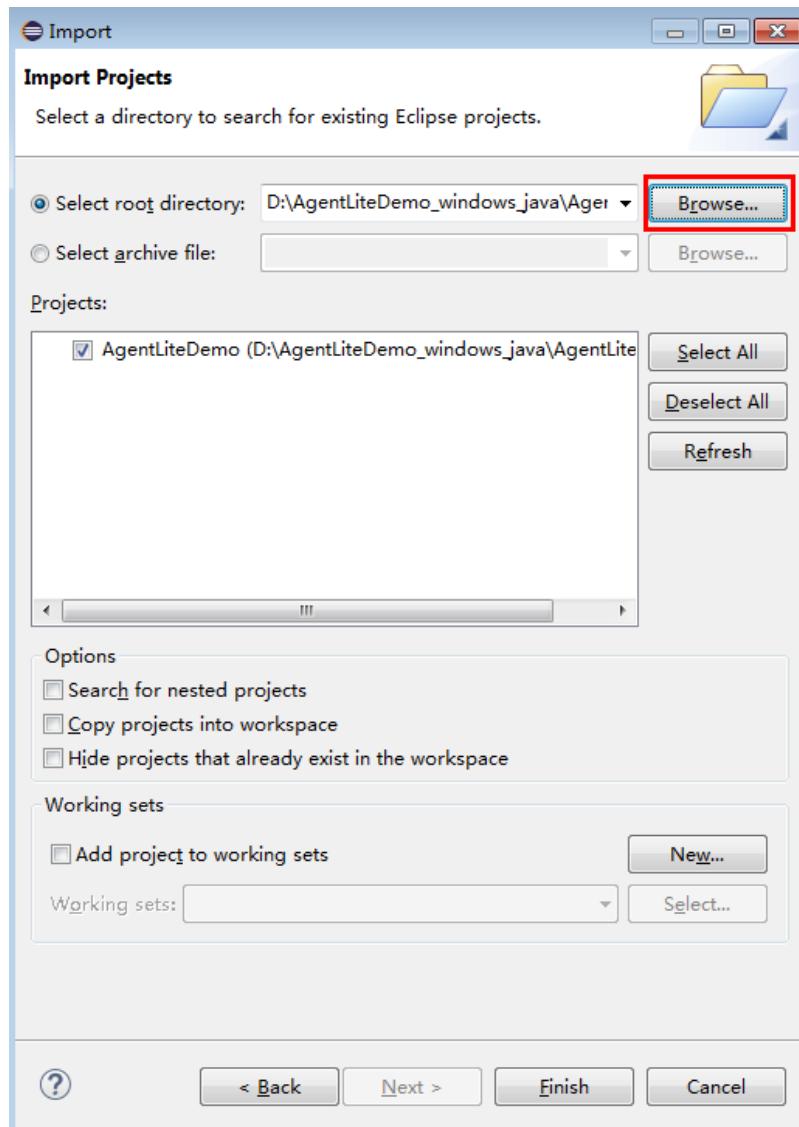


**步骤2** 导入AgentLiteDemo工程。

- 打开Eclipse，单击“File > Import”进入导入现有工程界面。
- 选择“General > Existing Projects into Workspace”，点击“Next”。



- 点击“Browse”，选择AgentLiteDemo解压后的路径，点击“Finish”。



----结束

### 6.3.4.5 初始化

在发起业务前，需要先初始化Agent Lite相关资源，调用API接口**BaseService.init()**，初始化Agent Lite资源，具体API的参数使用参考Agent Lite API接口文档。

调用**BaseService.init(String workPath, String logPath)**初始化AgentLite资源。

```
res = BaseService.init("./workdir", null);
```

### 6.3.4.6 绑定

设备或网关第一次接入物联网平台时需要进行绑定操作，从而将设备或网关与平台进行关联。开发者通过传入设备序列号以及设备信息，将设备或网关绑定到物联网平台。

绑定前先调用API接口**BindConfig.setConfig()**设置绑定服务器（即物联网平台）的IP与端口，接着调用API接口**BindService.bind()**进行绑定，主要入参为“verifyCode”（即

设备的标识) 和必要的设备信息(包括“nodeId”、“厂家Id”、“设备类型”、“设备型号”、“协议类型”，其中“nodeId”的值与“verifyCode”保持一致)。

调用 `BindConfig.setConfig()` 设置绑定参数(平台的IP和端口)。

```
//绑定配置
private static void configBindPara() {
    boolean res = false;
    res = BindConfig.setConfig(BindConfig.BIND_CONFIG_ADDR, "100.112.73.44");
    res = BindConfig.setConfig(BindConfig.BIND_CONFIG_PORT, "8943");
    BaseService.setConfig(8,18,"0");
    if (false == res) {
        System.out.println(" ===== set BindConfig failed =====");
    }
    System.out.println("startBind platformIP =" + "100.112.73.44" + ":" + "8943");
}
```

注册观察者对设备绑定结果进行相应处理。

```
//网关绑定
AgentLiteBind agentLiteBind = AgentLiteBind.getInstance();
BindService bindService = BindService.getInstance();
bindService.registerObserver(agentLiteBind);
agentLiteBind.bindAction();
```

调用 `BindService.bind(String verifyCode, IotaDeviceInfo deviceInfo)` 绑定设备。

```
public void bindAction() {
    System.out.println(" ===== start bind ===== ");
    String nodeId = "1234512345";
    String verifyCode = "1234512345";
    String manufactureId = "Huawei";
    String deviceType = "Gateway";
    String model = "AgentLite01";
    String protocolType = "HuaweiM2M";
    deviceInfo = new IotaDeviceInfo(nodeId, manufactureId, deviceType, model, protocolType);

    //绑定配置
    configBindPara();

    //发起绑定请求
    BindService.bind(verifyCode, deviceInfo);
}
```

#### 说明

- 设备或网关绑定成功，后续就不再需要再绑定了，除非设备或网关被删除，才需要重新绑定。
- `bindAction()`方法中“deviceInfo”存储设备信息，“verifyCode”(即nodeId)为设备唯一标示符。
- 设备绑定成功会收到广播，广播内容请参考[设备绑定](#)接口的返回结果说明和demo中`update`函数的处理。

### 6.3.4.7 登陆

设备或网关绑定成功后或重启后，需要进行登录的流程，在设备或网关成功登录物联网平台后，才可以进行其它服务操作，比如接入其他传感器，数据上报等等。如果设备或网关登录成功，那么设备或网关在平台的状态显示为已在线。

登录前需要通过参数配置接口 `LoginConfig.setConfig()` 传入所需的登录信息，再调用 API 接口 `LoginService.login()` 进行登录平台的流程，具体 API 的参数使用参考 Agent Lite 接口文档中的[设备登录](#)接口说明。

设置登陆参数。

```
private static void configLoginPara() {  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_DEVICEID, GatewayInfo.getDeviceID());  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_APPID, GatewayInfo.getAppID());  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_SECRET, GatewayInfo.getSecret());  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_IOCM_ADDR, "100.112.73.44");  
    //LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_IOCM_ADDR, GatewayInfo.getHaAddress());  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_IOCM_PORT, "8943");  
    //LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_MQTT_ADDR, GatewayInfo.getHaAddress());  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_MQTT_ADDR, "100.112.73.44");  
    LoginConfig.setConfig(LoginConfig.LOGIN_CONFIG_MQTT_PORT, "8883");  
}
```

### 说明

- “设备Id”（即网关Id，“LOGIN\_CONFIG\_DEVICEID”），“appId”（“LOGIN\_CONFIG\_APPID”）和“密码”（“LOGIN\_CONFIG\_SECRET”），这些信息是都是从网关绑定成功的广播中得到的。
- “平台HTTP地址”（“LOGIN\_CONFIG\_IOCM\_ADDR”）和“MQTT地址”（“LOGIN\_CONFIG\_MQTT\_ADDR”）一般是同一个地址，可以从绑定成功的广播中得到。一般情况下，这个地址和Agent Lite设备或网关对接的平台地址一致。
- 绑定成功的广播参数获取可以参考demo中**update**函数的处理。

注册广播接收器对设备登录结果进行相应处理。

```
//网关登陆  
AgentLiteLogin agentLiteLogin = AgentLiteLogin.getInstance();  
LoginService loginService = LoginService.getInstance();  
loginService.registerObserver(agentLiteLogin);  
agentLiteLogin.loginAction();
```

调用**LoginService.login()**进行直连设备登录。

```
public void loginAction() {  
    System.out.println(" ===== start login ===== ");  
    configLoginPara();  
    LoginService.login();  
}
```

### 说明

开发者可以参考附录中的[流程图](#)，以便对整体流程有更全面的理解。

## 6.3.4.8 上传 Profile 并注册设备

下载[Profile模板](#)，并上传模板中的profile文件：“Gateway\_Huawei\_AgentLite01.zip”和“Motion\_Huawei\_test01.zip”。

**步骤1** 登录SP Portal，进入“设备管理”，选择对应的应用，选择“产品模型”，点击右上角“+新增产品模型”下的“本地导入”，进入“手动导入产品”界面。

图 6-65 导入产品模型



**步骤2** 在弹出窗口中输入“产品名称”，添加“资源文件”，点击“确定”，将Profile导入到物联网平台。

图 6-66 手动导入产品



**步骤3** 选择“设备管理 > 设备 > 设备注册 > 单个注册”，点击右上角“+创建”，进入“单个设备注册”页面。

图 6-67 注册单个设备



**步骤4** 在弹出窗口中填写设备必要的信息。

图 6-68 填写设备基本信息

设备注册 > 单个设备注册

基本信息

\* 注册类型 动态密码模式  
说明：在平台配置设备的“设备标识码”（标识码要求唯一，如IMEI、MAC地址等），设备在接入平台时携带该标识码完成接入验证。

\* 设备识别码 aa123456789

\* 设备定义 自定义 产品

\* 设备类型 WaterMeter

\* 厂商名称 HZYB

\* 生产商ID TestUtf8Manuld

\* 设备型号 TestUtf8ModelM2M

\* 协议类型 LWM2M

可选信息

安全设备 请选择  
说明：安全设备由5684端口接入,非安全设备由5683端口接入

设备名称 设备名称 (数字,字母,中文, <=30个字符)

提交 取消

### 说明

- “设备名称”可以随意取。
- “注册类型”有“动态密码模式”、“初始密码模式”，默认选择“动态密码模式”。
- “设备标识”（NodeID）需要与AgentLiteDemo所运行的设备上设置的设备标识（即MAC地址）一致。
- “设备类型”、“厂商名称”、“厂商ID”、“协议类型”与profile中的定义保持一致即可。
- 其他信息根据实际情况填写，不影响业务功能。非必要信息不必填写。

### ----结束

#### 6.3.4.9 上线

设备或网关登陆成功后，就会上线，可以登录SP Portal，选择“设备管理 > 设备 > 所有设备”，查看设备状态显示为“在线”。



#### 6.3.4.10 数据上报和数据发布

##### 6.3.4.10.1 使用说明

设备或网关向物联网平台上报数据可以通过调用SDK的“设备服务数据上报”接口或“数据发布”接口：

- “设备服务数据上报”接口：topic固定为“/cloud/signaltrans/v2/categories/data”；deviceId, requestId和服务Id由SDK组装为消息的header；serviceProperties由SDK组装为消息的body。消息组装格式为JSON。

- “数据发布”接口：topic作为入参可以调整，可以为“/cloud/signaltrans/v2/categories/data”，也可以传入其他topic；“serviceData”参数作为消息体（包括header和body），SDK只进行透传，不进行格式调整和组装。

因此：当开发者希望使用SDK的固定topic（“/cloud/signaltrans/v2/categories/data”）和固定格式（JSON）上报数据时，可以调用**设备服务数据上报**接口；当开发者希望使用自定义的topic和格式上报数据时，可以调用**数据发布**接口。

### 6.3.4.10.2 设备服务数据上报

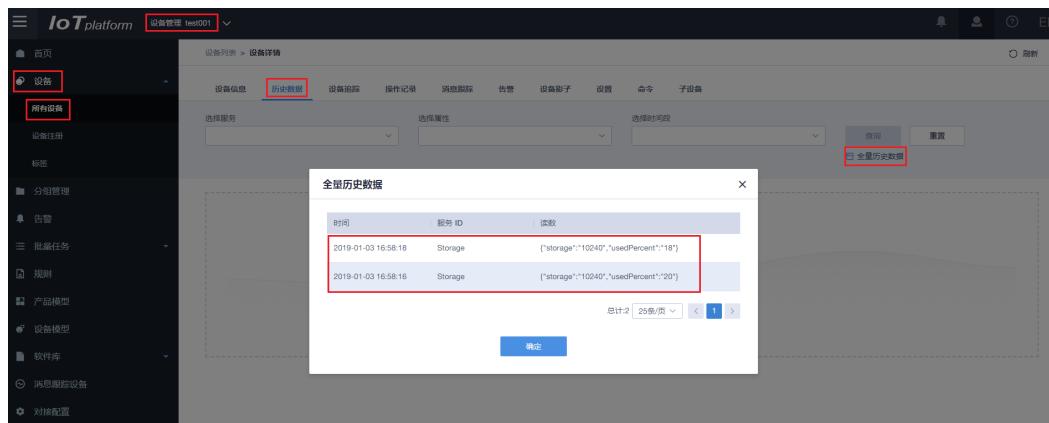
设备或网关登录成功后可以调用**DataTransService.dataReport(int cookie, String requestId, String deviceId, String serviceId, String serviceProperties)**接口上报数据。

```
public void dataReportAction(String properties) {  
    System.out.println(" ===== start WGdataReport ===== ");  
    int cookie;  
    String deviceId = GatewayInfo.getDeviceID();  
  
    Random random = new Random();  
    cookie = random.nextInt(65535);  
    System.out.println(" cookie:" + cookie + "deviceId :" + deviceId);  
    DataTransService.dataReport(cookie, null, deviceId, "Storage", properties);  
}
```

注册广播接收器对网关数据上报结果进行相应处理。

```
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();  
DataTransService dataTransService = DataTransService.getInstance();  
dataTransService.registerObserver(agentLiteDataTrans);
```

上报成功后可以在SP Portal “设备详情 > 历史数据 > 全量历史数据” 中看到上报的数据了。



### 6.3.4.10.3 数据发布

设备或网关登录成功后可以调用**DataTransService.mqttDataPub(int cookie, String topic, int qos, byte[] serviceData)**接口发布数据。

```
private void gatewayDataPub(int cookie, String topic, int qos, byte[] serviceData) {  
    DataTransService.mqttDataPub(cookie, topic, qos, serviceData);  
}
```

注册广播接收器对网关数据上报结果进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(mqttDataPubRsp, new  
IntentFilter(DataTransService.TOPIC_MQTT_PUB_RSP));
```



## 说明

- “Topic” 是要发布数据的topic。
- “Qos” 是mqtt协议的一个参数。
- “serviceData” 实际上是一个json字符串，内容是键值对（可以有多组键值对）。每个键是profile中定义的属性名（propertyName），值就是具体要上报的内容了。

### 6.3.4.11 命令接收和数据接收

#### 6.3.4.11.1 使用说明

设备或网关从物联网平台接收数据可以通过调用SDK的“设备命令接收”接口或“数据接口”接口：

- “设备命令接收”接口：只接收topic为“/gws/deviceid/signaltrans/v2/categories/”的消息，并对消息中的header进行解析。
- “数据接收”接口：接收topic订阅过的消息，且不对接收到的消息进行解析，仅进行透传。

因此：当开发者希望设备或网关只接收topic为“/gws/deviceid/signaltrans/v2/categories/”的消息，且对消息中的header进行解析时，可以调用[设备命令接收](#)接口；当开发者希望设备或网关接收topic订阅过的消息，且仅对消息进行透传时，可以调用[数据接收](#)接口。

#### 6.3.4.11.2 设备命令接收

北向应用可以调用物联网平台的应用侧API接口给网关下发命令，所以网关得随时监听命令下发的广播，以便接收到命令时进行相应业务处理。

注册广播接收器对命令下发进行相应处理。

```
//sensor数据上报
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();
DataTransService dataTransService = DataTransService.getInstance();
dataTransService.registerObserver(agentLiteDataTrans);
agentLiteDataTrans.subdevDataReport();
```

#### 6.3.4.11.3 数据接收

应用服务器可以调用物联网平台的应用侧API接口给设备或网关下发数据，所以设备或网关需要随时检测数据下发的广播，以便在接收到数据时进行相应业务处理。

在调用SDK的“数据接收”接口前，需要先调用SDK的“Topic订阅”接口向物联网平台订阅相应topic的数据。完成订阅后，物联网平台才可以将该topic的消息发送给设备或网关。订阅topic方法详见[Topic的订阅](#)。

注册DataTransService.TOPIC\_MQTT\_MSG\_RECV\_RSP广播接收器对数据下发进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(onRecvMsg, new IntentFilter(DataTransService.TOPIC_MQTT_MSG_RECV_RSP));
```

具体的处理函数onRecvMsg:

```
public static boolean onRecvMsg(UspMessage uspMsg) {
    IotaMessage iotaMsg = new IotaMessage(IodevService.IODEV_MSG_MQTT_RECV_MSG);
    iotaMsg.addString(DATATRANS_IE_TOPIC,
        uspMsg.getString(IodevService.IODEV_IE_TOPIC));
    iotaMsg.addString(DATATRANS_IE_MSGCONTENT,
```

```
uspMsg.getString(IodevService.IODEV_IE_MSG_CONTENT));
return BaseService.sendBroadCast(iotaMsg,
TOPIC_MQTT_MSG_RECV_RSP,
}
DATATRANS_BROADCAST_IE_IOTAMSG);
```

#### 说明

网关接收到的数据可能是给网关本身的，也可能是给网关下面的某个非直连设备，此接口只是作为下发数据透传的接口，具体数据的功能需要设备或网关自己解析，并做相应的处理。

### 6.3.4.12 添加非直连设备

在设备或网关登录成功后就可以调用**HubService.addDevice(int cookie, IotaDeviceInfo deviceInfo)**接口添加非直连设备。

```
public void addSensor() {
System.out.println(" ===== addSensor! ===== ");
int cookie;
Random random = new Random();
cookie = random.nextInt(65535);

String nodeId = "5432154321";
String manufatrueId = "Huawei";
String deviceType = "Motion";
String model = "test01";
String protocolType = "MQTT";
deviceInfo = new IotaDeviceInfo(nodeId, manufatrueId, deviceType, model, protocolType);
HubService.addDevice(cookie, deviceInfo);
}
```

注册广播接收器对添加设备结果进行相应处理。

```
//sensor添加
AgentLiteHub agentLiteHub = AgentLiteHub.getInstance();
HubService hubService = HubService.getInstance();
hubService.registerObserver(agentLiteHub);
agentLiteHub.addSensor();
```

#### 说明

- 这里添加的非直连设备的profile已经上传了，详见[上传Profile并注册设备](#)步骤。
- 这里非直连设备的设备固有信息是测试数据。真实情况下，网关往往需要跟具体的非直连设备交互，才能得到具体的设备固有信息。
- 添加非直连设备成功后就能从广播中得到非直连设备的“deviceId”。

非直连设备添加成功后可以在网关的子设备列表中看到一条记录。



### 6.3.4.13 非直连设备状态更新

非直连设备添加上时，一般情况下是“离线”状态。所以在非直连设备添加成功后，或者在非直连设备上报数据前，要调用**HubService.updateDeviceStatus(int cookie, String deviceId, String status, String statusDetail)**进行设备状态更新。

```
public void updateDeviceStatus() {
    System.out.println(" ====== updateDeviceStatus ======");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);
    String sensorId = GatewayInfo.getSensorId();
    System.out.println("cookie = " + cookie);
    System.out.println("sensorId = " + sensorId);

    HubService.updateDeviceStatus(cookie, sensorId, "OFFLINE", "NOT_ACTIVE");
}
```

注册广播接收器对非直连设备状态更新结果进行相应处理。

```
//sensor添加
AgentLiteHub agentLiteHub = AgentLiteHub.getInstance();
HubService hubService = HubService.getInstance();
hubService.registerObserver(agentLiteHub);
agentLiteHub.addSensor();
```

#### 6.3.4.14 非直连设备数据上报

请参考[数据上报和数据发布](#)章节，调用**DataTransService.dataReport**或**DataTransService.mqttDataPub**接口进行数据上报，各个参数使用非直连设备的相关数据即可，此处不再复述。

注册广播接收器对非直连设备数据上报结果进行相应处理。

```
//sensor数据上报
AgentLiteDataTrans agentLiteDataTrans = AgentLiteDataTrans.getInstance();
DataTransService dataTransService = DataTransService.getInstance();
dataTransService.registerObserver(agentLiteDataTrans);
agentLiteDataTrans.subdevDataReport();
```

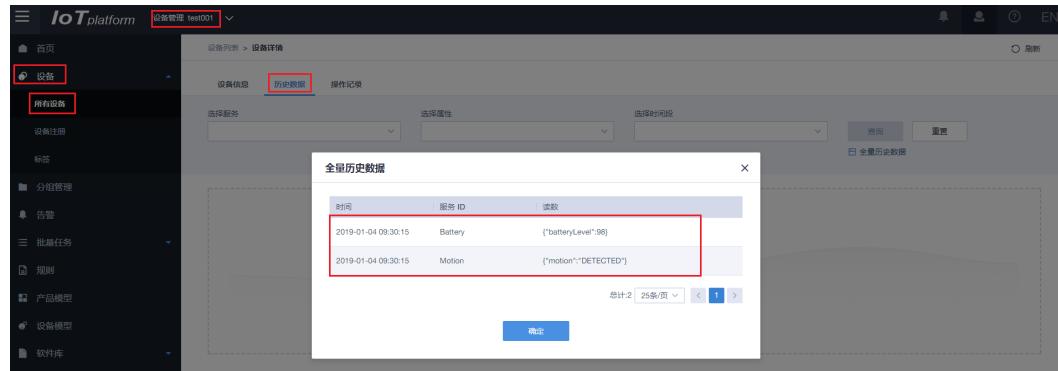
调用 **DataTransService.dataReport(int cookie, String requestId, String deviceId, String serviceId, String serviceProperties)**进行非直连设备数据上报。

```
public void subdevDataReport() {
    System.out.println(" ====== subdevDataReport! ======");
    int cookie;
    Random random = new Random();
    cookie = random.nextInt(65535);

    String deviceId = GatewayInfo.getSensorId();

    Gson iotGson = new Gson();
    JsonObject data = new JsonObject();
    data.addProperty("batteryLevel", "3");
    DataTransService.dataReport(cookie, null, deviceId, "Battery", iotGson.toJson(data));
}
```

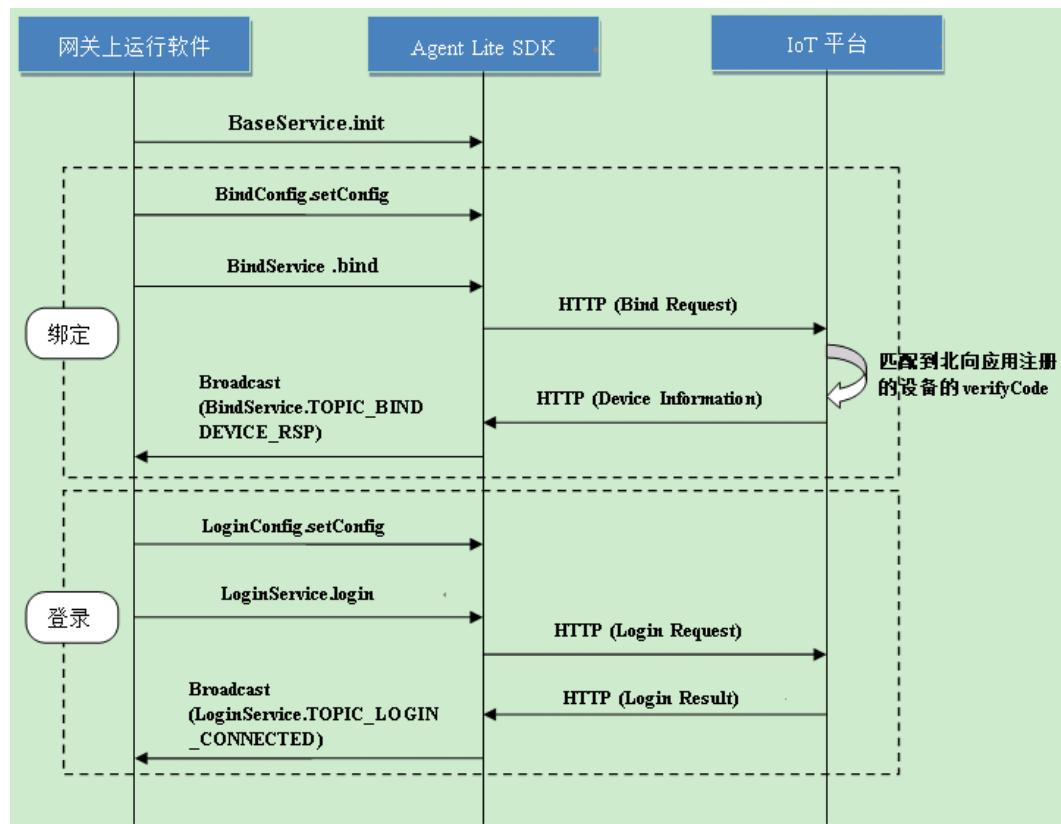
设备数据上报成功后，可以在非直连设备的“设备详情 > 历史数据 > 全量历史数据”中查看上报的数据。



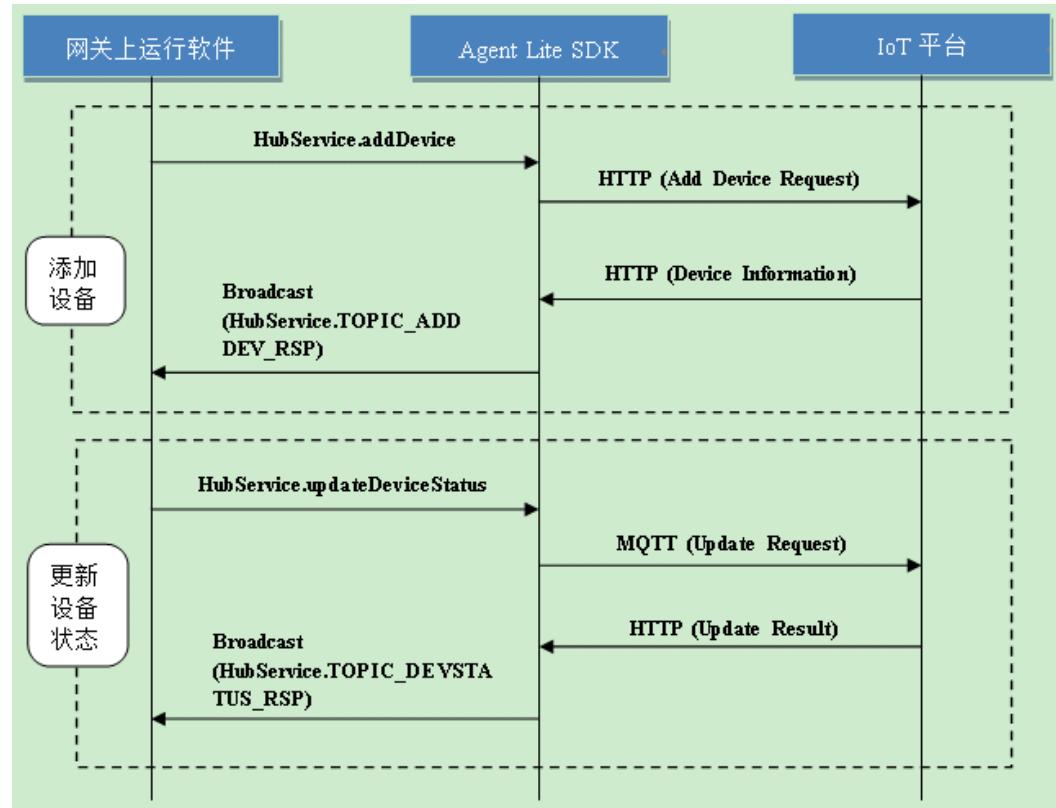
### 6.3.4.15 附录

#### 6.3.4.15.1 流程图

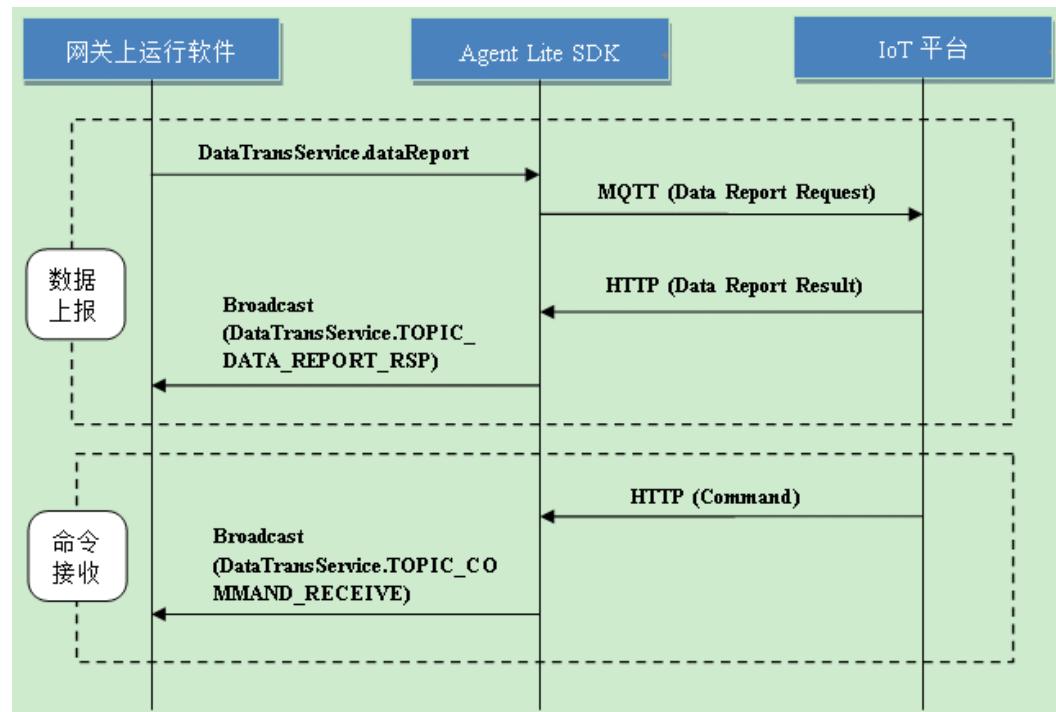
##### ?1. 绑定和登录



## ? .2. 添加非直连设备和设备状态更新



## ? .3. 数据上报和命令接收



### 6.3.4.15.2 Topic 的订阅

网关登录成功后可以调用**mqttSubTopic(int cookie, String topic, int qos)**接口订阅 Topic。

```
private void gatewayDataSubTopic(int cookie, String topic, int qos) {  
    DataTransService.mqttSubTopic(cookie, topic, qos);  
}
```

注册广播接收器对订阅结果进行相应处理。

```
LocalBroadcastManager.getInstance(this).registerReceiver(mqttSubTopicRsp, new  
IntentFilter(DataTransService.TOPIC_MQTT_SUB_RSP));
```

#### 说明

- “Topic” 是要发布数据的topic。
- “Qos” 是mqtt协议的一个参数。

# 7 应用侧 SDK 使用指南

## 7.1 JAVA SDK 使用指南

### 7.1.1 开发者必读

- 本文以提供的北向Java SDK Demo为例说明如何使用JAVA SDK与IoT平台对接，包括证书配置及回调等。
- Demo以Java工程为例，每个类（除工具类外）都包含了main方法，可单独运行，旨在演示如何调用SDK接口。

### 7.1.2 开发环境要求

开发环境要求

开发平台	开发环境	配套要求	推荐的操作系统
IoT	1) J2EE for Java Developers 2) <b>Maven</b> 插件：m2e - Maven Integration for <b>Eclipse</b> (includes Incubating components)	JDK 1.8 及以上版本	Windows7

SDK包为纯JAVA的JAR包，在使用上没有特殊限制，JDK在1.8及以上版本即可。

### 7.1.3 下载相关开发资源

下载北向JAVA SDK Demo及北向JAVA SDK:

- [北向JAVA SDK Demo](#)
- [北向JAVA SDK](#)
- [北向JAVA SDK API参考](#)

SDK放在lib目录下，SDK依赖的jar包放在\testSDK\api-client-test\_lib下，开发者也可从maven仓库下载。

**图 7-1 SDK 依赖的 jar 包**

commons-beanutils-1.8.0.jar	2018/7/20 17:17	Executable Jar File	226 KB
commons-collections-3.2.1.jar	2018/7/20 17:17	Executable Jar File	562 KB
commons-lang-2.5.jar	2018/7/20 17:17	Executable Jar File	273 KB
ezmorph-1.0.6.jar	2018/7/20 17:17	Executable Jar File	85 KB
httpclient-4.5.2.jar	2018/7/20 17:17	Executable Jar File	720 KB
httpcore-4.4.4.jar	2018/7/20 17:17	Executable Jar File	320 KB
jackson-annotations-2.5.4.jar	2018/7/20 17:17	Executable Jar File	39 KB
jackson-core-2.5.4.jar	2018/7/20 17:17	Executable Jar File	225 KB
jackson-databind-2.5.4.jar	2018/7/20 17:17	Executable Jar File	1,118 KB
jcl-over-slf4j-1.7.25.jar	2018/7/20 17:17	Executable Jar File	17 KB
json-lib-2.4.jar	2018/7/20 17:17	Executable Jar File	156 KB
logback-classic-1.1.11.jar	2018/7/20 17:17	Executable Jar File	302 KB
logback-core-1.1.11.jar	2018/7/20 17:17	Executable Jar File	465 KB
slf4j-api-1.7.22.jar	2018/7/20 17:17	Executable Jar File	41 KB

Demo工程依赖的jar包放在components文件夹下，开发者也可从maven仓库下载。

**图 7-2 Demo 工程依赖的 jar 包**

httpmime-4.5.2.jar
json-lib-2.4.jar
netty-all-4.0.27.Final.jar
spring-boot-starter-web-1.5.9.RELEASE.jar

#### 说明

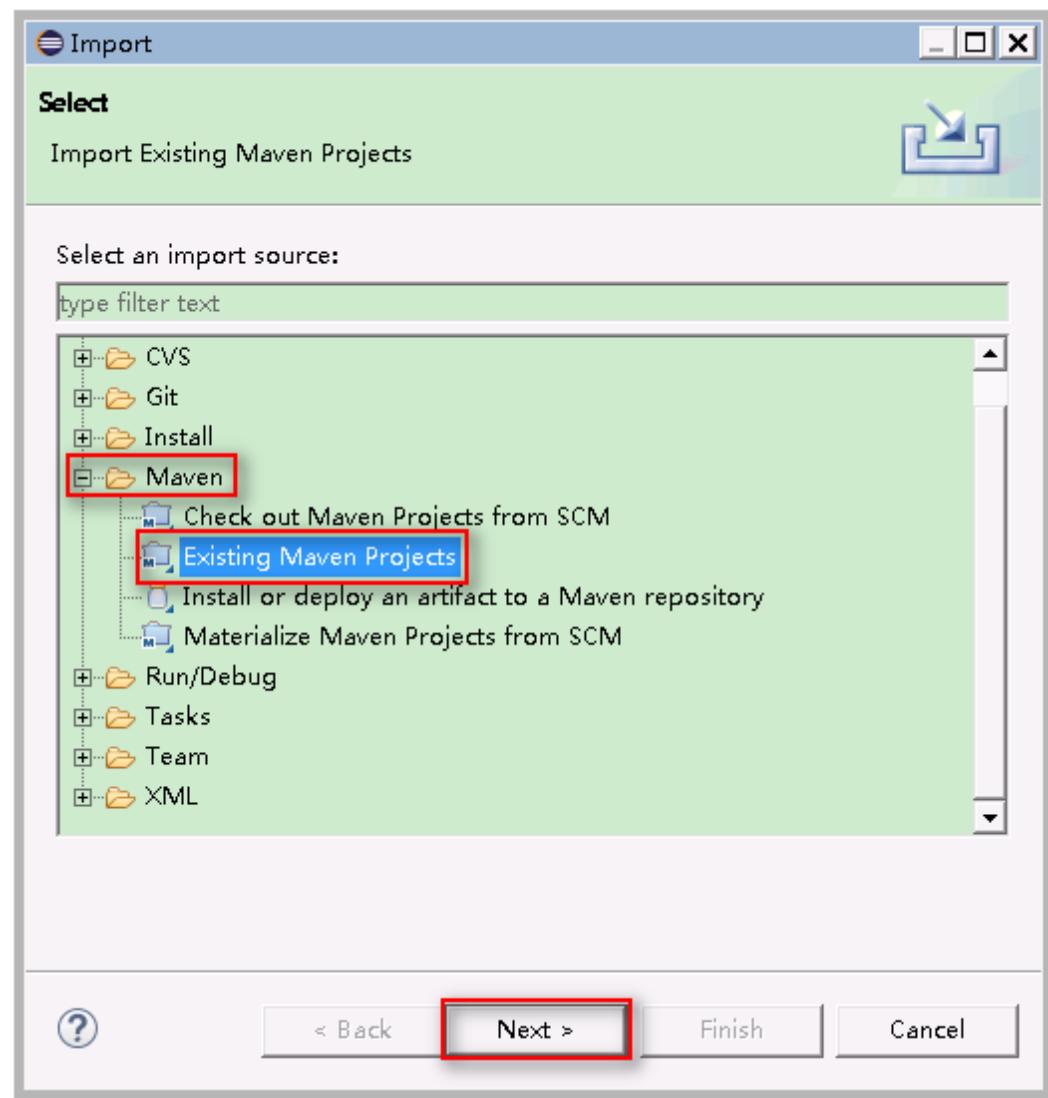
Demo工程中的lib下已包含SDK库。

### 7.1.4 导入 Demo 工程

**步骤1** 将下载的North\_JAVA\_SDK\_Demo.zip解压到本地。

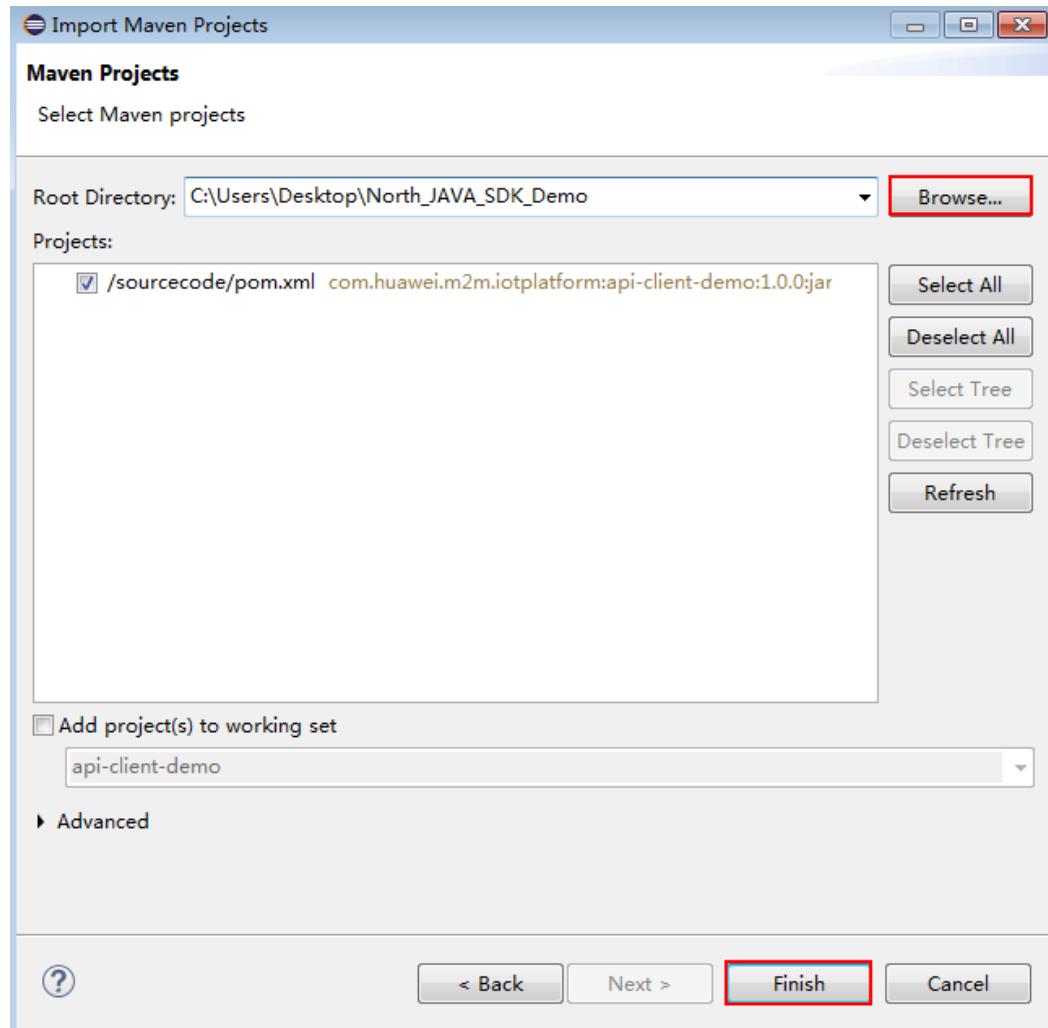
**步骤2** 打开eclipse，选择菜单“File”->“Import”，再选择“Maven”->“Existing Maven Projects”，点击“Next”。

图 7-3 导入 Demo 工程 1



步骤3 点击“Browse”，选择demo解压后的路径，点击Finish。

图 7-4 导入 Demo 工程 2



----结束

### 7.1.5 初始化及证书配置

新建一个**NorthApiClient**实例，设置好**ClientInfo**（包括平台IP、端口、appId和密码），再初始化证书：

```
NorthApiClient northApiClient = new NorthApiClient();
PropertyUtil.init("./src/main/resources/application.properties");
ClientInfo clientInfo = new ClientInfo();
clientInfo.setPlatformIp(PropertyUtil.getProperty("platformIp"));
clientInfo.setPlatformPort(PropertyUtil.getProperty("platformPort"));
clientInfo.setAppId(PropertyUtil.getProperty("appId"));
clientInfo.setSecret(PropertyUtil.getProperty("secret"));
northApiClient.setClientInfo(clientInfo);
northApiClient.initSSLConfig(); //默认使用测试证书，且不进行主机名校验
```

**注意**

- 平台IP、端口、appId和密码都是从配置文件./src/main/resources/application.properties中读取的，因此，当这些信息发生变化时，只要修改配置文件，不用修改应用服务器的代码。

如果不使用测试证书，可使用指定证书（如商用证书）：

```
NorthApiClient northApiClient = new NorthApiClient();

PropertyUtil.init("./src/main/resources/application.properties");

ClientInfo clientInfo = new ClientInfo();
clientInfo.setPlatformIp(PropertyUtil.getProperty("platformIp"));
clientInfo.setPlatformPort(PropertyUtil.getProperty("platformPort"));
clientInfo.setAppId(PropertyUtil.getProperty("appId"));
clientInfo.setSecret(getAesPropertyValue("secret"));

SSLConfig sslConfig= new SSLConfig();
sslConfig.setTrustCaPath(PropertyUtil.getProperty("newCaFile"));
sslConfig.setTrustCaPwd(getAesPropertyValue("newCaPassword"));
sslConfig.setSelfCertPath(PropertyUtil.getProperty("newClientCertFile"));
sslConfig.setSelfCertPwd(getAesPropertyValue("newClientCertPassword"));

northApiClient.setClientInfo(clientInfo);
northApiClient.initSSLConfig(sslconfig); //使用指定的证书，且默认使用严格主机名校验
```

使用指定证书时，如果不使用严格主机名校验，在调用**northApiClient.initSSLConfig(sslconfig)**之前可以自行设置主机名校验方法：

```
northApiClient.setHostnameVerifier(new HostnameVerifier() {
    public boolean verify(String arg0, SSLSession arg1) {
        // 自定义主机名校验
        .....
        return true;
    }
});
```

**注意**

- 主机名校验方法应以安全为原则，不应该直接返回true。
- 本章节所指的证书是平台提供的，在调用平台接口过程中使用；一般情况下，与回调使用的证书不一样。

## 7.1.6 业务接口调用方法

在[初始化及证书配置](#)步骤中设置好**NorthApiClient**实例后才能调用其他业务接口。以下几个接口为例说明如何调用业务接口：

### 鉴权

```
//得到NorthApiClient实例后，再使用northApiClient得到鉴权类实例
Authentication authentication = new Authentication(northApiClient);

//调用鉴权类实例authentication提供的业务接口，如getAuthToken
AuthOutDTO authOutDTO = authentication.getAuthToken();

//从返回的结构体authOutDTO中获取需要的参数，如accessToken
String accessToken = authOutDTO.getAccessToken();
```

### 订阅

```
//得到NorthApiClient实例后，再使用northApiClient得到订阅类实例
SubscriptionManagement subscriptionManagement = new SubscriptionManagement(northApiClient);

//先设置好subDeviceData的第一个入参SubDeviceDataInDTO结构体
SubDeviceDataInDTO sddInDTO = new SubDeviceDataInDTO();
ddInDTO.setNotifyType("deviceDataChanged");
//需要根据实际情况修改回调的ip和端口
ddInDTO.setCallbackUrl("https://XXX.XXX.XXX.XXX:8099/v1.0.0/messageReceiver");
try {
    //调用订阅类实例subscriptionManagement提供的业务接口，如subDeviceData
    SubscriptionDTO subDTO = subscriptionManagement.subDeviceData(sddInDTO, null, accessToken);
    System.out.println(subDTO.toString());
} catch (NorthApiException e) {
    System.out.println(e.toString());
}
```

## 注册设备

```
//得到NorthApiClient实例后，再使用northApiClient得到设备管理类实例
DeviceManagement deviceManagement = new DeviceManagement(northApiClient);

//设置好注册设备接口的第一个入参RegDirectDeviceInDTO2结构体
RegDirectDeviceInDTO2 rddInDTO = new RegDirectDeviceInDTO2();
String nodeid = "86370303XXXXXX"; //this is a test imei
String verifyCode = nodeid;
rddInDTO.setNodeId(nodeid);
rddInDTO.setVerifyCode(verifyCode);
rddInDTO.setTimeout(timeout);

//调用设备管理类实例deviceManagement提供的业务接口，如regDirectDevice
RegDirectDeviceOutDTO rdddod = deviceManagement.regDirectDevice(rddInDTO, null, accessToken);

//从返回的结构体rdddod中获取需要的参数，如deviceId
String deviceId = rdddod.getDeviceId();
```

### 说明

关于哪些参数需要设置，请查看《[JAVA SDK API参考文档](#)》对于可选参数，如果业务不需要，可以不设置或者设置为null。

## 7.1.7 回调接口实现

新建一个类并继承**PushMessageReceiver**，可以参考Demo中的**PushMessageReceiverTest**类，需要接收哪一类消息就重写对应的方法，如：

```
@Override
public void handleDeviceAdded(NotifyDeviceAddedDTO body) {
    System.out.println("deviceAdded ==> " + body);
    //TODO deal with deviceAdded notification
}
```

### 说明

- 接收到平台推送的消息后，开发者需要根据业务进行处理，但不建议进行复杂计算、I/O操作或者可能长时间等待的动作，可以先写数据库，应用进入相应界面或者刷新界面再从数据库取数据并进行数据处理。
- 回调路径已在SDK中设置好了，所以在订阅时要注意订阅对应的回调地址，具体可参考[《JAVA SDK API参考文档》](#)文档中消息推送章节的接口。
- 回调的IP地址则是服务器的地址，需要是公网地址。
- Demo工程的回调端口配置在src\main\resource\application.properties中：  
`#specify the port of the web application  
server.port=8099`

## 7.1.8 回调证书制作、配置及上传

本章节以自签名证书为例。如果是使用商用证书，请直接向CA机构申请。

**步骤1** 打开windows命令行窗口，输入where java，找到jdk所在路径，进入jdk的bin路径。

```
where java  
cd /d {jdk的bin路径}
```

图 7-5 查找并进入 jdk 的 bin 路径

C:\Users\...> where java  
C:\ProgramData\Oracle\Java\javapath\java.exe  
C:\Program Files\Java\jdk1.8.0\_45\bin\java.exe  
  
C:\Users\...> cd /d C:\Program Files\Java\jdk1.8.0\_45\bin  
  
C:\Program Files\Java\jdk1.8.0\_45\bin>

**步骤2** 使用如下命令生成tomcat.keystore文件。

```
keytool -genkey -v -alias tomcat -keyalg RSA -keystore tomcat.keystore -validity 36500
```

图 7-6 生成 tomcat.keystore 文件

C:\Program Files\Java\jdk1.8.0\_45\bin>keytool -genkey -v -alias tomcat -keyalg RSA -keystore tomcat.keystore  
输入密钥库口令:  
再输入新口令:  
您的名字与姓氏是什么? **输入应用服务器的IP或域名**  
[Unknown]: 192.168.1  
您的组织单位名称是什么?  
[Unknown]: O=...ct  
您的组织名称是什么?  
[Unknown]: O=...t  
您所在的城市或区域名称是什么?  
[Unknown]: sz  
您所在的省/市/自治区名称是什么?  
[Unknown]: gd  
该单位的双字母国家/地区代码是什么?  
[Unknown]: CN  
CN=**输入CN是否正确?**  
[否]: y  
正在为以下对象生成 2,048 位RSA密钥对和自签名证书 <SHA256withRSA> <有效期为 36,500 天>:  
CN=**输入 <tomcat> 的密钥口令  
<如果和密钥库口令相同, 按回车>:**

### 注意

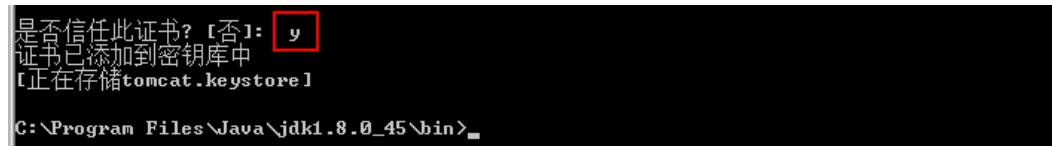
- 如果jdk的bin目录下已有tomcat.keystore，建议先将已有的tomcat.keystore移到别的路径下。
- “您的名字与姓氏是什么”要输入应用服务器的IP或域名。
- <tomcat>的密钥口令要与密钥库口令设置一致（最后一步按回车即可），输入的密钥库口令要记住，后续配置会使用到。

**步骤3** 将IoT平台提供的根证书ca.pem放到jdk的bin目录下，并使用如下命令将其加到tomcat.keystore的信任证书链中。

```
keytool -import -v -file ca.pem -alias iotplatform_ca -keystore tomcat.keystore
```

C:\Program Files\Java\jdk1.8.0\_45\bin>keytool -import -v -file ca.pem -alias iotplatform\_c  
输入密钥库口令:  
所有者: CN=**输入所有者CN**  
发布者: CN=**输入发布者CN**

输入密钥库口令，查看导入的证书内容，确认无误后，输入y即可。



#### 说明

- 平台的测试根证书ca.pem可以在JAVA SDK包的cert目录下找到。
- 将IoT平台提供的根证书ca.pem加到tomcat.keystore的信任证书链后，由ca.pem签发的子证书就能得到应用服务器的信任。

**步骤4** 将tomcat.keystore放到Demo工程目录下，例如：src\main\resources，打开src\main\resource\application.properties，添加如下配置。其中，server.ssl.key-store为tomcat.keystore所在路径，server.ssl.key-store-password为密钥库口令。

```
#one-way authentication (server-auth)
server.ssl.key-store= ./src/main/resources/tomcat.keystore
server.ssl.key-store-password=741852963.
```

**步骤5** 右键单击**PushMessageReceiverTest**，选择Run As->Java Application，运行Demo中的**PushMessageReceiverTest**类。运行结果如下：

#### 说明

当有数据推送到应用服务器时，就会进入相应的回调函数中。

**图 7-7 PushMessageReceiverTest 类运行结果**

The screenshot shows an IDE interface with the following details:  
1. \*\*Code Editor:\*\* Shows Java code for the PushMessageReceiverTest class, specifically overriding handleDeviceAdded, handleBindDevice, and handleDeviceInfoChanged methods to print the received body to System.out.  
2. \*\*Console:\*\* Shows the execution output of the application. It includes the Spring Boot logo and the message "deviceAdded ==> NotifyDeviceAddedDTO [notifyType=deviceAdded, deviceId=...]" which is highlighted with a red box.

```
37 //override the callback functions if needed, otherwise, you can delete them.
38
39 @Override
40 public void handleDeviceAdded(NotifyDeviceAddedDTO body) {
41     System.out.println("deviceAdded ==> " + body);
42 }
43
44 @Override
45 public void handleBindDevice(NotifyBindDeviceDTO body) {
46     System.out.println("bindDevice ==> " + body);
47 }
48
49 @Override
50 public void handleDeviceInfoChanged(NotifyDeviceInfoChangedDTO body) {
```

**步骤6** 使用浏览器打开回调地址https://server:8099/v1.0.0/messageReceiver，并查看证书。

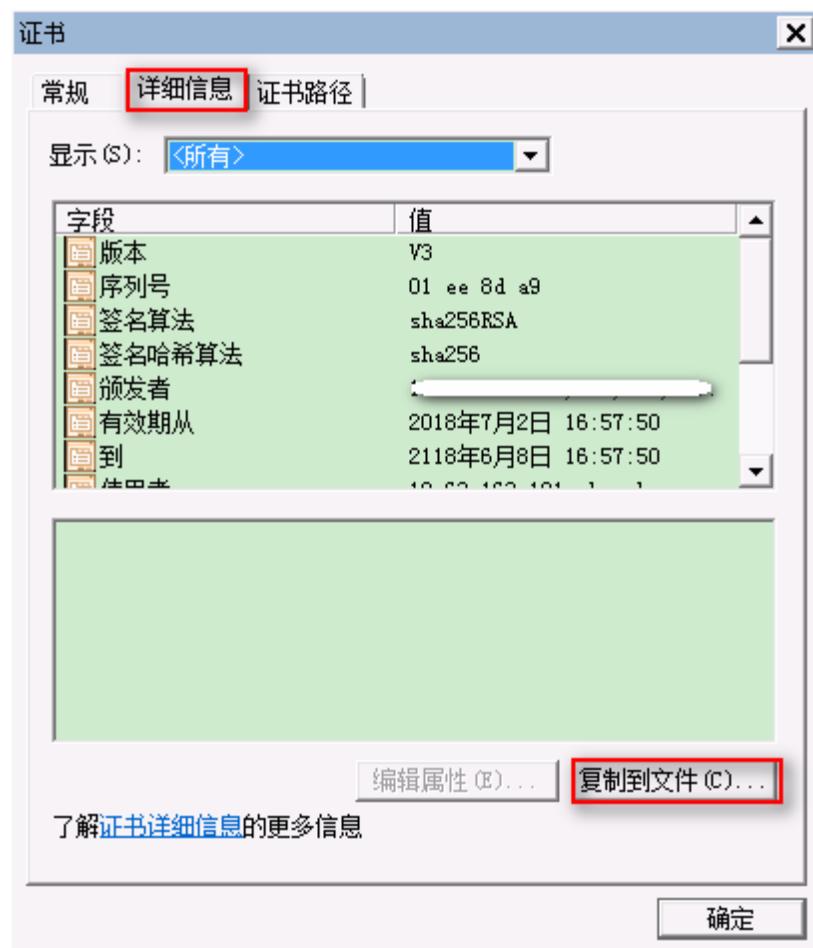
server是应用服务器的地址（即本机地址），8099是在application.properties中配置的端口。

图 7-8 查看证书



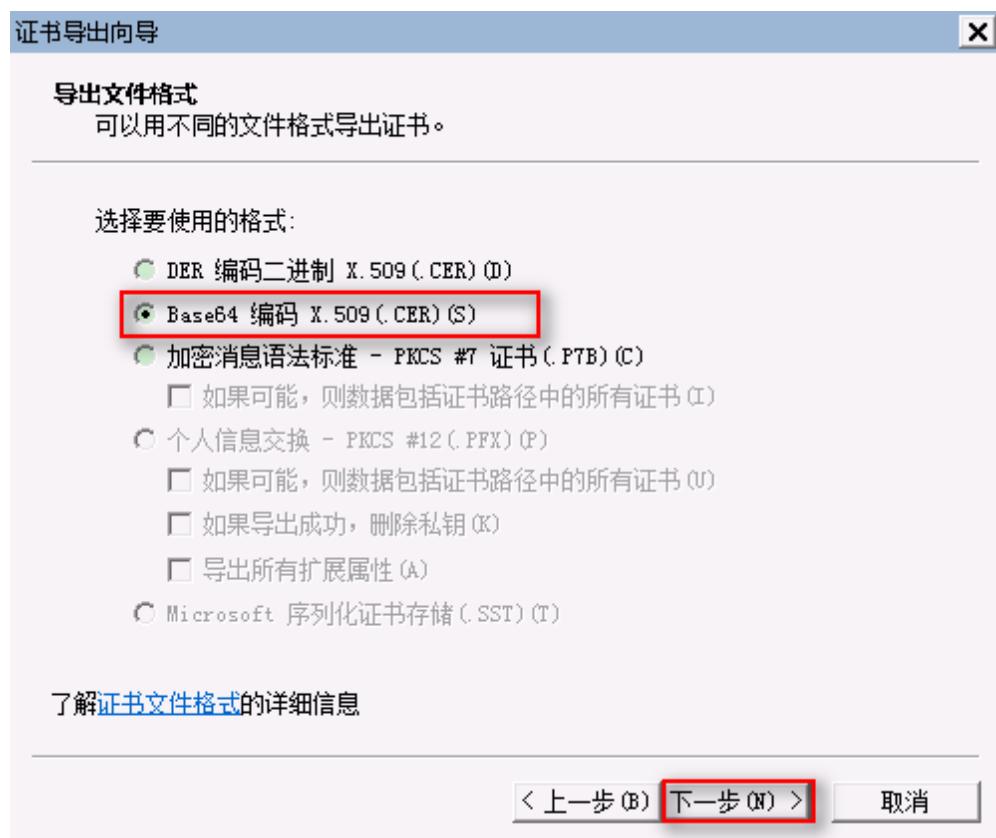
步骤7 切换到详细信息，点击“复制到文件”。

图 7-9 详细信息



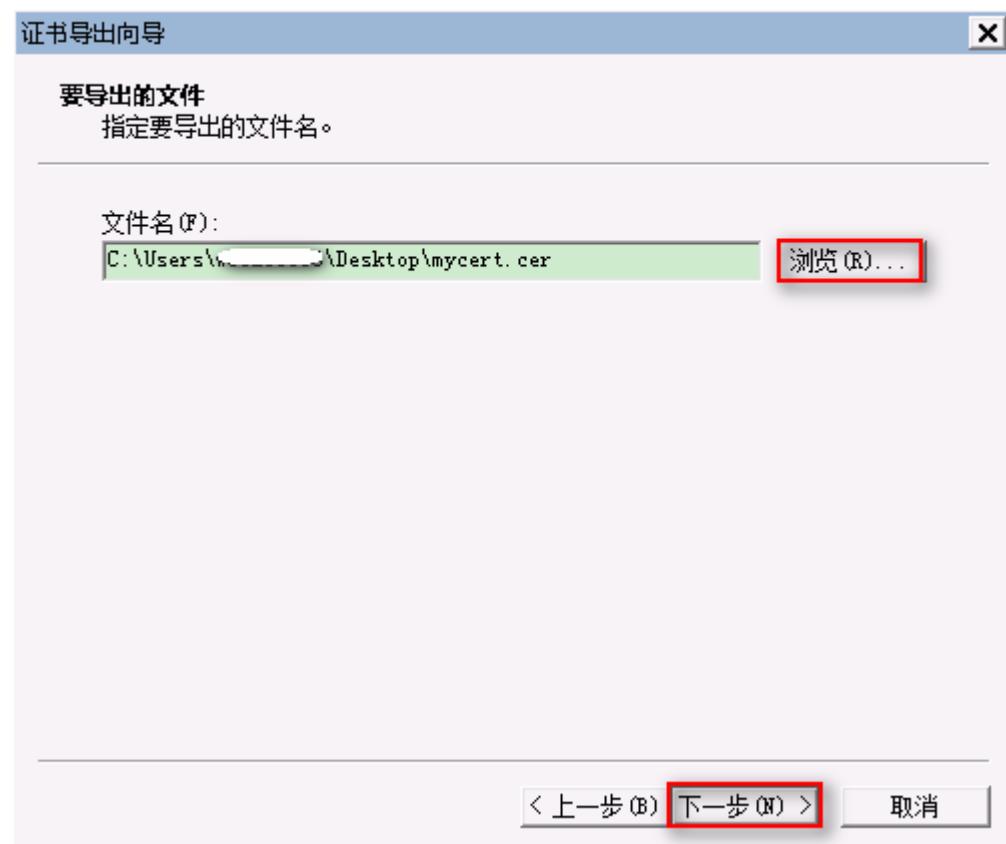
步骤8 点击“下一步”，选择Base64编码，再点击“下一步”。

图 7-10 选择编码格式



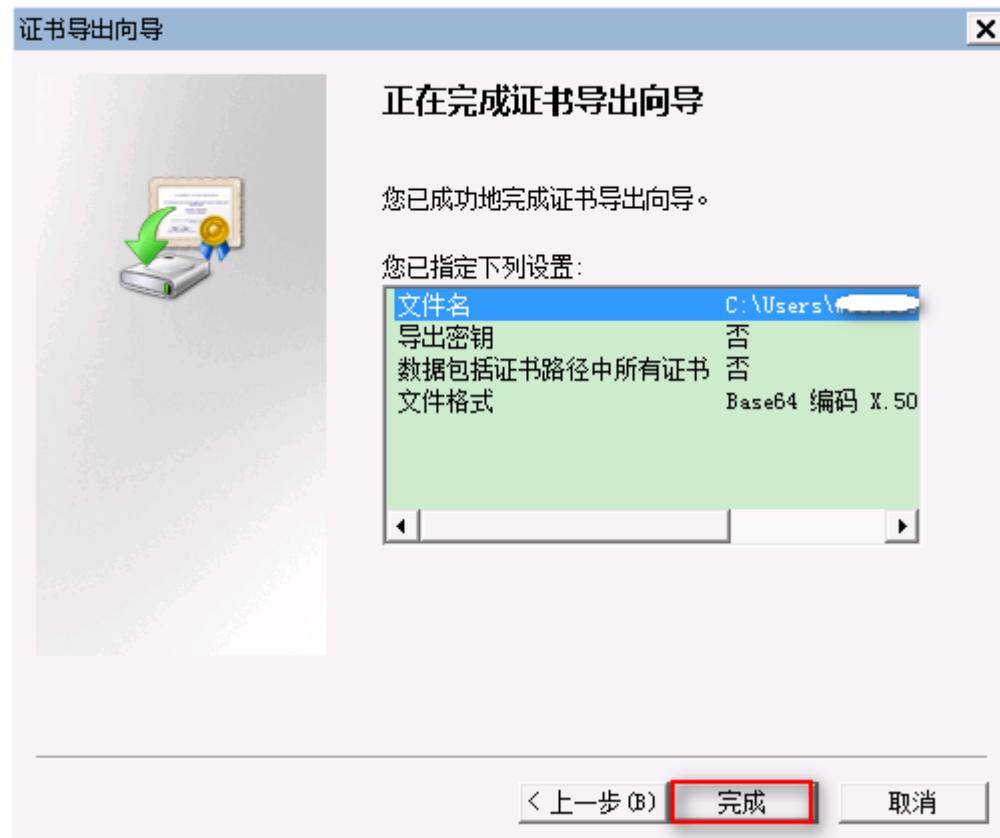
**步骤9** 选择“浏览”，选择一个路径，输入文件名，点击“保存”，回到证书导出向导，点击“下一步”。

图 7-11 指定导入文件名



点击“完成”。

图 7-12 完成证书导出

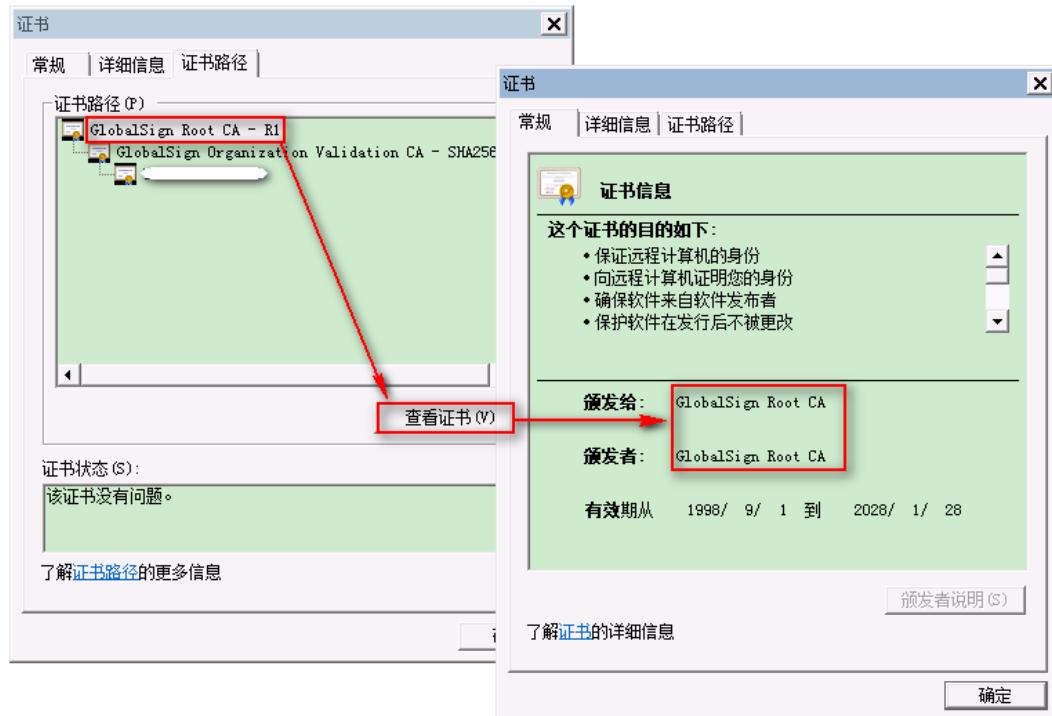


**注意**

如果应用服务器最后部署到云上，可能会有多级证书，建议在部署完成后  
再导出证书。此时需要将证书链上面几级的证书一一导出。

**步骤10** 打开“证书路径”页签，可以看到有多级证书，点击证书路径中的某个证书，点击  
“查看证书”，然后重复上面的步骤导出此级证书。

图 7-13 逐级导出证书



**步骤11** 将所有证书导出后，使用文本编译器打开证书，将证书内容复制到一个文件中，头尾相连，并保存为.pem格式的文件。该文件需要上传到IoT平台相应的应用下。

图 7-14 合并证书

```
24 5gaRQBi5+MHt39tBquCWIMnN2BU4gcmU7qKEKQsTb47bDN01Atukix1E0kF6BWlK
25 WE9gyn6CagsCqiUXObXbf+eEZSqVir2G316BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
26 4fQRbxCl1fznQgUy286dUV4otp6F01vvpx1FQHK0tw5rDgb7MzViCbidJ4vEZV8N
27 hnacRHr21Vz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
28 -----END CERTIFICATE-----
29 -----BEGIN CERTIFICATE-----
30 MIIFODCCBCCgAwIBAgIQUT+5dDhwztRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
31 yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBjbMuMR8wHQYDVQQL
32 ExZWZXJpU2lnbiBUcnVzdCBOZXRB3b3JrMTowOAYDVQQLEzEoYykGMjAwNiBWZXJp
33 U2lnbiwgSW5jLiAtIEZvcibhdXRob3JpemVkJHVzzSBvbmx5MUUwQwYDVQQDEzxW
34 ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcm1tYXJ5IEhlcnRpZmljYXRpb24gQXV0
```

### 注意

- Demo中的配置为单向认证，在导出证书后需要修改为双向认证，将以下配置打开（去掉注解，修改tomcat.keystore目录与密码），由于已将平台的根证书加入tomcat.keystore的信任证书链中，所以不需要再做其他修改，重启一下服务器即可。

```
#two-way authentication (add client-auth)
server.ssl.trust-store=../src/main/resources/tomcat.keystore
server.ssl.trust-store-password=741852963.
server.ssl.client-auth=need
```

- 单向认证较之双向认证安全度低，请使用双向认证。

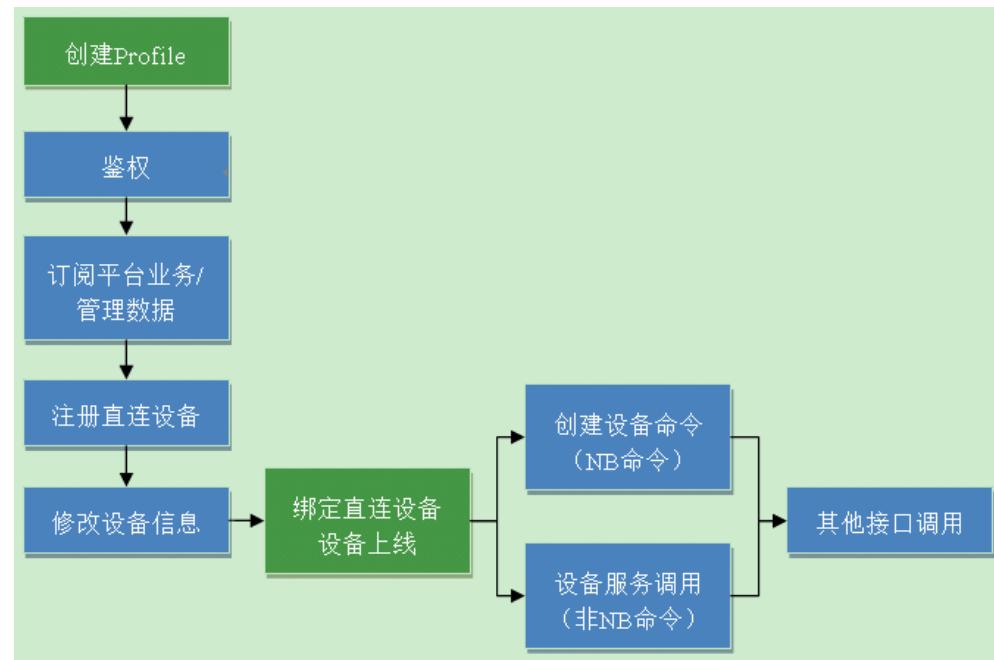
### ---结束

## 7.1.9 业务接口调用流程及注意事项

业务接口调用方法一节已说明如何调用接口，其他的接口调用方法与之类似，请直接参考Demo代码，不再逐一说明。

- 请按如下流程调用业务接口。

图 7-15 调用业务接口流程



- Demo中使用的Profile如下图所示，只有一个Brightness服务，Brightness服务下有一个brightness属性和一个PUT命令。在调用创建设备命令或设备服务调用等接口时，如果不是使用以下Profile内容，请将相关服务、属性或者命令名称修改为相应的名称。

图 7-16 Profile 文件



- 创建新的Profile方法：

登录开发者Portal->Profile开发->Profile在线开发->自定义产品->创建全新产品->设置设备类型、厂家名、厂家ID、设备型号等->点击确定->添加服务（根据设备功能添加属性和命令）->保存。

#### 说明

建议在定义好Profile后再调用接口注册设备。

- 修改设备信息接口使用到的字段值如设备类型、厂家名、厂家ID、设备型号要与Profile定义的保持一致。
- accessToken可以由SDK管理，也可由第三方应用自行管理，具体参考具体可参考[《JAVA SDK API参考文档》](#)中“应用安全接入->定时刷新token”章节的说明。

### 7.1.10 SDK 独立运行测试

SDK包中提供了可独立运行的jar包，用于测试平台北向接口。可独立运行测试的jar包在testSDK目录下：

图 7-17 可独立运行的 jar 包



- 修改config.properties后再双击运行runMe.bat即可进行测试

图 7-18 修改 config.properties

```
1 #please modify the value of platformIp/platformPort,
2 platformIp=10.10.10.10
3 platformPort=8743
4 appId=x01D12.....MMWE5a8D1a
5 secret=gPnTWc....kkf12P8f4a
6 #the value of newCaFile and newClientCertFile should
7 newCaFile=
8 newCaPassword=
9 newClientCertFile=
10 newClientCertPassword=
11 #hostNameVerify default value is true, true means se
12 hostNameVerify=
```

- 如果使用商用证书，请直接将证书放在testSDK目录下面（证书名字不可以与ca.jks或者outgoing.CertwithKey.pkcs12相同），并在config.properties中配置证书名及密码；如果使用测试证书，则不需要修改config.properties中的证书信息。

- 测试结果会在最前面输出：[y]表示测试通过；[x]则表示出错，请仔细查看该行的错误提示或说明。
- 运行jar包需要依赖JDK，请确认已安装JDK并设置了系统环境变量。  
运行runMe.bat的结果如下：

图 7-19 runMe.bat 运行结果

```
Begin test...
=> D:\0e... -> 0e... -> SDK
[y] getToken() succeeded, get accessToken successfully with inner certificates
AuthOutDTO [accessToken=57a37baa92chbfca3d763ffe8e9e9d73, tokenType=bearer, refreshToken=4ba44a
[y] refreshAuthToken() succeeded, AuthRefreshOutDTO [accessToken=309c198bc768ca31eb83996936fba2
3762bcf2]
[y] regDirectDevice() succeeded, DirectDeviceRegOutDTO [deviceId=ef72daa2-772d-4a9b-a2bc-71a48a
[y] modifyDeviceInfo() succeeded
[y] deleteDevice() succeeded
```

## 7.2 PHP SDK 使用指南

### 7.2.1 开发者必读

- 本文以提供的北向PHP SDK Demo为例说明如何使用PHP SDK与IoT平台对接，包括证书配置及回调等。
- Demo以PHP脚本为例，每个脚本文件都可单独运行，旨在演示如何调用SDK接口。

### 7.2.2 开发环境要求

开发环境要求

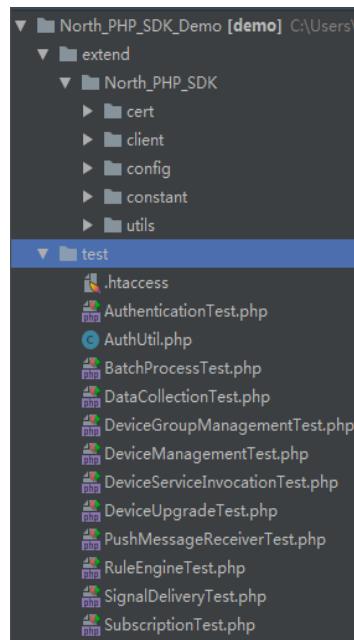
开发平台	开发环境	推荐的操作系统
IoT	1. XAMPP 7.2.9 2. PhpStorm 2018.2.3	Windows7

SDK包为纯PHP的.zip格式的压缩包。

### 7.2.3 下载相关开发资源

下载北向PHP SDK Demo及北向PHP SDK:

- [北向PHP SDK Demo](#)
- [北向PHP SDK](#)
- [北向PHP SDK API参考](#)

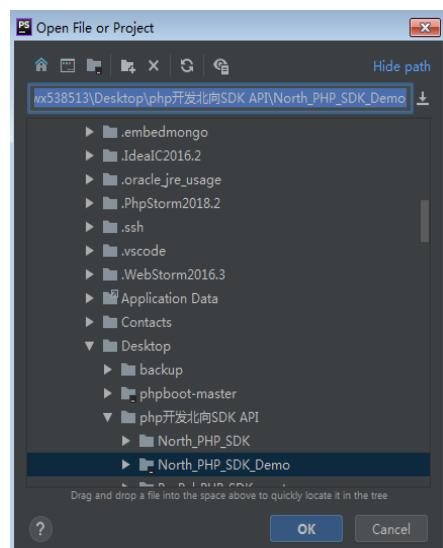
**图 7-20 SDK Demo 目录结构****说明**

Demo工程中的extend下已包含SDK库。

## 7.2.4 导入 Demo 工程

**步骤1** 将下载的“North\_PHP\_SDK\_Demo.zip”解压到本地。

**步骤2** 打开PhpStorm，选择菜单“File > Open”，再选择demo解压后的路径，点击“OK”。

**图 7-21 导入 Demo 工程**

----结束

## 7.2.5 初始化及证书配置

新建一个**NorthApiClient**实例，设置好**ClientInfo**（包括平台IP、端口、appId和密码），再初始化证书。

```
$northApiClient = new NorthApiClient();  
  
PropertyUtil::init()  
  
$clientInfo = new ClientInfo();  
$clientInfo = new ClientInfo();  
$clientInfo->platformIp = PropertyUtil::getProperty("platformIp");  
$clientInfo->platformPort = PropertyUtil::getProperty("platformPort");  
$clientInfo->appId = PropertyUtil::getProperty("appId");  
$clientInfo->secret = PropertyUtil::getProperty("secret");  
  
$northApiClient->clientInfo = $clientInfo;  
$northApiClient->initSSLConfig(); //默认使用测试证书，且不进行主机名校验
```

### 注意

平台IP、端口、appId和密码都是从配置文件“./config/application.ini”中读取的，因此，当这些信息发生变化时，只要修改配置文件，不用修改应用服务器的代码。

如果不使用测试证书，可使用指定证书（如商用证书）。

```
self::$northApiClient = new NorthApiClient();  
  
PropertyUtil::init()  
  
$clientInfo = new ClientInfo();  
$clientInfo = new ClientInfo();  
$clientInfo->platformIp = PropertyUtil::getProperty("platformIp");  
$clientInfo->platformPort = PropertyUtil::getProperty("platformPort");  
$clientInfo->appId = PropertyUtil::getProperty("appId");  
$clientInfo->secret = PropertyUtil::getProperty("secret");  
  
$sslConfig= new SSLConfig();  
$sslConfig->trustCaPath = PropertyUtil.getProperty("newCaFile");  
$sslConfig->selfCertPath = PropertyUtil.getProperty("newClientCertFile");  
  
self::$northApiClient->clientInfo = $clientInfo;  
$northApiClient->initSSLConfig($sslConfig); //使用指定的证书，且默认使用严格主机名校验
```

### 注意

- 主机名校验方法应以安全为原则，不应该直接返回true。
- 本章节所指的证书是平台提供的，在调用平台接口过程中使用；一般情况下，与回调使用的证书不一样。

## 7.2.6 业务接口调用方法

在**初始化及证书配置**步骤中设置好**NorthApiClient**实例后才能调用其他业务接口。以下几个接口为例说明如何调用业务接口：

### 配置日志

```
# 在每个业务接口中都打印了日志
# 开发者可通过修改config/log4php.xml配置文件来控制日志的输出
<param name="file" value="d:/php_sdk/log/php_sdk.log" />
value: 日志的输出路径, 默认在工作路径 "d:/php_sdk/log/" , 默认日志名称为 "php_sdk.log" 。
```

```
<level value="INFO" />

# level: 日志等级, 默认 "INFO" , 如果value= "OFF" 则会关闭日志。
```

## 鉴权

```
//得到NorthApiClient实例后, 再使用$northApiClient得到鉴权类实例
$authentication = new Authentication($northApiClient);

//调用鉴权类实例authentication提供的业务接口, 如getAuthToken
$authOutDTO = $authentication->getAuthToken();

//从返回的结构体authOutDTO中获取需要的参数, 如accessToken
$accessToken = $authOutDTO->accessToken;
```

## 订阅

```
//得到NorthApiClient实例后, 再使用northApiClient得到订阅类实例
$subscriptionManagement = new SubscriptionManagement($northApiClient);

//先设置好subDeviceData的第一个入参SubDeviceDataInDTO结构体
$sddInDTO = new SubDeviceDataInDTO();
$ddInDTO->notifyType= "deviceDataChanged";
//需要根据实际情况修改回调的ip和端口
$ddInDTO->callbackUrl = "https://XXX.XXX.XXX.XXX:8099/v1.0.0/messageReceiver";
try {
    //调用订阅类实例subscriptionManagement提供的业务接口, 如subDeviceData
    $subDTO = $subscriptionManagement->subDeviceData($sddInDTO, null, $accessToken);
    echo $subDTO;
} catch (NorthApiException $e) {
    echo $e;
}
```

## 注册设备

```
//得到NorthApiClient实例后, 再使用northApiClient得到设备管理类实例
$deviceManagement = new DeviceManagement($northApiClient);

//设置好注册设备接口的第一个入参RegDirectDeviceInDTO2结构体
$rddInDTO = new RegDirectDeviceInDTO2();
$nodeid = "86370303XXXXXX"; //this is a test imei
$verifyCode = $nodeid;
$rddInDTO->nodeId = $nodeid;
$rddInDTO->verifyCode = $verifyCode;
$rddInDTO->timeout = $timeout;

//调用设备管理类实例deviceManagement提供的业务接口, 如regDirectDevice
$rddod = $deviceManagement->regDirectDevice($rddInDTO, null, $accessToken);

//从返回的结构体rddod中获取需要的参数, 如deviceId
$deviceId = $rddod->deviceId;
```

### 说明

关于哪些参数需要设置, 请查看《PHP SDK API参考文档》对于可选参数, 如果业务不需要, 可以不设置或者设置为null。

## 7.2.7 回调接口实现

可以参考Demo中的**PushMessageReceiverTest**类, 需要接收哪一类消息就重写对应的方法, 如:

```
function handleDeviceAdded($body) {  
    echo "deviceAdded ==> " . $body . "\r\n";  
    //TODO deal with deviceAdded notification  
}
```

#### 说明

- 接收到平台推送的消息后，开发者需要根据业务进行处理，但不建议进行复杂计算、I/O操作或者可能长时间等待的动作，可以先写数据库，应用进入相应界面或者刷新界面再从数据库取数据并进行数据处理。
- 回调路径已在SDK中设置好了，所以在订阅时要注意订阅对应的回调地址，具体可参考《PHP SDK API参考文档》文档中消息推送章节的接口。
- 回调的IP地址则是服务器的地址，需要是公网地址。
- Demo工程的回调端口配置在“config\application.ini”中：  
`#specify the port of the web application server.port=8099`

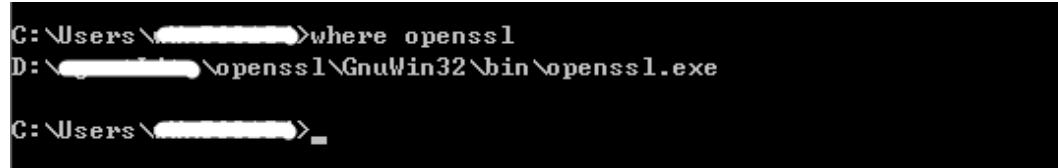
## 7.2.8 回调证书制作、配置及上传

本章节以自签名证书为例。如果是使用商用证书，请直接向CA机构申请。

### 步骤1 首先在windows或者Linux上安装openssl。

```
where openssl
```

图 7-22 Windows 上已经安装 openssl



```
C:\Users\[redacted]>where openssl  
D:\[redacted]\openssl\GnuWin32\bin\openssl.exe  
C:\Users\[redacted]>_
```

### 步骤2 创建测试CA证书。

创建私钥。

```
openssl genrsa -out ca-key.pem 1024
```

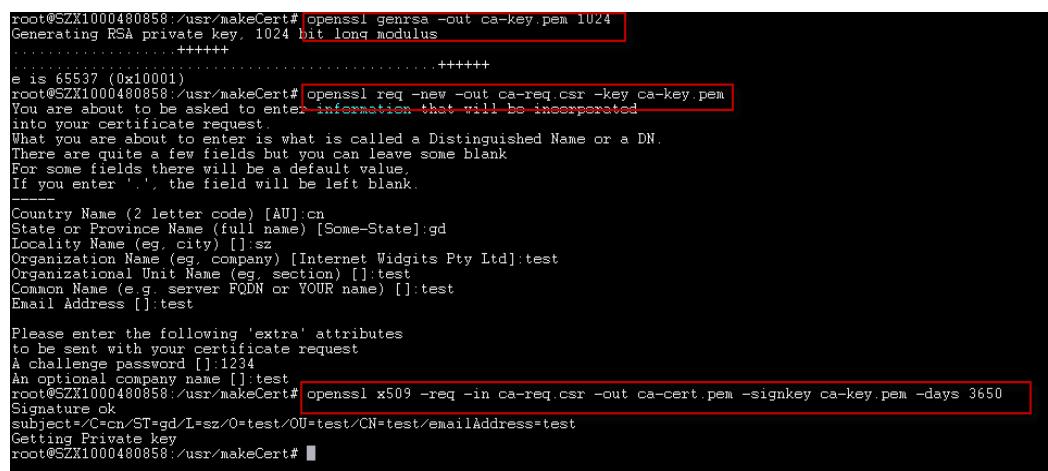
创建证书请求。

```
openssl req -new -out ca-req.csr -key ca-key.pem
```

自签署证书。

```
openssl x509 -req -in ca-req.csr -out ca-cert.pem -signkey ca-key.pem -days 3650
```

图 7-23 创建 CA 证书



```
root@SZX1000480858:/usr/makeCert# openssl genrsa -out ca-key.pem 1024  
Generating RSA private key, 1024 bit long modulus  
+++++  
e is 65537 (0x10001)  
root@SZX1000480858:/usr/makeCert# openssl req -new -out ca-req.csr -key ca-key.pem  
You are about to be asked to enter information that will be incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value.  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [AU]:cn  
State or Province Name (full name) [Some-State]:gd  
Locality Name (eg, city) []:sz  
Organization Name (eg, company) [Internet Widgets Pty Ltd]:test  
Organizational Unit Name (eg, section) []:test  
Common Name (e.g. server FQDN or YOUR name) []:test  
Email Address []:test  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:1234  
An optional company name []:test  
root@SZX1000480858:/usr/makeCert# openssl x509 -req -in ca-req.csr -out ca-cert.pem -signkey ca-key.pem -days 3650  
Signature ok  
subject=/C=cn/ST=gd/L=sz/O=test/OU=test/CN=test/emailAddress=test  
Getting Private key  
root@SZX1000480858:/usr/makeCert# _
```

**步骤3 生成server证书。**

创建私钥。

```
openssl genrsa -out server.key 1024
```

创建证书请求。

```
openssl req -new -out server-req.csr -key server.key
```

自签署证书。

```
openssl x509 -req -in server-req.csr -out server.cert -signkey server.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
```

**图 7-24 生成 server 证书**

```
root@SZX1000480858:/usr/makeCert# openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
root@SZX1000480858:/usr/makeCert# openssl req -new -out server-req.csr -key server.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value
If you enter '.', the field will be left blank.
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:gd
Locality Name (eg. city) []:sz
Organization Name (eg. company) [Internet Widgits Pty Ltd]:test
Organizational Unit Name (eg. section) []:test
Common Name (e.g. server FQDN or YOUR name) []:test
Email Address []:test

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:test
root@SZX1000480858:/usr/makeCert# openssl x509 -req -in server-req.csr -out server.cert -signkey server.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
Signature ok
subject:/O=cn-ST*gd:L=sz/O=test/OU=test/CN=test/emailAddress=test
Getting Private Key
root@SZX1000480858:/usr/makeCert#
```

**步骤4 生成client证书。**

创建私钥

```
openssl genrsa -out client.key 1024
```

创建证书请求

```
openssl req -new -out client-req.csr -key client.key
```

自签署证书

```
openssl x509 -req -in client-req.csr -out client.cert -signkey client.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
```

**图 7-25 生成 client 证书**

```
root@SZX1000480858:/usr/makeCert# openssl genrsa -out client.key 1024
Generating RSA private key, 1024 bit long modulus
.....+++++
e is 65537 (0x10001)
root@SZX1000480858:/usr/makeCert# openssl req -new -out client-req.csr -key client.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value
If you enter '.', the field will be left blank.
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:gd
Locality Name (eg. city) []:sz
Organization Name (eg. company) [Internet Widgits Pty Ltd]:test
Organizational Unit Name (eg. section) []:test
Common Name (e.g. server FQDN or YOUR name) []:test
Email Address []:test

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:test
root@SZX1000480858:/usr/makeCert# openssl x509 -req -in client-req.csr -out client.cert -signkey client.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
Signature ok
subject:/O=cn-ST*gd:L=sz/O=test/OU=test/CN=test/emailAddress=test
Getting Private Key
root@SZX1000480858:/usr/makeCert#
```

**注意**

- 每个步骤的创建证书请求中的配置，请自行设置
- “Common Name (e.g. server FQDN or YOUR name) []” 要输入应用服务器的IP或域名。

**步骤5** 在Demo工程中，已经配置好生成的server证书。

**步骤6** 单击“启动” PushMessageReceiverTest，选择“Run” “PushMessageReceiverTest”，运行Demo运行结果如下：

**说明**

当有数据推送到应用服务器时，就会进入相应的回调函数中。

**图 7-26 PushMessageReceiverTest 类运行结果**

The screenshot shows a terminal window with the following content:

```
149
150     function handleDeviceAdded($body) {
151         echo "deviceAdded ==> " . $body . "\r\n";
152         //TODO deal with deviceAdded notification
153     }
154
155
156     function handleBindDevice($body) {
157         echo "bindDevice ==> " . $body . "\r\n";
158         //TODO deal with BindDevice notification
159     }
160
161
handleDeviceAdded()
```

Run: PushMessageReceiverTest.php

deviceAdded ==> NotifyDeviceAddedDTO [notifyType=deviceAdded, deviceId=c5C913E5-1622-4A51-8441-003CC5E5A7

**步骤7** 使用浏览器打开回调地址“<https://server:8099/v1.0.0/messageReceiver>”，并查看证书。

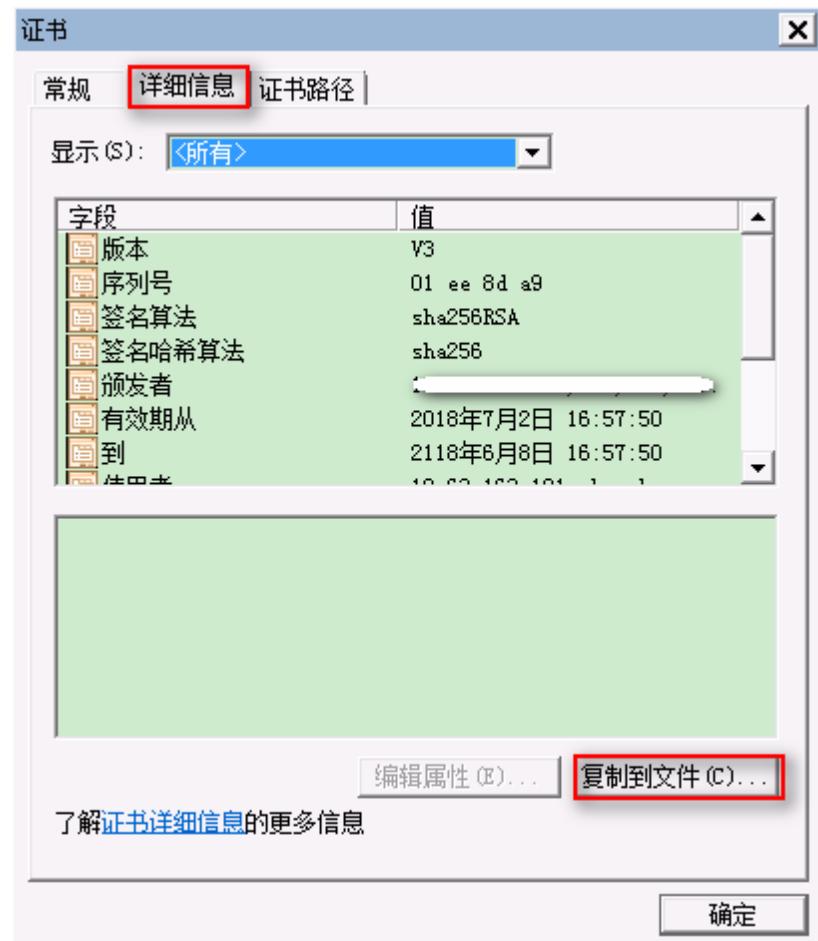
“server”是应用服务器的地址（即本机地址），“8099”是在**application.properties**中配置的端口。

**图 7-27 查看证书**



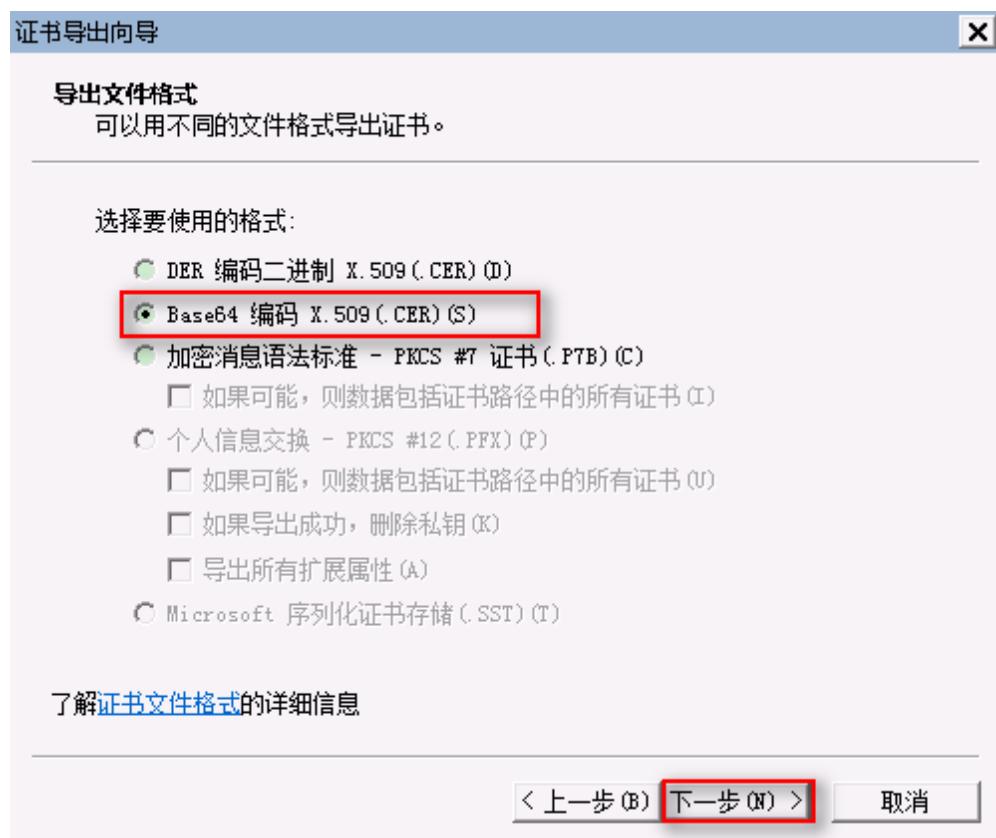
**步骤8** 切换到“详细信息”，点击“复制到文件”。

**图 7-28 详细信息**



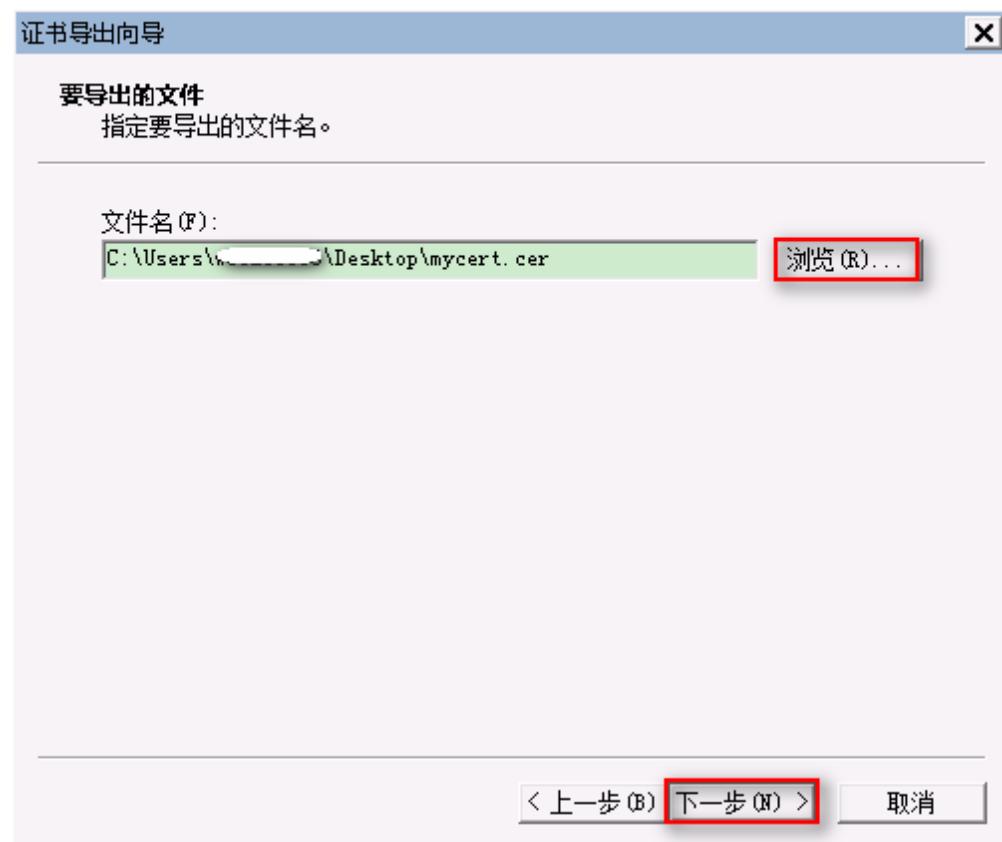
**步骤9** 点击“下一步”，选择Base64编码，再点击“下一步”。

图 7-29 选择编码格式



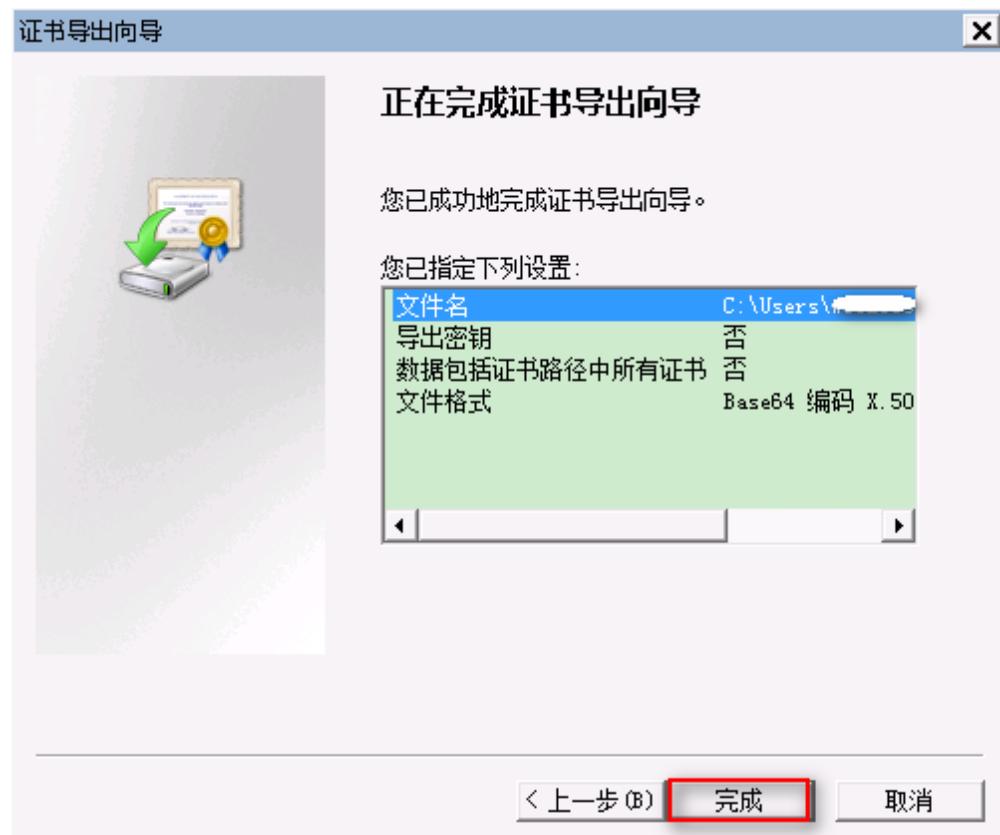
**步骤10** 选择“浏览”，选择一个路径，输入文件名，点击“保存”，回到证书导出向导，点击“下一步”。

图 7-30 指定导入文件名



点击“完成”。

图 7-31 完成证书导出

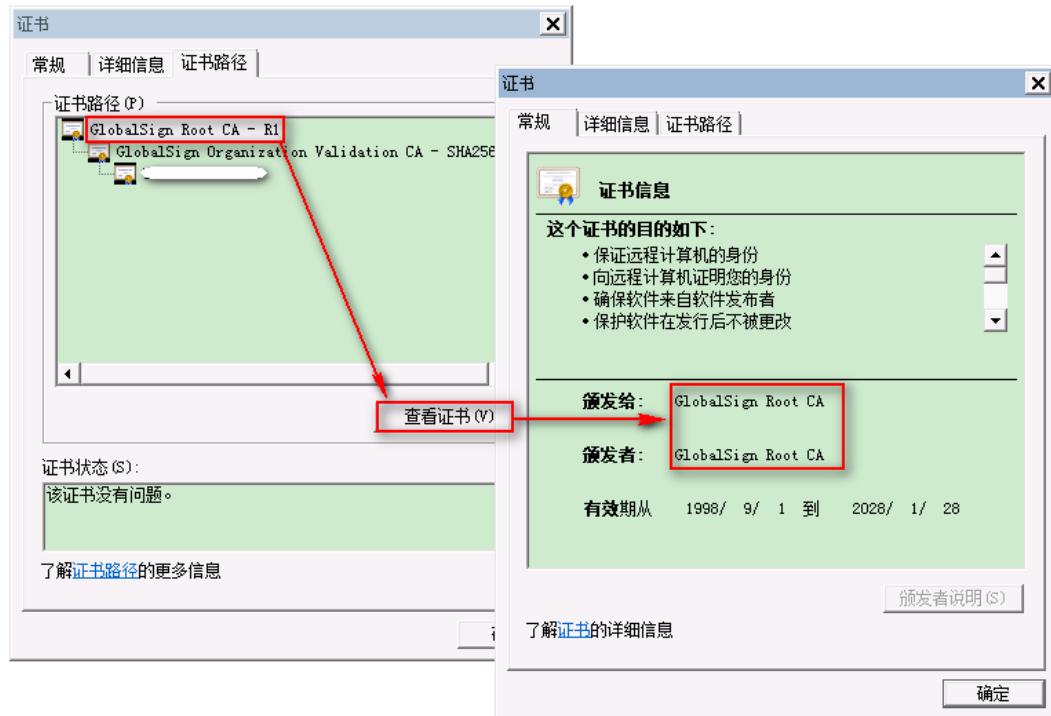


**注意**

如果应用服务器最后部署到云上，可能会有多级证书，建议在部署完成后导出证书。此时需要将证书链上面几级的证书一一导出。

**步骤11** 打开“证书路径”页签，可以看到有多级证书，点击证书路径中的某个证书，点击“查看证书”，然后重复上面的步骤导出此级证书。

图 7-32 逐级导出证书



**步骤12** 将所有证书导出后，使用文本编译器打开证书，将证书内容复制到一个文件中，头尾相连，并保存为.pem格式的文件。该文件需要上传到IoT平台相应的应用下。

图 7-33 合并证书

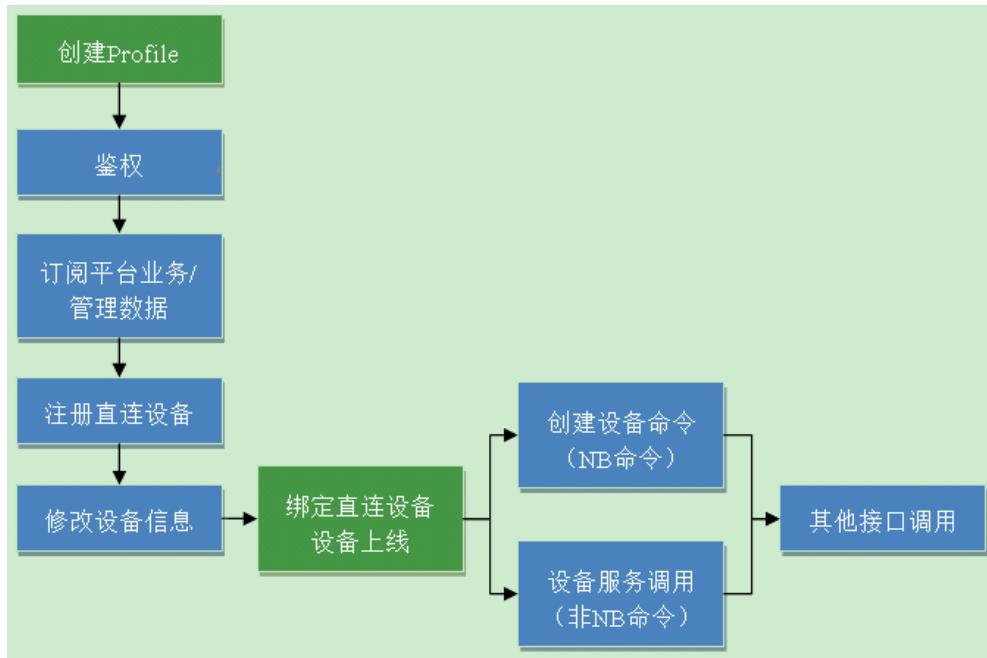
```
24 5gaRQBi5+MHt39tBquCWIMnN2BU4gcmU7qKEKQsTb47bDN01AtukixlE0kF6BWlK
25 WE9gyn6CagsCqiUXObXbf+eEZSqVir2G316BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
26 4fQRbxCl1fznQgUy286dUV4otp6F01vvpx1FQHK0tw5rDgb7MzVIcbidJ4vEZV8N
27 hnacRHr21Vz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
28 -----END CERTIFICATE-----
29 -----BEGIN CERTIFICATE-----
30 MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
31 yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBjbMuMR8wHQYDVQQL
32 ExZWZXJpu2lnbiBUcnVzdCBOZXRB3b3JrMTowOAYDVQQLEzEoYykGMjAwNiBWZXJp
33 U2lnbiwgSW5jLiAtIEZvcibhdXRb3JpemVkiHVzzSEvbmx5MUUwQwYDVQQDEzoW
34 ZXJpu2lnbiBDbGFzcyAzIFB1YmxpYyBQcm1tYXJ5IEhlcnRpZmljYXRpb24gQXVO
-----结束
```

## 7.2.9 业务接口调用流程及注意事项

[业务接口调用方法](#)一节已说明如何调用接口，其他的接口调用方法与之类似，请直接参考Demo代码，不再逐一说明。

- 请按如下流程调用业务接口。

图 7-34 调用业务接口流程



- Demo 中使用的 Profile 如下图所示，只有一个 Brightness 服务， Brightness 服务下有一个 brightness 属性和一个 PUT 命令。在调用创建设备命令或设备服务调用等接口时，如果不是使用以下 Profile 内容，请将相关服务、属性或者命令名称修改为相应的名称。

图 7-35 Profile 文件



- 创建新的Profile方法：

登录开发者 Portal，选择“产品 > 产品开发 > 添加 > 自定义产品 > 自定义产品”，设置“产品名称”、“型号”、“厂商ID”、“所属行业”、“设备类型”、“接入应用层协议类型”等产品信息，点击“确定”。然后点击“新建服务”（根据设备功能添加属性和命令）最后点击“确定”。

#### 说明

建议在定义好Profile后再调用接口注册设备。

- 修改设备信息接口使用到的字段值如“设备类型”、“厂家名”、“厂家ID”、“设备型号”要与Profile定义的保持一致。

- accessToken可以由SDK管理，也可由第三方应用自行管理，具体参考具体可参考《PHP SDK API参考文档》中“应用安全接入 > 定时刷新token”章节的说明。

## 7.3 Python SDK 使用指南

### 7.3.1 开发者必读

- 本文以提供的北向Python SDK Demo为例说明如何使用Python SDK与IoT平台对接。
- Demo以Python工程为例，以invokeapiTest下的每个类都包含了main方法，可单独运行，旨在演示如何调用SDK接口。

### 7.3.2 开发环境要求

开发环境要求

开发平台	开发环境	配套要求	推荐的操作系统
IoT	1. 开发工具： JetBrains PyCharm 2018.1.4 x64 2. Python Project Interpreter: Python 3.7 或Anaconda 3	Python 3.7	Windows7

SDK包为纯Python的egg包，在使用上没有特殊限制，Python在3.7及以上版本即可。

### 7.3.3 下载相关开发资源

下载北向Python SDK Demo及北向Python SDK：

- [北向Python SDK Demo](#)
- [北向Python SDK](#)
- [北向Python SDK API参考](#)

在此Demo中，SDK以及SDK依赖的模块已经放在本地Python 3.7目录下，项目中选择解释器为Python 3.7即可，如果缺少依赖的模块，开发者也可从PyCharm中选择缺少的库自行下载。

下面说明SDK如何加入到Demo中

将打包后的SDK解压至本地Python安装路径“Lib\site-packages”下（此文档的本地路径为“D:\python\Lib\site-packages”）即可在工程中导入。

图 7-36 如何添加 SDK 到 demo 中

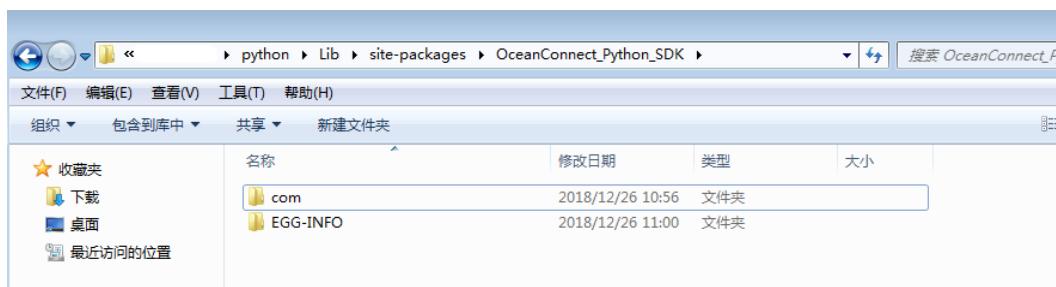
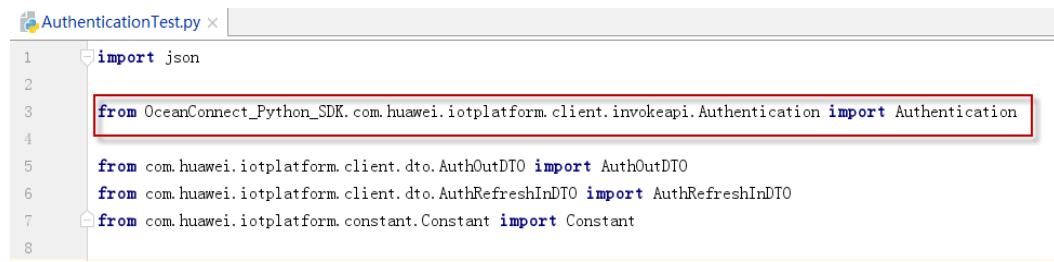


图 7-37 在 Demo 中导入



```
AuthenticationTest.py x
1 import json
2
3 from OceanConnect_Python_SDK.com.huawei.iotplatform.client.invokeapi.Authentication import Authentication
4
5 from com.huawei.iotplatform.client.dto.AuthOutDTO import AuthOutDTO
6 from com.huawei.iotplatform.client.dto.AuthRefreshInDTO import AuthRefreshInDTO
7 from com.huawei.iotplatform.constant.Constant import Constant
8
```

 说明

Demo 工程中已包含 SDK 库。

### 7.3.4 导入 Demo 工程

- 步骤1 将下载的OceanConnect \_ Python \_ SDK \_ Demo.zip解压到本地。
- 步骤2 打开PyCharm，选择“File > Open”，再选择Demo解压后的路径，点击“OK”。
- 步骤3 配置**Project Interpreter**，选择“File > Settings > Project: OceanConnect \_ Python \_ SDK \_ Demo > Project Interpreter”，选在SDK所在的解释器，点击“Apply”，最后点击“OK”即可正确配置工程所需要的SDK。

图 7-38 进入 Settings 界面

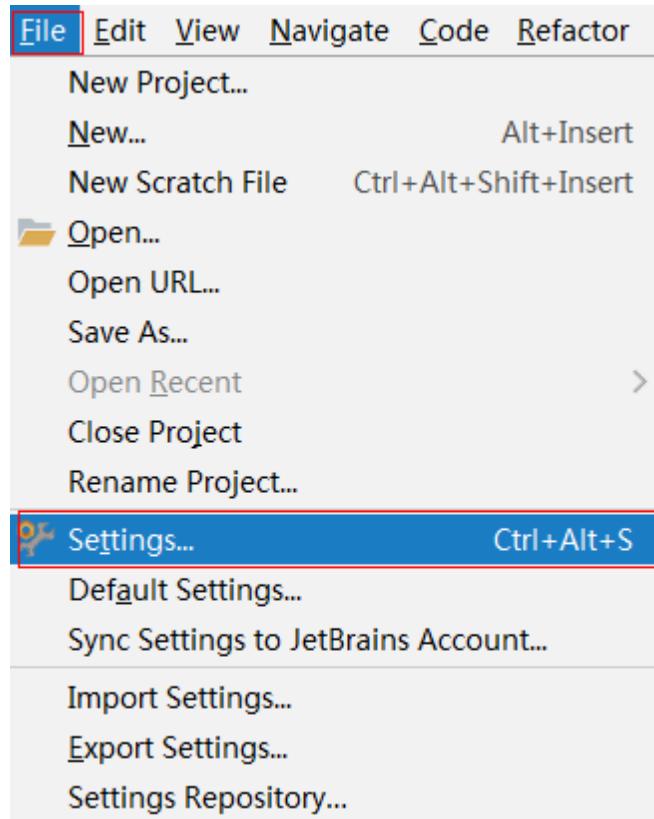
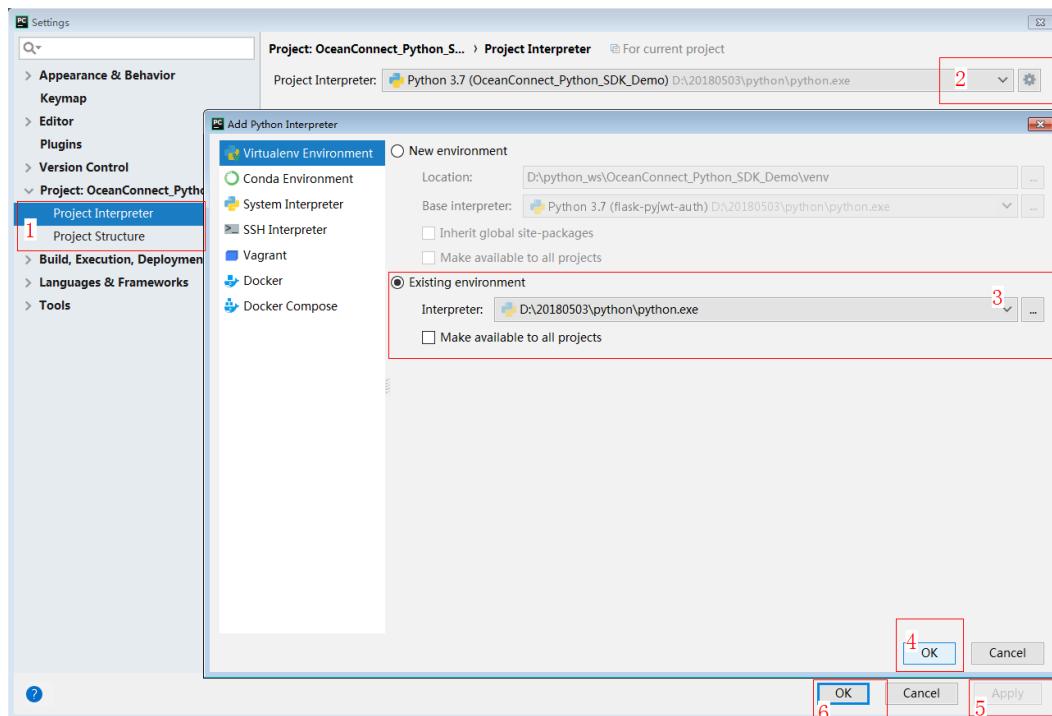


图 7-39 配置 Project Interpreter



----结束

### 7.3.5 初始化及证书配置

在“Constant.py”中，设置好“ClientInfo”（包括平台IP、端口、appId和密码）：

```
class Constant(object):
    # 工作路径
    workPath = os.path.join('d:/python_sdk/')

    # 读取证书
    def readCertificate(self):
        certFilePath = os.path.join(Constant.workPath, 'cert/client.crt')
        certFilePath2 = os.path.join(Constant.workPath, 'cert/client.key')
        cert = (certFilePath, certFilePath2)
    return cert

    # 读取配置文件
    def readConfFile(self):
        configFilePath = os.path.join(Constant.workPath, 'resources/application.ini')
        cf.read(configFilePath)
        platformIp = cf.get("CLIENT_INFO", "platformIp")
        platformPort = cf.get("CLIENT_INFO", "platformPort")
        appId = cf.get("CLIENT_INFO", "appId")
        secret = cf.get("CLIENT_INFO", "secret")
    return platformIp, platformPort, appId, secret

    # 设置北向信息
    def clientInfo(self):
        clientInfo = ClientInfo()
        clientInfo.setPlatformIp((Constant().readConfFile())[0])
        clientInfo.setPlatformPort((Constant().readConfFile())[1])
        clientInfo.setAppId((Constant().readConfFile())[2])
        clientInfo.setSecret((Constant().readConfFile())[3])
        clientInfo = DictUtil.dto2dict(clientInfo)
    return clientInfo
```

**注意**

- 配置文件（**application.ini**）以及证书都分别放置在工作路径“d:/python\_sdk/”下面的**cert**和**resources**文件夹中，用户也可自行设置工作路径。
- 平台IP、端口、appId和密码都是从配置文件中读取的，因此，当这些信息发生变化时，只要修改配置文件，不用修改应用服务器的代码。

## 7.3.6 业务接口调用方法

在[5 初始化及证书配置](#)中设置好**Constant.py**后才能调用其他业务接口。所有业务接口的测试都在“`invokeapiTest.py`”中。

以下几个接口为例说明如何调用业务接口：

### 配置日志

```
# 在每个业务接口中都配置了日志，具体日志实现可参考LogUtil.py:  
# 开发者可通过修改入参来控制日志的输出，以下三个参数都可自行设置：  
# logPath: 日志的输出路径，默认在工作路径d:/python_sdk/log/  
# level: 日志等级，默认DEBUG（最低等级，即大于等于该等级的日志级别都会输出），如果level=0没有日志输出  
# logFilename: 日志名称  
  
def setLogConfig(self,  
    logPath=s.path.join(Constant.workPath, 'log/'),  
    level=logging.DEBUG,  
    logFilename="python_sdk.log"):
```

### 鉴权

```
# 在AuthenticationTest.py中对鉴权和刷新token接口进行测试：  
if __name__ == "__main__":  
    from com.huawei.iotplatform.utils.LogUtil import Log  
  
    # 实例化Authentication()  
    authentication = Authentication()  
  
    # 调用鉴权类实例authentication提供的业务接口，如getAuthToken  
    ag = authentication.getAuthToken(Constant().clientInfo())  
    print("===== get access token =====")  
    print("result:", ag + "\n")  
  
    # 从返回的ag中获取需要的参数，如accessToken  
    authOutDTO = AuthOutDTO()  
    authOutDTO.setAccessToken(json.loads(ag)[`accessToken`])  
    print("token", authOutDTO.getAccessToken())
```

### 订阅

```
# 在SubscriptionManagementTest.py中对订阅平台业务数据接口进行测试：  
class SubscriptionManagementTest(object):  
    def subDeviceBusinessData(self):  
        sdbdInDTO = SubDeviceBusinessDataInDTO()  
        sdbdInDTO.notifyType = "bindDevice"  
        sdbdInDTO.callbackUrl = "https://XXX.XXX.XXX.XXX:443/callbackurltest"  
        return sdbdInDTO  
  
    if __name__ == "__main__":  
        from com.huawei.iotplatform.utils.LogUtil import Log  
  
        # 实例化  
        smTest = SubscriptionManagementTest()  
        subscriptionManagement = SubscriptionManagement()
```

```
# get accessToken at first
result = Authentication().getAuthToken(Constant().clientInfo())
authOutDTO = AuthOutDTO()
authOutDTO.setAccessToken(json.loads(result)[`accessToken`])
accessToken = authOutDTO.getAccessToken()

# 调用订阅类实例subscriptionManagement提供的业务接口, 如subDeviceData
ss = subscriptionManagement.subDeviceBusinessData(smTest.subDeviceBusinessData(), accessToken)
print("===== subscribe to device business data notification =====")
print("result:", ss + "\n")
```

## 注册设备

```
# 在DeviceManagementTest.py中对注册设备接口进行测试:
class DeviceManagementTest(object):
    def regDirectDeviceInfo(self):
        rddInDto = RegDirectDeviceInDTO()
        rddInDto.nodeId = "AAA" + str(random.randint(0, 9999))
        return rddInDto

    if __name__ == "__main__":
        from com.huawei.iotplatform.utils.LogUtil import Log

        # 实例化
        dmTest = DeviceManagementTest()
        deviceManagement = DeviceManagement()

        # get accessToken at first
        result = Authentication().getAuthToken(Constant().clientInfo())
        authOutDTO = AuthOutDTO()
        authOutDTO.setAccessToken(json.loads(result)[`accessToken`])
        accessToken = authOutDTO.getAccessToken()

        # 调用设备管理类实例deviceManagement提供的业务接口, 如regDirectDevice
        dr = deviceManagement.regDirectDevice(dmTest.regDirectDeviceInfo(), None, accessToken)
        print("===== register a new device =====")
        print("result:", dr + "\n")

    # 从返回的dr中获取需要的参数, 如deviceId
    rddod = RegDirectDeviceOutDTO()
    rddod.setDeviceId(json.loads(dr)[`deviceId`])
    deviceId = rddod.getDeviceId()
    print("deviceId==", deviceId + "\n")
```

### 说明

关于哪些参数需要设置, 请查看《[Python SDK API参考文档](#)》对于可选参数, 如果业务不需要, 可以不设置。

## 7.3.7 回调接口实现

回调接口的启动在“PushMessageReceiverTest.py”中, 配置请看以下说明。

```
if __name__ == '__main__':
    # 回调地址和端口
    callbackUrl = "XXX.XXX.XXX.XXX"
    port = 8099

    # 这里使用了pyOpenSSL自带证书, 注意ssl_context的值必须是adhoc
    app.run(host=callbackUrl, port=port, ssl_context='adhoc')

    # 使用自己配置的证书, ssl_context的值配置如下: 此处放置的是第8章节步骤3生成的server证书
    app.run(host=callbackUrl, port=port, ssl_context='d:/python_sdk/cert/server.crt', 'd:/python_sdk/cert/server.key'))
```

具体业务实现在“PushMessageReceiver.py”中的“PushMessageReceiver”类中, 可以参考Demo中的“PushMessageReceiver”类, 需要接收哪一类消息就改写对应的方法, 如:

```
class PushMessageReceiver(object):
    def handleDeviceAdded(self):
        print("deviceAdded ==> ", request.json)
#TODO deal with deviceAdded notification
    pass
```



## 说明

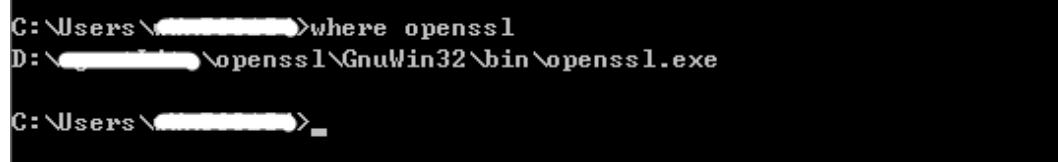
- 回调的IP地址则是服务器的地址，需要是公网地址，端口自行配置。
- 回调证书请参考[回调证书制作、配置及上传](#)。
- 接收到平台推送的消息后，开发者需要根据业务进行处理，但不建议进行复杂计算、I/O操作或者可能长时间等待的动作，可以先写数据库，应用进入相应界面或者刷新界面再从数据库取数据并进行数据处理。
- 回调路径已在SDK中设置好了，所以在订阅时要注意订阅对应的回调地址，具体可参考《Python SDK API参考》文档中“消息推送”章节的接口。

### 7.3.8 回调证书制作、配置及上传

**步骤1** 首先在windows或者Linux上安装openssl。

```
where openssl
```

图 7-40 Windows 上已经安装 openssl



```
C:\Users\[redacted]>where openssl
D:\[redacted]\openssl\GnuWin32\bin\openssl.exe

C:\Users\[redacted]>_
```

**步骤2** 创建测试CA证书。

创建私钥

```
openssl genrsa -out ca-key.pem 1024
```

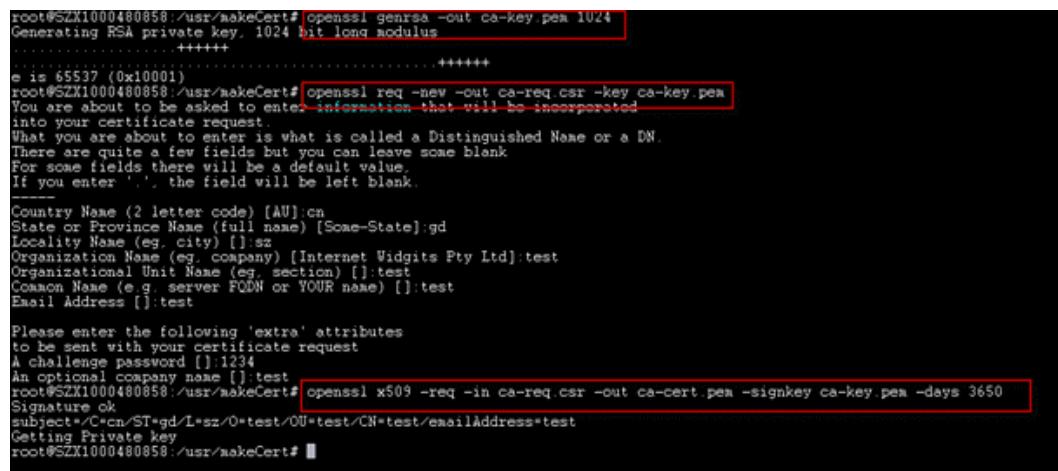
创建证书请求

```
openssl req -new -out ca-req.csr -key ca-key.pem
```

自签署证书

```
openssl x509 -req -in ca-req.csr -out ca-cert.pem -signkey ca-key.pem -days 3650
```

图 7-41 创建 CA 证书



```
root@SZX1000480858:/usr/makeCert# openssl genrsa -out ca-key.pem 1024
Generating RSA private key, 1024 bit long modulus
+++++
e is 65537 (0x10001)
root@SZX1000480858:/usr/makeCert# openssl req -new -out ca-req.csr -key ca-key.pem
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value.
If you enter '.', the field will be left blank.
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:gd
Locality Name (eg. city) []:sz
Organization Name (eg. company) [Internet Widgets Pty Ltd]:test
Organizational Unit Name (eg. section) []:test
Common Name (e.g. server FQDN or YOUR name) []:test
Email Address []:test

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []:test
root@SZX1000480858:/usr/makeCert# openssl x509 -req -in ca-req.csr -out ca-cert.pem -signkey ca-key.pem -days 3650
Signature ok
subject=/C=cn/ST=gd/L=sz/O=test/OU=test/CN=test/emailAddress=test
Getting Private key
root@SZX1000480858:/usr/makeCert# _
```

**步骤3** 生成server证书。

创建私钥

```
openssl genrsa -out server.key 1024
```

创建证书请求

```
openssl req -new -out server-req.csr -key server.key
```

自签署证书

```
openssl x509 -req -in server-req.csr -out server.cert -signkey server.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
```

**图 7-42 生成 server 证书**

```
root@GZK10004#0059:/usr/makeCert# openssl genrsa -out server.key 1024
Generating RSA private key, 1024 bit long modulus
...+++++
e is 65537 (0x10001)
root@GZK10004#0059:/usr/makeCert# openssl req -new -out server-req.csr -key server.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value
If you enter '' the field will be left blank.
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:gd
Locality Name (eg. city) []:sz
Organization Name (eg. company) [Internet Widgits Pty Ltd]:test
Organizational Unit Name (eg. section) []:test
Common Name (e.g. server FQDN or YOUR name) []:test
Email Address []:test

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []: test
root@GZK10004#0059:/usr/makeCert# openssl x509 -req -in server-req.csr -out server.cert -signkey server.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
./makecert ok
subject:/C=cn/ST=gd/L=sz/O=test/OU=test/CN=test/emailAddress=test
Getting Private Key
Getting Private Key
root@GZK10004#0059:/usr/makeCert#
```

**步骤4** 生成client证书。

创建私钥

```
openssl genrsa -out client.key 1024
```

创建证书请求

```
openssl req -new -out client-req.csr -key client.key
```

自签署证书

```
openssl x509 -req -in client-req.csr -out client.cert -signkey client.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
```

**图 7-43 生成 client 证书**

```
root@GZK10004#0059:/usr/makeCert# openssl genrsa -out client.key 1024
Generating RSA private key, 1024 bit long modulus
...+++++
e is 65537 (0x10001)
root@GZK10004#0059:/usr/makeCert# openssl req -new -out client-req.csr -key client.key
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN
There are quite a few fields but you can leave some blank
For some fields there will be a default value
If you enter '' the field will be left blank.
Country Name (2 letter code) [AU]:cn
State or Province Name (full name) [Some-State]:gd
Locality Name (eg. city) []:sz
Organization Name (eg. company) [Internet Widgits Pty Ltd]:test
Organizational Unit Name (eg. section) []:test
Common Name (e.g. server FQDN or YOUR name) []:test
Email Address []:test

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:1234
An optional company name []: test
root@GZK10004#0059:/usr/makeCert# openssl x509 -req -in client-req.csr -out client.cert -signkey client.key -CA ca-cert.pem -CAkey ca-key.pem -CAcreateserial -days 3650
./makecert ok
subject:/C=cn/ST=gd/L=sz/O=test/OU=test/CN=test/emailAddress=test
Getting Private Key
Getting Private Key
root@GZK10004#0059:/usr/makeCert#
```

**注意**

- 每个步骤的创建证书请求中的配置，请自行设置。
- “Common Name (e.g. server FQDN or YOUR name) []” 要输入应用服务器的IP或域名。

**步骤5** 在Demo工程中，已经配置好生成的server证书。

**步骤6** 单击“启动” **PushMessageReceiverTest**，选择“Run” “PushMessageReceiverTest”，运行Demo运行结果如下：

**说明**

当有数据推送到应用服务器时，就会进入相应的回调函数中。

**图 7-44 PushMessageReceiverTest 类运行结果**



```
class PushMessageReceiver(object):  
  
    def handleDeviceAdded(self):  
        print("deviceAdded ==> ", request.json)  
        pass  
  
    def handleBindDevice(self):  
        print("bindDevice ==> ", request.json)  
        pass
```



**步骤7** 使用浏览器打开回调地址“<https://server:8099/v1.0.0/messageReceiver>”，并查看证书。

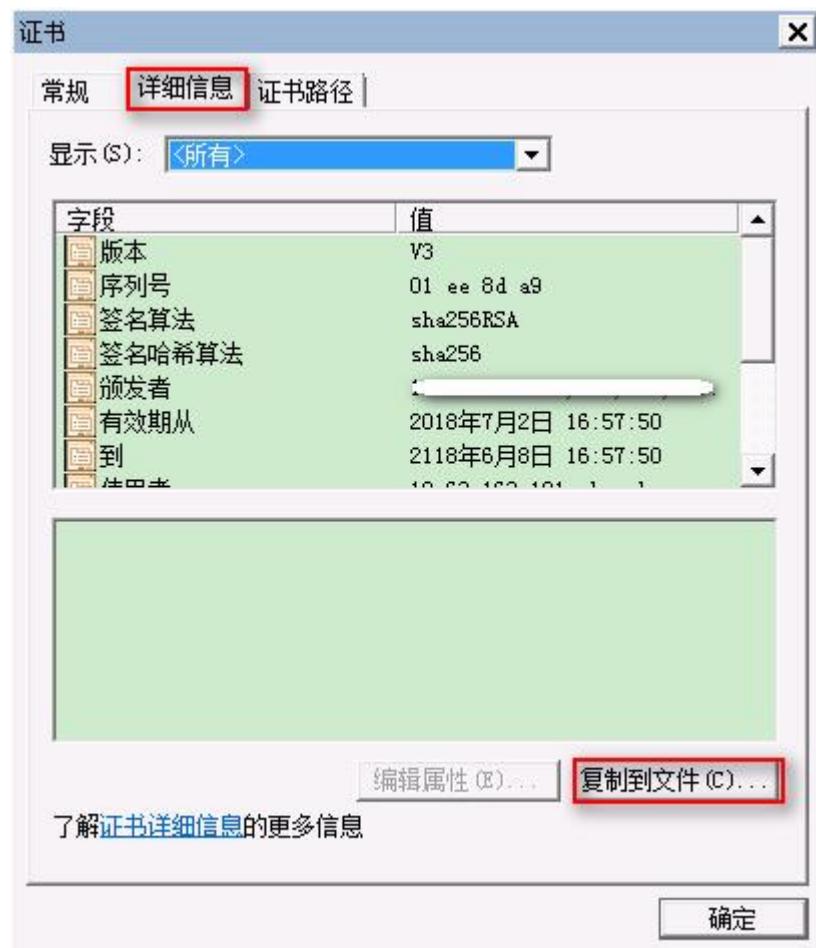
“server”是应用服务器的地址（即本机地址），“8099”是在**PushMessageReceiverTest**中配置的端口。

图 7-45 查看证书



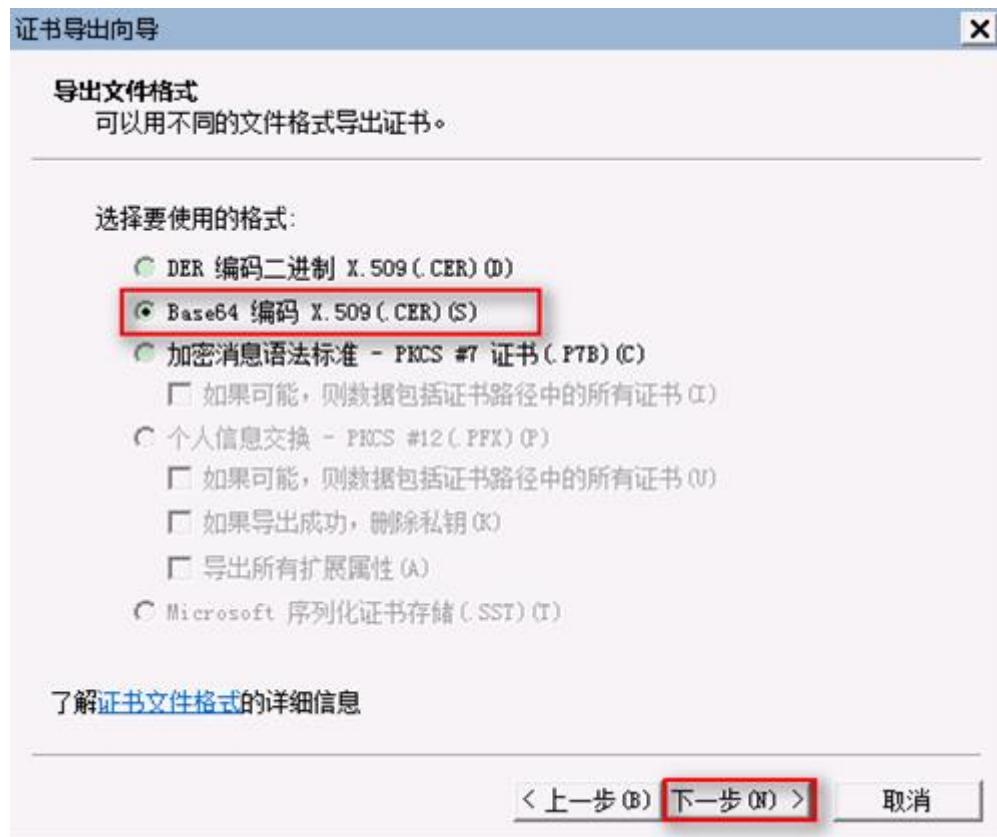
步骤8 切换到“详细信息”，点击“复制到文件”。

图 7-46 详细信息



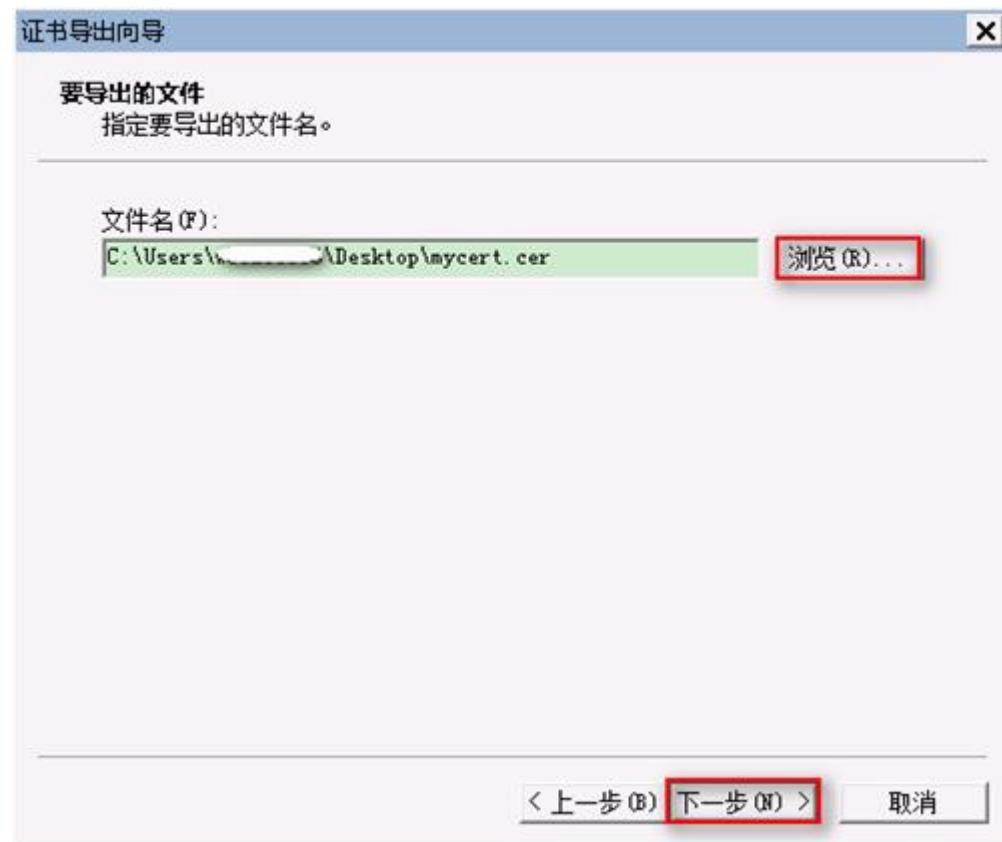
步骤9 点击“下一步”，选择Base64编码，再点击“下一步”。

图 7-47 选择编码格式



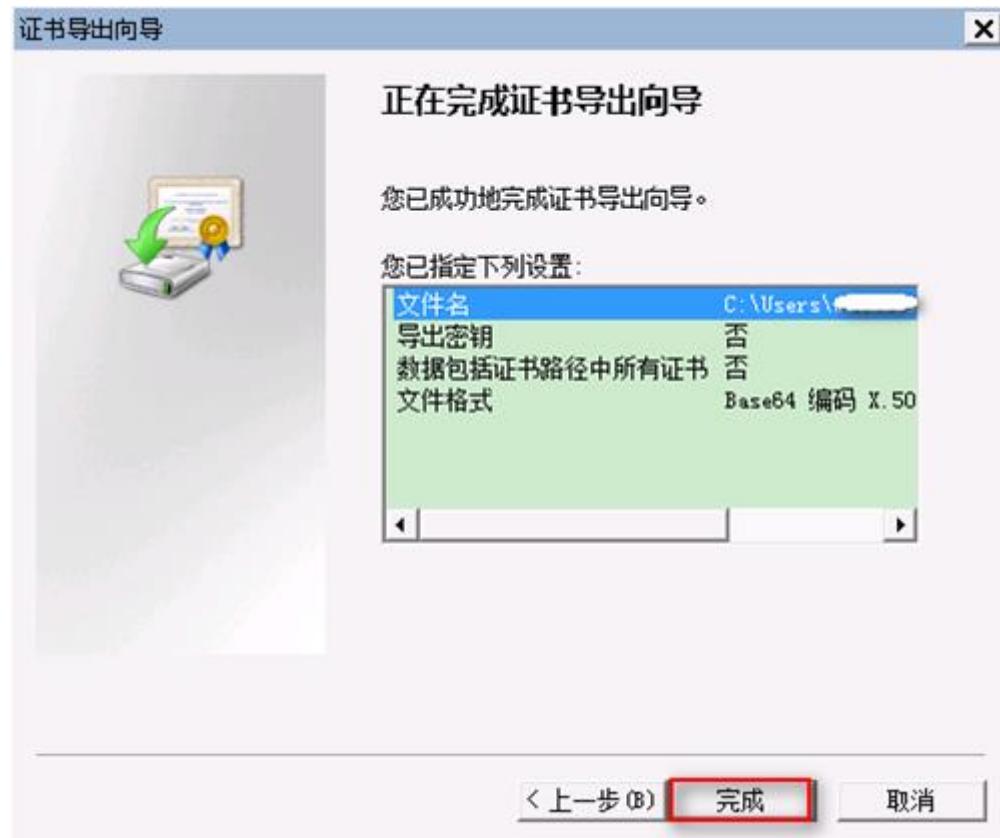
**步骤10** 选择“浏览”，选择一个路径，输入文件名，点击“保存”，回到证书导出向导，点击“下一步”。

图 7-48 指定导入文件名



点击“完成”。

图 7-49 完成证书导出



**注意**

如果应用服务器最后部署到云上，可能会有多级证书，建议在部署完成后导出证书。此时需要将证书链上面几级的证书一一导出。

**步骤11** 打开“证书路径”页签，可以看到有多级证书，点击证书路径中的某个证书，点击“查看证书”，然后重复上面的步骤导出此级证书。

图 7-50 逐级导出证书



**步骤12** 将所有证书导出后，使用文本编译器打开证书，将证书内容复制到一个文件中，头尾相连，并保存为.pem格式的文件。该文件需要上传到IoT平台相应的应用下。

图 7-51 合并证书

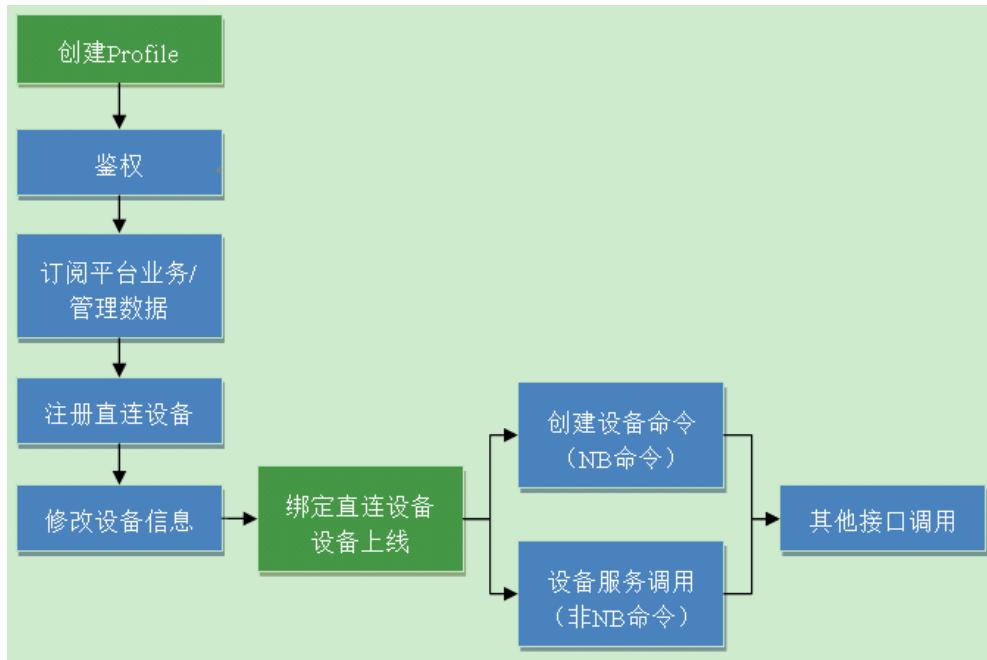
```
24 5gaRQBi5+Mht39tBquCWIMnN2BU4gcmU7qKEKQsTb47bDN01AtukixlE0kF6BWlK
25 WE9gyn6CagsCqiUXObXbf+eEZSqVir2G316BFoMtEMze/aiCKm0oHw0LxOXnGiYZ
26 4fQRbxCl1fznQgUy286dUV4otp6F01vvpx1FQHKOtW5rDgb7MzVIcbidJ4vE2V8N
27 hnacRHr21Vz2XTIIM6RUthg/aFzyQkqFOFSDX9HoLPKsEdao7WNq
28 -----END CERTIFICATE-----
29 -----BEGIN CERTIFICATE-----
30 MIIFODCCBCCgAwIBAgIQUT+5dDhwtzRAQY0wkwaZ/zANBgkqhkiG9w0BAQsFADCB
31 yjELMAkGA1UEBhMCVVMxFzAVBgNVBAoTDlZlcmlTaWduLCBjbMuMR8wHQYDVQQL
32 ExZWZXJpU2lnbiBUcnVzdCBOZXR3b3JrMTowOAYDVQQLEzEoYykgMjAwNiBWZXJp
33 U2lnbiwgSW5jLiAtIEZvcibhdXRob3JpemVkIHZvZSBvbmx5MUUwQwYDVQQDEzxW
34 ZXJpU2lnbiBDbGFzcyAzIFB1YmxpYyBQcmIyZJ5IEEN1cnRpZmljYXRpb24gQXV0
-----结束
```

### 7.3.9 业务接口调用流程及注意事项

初始化及证书配置一节已说明如何调用接口，其他的接口调用方法与之类似，请直接参考Demo代码，不再逐一说明。

- 请按如下流程调用业务接口。

图 7-52 调用业务接口流程



- Demo 中使用的 Profile 如下图所示，只有一个 Brightness 服务， Brightness 服务下有一个 brightness 属性和一个 PUT 命令。在调用创建设备命令或设备服务调用等接口时，如果不是使用以下 Profile 内容，请将相关服务、属性或者命令名称修改为相应的名称。

图 7-53 Profile 文件



- 创建新的Profile方法：

登录开发者 Portal，选择“产品 > 产品开发 > 添加 > 自定义产品 > 自定义产品”，设置“产品名称”、“型号”、“厂商ID”、“所属行业”、“设备类型”、“接入应用层协议类型”等产品信息，点击“确定”。然后点击“新建服务”（根据设备功能添加属性和命令）最后点击“确定”。

#### 说明

建议在定义好Profile后再调用接口注册设备。

- 修改设备信息接口使用到的字段值如“设备类型”、“厂家名”、“厂家ID”、“设备型号”要与Profile定义的保持一致。

- accessToken可以由SDK管理，也可由第三方应用自行管理，具体参考具体可参考《[Python SDK API参考文档](#)》中“应用安全接入 > 定时刷新token”章节的说明